Published in Towards Data Science

Ryan Clukey Follow

Mar 18, 2021 · 10 min read · Listen

Save

Forecasting with Bayesian Dynamic Generalized Linear Models in Python

A Case Study Comparing Bayesian and Frequentist Approaches to Multivariate Times Series Data

Image by Original Author.

Forecasting is critical for nearly all businesses when planning for revenue goals, inventory management, headcount, and other economic considerations essential for managing a successful business. Highly accurate forecasts are often difficult to achieve, but essential to enable businesses plan for shifts and changes in a dynamic market. Often forecast models can (and should) take advantage of additional inputs or variables that may help explain the variability in the target or dependent series. For example, predicting the number of units sold based on visits to the website, or forecasting monthly revenue based on monthly marketing spend. As is often the case, the available data is often limited or incomplete, which complicates the ability to use standard algorithms to generate accurate forecasts.

In this article I'll introduce the Bayesian approach to multivariate time series and provide a contrast to traditional frequentist methods, like ARIMA. Time series is a special case of regression where the independent variable is a regular interval time measure (i.e. weeks, months, years, etc.), along with potential exogeneous features which may be useful in explaining the residual variation of a sequential series. ARIMA (Auto Regressive, Integrated, Moving Average) has been one of the standard approaches to time series forecasting; it is a powerful algorithm and widely used across many industries. However, there are many complex considerations: the data should be stationary, there may be multiple trends, independent variables are often lagged, there is seasonality — sometimes multiple seasons in the same data, there must be a considerable amount of available data, etc. For these reasons, time series is a difficult statistical technique to master, and generating accurate forecasts is quite challenging.

The Bayesian approach offers a probabilistic approach to time series to reduce uncertainty and incorporate "prior" information. These models are referred to as Dynamic Linear Models or Structural Time Series (state space models). They work by fitting the structural changes in a time series dynamically — in other words, evolving and updating the model parameters over time with the addition of new information. In contrast, ARIMA estimates parameters for the series, which remain fixed, then uses Maximum Likelihood estimation for determining the time series predictions. Bayesian methods use MCMC (Monte Carlo Markov Chains) to generate estimates from distributions. For this case study I'll be using Pybats — a Bayesian Forecasting package for Python. For those who are interested, and in-depth article on the statistical mechanics of Bayesian methods for time series can be found here.

Case Study

In this case study we evaluate the effect of two independent time series (covariates) on our dependent variable: total number of monthly vehicle purchases for a particular auto manufacturer. Since this is count data, we are looking at a Poisson process, which assumes a different underlying distribution than that of Gaussian. Poisson models are based on counts, and therefore the lowest possible value is 0. The two covariates include: spend on marketing, aggregated at the monthly level and a measure of consumer sentiment for the given brand, normalized to a scale of 0–100.

Data

Our data consists of 48 observations between January 2017 and December 2020. There are no missing values. The data represent monthly counts of vehicle sales, average marketing spend (in tens of thousands) and average consumer sentiment — a proprietary metric. Since we are using Python, I'll provide some code snippets through to demonstrate the analysis steps.

```
# Import the necessary libraries
import pandas as pd
import numpy as np
from scipy import stats
import pmdarima as pmd
import matplotlib.pyplot as plt
import pybats
from pybats.loss_functions import MAPE
from pybats.analysis import analysis
from pybats.point_forecast import median

# import the data
dfv = pd.read_csv("vechicle_data.csv")
```

```
dfv.head()
```

