Time Series Forecasting with Tree-Based Ensemble Methods: Part II

April 15, 2022

About the Presenter

- Brett Efaw
- Data Scientist at Idaho Power Company
- BEfaw@idahopower.com or brettefaw@gmail.com
- Work with various business groups to implement Data Science solutions
  - Decision support
  - Predictive modeling
  - Forecasting
  - BI and DS reporting/analysis applications

Overview

- Part I (DS Seminar 2022-02-25):
  - [data_science_seminar/2022_1 at master · lselector/data_science_seminar · GitHub]
  - Time series forecasting approaches
  - Compare XGBoost to Auto-ARIMA on a simple simulated data set
- Part II (DS Seminar 2022-04-15):
  - Use a more complex simulated data set
  - Real world data set (from Kaggle)
  - Auto-ARIMA with exogenous regressors
- Simulated time series data set
  - Multiple cyclic patterns/seasonality
  - Non-linear trend
- Modeling/forecasting process
  - Extract the “trend” from the time series
  - Model 1: Fit and forecast the trend component
  - Model 2: Use XGBoost (or similar tree-based algo) to model remainder
  - Forecast = combination of model 1 + model 2 predictions

(A More Interesting) Simulated Data Set

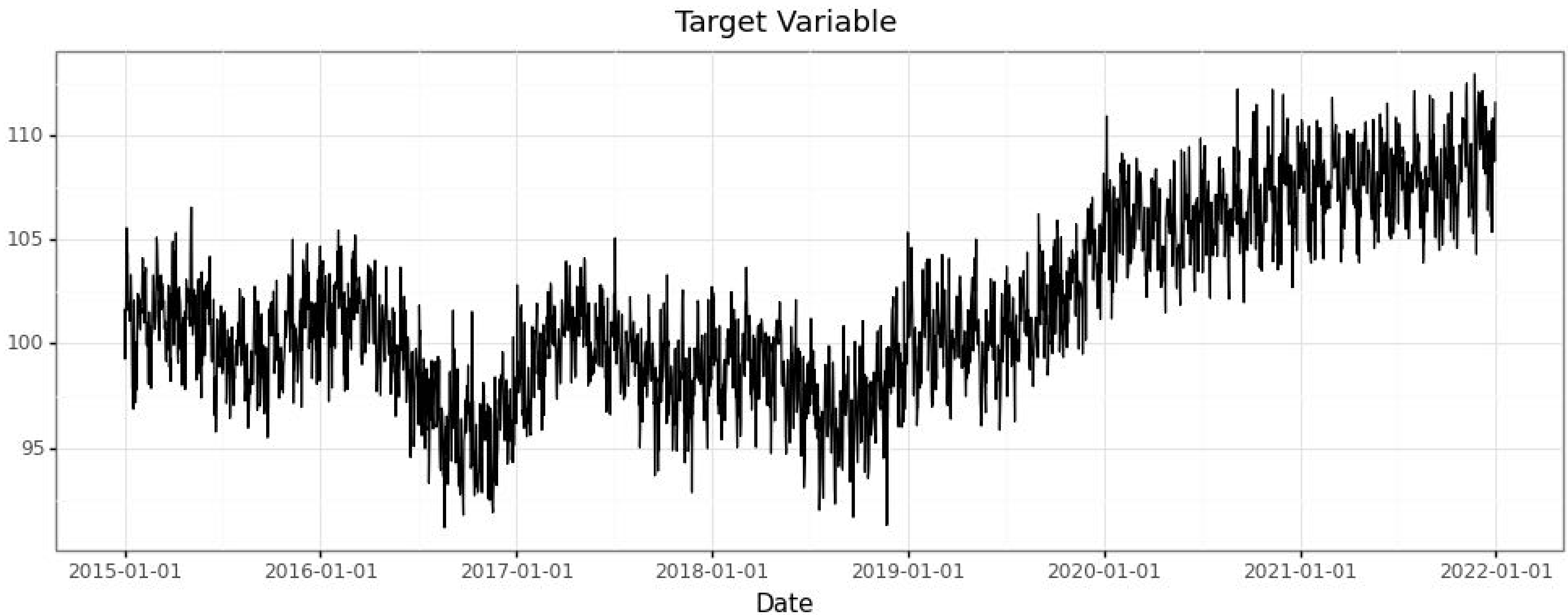
- Cyclic pattern features:
  - Day of week
  - Day of month
  - Day of year
- Sequence features:
  - Day number
  - Year
- Holiday effects:
  - Holiday
  - Day before/after holiday
- More realistic trend:
  - Non-linear
  - Randomly-spaced up/down steps
    - numpy.random.choice()
  - Smoothed out across time
    - savgol_filter()

```
df["dw"] = df["dt"].dt.dayofweek+1
df["dm"] = df["dt"].dt.day
df["dy"] = df["dt"].dt.dayofyear
df["dn"] = df.index+1
df["yr"] = df["dt"].dt.year
```

```
cal = calendar()
holidays = cal.holidays(start = df["dt"].min(), end = df["dt"].max())
df["hd"] = df["dt"].isin(holidays).astype(int)          # Federal Holiday
df["bh"] = df["hd"].shift(periods = -1, fill_value = 0)  # Day Before Holiday
df["ah"] = df["hd"].shift(periods = +1, fill_value = 0)  # Day After Holiday
```

