I am pleased to announce a new R package – forecastLM. The package, as the name implies, provides applications for forecasting regular time series data with a linear regression model (based on the lm function from the **stats** package). It supports both ts and tsibble objects as inputs and enables simple extractions of features from the input object on the fly. Example for such features:

* Single or Multiple seasonal components (when applicable)
* Different types of trends (regular, log, exponential, polynomial)
* Adding past lags
* Piecewise regression
* Variables selection with stepwise regression
* Handle events and outliers

In addition, it provides interactive data visualization tools utilizing the plotly package.

**Installation**

Library ("forecastLM")

**Forecasting the residential demand for natural gas in New York**

In the example below, we will use the trainLM function to train a linear regression model to forecast the residential demand for natural gas in New York state. As the goal is to demonstrate the key functions of the package, we will skip the descriptive analysis process (which generally you shouldn’t!) and focus on the training the forecasting process.

The demand for natural gas in New York state is one of the datasets available on the package and was sourced from the [US Energy Information Administration](https://www.eia.gov/) API. Let’s load the series and review its main characteristics:

library(forecastLM)

data("ny\_gas")

class(ny\_gas)

## [1] "tbl\_ts" "tbl\_df" "tbl" "data.frame"

head(ny\_gas)

## date y

## 1 1997-01-01 62019

## 2 1997-02-01 56003

## 3 1997-03-01 51213

## 4 1997-04-01 37597

## 5 1997-05-01 24082

## 6 1997-06-01 14469

The ny\_gas series is a regular time-series with monthly frequency in a tsibble format. We will use the ts\_plot function from the TStudio package to plot the series:

library(TSstudio)

ts\_plot(ny\_gas,

title = "The New York Natural Gas Residential Monthly Consumption",

Ytitle = "Million Cubic Feet",

Xtitle = "Source: US Energy Information Administration (Jan 2020)")

As can see in the plot above, the series has strong seasonality. Although it is not pronounced well in the plot, for simplicity, we will assume that the series has linear growth.

**Basic forecasting model**

We will start with a simplistic model, fitting a linear trend and monthly seasonality with the trend and seasonal arguments, respectivly:

md1 <- trainLM(input = ny\_gas,

y = "y",

trend = list(linear = TRUE),

seasonal = "month")

The trainLM function return a list with 5 objects:

* model – the lm model output
* fitted – the fitted values
* residuals – the residuals (i.e., actuals – fitted)
* parameters – the model parameters
* series – the intput series with the features that build to fit the model

summary(md1)

## Length Class Mode

## model 13 lm list

## fitted 2 data.frame list

## residuals 2 tbl\_ts list

## parameters 13 -none- list

## series 4 tbl\_ts list

You can use the summary function to view the model summary:

summary(md1$model)

##

## Call:

## stats::lm(formula = f, data = df1)

##

## Residuals:

## Min 1Q Median 3Q Max

## -21026.5 -2741.6 -412.8 2853.0 23582.2

##

## Coefficients:

## Estimate Std. Error t value Pr(>|t|)

## (Intercept) 66720.017 1340.437 49.7748 < 0.00000000000000022 \*\*\*

## monthFeb -5094.059 1704.010 -2.9895 0.003061 \*\*

## monthMar -12667.248 1704.027 -7.4337 0.000000000001514748 \*\*\*

## monthApr -34179.741 1704.056 -20.0579 < 0.00000000000000022 \*\*\*

## monthMay -49716.104 1704.096 -29.1745 < 0.00000000000000022 \*\*\*

## monthJun -57592.685 1704.148 -33.7956 < 0.00000000000000022 \*\*\*

## monthJul -60314.048 1704.211 -35.3912 < 0.00000000000000022 \*\*\*

## monthAug -61100.585 1704.285 -35.8512 < 0.00000000000000022 \*\*\*

## monthSep -60509.165 1704.371 -35.5023 < 0.00000000000000022 \*\*\*

## monthOct -53000.442 1704.468 -31.0950 < 0.00000000000000022 \*\*\*

## monthNov -34896.516 1723.350 -20.2492 < 0.00000000000000022 \*\*\*

## monthDec -14487.892 1723.401 -8.4066 0.000000000000002769 \*\*\*

## linear\_trend 27.059 4.416 6.1274 0.000000003276170518 \*\*\*

## ---

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##

## Residual standard error: 5778.6 on 261 degrees of freedom

## Multiple R-squared: 0.94076, Adjusted R-squared: 0.93804

## F-statistic: 345.41 on 12 and 261 DF, p-value: < 0.000000000000000222

The plot\_res function provides a residuals plot. Besides the standard residual plot (e.g., residuals, residauls correlation and normality) it provides on the top of the plot a comparson view of the fitted values against the series itself. This is where the power of interactivity come into action as it enables to:

