I used the Thanksgiving break to push a new update of the [TSstudio](https://github.com/RamiKrispin/TSstudio) package to CRAN (version 0.1.3). The new version includes an update for the ts\_backtesting function along with two new function – ts\_to\_prophet for converting time series objects to a [prophet](https://facebook.github.io/prophet/docs/quick_start.html#r-api) input format (i.e., ds and y columns), and ccf\_plot for lags plot between two time series. The package can be installed from either CRAN or Github:

# CRAN

install.packages("TSstudio")

# Github

# install.packages("devtools")

devtools::install\_github("RamiKrispin/TSstudio")

library(TSstudio)

packageVersion("TSstudio")

## [1] '0.1.3'

**Converting time series object to a prophet format**

The ts\_to\_prophet function converting ts, xts and zoo objects into prophet input format (i.e., data frame with two columns – ds for date and y for the series values). For instance, convertig the USgas series to a prophet object:

data("USgas")

ts\_info(USgas)

## The USgas series is a ts object with 1 variable and 225 observations

## Frequency: 12

## Start time: 2000 1

## End time: 2018 9

USgas\_prophet <- ts\_to\_prophet(USgas)

head(USgas)

## [1] 2510.5 2330.7 2050.6 1783.3 1632.9 1513.1

head(USgas\_prophet)

## ds y

## 1 2000-01-01 2510.5

## 2 2000-02-01 2330.7

## 3 2000-03-01 2050.6

## 4 2000-04-01 1783.3

## 5 2000-05-01 1632.9

## 6 2000-06-01 1513.1

In the case of a ts object, where the index is not a date object, the function extracts the time component from the first observation and use it along with the frequency of the series to estimate the date column of the prophet data frame. For instance, in the case of a monthly series, where the time object provides only the year and the month, by default the day component of the date object will be set to 1. Alternatively, if known, you can set the date of the first observation with the start argument. For example, if the USgas series is being captured during the mid of the month (or every 15th of the month):

USgas\_prophet <- ts\_to\_prophet(USgas, start = as.Date("2000-01-15"))

head(USgas\_prophet)

## ds y

## 1 2000-01-15 2510.5

## 2 2000-02-15 2330.7

## 3 2000-03-15 2050.6

## 4 2000-04-15 1783.3

## 5 2000-05-15 1632.9

## 6 2000-06-15 1513.1

Similarly, the function can handle xts and zoo objects:

data("EURO\_Brent")

ts\_info(EURO\_Brent)

## The EURO\_Brent series is a zoo object with 1 variable and 378 observations

## Frequency: monthly

## Start time: May 1987

## End time: Oct 2018

head(EURO\_Brent)

## May 1987 Jun 1987 Jul 1987 Aug 1987 Sep 1987 Oct 1987

## 18.58 18.86 19.86 18.98 18.31 18.76

ts\_to\_prophet(EURO\_Brent) %>% head()

## ds y

## 1 1987-05-01 18.58

## 2 1987-06-01 18.86

## 3 1987-07-01 19.86

## 4 1987-08-01 18.98

## 5 1987-09-01 18.31

## 6 1987-10-01 18.76

**Lags plots of two series**

The second function, ccf\_plot, provides an interactive and intuitive visualization of the cross-correlation between two time series, by plotting a series against another series (and its lags) and calculating the correlation between the two with the ccf function. For instance, let’s use the function to plot the relationship between the unemployment rate and the total vehicle sales in the US:

data("USUnRate")

ts\_info(USUnRate)

## The USUnRate series is a ts object with 1 variable and 850 observations

## Frequency: 12

## Start time: 1948 1

## End time: 2018 10

data("USVSales")

ts\_info(USVSales)

## The USVSales series is a ts object with 1 variable and 514 observations

## Frequency: 12

## Start time: 1976 1

## End time: 2018 10

ccf\_plot(x = USVSales, y = USUnRate)

The function automatically aligned and used only the overlapping observations of the two series before calculating the cross-correlation values between the series and the lags of the second series (where the 0 lag represents the series itself, and negative lags represent the leading lags). The title of each plot specifies the lag number and the cross-correlation value. The lags argument of the function defines the number of lags in the plot, where the use of negative lags defines the leading indicators. For example, setting the lags argument to -6:6 will plot the first 6 lags, the series itself and the first 6 leading lags of the series:

ccf\_plot(x = USVSales, y = USUnRate, lags = -6:6)

**Forecasting with backtesting and xreg**

The ts\_backtesting function for training and testing multiple models (e.g., auto.arima, HoltWinters, nnetar, etc.) with backtesting approach, is now supporting the xreg component of the auto.arima, nnetar ( [forecast](https://github.com/robjhyndman/forecast) package)and their embedment in the hybridModel model ( [forecastHybrid](https://github.com/ellisp/forecastHybrid) package). The use of the xreg component is straightforward and required two components:

* The predictors – or the regressors component in a vector or matric format will be used as an input to the model xreg argument. The length of this input must be aligned with the length of the input series
* The future values of the predictors – a vector or matrix must correspond to the inputs which used as predictors, where the length of this component must be aligned to the forecast horizon (or the h argument of the function). This setting of this component is done with the xreg.h argument

For instance, let’s forecast the monthly consumption of natural gas in US in the next 5 years (or 60 months) by regressing the USgas series with its Fourier terms, using auto.arima, nnetar and hybridModel models. We will use the fourier function from the forecast package to generate both the inputs for the regression model (x\_reg) and future values for the forecast itself (x\_reg.forecast):

# Setting the forecast horizon

h <- 60

library(forecast)

# Creating the xreg component for the regression

x\_reg <- fourier(USgas, K = 5)

# Creating the xreg component for the forecast

x\_reg.forecast <- forecast::fourier(USgas, K = 5, h = h)

Note that the ts\_backtesting function automatically split and aligned the xreg component according to the expanding window movement of the function. We will set the function to run backtesting using 6 periods/splits to train auto.arima, nnetar and hybridModel models, in order to examine the performance of the models over time:

md <- ts\_backtesting(ts.obj = USgas,

error = "MAPE",

models = "anh",

periods = 6,

h = h,

xreg.h = x\_reg.forecast,

a.arg = list(xreg = x\_reg),

h.arg = list(models = "aetsfz",

a.args = list(xreg = x\_reg),

verbose = FALSE),

n.arg = list(xreg = x\_reg),

plot = FALSE)

## Model\_Name avgMAPE sdMAPE avgRMSE sdRMSE

## 1 auto.arima 4.713333 1.448857 123.5483 33.67451

## 2 hybrid 4.786667 1.545039 122.6517 34.40697

## 3 nnetar 9.158333 2.652051 224.1967 60.38386

We can now review the performance of each model using the summary plot:

md$summary\_plot

The summary plot provides the error distribution of each model and the plot forecasting model which performed best on the backtesting. The output contains the models’ performance on the backtesting (i.e., summary plot and leaderboard). In this case, since we set the error argument to MAPE, the function selected the auto.arima final forecast. Yet, you can see in the plot that the error rate of the hybrid model is more stable compared to the auto.arima and it might be a better choice (the hybrid contains both the auto.arima and other models, which potentially helps to hedge the error). All the models’ information available on the Forecast\_Final folder. For example, you can pull the auto.arima model and check its residuals:

check\_res(md$Forecast\_Final$auto.arima)

The plan for future releases is to expend the functionality of the ts\_backtesting function, by adding additional models (e.g., tslm, prophet, etc.) and expend the window setting of the backtesting (adding sliding window option).