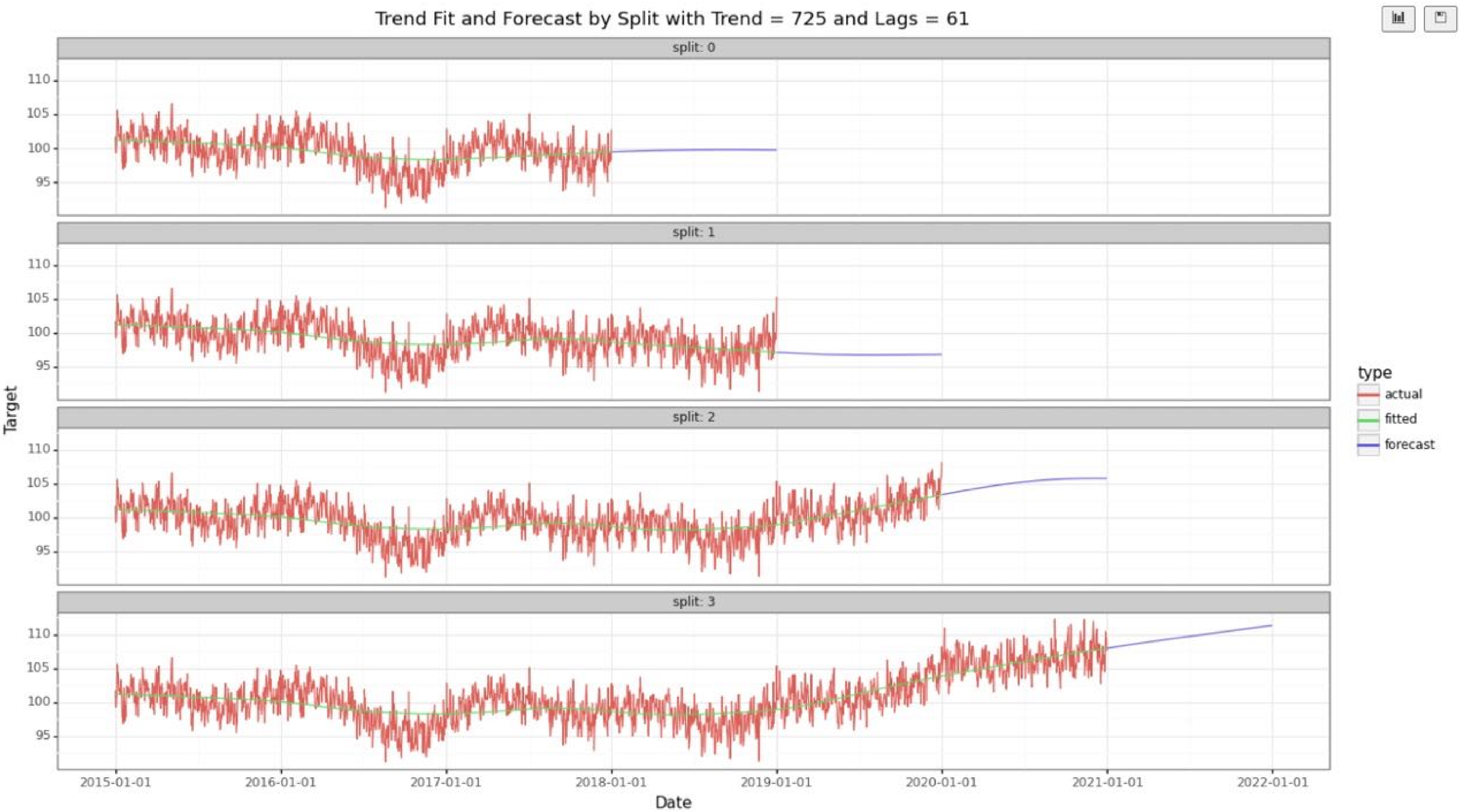
```
# Noise for each date
df["noise"] = np.random.normal(0, 1.0, len(df))
# Step changes randomly dispersed throughout the time series
chngpts = sorted(np.random.choice(df["dn"], size=50, replace=False))
df["change"] = np.where(df["dn"].isin(chngpts), 1, 0)
df["step"] = np.random.choice([-1,1], size=len(df), p=[0.5,0.5]) * df["change"]
df["step"] = df["step"].cumsum()
# Smooth step changes (i.e. non-linear trend)
df["trend"] = savgol_filter(df["step"].values, window_length=100, polyorder=1)
```

	dt	dw	dm	dy	dn	yr	sdw	cdw	sdm	cdm	sdy	cdy	hd	bh	ah	noise	change	step	trend	y
0	2015-01-01	4	1	1	1	2015	0.433884	-0.900969	0.000000	1.000000	0.000000	1.000000	1	0	0	-1.626860	0	0	-0.217624	101.688431
1	2015-01-02	5	2	2	2	2015	-0.433884	-0.900969	0.201299	0.979530	0.017166	0.999853	0	0	1	0.613607	0	0	-0.210399	99.266203
2	2015-01-03	6	3	3	3	2015	-0.974928	-0.222521	0.394356	0.918958	0.034328	0.999411	0	0	0	-1.501417	0	0	-0.203174	99.445012
3	2015-01-04	7	4	4	4	2015	-0.781831	0.623490	0.571268	0.820763	0.051479	0.998674	0	0	0	-0.344402	0	0	-0.195950	101.743491
4	2015-01-05	1	5	5	5	2015	0.000000	1.000000	0.724793	0.688967	0.068615	0.997643	0	0	0	2.279467	0	0	-0.188725	105.570760

