Sales Forecasting with Gradient Boosted Machines and ARIMA

EN.625.740 Data Mining

Jack Fletcher

NOTE: This is the `pdf` version of the paper. I **strongly** recommend using the `html` version instead. Some plots are out of order and/or cut off in the pdf.

1. Introduction

The purpose of this project is to compare and contrast the forecasting ability and application of Machine Learning (ML) techniques with more traditional methods such as ARIMA using data from an ongoing Kaggle competition (Alexis Cook 2021). A growing body of evidence is accumulating that suggests that ML forecasting methods tend to outperform traditional methods in large datasets with many independent variables and many individual time series. One of the primary hypotheses for this performance gap is that ML models are able to learn from many time series at once (i.e., within the same model) whereas traditional methods like ARIMA can only model one series at a time (Kontopoulou et al. 2023). In univariate contexts, traditional methods perform extremely well and performance gains from ML methods are negligible (and often perform worse). That said, businesses and organizations rarely find themselves in situations with access only to univariate series; on the contrary, organizations often have access to huge datasets and exquisite computational resources.

One of the primary use cases for forecasting in the industry is sales forecasting. Many consumer products organizations and grocery stores sell hundreds or thousands of products, all (or most) of which require forecasts for inventory management purposes. This project in particular looks at sales data from a large grocery store chain in Ecuador. I compare the forecasting ability of more "traditional" statistical methods such as ARIMA and SARIMA with gradient boosted regression trees. My findings closely mirror the results of recent forecasting competitions such as the M4 and M5 competitions (Makridakis, Spiliotis, and Assimakopoulos 2022); gradient boosted machines tend to perform better than more traditional methods.

2. A Primer on Time Series Forecasting

Stationarity, Seasonality, and Trend

Stationarity is one of the most important concepts in time series analysis. The basic idea is this: for a series to be forecastable, we need to be able to use its recorded history to predict its future. A series that wildly changes and has no predictable pattern will, by definition, not be forecastable. From a technical perspective, a series that is said to be "covariance stationary" has a constant mean and stable covariance structure. A constant mean is self explanatory, but a stable covariance structure less so. One way to think about what it means to have a "stable covariance structure" is to consider what we are attempting to do when we forecast. We are trying to use the observed relationships in our data to predict some future value. In doing so, one of the things we are assuming is that those relationships we are learning from in the past will remain into the future! If this expectation is not met, then what use is trying to learn from the past? A more technical way to describe this requirement follows.

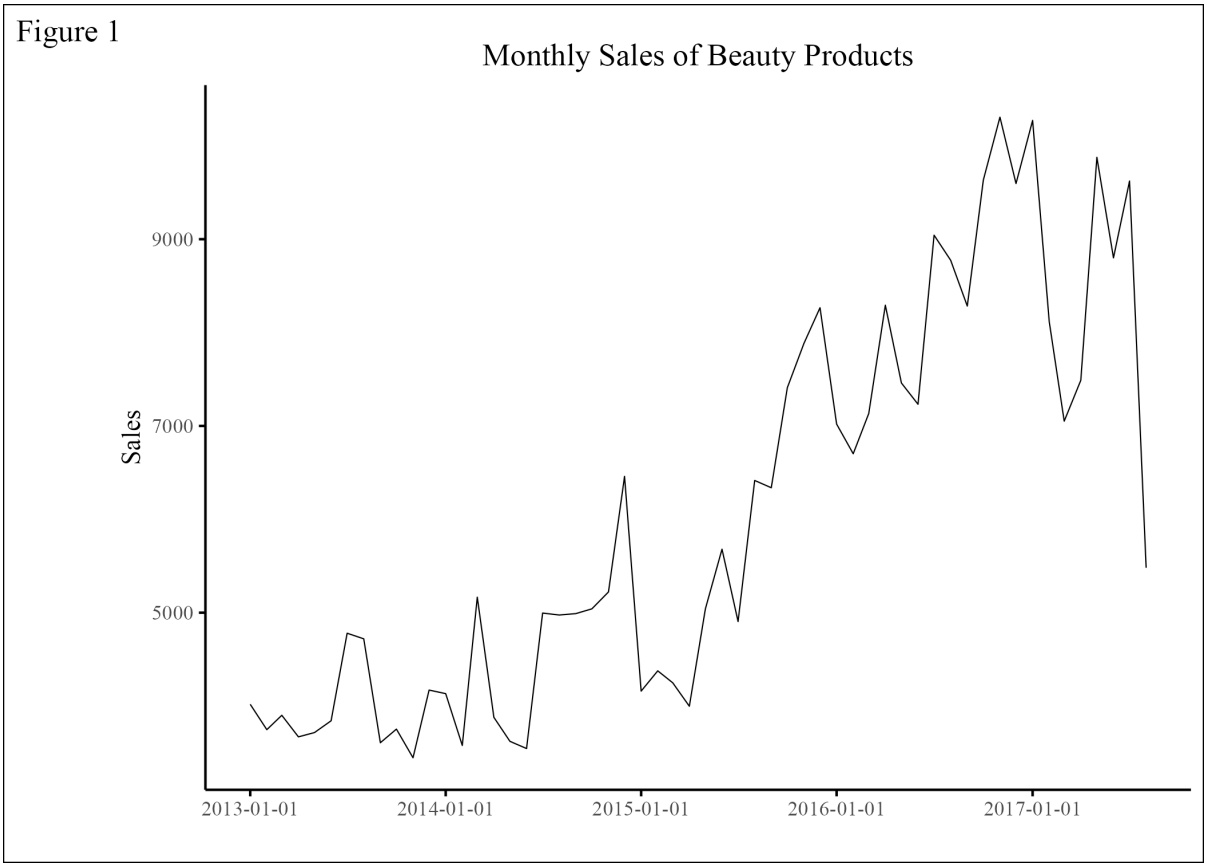
y_t is our observed outcome (sales) at time t. Autocovariance is the covariance between y_t and $y_{t-\tau}$ where τ is some displacement ("lag"). So, if $\tau = 1$, then y_{t-1} is just the single lag of sales. The autocovariance between y and some displacement of y is given by

$$cov(y_t, y_{t-\tau}) = E(y_t - \mu)(y_{t-\tau} - \mu)$$

To meet the definition of stationarity, then the autocovariance *can only depend on the displacement, not on time.* In words, the covariance between sales today and seven days ago must be the same as the covariance between sales 7 days ago and sales 7 days before then.

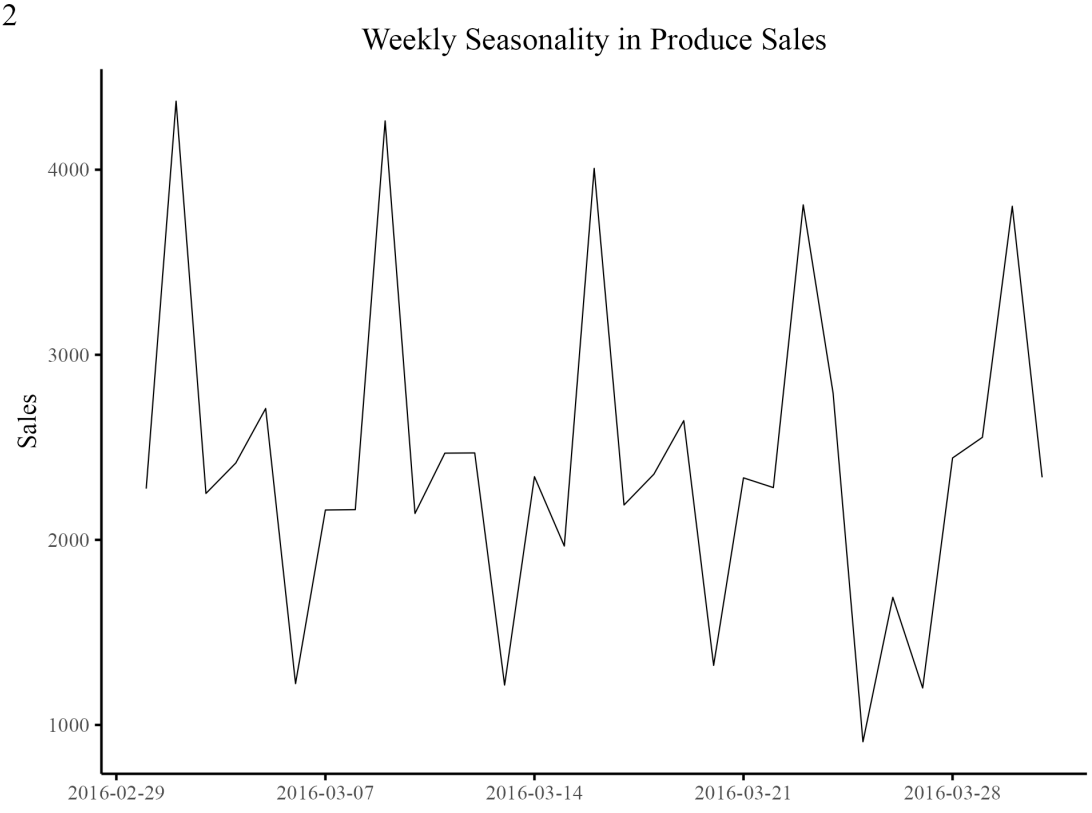
One of the most ubiquitous violations of stationarity occurs when our time series exhibits a trend. A trend is best understood visually. In Figure 1, we observe a very significant upward trend overtime in the sales of beauty products. This obviously violates the requirement of a constant mean. Trends are extremely common in many time series and can take many different forms such as linear, quadratic, exponential, etc.