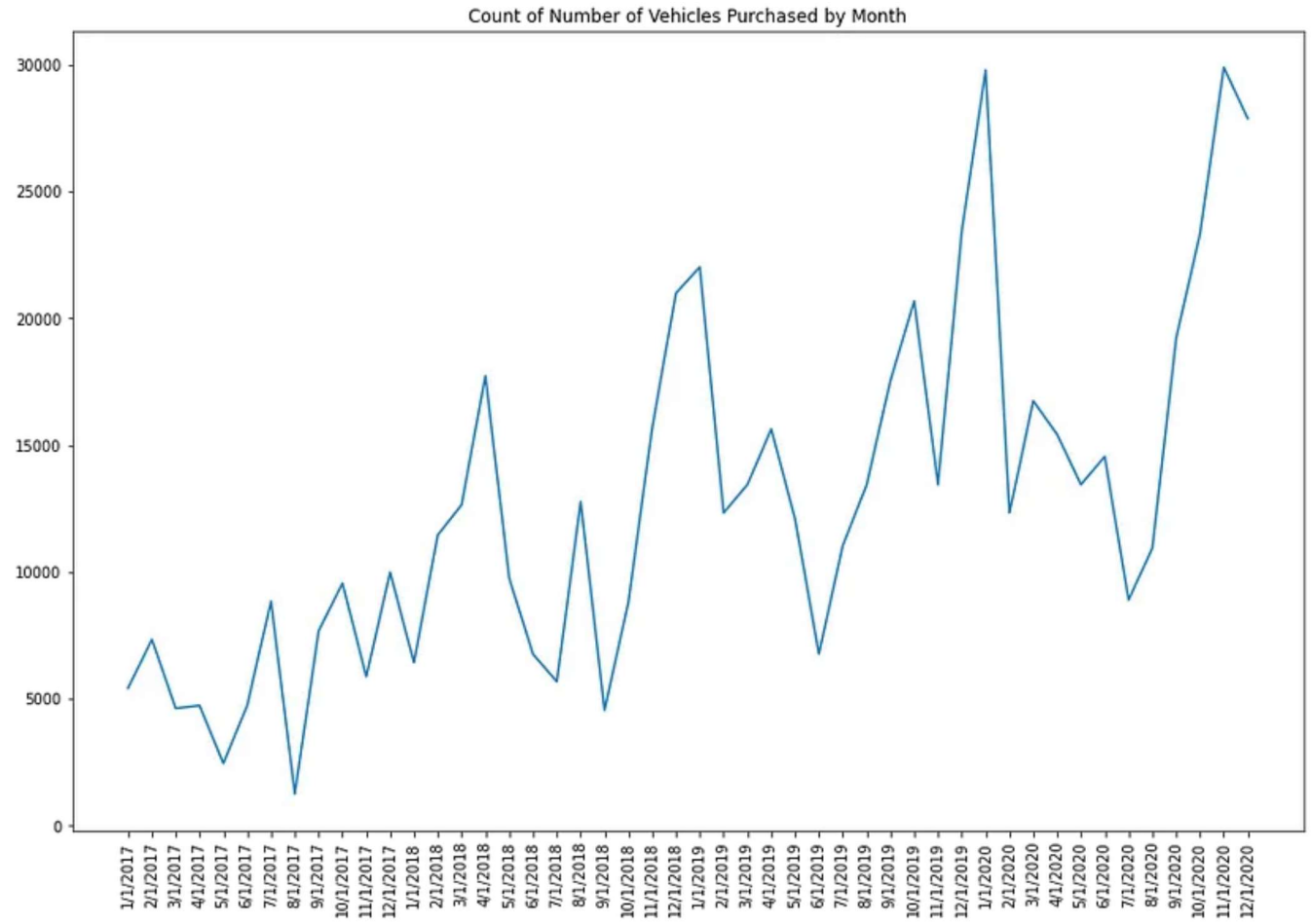
	Date	Transactions	Marketing_Spend	Consumer_Sentiment
0	1/1/2017	5434	30	32
1	2/1/2017	7345	100	34
2	3/1/2017	4634	85	38
3	4/1/2017	4743	86	32
4	5/1/2017	2466	50	29

Our dataset above shows the three variables we will be using for this analysis: Transactions is the dependent variable, with Marketing Spend and Consumer Sentiment as two Independent variables (or covariates) in the Dynamic Regression model. What we're essentially saying here is: do Marketing Spend and Consumer Sentiment have a time-varying effect on the number of Transactions; can we model their effects and use the knowledge we have of these variables to generate reasonable predictions of Transactions beyond our current dataset?

Let's take a look at the 3 separate time series we have so far:

