R documentation

of ‘man/autoDetrender.Rd’ etc.

February 16, 2017

`autoDetrender` *Find out the change point in official Yield and calls untrendIt to verify it’s effects*

Usage

```
autoDetrender()
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function ()
{
    tableXregression <- merge(yieldPrev$flatYield, yieldPrev$relatedModel,
        by = "YEAR")
    coluClean <- function(cola) {
        if (min(tableXregression[, cola]) == max(tableXregression[,
            cola])) {
            return(cola)
        }
    }
    dirtyCol <- lapply(X = seq(1, length(names(tableXregression))),
        FUN = coluClean)
    if (!is.null(unlist(dirtyCol))) {
        tableXregression <- tableXregression[, -unlist(dirtyCol)]
    }
    if (pettitt.test(tableXregression$OFFICIAL_YIELD)$p.value <=
        0.5 & pettitt.test(tableXregression$OFFICIAL_YIELD)$estimate >
        1) {
      untrendingIt(tableXregression$YEAR[pettitt.test(tableXregression$OFFICIAL_YIELD)$estimate],
            tableXregression)
    }
    while (yieldPrev$refeedAutoTrend < yieldPrev$currentYear) {
        try(untrendingIt(yieldPrev$refeedAutoTrend, tableXregression))
    }
  }
```

`autoProposal` *try to automatically detrend the official data, if success it proposes to adopt the changes*

Usage

```
autoProposal()
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function ()
{
    if (yieldPrev$flattyn == "y") {
        suppressMessages(suppressWarnings(autoDetrender()))
        dev.new()
        attach(yieldPrev)
        flatPlot <- ggplot(yieldPrev$flatYield) + geom_point(color = "black",
            aes(x = YEAR, y = OFFICIAL_YIELD, group = 1)) + geom_line(color = "black",
            aes(x = YEAR, y = OFFICIAL_YIELD, group = 1)) + geom_smooth(method = "lm",
            color = "brown", se = FALSE, aes(x = YEAR, y = OFFICIAL_YIELD,
                group = 1)) + geom_smooth(method = "loess", color = "red",
            se = FALSE, aes(x = YEAR, y = OFFICIAL_YIELD, group = 1)) +
            geom_smooth(method = "lm", formula = y ~ splines::bs(x,
                3), color = "orange", se = FALSE, aes(x = YEAR,
                y = OFFICIAL_YIELD, group = 1))
        flatPlot <- flatPlot + geom_rect(data = yieldPrev$breakPoint,
            aes(xmin = begin, xmax = finish, ymin = -Inf, ymax = +Inf),
            fill = "yellow", alpha = 0.3)
        plot(flatPlot)
        detach(yieldPrev)
       cat("This is the result of an automated de trending procedure. \n Do you want to use it? \n y : save it a
        appAuto <- scan(, what = "text", nmax = 1)
        while (appAuto != "y" & appAuto != "n") {
            cat("answer y or n \n ")
            appAuto <- scan(, what = "text", nmax = 1)
        }
        if (appAuto == "n") {
            rm(list = c("breakPoint", "flatYield", "due2trend",
                "friendShip", "flattyn", "safeTrend", "yieldTrend",
                "flatOff", "tableXregression", "model_formula",
                "CVmsRes", "expYield", "omniYield", "modelLM",
                "PCmodel"), envir = yieldPrev)
            yieldPrev$flatYield <- yieldPrev$actualYield
        }
        dev.off()
    }
  }
```

`batchFore` *Batch alias of virgilio()*

Description

this calls all the other required function for the regression to be done

Usage

```
batchFore(impreadSet, modKind)
```

Arguments

`impreadSet`	list of arguments for importYield
`modKind`	Need to be specified to automatically select the standard or enhanced model kind in modSel

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (impreadSet, modKind)
{
}
```

`breakTrends` *select where to break the trends and plot usefull graphs*

Usage

```
breakTrends()
```

`checkTrends` *Plot usefull graph to check the existance of trends*

Description

It plots and asks for removing trend or no. In virgilio there is no other way to stop removing trends than answer something ese than "y"

Usage

```
checkTrends()
```

Note

If it seems almost useless, note the role it assumes in the default (virgilio()'s one) path of work.

`configure` *import all the data needed*

Usage

