Product Sales Forecasting for Local Retail Store

Isaac Jeon
MS Computer Science
University of Colorado Boulder
Boulder, CO
isje4596@colorado.edu

Analysis and prediction of product sales provide valuable information that businesses can use in gauging product demand and restocking of items. This project makes use of a dataset of transactions for a single small clothing retail store while incorporating weather data to analyze and predict customer foot traffic. Deep learning methods such as LSTMs may struggle for smaller datasets. Fortunately, statistical methods such as exponential smoothing and ARMA (Autoregressive Moving Average) models as well as machine learning algorithms like LightGBM may be viable for small to medium-sized datasets. In particular, extensive feature engineering such as the addition of holiday effects and lag features informed by a combination of exploratory data analysis and domain knowledge may compensate for the smaller amount of available data.

Models were built using Holt-Winters, SARIMAX, LightGBM, and Prophet. Evaluating on 13 test folds of 28 days each, the Prophet model performed the best with an average RMSE of 3.6152, while LightGBM and SARIMAX performed similarly to each other at 3.7016 and 3.7024 respectively, followed by an ensemble average of two Holt-Winters models using different seasonal periods at an RMSE of 3.7633. Model performance seemed to largely depend on how well it can capture seasonal patterns. However, different models performed better in different folds, raising the possibility that certain models may excel over others under different contexts. Ensembling the four models by taking the average of their predictions stabilizes these forecasts and results in improved performance with an RMSE of 3.5843. These errors are too high for practical use, highlighting the limitations that arise from the high variability at a granular level, especially for smaller datasets. However, forecasting at broader intervals such as at a weekly level may still be viable for real-world application.

1 Introduction

Retail sales forecasting is a common problem of interest for retailers, which can be utilized for predicting demand of certain products as well as for inventory management by taking into account potential future sales. While these benefits apply to large retailers like Walmart and Target for maximizing sales, small “mom and pop” stores with much lower employee counts can also see immense gains from such solutions, arguably to an even greater extent as the risks made from over or underestimating future sales can be proportionally more costly. However, there are additional challenges that arise in this setting: the quality and quantity of data. Many local businesses are unlikely to have vast amounts of clean and rich data, and any usable data would be a result of simple bookkeeping. On the other hand, working at a smaller scale can allow for easier understanding of the ins and outs of that particular store, which can aid in feature engineering that may compensate for the relative lack of data.

For this project, I will utilize data containing a small individually-owned clothing retailer’s history of sales transactions to analyze and predict 28-day forecasts of daily sales for a particular product. In particular, I analyze correlations, trends, and seasonality in the data to aid in feature engineering as well as my choices for modeling. I then build models using four approaches: **Holt-Winters**, **SARIMAX**, **LightGBM**, and **Prophet**, as well as an ensemble of these models. Finally, I evaluate the results with expanding window cross-validation on a number of metrics.

2 Related Works

The ***Corporación Favorita Grocery Sales Forecasting* Kaggle competition** provides data from their grocery stores [1]. While the scale of the dataset is much larger and the genre of products is different (as opposed to my clothing dataset), we can get a general idea of which techniques may or may not work well. Ganguly et al. [2] found that a Linear Regression method fell short due to the grocery dataset’s complexity, and that machine learning algorithms like Random Forest, Gradient Boosting, Support Vector Regression, and XGBoost performing comparatively better, with Random Forest coming out on top out of these approaches. Other methods include deep learning methods such as LSTMs or CNNs. The winning model of the competition, however, seems to be based on LightGBM (Light Gradient-Boosting Machine). It seems that tree-based models seem to do well for this particular task.

Walmart also provides a similar dataset in the ***M5 Forecasting - Accuracy* Kaggle competition** [3], in which ensembles of LightGBM models were extensively used among the top competitors. However, simpler models such as exponential smoothing (such as Holt-Winters) as well as

ARIMA models were found to be viable especially at a store or product-store level [4].

3 Proposed Work

My project involves the daily sales of a single style of work pants from a specific brand aggregated across different styles and colors for a single store. The goal is to analyze and identify important factors and trends in the data, and forecast sales for up to 28 days into the future.

The dataset was sourced from a local clothing retail store owned and operated by a relative of mine. While this seems like a simplification of the related works above, it differs in its context. First, the dataset is much smaller in scale and likely not as tidy as those from larger corporations. In addition, I have actually worked at this store a number of years ago so I have some domain knowledge that I can leverage for this project. While I am far from an expert in this field, I am fairly familiar with the dataset and what may be required for data cleaning, interpreting the results of analyses, as well as feature engineering. This is somewhat akin to a data analyst taking input from a domain specialist, and by taking both roles I might hope to gain some insight into what each may require from the other.

In addition, I chose a specific product that could, in a sense, be considered representative of the data as a whole, similar to a cheeseburger at McDonald's. This particular pants style is the most basic of its category, is relatively affordable, and is not as affected by seasonality compared to other items like shorts or jackets. It is also least likely to be affected by external factors, or rather, as a "staple" item if it is affected by external factors, other items are likely to be similarly affected (this isn't completely true, as I'll briefly touch up later). Therefore, analysis on this specific product can be a good baseline in judging the feasibility of future projects involving this store's dataset, as well as be a starting point for extending to other problems such as analysis of seasonally-sensitive items or forecasting on a different level such as forecasting across different products or across different colors and/or sizes of the same product style.

3.1 Dataset

The dataset contains a time series of dates, times, invoice number (i.e. the ID number of a single transaction), description of a unique product, quantity of the item purchased, price, and sales status. The product description is a string that contains the product name, and if applicable, color and/or size. Sale status indicates whether the item was sold, discounted, returned (exchanged or refunded), or voided (cancelled, usually in cases of error on the cashier's part), which is necessary for adjusting prices or completely removing certain rows. There are other irrelevant or unnecessary columns that I omitted. Each row involves a single item purchased in a specific transaction, meaning that a transaction with multiple items will be represented by multiple rows (with the same date, time, and invoice number).

Data starts from January 2, 2018 and ends on June 1, 2025. Data exists for prior years but the store moved location at some point resulting in a number of changes during its transition, so I chose to start from a year that better reflects the store in its current state.

3.2 Data Preprocessing and Cleaning

Before anything, I needed to process the data in the proper format for my problem, specifically the daily sales counts for the product. I also included the base price of the product (price goes up at certain size thresholds) which could be used to identify price increases. Basically, I filtered for sales of this specific product based on product description, ignoring color and size, and summed the quantities sold for each day.

While this seems straightforward, there were a number of issues I addressed. The few notable issues were: Single transactions of a large quantity of the item, non-standard sales statuses (voided, discounted, returns), and dates with no sales of the product.

The first issue just involved removing outliers above a certain threshold as they would skew the data if left alone. These are often large purchases from local businesses or organizations and in reality would be handled differently from regular transactions. For the second issue, voided sales were simply removed, while discounted sales had prices reverted to their original retail price. Returns were handled more carefully.