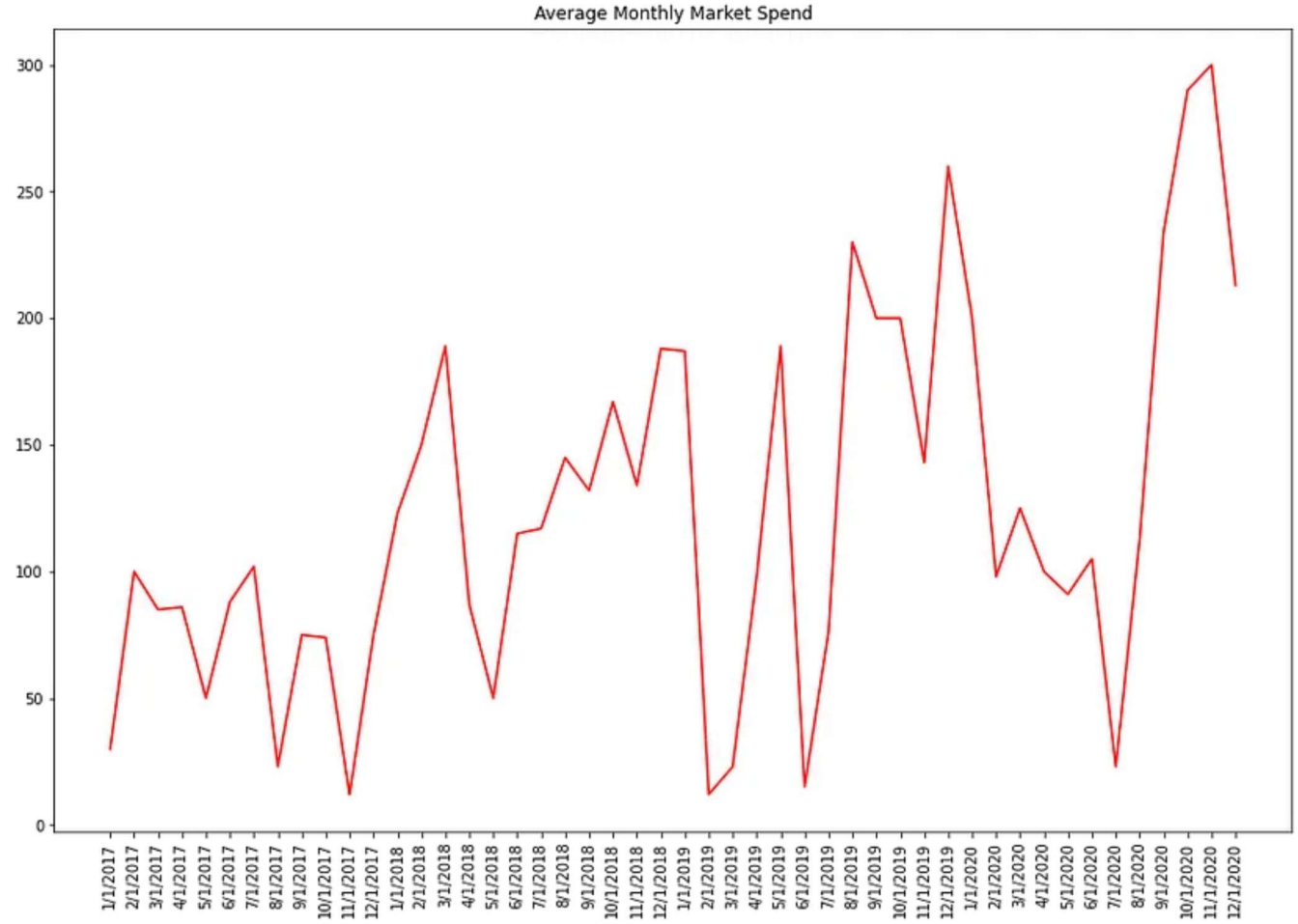
DV: Number of Vehicles Sold

Simply visual examination reveals a couple of important things happening in this series: 1) the data is not stationary, 2) there appears to be a possible seasonal effect, 3) the variance increases with time. For ARIMA, these would be problematic, and our series would be subject to multiple "adjustments" such as differencing and seasonal decomposition in order to achieve the requirement of stationarity.



