**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

*# Plots*

**import** **matplotlib.pyplot** **as** **plt**

plt.style.use('fivethirtyeight')

plt.rcParams['lines.linewidth'] = 1.5

%**matplotlib** inline

*# Warnings configuration*

**import** **warnings**

warnings.filterwarnings('ignore')

[**Skforecast**](https://github.com/JoaquinAmatRodrigo/skforecast), a library containing the classes and functions needed to adapt any **Scikit-learn** regression model to forecasting problems, is used. It can be installed in the following ways:

pip install skforecast

A specific version:

pip install skforecast==0.3

Last version (unstable):

pip install git+https://github.com/JoaquinAmatRodrigo/skforecast#master

**from** **sklearn.linear\_model** **import** LinearRegression

**from** **sklearn.linear\_model** **import** Lasso

**from** **sklearn.ensemble** **import** RandomForestRegressor

**from** **sklearn.metrics** **import** mean\_squared\_error

**from** **sklearn.preprocessing** **import** StandardScaler

**from** **sklearn.pipeline** **import** make\_pipeline

**from** **skforecast.ForecasterAutoreg** **import** ForecasterAutoreg

**from** **skforecast.ForecasterAutoregCustom** **import** ForecasterAutoregCustom

**from** **skforecast.ForecasterAutoregMultiOutput** **import** ForecasterAutoregMultiOutput

**from** **skforecast.model\_selection** **import** grid\_search\_forecaster

**from** **skforecast.model\_selection** **import** backtesting\_forecaster

**from** **joblib** **import** dump, load

*# Data download*

url = 'https://raw.githubusercontent.com/JoaquinAmatRodrigo/skforecast/master/data/h2o\_exog.csv'

data = pd.read\_csv(url, sep=',')

*# Data preparation*

data = data.rename(columns={'fecha': 'date'})

data['date'] = pd.to\_datetime(data['date'], format='%Y/%m/**%d**')

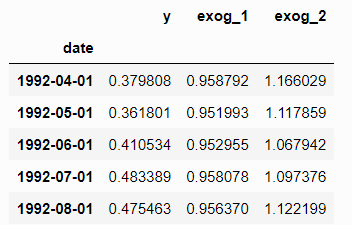
data = data.set\_index('date')

data = data.rename(columns={'x': 'y'})

data = data.asfreq('MS')

data = data.sort\_index()

data.head()



*# Verify that a temporary index is complete*

(data.index == pd.date\_range(start=data.index.min(),

end=data.index.max(),

freq=data.index.freq)).all()

*# Fill gaps in a temporary index*

*# data.asfreq(freq='30min', fill\_value=np.nan)*

*# Split data into train-test*

*# Last 36 months are for test*

steps = 36

data\_train = data[:-steps] # BLUE

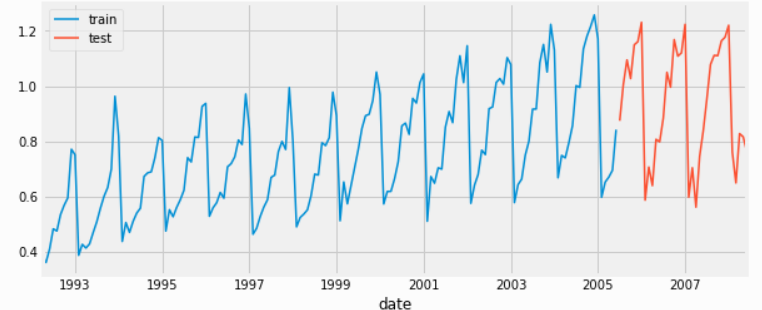
data\_test = data[-steps:] # RED

fig, ax=plt.subplots(figsize=(9, 4))

data\_train['y'].plot(ax=ax, label='train')

data\_test['y'].plot(ax=ax, label='test')

ax.legend();



**ForecasterAutoreg**

With the ForecasterAutoreg class, a model is created and trained from a RandomForestRegressor regressor with a time window of 6 lags. This means that the model uses the previous 6 months as predictors.