The third issue was a bit more complicated. There are three possibilities for days with no sales: it was a normal business day that just didn't have any sales of this particular product, it was a holiday that the store was closed for (i.e. Christmas, New Year's Day), or the store was closed due to unforeseen events (in particular, a lockdown during the Covid-19 pandemic). The first two cases are simple, as we simply create rows for the dates with zero counts. For the closed holidays, I added a column indicating whether or not the store was a regularly-scheduled closed day.

As for the third case, it would be hard to catch single days where they suddenly needed to close (unless the store kept it on record somewhere), although even then it may not be a big issue to ignore these instances. However, during the COVID-19 pandemic, California implemented a "stay at home" order, requiring non-essential services to close. This forced the store to close from March 20 to May 7, 2020, until May 8 when the lockdown was partially lifted for certain businesses including clothing stores. As a result, there are 7 consecutive weeks of missing data, which will be a problem for time series models. While some algorithms like LightGBM do not necessarily require data for all dates, there will be issues when adding lag and rolling window features, which rely on values from past instances. As such, rather than assigning zero counts or completely ignoring these instances, it would be better to impute target values based on other values from the dataset.

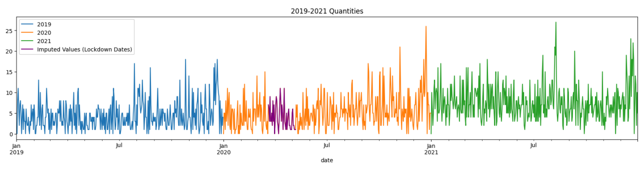


Fig. 1 Plot quantities in 2019-2021 after imputing missing values (purple) during the 2020 lockdown.

For the lockdown period, I imputed the median of the target values from 357, 364, and 371 days prior, multiplied by the ratio of the sum of values from 28 days (4 weeks) before and after the lockdown period to that of the corresponding days (364 days prior) from 2019, and rounded to the nearest integer. Essentially, for each of these dates I chose a value based on “similar” past dates (same day of week, roughly one year prior), and multiplied by a “yearly effect” value to try to take into account the change in trend from the prior to current year.

3.3 Exploratory Data Analysis

The data (Fig. 2) shows a right-skewed distribution with a mean of roughly 5.91 slightly above the median of 5. A relatively high standard deviation of 4.39 compared to the mean suggests notable variability. Values range from 0 to 36, although with a 75th percentile of 8. Certain models assume normality, so applying a transformation to the target may improve performance.

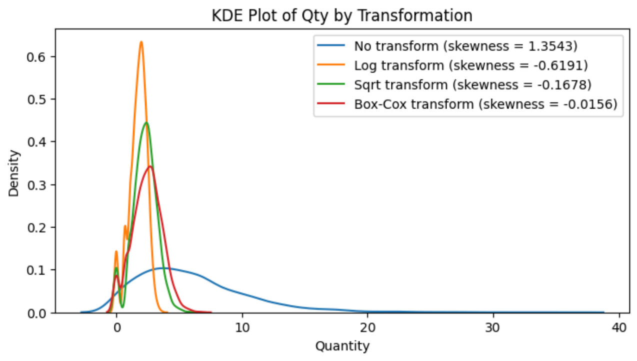


Fig. 2 KDE plot of target (blue), as well as of log, square-root, and Box-Cox transformations of the target. Skewness measures the asymmetry of the distribution, with values closer to 0 indicating more symmetry.

In particular, the **Box-Cox** transformation results in a fairly normal distribution with a skewness value of -0.0156, indicating a slightly heavier left-tail likely due to the counts being lower-bounded on 0. Box-Cox, along with the log transformation, only takes positive values. Since the counts can be zero, we would add a constant to each target value (adding 1 is simplest, but may be another value that can be tuned), and revert the transformation and subtract the constant from resulting predictions. The statsmodels Holt-Winters method includes a parameter for applying the Box-Cox transformation, although it may need to be manually applied for other methods such as SARIMAX.

3.3.1 Autocorrelation and Partial Autocorrelation

Autocorrelation measures the correlation of a time series with a delayed version of itself. For instance, autocorrelation of a daily time series with 3 lags would be the correlation of each day’s value with the value from 3 days earlier respectively. **Partial autocorrelation** is similar but removes the influence of earlier lags, thus representing the direct effect of the lag. Continuing the previous example, the ACF at lag 3 would be influenced by correlations at lags 1 and 2, but PACF removes those effects and gives an isolated measure of the correlation between the current and 3-day lagged values.

The **ACF and PACF plots** (Fig. 3) can be used to identify the **AutoRegressive (AR)** and **Moving Average (MA)** terms for the SARIMAX model (the p and q parameters, respectively), as well as seasonality. As the number of lags increases, we would expect correlation to decrease (we would expect higher correlation with closer days rather than further days). Values may also be more correlated with certain lags due to other reasons. If we identify patterns in the plots, this could indicate seasonality.

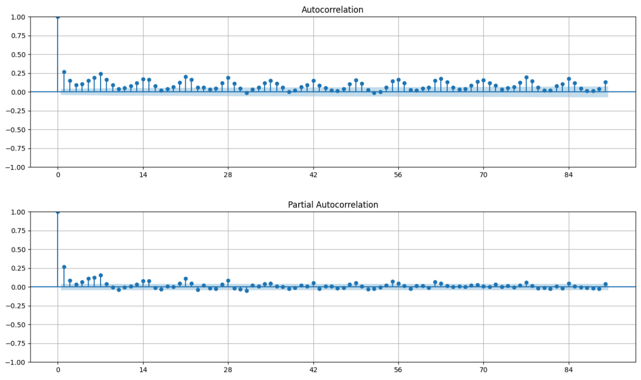


Fig. 3.1 ACF (Autocorrelation Function) and PACF (Partial Autocorrelation Function) plots with 90 lags. Values outside the light blue band are considered statistically significant.

From the plots, we see that ACF decays slowly, while PACF decreases relatively quicker. This suggests an autoregressive process, and we might try fitting the model with p = 1 or 2. This parameter along with q will be tuned anyways, but this gives us an idea of which values to try first.

There are also peaks every 7 lags, indicating **weekly seasonality**. Empirically, it's expected that the sales count on a Saturday (usually the busiest day of the week) would be closer to that from the previous Saturday, rather than with the sales count 3 days before on Wednesday (one of the slower days of the week). Therefore, it may be best to model with the assumption of weekly seasonality.