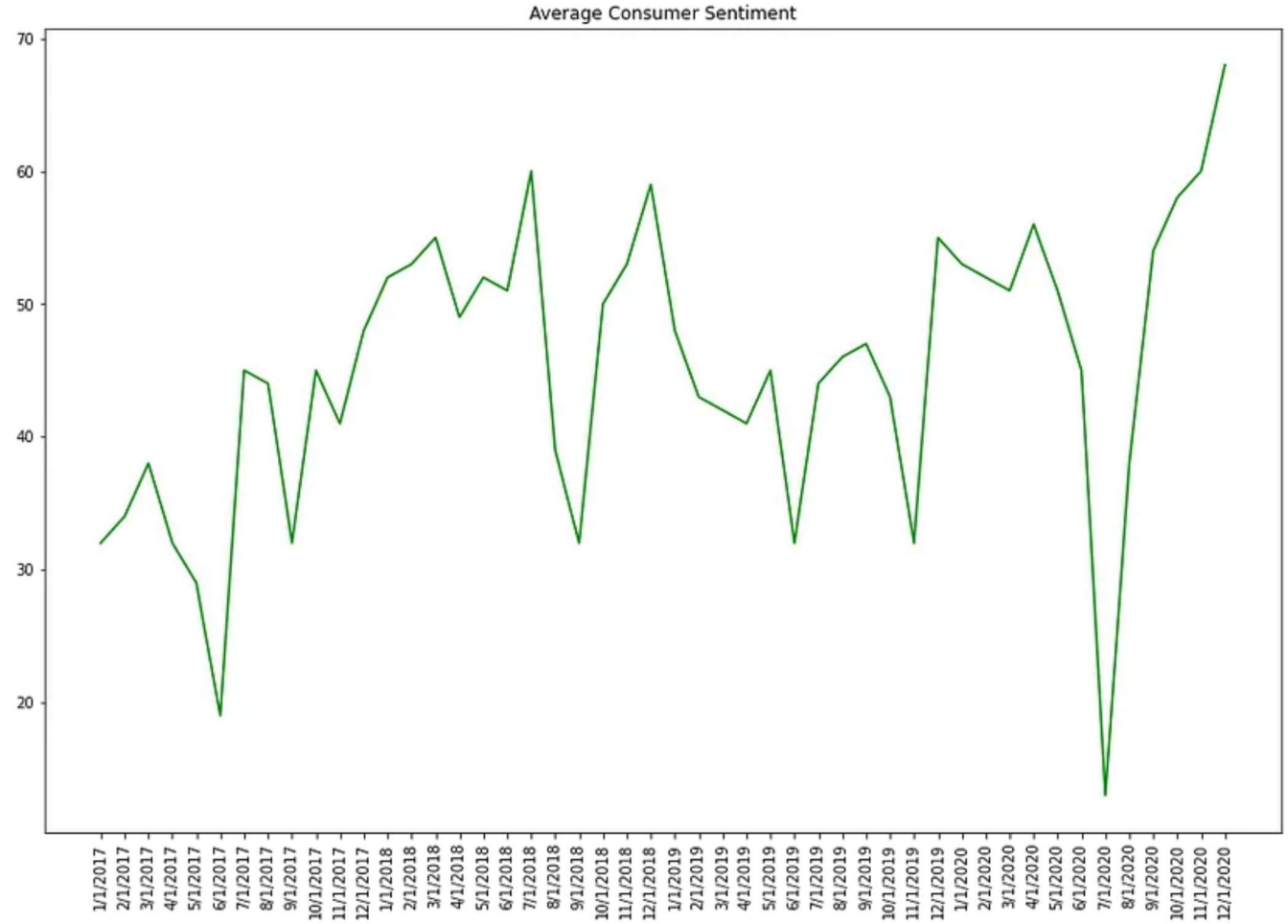
IV 1: Marketing Spend

The plot below shows the variability of Average Marketing Spend over the same period of time. This series is less clear. There might be seasonality, there might be shifts in trend; the series does not appear stationary. However, as mentioned, such considerations are irrelevant for Bayesian approaches to time series.



IV 2: Consumer Sentiment

Our metric of Consumer Sentiment follows closely with that of Marketing Spend, which we might expect given that marketing efforts should have an effect on consumer sentiment.



Model Construction

To build the model we will be build a function in Python to make things a little easier. There are a number of parameters to adjust in the model itself, including learning and decay rates, seasonality, how long the prior period should be (to learn the prior variance), etc. I'll not go into those here, as descriptions and guidelines are provided by the Pybats tutorial.

```
def bayes_forecast(iv,dv):
    '''
    This functions runs the Pybats algorithm by taking two
parameters: an independent variable matrix and a dependent variable.
    Both elements must be sequential time series.
    '''