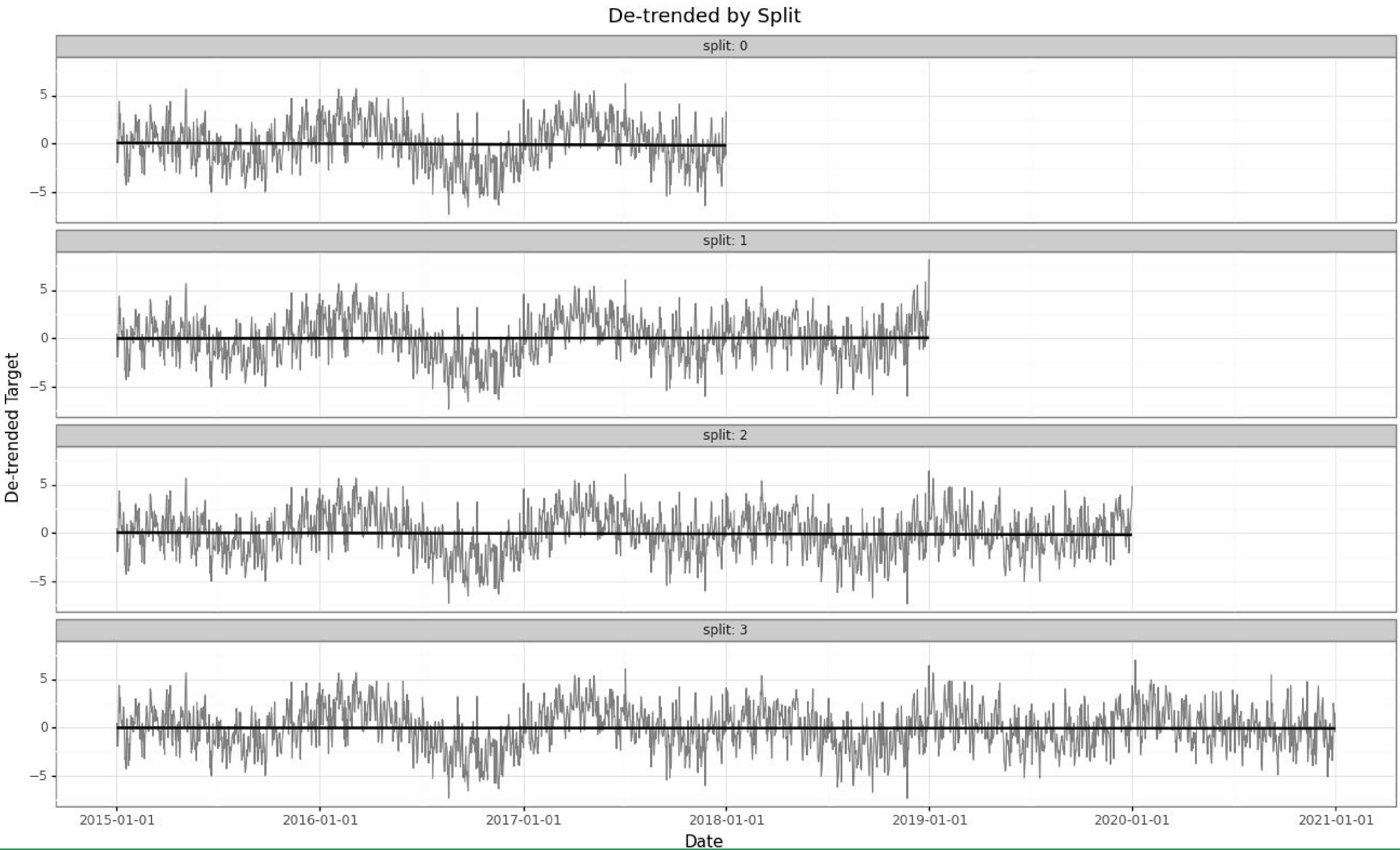


Model 1: Fit and Forecast Trend

- M1-a. Extract the trend
  - Use statsmodels.tsa.seasonal.STL() to decompose the time series (i.e. extract the trend)
  - Trend = the number of time steps to include in the decomposition
    - larger values = smoother trend (favors long-term)
    - smaller values = wigglier trend (favors short-term)
- M1-b. Model the trend
  - Use statsmodels.tsa.ar_model.AutoReg() to model (and forecast) the trend
  - Lags = the number of time steps prior to consider as lagged inputs in regression model
- M1-c. Detrend the past (need these for Model 2)
- M1-d. Forecast the trend using model in M1-b
- Other parameters
  - Period = the frequency of the time series (daily = 7, monthly = 12, quarterly = 4, etc.)
  - Number of splits = the number of train/test splits for back-casting / model evaluation
  - Test size = the number of time steps in the forecast horizon

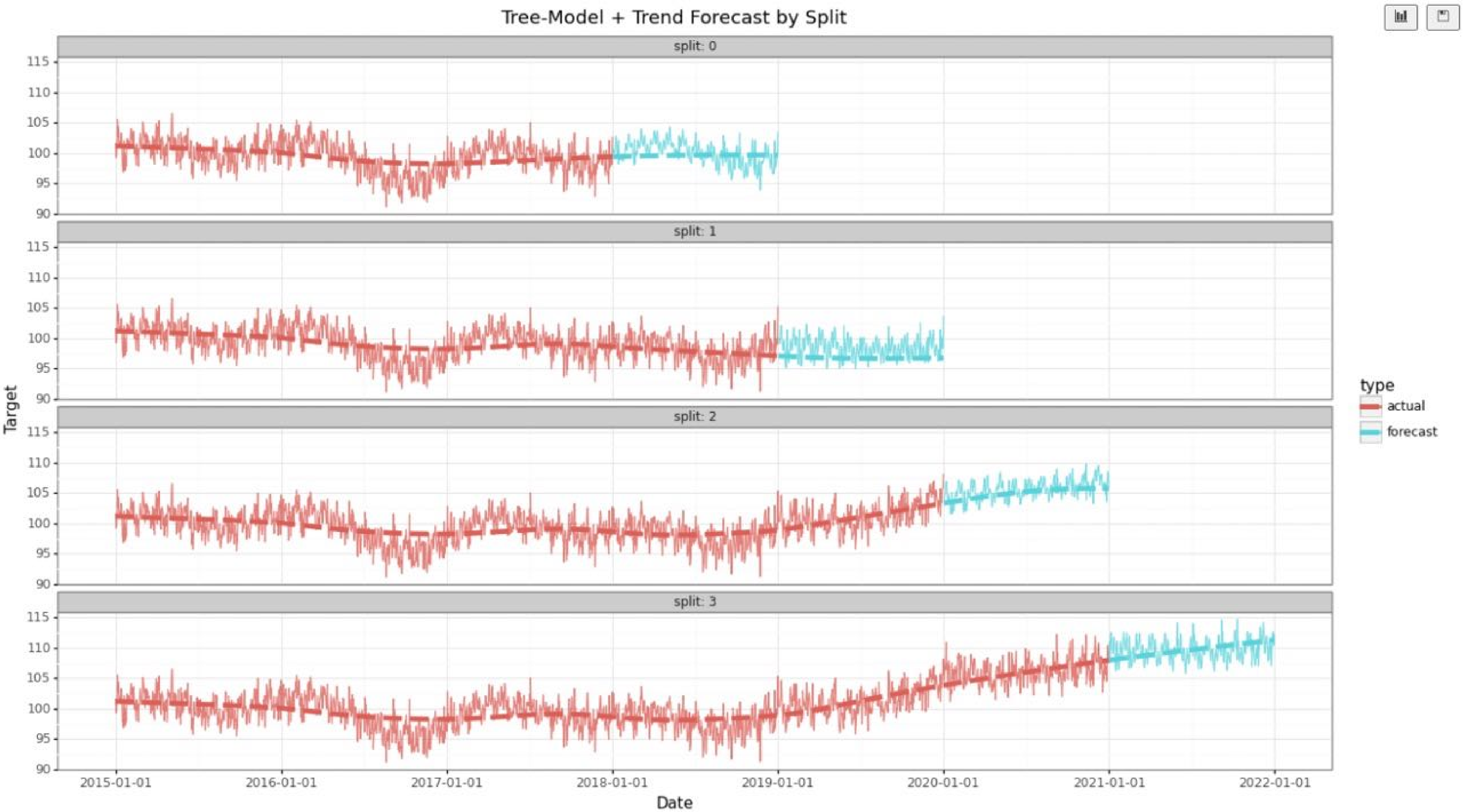


Trend Fit and Forecast by Split with Trend = 725 and Lags = 61
split: 0
split: 1
split: 2
split: 3
110
105
100
95
Target
Date
2015-01-01
2016-01-01
2017-01-01
2018-01-01
2019-01-01
2020-01-01
2021-01-01
2022-01-01
type
actual
fitted
forecast


De-trended by Split
split: 0
split: 1
split: 2
split: 3
5
0
−5
De-trended Target
2015-01-01
2016-01-01
2017-01-01
2018-01-01
2019-01-01
2020-01-01
2021-01-01
Date