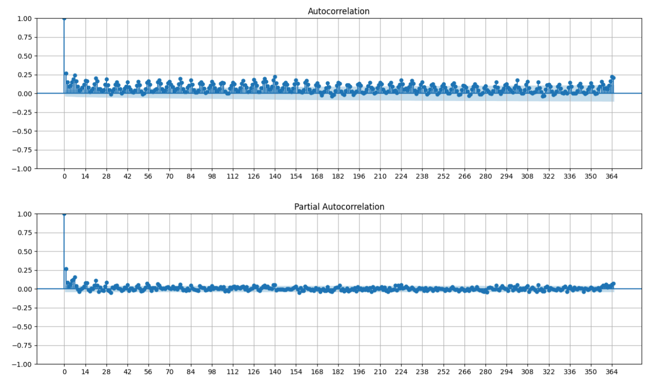


Fig. 3.2 ACF and PACF plots with 365 lags.

However, an ACF plot with 365 lags shows a sudden spike at 364-365 lags, while the corresponding PACF plot shows a lesser increase but with values being statistically significant (outside the light blue band). This suggests an additional yearly seasonality, which complicates the situation for Holt-Winters and SARIMAX which only assume one seasonal period.

3.3.2 Stationarity

A time series is **stationary** if its statistical properties such as mean and variance do not depend on time. SARIMAX assumes stationarity, although this is not a requirement for the other three approaches in this project. The **Augmented Dickey-Fuller (ADF) test** can be used to check for stationarity. Applying the test to the data results in a p-value lower than 0.05, so we can reject the null hypothesis of non-stationarity and conclude that the time series is stationary.

3.3.3 Seasonal and Trend Analysis with STL Decomposition

STL decomposition (Seasonal-Trend decomposition using LOESS) decomposes a time series into its trend, seasonal, and residual (noise) components. The sum of these three components results in the original time series.

If we decompose and plot the comSTponents based on a 7-day period, we can make out a weekly up-and-down pattern (Fig. 4.1). Generally, sales are higher on the weekends and lower on the weekdays. However, the degree to which sales increase or decrease each week varies. From experience, this store usually sees high numbers of sales during the days leading up to Christmas, as well as the period between the end of July to early August (which can be referred to as the "back-to-school" shopping season). This can be seen in spikes in the trend and seasonal plots, as sales of this product (as well as a number of others) may significantly increase during those periods.

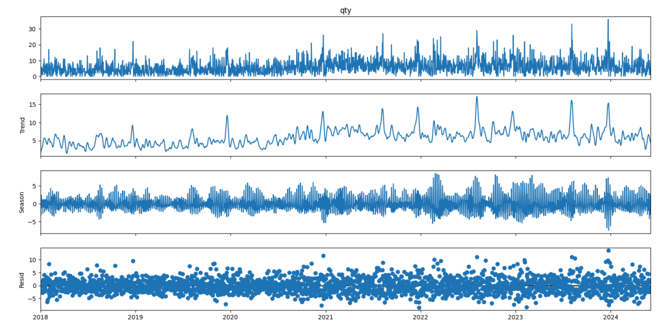


Fig. 4.1 Plots of trend, seasonal, and residual components from STL decomposition with 7-day periods

The residual plot shows that the residuals are clumped around a mean of 0, which is a good sign. However, there seems to be some more spread over time, beginning around the middle of 2020 which happens to mark the started of an "era" of uncertainty characterized by a number of atypical circumstances including manufacturing issues due to the pandemic, changes in fashion (to the benefit of the store), and a social media trend temporarily bringing in an unexpected demographic.

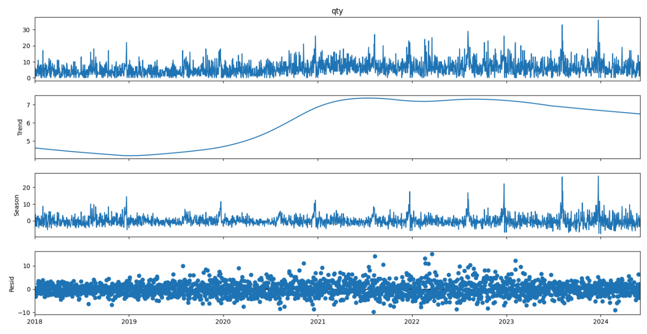


Fig. 4.2 Plots of components from STL decomposition with 365-day periods

If we decompose the data with 365-day periods, we see a much smoother trend curve, as opposed to the noisy trend curve from the weekly decomposition. There is a significant increase in trend between 2020 and 2021, despite the pandemic. One reason for this is that due to difficulties in manufacturing during the pandemic, manufacturers prioritized the production of the most standard and/or popular products, which includes this product. As a result, this style of pants was not as negatively affected by shortages (at least for the basic color options). On the contrary, sales were likely proportionally higher due to the lower availability of other options at that time. The trend plateaus for about 2 years before steadily declining.

In the yearly seasonality plot, there is a lot of noise (since the data is at a daily level) but we do see spikes around August and December each year, coinciding with the previously mentioned back-to-school and Christmas shopping seasons. There also seems to be slightly higher values around March-April and October-November.

A benefit to decomposing the time series into its components is that we can separately model each component. In particular, with nested seasonalities such as in this case where we have yearly seasonality on top of weekly seasonality, for which we can remove one of the seasonal components. For instance, we can subtract the yearly seasonal component and further decompose the remaining trend and residual parts with weekly periods.

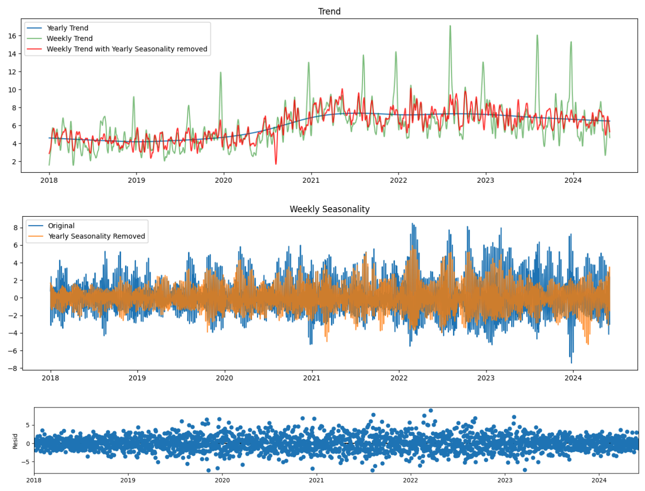


Fig. 4.3 Plots of components from STL decomposition with 7-day periods on time series with 365-day seasonal component removed compared with the same decomposition on the original data.

Decomposing the time series after removing the yearly seasonal component, we do see a reduction in noise across all three components. The trend fits the smooth yearly trend curve more closely, the variance in weekly seasonally is somewhat diminished, and the residuals in the earlier and later years are less spread out. Doing this the other way around by first removing weekly seasonality results in similar reductions in noise. Of course, there still seems to be unexplained factors resulting in the observed noise, but it's clear that both weekly and yearly seasonality.

