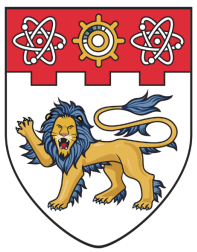

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4041 Machine Learning

Group 57: Project Report
Kaggle, Store Item Demand Forecasting Challenge

Tan Yik Rong Regan (U2020019H)
Riley Ang Xile (U2022075E)
Tan Zuu-Yuaan@Victor Tan (U2021863J)

Contents

1. Team Structure

Name	Role/Contribution
Victor Tan	Exploratory Data Analysis, Data Cleaning, Feature Engineering, Statistical Tests, Project Report and Video
Regan Tan	Machine Learning Models , Project Report and Video
Riley Ang	Time Series Models, Project Report and Video

2. Problem Statement

Time series forecasting holds significant applications in domains such as economics, financial markets, and business analytics. As such, we chose a time series sales forecasting problem for our course project. The objective of this machine learning project is to forecast sales over a 90 day period for 50 distinct items at 10 different stores. The dataset given encompassed time series sales data for these stores and items spanning a 5 year period [1]. Our programming language of choice for this project is Python owing to their extensive range of libraries and frameworks specifically tailored for data science tasks.

3. Exploratory Data Analysis

We start our workflow by conducting basic exploratory data analysis (EDA). In order to ensure an impartial statistical analysis, we first split the given data into training and validation sets and only conduct EDA on the training set. This prevents the introduction of biases during our EDA. Considering our forecasting horizon spans a duration of 90 days, we have to ensure our validation set extends for a period at least as long as the intended forecast window. Consequently, we divide the given dataset into a training set spanning from January 1, 2013, to September 31, 2017, while the validation set encompasses the subsequent period from October 1, 2017, to December 31, 2017.

3.1 Data Representation

A time series inherently exhibits a temporal relationship between successive rows. However, the provided dataset includes columns for store and item, with each row denoting daily sales for a specific store-item combination. Thus, there could exist multiple rows for the same date. In order to conduct a meaningful analysis, we must treat each store-item combination as a unique item or aggregate the data, store-wise or item-wise to preserve the temporal relationship in their time series.

3.2 Store and Item Dependencies

To analyze how sales vary across different stores and items, we first aggregate total sales across stores for each item and total sales across items for each store. We then standardize our data across each item or store to remove the effects of scale and variance [2] and plot their sales z-score (Fig. 1).

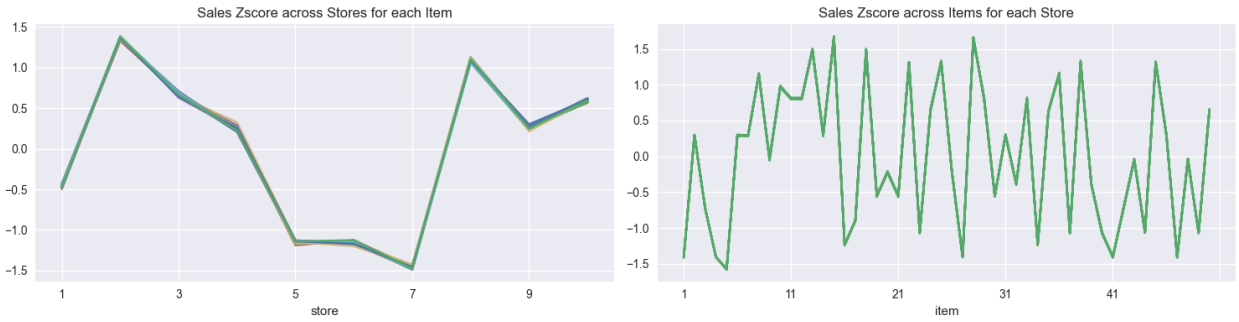

Fig. 1: Store and item total sales z-score

The plots indicate a consistent sales pattern across both stores and items. The trend for items remains consistent regardless of the store they are sold in (Fig. 1 left plot), and similarly, the trend for each store stays uniform across the different items they carry (Fig. 1 right plot). This is given by the fact that the z-score plots for sales of each item and of each store are stacked almost precisely on top of one another. To illustrate, this means that for each item, store 2's sales will always be greater than store 7's sales and for each store, item 15's sales will always be greater than item 5's sales. We can thus conclude that there are no degeneracies in store and item features.

3.2.1 Store Time Series and Histograms

We compute each store's average sales per day and plot their time series (Fig. 2). It becomes apparent that similar structures and seasonal patterns are prevalent. Unsurprisingly, there exists a clear distinction in the overall magnitude of their respective sales. Histogram plots also show that the majority of stores exhibit a right-skewed distribution characterized by fatter right tails which translates to outliers in the higher end of sales (Fig. 3). Notably, each store has different degrees of skewness and variance which results in slightly different histogram shapes.

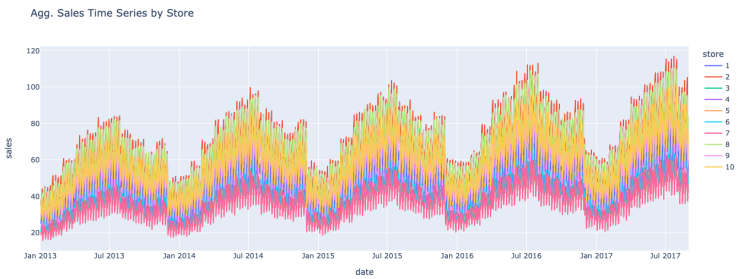

Fig. 2: Time series for each store's mean sales

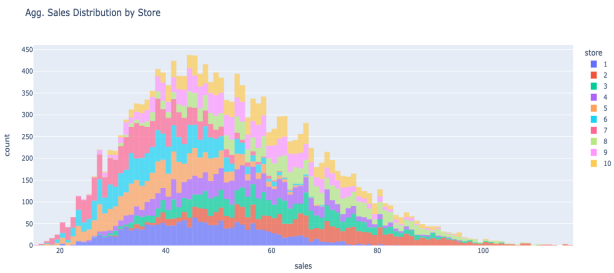

Fig. 3: Histogram plots for each store's mean sales

3.2.1 Item Time Series and Histograms

We plot each item's time series just like we did for stores by computing the average sales per day for each item and find similar structures and periodic patterns also exist for each item's sales time series (Fig. 4). Similarly, we see a right-skewed distribution across all items (Fig. 5).

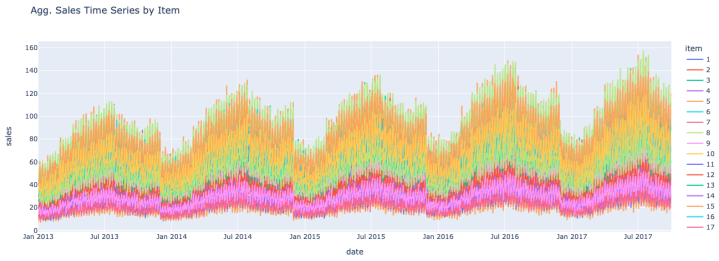

Fig. 4: Time series for each item's mean sales

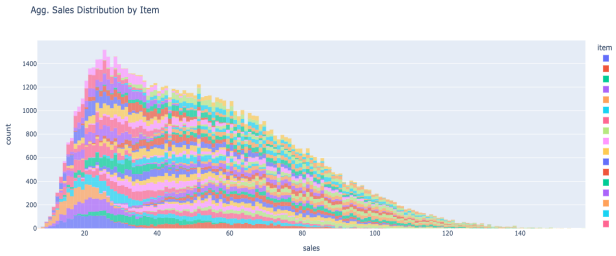

Fig. 5: Histogram plots for each item's mean sales

Section 3.2 thus suggests each store or item has different sales distribution and magnitudes and that incorporating store and item features as covariates when constructing our models might be beneficial. Alternatively, we can simply fit different models for each store-item combination to account for the variations in their respective distributions.

3.3 Temporal Dependencies

Next, we want to isolate the effects of time on sales regardless of store or item. To do this, we add temporal features such as "day of week" and "month" to name a few, and find the average total sales across each of our temporal features for each year. Likewise, we standardize our data and plot sales z-scores across temporal features (Fig. 6).