* Quickly identify outliers or observations that the model did not fit well, and linked to the event in the series itself
* As the residuals and the fitted vs. actuals plots are both linked, when zoom-in on one, it will automatically will zoom-in on the corresponding observations on the second plot (try it!)

plot\_res(md1)

**Adding lags regressors**

As can see in the ACF plot on the residual plot above, the residuals are correlated. This mainly implies that some patterns left on the data which the model did not capture. One way to capture those patterns is by regress the series with its past lags. This is generally equivalent to an AR (Autoregressive) process. The lags argument enables you to add lags regressor by defining the lags number. For example, let’s add to the model the first and seasonal lags by setting the lags argument to c(1, 12):

md2 <- trainLM(input = ny\_gas,

y = "y",

trend = list(linear = TRUE),

seasonal = "month",

lags = c(1,12))

summary(md2$model)

##

## Call:

## stats::lm(formula = f, data = df1)

##

## Residuals:

## Min 1Q Median 3Q Max

## -16244.65 -1722.06 -127.02 1737.53 15241.24

##

## Coefficients:

## Estimate Std. Error t value Pr(>|t|)

## (Intercept) 32265.964782 4365.570790 7.3910 0.00000000000224972 \*\*\*

## monthFeb -12055.401516 1722.177908 -7.0001 0.00000000002402279 \*\*\*

## monthMar -16236.106785 1758.014778 -9.2355 < 0.00000000000000022 \*\*\*

## monthApr -31687.887252 2407.133951 -13.1642 < 0.00000000000000022 \*\*\*

## monthMay -34236.087062 3128.014939 -10.9450 < 0.00000000000000022 \*\*\*

## monthJun -33093.376259 3698.769808 -8.9471 < 0.00000000000000022 \*\*\*

## monthJul -31397.220717 3977.363059 -7.8940 0.00000000000009549 \*\*\*

## monthAug -30800.918841 4071.729734 -7.5646 0.00000000000076668 \*\*\*

## monthSep -29800.859834 4068.696270 -7.3244 0.00000000000338633 \*\*\*

## monthOct -23509.033810 3764.733548 -6.2445 0.00000000184222790 \*\*\*

## monthNov -11343.231562 2975.266737 -3.8125 0.0001738 \*\*\*

## monthDec -2268.087593 1939.589740 -1.1694 0.2433840

## linear\_trend 10.714705 4.418472 2.4250 0.0160272 \*

## lag\_1 0.513078 0.054328 9.4441 < 0.00000000000000022 \*\*\*

## lag\_12 0.118482 0.056521 2.0962 0.0370788 \*

## ---

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##

## Residual standard error: 4895.8 on 247 degrees of freedom

## Multiple R-squared: 0.95857, Adjusted R-squared: 0.95622

## F-statistic: 408.16 on 14 and 247 DF, p-value: < 0.000000000000000222

You can see on the summary of the model above that two additional variables were added by the trainLM function – lag\_1 and lag\_12, corresponding to the first and seasonal lags of the series. Those new features added to the input series and available on the output of the trained model:

head(md2$series, 13)

## # A tsibble: 13 x 6 [1M]

## date y month linear\_trend lag\_1 lag\_12

##

## 1 1997 Jan 62019 Jan 1 NA NA

## 2 1997 Feb 56003 Feb 2 62019 NA

## 3 1997 Mar 51213 Mar 3 56003 NA

## 4 1997 Apr 37597 Apr 4 51213 NA

## 5 1997 May 24082 May 5 37597 NA

## 6 1997 Jun 14469 Jun 6 24082 NA

## 7 1997 Jul 9719 Jul 7 14469 NA

## 8 1997 Aug 10267 Aug 8 9719 NA

## 9 1997 Sep 9878 Sep 9 10267 NA

## 10 1997 Oct 17385 Oct 10 9878 NA

## 11 1997 Nov 34936 Nov 11 17385 NA

## 12 1997 Dec 48074 Dec 12 34936 NA

## 13 1998 Jan 54949 Jan 13 48074 62019

In this case, as can notice in the residuals plot below, the lag variables helped in reducing the correlation of the residuals.

plot\_res(md2)

**Note:** When forecasting the feature values of the series with the forecastLM function (see example below), for lag n the function will automatically map the last n observations for the corresponding first n observations. If the forecast horizon is greater than n observations, it will utilize the corresponding predicted values from observation n+1 and moving forward.