Figure 1



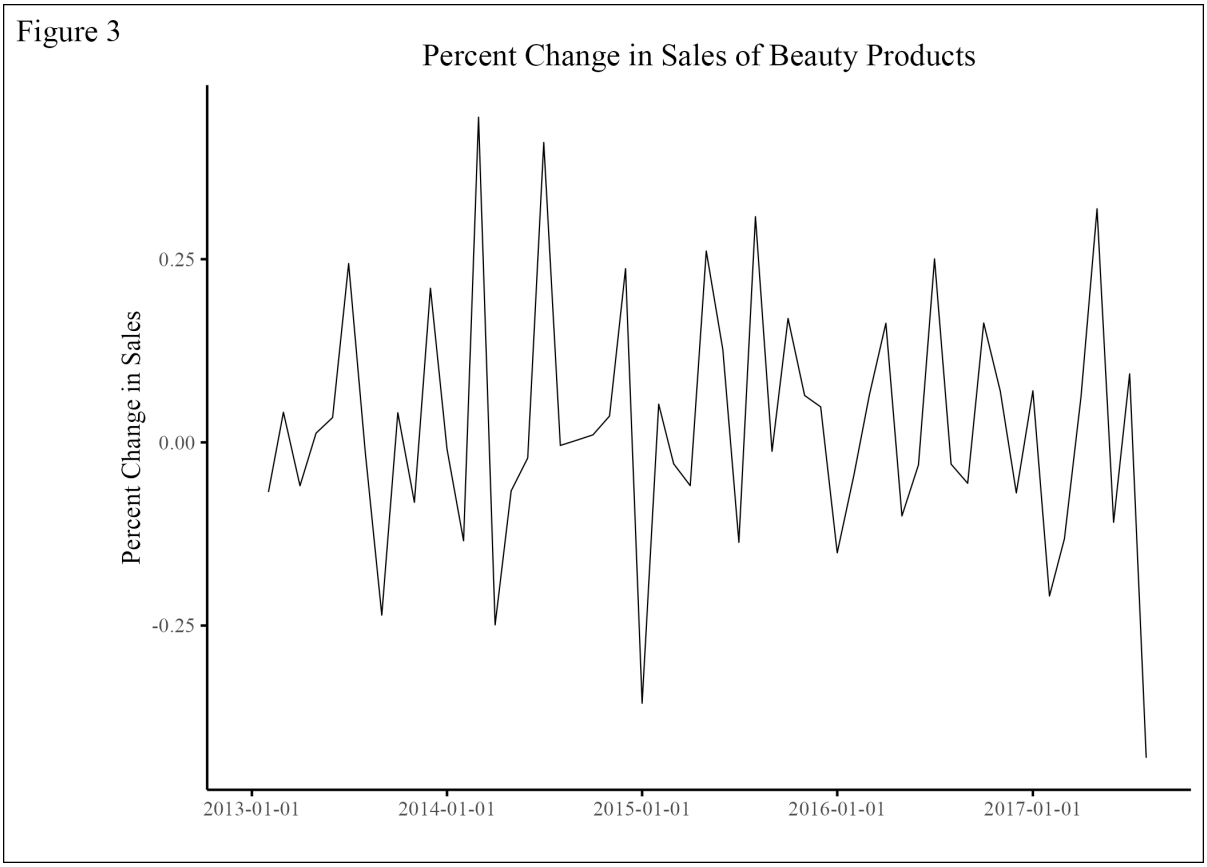
Seasonality in a time series is also best understood visually. In Figure 2 we notice a pattern in produce sales that repeats weekly; at the start of the week, produce sales spike and then sharply decline later in the week. This repetitive pattern is called "seasonality". In this case, we are seeing weekly seasonality, which is very typical of sales data. It is also possible to have other kinds of seasonality such as monthly and yearly.

Figure 2



Both trend and seasonality constitute violations of the stationarity requirement. Luckily, it is often the case that simple transformations of the data series can make them stationary (and therefore predictable). For example, while the volume of beauty product sales clearly has a significant upward trend, Figure 3 shows that the *percent change* in beauty product sales does not.

Figure 3

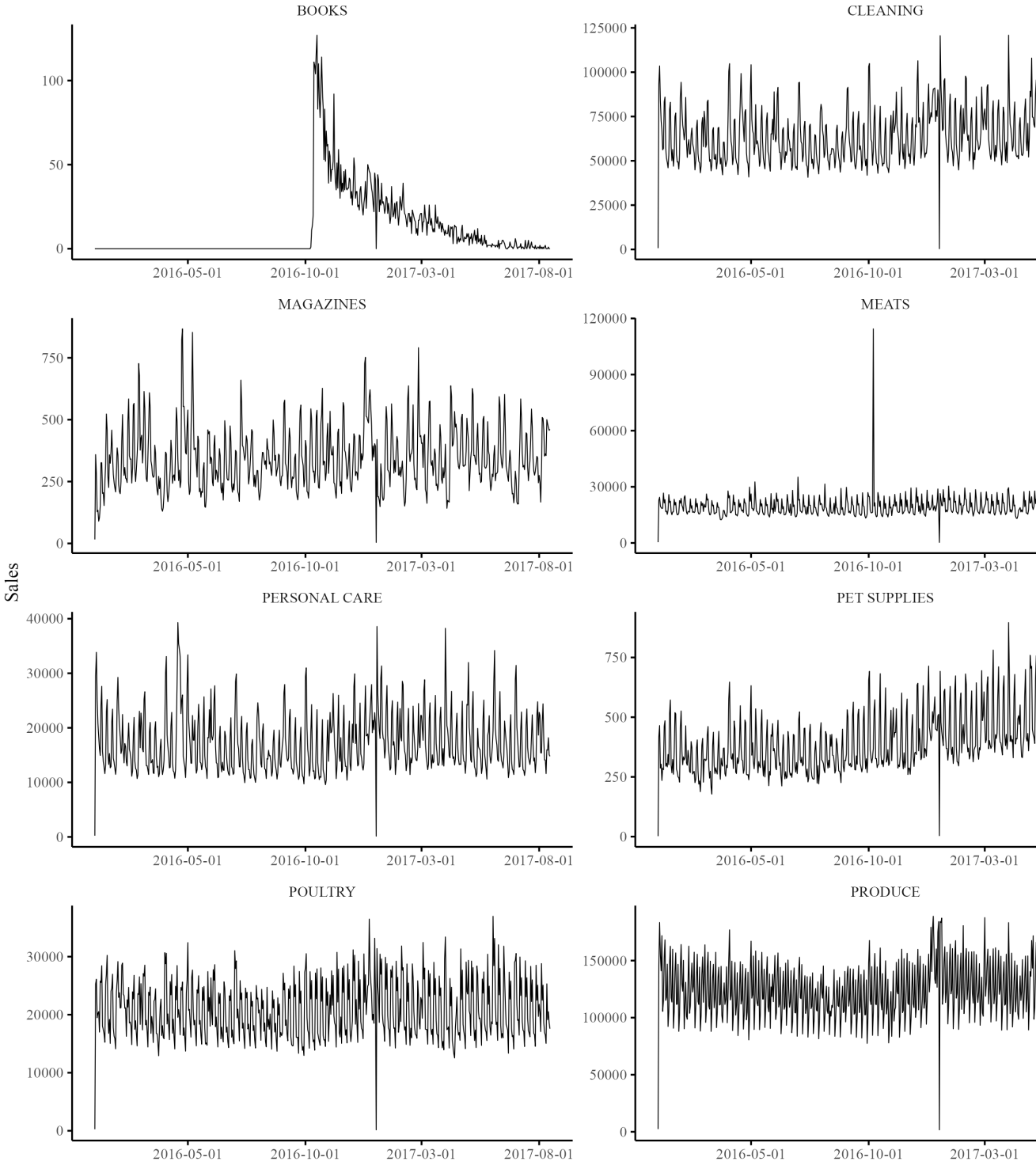


3. The Dataset

The dataset for this project comes from an ongoing Kaggle competition to forecast sales for a large grocery store chain in Ecuador (Alexis Cook 2021). The dataset is at the product family and store level, and contains daily sales volume of 33 product families from 54 stores between January 2013 and August 2017. In total, there are about 3 million observations.