    # first check if the iv = None, indicating this would be a
univariate series
    if iv is None:
        x = None
    else:
        x = iv.values

    y = dv.values

    # set the one-step-ahead value; by default we want 1
```

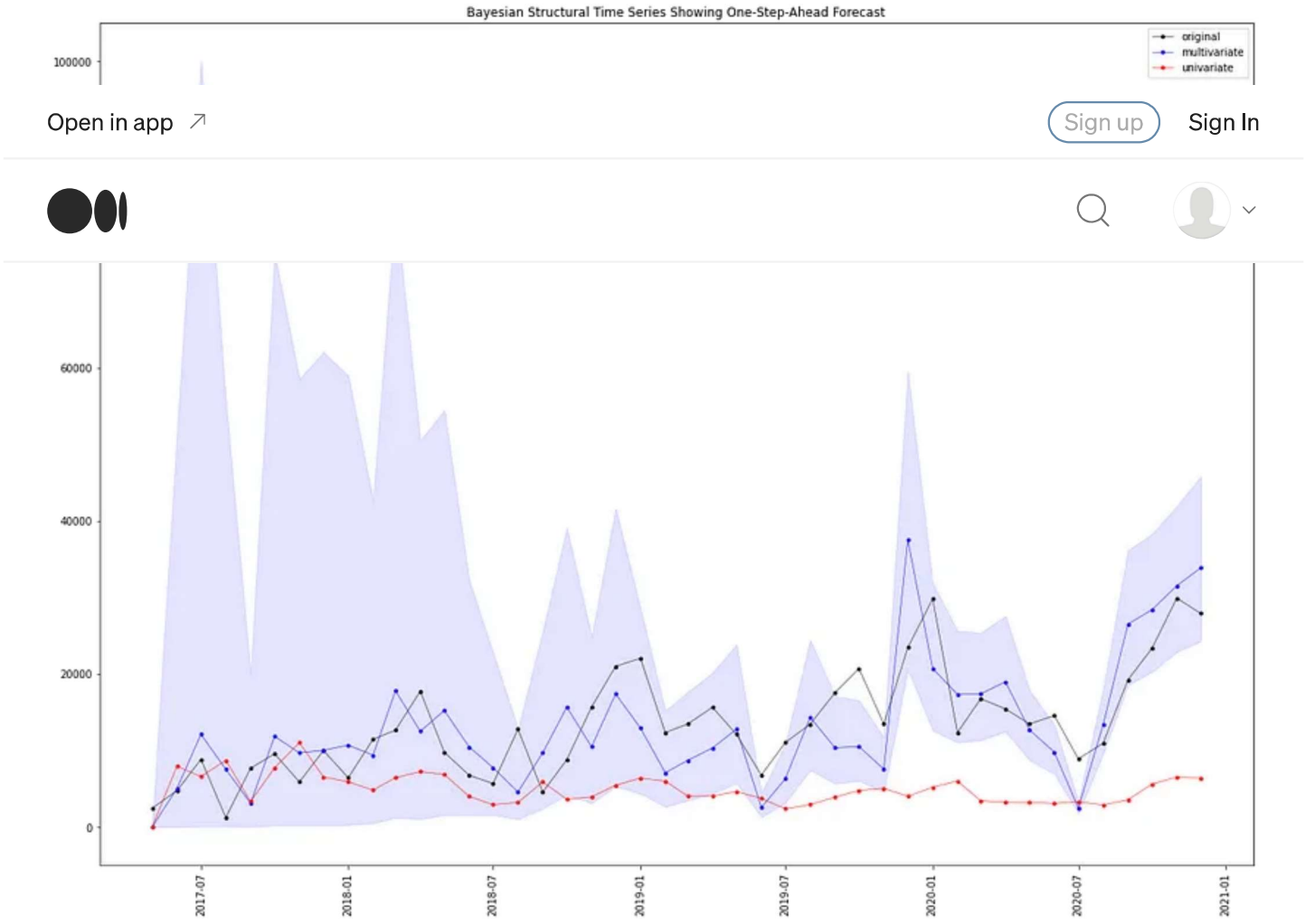
```
    k = 1
    forecast_start = 0
    forecast_end = len(y)-1

    mod, samples = analysis(Y=y, X=x, family='poisson',
       forecast_start=forecast_start,
       forecast_end=forecast_end,
       k=k,
       ntrend=1,
       nsamps=5000,
       seasPeriods=[12],
       seasHarmComponents=[[1,2]],
       prior_length=4,
       deltrend=0.94,
       delregn=0.90,
       delVar=0.98,
       delSeas=0.98,
       rho=.6,
       )

    forecast = median(samples)

    # set confidence interval for in-sample forecast
    credible_interval=95
    alpha = (100-credible_interval)/2
    upper=np.percentile(samples, [100-alpha], axis=0).reshape(-1)
    lower=np.percentile(samples, [alpha], axis=0).reshape(-1)
    print("MAPE:", MAPE(y[-18:], forecast[-18:]).round(2))

    #Generate the Bayesian Future Forecast
    return mod, forecast, samples, y

mv_mod, mv_for, mv_samp, mv_y = bayes_forecast(dfv.iloc[:,2:4],
dfv.Transactions)
```