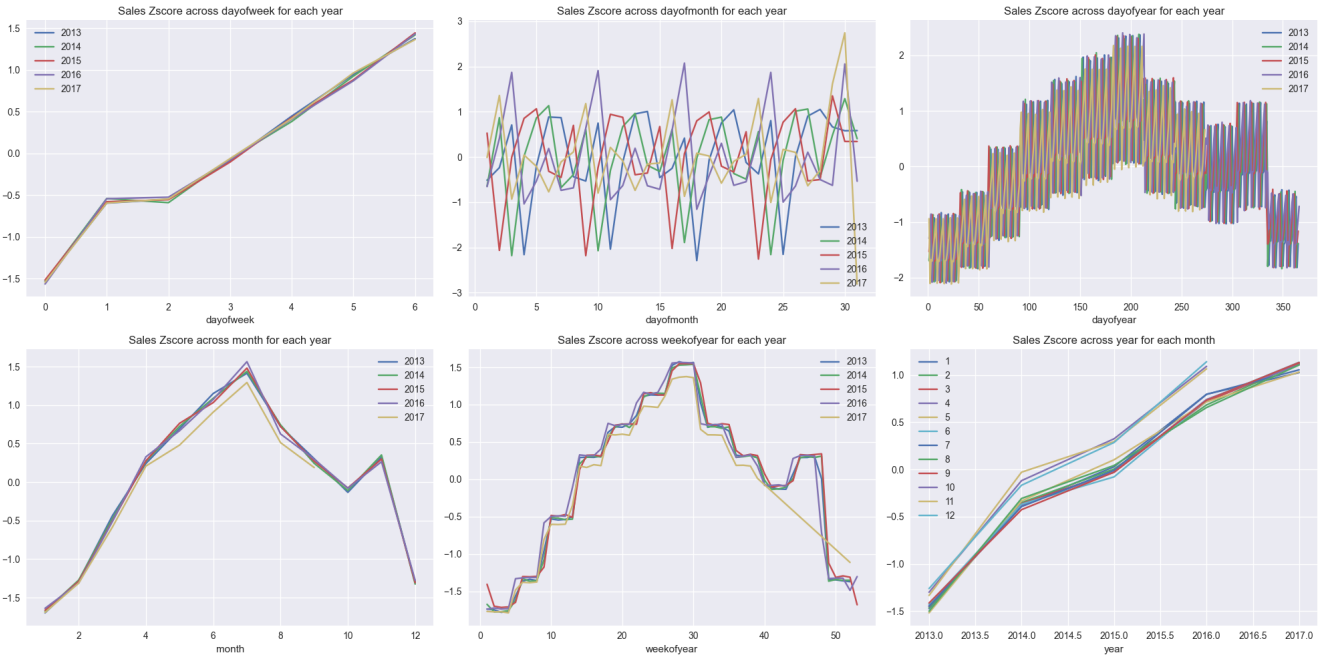

Fig. 6: Temporal features sales z-score

Each time series in our plots represents the variation across the temporal features (x-axis) for a particular year except the bottom right most plot of each figure, which represents the variation across years (x-axis) for a particular month. For each month, on a year-on-year basis, the trend is generally increasing and this corroborates the slightly upwards drift we see in our time series plots (Fig. 2 & 4).

The seasonal patterns we observed in our time series plots (Fig. 2 & 4) becomes more apparent and these patterns are most obvious across “dayofweek”, “month” and perhaps “weekofyear”. These features hold patterns that persist across time as each z-score time series for different years are stacked almost right on top of each other (Fig. 6). We can thus conclude that temporal dependencies and by extension, these temporal features are not degenerate.

This hints at strong annual and weekly seasonality and suggests that using seasonal time series models or including these temporal features as covariates in our regression models would help us capture the seasonalities.

3.4 Store, Item and Temporal Dependencies

Finally, we want to study how sales for different stores and items vary across the chosen temporal features with more discernible patterns (“dayofweek”, “month” and “year”). We aggregate sales across each item or store and the chosen temporal features and plot their sales z-score (Fig. 7).

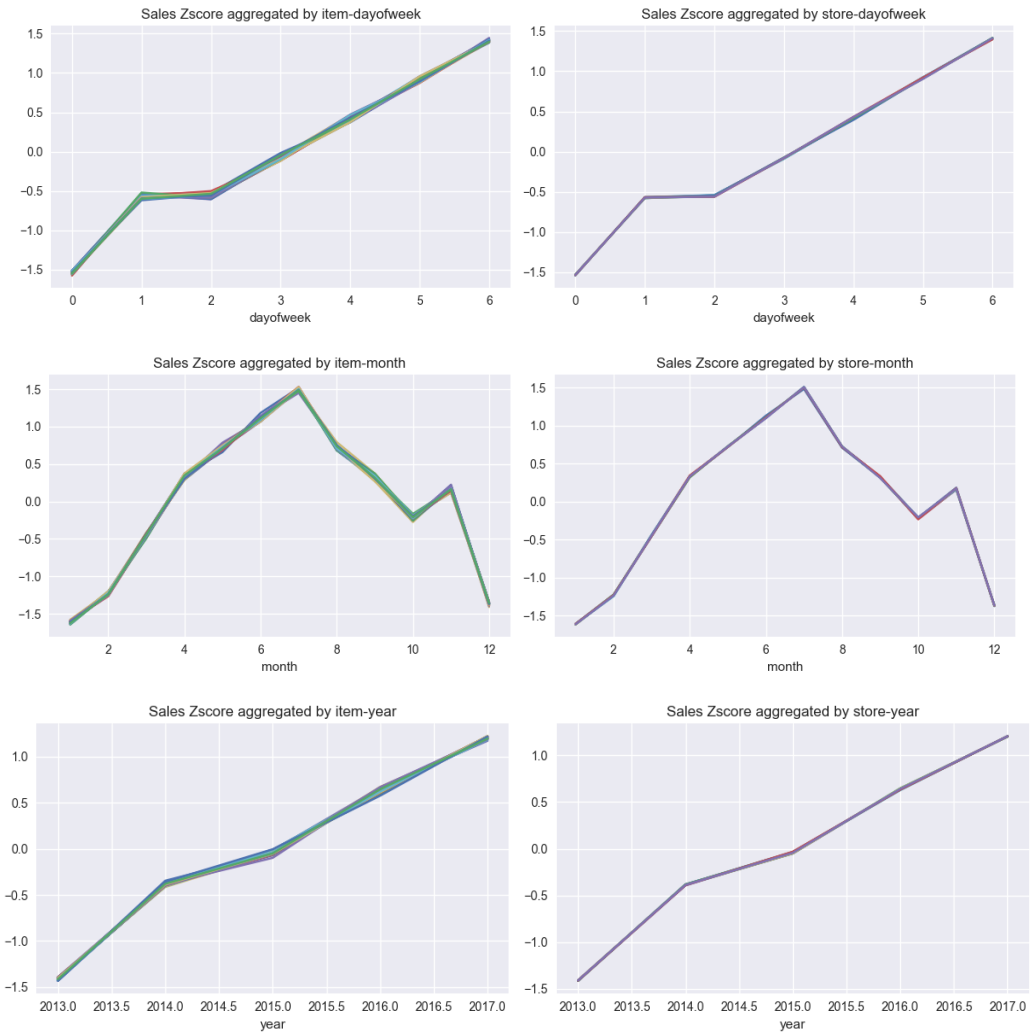

Fig. 7: Sales z-score across item/store and temporal features

We see that even amongst individual item and store groups, the patterns across each year and week are apparent. Historically, sales have generally been on the rise on a year-on-year basis for all stores and items, showing cyclical patterns where sales would rise up till July, before falling till December. Across the week, sales generally rose with a plateau across Tuesday and Wednesday.

This implies we could use weekly and annual periodicities to extrapolate future trends since these patterns tend to persist across time and are universal for store and item alike.

3.5 Time Series Decomposition

We have seen from the above that our data exhibits strong weekly and annual seasonalities. However, it is often beneficial to mathematically decompose a time series, isolating the trend-cycle and seasonal components for further analysis [3]. In addition, the decomposition can also be used in forecasting, where we individually forecast each component before aggregating them.

The Seasonal-Trend Decomposition Procedure Based on Loess (STL) [4] is a versatile and robust method for decomposing time series which extracts a single seasonal component, a trend component and a residual component. It does so by applying Loess which fits a smooth curve to scatterplots using polynomials [5]. However, STL is unable to decompose time series with multiple seasonal components which our dataset clearly exhibits. For that, we use Multiple Seasonal-Trend Decomposition using Loess (MSTL) [6]. MSTL assumes that the time series can be expressed as an additive decomposition (Eqn. 1) and builds on STL by iteratively extracting each seasonal component. MSTL has also been proven to be accurate, computationally efficient and robust to outliers [6].

$$y_t = \widehat{T}_t + \widehat{S}^{(1)}_t + \widehat{S}^{(2)}_t + \ldots + \widehat{S}^{(N)}_t + \widehat{R}_t$$

Eqn. 1: Additive decomposition with multiple seasonal components

Since our above EDA has shown that most store-item time series exhibit very similar seasonal and trend patterns, we choose only a specific store-item (store 2, item 28) time series to decompose.