One of the complicating factors of the problem before us is that rather than having a single time series to forecast, our problem may call for as many as 1,782 forecasts! (One for each of the 33 x 54 product family-store combinations). To further complicate matters, it is very reasonable to expect that the trend and seasonal components of each individual product family-store combinations could be different (see Figure 4).

Figure 4


BOOKS
CLEANING
MAGAZINES
MEATS
PERSONAL CARE
PET SUPPLIES
POULTRY
PRODUCE
Sales
2016-05-01
2016-10-01
2017-03-01
2017-08-01

The primary dataset itself has a fairly limited number of variables (see Table 1). Luckily, often we can produce fairly robust forecasts with only the outcome variable itself (as long as it is roughly stationary!). In addition, we can use what data we do have to create many more features including things like calendar features (day, day of week, month, holidays), store features, and window aggregations.

Table 1: Table 1

Variable	Description
Date	The current date
Store Number	Corresponds to the store where the sale occurred
Family	The product family
Sales	The target/dependent variable
On Promotion	The number of items within the product family that were promoted that day

We also have access to a secondary dataset with store metadata containing things like the city and state that a store is in. In the next section, we discuss some of the model-specific variables that are created which characterize the overall structure of the model itself.

4. Methods

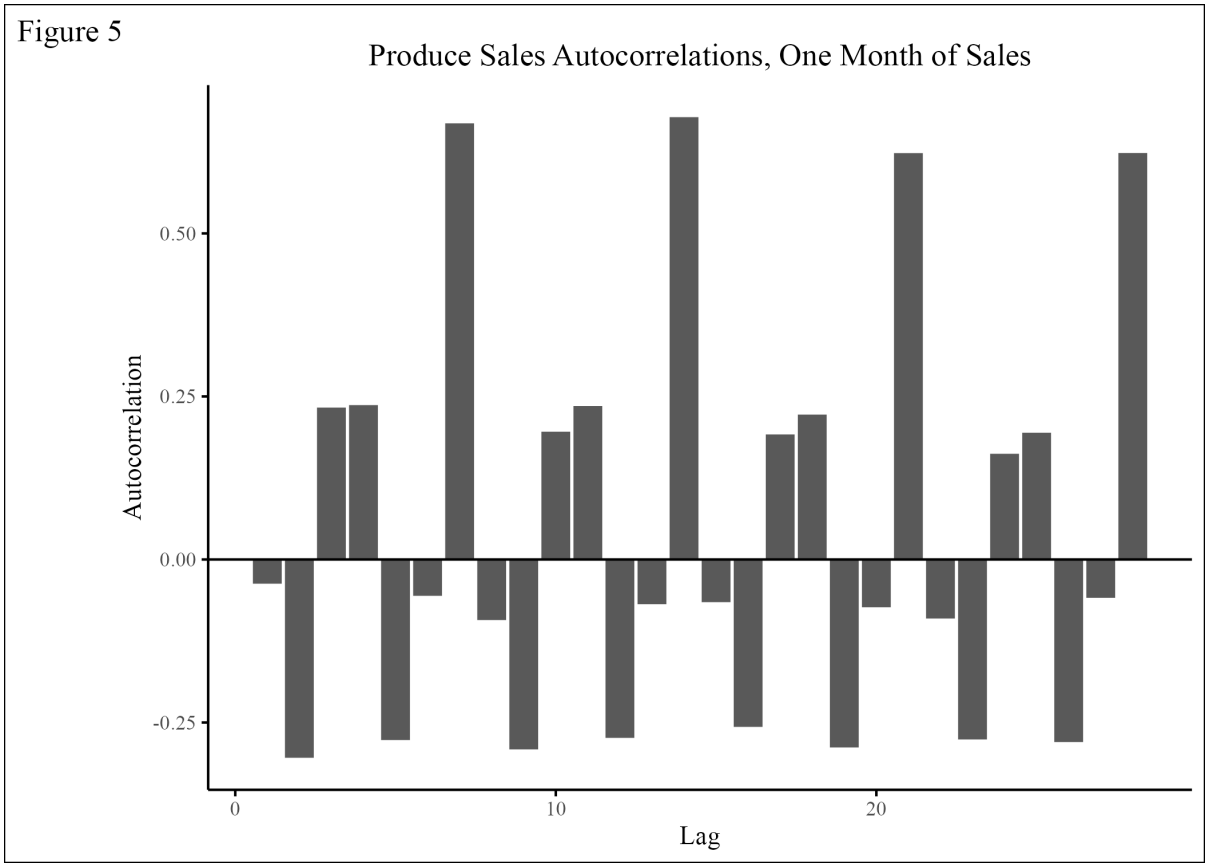
This section describes the various models used in this project to forecast sales. Broadly, I used both traditional forecasting approaches (such as ARIMA) as well as a Machine Learning approach (gradient boosted machine). In this section, I briefly review the concepts underpinning each approach, beginning with the statistical models and finishing with gradient boosted machines.

Decomposing Time Series with AR, MA, ARMA, ARIMA, and SARIMA

Autocorrelation

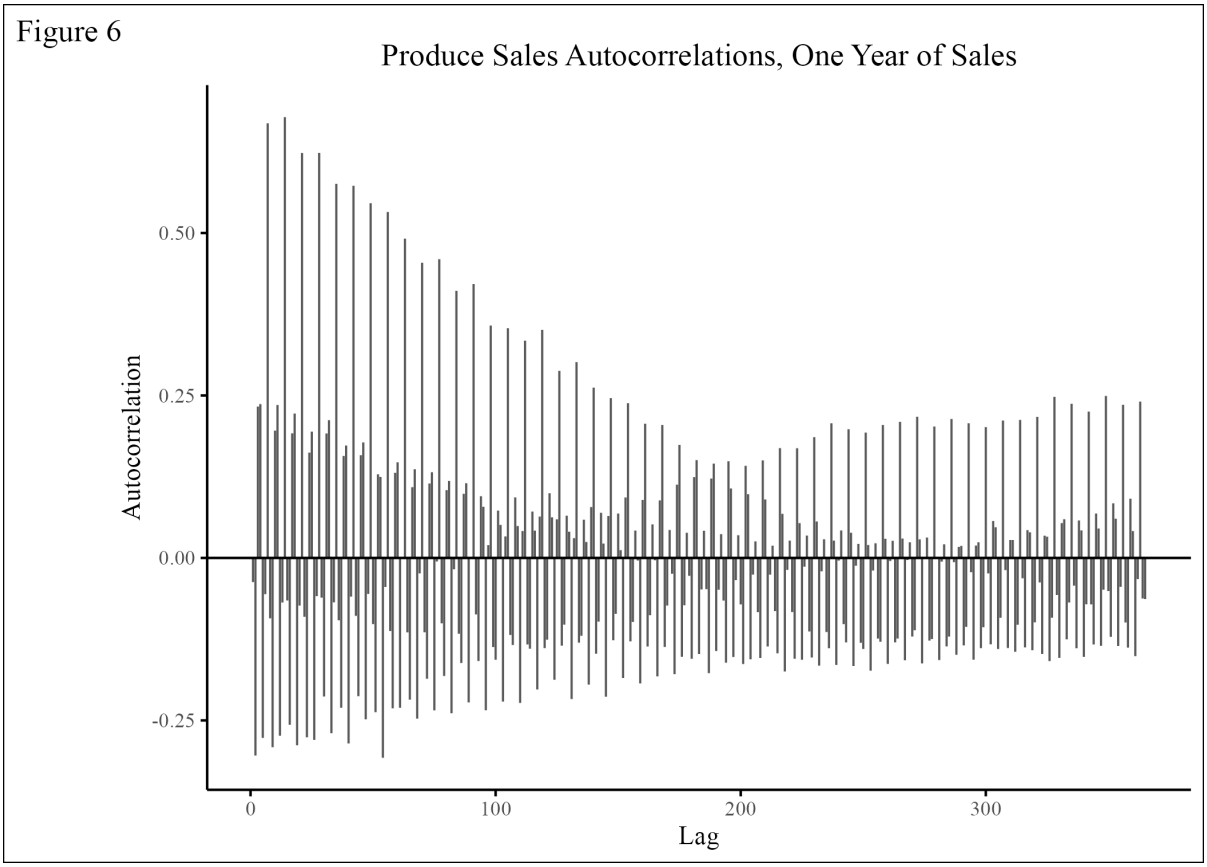
In Section 2 we discussed the concept of autocovariance. An obviously related idea, autocorrelation, is foundational for many of the statistical methods for forecasting. Most forecasting models are exploiting the autocorrelations between observations to make predictions about future observations. One way to visualize this is through the use of an autocorrelation plot (see Figure 5). In short, this plot displays the correlation between a series and its lags. Figure 5 show significant autocorrelation between sales and its weekly lags (at displacements 7, 14, 21, and 28).

Figure 5



We can increase the range of our plot to show autocorrelations for an entire year. Interestingly, it appears that the strong weekly autocorrelations (the tallest bars in the plot) steadily weaken until about 6 months where they taper off and remain constant for the remainder of the year.