*# Create and train forecaster*

forecaster = ForecasterAutoreg(

regressor = RandomForestRegressor(random\_state=123),

lags = 6

)

forecaster.fit(y=data\_train['y'])

forecaster

**Predictions**

Once the model is trained, the test data is predicted (36 months into the future).

*# Predictions*

steps = 36

predictions = forecaster.predict(steps=steps)

predictions.head(5)

fig, ax = plt.subplots(figsize=(9, 4))

data\_train['y'].plot(ax=ax, label='train') # BLUE

data\_test['y'].plot(ax=ax, label='test') # RED

predictions.plot(ax=ax, label='predictions') # ORANGE

ax.legend();

Chart

Description automatically generated

### Prediction error in the test set

The error that the model makes in its predictions is quantified. In this case, the metric used is the mean squared error (mse).

*# Error*

error\_mse = mean\_squared\_error(

y\_true = data\_test['y'],

y\_pred = predictions

)

print(f"Test error (mse): **{**error\_mse**}**")

Test error (mse): 0.073268

### Hyperparameter tuning

The trained ForecasterAutoreg uses a 6 lag time window and a [Random Forest](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html) model with the default hyperparameters. However, there is no reason why these values are the most suitable.

In order to identify the best combination of lags and hyperparameters, **Skforecast** has the grid\_search\_forecaster function. This function uses [backtesting](https://joaquinamatrodrigo.github.io/skforecast/latest/guides/backtesting.html) to compare the performance obtained with each configuration of the model. It is important not to include the test data in the search process to avoid overfitting problems.

In this example, for the first fold, the initial 50% of the observations are the training data and, the next 10 steps represent the validation set. In successive folds, the training set will contain all data used in the previous fold and, the next 10 steps will be used as new validation data. This process will be repeated until the entire training data set is used.

*# Hyperparameter Grid search*

forecaster = ForecasterAutoreg(

regressor = RandomForestRegressor(random\_state=123),

lags = 12 *# This value will be replaced in the grid search*

)

*# Regressor's hyperparameters*

param\_grid = {'n\_estimators': [100, 500],

'max\_depth': [3, 5, 10]}

*# Lags used as predictors*

lags\_grid = [10, 20]

results\_grid = grid\_search\_forecaster(

forecaster = forecaster,

y = data\_train['y'],

param\_grid = param\_grid,

lags\_grid = lags\_grid,

steps = 10,

refit = **True**,

metric = 'mean\_squared\_error',

initial\_train\_size = int(len(data\_train)\*0.5),

return\_best = **True**,

verbose = **False**

)

Number of models compared: 12

loop lags\_grid: 0%| | 0/2 [00:00<?, ?it/s]

loop param\_grid: 0%| | 0/6 [00:00<?, ?it/s]

loop param\_grid: 17%|██████▎ | 1/6 [00:01<00:07, 1.45s/it]

loop param\_grid: 33%|████████████▋ | 2/6 [00:07<00:16, 4.04s/it]

loop param\_grid: 50%|███████████████████ | 3/6 [00:08<00:08, 2.88s/it]

loop param\_grid: 67%|█████████████████████████▎ | 4/6 [00:16<00:09, 4.97s/it]

loop param\_grid: 83%|███████████████████████████████▋ | 5/6 [00:18<00:03, 3.77s/it]

loop param\_grid: 100%|██████████████████████████████████████| 6/6 [00:26<00:00, 5.08s/it]

loop lags\_grid: 50%|███████████████████▌ | 1/2 [00:26<00:26, 26.26s/it]

loop param\_grid: 0%| | 0/6 [00:00<?, ?it/s]

loop param\_grid: 17%|██████▎ | 1/6 [00:02<00:13, 2.70s/it]

loop param\_grid: 33%|████████████▋ | 2/6 [00:15<00:33, 8.43s/it]

loop param\_grid: 50%|███████████████████ | 3/6 [00:16<00:15, 5.28s/it]

loop param\_grid: 67%|█████████████████████████▎ | 4/6 [00:25<00:13, 6.66s/it]

loop param\_grid: 83%|███████████████████████████████▋ | 5/6 [00:27<00:04, 4.91s/it]

loop param\_grid: 100%|██████████████████████████████████████| 6/6 [00:35<00:00, 6.05s/it]

loop lags\_grid: 100%|███████████████████████████████████████| 2/2 [01:01<00:00, 30.90s/it]