We're going to use our Python function above to also run the equivalent model without the covariates — so a standard univariate time series. We're doing this to determine if including the independent variables in the model is having the intended effect of reducing the overall residual error in the model. In other words, does including the variables improve our understanding of the data; is it worth including them in the model?

```
# Calculate Bayesian estimate for the univariate model
uv_mod, uv_for, uv_samp, uv_y = bayes_forecast(None,
dfv.Transactions)
```

Results

The plot below shows the results of our analysis. There are a number of observations we have. First, the model had a difficult time learning the beginning structure of the series, as evidenced by the wide credible intervals at the beginning of the series. Eventually the model parameters "learned" and the one-step ahead predictions for the multivariate model begin to more closely and follow the original series. This what we expected. However, note that the Univariate series did quite a poor job at capturing the movement and values of the original series. While it appears to learn the trend, its predictions are substantially and consistently off from the original dependent values, especially later in the series. This tells us something about the efficacy of the additional covariates in the model — we'll come to that next.



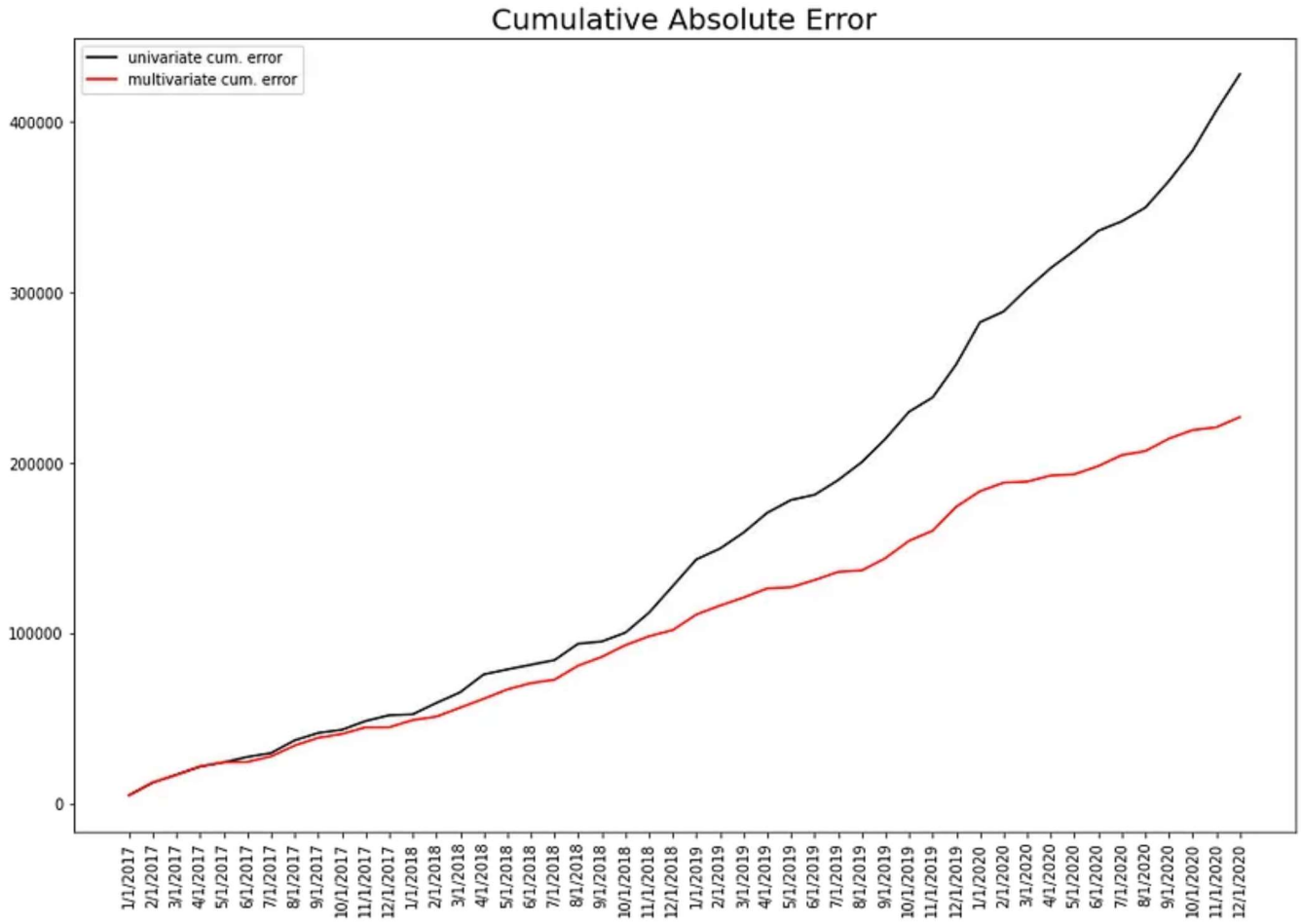
One-Step-Ahead Predictions

The plot above shows the output from the Bayesian forecast. It shows one-step-ahead predictions with a 95% Credible Interval. This differs from ARIMA, which produces in-sample predictions.

An important difference between in-sample ARIMA predictions and those made with Pybats: In ARIMA, the in-sample predictions are practically quite useless. They reflect how well the estimated parameters fit the data, and to this end you can very easily over-fit the data simply by over parameterizing your model; it tells us nothing about how well the parameters perform on unseen data (not to mention the parameters are fixed). with Pybats, it's not really an "in-sample" prediction. It's a one-step-ahead prediction starting from the beginning of the series and updating each parameter as you move through the series. Therefore, each step-ahead is actually a true "out-of-sample prediction", and thus there error from the prediction reflects true out-of-sample estimates based on the posterior probabilities. For ARIMA to do this, you would need to specify a hold-out sample at the end of your series, then implement a for-loop that iterates of each data point in the hold-out sample, update the model, move to the next point, update the model, etc.

Bayesian: Comparing Univariate

Did our multivariate Bayesian model perform better than the univariate model? We can observe this difference more readily by comparing the cumulative absolute error in the multivariate and univariate predictions. The plot below compares the cumulative error between the two models (univariate and multivariate). It shows that the covariates begin to explain more of the variability in the time series model as time progresses — which is what we would expect as the model learns and adjusts the parameters based on previous values.



graph comparing cumulative error between univariate and multivariate Bayesian Time Series models

Additionally we can examine the MAPE (mean absolute percentage error) for each model, and determine which model is more accurate in the one-step-ahead forecasts. In this case we examined the predictions for the previous 12 months and compared the MAPE. For the multivariate model we achieved a MAPE of ~20%, while the univariate model achieved a MAPE of 54%. 20% leaves a lot of room for improvement, but it's certainly much better than 54! The MAD (mean absolute deviation) for the multivariate model is ~3,300, which means that our vehicle transaction estimates are estimated to be off by about 3,300 units each month.

ARIMA

Let's take a look at how ARIMA does with our data. First, we'll need to create a train/test split in the data; here we'll use the last 12 months of the series as a holdout sample.

```
dfv_train = dfv[:-12]
dfv_test = dfv[-12:]
dfv_test
```

We're using pmdarima, which is a convenient package that has built a wrapper around the Statsmodels implementation of SARIMAX.

```
mod = pmd.auto_arima(y=dfv_train.Transactions,
exogenous=dfv_train.iloc[:,2:4], stepwise=True, m=12, seasonal=True)
y_pred = mod.predict_in_sample(exogenous=dfv_train.iloc[:,2:4],
return_conf_int=True, alpha=0.05)
a_low = y_pred[1][:,0]
a_high = y_pred[1][:,1]
mod.summary()
```

Our output from the SARIMX (because it's seasonal, and there are two exogenous covariates) is shown below. There's lots of complexity here. First, the model is differenced, we observe a moving average component in both the main series, and in the seasonal component, the seasonal effect is also differenced. Clearly the data is not stationary. Neither of our covariates were found to be statistically significant in the model; according to ARIMA they are having no effect. This is a different outcome than what the Bayesian model observed.