Figure 6



Autoregressive models

If we want to forecast produce sales, one of the first things we might try is to exploit the autocorrelational structure of the series. Autoregressive ("AR") models do just that. An autoregressive model for sales is simply a regression of sales on lags of sales. An AR(1) model is a regression of sales on a single lag of sales,

$$sales_t = \beta_0 + \beta_1 sales_{t-1} + u_t$$

An AR(2) model is a regression of sales on two lags of sales,

$$sales_t = \beta_0 + \beta_1 sales_{t-1} + \beta_2 sales_{t-2} + u_t$$

and so on.

Here is what an AR(1) may look like.

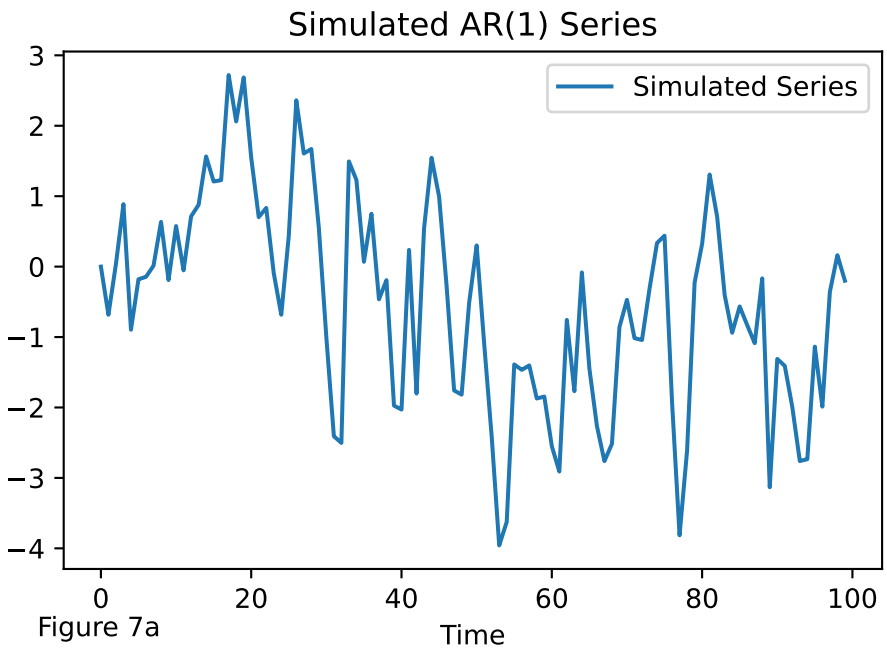


Figure 7a

Here is the same series, this time with a forecast for the last 10 steps of the series using an AR(1) model.

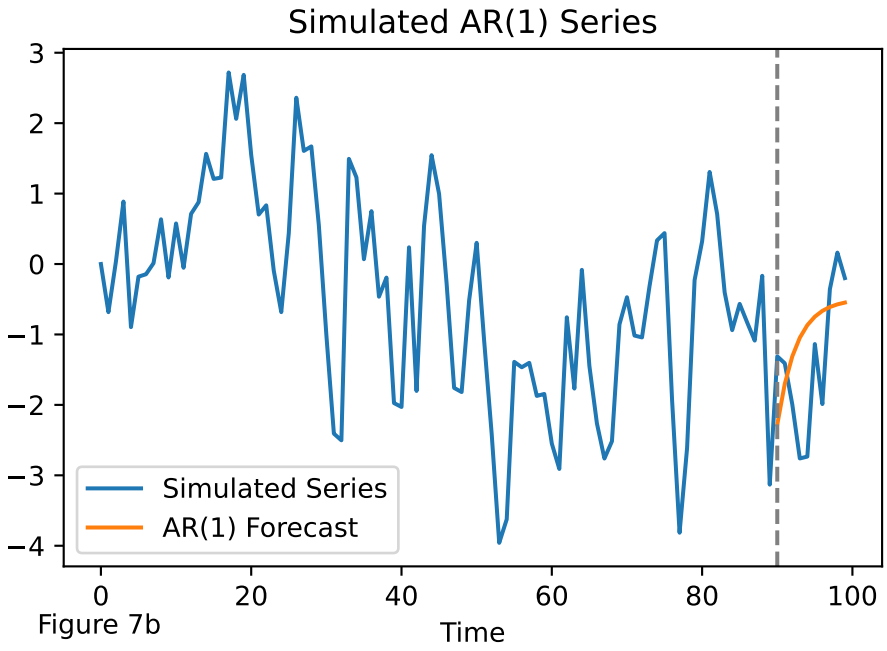


Figure 7b

Clearly, autoregressive models steadily revert back to the mean of the series over time.

Moving average models

Moving average (MA) models work similarly to AR models except rather than include autoregressive terms ("lags") of the outcome variable, MA models include lagged errors. An MA(1) model includes one lagged error term:

$$sales_t = \beta_0 + \beta_1 u_{t-1} + u_t$$

Of course, we cannot actually observe population errors, so to make things operational we substitute in their estimate (the residuals!):

$$sales_t = \beta_0 + \beta_1 \hat{u}_{t-1} + u_t$$

A simulated MA(3) series is below. MA(q) models have a q-period long memory. After q periods, their prediction becomes the unconditional mean of the series. Because of this, AR models generally capture persistent dynamics that MA models cannot.

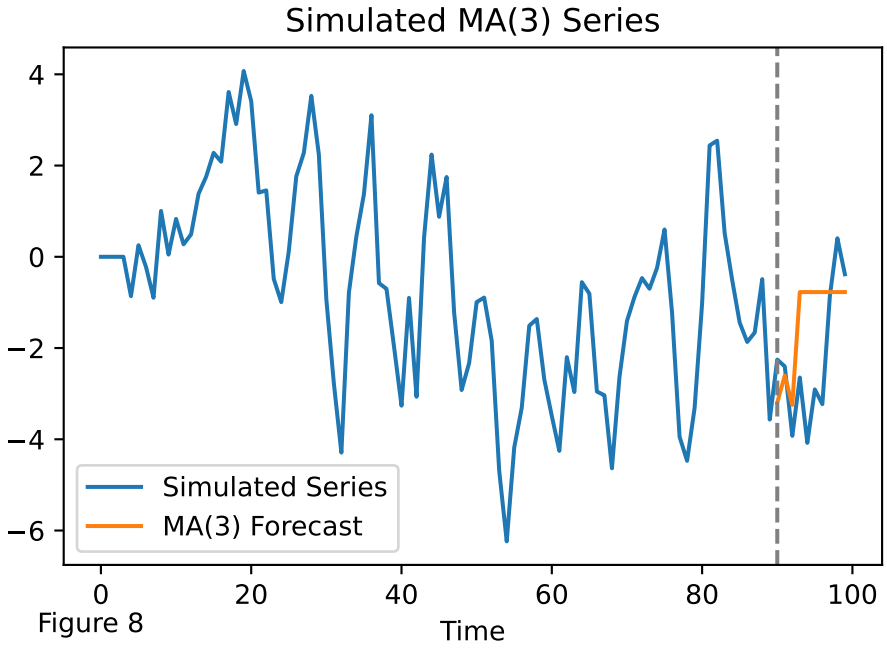


Figure 8

Autoregressive moving average (ARMA) models

In the real world, most time series exhibit both AR and MA dynamics. It turns out we can implement a model that combines both of these components. So-called "ARMA" models contain both autoregressive *and* moving average terms. For example, an ARMA(1,1):

$$sales_t = \beta_0 + \beta_1 sales_{t-1} + \delta_1 \hat{u}_{t-1} + u_t$$

Autoregressive integrated moving average (ARIMA) models

One weakness of AR, MA, and ARMA models is that they cannot handle data with a trend. Below is a simulated ARMA(2, 1) process with a positive linear trend. The orange line is a forecast from an ARMA(2, 1) model. The presence of the trend makes the data non-stationary and therefore not able to be forecasted with AR and MA terms alone. Enter ARIMA. It turns out that we can difference the data *and then* model it with AR and MA terms. An ARIMA(2, 1, 1) model transforms the series by first differencing and then applying 2 autoregressive terms and 1 moving average term. By first differencing, our forecast is now much better (green line).