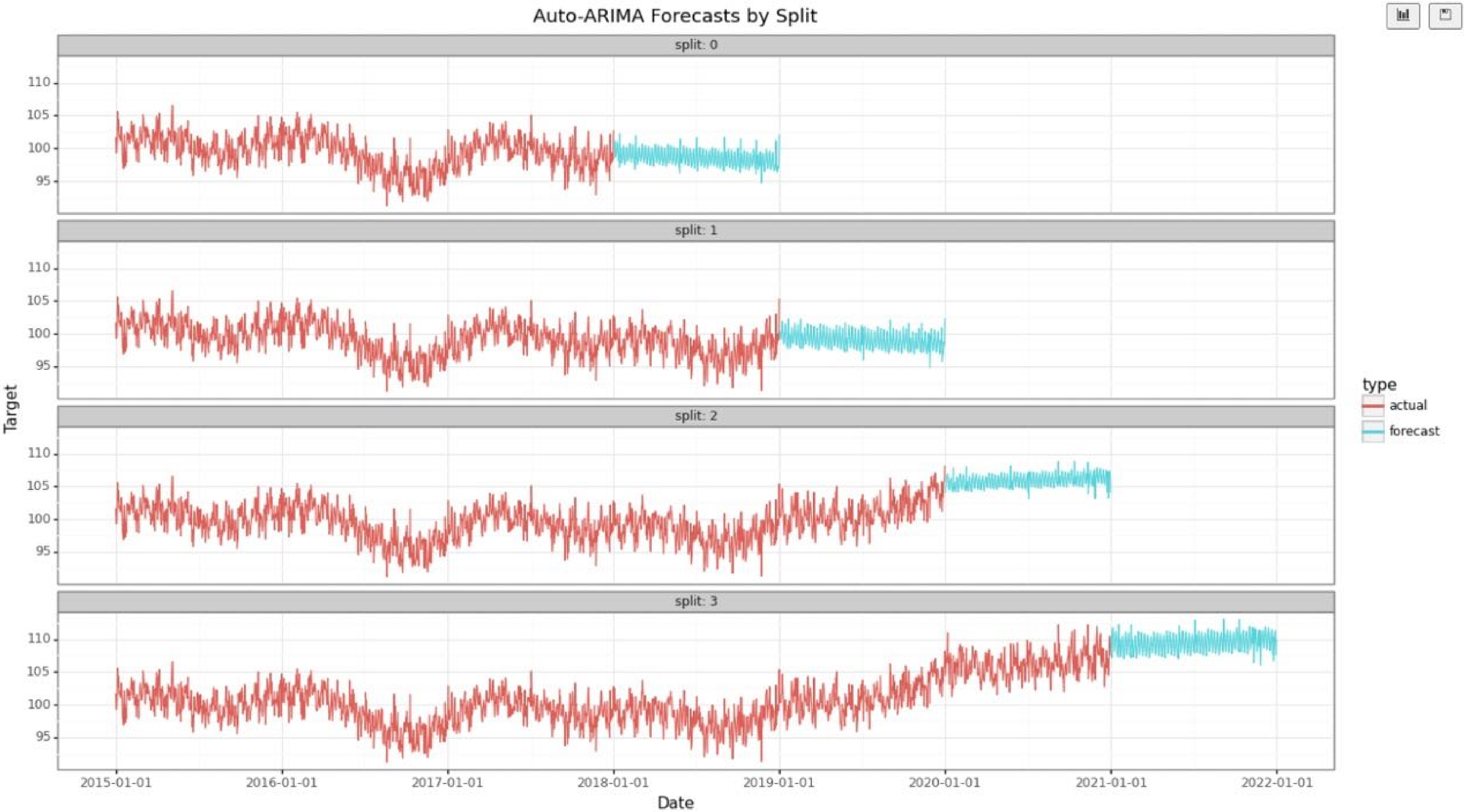
Model 2: Tree-Based Ensemble

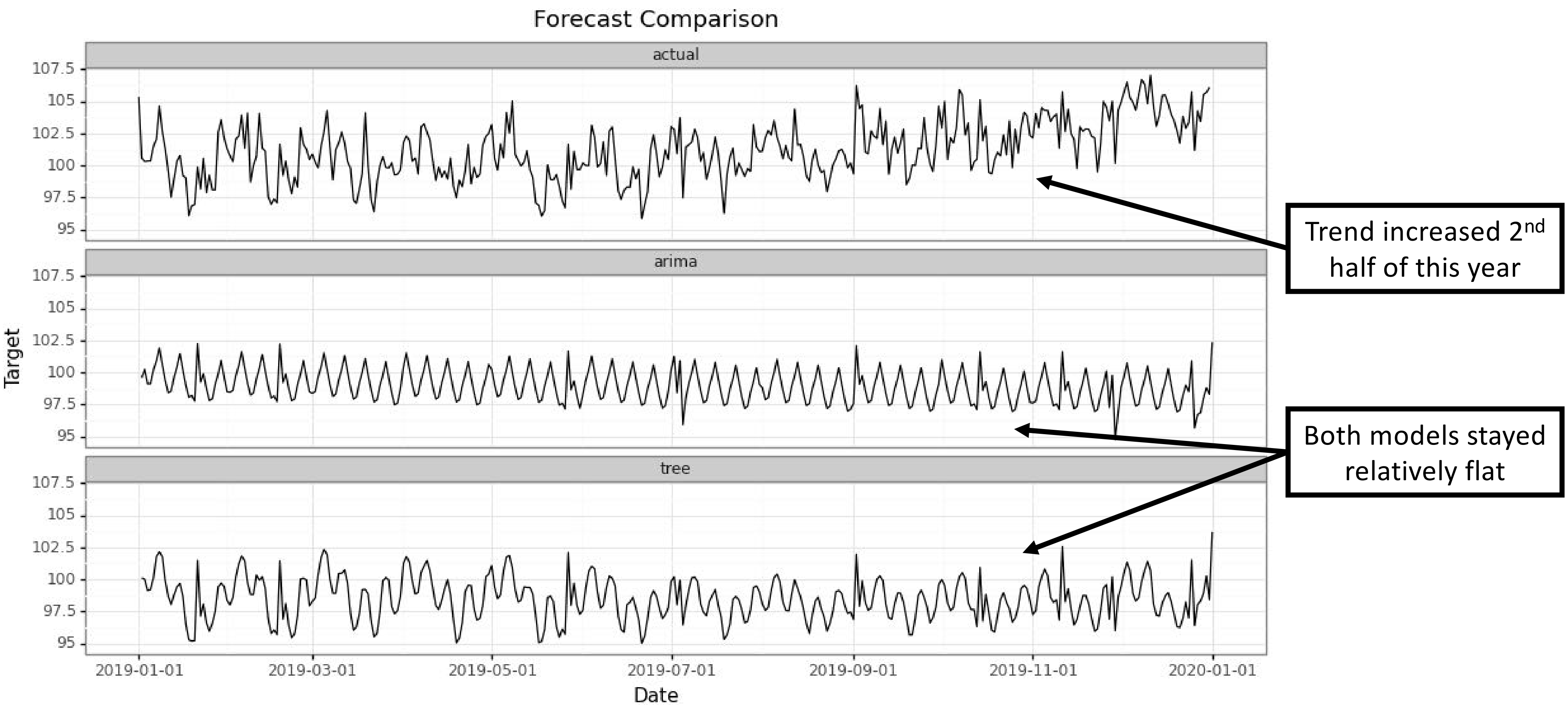
- Use tree-based ensemble to model de-trended time series
  - De-trending ensures the tree-based model can safely predict within the range of training examples.
- Features are calendar-based:
  - Day-of-week
  - Day-of-month
  - Day-of-year
  - Holidays (adjacent)
  - Month-of-year
  - Etc.
- HistGradientBoostingRegressor() parameters
  - Max iterations = 300
  - Max tree depth = 30
  - Learning rate = 0.1
  - L2 regularization = 0.1
- Forecast = trend forecast + tree-based prediction


Tree-Model + Trend Forecast by Split
split: 0
split: 1
split: 2
split: 3
Target
Date
115
110
105
100
95
90
2015-01-01
2016-01-01
2017-01-01
2018-01-01
2019-01-01
2020-01-01
2021-01-01
2022-01-01
type
actual
forecast