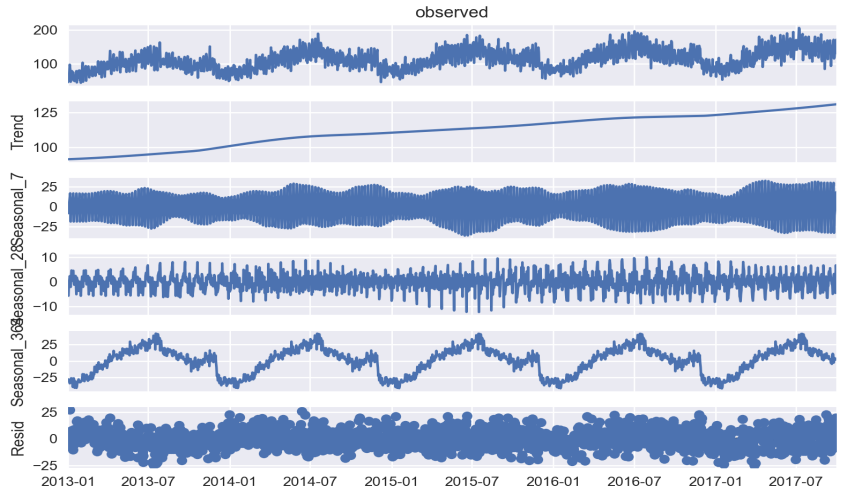

Fig. 8: Store 2, item 28's sales time series multiple seasonality decomposition using MSTL

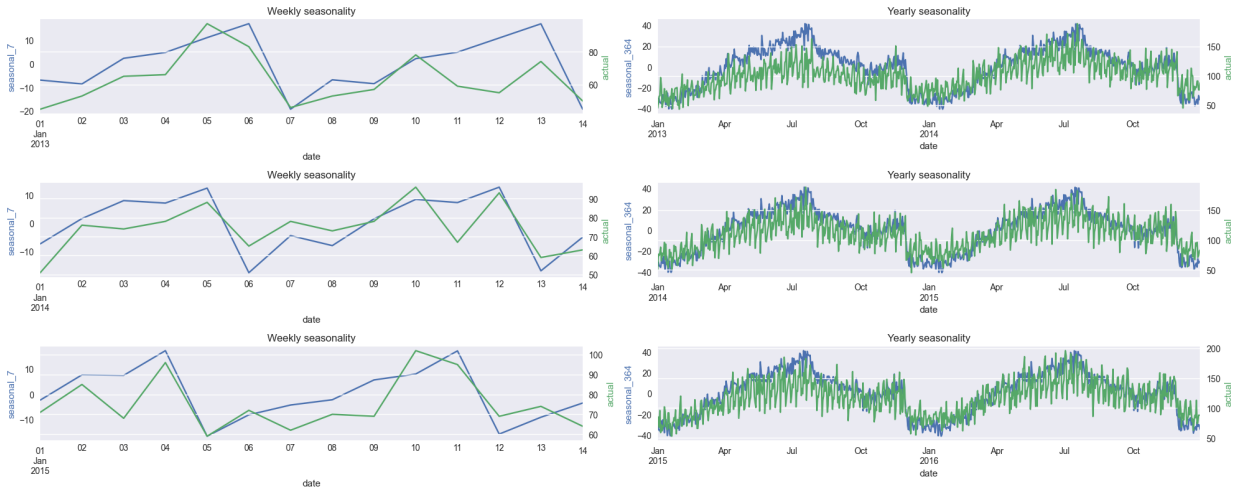

Fig. 9: Actual time series vs. MSTL decomposed components

We can see the seasonal components are decently captured, especially the yearly seasonality pattern. Weekly seasonality is also apparent as sales tend to rise during the first half of the week, plateauing midweek, before rising again in the later half of the week.

3.5.1 Residual Analysis

We turn to residual analysis to analyze if the MSTL algorithm has effectively captured our trend and seasonal components.

$$\hat{R}_t = y_t - \hat{y}_t$$

$$\text{where } \hat{y}_t = \hat{T}_t + \hat{S}^{(1)}_t + \hat{S}^{(2)}_t + \ldots + \hat{S}^{(N)}_t$$

Eqn 2: Residuals and fitted time series

Visually, the histogram resembles a normal distribution with mean approximately at zero. The time series plot also resembles white noise and shows that the variation stays pretty much constant.

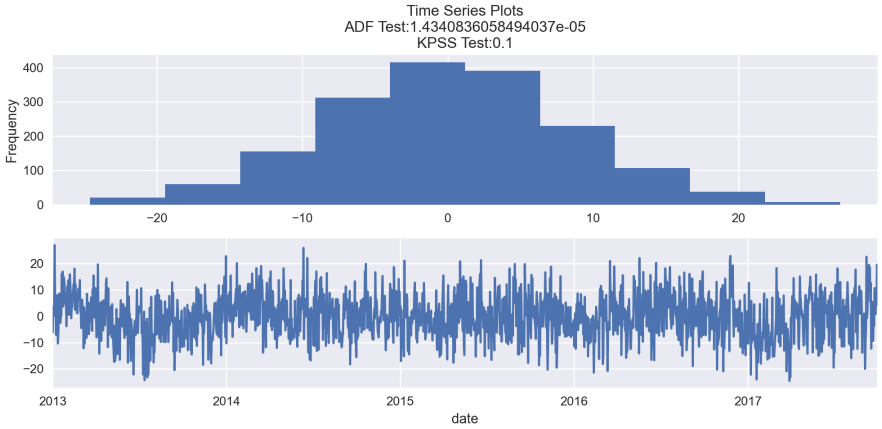

Fig. 10: Time series plots and ADF, KPSS p-values for raw time series of MSTL decomposition residuals

On top of visual inspection, we wish to study if 1) our residuals are stationary and 2) if they are independent and identically distributed to avoid spurious results [7]. Residuals are the difference

between our fitted model and the true time series (Eqn. 2). Any model that does not satisfy these properties have not accounted for all available information and can be improved [8].

3.5.1.1 Stationarity

We conduct 2 statistical tests – the Augmented Dicky Fuller (ADF) [9] and the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) tests [10], to check for stationarity of a time series. A stationary time series is a series whose statistical properties like mean and variance do not vary with time. After performing both tests, the ADF test produces a p-value lower than our designated threshold of 0.05, while the KPSS test results in a p-value higher than this threshold (seen in Fig. 11's title). Consequently, we reject the null hypothesis of the ADF test (series has a unit root and is not stationary) and do not reject the null hypothesis of the KPSS test (series is trend stationary). This leads to the conclusion that the time series in question is indeed stationary.

3.5.1.2 Autocorrelation

Autocorrelation (ACF) measures the linear relationship between a time series and lagged values of itself while partial autocorrelation (PACF) measures the relationship between 2 observations in a time series after removing the effects of observations between them.

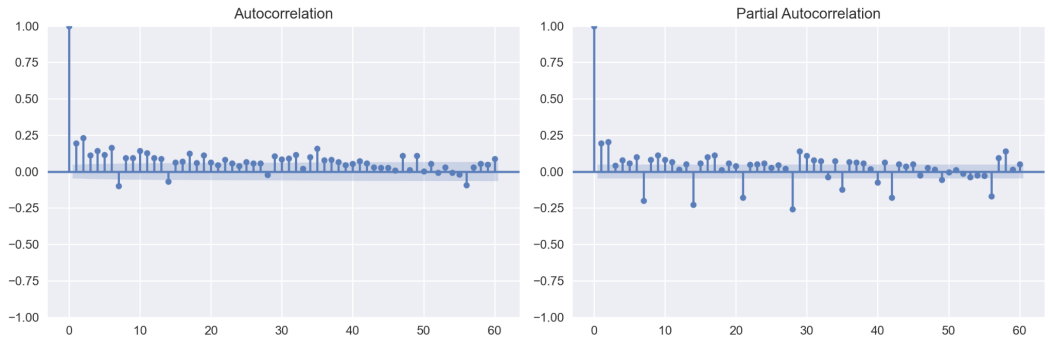

Fig. 11: Residuals correlogram and partial correlogram

If a time series is autocorrelated, it would have statistically significant correlation coefficients at appropriate lags (above the shaded blue regions in Fig. 11).