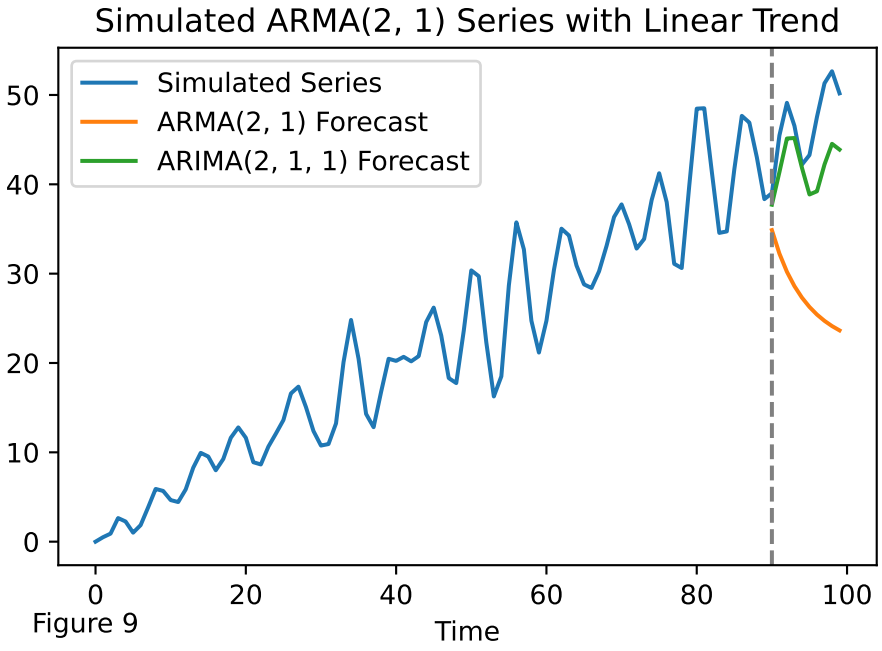


Figure 9

Seasonal ARIMA (SARIMA) and including other X variables

In the plot below, we show a forecast of grocery sales using an ARIMA(1, 0, 1) model.

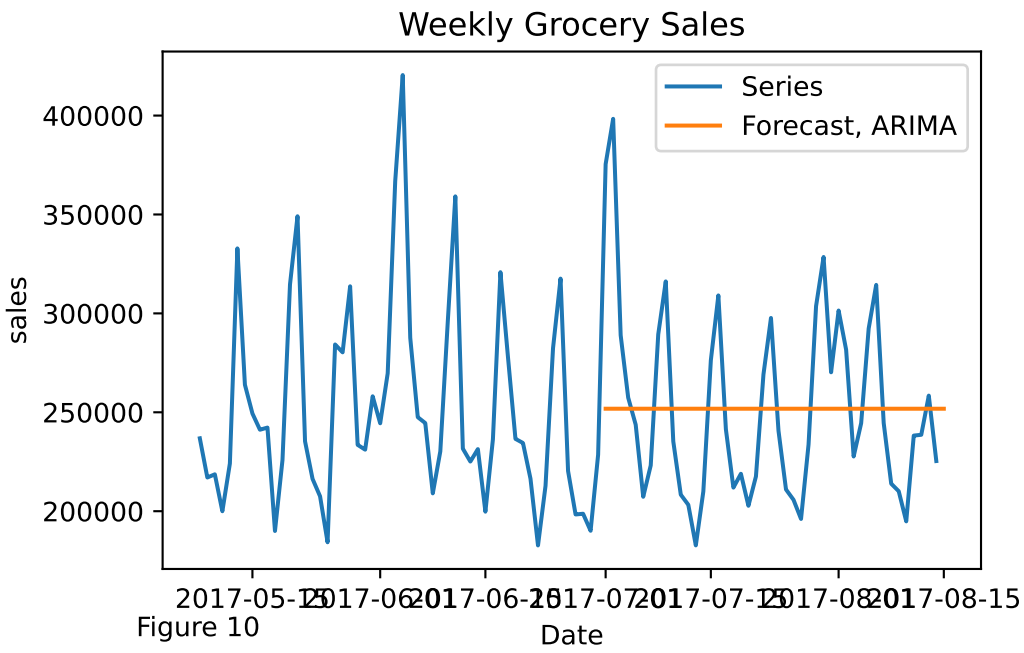


Figure 10

It turns out that this forecast amounts to little more than a straight line. Our model is currently missing out on all the seasonality that is clearly present in grocery sales. To capture this, we can use what is called a *seasonal* ARIMA. We know from our analysis in Section 2 that produce & grocery sales exhibit strong weekly seasonality. One way to handle this is to include *seasonal* AR, MA, and I terms. Our typical ARIMA(1, 0, 1) model is

$$sales_t = \beta_0 + \beta_1 sales_{t-1} + \delta_1 \hat{u}_{t-1} + u_t$$

Including seasonal AR and MA terms,

$$sales_t = \beta_0 + \beta_1 sales_{t-1} + \beta_2 sales_{t-7} + \delta_1 \hat{u}_{t-1} + \delta_2 \hat{u}_{t-7} + u_t$$

We now have a SARIMA(1, 0, 1)(1, 0, 1) model. Many statistical packages allow a user to input a `season_length` argument which tells the computer how long the seasonal/cyclical pattern lasts. Since we are dealing with weekly seasonality, our `season_length` parameter should be set to `7`.

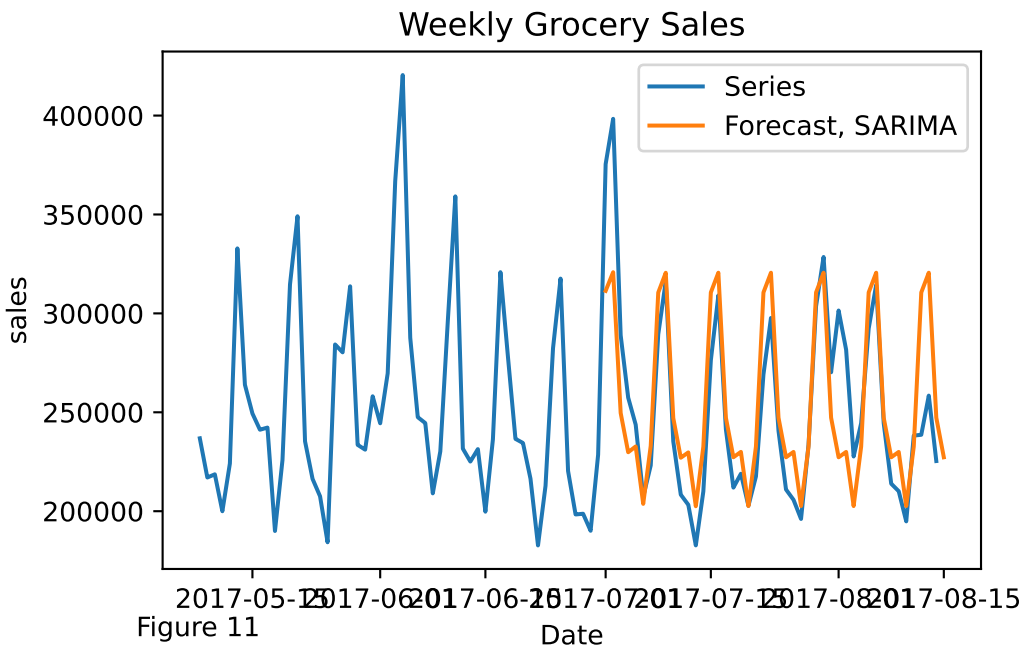


Figure 11

Our model is clearly *much* better than before.

Finally, it can often be helpful to include other $\mathbf{X}$ variables in our model when available. This can be especially useful when we actually know what future values of $\mathbf{X}$ will be during the period for which we are making a forecast. In sales forecasting, this often includes things like known future price changes.

We can include these variables in the typical way. A general model with AR, MA, and additional explanatory variables is given by:

$$sales_t = \mathbf{y}^T\theta + \mathbf{u}^T\phi + \mathbf{x}^T\beta$$

where $\mathbf{y}$ is a vector of lagged sales values (with AR coefficients θ), $\mathbf{u}$ is a vector of lagged error terms (with MA coefficients ϕ), and $\mathbf{x}$ is a vector of explanatory variables.

Auto ARIMA

In this project, Auto Arima was used to select the best SARIMA order. Auto ARIMA is a relatively simple approach to finding the optimal AR, MA, and I order (and their seasonal counterparts) using something akin to grid search. In addition, Auto ARIMA carries out a number of statistical tests for stationarity and stability of estimates (Rob J. Hyndman and Khandakar 2008). Auto ARIMA searches for models that produce a low AIC/BIC.

$$AIC = \exp\left(\frac{2k}{T}\right)\frac{\sum_{t=1}^{T} e_t^2}{T}$$

$$BIC = T^{\frac{k}{T}}\frac{\sum_{t=1}^{T} e_t^2}{T}$$

Where k is the number of model parameters, T is the number of time-steps (observations), and e^2 are the in-sample squared residual errors.

Both AIC and BIC are estimates of expected *out-of-sample* forecasting performance. They heavily penalize degrees of freedom so as to balance model complexity and performance.

Gradient Boosted Machines

Decision Trees & The Classification and Regression Tree (CART) Algorithm

Decision trees are powerful and versatile Machine Learning algorithms that serve as the foundation for gradient boosted decision trees and random forests. Whether we are dealing with a classification or regression task, decision trees utilize something called the *classification and regression tree* (CART) algorithm to make predictions. The algorithm begins by splitting the training set in half on one variable at some value of that variable. In a regression setting, the mean target value for each subset becomes the prediction and the mean squared error (or another user-specified loss) for each subset is calculated. The algorithm greedily searches for variables and thresholds that produce the best performance across each subset. Once the lowest loss is achieved, the algorithm repeats the entire process on further subsets of the original two subsets. In general, the algorithm proceeds in this fashion until it reaches a user-defined "maximum depth" (the number of optimal splits found) or until there is no improvement to the loss function by further splits.