Auto-ARIMA Procedure

- Compare trend + tree-based model to auto-ARIMA procedure
- Implemented in Python via the pmdarima library
  - based on auto.arima() function from R forecast package
- SARIMAX = **S**easonal **A**uto-**R**egressive **I**ntegrated **M**oving-Average with e**X**ogenous
  - ARIMA, plus:
  - Enable seasonality parameter
  - Include exogenous regressors (calendar features)
- Parameters
  - D = 1 (difference at lag-1)
  - Period = 7 (daily frequency)
  - Seasonal = True
  - Stepwise = True
  - Exogenous = matrix of external regressors (calendar features)
  - Max iterations = 10 (default = 50)
- Note: For real-world data sets, try adding Fourier transforms instead of calendar features as exogenous regressors


Auto-ARIMA Forecasts by Split
split: 0
split: 1
split: 2
split: 3
110
105
100
95
Target
Date
2015-01-01
2016-01-01
2017-01-01
2018-01-01
2019-01-01
2020-01-01
2021-01-01
2022-01-01
type
actual
forecast


Forecast Comparison
actual
arima
tree
107.5
105
102.5
100
97.5
95
Target
2019-01-01
2019-03-01
2019-05-01
2019-07-01
2019-09-01
2019-11-01
2020-01-01
Date
Trend increased 2nd half of this year
Both models stayed relatively flat

Comparison Between Models on Simulated Data Set

- STL/AutoReg Trend + XGBoost
  - Train Time*:
    - Trend: from 0:00:0.49 to 0:00:0.87
    - De-Trending: 0:00:0.01
    - XGBoost: from 0:00:1.26 to 0:00:1.36
    - Total: ~2-3 seconds per split
  - Accuracy:
    - from 1.3% to 3.2%
    - Avg across splits: 2.3%

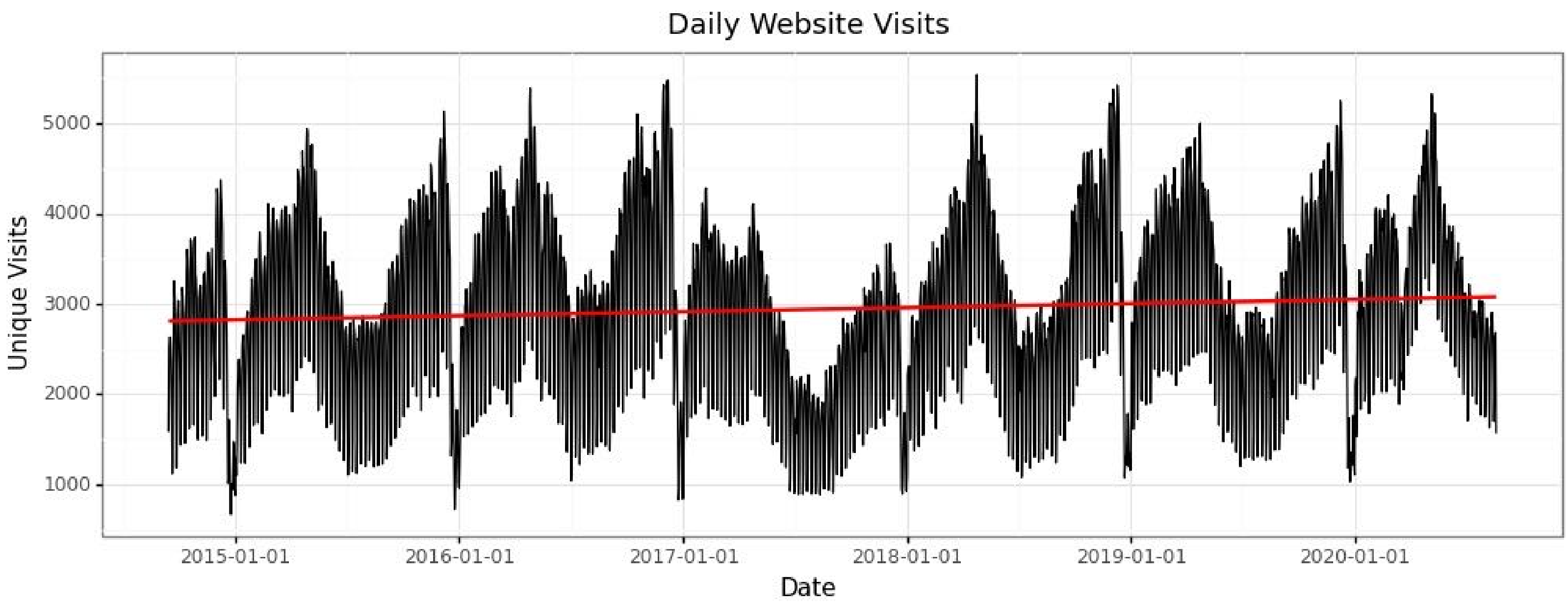
- Auto-ARIMA
  - Train Time*:
    - from 1min 16sec to 3min 28sec
  - Accuracy:
    - from 1.4% to 3.1%
    - Avg across splits: 2.1%