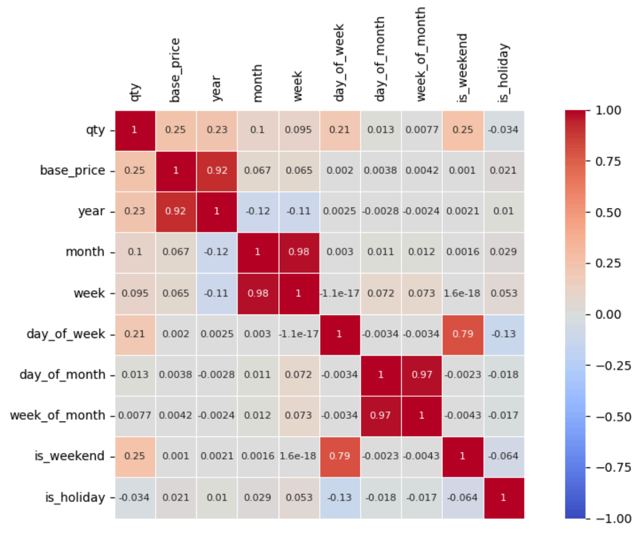
To summarize the potential issues, the data has nested (both weekly and yearly) seasonality, nonlinear trend and seasonality, and additional external factors that cannot be explained by trend or seasonality. The first two issues are problematic for Holt-Winters and SARIMAX but are less of an issue for LightGBM and Prophet, while the third issue is more complex. There are multiple possible approaches, but ultimately the performance of each model will likely depend on the flexibility of the method as well as the engineered features (if applicable).

3.4 Feature Engineering

LightGBM is not a time series model at its core and heavily relies on the engineered features to learn the temporal structure of the data. Holt-Winters aside, SARIMAX and Prophet may also take extra features in the form of exogenous regressors and/or holiday effects. Consequently, feature engineering will be the deciding factor in the success of these models, at least for those that can make use of those features.

3.4.1 Features based on dates

In addition to the product's base price, I added features based on the date. I added numeric/binary features for *year*, *month*, *week*, *day_of_week*, *day_of_month*, *week_of_month*, *is_weekend* (Saturday and Sunday), *is_holiday* (from the python *holidays* package), and *closed_day* (days when the store is closed). Most of these are self-explanatory. The *day_of_week* feature is an integer from 0 to 6 representing Monday through Sunday respectively. The *week* feature is the week number of the year based on the ISO calendar system, in which weeks start on Monday and are numbered from 1 to 52 or 53.



	qty	base_price	year	month	week	day_of_week	day_of_month	week_of_month	is_weekend	is_holiday
qty	1	0.25	0.23	0.1	0.095	0.21	0.013	0.0077	0.25	-0.034
base_price	0.25	1	0.92	0.067	0.065	0.002	0.0038	0.0042	0.001	0.021
year	0.23	0.92	1	-0.12	-0.11	0.0025	-0.0028	-0.0024	0.0021	0.01
month	0.1	0.067	-0.12	1	0.98	0.003	0.011	0.012	0.0016	0.029
week	0.095	0.065	-0.11	0.98	1	-1.1e-17	0.072	0.073	1.6e-18	0.053
day_of_week	0.21	0.002	0.0025	0.003	-1.1e-17	1	-0.0034	-0.0034	0.79	-0.13
day_of_month	0.013	0.0038	-0.0028	0.011	0.072	-0.0034	1	0.97	-0.0023	-0.018
week_of_month	0.0077	0.0042	-0.0024	0.012	0.073	-0.0034	0.97	1	-0.0043	-0.017
is_weekend	0.25	0.001	0.0021	0.0016	1.6e-18	0.79	-0.0023	-0.0043	1	-0.064
is_holiday	-0.034	0.021	0.01	0.029	0.053	-0.13	-0.018	-0.017	-0.064	1

Fig. 5 Correlation matrix of initial features. Excludes *closed_day*.

Looking at the correlation matrix of these features with the target and each other, we see that *is_weekend*, *day_of_week*, *year*, and *base_price* are fairly positively correlated with the target, and *month* and (ISO) *week* are somewhat positively correlated. *Is_holiday* seems to be slightly negatively correlated with the target, although this is likely due to the inclusion of holidays like Christmas during which the store is always closed and are guaranteed to have zero values for the target.

There is **multicollinearity** which can be seen from the correlations between some of the features, which is not a surprise given that they are mostly taken from the date. Nonlinear tree-based methods like Random Forest and LightGBM are fairly robust to multicollinearity and implicitly select features, but methods with linear regressors like SARIMAX and Prophet are more sensitive to this issue and we would need to be more selective of the regressors to include. This includes the additional features that I will be adding next.

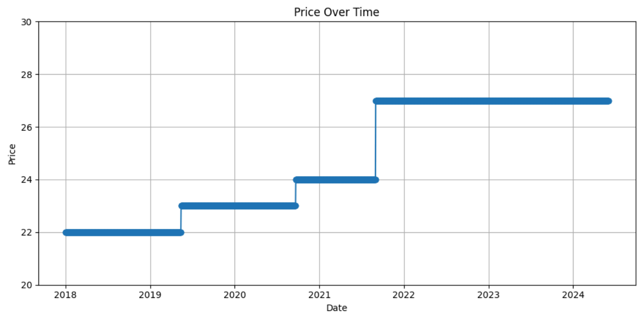


Fig. 6 Plot of (base retail) price over time.

There is a high correlation between *base_price* and *year*. The reason for this is the *base_price* only increases over time and fairly infrequently. In fact, the plot of the price over time (Fig. 6) shows a pattern that is similar to the yearly trend (Fig. 4.2). In addition, the positive correlation between *base_price* and the target suggests that there are actually *more* sales of the product when it costs more. It seems that *base_price* effectively represents trend over time, explaining its high correlation with *year*.

3.4.2 Shopping Seasons

The yearly seasonal component plot (Fig. 4.2) shows yearly spikes at two particular periods. In the retail context, these can be interpreted as the back-to-school (late July to early August) and Christmas shopping seasons. While the product in question are work pants that are primarily worn by adult men, they are also often purchased as schoolwear for highschoolers, explaining the higher demand in the days leading up to the new school year. Similar to the *is_holiday* feature, including separate features for each of these shopping seasons likely improve performance. The issue, however, is how to identify which range of dates to include for each seasonal period.

The Christmas shopping season would include a certain number of days before Christmas. I simply included binary features for 2, 6, 10, and 14 days before Christmas.

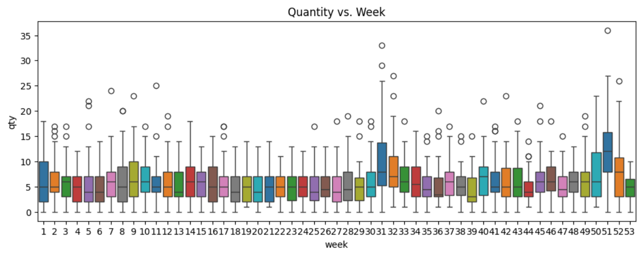


Fig. 7 Box plot of target *qty* vs. (ISO) *week*.