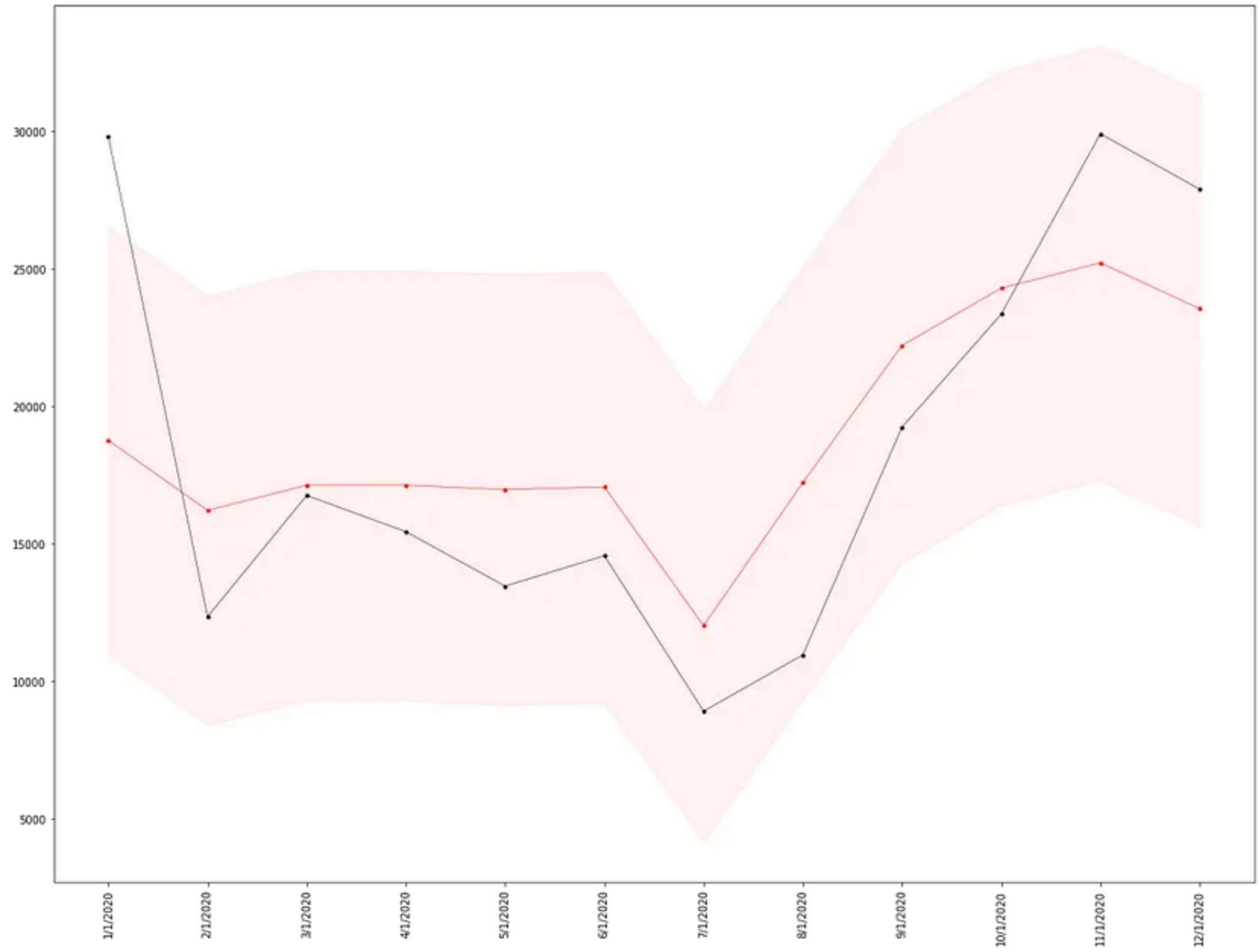
SARIMAX Results

Dep. Variable:	y	No. Observations:	36
Model:	SARIMAX(0, 1, 1)x(0, 0, 1, 12)	Log Likelihood	-337.432
Date:	Wed, 17 Mar 2021	AIC	686.865
Time:	09:10:52	BIC	696.197
Sample:	0	HQIC	690.086
	- 36		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	262.0802	107.630	2.435	0.015	51.128	473.032
Marketing_Spend	28.4370	16.519	1.721	0.085	-3.939	60.813
Consumer_Sentiment	91.6166	80.039	1.145	0.252	-65.257	248.490
ma.L1	-0.9459	0.198	-4.767	0.000	-1.335	-0.557
ma.S.L12	-0.0419	0.299	-0.140	0.889	-0.629	0.545
sigma2	1.576e+07	0.000	6.45e+10	0.000	1.58e+07	1.58e+07

Ljung-Box (L1) (Q):	2.99	Jarque-Bera (JB):	0.62
Prob(Q):	0.08	Prob(JB):	0.73
Heteroskedasticity (H):	2.50	Skew:	0.23
Prob(H) (two-sided):	0.13	Kurtosis:	2.53

How well did the ARIMA model perform in the hold-out sample of 12 months? Not too bad. we observed a MAPE of 22% The Bayesian model had a MAPE of 20% for the same 12-month time period.