`Forecaster` refitted using the best-found lags and parameters, and the whole data set:

Lags: [ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]

Parameters: {'max\_depth': 3, 'n\_estimators': 500}

Backtesting metric: 0.008055003809735634

*# Grid Search results*

results\_grid

|  | **lags** | **params** | **metric** | **max\_depth** | **n\_estimators** |
| --- | --- | --- | --- | --- | --- |
|  | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... | {'max\_depth': 3, 'n\_estimators': 500} | 0.008055 | 3 | 500 |
| **6** | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... | {'max\_depth': 3, 'n\_estimators': 100} | 0.008220 | 3 | 100 |
| **8** | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... | {'max\_depth': 5, 'n\_estimators': 100} | 0.008326 | 5 | 100 |
| **9** | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... | {'max\_depth': 5, 'n\_estimators': 500} | 0.008330 | 5 | 500 |
| **11** | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... | {'max\_depth': 10, 'n\_estimators': 500} | 0.008336 | 10 | 500 |
| **10** | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... | {'max\_depth': 10, 'n\_estimators': 100} | 0.008359 | 10 | 100 |
| **5** | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] | {'max\_depth': 10, 'n\_estimators': 500} | 0.028249 | 10 | 500 |
| **3** | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] | {'max\_depth': 5, 'n\_estimators': 500} | 0.028386 | 5 | 500 |
| **1** | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] | {'max\_depth': 3, 'n\_estimators': 500} | 0.028830 | 3 | 500 |
| **0** | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] | {'max\_depth': 3, 'n\_estimators': 100} | 0.028883 | 3 | 100 |
| **4** | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] | {'max\_depth': 10, 'n\_estimators': 100} | 0.029012 | 10 | 100 |
| **2** | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] | {'max\_depth': 5, 'n\_estimators': 100} | 0.029949 | 5 | 100 |

The best results are obtained using a time window of 20 lags and a Random Forest set up of {'max\_depth': 3, 'n\_estimators': 500}.

### Final model

Finally, a ForecasterAutoreg is trained with the optimal configuration found by validation. This step is not necessary if return\_best = True is specified in the grid\_search\_forecaster() function.

* max\_depth = 3 (3,5, 10)
* n\_estimators = 500 (100,500)
* random\_state = 123
* lags = 20 Usara los 20 puntos previos como predictor
* steps = 36 Numero de periodos

*# Create and train forecaster with the best hyperparameters*

regressor = RandomForestRegressor(max\_depth=**3**, n\_estimators=**500**, random\_state=**123**)

forecaster = ForecasterAutoreg(

regressor = regressor,

lags = **20**

)

forecaster.fit(y=data\_train['y'])

*# Predictions*

predictions = forecaster.predict(steps=steps)

*# Plot*

fig, ax = plt.subplots(figsize=(9, 4))

**data\_train['y'].**plot(ax=ax, label='train')

**data\_test['y'].**plot(ax=ax, label='test')

**predictions**.plot(ax=ax, label='predictions')

ax.legend();

Chart

Description automatically generated

*# Error de test*

error\_mse = mean\_squared\_error(

y\_true = data\_test['y'],

y\_pred = predictions

)

print(f"Test error (mse): **{**error\_mse**}**")

Test error (mse): 0.004392699

The optimal combination of hyperparameters significantly reduces test error.