The back-to-school season is a bit more complicated to identify, as the first day of school changes each year and can even differ between school districts. Ideally, we would look up the first day of school each year. However, I went with a simpler route by identifying a plausible range of dates based on prior years. The box plot (Fig. 7) shows generally higher values on week 31 (usually starting in the last couple days of July) and possibly week 32. I also looked up last year's date for the first day of school in the school district in the same city as the store, and it fell on week 33. Therefore, I included a *back_to_school* feature for dates during weeks 31 and 32. Prophet is a bit more flexible with holiday effects so for the Prophet model, I increased the range to include the last 7 days of July to the first 15 days of August.

The back-to-school and Christmas season features are fairly positively correlated with the target based on correlation coefficients, and may help the models capture the seasonal spikes.

3.4.3 Price Change Features

The actual price amount is unlikely to matter for this product (it may matter when dealing with multiple different products). What is probably more useful from the base_price feature are the price increases and the degree to which they increase. In addition, price increases could possibly have delayed effects on the target (maybe a . Therefore, I added price change features based on the ratio of current and a past date's price. The features *price_change_i* use the price from *i* days before. In other words, these features represent the multiplicative increase in price for each day relative to a prior point in time. I added features for 1, 2, 3, 4, 6, and 9 months prior as well as 1 year prior.

The price increases are initially slightly negatively correlated before becoming slightly positively correlated after 4+ months, with a 0.073 correlation for 1 year. These correlations may again be influenced to some extent by correlations with other features such as *month*, *week*, and *year*.

3.4.4 Lag and Rolling Window Features

LightGBM, unlike the other mentioned methods, does not inherently handle sequential or seasonal data. **Lag features** are values from past "lags" (in this case, past days), while **rolling window features** are aggregated values (i.e. mean, standard deviation, max, min, etc.) of a window of past days. For example, a rolling mean of lag 7 and window 3 would be the average of the values from 7, 8, and 9 days prior. The previous price change features are based on lagged values of *base_price*.

It is important not to include lags shorter than the horizon (in this case, the number of days to forecast into the future). For this project, we are aiming to predict sales up to 28 days into the future i.e. a horizon of 28. Lag and rolling window features are based on past target values. If we use a lag less than 28, such as a lag of 7, the 7-lag values would be unknown starting from the 8th day (we can't use tomorrow's true value as a feature for 8 days into the future since tomorrow's sale count hasn't been observed yet). Therefore, lagged features need to be shifted back at least 28 days.

I just added a number of features of different combinations of lags and windows. As these features rely on past data, the earlier dates in the dataset will have null values for these features. I used lags of up to one year, so I dropped the 2018 data from the dataset.

Since LightGBM is generally robust to multicollinearity and implicitly does feature selection to some extent, I fit a LightGBM model with all 108 features, then trimmed the number of features down based on SHAP values to reduce overfitting.

To avoid multicollinearity with SARIMAX, we would want to avoid choosing exogenous regressors that are correlated with other regressors. SARIMAX can only handle one seasonal period at a time (e.g. weekly or yearly). It also has difficulty with longer seasonal periods, so it would probably be better to use 7-day seasonal periods and use regressors to handle the yearly seasonality. Of the lag and rolling window features, I only include *lag_363_mean_7* (mean of values from 363-365 days prior) as an regressor as it has the highest correlation (0.34) with the target of these features and can give the model information that it otherwise wouldn't with weekly periods.

3.5 Methods

I will build models using Holt-Winters, SARIMAX, LightGBM, and Prophet, as well as using an ensemble of the average of their predictions.

The **Holt-Winters** method, also known as **triple exponential smoothing**, applies exponential smoothing three times and breaks the time series down into three components: level, trend, and seasonality. It works well on data with clear trends and seasonality and is quite efficient in terms of speed, but may be overly-simple as it assumes linear trends and seasonality and does not support multiple seasonalities or external regressors, all of which may raise issues with our data.

ARIMA (Autoregressive Integrated Moving Average) models the time series as a linear combination of its past values, past forecast errors, and differenced values. **SARIMAX (Seasonal ARIMA with Exogenous Regressors)** is an extension of ARIMA (which in turn is a generalization of ARMA) that adds support for seasonality and exogenous regressors. Like Holt-Winters, it still assumes linearity and one seasonal period, but the latter issue could be addressed to some extent with regressors. The larger downside is that it can be computationally expensive, making it time-consuming to tune.

LightGBM (Light Gradient-Boosting Machine) is a tree-based machine learning algorithm. It can handle nonlinearities and multiple seasonalities (via feature engineering, and is also relatively fast. But because it is not a time series model in itself, it requires extensive feature engineering, cannot extrapolate trends, and accuracy of forecasts can degrade quickly over long horizons.

Prophet is Meta's open-source forecasting method that is based on an additive model that, according to Meta, fits non-linear trends with multiple seasonalities as well as holiday effects [8]. As such, they can handle the issues previously mentioned with the first two methods. It can also run efficiently especially for small to medium-sized datasets, and has a relatively simple API that makes it easy to use.

While offering more flexibility compared to Holt-Winters and SARIMAX and is probably the best of the four methods for long-term forecasting, Prophet still has its limitations. For instance, it would be more accurate to say that it models *semi-nonlinear* trends through piecewise linear or logistic functions, which still involves some rigidity. However, the STL decomposition (Fig. 4.2) does seem to suggest that this may not be an issue. Also, it does not model autocorrelation unlike SARIMAX (inherently included), LightGBM (through lag features), and to some extent Holt-Winters (implicitly through smoothed past values). In other words, while it models trend, seasonality, and specified holiday effects, it does not assume dependencies with past values (e.g. today's value depends on yesterday's value). One workaround is to add lag features, but this may lead to overfitting with how the model is implemented so I will only add holiday effects for the Prophet model.

I will also ensemble the models using the average of their forecasts to see if it improves performance. There are other options of ensembling such as taking a weighted mean based on metrics or tuning the weights, but for the sake of time I will just use the simple average.

4 Evaluation

4.1 Evaluation Metrics

To evaluate the predictive models, I will use four metrics: RMSE, RMSLE, MAE, and SMAPE.

- $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \widehat{y}_i\right)^2}$
- $RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(log(y_i + 1) - log(\widehat{y}_i + 1)\right)^2}$
- $MAE = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \widehat{y}_i\right|$
- $SMAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|y_i - \widehat{y}_i\right|}{\left(\left|y_i\right| + \left|\widehat{y}_i\right|\right)^2 / 2}$

Mean Absolute Error (MAE) is the average distance between predicted and actual values, and is easy to interpret. **Root Mean Squared Error (RMSE)** is similar but penalizes larger differences. I will minimize RMSE (or maximize the negative MSE when using sklearn's RandomizedSearchCV) during hyperparameter tuning.