*4 splits with 365 test size each

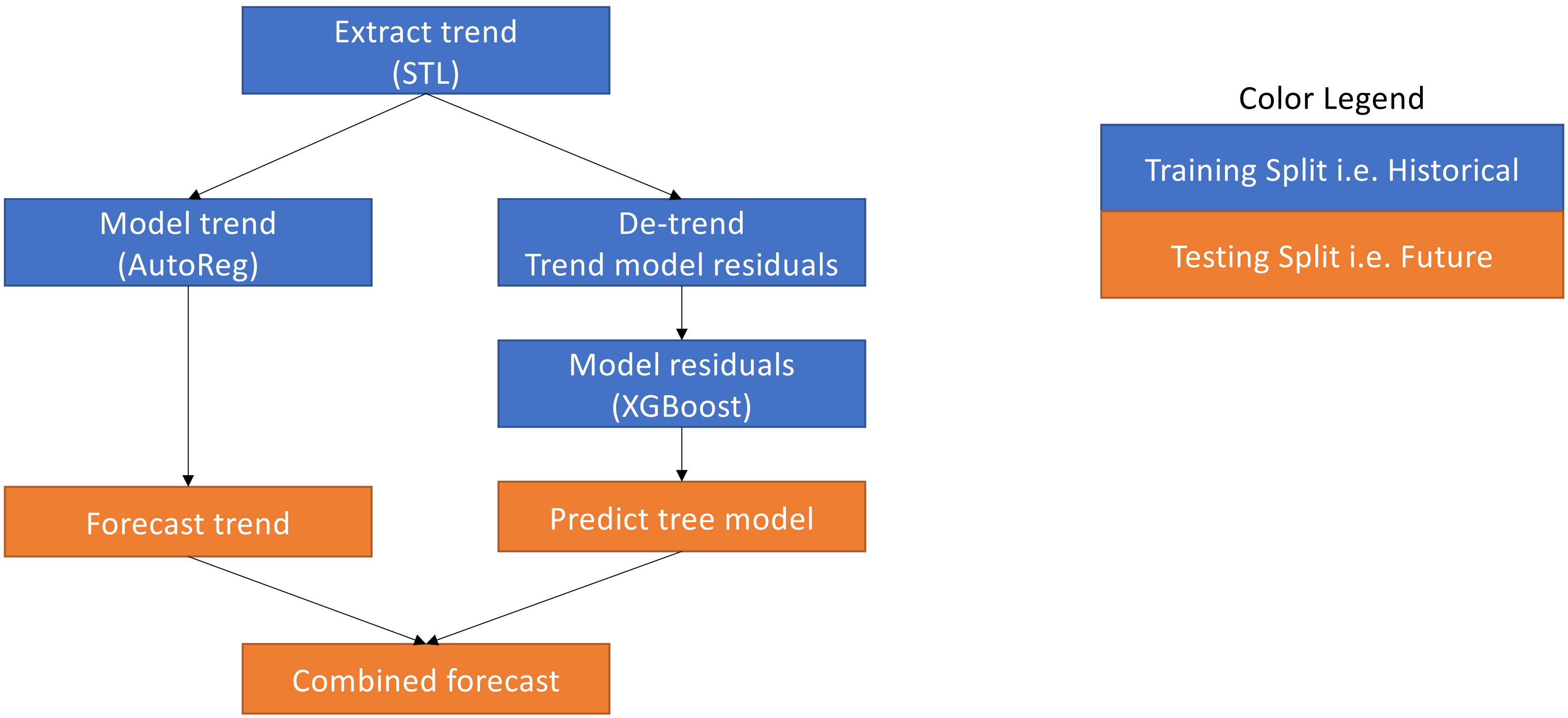
Real-World Data Set

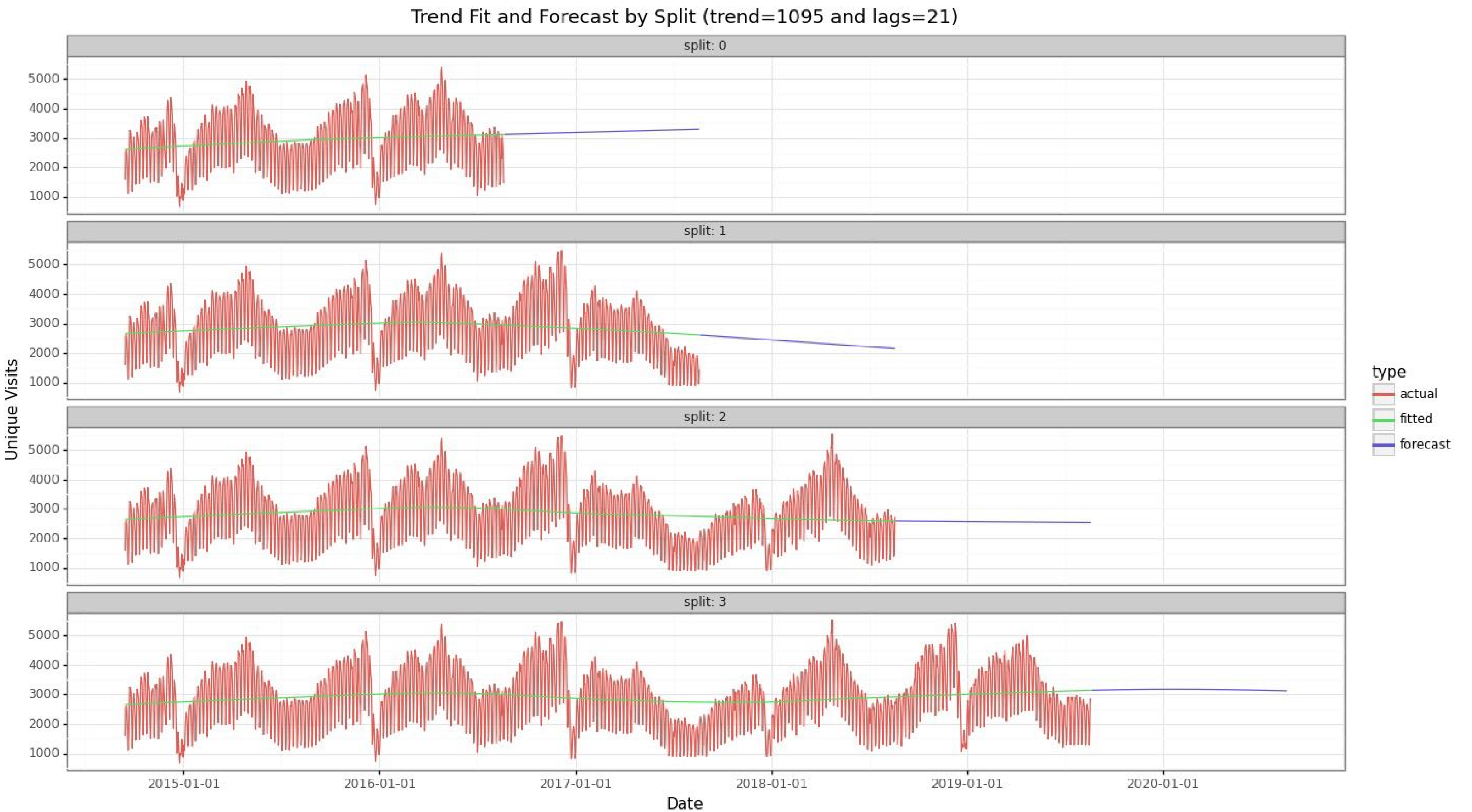
Data Set on Kaggle: Daily website visitors (time series regression) | Kaggle