## Backtesting

The [Backtesting](https://joaquinamatrodrigo.github.io/skforecast/latest/guides/cross-validation-backtest.html) process consists of simulating the behavior that the model would have had if it had been run on a recurring basis, for example, predicting at intervals of 3 years (36 months) a total of 9 years. This type of evaluation can be easily applied with the backtesting\_forecaster() function.

**Backtesting with refit**

The model is trained each tieme before making the predictions, in this way, the model use all the information available so far. It is a variation of the standar cross-validation but, instead of making a random distribution of the observations, the training set is increased sequentially, maintaining the temporal order of the data.

Chart

Description automatically generated

**Backtesting without refit**

After an initial train, the model is used sequentially without updating it and following the temporal order of the data. This strategy has the advantage of being much faster since the model is only trained once. However, the model does not incorporate the latest information available so it may lose predictive capacity over time.

Chart, waterfall chart

Description automatically generated

*# Backtesting*

n\_test = 36\*3 *# The last 9 years are separated for the backtest*

data\_train = data[:-n\_test]

data\_test = data[-n\_test:]

metric, predictions\_backtest = backtesting\_forecaster(

forecaster = forecaster,

y = data['y'],

initial\_train\_size = len(data\_train),

steps = steps,

metric = 'mean\_squared\_error',

refit = **True**,

verbose = **True**

)

print(f"Backtest error: **{**metric**}**")

Information of backtesting process

----------------------------------

Number of observations used for initial training: 87

Number of observations used for backtesting: 108

Number of folds: 3

Number of steps per fold: 36

Data partition in fold: 0

Training: 1992-04-01 00:00:00 -- 1999-06-01 00:00:00

Validation: 1999-07-01 00:00:00 -- 2002-06-01 00:00:00

Data partition in fold: 1

Training: 1992-04-01 00:00:00 -- 2002-06-01 00:00:00

Validation: 2002-07-01 00:00:00 -- 2005-06-01 00:00:00

Data partition in fold: 2

Training: 1992-04-01 00:00:00 -- 2005-06-01 00:00:00

Validation: 2005-07-01 00:00:00 -- 2008-06-01 00:00:00

Backtest error: [0.01057898]

In [20]:

fig, ax = plt.subplots(figsize=(9, 4))

data\_test['y'].plot(ax=ax, label='test')

predictions\_backtest.plot(ax=ax, label='predictions')

ax.legend();

Chart, line chart

Description automatically generated

### Predictors importance

Since the ForecasterAutoreg object uses **Scikit-learn** models, the importance of predictors can be accessed once trained. When the regressor used is a LinearRegression(), Lasso() or Ridge(), the coefficients of the model reflect their importance, obtained with the get\_coef() method. In GradientBoostingRegressor() or RandomForestRegressor() regressors, the importance of predictors is based on impurity reduction and is accessible through the get\_feature\_importance() method. In both cases, the values returned are sorted as the lags order.

*# Predictors importance*

forecaster.get\_feature\_importance()

|  | **feature** | **importance** |
| --- | --- | --- |
| **0** | lag\_1 | 0.009412 |
| **1** | lag\_2 | 0.087268 |
| **2** | lag\_3 | 0.012754 |
| **3** | lag\_4 | 0.001446 |
| **4** | lag\_5 | 0.000401 |
| **5** | lag\_6 | 0.001386 |
| **6** | lag\_7 | 0.001273 |
| **7** | lag\_8 | 0.006926 |
| **8** | lag\_9 | 0.005839 |
| **9** | lag\_10 | 0.013076 |
| **10** | lag\_11 | 0.008868 |
| **11** | lag\_12 | 0.816041 |
| **12** | lag\_13 | 0.001266 |
| **13** | lag\_14 | 0.019411 |
| **14** | lag\_15 | 0.008746 |
| **15** | lag\_16 | 0.001766 |
| **16** | lag\_17 | 0.000578 |
| **17** | lag\_18 | 0.000329 |
| **18** | lag\_19 | 0.000853 |
| **19** | lag\_20 | 0.002359 |