Root Mean Squared Logarithmic Error (RMSLE) measures relative error as a ratio and is good for skewed data, as with this right-skewed data. For instance, predicting 10 for 5 and 100 for 50 are both off by 2 times the actual value (technically, they're treated slightly differently since each value is increased by 1 to handle zero values). RMSLE also penalizes underestimations more than overestimations, which is often useful in various business contexts. In the context of product demand, overestimations can result in a surplus of inventory from over-ordering while underestimations can lead to loss of potential sales. If surpluses are less of a risk, we would care more about underestimating demand. In contrast, RMSE treats the size of over- and underestimates the same and punishes larger mistakes, the latter of which can penalize models for bad predictions of outliers which can skew results.

Symmetric Mean Absolute Percentage Error (SMAPE) measures the percentage difference between actual and predicted values, and like RMSLE, is less sensitive to outliers and handles zeroes unlike MAPE (division by 0 occurs only when both actual and predicted values are 0, but can be handled by return 0 or ignoring these points). Compared to RMSLE, SMAPE reports error as a percentage rather than ratio which may be more intuitive, and it penalizes over- and underestimates equally. It is also scale-independent, so it can be used to compare with other time series data of different scales such as other products or the same product's sales at another store. However, while it is more forgiving to mispredictions of large outliers, SMAPE is much more sensitive to small values, especially near 0. For example, if the actual value is 0 and the predicted value is 1 (or vice versa), it would have an error of 200% which may lead to a misleading SMAPE score. At a daily level, it is not rare to sell none of a specific product on some days so we may see some inflated values for SMAPE. At a less granular level such as weekly, no sales for one period of time may be unlikely (depending on the product) which would make SMAPE more of a meaningful metric.

Different models may do better than others under different contexts, and evaluating them with these four metrics can give us insight into how well they perform under different situations.

4.2 Experimental Setup

I split the data into a training set to fit the models and a test set for evaluation. Because it is a time series, I split the data chronologically rather than randomly since the training set should contain data for dates before those in the test set. The forecast horizon is 28 days so in practice, the model would need to be evaluated on at most 28 days after the last day in the training set. It would also be ideal to evaluate the models on at least one year of test data to see how they perform across different periods of the year.

I will use an **expanding window cross-validation** approach for evaluation as well as for hyperparameter tuning. I took the following steps:

1. Use the last 364 days (roughly 1 year) to create 13 folds of 28 (forecast horizon size) consecutive days each.
2. Fit the model with the initial training set and evaluate on the first fold for the dates that chronologically occur directly after the last day in the training set, and record the metrics.
3. Expand the training set window by adding the previous test fold to the training set, and repeat for each consecutive fold.
4. Take the average of each evaluation metric to measure the average performance of the model across the 13 folds.

Forecasting accuracy decreases for days further in the future, so simply evaluating using one-year's worth of data would not be reflective of the actual performance of the model. Assuming the model would be refitted on updated training data for each forecast, expanding window cross-validation would effectively simulate the model's real-world application.

The test set includes the last 364 days in the dataset from June 3, 2024 to June 1, 2025 (which is split into 13 folds for evaluation). In addition, the 364 days from June 5, 2023 to June 2, 2024 are used for cross-validation during hyperparameter tuning.

One final thing to mention is that predictions for closed days (e.g. Christmas) can be ignored because they will always have zero counts, so I will manually set their predictions to 0.

4.3 Results

As a baseline, I made a **naïve forecast** by simply taking the average of the training set values and using it as the predicted value for all forecasts on the test, resulting in RMSE = 4.3680, MAE = 3.3521, RMSLE = 0.8260, and SMAPE = 67.9012 (compare with Fig. 12.1).

4.3.1 Holt-Winters

Since Holt-Winters only handles one type of seasonality at a time, I fit models for both seasonal_periods = 7 (SP=7) and 364 (SP=364). 364 periods performs better than 365 since it lines up with the weekly seasonality due to being divisible by 7 and isn't sensitive to leap years.

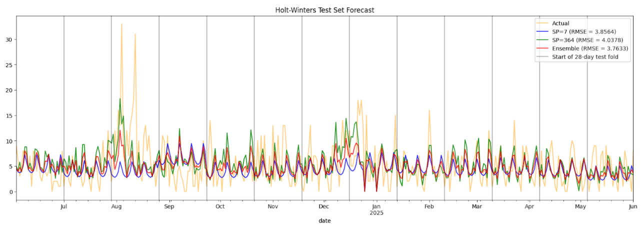


Fig. 8 Forecasts on the 13 test-folds for Holt-Winters with 7 (green) and 364 (red) seasonal_periods and an ensemble average (blue), with associated average RMSEs across folds included in the legend.

The SP=7 model has a lower RMSE of 3.8594 compared to the SP=364 model's RMSE of 4.0378, both of which are an improvement to the naïve forecast RMSE of 4.3580. However, the SP=364 model has a lower RMSE of 3.5439 for fitted values (prediction on training set) compared to the SP=7 model's fitted RMSE of 4.1436. SP=364 is more flexible in capturing spikes in the data but overfits on the training data, while SP=7 makes smaller errors but has less variability in values since Holt-Winters assumes constant seasonal patterns.

An ensemble model that averages the predictions of the SP=7 and SP=364 models results in a more balanced forecast with a lower RMSE of 3.7633, as it keeps some of the flexibility of the SP=364 predictions while being kept in check by the SP=7 predictions (Fig. 9).

4.3.2 SARIMAX

For SARIMAX, I fit one model without exogenous regressors (SARIMA) and another with regressors. Both used 7 periods for seasonality. I chose three regressors for the SARIMAX model: *back_to_school*, *6d_to_christmas*, and *lag_363_mean_3* (the mean target value for 363-365 days prior). To capture yearly seasonality, I chose these three regressors that had relatively high correlation coefficients with the target. I limited it to these three to avoid multicollinearity. While it is possible to try different combinations of regressors, SARIMAX takes substantially longer to fit which makes tuning very time-consuming.

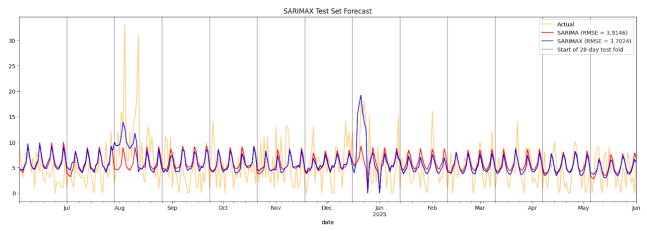


Fig. 9 Forecasts on the 13 test-folds for SARIMA (red) and SARIMAX (blue).