We notice that while the residuals are stationary, there still exists significant autocorrelations as there are many lags with statistically significant autocorrelation coefficients. This means that the MSTL decomposition algorithm was still not able to capture the autocorrelation from our residuals and can be improved. Later on, we will explore dynamic regression models [11] that allow us to model autocorrelated residuals with Autoregressive Moving Average (ARMA) models and its variants.

4. Data Preprocessing

Many time series models require stationarity to avoid spurious results. Thus, we have to ensure stationarity in the time series we are trying to forecast. Our underlying sales time series is clearly non-stationary as it exhibits both trend and seasonality, causing the mean and variance to vary with time. To remove trend non-stationarity we take the first difference of our time series (Eqn. 4). Since our data also exhibits seasonality, we apply a seasonal difference (Eqn. 3) as well [12].

$$y_t' = y_t - y_{t-1}$$

$$y_t'' = y_t - y_{t-m}$$

Eqn. 3: First difference, y_t' and seasonal difference, y_t'' of time series y_t where $m = 7$

We observe that taking the first difference and a weekly seasonal difference is sufficient to ensure stationarity in our time series (Fig. 12 & 13).

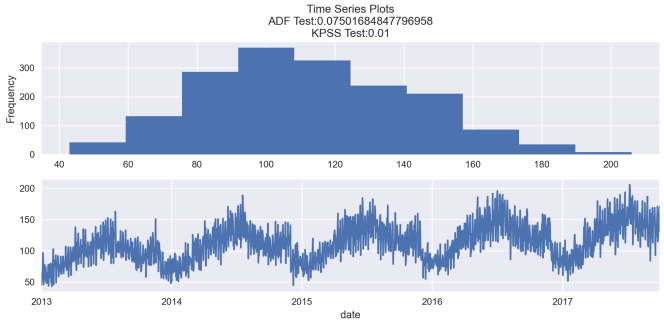

Fig. 12: Raw time series

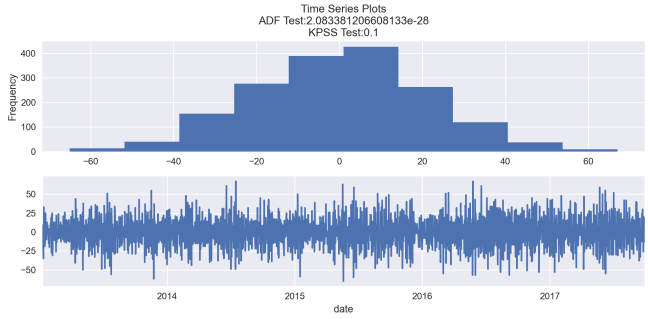

Fig. 13: First and seasonal differenced time series

5. Feature Engineering

Before moving on to modeling, we want to feature engineers based on the conclusions of our EDA in section 3. In summary, our EDA in section 3 showed:

1. Consistent sales pattern across both stores and items
2. Each store and item group have different sales distribution
3. Sales time series has an upward trend component
4. Sales time series exhibit multiple seasonal components, specifically weekly and annually, that persists across time
5. Sales time series is non-stationary and needs to be transformed for models that require stationarity

With the above considerations in mind, we want to feed the following features to our models:

1. Store and Item: categorical features to capture store, item patterns
2. Temporal Features: categorical features included to capture seasonal patterns
3. Lagged Sales Features: numerical features that capture sales performance 1 year ago.
4. Average Sales Features: numerical features of historical average sales across different day of weeks and months that capture long and short term trends
5. Sales Rolling Average Features: numerical features that encapsulate recent sales dynamics.

6. Methodology

Our methodology consists of experimenting with both time series forecasting models and traditional machine learning algorithms.

Since the given dataset is periodic in nature, we can attempt to fit a smooth curve through polynomials and Fourier terms using our MSTL decomposition algorithm. Additionally, ARMA models have also been shown to provide a parsimonious description of stationary series and is a useful tool when it comes to understanding the structure of our data. We will also experiment with traditional machine learning algorithms such as XGBoost as this allows us to include more covariates in our model, possibly improving our forecasting accuracy [13].

We assess our forecast accuracy with the symmetric mean absolute percentage error (SMAPE) as this is the evaluation metric used in the Kaggle competition. However, we train our models with mean squared error as our loss function for reasons of mathematical convenience and consequently help guide our model to output unbiased estimators on average. Furthermore, training a model on SMAPE tends to result in heavily biased forecasts as it carries over the shortcomings of training on MAPE [14].

7. Time Series Models

We start by fitting simple time series models before incrementally increasing their complexity. Since most time series models struggle to include covariates when modeling unlike non-linear machine learning approaches, we fit our time series models to individual store-item time series.**7.1 Benchmark Model**

Our baseline model forecasts all future values to be the value of the last observation of a simple moving average. The simple moving average represents the unweighted mean of the previous *k* data points. We compute the simple moving average (Eqn. 4) and set a window of 365 days.

$$\widehat{y}_T = \frac{1}{k} \sum_{i=t-k+1}^{t} y_i$$

Eqn. 4: Simple moving average of time series y of window k

7.1.1 Results

This model returned a SMAPE of 22.44% for our training set and a SMAPE of 20.20% on the validation set. There is little point in studying the residuals for this approach since intuitively we know there is much more information to be captured. Extrapolating the moving average of the past year for different store, item time series clearly does not capture dynamic movements but serves as a good benchmark.

7.2 Time Series Decomposition Forecast

From our EDA above, we have shown that there exists more than 1 seasonality component in our time series. With our MSTL decomposition, we have also shown that our time series can be decomposed into trend, seasonal and residual components. Rewriting Eqn. 1, we get Eqn. 5 where $\widehat{A}_t = \widehat{T}_t + \widehat{R}_t$ represents the seasonally adjusted component comprising the trend and residual components.

$$y_t = \widehat{A}_t + \widehat{S}^{(1)}{}_t + \widehat{S}^{(2)}{}_t$$

Eqn. 5: Time series additive decomposition into seasonally adjusted component and 2 seasonal components

We can forecast a decomposed time series by forecasting the seasonally adjusted component, the weekly seasonal component, and the annual seasonal component separately [15]. To forecast our seasonally adjusted and seasonal components, our first approach consists of naively fitting a polynomial to the seasonally adjusted component, and using Fourier series approximation to fit the periodic seasonal components. We use polynomials of degree 2 and 10 Fourier terms as part of our Fourier Series approximation (Eqn. 6)

$$f(x) = a + \sum_{k=1}^{10} [\alpha_k \, sin(2\pi kx/m) + \beta_k \, cos(2\pi kx/m)]$$

Eqn. 6: Fourier series function approximation

For each seasonal component, we first use Fast Fourier Transform (FFT) to identify the most significant frequencies. We then feed these frequencies as our initial guess for fitting a Fourier series to these seasonal component. We then extrapolate our polynomial and Fourier series and sum them up to get forecasts for our validation set.

7.2.1 Residual Analysis and Results

In section 3.5.1, we studied the residuals to determine how much information our MSTL model is capturing. Likewise, for our time series decomposition forecast and for subsequent time series models, we would like to investigate the presence of non-stationarity and autocorrelation of our residuals to better understand if our models has captured all available information. Since we have 500 unique fitted models for each store and item, we write a simple script that tests for non-stationarity and autocorrelation in residuals. We use the 2 stationarity tests introduced in 3.5.1.1, the ADF and KPSS tests and employ a portmanteau test, specifically the Ljung-Box test [8] that checks if the first l autocorrelations are significantly different from white noise process to test for autocorrelation.

This model returned a SMAPE of 13.53% for our training set and a SMAPE of 14.06% for our validation set which represents a rather drastic improvement as compared to our benchmark model. Submitting out of sample predictions to Kaggle gave us a SMAPE of 15.81% on the public scoreboard which is decent for the simplicity of this model. While our SMAPE score paints a rosy picture, we find that 73/500 residuals failed the stationarity tests (i.e residuals are not stationary) and 372/500 residuals failed the Ljung-Box test (i.e residuals are autocorrelated). This potentially indicates spurious regression results which might be misleading to the true underlying relationship between our variables.

7.3 Dynamic Regression Models

Some clear disadvantages of curve fitting our seasonally adjusted and seasonal components is that it tends to overfit and fails to capture more granular and intricate details.