```
configure()
```

Arguments

`depth`

Details

In normal mode, you ha only to follow the screen instruction. Here are supported two database structures.

Wide table split in two files: an official one and a simulated one.

Long table where all the data are stored in one single file.

More combination of Long-Wide and single-two(-more?) may get supported in future releases.

Please note that some column name are required (and case sensitive): YEAR,DECADE,NUTS_CODE.

OFFICIAL_YIELD can be coerced to OFFICIAL_YIELD from an amount of writings (cases, underscore alternatives).

It supports only the .csv input format, please note that it get rid of several kind of csv-like standards. Anyway it will fail if , (comma) is used as decimal separator; I strongly suggest to adopt the standard . (dot) .

`cutTrend` *Actually do the cutting of trends, not simply setting it*

Description

Two input parameters: It receives the edges to cut at: "begin" and "end".

The resulting trend are saved for their contribution to each year yield. If a trend ends before the time series data, it's results are preserved (and not increased) in the following years.

Usage

```
#called by sewTrends()
cutTrend(inizio, fine)
```

Arguments

`inizio`	is the lower year (included) of the trended lapse
`fine`	is the upper year (included) of the trended lapse

Note

The lower year included in the trend comes out as it get in. Considering that the trand is, at that time, experiencing the year of its start no changes actually occur.

Mathematically, the trend is accounted to be linear, and so described by the following:

$yield_n~unTrended= yield_n~Offical - (year_n - year_startTrend) * Trend_coefficient $

So that is clear that for n = startTrend (aka "inizio") it is that the trend and the unTrended versions of yield are the same.

`foreYield-package` *forecasting yield result*

Description

Using past yield and model predictors, forecast the yield resulting for the current (the last) year (simulated year). Nothing new but the interface "virgilio()" providing easy use without any (almost) knowledge of R

Details

The DESCRIPTION file: This package was not yet installed at build time.

Index: This package was not yet installed at build time.

Author(s)

Fosco Vesely

Maintainer: Fosco Vesely <fosco@tana.it>

References

~~ Literature or other references for background information ~~

See Also

~~ Optional links to other man pages, e.g. ~~ ~~ <pkg> ~~

Examples

```
~~ simple examples of the most important functions ~~
```

`importYield` *import csv, based on list argument*

Description

In order to perform batch processing, configure() is reshaped in this. it allows do.call(importYield, args), which is faster than interactive mode.

Usage

```
importYield(offiData, simuData, allData, crp, cnt, dec)
```

Arguments

`offiData`	Official yield data csv
`simuData`	Regressor coming from the simulation. csv file position
`allData`	In case you have a long table containing all you data
`crp`	crop filter (optional)
`cnt`	country filter (optional)
`dec`	decade filter (optional, if any the last one is assumed)

Details

main CSV standards ad customs are handled so that this is supposed to be free from read.csv related error. (such the hope, at least). see also impreadSet

Value

This function returns actualYield and relatedModel as dataframe stored in yieldPrev environment

`loadYieldSession` *simply restore a past session (saved)*

Usage

```
loadYieldSession()
```

`modSel` *Interactive choice of the model you like more*

Description

Under the bonnet it does some things more. It proceed with the Cross validation and the prediction for the target year.

Usage

```
modSel()
```

`responseYield` *Some adjustment and mainly prints the results*

Description

At this stage, the work is done and the results are printed. You can read information about the prediction, the Cross Validation results (plotted in detail). At this stage a PCA regression is processed (by itself) and it's results are printed.

So that you have for comparison:prediction,CV,PCR,TimeSeriesForecast.

The results are based on the data as the came out of the un-trending process so that have to be added for the trend effects. This is done (explicitly) for the predicted one only. All the others have to summed to the current trend by hand (if you prefer them to your own result after the comparison)

Usage

```
responseYield()
```

`saveYieldSession` *Simply saves yieldPrev environment for future use*

Description

The environment is saved with a name containing date, country code and crop code.

Usage

```
saveYieldSession()
```

`sewTrends` *Tuning systemic trends*

Description

If yield experience a trend the models don't grasp you have to remove it to get the right lm. But some part of it may, anyway get explained by the models... Here is the sewing it provides graphs to understand if at least part of the trend you find with checkTrends() is external to simulated parameters. Prompts interactively for the choice of the best parameters describing the trend in the time lapse you selected (from inizio to fine). It also plot a graph with the parameters and yields "normalized" so that matching the trends is easier. Normalization happens around the mean each parameter given in percentage.

Usage

```
#breakTrends() call this
sewTrends(inizio, fine)
```

Arguments

`inizio`	is the lower year (included) of the trended lapse
`fine`	is the upper year (included) of the trended lapse

Details

checkTrends() call include both "inizio" and "fine",

Examples

```
#Designed to be called by breakTrends() (which itself is called by checkTrends())
```

`untrendingIt` *automatically do what supposed to interactively done in checkTrends() breakTrends() sewTrend. then calls cutTrend()*

Usage