SARIMA suffers the similar issues to Holt-Winters with 7 seasonal periods, and fails to capture yearly seasonal spikes. External regressors add extra context to the model for certain seasonal periods as well as the previous year's values, leading to an improved SARIMAX model with RMSE of 3.7024, slightly beating out the Holt-Winters ensemble. While the predictions still have less variation than the Holt-Winters ensemble, it can also be said that the predictions are more stable and less prone to many large errors.

4.3.3 LightGBM

I first fit a LightGBM model using all 108 features, resulting in an RMSE of 3.7024, basically exactly the same as SARIMAX. There are some differences, as LightGBM does significantly better on the fitted values which suggests overfitting for LightGBM but possibly less underfitting compared to SARIMAX, while SARIMAX does slightly better in the other metrics like MAE and SMAPE.

While feature selection isn't as important for tree-based methods, which choose the optimal feature at each fold, dimensionality reduction can still result in performance improvements by potentially reducing overfitting and increasing speeds.

LightGBM does provide **feature importances** to measure the importance of each feature to the model. By default, these values are based on the number of times each feature was used to split the data but this can be misleading for features that might be important for a minority of instances (e.g. days leading up to Christmas). Instead, I perform feature selection by computing **SHAP values** as a measure of feature importance and removing features with SHAP values under a certain threshold.

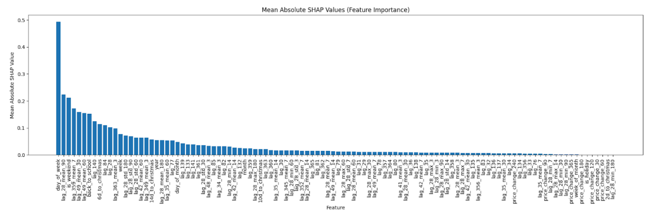


Fig. 10 Mean absolute SHAP values over 13 validation-folds for LightGBM with 108 features.

By looking at the SHAP values, we can get a sense of which features are influential for the model, and which are unimportant. Interestingly, all of the price change features have very low values suggesting that price changes barely if at all affect the number of sales for this particular product. Meanwhile, as expected there are a number of seasonal features at the upper end of the spectrum including *day_of_week*, *back_to_school*, and *6d_to_christmas*, along with a number of lag and rolling window features.

I tuned the threshold for removing features by fitting and evaluating models through cross-validation, which resulted in an optimal threshold of 0.7. This resulted in a large reduction from 108 to just 14 features (the leftmost 14 features in Fig. 10). In addition to only requiring a fraction of the original feature set, this reduction resulted in a very slight improvement of RMSE down to 3.7016, but more importantly reducing the MAE, RMSLE, and sMAPE scores below those of the SARIMAX models. While the actual differences in accuracy might be negligible between the SARIMAX and reduced LightGBM models, the LightGBM model runs significantly faster (discussed in Section 5).

4.3.4 Prophet

The Prophet model had improved scores for each of the four metrics, including an RMSE of 3.6152. Note that this was with holiday effects but without regressors including no lag features. Prophet also has a function to easily plot the trend, holiday, as well as the weekly and yearly seasonal components (Fig. 11).

The trend, despite being piecewise linear, is similar to that of the STL decomposition (Fig. 4.2). We also see strong holiday effects for back-to-school and the leadup to Christmas, and possibly important effects for other holidays. We also get an expected pattern for weekly seasonality, but we also get an interesting smoothed curve for yearly seasonality. There are peaks around August and December, possibly effects that weren't fully captured by the holiday component. However, we also seem to see patterns for other months, including smaller peaks around March-April and October-November (as was somewhat observed in Fig. 4.2) and dips around February and May.

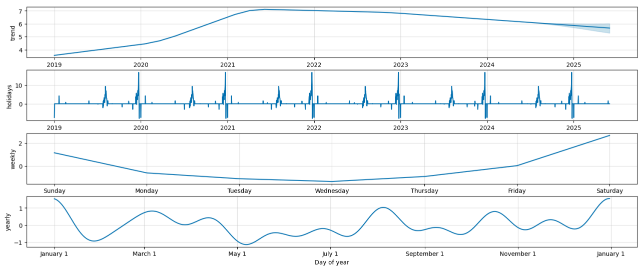


Fig. 11 Prophet trend, holidays, weekly, and yearly component plots

Despite not utilizing an autoregressive approach, Prophet is able to model trends and seasonalities in a flexible manner allowing it to discover patterns that weren't as easily identified through other methods, likely being a factor to its better performance.

4.3.5 Ensemble Average and Model Comparisons

An ensemble model was built by simply taking the average of the best models for each of the four methods, and resulted in lower RMSE, MAE, and SMAPE, and only being slightly beaten out by Prophet in RMSLE.

Metric	Holt-Winters	SARIMAX	LightGBM	Prophet	Ensemble
RMSE	3.7633	3.7024	3.7016	3.6152	3.5843
MAE	2.9712	2.9481	2.9455	2.8855	2.8705
RMSLE	0.7043	0.7243	0.7198	0.6981	0.7020
SMAPE	65.9368	63.9775	63.6657	64.0489	63.4908

Fig. 12.1 RMSE, MAE, RMSLE, and SMAPE averages across 13 test-folds for the Holt-Winters (ensemble average of 7 and 364 seasonal periods), SARIMAX with three exogenous regressors, LightGBM with 14 features, Prophet with holiday effects (no regressors), and an ensemble of the average of the four models' predictions.

When comparing the average of metrics across folds (Fig. 13.1), Prophet seems to be the best single method for RMSE, MAE, and RMSLE, while LightGBM does somewhat better in SMAPE. Holt-Winters, despite having lower RMSE, MAE, and SMAPE, has a comparable RMSLE to Prophet.

In terms of speed, the Holt-Winter model took approximately 0.5 seconds and 1.24 seconds with seasonal_periods = 7 and 364 respectively. A tuned Prophet model took 0.38 seconds, although a Prophet model with default parameters took significantly longer at 4.0 seconds. LightGBM with 14 features took just 0.04 seconds to fit, but even with 108 features it required 0.34 seconds which is still faster than the other methods. However, it does take a considerable amount of time to compute the SHAP values of the 108 features. The runtime for single fit of the SARIMAX model was 25.75 (14.04 seconds without regressors), making it by far the most computationally expensive of these approaches.

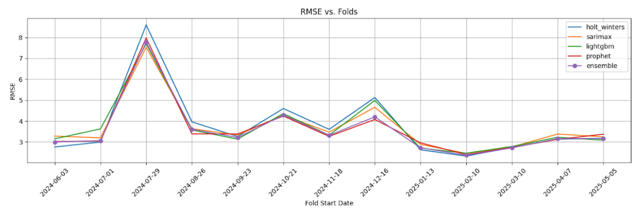


Fig. 12.2 RMSE of each test-fold for each model.