To make predictions with a trained decision tree, instances are fed into the model and travel down the decision tree taking the path that was generated in the training process corresponding to the lowest possible loss at each split point. One interesting thing to note about decision tree regression (and classification) is that all instances that land on the same leaf node will get the same prediction even if their $\mathbf{x}$ values are slightly different.

Boosting - Combining Many Decision Trees

The idea behind gradient boosting and other *ensemble methods* is to combine many models into one, thereby producing more accurate predictions (hopefully!). Gradient boosted decision trees do this by sequentially fitting decision trees one after another, each with the goal of predicting the *errors* that the prior model made. In a regression context, the final prediction

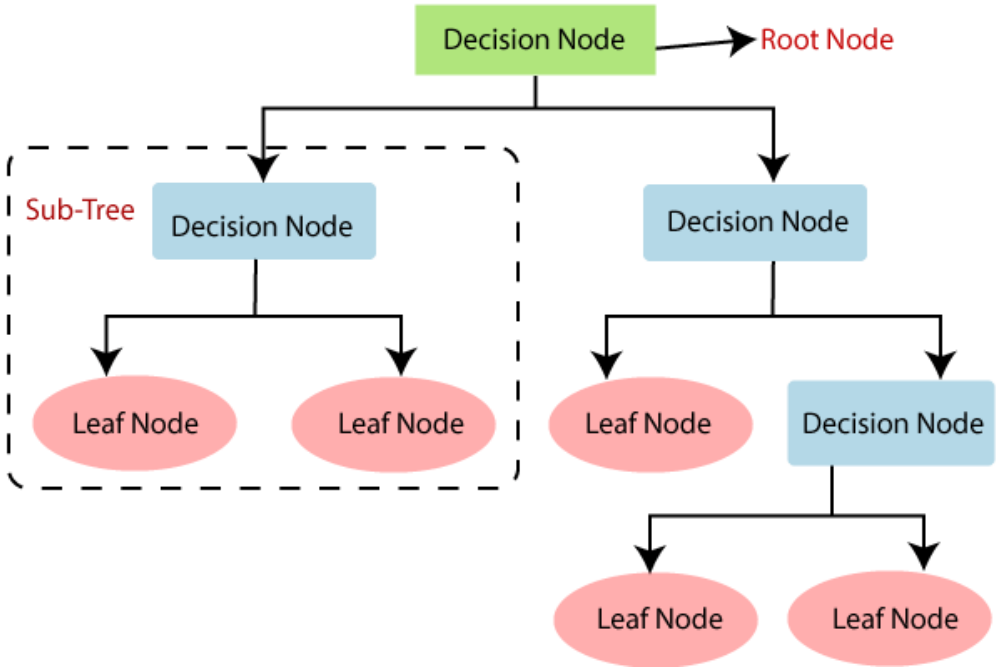


Figure 1: Source: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm

made for a particular instance is simply the sum of all predictions from each of the regression trees.

Regularization

Before moving on, it is worth mentioning one of the biggest weaknesses of decision trees (and therefore gradient boosted decision trees): overfitting. Decision trees are extremely flexible nonparametric models that can easily learn from patterns that are not part of the true data generating process. Because of this, there are a fair number of regularization strategies that can be employed to limit the risk of overfitting. One of the most powerful strategies has already been mentioned: the maximum depth of the tree. By limiting the number of splits, we are reducing the risk that the model fits to noise in the data. Other strategies include:

- Setting a minimum requirement for the number samples that must be present in a node before a split is made
- Setting a maximum number of variables that are searched through by the CART algorithm before settling on an optimal split
- Setting a maximum number of leaf nodes that a tree can have

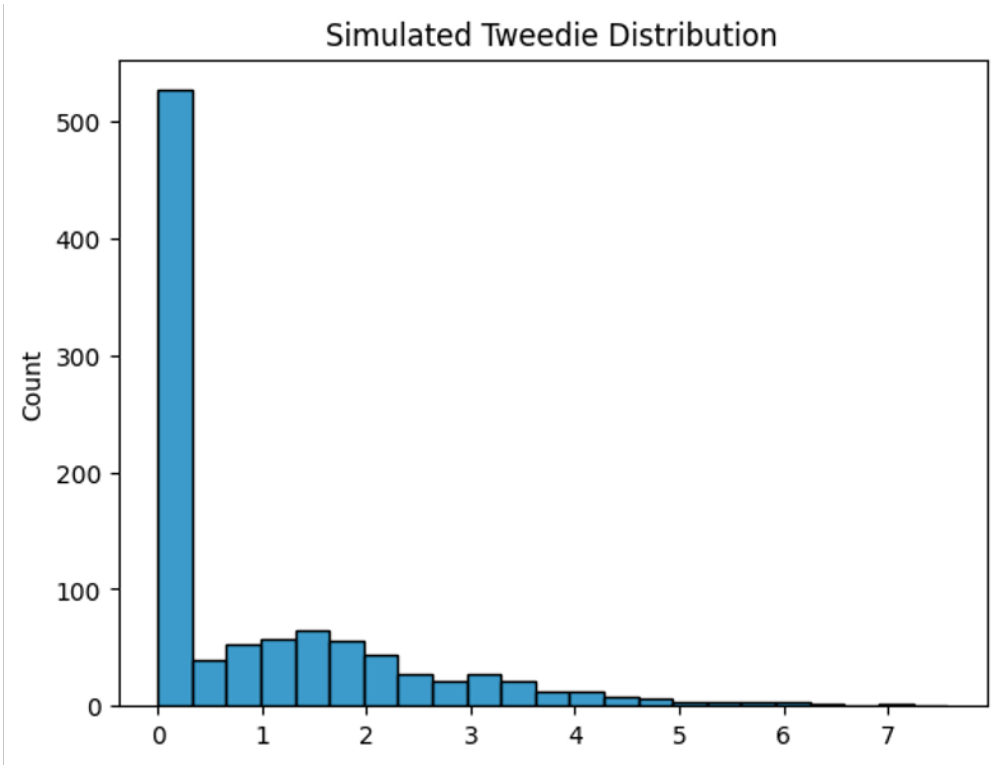
We can also regularize at the level of the ensemble rather than the individual trees. For example, during the training process we can make the algorithm randomly select instances to use before growing a new tree. We can employ the same strategy with the variables themselves; the algorithm must randomly select a proportion of the features to use in growing the next tree. Finally, we can also limit the number of trees that are grown. A particularly handy strategy is called "early stopping" and involves monitoring the training and validation loss for each tree. Once validation loss has leveled off, the algorithm can stop itself from growing more trees.

Feature Engineering & Loss Function

As mentioned in Section 3, the original dataset had a fairly limited number of variables with which to build a model. That said, through feature engineering I was able to create many more. The features used included:

- The number of products from a particular family that were on promotion
- Calendar features (day, month, day of week)
- Window averages (7, 14, 21, 28 day)
- Window standard deviations (7, 14, 21, 28 day)
- Lagged sales (7, 14, 21, 28 day)
- Differenced sales (7, 14, 21, 28 day)
- Store type (dummy variable encoded)
- Store cluster (dummy variable encoded)

The loss function used in the boosted regression trees was a Tweedie loss (Chen et al. 2020). Tweedie distributed random variables are typically right skewed and zero-inflated. Tweedie distributions are most typical in insurance pricing and claims, where there are significant numbers of individuals that make no claims followed by a long right tail distribution. The sales data in this project also mirrored a Tweedie distribution; many smaller product families have very few or zero daily sales on somewhat frequent basis. Following significant success of Tweedie loss in the M4 and M5 competitions (Januschowski et al. 2022), I decided to compare performance of models trained with a Tweedie loss to models trained with a squared error loss. As expected, the squared error loss results in very poor performance for smaller product families that make fewer sales.



5. Training and Evaluation

Training and validating time series models is different from training and validating other kinds of prediction models. The key difference stems from the fact that time series data are not independently and identically distributed (i.i.d). Whereas in some contexts we can randomly split our data into training and testing sets (for example, with cross sectional data), to do so with time series data would be to ignore its time-based structure. With forecasting, all that matters is how well the model can predict the future; when trying to estimate our model's performance, we should only ever test on future, unseen data.

Error Metrics

Briefly, the error metrics I use in this project include more traditional measures like Mean Squared Error (MSE) as well as more recent "scale-free" errors such as Root Mean Square Scaled Error (RMSSE).

Scale-Dependent Errors

So-called "scale dependent errors" are, as their name imply, dependent on the scale of the target outcome. Therefore, it is inappropriate to compare these measures across forecast models for different series.