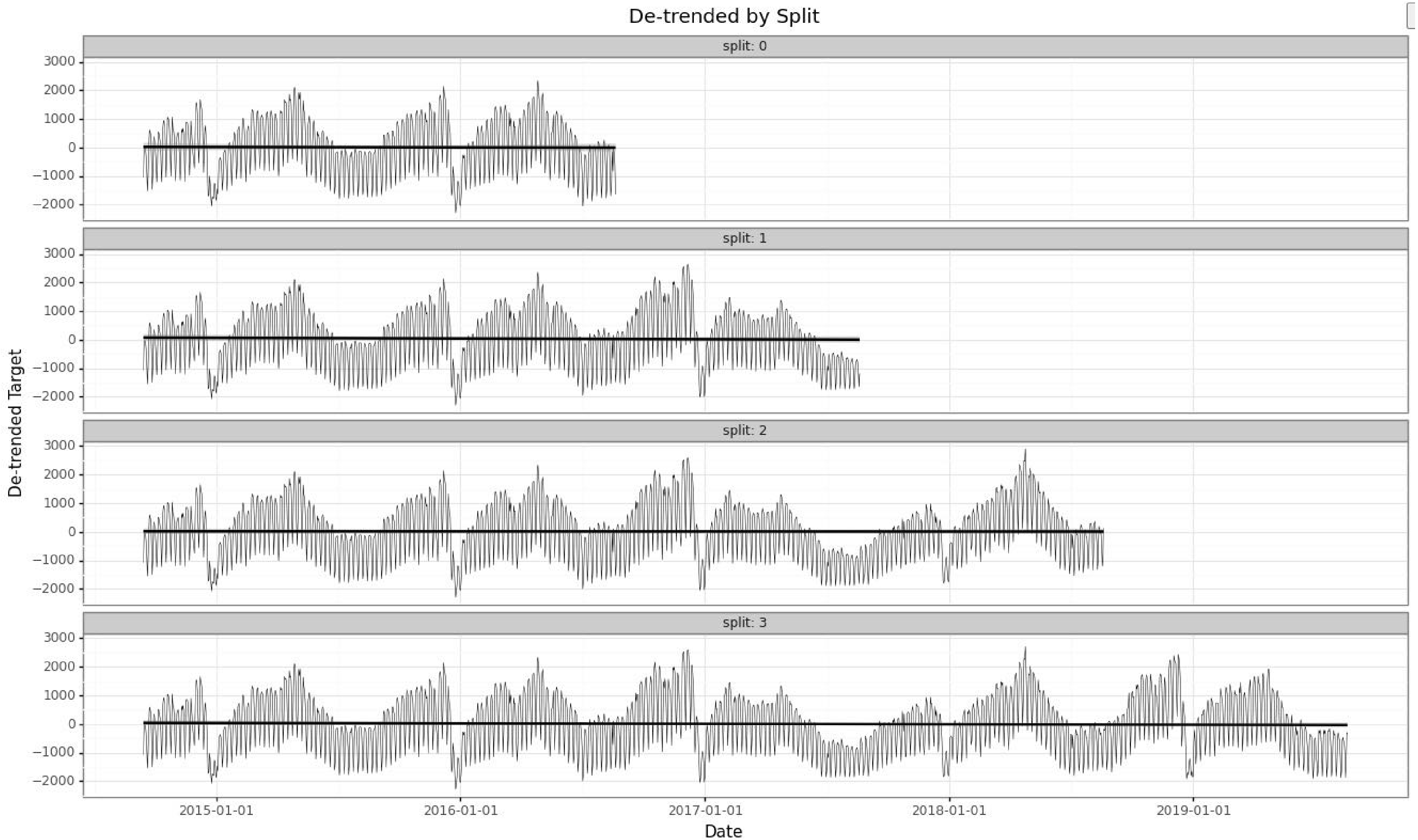
This file contains 5 years of daily time series data for several measures of traffic on a statistical forecasting teaching notes website whose alias is statforecasting.com. The variables have complex seasonality that is keyed to the day of the week and to the academic calendar. The patterns you see here are similar in principle to what you would see in other daily data with day-of-week and time-of-year effects.



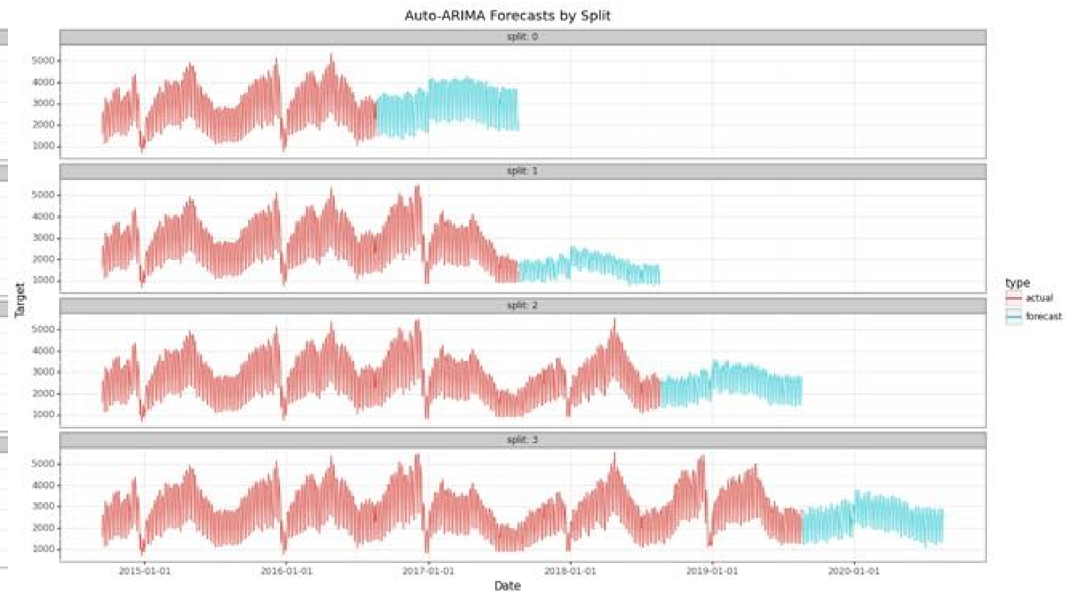
Trend + Tree-Based Modeling Procedure




Trend Fit and Forecast by Split (trend=1095 and lags=21)
split: 0
split: 1
split: 2
split: 3
Unique Visits
1000
2000
3000
4000
5000
Date
2015-01-01
2016-01-01
2017-01-01
2018-01-01
2019-01-01
2020-01-01
type
actual
fitted
forecast


De-trended by Split
split: 0
split: 1
split: 2
split: 3
3000
2000
1000
0
−1000
−2000
De-trended Target
2015-01-01
2016-01-01
2017-01-01
2018-01-01
2019-01-01
Date

Conclusion



Method	Avg Error (across 4 splits)	Avg Train Time (across 4 splits)
Trend + Tree-Model	14.6%	~2sec
Auto-ARIMA (SARIMAX)	21.6%	~2min