From our EDA, we have shown that our time series is non-stationary and exhibits strong seasonality. It would therefore be reasonable to experiment with Seasonal Autoregressive Integrated Moving Average (SARIMA) models [16] as they allow us to ensure stationarity with first and seasonal differences and capture seasonalities. While SARIMA has proven to be a state of the art time series modeling approach, it has two major drawbacks when it comes to seasonalities – multiple seasonalities and long seasonalities. Our data exhibits multiple seasonalities, one of which being a yearly seasonality which is considered too long for SARIMA models to handle [17].

We can solve this issue with dynamic regression models [11] that allow us to account for multiple long seasonal periods by including seasonal features [17]. These external seasonal features are not part of our observed sales time series and are often termed as exogenous variables. This reduces our traditional SARIMA model to a linear regression (Eqn. 7) where the errors, η_t which are typically assumed to be white noise, now follows a SARIMA model [18]. Note that SARIMA models are generalizations of ARIMA models.

$$y_t = \beta_0 + \beta_1 x_{1,t} + \ldots + \beta_k x_{k,t} + \eta_t$$

$$(1 - \sum_{i=1}^{p} \phi_i L^d)(1 - L^d)(1 - \sum_{i=1}^{P} \Phi_i L^D)(1 - L^D)\eta_t = (1 + \sum_{i=1}^{q} \theta_i L^d)(1 - \sum_{i=1}^{Q} \Theta_i L^D)\epsilon_t$$

Eqn. 7: Dynamic Regression Model equation where η_t follows a SARIMA model [11]

L is the lag operator, ϕ_i are coefficients for AR terms, θ_i are coefficients for MA terms, Φ_i are coefficients for seasonal AR terms, Θ_i are coefficients for seasonal MA terms and ϵ_t are errors which are assumed to follow a white noise process. SARIMA models are denoted as SARIMA($p, d, q)(P, D, Q)s$ where p and q denote the non-seasonal AR and MA orders respectively while P and Q denote the seasonal AR and MA orders. d and D denote the order of differencing and seasonal differencing respectively. Lastly, s denotes the periodicity of the seasonality.

To determine the parameters for p, d, q, P, D, Q, s, we utilize pmd's Auto-ARIMA, an automation model that seeks to identify the optimal order for a SARIMA model through optmizing for information criterions [19]. Auto-ARIMA works by first ensuring stationarity (setting d and D values) before searching for the best p, q, P, Q parameters within user-defined ranges.

We will run auto-ARIMA for a single store-item time series and the set of exogenous features to be fed into our model to determine the optimal SARIMA model parameters. While there exists heuristics to determine optimal model parameters for SARIMA models [16, 20], we prefer an auto-ARIMA as it optimizes on a metric such as Akaike's Information Criterion [21] that penalizes the fit of the model with model complexity. We perform cross-validation on the models recommended by auto-ARIMA using our validation set. The model that yields the lowest SMAPE on this set, and is also among the top three models with the lowest AIC values, is selected. We would then use this optimal SARIMA model to train all 500 unique store-item time

series following earlier arguments on non-degeneracy of seasonality structures across store-item time series.

7.3.1 Temporal Features: Residual Analysis and Results

Our first attempt at including exogenous variables would be to include weekday and months one-hot-encoded temporal features given that we want to capture weekly and yearly seasonality [17]. We one-hot-encode our temporal features since there does not exist natural ordinal rankings between them in terms of sales. For example, September's sales are not greater than July's sales as seen in our EDA above. We run Auto-ARIMA and find that the optimal model for our regression errors is a SARIMA$(4, 1, 1)(0, 0, 0)0$.

This returned a SMAPE of 13.00% for our training set and a SMAPE 12.66% for our validation set. Submitting our forecasts to Kaggle gave a SMAPE of 14.90% on the public scoreboard. Upon studying our residuals, we find that 5/500 failed the stationarity tests while 109/500 failed the Ljung-Box tests. This is a great improvement when it comes to capturing information in our given time series. Investigating further, we find that visually the autocorrelation is not particularly large (Fig. 14) and is unlikely to have any impact on forecasting. We can conclude that this experiment was a success.

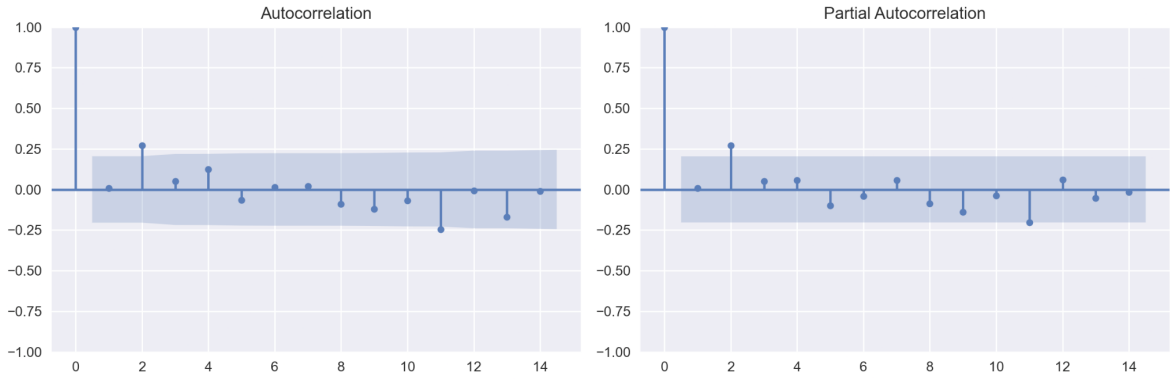

Fig. 14: Residuals ACF and PACF plots for a store 1, item 1 which failed Ljung Box test

7.3.2 Dynamic Harmonic Regression: Residual Analysis and Results

Another approach to capturing long seasonal periods would be to use up to *K* terms of a Fourier series as exogenous variables [22, 23]. This approach allows us to capture any length seasonality and for data with more than one seasonal period, Fourier terms of different frequencies can be included (Eqn. 8)

$$y_t = a + \sum_{k=1}^{K} [\alpha_k \, sin(2\pi kt/m) + \beta_k \, cos(2\pi kt/m)] + \eta_t$$

Eqn. 8: Fourier terms as exogenous variables where η_t is the SARIMA model

We set *K*=2 and include annual and yearly seasonalities as periods to capture the multiple seasonalities displayed by our time series. The optimal model parameters after running auto-ARIMA is a SARIMA$(2, 1, 1)(0, 0, 0)0$.

This returned a SMAPE of 14.46% for our training set and a SMAPE of 15.80% for our validation set and on the Kaggle public scoreboard, a SMAPE of 15.77%. This time, our residual analysis shows that for almost every store and item, the model fitted certainly hints at spurious regression results as 281/500 time series failed the stationarity tests while 451/500 failed the Ljung-Box test.

We can conclude that using Fourier Terms as exogenous variables may not be a good choice when it comes to capturing seasonalities. Alternatively, this could imply the use of additional terms.

7.3.3 Temporal Features and Fourier Terms: Residual Analysis and Results

Finally, we combine the above 2 approaches and use temporal features as well as fourier terms to further guide our dynamic regression model. We find the optimal model parameters to be SARIMA$(5, 1, 1)(0, 0, 0)0$. Note that up until this point, none of the optimal model parameters include seasonal differencing or seasonal terms. This could be attributed to the fact that all seasonalities are already accounted for by our exogenous variables and the residuals or errors to be modelled by SARIMA does not encapsulate any additional seasonality. This is in line with our EDA analysis.

This returned a SMAPE of 13.02% for our training set and a SMAPE of 12.59% for our validation set. This approached proved to be the best one yet for our in-sample period. However, upon submitting on Kaggle for true out-of-sample forecasts, it returned a SMAPE of 24.73% hinting at a rather prevalent variance problem. This is despite most time series residuals passing our stationarity tests (only 7/500 failed) and Ljung-Box tests (only 104/500 failed). Likewise looking at visual plots proves that the autocorrelations are not particularly large (Fig. 15).