**Variables selection with stepwise regression**

The step argument enables you to apply a stepwise regression method for variable selection based on the [AIC](https://en.wikipedia.org/wiki/Akaike_information_criterion) (Akaike Information Criterion). For example, let’s add some noise by adding the 6th lag, in addition to the 1st and 12th lags:

md3 <- trainLM(input = ny\_gas,

y = "y",

trend = list(linear = TRUE),

seasonal = "month",

lags = c(1, 6, 12),

step = TRUE)

## Start: AIC=4467.39

## y ~ month + linear\_trend + lag\_1 + lag\_6 + lag\_12

##

## Df Sum of Sq RSS AIC

## - lag\_6 1 25831874 5920397742 4466.53

## 5894565868 4467.39

## - lag\_12 1 91670535 5986236404 4469.43

## - linear\_trend 1 166735011 6061300879 4472.69

## - lag\_1 1 2023866568 7918432436 4542.72

## - month 11 5233830014 11128395882 4611.88

##

## Step: AIC=4466.53

## y ~ month + linear\_trend + lag\_1 + lag\_12

##

## Df Sum of Sq RSS AIC

## 5920397742 4466.53

## - lag\_12 1 105325824 6025723567 4469.15

## - linear\_trend 1 140951656 6061349399 4470.70

## - lag\_1 1 2137859906 8058257649 4545.30

## - month 11 5687037904 11607435647 4620.92

summary(md3$model)

##

## Call:

## stats::lm(formula = y ~ month + linear\_trend + lag\_1 + lag\_12,

## data = df1)

##

## Residuals:

## Min 1Q Median 3Q Max

## -16244.65 -1722.06 -127.02 1737.53 15241.24

##

## Coefficients:

## Estimate Std. Error t value Pr(>|t|)

## (Intercept) 32265.964782 4365.570790 7.3910 0.00000000000224972 \*\*\*

## monthFeb -12055.401516 1722.177908 -7.0001 0.00000000002402279 \*\*\*

## monthMar -16236.106785 1758.014778 -9.2355 < 0.00000000000000022 \*\*\*

## monthApr -31687.887252 2407.133951 -13.1642 < 0.00000000000000022 \*\*\*

## monthMay -34236.087062 3128.014939 -10.9450 < 0.00000000000000022 \*\*\*

## monthJun -33093.376259 3698.769808 -8.9471 < 0.00000000000000022 \*\*\*

## monthJul -31397.220717 3977.363059 -7.8940 0.00000000000009549 \*\*\*

## monthAug -30800.918841 4071.729734 -7.5646 0.00000000000076668 \*\*\*

## monthSep -29800.859834 4068.696270 -7.3244 0.00000000000338633 \*\*\*

## monthOct -23509.033810 3764.733548 -6.2445 0.00000000184222790 \*\*\*

## monthNov -11343.231562 2975.266737 -3.8125 0.0001738 \*\*\*

## monthDec -2268.087593 1939.589740 -1.1694 0.2433840

## linear\_trend 10.714705 4.418472 2.4250 0.0160272 \*

## lag\_1 0.513078 0.054328 9.4441 < 0.00000000000000022 \*\*\*

## lag\_12 0.118482 0.056521 2.0962 0.0370788 \*

## ---

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##

## Residual standard error: 4895.8 on 247 degrees of freedom

## Multiple R-squared: 0.95857, Adjusted R-squared: 0.95622

## F-statistic: 408.16 on 14 and 247 DF, p-value: < 0.000000000000000222

As expected, the stepwise regression dropped the 6th lag variable.