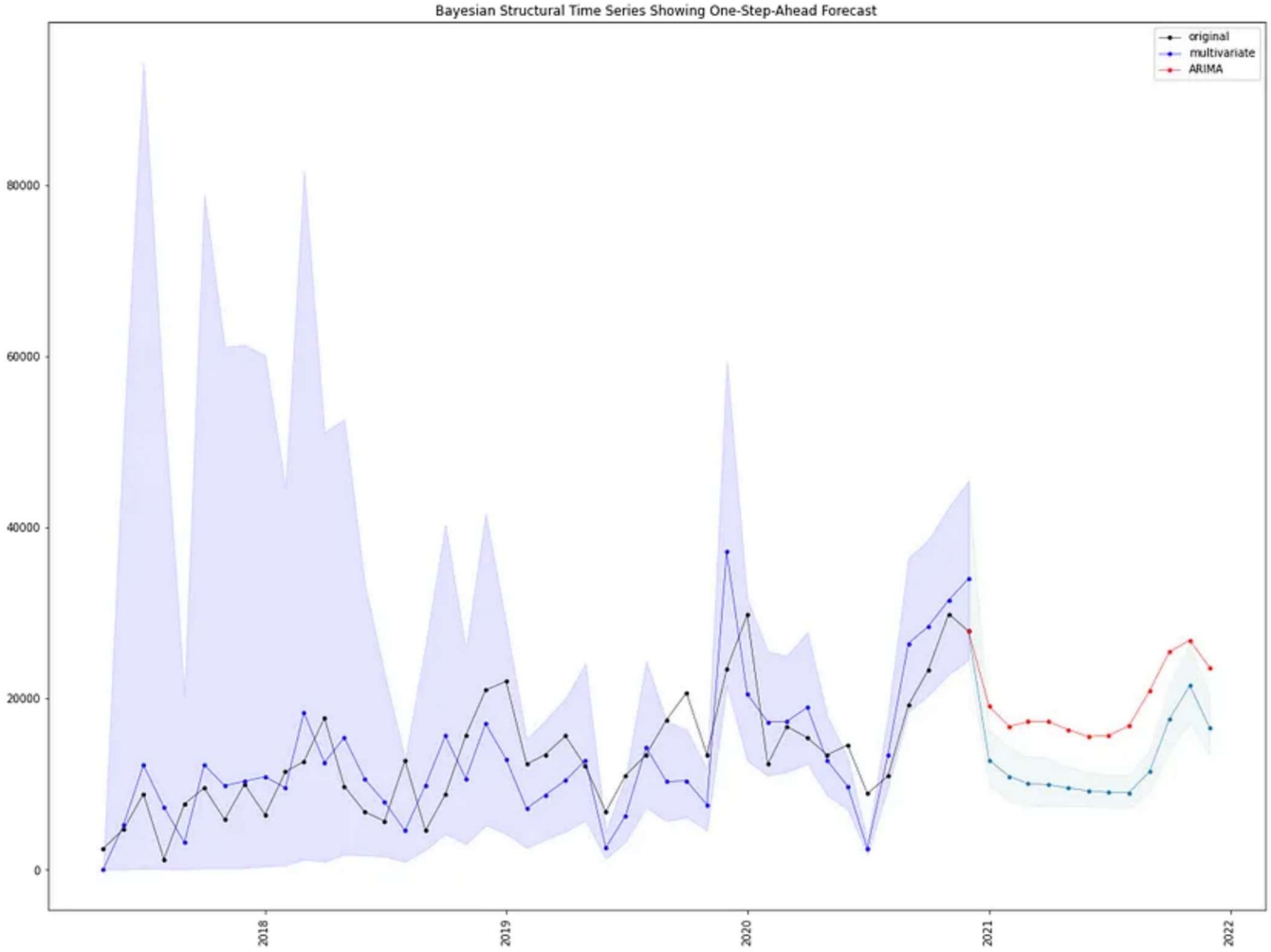


Future Predictions

The next step is to predict the future using the model we have created. Because we have introduced additional covariates, we will also need to predict (or somehow know) the future values of those covariates as well. In reality the practitioner will need to use their domain expertise to generate reliable estimates of the predictors out into the future. In some cases, the best approach may be to just use another model to forecast future values for these variables. However, cautionary note: generating forecasts from forecasts is potentially dangerous territory! For the purposes of this example, we estimated the 12-month forecast for each covariate using a univariate BTS model (Pybats using just a single variable), and used those projections as inputs when predicting the target series — against our better judgement.

The plot below shows how the model predictions the next 12 months. I've also provided the comparable ARIMA prediction using the same data. Note that ARIMA predictions are notably higher than those of the Bayesian model. We'll need to track these as data comes in each month. I wouldn't be surprised if the ARIMA model

were constantly off, and the Bayesian model was more accurate, given the historical pattern of transactions.



Conclusion

This article provided a brief introduction to using Pybats for multivariate Bayesian forecasting. There tool is quite powerful, and worth looking into for those needing to produce accurate forecasts leveraging the power of dynamic linear regression models. The Bayesian approach more than makes up for the short-comings of ARIMA, especially where there is insufficient data, multiple variables and needing to understand the relative importance of variables in the model — in a more transparent way than ARIMA can provide.

Time Series Forecasting Bayesian Machine Learning