Mean Squared Error

The MSE is defined as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

A weakness of MSE is that it puts relatively more weight on outliers. For that reason, mean absolute error (MAE) is sometimes preferred. That said, for my purposes I decided to stick with MSE.

Root Mean Squared Error

MSE is a little hard to interpret because it is in squared units of our target. RMSE is an alternative to MSE measured in the same units as our target and is therefore easier to interpret.

$$RMSE = \sqrt{MSE}$$

Scale-Independent Errors

Scale-dependent errors are not ideal for our project. As we mentioned in Section 3, we are essentially trying to forecast roughly 1,700 individual time series: one for each of the 33 times 54 different product family-store combinations. In our data, some product families such as produce and cleaning sell vastly more units than product families like electronics and automotive. Because of this, metrics that are independent of the target scale are attractive.

Root Mean Square Scaled Error (RMSSE)

RMSSE is defined as the ratio of the MSE of our forecast divided by the MSE of a naive forecast calculated in-sample (over the training set). Most often the naive forecast is simply a one-step ahead forecast of each value in the training set.

$$RMSSE = \frac{MSE}{MSE_{naive}}$$

If $RMSSE < 1$ then the forecast is better than a naive one-step ahead forecast.

Weighted Root Mean Square Scaled Error (WRMSSE)

We can calculate and RMSSE for each individual series in our dataset. But how do we assess the relative performances of product families of different sales volume? For illustrative purposes, imagine the RMSSE for our produce forecast was 0.6 and the RMSSE for our electronics forecast was 1.1. One way to assess the relative performance is to consider the situation from the point of view of the grocery store company. If produce sales were more important to the company than electronics sales (perhaps from a revenue or sales volume perspective), then perhaps we would want to weigh produce forecast performance more heavily in some aggregate measure. Weighted Root Mean Square Scaled Error attempts to do this.

$$WRMSSE = \sum_{i=1}^{N} w_i * RMSSE_i$$

where N is the number of series we are forecasting (the number of product families, for example). The weights, w, can be calculated in many ways. For my project I used the relative proportion of sales within each product family over the last month of the training set (which was similar to the weight used in the original Kaggle competition).

Forecast Horizons & Validation Strategy

A forecasting horizon is the number of periods that we are predicting into the future. In this project, I use a 28-day (4 week) long forecasting horizon. That is, I am forecasting a sales value for 28 individual days into the future. Traditional statistical forecasting models such as ARIMA use *recursive forecasting* to predict each day in an iterative fashion, one after another. For example, an ARIMA model will predict tomorrow's value and then use tomorrow's *predicted value* to help inform what the value two days from now will be. This process continues until we have a prediction for each day of our horizon. Another approach to forecasting each period in our horizon is sometimes called *direct multi-step* forecasting. Using this strategy, multiple models are trained to directly predict a specific period within the forecasting horizon. For example, we may have 28 individual models, each trained to predict one specific day of the

month. An alternative strategy may be to train one model for each week of the month (this is the forecasting strategy I use for the ML models in this project).

As mentioned above, when training forecasting models it really only ever makes sense to evaluate our models on future, unseen data. The validation setup utilized in this project is as follows.

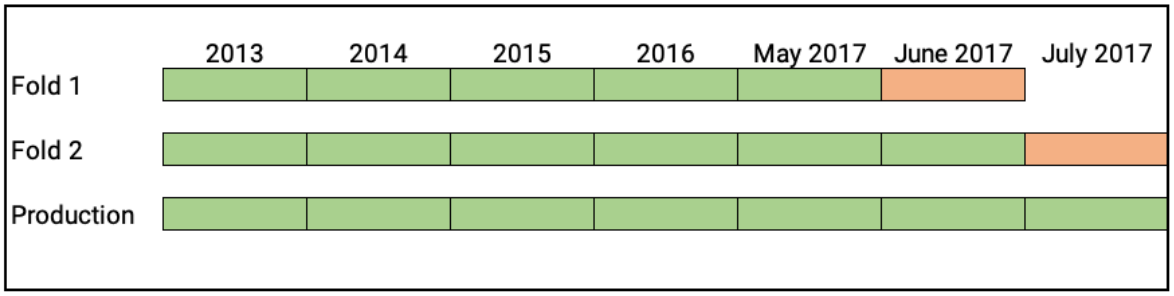
Validation Setup

Training: *2013-01-01 through 2017-05-31*

Validation 1: *2017-06-01 through 2017-06-28*

Validation 2: *2017-07-01 through 2017-07-28*

First, I trained a model using just the training set and then predicted on the first validation set. Then, I combined the training and first validation sets and retrained a new model to predict the second validation set. If this model was going to be used in production, I would retrain once again using the final validation set and use this model to predict the future.



This process could be improved by adding additional folds. Further, one potential weakness of this approach is that good performance in June (validation set 1) does not necessarily imply good performance in July (validation set 2). This is one of the realities of time series forecasting; the future is always uncertain and future "innovations" (random, unpredictable shocks) are by definition unforecastable. Given more time, one reasonable strategy could be to use prior Julys exclusively as the validation sets (still, there is no guarantee that *this* July is similar to *last* July). As we will see in the Results section of this paper, the performances in each validation set were quite comparable.

6. Results & Discussion

In this project I broadly performed three sets of analyses:

- Statistical models were used to forecast sales at the *product family* level
- GBMs were used to forecast sales at the *product family* level

- GBMs were used to forecast sales at the *product family-store* level. These forecasts were then aggregated up to the *product family* level to produce a bottom-up forecast.

With 1,700 individual product family-store series (totaling 3 million rows of data), training individual ARIMA models for each was not feasible. While training a single ARIMA model is computationally inexpensive, the use of auto ARIMA complicates matters. For each series, dozens of combinations of ARIMA orders are searched through to select the best model. Therefore, I elected to aggregate the data to the product family level. This results in 33 individual models rather than 1,700. It was at this level that comparisons between GBM and statistical models were made. The results are displayed below in Table 2. In general, the GBMs outperform the statistical models in both validation sets.

Table 2

	Mean RMSSE		Weighted RMSSE	
	Val 1	Val 2	Val 1	Val 2
ARIMA	0.955	0.873	1.095	0.971
SARIMA	0.579	0.55	0.541	0.463
SARIMAX	0.558	0.624	0.577	0.514
GBM (Bottom-up)	0.52	0.505	0.536	0.481
GBM (Direct)	0.541	0.492	0.596	0.507

Validation Set 1 Performance Summary

The lowest mean RMSSE across all product families in the first was achieved by the bottom-up GBM forecast (0.52). This performance was several points better than the performance achieved by the other models tested. The lowest weighted RMSSE in the first validation set was also achieved by the bottom-up GBM forecast (0.536), though closely followed by the seasonal ARIMA.

Validation Set 2 Performance Summary

The lowest mean RMSSE was essentially a tie between the bottom-up and direct GBM forecasts (around 0.5). Interestingly, the lowest weighted RMSSE was achieved by the seasonal ARIMA, closely followed by the GBMs.

It turns out that when weighting the RMSSE scores by the size of the product family, the relative performances of the ARIMAs and GBMs converge. This is likely because the comparative advantage of GBMs (and their Tweedie loss) to be more accurate on the smallest product families becomes less important.

Below I plot the top 4 product families (in terms of sales) alongside their forecast from the best GBM (bottom-up) and the best seasonal ARIMA. Both forecasts seem to do quite well,

but a few differences between the two illustrate how the ML forecast may be a little more flexible in the face of irregular shocks to the series. Note how in late June/ early July there were shocks to sales across all 4 product families. The shape of the series completely changes in both Produce and Poultry, something that the XGB model handles much better than the seasonal ARIMA.

These results provide some evidence that Machine Learning approaches such as GBMs can offer great forecasting performance. While the GBMs generally performed slightly better, the traditional methods such as seasonal ARIMA still do quite well. Of note, very little hyperparameter tuning was done in training the ML models for this project. With more time spent tuning models, I can imagine that performance will only improve. Apart from slight performance gains, the advantage of GBMs over the more traditional methods was their ability to forecast at the product-store level, something that was computationally prohibitive for me using ARIMA.

One result that I found interesting was the worse performance of the seasonal ARIMA model with additional regressors. My leading hypothesis is that the additional regressors simply added unhelpful noise to the prediction. With more time, it would probably be beneficial to add a layer of feature selection to the training process (to get rid of unhelpful variables)

7. Notes

All of the statistical analysis for this project was done in python using the `statsforecast`, `sklearn`, and `xgboost` libraries. In addition, plotting was done in both `matplotlib` & `seaborn` as well as `ggplot2` using R.

8. References

Alexis Cook, inversion, DanB. 2021. "Store Sales - Time Series Forecasting." Kaggle. https://kaggle.com/competitions/store-sales-time-series-forecasting.

Chen, Chaochao, Ziqi Liu, Jun Zhou, Xiaolong Li, Yuan Qi, Yujing Jiao, and Xingyu Zhong. 2020. "How Much Can A Retailer Sell? Sales Forecasting on Tmall." *CoRR* abs/2002.11940. https://arxiv.org/abs/2002.11940.