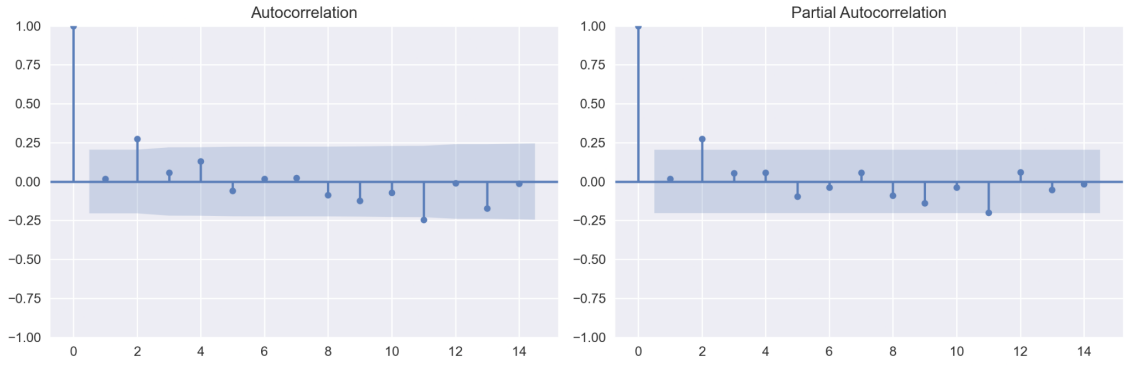

Fig. 15: Residuals ACF and PACF plots for a store 1, item 1 which failed Ljung Box test

8. Machine Learning Models

While our time series models produce decent forecast accuracy results, they are parametric and linear in nature, and are inefficient in training. SARIMA models are a class of linear models using historical data to forecast future values and our time series decomposition approach assumes a certain fixed distribution across sales for each store and item. They are also inefficient when it comes to training since we have to train 500 unique models (each for a particular item in a store). Given that, we do not wish to tune hyperparameters and cross-validate for 500 unique

models. To top it off, time series models cannot handle the non-linearities in external covariates. As such, we turn to traditional machine learning models for faster training speeds and in an attempt to capture the non-linearity in our data.

8.1 Base Model

We start with a base model that outputs predictions of a particular item at a particular store as the historical average of their sales, across the entire training set, for each day of the month. For example, for store 1, item 1's prediction output on the 1st of January would equal the historical average for store 1, item 1's sales across our training set on all 1st of Januarys. This shrinks our sample size to approximately 150 (30 days a month * 5 years of data). This will definitely be a sub-optimal solution which will form the benchmark for future models. The SMAPE for the train set is 17.25% and 22.50% for the validation set.

8.2 Factor Model

We can make improvements to our naive bayes model drawing from the conclusions of our EDA.

8.2.1 Analysis of Seasonal Sales Variability using the Coefficient of Variation

From our EDA, we observe that the sales data exhibit stable seasonal and weekly patterns over the years where the first and seasonal differenced time series have been tested to be stationary. We will now check for the stability of the coefficient of variation (CV) over time so that we can check for any pattern degeneracies across time. CV is calculated by dividing the standard deviation by the mean for the specific groupings. From our plots, we observe that the patterns are consistent and stable. For instance, the plots are parallel and there are no sudden spikes or drops and the CV for sales on Mondays in March does not change much from year to year.

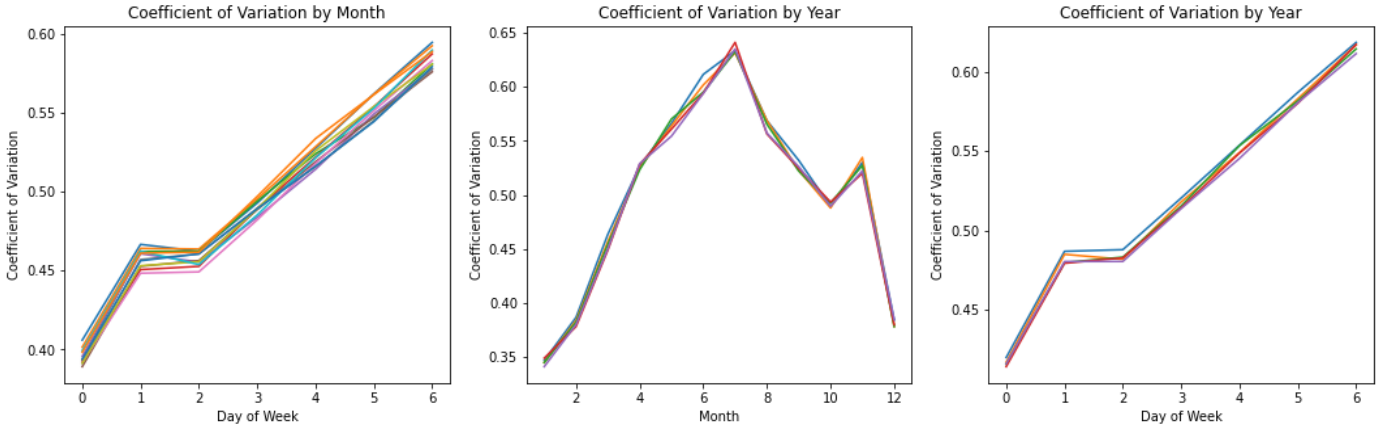

Fig 16: Coefficient of Variation across months, years and day of week

This further suggests that the relative variability of the sales data is consistent over time and across different levels of sales. Thus, we can effectively treat the "month", "year", "day of the week", "item" and "store" as completely independent factors to sales prediction without the need for more complex or dynamic modeling approaches.

8.2.2 Observations to the Relative Sales by Year

From our EDA, we also observe that the normalized sales by year increases at a decreasing rate (dividing yearly sales number across all stores by the average sales across 5 years). To predict the sales in 2018, we will require extrapolation of the relative sales number in 2018.

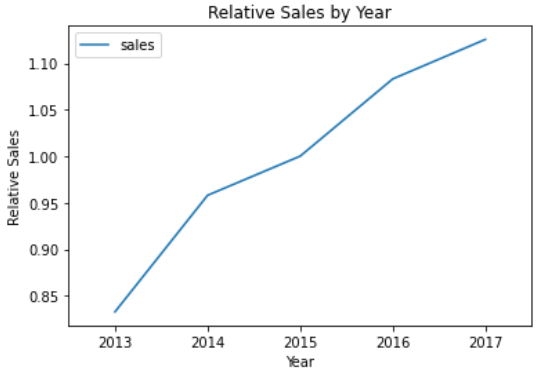

Fig 17: Relative Sales from 2013 to 2017

8.2.3 Improved Model

Let $A_{i,s}$ be the historical average sales for Item i at Store s.

Let W_d be the factor for day of week where d = 1 if it is a Monday.

Let M_m be the factor for the month m.

Let G_y be the annual growth factor in year y.

$$Predicted\ Sales = A_{i,s} * W_d * M_m * G_y$$

Eqn. 9: Predicted Sales factor equation

Before we can predict the sales, we need to derive a suitable approximation for G_{2018}. Since there are only 5 data points, this will be tricky as we will run into risks of underfit. Regardless, I will predict the G with 6 different regressors, and select the regressors based on the SMAPE in our validation set. For our train and validation data set, a linear fit will appear to be a better fit than a quadratic fit. However, this can be accounted for by the fact that relative sales value could be overestimated if sales increase at a decreasing rate with time which can be better observed when we calculate the relative sales for the entire dataset (till 2017-12-31).

Model	Train SMAPE (%)	Test SMAPE (%)
Linear	18.12	12.43
Quadratic	17.69	12.61
Ridge	16.77	13.23
Lasso	15.30	16.04

Elastic Net	16.23	13.81
SVR	15.24	16.38

SMAPE further improved for the Quadratic model in the validation set (down to 12.38%) when we added an exponential decay function for our weight to the quadratic fit. This allows the model to place more emphasis on more recent data points. 4 different weights were tested as seen below.

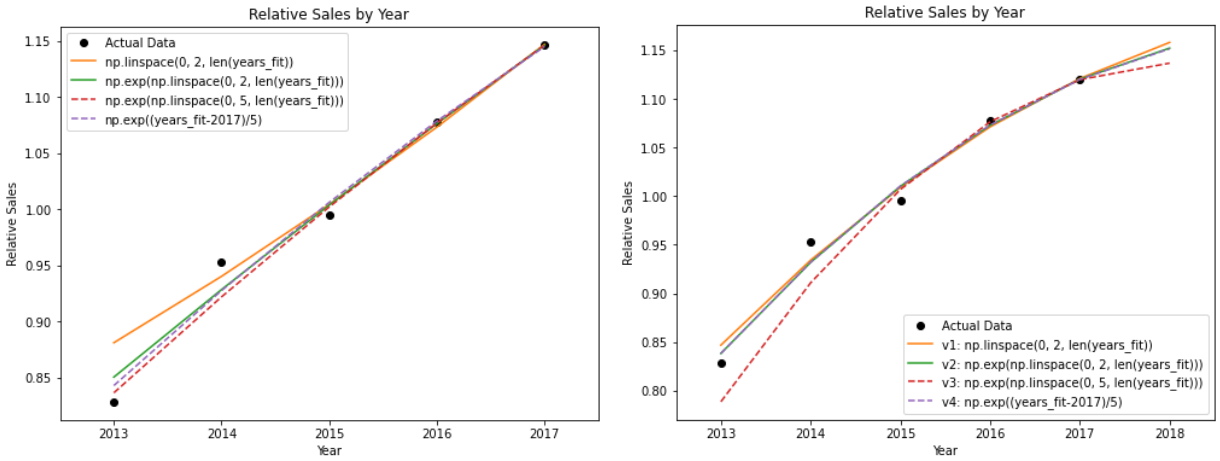

Fig 18: Quadratic Fit with Different Weights from 2013-01-01 to 2017-10-01 vs 2013-01-01 to 2017-12-31

8.2.4 Results

Based on the abovementioned model, with the chosen weights, we extrapolated the relative sales in 2018 to be 1.1367. Submitting this prediction to Kaggle gave us a SMAPE of 13.87% when we round the predictions to the nearest integer. Considering the simplicity of our solution, it was a stark improvement from the base model and will form the benchmark for subsequent machine learning models.