```
untrendingIt(inizio, tableXregression)
```

Arguments

`inizio`	first year in which the trend appears (automatically found by pettit.test in autoDetrender)
`tableXregression`	All the data merged from actualYield and related model

Details

it is designed tto be called from autoDetrender

Value

stores yieldPrev$safeTrend and calls cutTrend()

`valiTrend` *Some statistical checks on the printed results*

Description

It performs two validation. That the data provided for the regression are free from noticeable trends (which is very likely to be). Mainly it checks that the trend marked doesn't cause the data to have a worse fitting (MSE) than without any treatment. If it were such, then it notify this and prompts for a reset of Trends as marked previously.

Usage

```
valiTrend()
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function ()
{
    train <- subset(yieldPrev$due2trend, yieldPrev$due2trend$trended ==
        0)
    train <- merge(train, yieldPrev$actualYield, by = "YEAR")
    train$trended <- NULL
    train <- merge(train, yieldPrev$relatedModel, by = "YEAR")
    noTrendMod <- lm(as.formula(yieldPrev$model_formula), data = train)
    test <- subset(yieldPrev$due2trend, yieldPrev$due2trend$trended !=
        0)
    test <- merge(test, yieldPrev$relatedModel, by = "YEAR")
    test$trended <- NULL
    solution <- subset(yieldPrev$due2trend, yieldPrev$due2trend$trended !=
        0)
    solution <- merge(solution, yieldPrev$flatYield, by = "YEAR")
    solution$trended <- NULL
    woTrend <- predict(noTrendMod, newdata = test, se.fit = TRUE,
        type = "response", level = 0.95, interval = "prediction")
    predCfg <- merge(solution, yieldPrev$omniYield, by = "YEAR")
    predCfg$clean <- woTrend$fit[, 1]
    DnoTREND <- predCfg$YIELD - predCfg$pred
    DwTREND <- predCfg$OFFICIAL_YIELD - predCfg$clean
    sigNO <- sqrt(mean((DnoTREND - mean(DnoTREND))^2))
    sigW <- sqrt(mean((DwTREND - mean(DwTREND))^2))
    if (sigNO < sigW) {
      cat(c("NOTE that the pointed Trends are afflicted by some kind of problem, a BETTER FITting model can be
    }
    asimErr <- skewness(woTrend$fit[, 1] - solution$OFFICIAL_YIELD)
    predError <- woTrend$fit[, 1] - solution$OFFICIAL_YIELD
    sigma <- sqrt(mean((predError - mean(predError))^2))
    danger <- asimErr/sigma
    if (danger >= 2.6) {
      cat(c("\n ADVICE: \n The marked trend related dynamics don't fit with the data! \n "))
    }
    if (sigNO < sigW | danger >= 2.6) {
        cat(c("Do you want to reset the trend marked and proceed again? (y/n) \n"),
            fill = T)
        reBea <- scan(, what = "text", nmax = 1)
        while (reBea != "y" & reBea != "n") {
            cat("answer y or n")
            reBea <- scan(, what = "text", nmax = 1)
        }
        if (reBea == "y") {
            rm(list = c("breakPoint", "flatYield", "due2trend",
                "friendShip", "flattyn", "safeTrend", "yieldTrend",
                "flatOff", "tableXregression", "model_formula",
                "CVmsRes", "expYield", "omniYield", "modelLM",
                "PCmodel"), envir = yieldPrev)
            virgilio()
        }
    }
  }
```

virgilio *guide you all along*

Description

It just provides a supervised path across the story. If you want to use other function step by step and not this one, the results are the same.

Usage

```
virgilio()
```

Details

Actually it doesn't do anything, but sort the other function (it doesn't save nor load) from earlier to later stages of the proceedings. It's something similar tto a scout or a tutor managing that everything works fine (NB: that's the aim).

Note

No argument are supposed when calling the function

See Also

configure()

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function ()
{
    if (any(ls() == "yieldPrev")) {
    }
    else {
        yieldPrev <- new.env()
        yieldPrev$.conflicts.OK <- c()
    }
    configure()
    checkTrends()
    while (yieldPrev$flattyn == "y") {
        breakTrends()
        checkTrends()
    }
    modSel()
    responseYield()
  }
```

`yieldPrev` *Workspace aimed to not get trubles in the main environment*

Description

For what I know there shouldn't be an omonymous environment caused by some(think/body) else. If it were it could be a proble as well as not...

Format

The format is: <environment: 0x7246a90>

Examples

```
By the way it is strongly related to save- and load- -session.
```