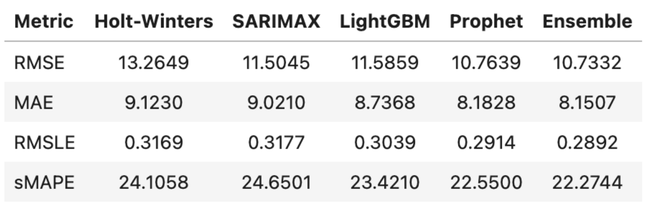
When comparing performance for each individual fold, there is no clear best model across all folds. It is possible that each model performs better under certain circumstances. For instance, despite having slightly worse average metrics compared to LightGBM and running much slower, it has the lowest RMSE during the fold containing the back-to-school dates. Holt-Winter, despite having the worst average RMSE, actually has the best RMSE for certain folds although struggles for other folds. These folds seem to correspond with yearly seasonality, suggesting that it performs worse with higher seasonal effects but conversely has more potential during stable seasons.

The ensemble is able to balance out the strengths and weaknesses of each model. While it might not always result in the best predictions for specific folds compared to individual models, it is able to make relatively good predictions across the folds.

5 Discussion

One major change that I made to this project was to change the problem itself and nearly start over from scratch. I originally tried to forecast customer traffic at a sub-hourly level, but it was a complex problem. Due to higher variance at finer granularity, Random Forest, LightGBM, and even Prophet models were making predictions that were basically predicting a smooth curve. This may be fine for predicting averages but the models did not even attempt to capture spikes and dips in the data, defeating the purpose of the project.

Changing the scope of the problem to a daily level gave more meaningful results, but still suffers from this issue. The reality is that even with daily rather than sub-hourly forecasts, the prediction errors are still too high. Even with a MAE of 2.87, most values are in the single digits (with a median of 5) so an actual value of 5 can easily be mispredicted as 2 or 8, which is reflected by the generally high SMAPE scores. This is due to the high variability of sales at a daily level. In practice, however, predicting daily sales may be overkill. Often, it may suffice to forecast at one- or even two-week intervals, which would also remove the need to model weekly seasonality.



Metric	Holt-Winters	SARIMAX	LightGBM	Prophet	Ensemble
RMSE	13.2649	11.5045	11.5859	10.7639	10.7332
MAE	9.1230	9.0210	8.7368	8.1828	8.1507
RMSLE	0.3169	0.3177	0.3039	0.2914	0.2892
sMAPE	24.1058	24.6501	23.4210	22.5500	22.2744

Fig. 13 Average metrics across test-folds of weekly sums of forecasts

If we were to group the actual and predicted values by week to predict weekly counts, we get results that may be more applicable for a real-world situation (such as restocking the product based on forecasted demand for the following week). For instance, the ensemble predictions are off by 8.15 on average at a 7-day level, compared to 2.87 at a 1-day level. It's also possible to get better weekly results through other ensemble methods such as through stacking by training another meta on the daily forecasted values.

Another challenge was dealing with unclear seasonal trends. I was able to identify the back-to-school and Christmas shopping seasons, but it was less clear as to how many days to include for those seasons. In addition there were other possible seasons with lower yet significant seasonal effects that weren't as easily identified. The Prophet model may have done better since it was able to automate this process to an extent. More experimentation with feature engineering may be able to improve results.

In addition to ensembling methods, more extensive feature engineering, and further hyperparameter tuning, there are other approaches that could be taken. TBATs is a statistical model that uses exponential smoothing like Holt-Winters, but can also deal with multiple seasonalities. In addition, like SARIMAX, incorporates ARMA (AutoRegressive Moving Average) terms among other extensions. As such, it can be better for handling complex time series data. There are also deep learning methods such as LSTMs (Long Short-Term Memory), CNNs (Convolutional Neural Networks), as well as NeuralProphet (an extension to Prophet utilizing neural networks) have been shown to be viable especially for capture complexities in data, although these models may not perform as well on smaller datasets.

6 Conclusion

Generally, the Prophet model gave the most accurate forecasts, while LightGBM was the second-best in terms of accuracy but was substantially faster than the other methods. SARIMAX was comparable to LightGBM in terms of accuracy but much slower. Holt-Winters was fast but was also the least accurate overall. At a per-fold level, however, each model had the potential to do better (or worse) than the others. As a result, the ensemble average returned somewhat stabilized predictions leading to slightly higher accuracy. Further feature engineering, exploring other methods that can deal with complex data, and experimentation with different ensembling methods may lead to further performance improvements.

While the evaluations of the daily forecasts were disappointing, results showed more promise for lower levels of granularity which is sufficient for many real-world applications. Furthermore, this project shows that even a dataset much smaller and limited in information than those of large corporations may still be viable for trend and seasonality analysis as well as forecasting, especially when domain knowledge is applied during feature engineering. While daily forecasts on a single product's sales may not be practical especially for smaller datasets, the problem and approaches used can be extended to lower granularities (as just mentioned) or to different product levels (different sizes/colors, between products, by categories, etc.). The results of this project show that there is potential to create business solutions that small local retailers may seek to improve their operations.

REFERENCES

[1] Data Science Practice: Sales Forecasting. Corporación Favorita Grocery Sales Forecasting Discussion, Kaggle, 2018. https://www.kaggle.com/competitions/favorita-grocery-sales-forecasting/discussion/47582

[2] Priyam Ganguly and Isha Mukherjee. 2024. Enhancing Retail Sales Forecasting with Optimized Machine Learning Models. arXiv preprint arXiv:2410.13773.

[3] Kaggle. 2020. *M5 Forecasting - Accuracy*. Retrieved June 14, 2025 from https://www.kaggle.com/c/m5-forecasting-accuracy/overview

[4] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. 2022. The M5 accuracy competition: Results, findings, and conclusions. Decision Support Systems 142 (2021), 113467. DOI:https://doi.org/10.1016/j.dss.2020.113467

[5] M5 Competition. 2020. *Code of Winning Methods: A1 - Preprocessing*. GitHub. Retrieved June 14, 2025 from https://github.com/Mcompetitions/M5-methods/blob/master/Code%20of%20Winning%20Methods/A1/3.%20code/1.%20preprocessing/1.%20preprocessing.ipynb

[6] Statsmodels Developers. 2025. Statsmodels User Guide. Retrieved June 14, 2025 from https://www.statsmodels.org/stable/user-guide.html

[7] Microsoft. 2025. LightGBM Documentation. Retrieved June 14, 2025 from https://lightgbm.readthedocs.io/en/stable/

[8] Facebook Research. 2025. Prophet: Forecasting at Scale. Retrieved June 14, 2025 from https://facebook.github.io/prophet/

[9] Rob J Hyndman and George Athanasopoulos. 2018. *Forecasting: Principles and Practice* (2nd ed.). OTexts. Retrieved June 14, 2025 from https://otexts.com/fpp2/