8.3 eXtreme Gradient Boosting

XGBoost is a highly regarded model for predictive tasks due to its ability to handle diverse data types and its robustness against overfitting, owing to built-in regularization. Its strength lies in feature importance evaluation, and its computational efficiency and performance [24]. For sales predictions that involve complex patterns, XGBoost is particularly effective, as it combines multiple decision trees to produce stable and accurate forecasts. Being a tree-based model, it can also capture non-linearity well, making it an excellent choice for our problem.

8.3.1 Feature Importance

For our analysis, we employed the previously mentioned features. It's crucial that these features not only enrich the dataset but also significantly contribute to the model's predictive accuracy. XGBoost's feature importance metrics like weight, gain, and cover are instrumental in pinpointing the most impactful features, allowing us to streamline the model by discarding redundant features and reducing the risk of overfitting [25]. The feature importance scores for

each feature used can be seen in Fig. 19. Our initial run of the XGBoost model, without any tuning, achieved a SMAPE of 12.94% in the train set and 12.97% in the test set. Our next step is to refine the model by focusing on the most relevant features and optimizing its hyperparameters.

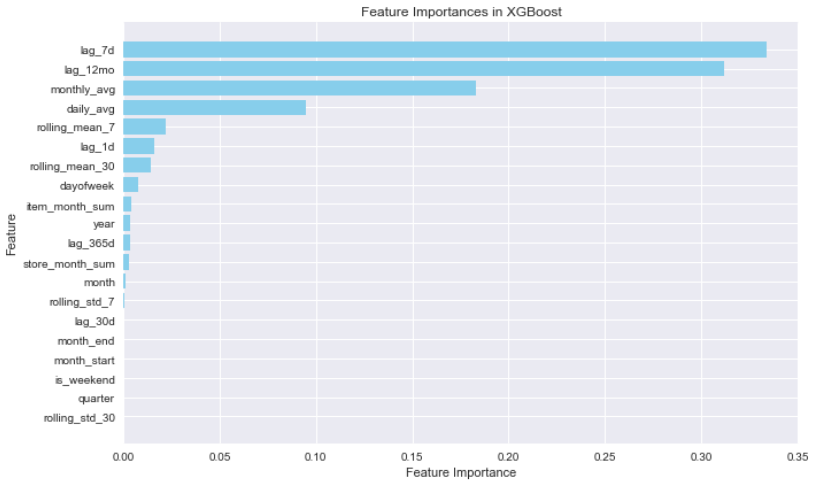

Fig 19: Feature Importance in XGBoost

8.3.2 Feature Selection

Feature selection was aimed at retaining only the most relevant predictors which passed a threshold for feature importance. By focusing on a subset of highly influential features, we aimed to improve the model's generalizability and robustness, ultimately leading to a more streamlined, efficient, and effective predictive model.

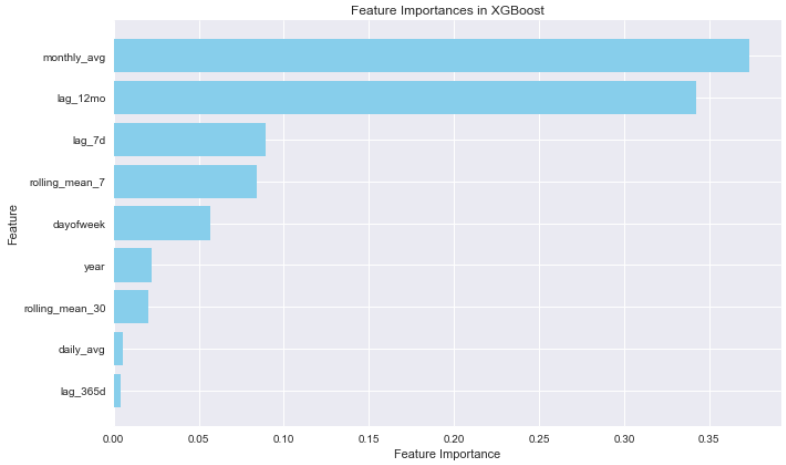

Fig 20: Feature Importance of Selected Features in XGBoost

8.3.3 Hyperparameter tuning

We used Optuna [26] for hyperparameter tuning of our XGBoost model, refining critical parameters to minimize MAE. Parameters like 'max_depth', 'min_child_weight', and 'gamma' were adjusted for model depth, overfitting control, and regularization, respectively. We also tuned 'subsample' and 'colsample_bytree' to regulate the data subset for tree construction, aiding in generalization. Additionally, 'reg_lambda' and 'reg_alpha' were optimized for stability, while varying 'learning_rate' and 'n_estimators' helped find the best training speed and number of

boosting stages [27]. This rigorous tuning, enhanced by Optuna's search efficiency and cross-validation across time periods boosted our model's accuracy and robustness.

8.3.4 Results

This final model yielded a SMAPE of 11.93% in the train set and 12.21% in the test set, marking a modest yet significant enhancement over the baseline model, with the added advantage of a reduced feature set. When this optimized model was submitted to the Kaggle competition, it achieved a SMAPE of 14.24%.

8.4 Random Forest Regressor

Comparing a Random Forest Regressor with XGBoost is beneficial as it juxtaposes a bagging method against a boosting one, offering insights into how different ensemble strategies fare on your dataset. Random Forest's resistance to overfitting and robust handling of missing values and outliers make it a valuable baseline, especially for interpretability and simplicity. This comparison can elucidate the impact of dataset complexity, feature importance, and hyperparameter sensitivity, aiding in choosing the optimal model based on performance and computational efficiency.

8.4.1 Feature Importance

Random Forest calculates importance differently from XGBoost, where importance is based on the decrease in impurity across trees, providing a direct measure of a feature's impact on model purity [28]. The bar chart below (Fig. 21) illustrates the feature importances.

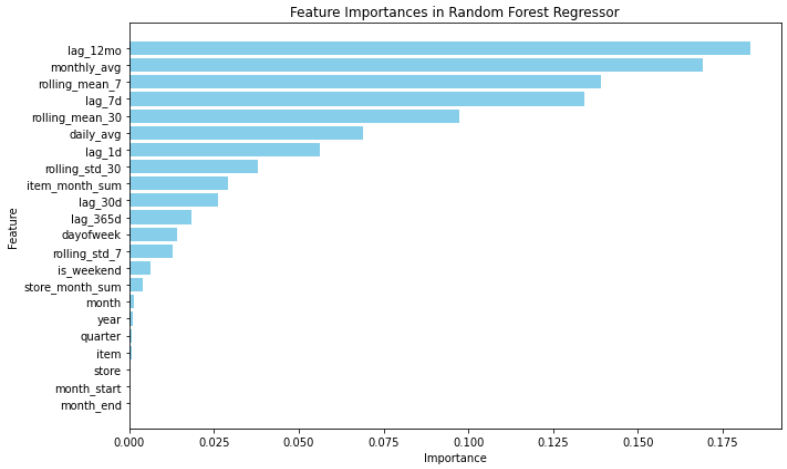

Fig 21: Feature Importance in Random Forest Regressor

Our initial run of the Random Forest model, without any tuning, achieved a SMAPE of 12.64% in the train set and 12.69% in the validation set. Our next step is to refine the model by focusing on the most relevant features and optimizing its hyperparameters.