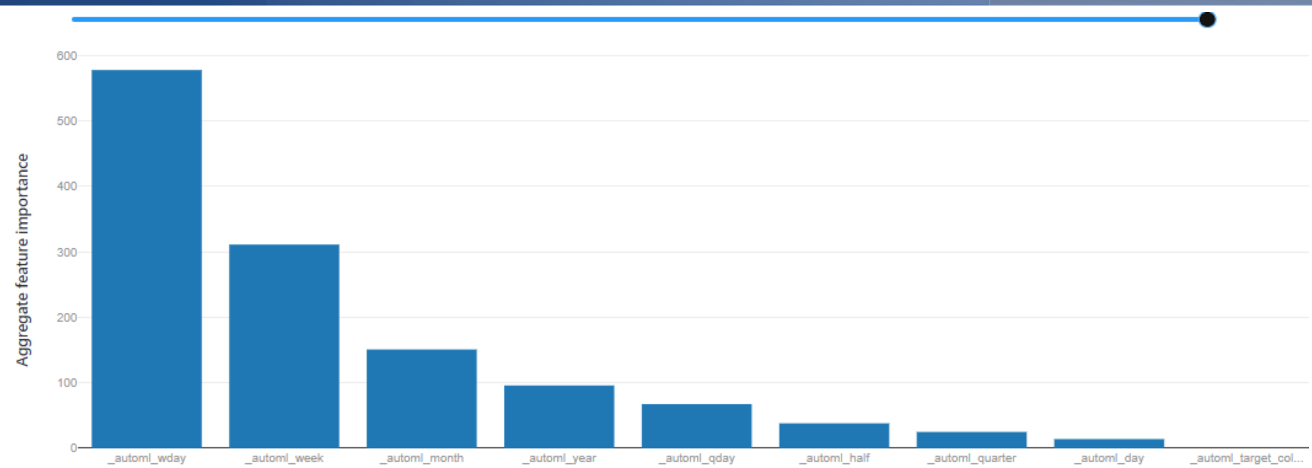
The End

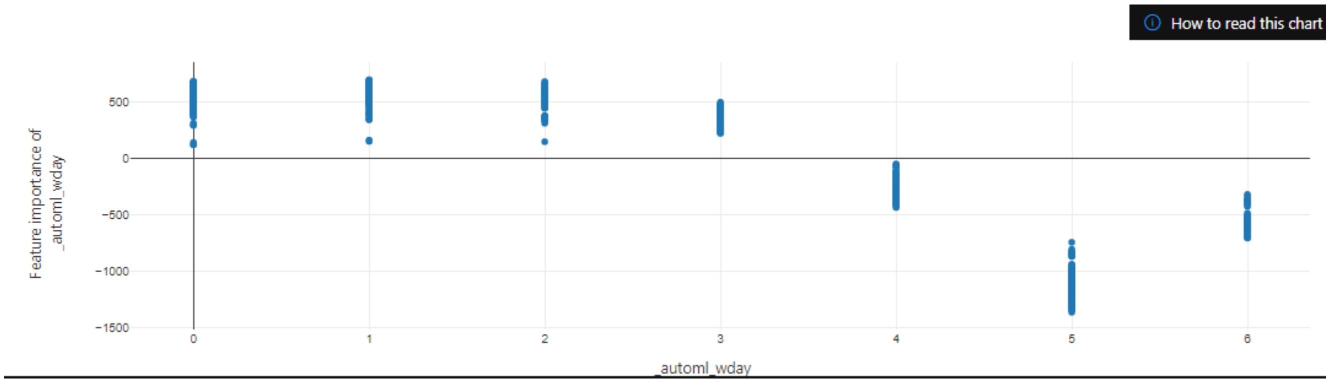
- Some other forecasting tools worth checking out:
  - Azure AutoML
    - Python SDK: [Set up AutoML for time-series forecasting - Azure Machine Learning | Microsoft Docs]
    - No Code: [Tutorial: Demand forecasting & AutoML - Azure Machine Learning | Microsoft Docs]
  - Modeltime (Business-Science)
    - R: https://business-science.github.io/modeltime/
  - Prophet
    - Github: https://github.com/facebook/prophet
    - Python: https://pypi.org/project/prophet/
    - R: https://cran.r-project.org/web/packages/prophet/index.html
  - Neural Prophet
    - https://pytorch.org/
    - Much faster than Prophet, uses PyTorch
    - Uses AR-Net for modeling autocorrelation
- Python notebooks and PPT will be available on Lev's repo

Bonus: Azure AutoML Experiment: Daily Website Data Set

Algorithm name	Explained	Nor... ↑	Sampling	Submitted time	Duration	Hyperparameter
MaxAbsScaler, ExtremeRandomTrees	View explanation	0.09609	100.00 %	Apr 15, 2022 2:22 PM	4s	bootstrap; criterion : mse; max_features : 0.8; min_samples_leaf : 0.00236468227726
MinMaxScaler, RandomForest		0.09786	100.00 %	Apr 15, 2022 11:42 AM	4s	bootstrap; criterion : mse; max_features : 0.7; min_samples_leaf : 0.00419663374756
MinMaxScaler, RandomForest		0.09786	100.00 %	Apr 15, 2022 11:42 AM	4s	bootstrap : true; criterion : mse; max_features : sqrt; min_samples_leaf : 0.00346623
MinMaxScaler, ExtremeRandomTrees		0.09980	100.00 %	Apr 15, 2022 11:42 AM	4s	bootstrap : true; criterion : mse; max_features : 0.7; min_samples_leaf : 0.00346623
StandardScalerWrapper, LightGBM		0.10009	100.00 %	Apr 15, 2022 11:42 AM	4s	min_data_in_leaf : 20
MinMaxScaler, ExtremeRandomTrees		0.10034	100.00 %	Apr 15, 2022 2:18 PM	4s	bootstrap; criterion : mse; max_features : 0.5; min_samples_leaf : 0.00508093718889
TCNForecaster	Not supported	0.10238	100.00 %	Apr 15, 2022 3:17 PM	7m 13s	
StandardScalerWrapper, XGBoostRegressor		0.10365	100.00 %	Apr 15, 2022 11:42 AM	4s	tree_method : auto
TCNForecaster	Not supported	0.10670	100.00 %	Apr 15, 2022 11:42 AM	17m 24s	
TCNForecaster	Not supported	0.10712	100.00 %	Apr 15, 2022 11:42 AM	1h 2m 43s	
ProphetModel		0.10724	100.00 %	Apr 15, 2022 11:42 AM	4s	
TCNForecaster	Not supported	0.11375	100.00 %	Apr 15, 2022 11:42 AM	15m 20s	
RobustScaler, ExtremeRandomTrees		0.11486	100.00 %	Apr 15, 2022 2:26 PM	4s	bootstrap : true; criterion : mse; max_features : 0.9; min_samples_leaf : 0.00901720

AutoML Explanations:
MaxAbsScaler, ExtremeRandomTrees





	MaxAbsScaler, ExtremeRandomTrees	StandardScalerWrapper, LightGBM	ProphetModel
Explained variance	0.75536	0.70605	0.74682
Mean absolute error	361.21	371.67	409.74
Mean absolute percentage error	12.425	12.407	14.893
Median absolute error	283.53	289.12	327.71
Normalized mean absolute error	0.074109	0.076256	0.084067
Normalized median absolute error	0.058172	0.059318	0.067237
Normalized root mean squared error	0.096089	0.10009	0.10724
Normalized root mean squared log error	0.083922	0.083404	0.11275
R2 score	0.72150	0.69794	0.65320
Root mean squared error	468.34	487.82	522.71
Root mean squared log error	0.17756	0.17647	0.23856
Spearman correlation	0.89632	0.87858	0.89201

Bonus: Comparing Different Ensemble Methods

```
# from sklearn.ensemble:
    GradientBoostingRegressor
    HistGradientBoostingRegressor # (best, inspired by LightGBM)
    RandomForestRegressor
    ExtraTreesRegressor

# from xgboost !pip install xgboost
    XGBRegressor
    XGBRFRegressor
```

All regressors had similar speed and accuracy.
But the best was **HistGradientBoostingRegressor** - the histogram-based gradient boosting.

- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingRegressor.html
- https://machinelearningmastery.com/histogram-based-gradient-boosting-ensembles/
- https://inria.github.io/scikit-learn-mooc/python_scripts/ensemble_hist_gradient_boosting.html

Why the "histogram-based" approach is faster and better?

- The bottleneck in training any boosted trees algorithm is the time for finding the best split between values when building the trees
- This time can be significantly reduced if we pre-process the data for a given tree – aggregate them into "bins"
- So, we convert the continuous float numbers into limited number of discrete integer values
- This "histogram binning" reduces the number of tests needed to find best split
- This makes finding the split much faster - but reduces the accuracy
- To increase accuracy, we can simply increase the number of trees