**Handling special events and outliers**

The events argument allows you to handle special events (e.g., non-seasonal) and outliers. This argument takes a list of input (or inputs), each represents a set of date or time that corresponding to the series index. It than transform those inputs to flag variables with the use of a hot-encoding method.  
For instance, you can observe from the series plot above that the peaks during the years 2015, 2018, and 2019 were higher than the usual. Potentially, This could be related to unusual weather patterns such as colder temperatures than the average. You can observe on the above-trained models’ residuals plots that the residuals during those peaks were significantly higher due to a bad fit of the model. Let’s flag those events with the events argument:

events1 <- list(outlier = c(as.Date("2015-01-01"), as.Date("2015-02-01"), as.Date("2018-01-01"), as.Date("2019-01-01")))

md4 <- trainLM(input = ny\_gas,

y = "y",

trend = list(linear = TRUE),

seasonal = "month",

lags = c(1, 12),

events = events1,

step = TRUE)

## Start: AIC=4438.35

## y ~ outlier + month + linear\_trend + lag\_1 + lag\_12

##

## Df Sum of Sq RSS AIC

## - lag\_12 1 30530567 5306830893 4437.87

## 5276300326 4438.35

## - linear\_trend 1 127518855 5403819181 4442.61

## - outlier 1 644097417 5920397742 4466.53

## - lag\_1 1 1513563678 6789864004 4502.43

## - month 11 5817216417 11093516743 4611.06

##

## Step: AIC=4437.87

## y ~ outlier + month + linear\_trend + lag\_1

##

## Df Sum of Sq RSS AIC

## 5306830893 4437.87

## - linear\_trend 1 166154848 5472985741 4443.94

## - outlier 1 718892674 6025723567 4469.15

## - lag\_1 1 1614763442 6921594336 4505.47

## - month 11 37114665006 42421495899 4960.48

summary(md4$model)

##

## Call:

## stats::lm(formula = y ~ outlier + month + linear\_trend + lag\_1,

## data = df1)

##

## Residuals:

## Min 1Q Median 3Q Max

## -16819.93 -1493.53 -18.83 1690.59 16469.41

##

## Coefficients:

## Estimate Std. Error t value Pr(>|t|)

## (Intercept) 41674.663767 2884.524303 14.4477 < 0.00000000000000022 \*\*\*

## outlier 15018.386255 2596.332971 5.7845 0.0000000219178283339 \*\*\*

## monthFeb -10354.582199 1643.264690 -6.3012 0.0000000013456455566 \*\*\*

## monthMar -15074.741500 1559.504225 -9.6664 < 0.00000000000000022 \*\*\*

## monthApr -33447.774825 1452.659748 -23.0252 < 0.00000000000000022 \*\*\*

## monthMay -39146.223276 1722.408636 -22.7276 < 0.00000000000000022 \*\*\*

## monthJun -39847.357090 2276.907447 -17.5006 < 0.00000000000000022 \*\*\*

## monthJul -38933.664027 2605.992821 -14.9401 < 0.00000000000000022 \*\*\*

## monthAug -38583.581362 2721.780637 -14.1759 < 0.00000000000000022 \*\*\*

## monthSep -37559.002207 2759.521582 -13.6107 < 0.00000000000000022 \*\*\*

## monthOct -30334.701374 2731.190821 -11.1068 < 0.00000000000000022 \*\*\*

## monthNov -15705.106848 2409.019610 -6.5193 0.0000000003948420140 \*\*\*

## monthDec -3101.475052 1754.057367 -1.7682 0.078267 .

## linear\_trend 11.330305 4.074312 2.7809 0.005838 \*\*

## lag\_1 0.453819 0.052348 8.6693 0.0000000000000005852 \*\*\*

## ---

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##

## Residual standard error: 4635.2 on 247 degrees of freedom

## Multiple R-squared: 0.96286, Adjusted R-squared: 0.96075

## F-statistic: 457.39 on 14 and 247 DF, p-value: < 0.000000000000000222

The outlier variable created and added to the regression by the function. You can notice in the residuals plot below the effect of the variable on the fit of the model:

plot\_res(md4)

**Forecasting**

Once the model is trained, forecasting the future observations of the series is strigthforward with the forecastLM function. The forecastLM function is fairly similar to the forecast function from the forecast package and it has the following arguments:

* model – an trainLM object
* newdata – a tsibble object with the future values of the external regressors (is used as an input for to train the model)
* h – the horizon of the forecast
* pi – the level of the prediction intervals, by default will calculate the 80% and 95% prediction intervals

fc4 <- forecastLM(md4, h = 60)

The plot\_fc function returns a visualization of the forecasted object:

plot\_fc(fc4)

The plot\_fc has six customized color themes. By default, the function will use the normal theme (as the plot above), which is similar to the **base** R plot function color theme (prediction intervals with shaded gray colors). In addition to the normal theme, there are classic, darkBlue, darkPink, darkGreen, and lightBeige themes. For example, we will change the theme to darkPink with the theme argument:

plot\_fc(fc4, theme = "darkPink")