8.4.2 Feature Selection

Similar to XGBoost, we set a threshold for feature importance, selecting only those variables that demonstrated a significant impact on the model's predictive accuracy. Another observation is that the selected features are relatively consistent to XGBoost's.

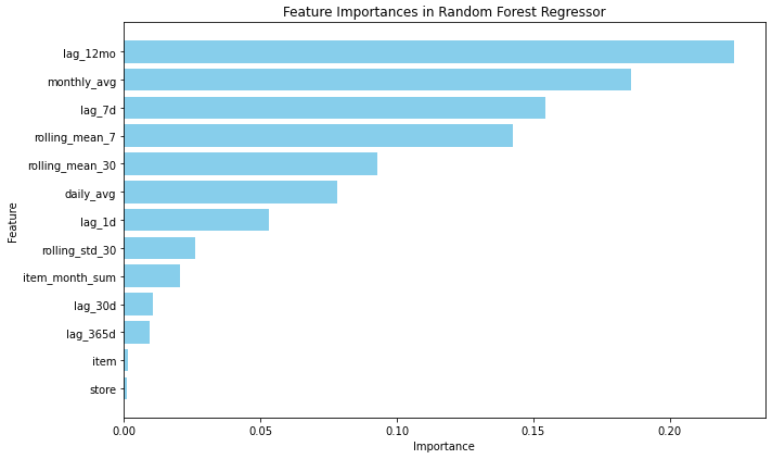

Fig 22: Feature Importance of Selected Features in Random Forest Regressor

8.4.3 Hyperparameter tuning

In this analysis, we've mirrored the XGBoost model's hyperparameter optimization process, utilizing the Optuna optimization framework which focused on key hyperparameters including the number of estimators, the maximum depth of trees, the minimum number of samples required to split a node, the minimum number of samples required at a leaf node, and number of features to consider when looking for the best split [29]. Additionally, various imputation strategies 'mean, median, most_frequent, constant' were evaluated to handle missing data effectively, ensuring the model's robustness. We also employed a time series cross-validation method similar to what was done for XGBoost.

8.4.4 Results

This final model yielded a SMAPE of 12.60% in the train set and 12.52% in the validation set, marking a slight improvement from the unoptimized model. When this optimized model was applied to the Kaggle competition, it achieved a SMAPE of 14.17%, which is a slight improvement to the XGBoost model.

9. Conclusion

While this project allowed us to learn and implement a myriad of time series forecasting tools coupled with complex statistical tests, the relatively simple and interpretable factor model in section 8.2.3 stood out. It achieved a commendable SMAPE of 13.87% (Top 15%) on the Kaggle leaderboard, a testament to its efficacy in capturing the essence of the simulated dataset. The simplicity of using the mean sales as a primary feature in our factor model underscored the power of fundamental statistical measures, often outperforming more complex approaches. This success may be attributed to the dataset's synthetic nature, where determining precise

multiplicative factors without succumbing to overfitting proved critical for out-of-sample accuracy (which were the likely causes of our time series models and other machine learning models performing less optimally when submitted to Kaggle).

Notably, our exploratory data analysis paved the way for a clean and insightful visualization of data trends, reinforcing the significance of thorough preliminary analysis. However, our methodology is not without its caveats. A heavy reliance on historical patterns to forecast future sales carries the risk of being upended by unforeseen market dynamics or disruptive global events. Therefore, continuous monitoring and updating of these models are paramount to maintain their predictive accuracy and relevance. Future users of these models should remain vigilant, recalibrating the models as new data unfolds to ensure their decisions remain data-driven and contextually sound.

References

1. *Kaggle. (n.d.). Retrieved from https://www.kaggle.com/competitions/demand-forecasting-kernels-only/overview*
2. *scikit-learn developers. (n.d.). 6.3 Preprocessing Data. Retrieved from https://scikit-learn.org/stable/modules/preprocessing.html*
3. *Hyndman, R. J., & Athanasopoulos, G. (2021). 3 Time series decomposition. In Forecasting: Principles and practice. S. l.: Otexts.*
4. *Cleveland, R. B., Cleveland, W. S., McRae, J. E., & Terpenning, I. (1990). STL: A seasonal-trend decomposition. J. Off. Stat, 6(1), 3-73.*
5. *Cleveland, W. S., & Devlin, S. J. (1988). Locally weighted regression: an approach to regression analysis by local fitting. Journal of the American statistical association, 83(403), 596-610.*
6. *Bandara, K., Hyndman, R. J., & Bergmeir, C. (2021). MSTL: A seasonal-trend decomposition algorithm for time series with multiple seasonal patterns. arXiv preprint arXiv:2107.13462.*
7. *Granger, C. W. J., & Newbold, P. (1974a). Spurious regressions in Econometrics. Journal of Econometrics, 2(2), 111–120. doi:10.1016/0304-4076(74)90034-7*
8. *Hyndman, R. J., & Athanasopoulos, G. (2021). 5.4 Residual Diagnostics. In Forecasting: Principles and practice. S. l.: Otexts.*
9. *Mushtaq, R. (2011). Augmented dickey fuller test. SSRN Electronic Journal. doi:10.2139/ssrn.1911068*
10. *Kwiatkowski, D., Phillips, P. C. B., Schmidt, P., & Shin, Y. (1992). Testing the null hypothesis of stationarity against the alternative of a unit root. Journal of Econometrics, 54(1–3), 159–178. doi:10.1016/0304-4076(92)90104-y*
11. *Hyndman, R. J., & Athanasopoulos, G. (2021). 10 Dynamic regression models. In Forecasting: Principles and practice. S. l.: Otexts.*
12. *Hyndman, R. J., & Athanasopoulos, G. (2021). 9.1 Stationarity and differencing. In Forecasting: Principles and practice. S. l.: Otexts.*
13. *Stack Exchange (2019). XGBoost vs Arima for time series analysis. Retrieved from https://datascience.stackexchange.com/questions/60678/xgboost-vs-arima-for-time-series-analysis*
14. *Kolassa, S. (2017). What are the shortcomings of the Mean Absolute Percentage Error (MAPE)? Retrieved from https://stats.stackexchange.com/questions/299712/what-are-the-shortcomings-of-the-mean-absolute-percentage-error-mape*
15. *Hyndman, R. J., & Athanasopoulos, G. (2021). 5.7 Forecasting with decomposition. In Forecasting: Principles and practice. S. l.: Otexts.*
16. *Hyndman, R. J., & Athanasopoulos, G. (2021). 9.9 Seasonal ARIMA Models. In Forecasting: Principles and practice. S. l.: Otexts.*
17. *Hyndman, R. J. (2013). Forecasting with daily data. Retrieved from https://robjhyndman.com/hyndsight/dailydata/*
18. *Hyndman, R. J. (2010). The ARIMAX model muddle. Retrieved from https://robjhyndman.com/hyndsight/arimax/*
19. *(N.d.). pmdarima: ARIMA estimators for Python. Retrieved from https://alkaline-ml.com/pmdarima/index.html*
20. *Hyndman, R. J., & Athanasopoulos, G. (2021). 9.5 Non-seasonal ARIMA Models. In Forecasting: Principles and practice. S. l.: Otexts.*
21. *McElreath, R. (2016). p. 189. In Statistical rethinking a Bayesian course with examples in R and Stan. Boca Raton: CRC Press, Taylor & Francis Group.*
22. *Hyndman, R. J. (2010). Forecasting with long seasonal periods. Retrieved from https://robjhyndman.com/hyndsight/longseasonality/*

23. *Hyndman, R. J., & Athanasopoulos, G. (2021). 10.5 Dynamic Harmonic Regression. In Forecasting: Principles and practice. S. l.: Otexts.*
24. *xgboost developers. (n.d.). Introduction to Boosted Trees. Retrieved from https://xgboost.readthedocs.io/en/latest/tutorials/model.html*
25. *Abu-Rmileh, A. (2021). Retrieved from https://towardsdatascience.com/be-careful-when-interpreting-your-features-importance-in-xgboost-6e16132588e7*
26. *Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. doi:10.1145/3292500.3330701*
27. *xgboost developers. (n.d.). XGBoost Parameters. Introduction to Boosted Trees. Retrieved from https://xgboost.readthedocs.io/en/latest/parameter.html*
28. *scikit-learn developers. (n.d.). sklearn.ensemble.RandomForestRegressor. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html*
29. *Dubey, A. (2023). Feature Selection Using Random forest. Retrieved from https://towardsdatascience.com/feature-selection-using-random-forest-26d7b747597f#:~:text=The%20more%20a%20feature%20decreases,final%20importance%20of%20the%20variable.*