Diebold, F. X. 2017. "Forecasting." 2017. http://www.ssc.upenn.edu/~fdiebold/Textbooks.html.

Federico Garza, Cristian Challú, Max Mergenthaler Canseco. 2022. "StatsForecast: Lightning Fast Forecasting with Statistical and Econometric Models." PyCon Salt Lake City, Utah, US 2022. https://github.com/Nixtla/statsforecast.

Géron, Aurélien. 2019. *Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow.* 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly.

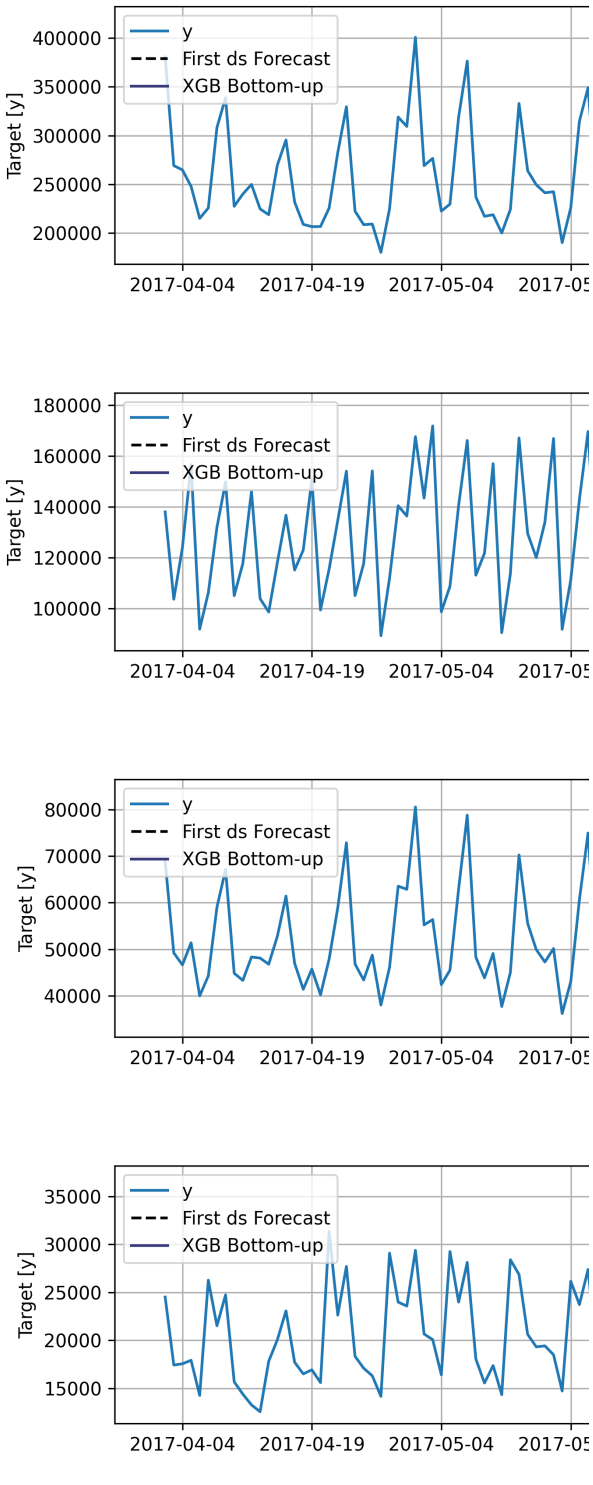


Figure 2: XGB Bottom-up

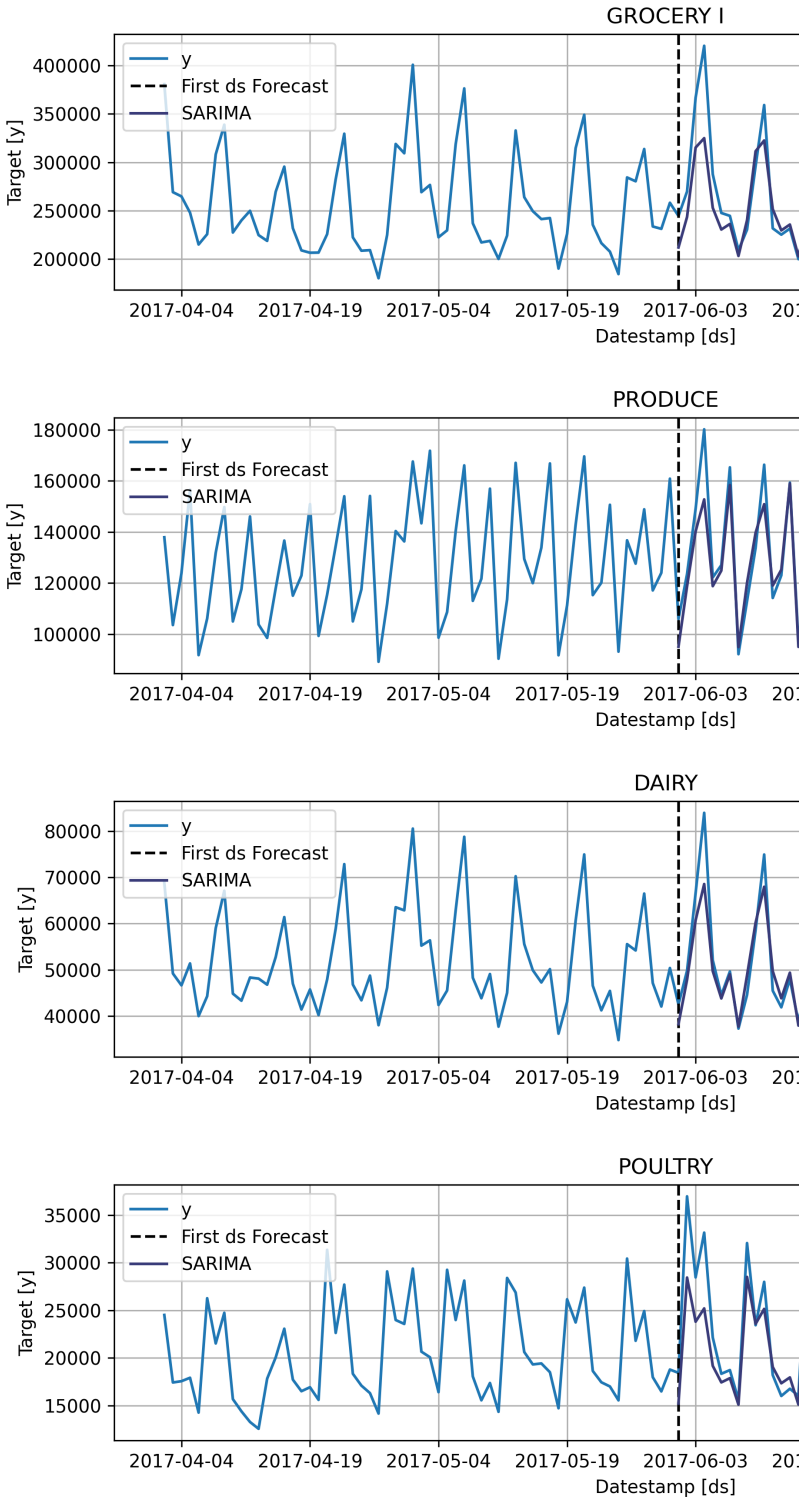


Figure 3: Seasonal ARIMA

Hyndman, Rob J. 2015. "Another Look at Forecast-Accuracy Metrics for Intermittent Demand." In *Business Forecasting: Practical Problems and Solutions*, 204–11. John Wiley & Sons.

Hyndman, Rob J., and Yeasmin Khandakar. 2008. "Automatic Time Series Forecasting: The Forecast Package for r." *Journal of Statistical Software* 27 (3): 1–22. https://doi.org/10.18637/jss.v027.i03.

Januschowski, Tim, Yuyang Wang, Kari Torkkola, Timo Erkkilä, Hilaf Hasson, and Jan Gasthaus. 2022. "Forecasting with Trees." *International Journal of Forecasting* 38 (4): 1473–81. https://doi.org/https://doi.org/10.1016/j.ijforecast.2021.10.004.

Kontopoulou, Vaia I., Athanasios D. Panagopoulos, Ioannis Kakkos, and George K. Matsopoulos. 2023. "A Review of ARIMA Vs. Machine Learning Approaches for Time Series Forecasting in Data Driven Networks." *Future Internet* 15 (8). https://doi.org/10.3390/fi15080255.

Makridakis, Spyros, Evangelos Spiliotis, and Vassilios Assimakopoulos. 2022. "The M5 Competition: Background, Organization, and Implementation." *International Journal of Forecasting* 38 (4): 1325–36. https://doi.org/https://doi.org/10.1016/j.ijforecast.2021.07.007.