School of Computer Science and Engineering

Semester 1 2021/2022

CZ/CE 4041 – Machine Learning

Kaggle Competition Project

Group 34

Name	Matric Number
Cai Xinrui	U1921389A
Cai Zixin	U1920995C
Chow Weize Aloysius	U1920718J
Shao Yakun	U1920578C

Contribution

Name	Work
Cai Xinrui	Ensemble Learning and Report Writing
Cai Zixin	Exploratory Data Analysis, Prophet and Report Writing
Chow Weize Aloysius	ARIMA, Periodic Factor Method and Video Editing
Shao Yakun	Feature Engineering, LightGBM and Report Writing

Everyone in the group agrees that all group members contributed significantly to the group project. Although we have segregated the contribution, multiple members were involved in each process. Everyone attended every meeting and contributed actively to every stage of the project.

No table of contents entries found.

1. Introduction

This report illustrates the approaches and methods utilised in the Kaggle "**Store Item Demand Forecasting**" challenge for the CZ/CE4041 Machine Learning Course Project. This project consisted of selecting one Kaggle competition challenge, where we were required to submit a proposed solution. Upon submitting the code, Kaggle will provide each team with an evaluation score and overall ranking, using metrics specific to the competition. This report includes the exploratory data analysis (EDA), feature engineering, optimisation, methodology for our suggested solution and the overall competition results.

This report includes a video presentation, the source code used, and a readme file for easy understanding.

Link to Video Presentation: https://www.youtube.com/watch?v=kn0XlWUCgi4&feature=youtu.be

1.1. Problem Definition

The competition we chose, "Store Item Demand Forecasting", consists of a multivariate time series dataset with 10 stores, each with 50 items – giving 500 permutations/time series. This dataset was daily, with 5 years of historical data for training. Participants must forecast the subsequent 3 months of daily sales for each permutation.
The evaluation metric set by Kaggle is the Symmetric Mean Absolute Percentage Error (SMAPE), which represents the accuracy based on percentage errors. The lower the SMAPE, the more accurate the prediction.

Link to Competition description:
https://www.kaggle.com/competitions/demand-forecasting-kernels-only/overview

1.2. Challenge

Unlike other prediction tasks, time series problems are generally challenging, requiring us to predict the future. However, it is also one of the most popular data science challenges companies face, such as demand forecasting, sales, and turnover.

The first challenge we faced for the time series was the single sample size of 1. Since we only have a single historical sample available, it is difficult to predict the future well. Many data techniques such as feature engineering and multivariate correlation analysis have to be applied to strengthen the prediction.

The second challenge was the lack of information regarding this dataset. It only had the store, item, and sales as columns. There was a lack of direct exogenous variables provided and a lack of context. It was not mentioned what market this was in, the type of items sold, and what country. These made feature engineering much harder – since we could not use variables like a country holiday, economic growth, inflation, etc.

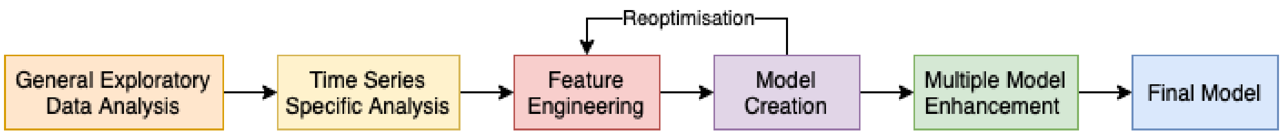
2. Our Approach

To forecast a time series problem, we opted to use various techniques, namely:

1. Periodic Factor Method (Fundamental Time Series Analysis)
2. ARIMA/SARIMAX (Autoregressive + Moving Average Models)
3. LGBM model (Gradient-Boosted Decision Tree) [3].
4. Facebook Prophet (Additive Regressive Model)
5. Ensemble Learning

Each method had a different methodology. For example, ARIMA uses a combination of two models, autoregressive and moving average models. Alternatively, LGBM uses a time series adapted gradient-boosted decision tree model.

We established various steps for our approach:


Reoptimisation
General Exploratory Data Analysis
Time Series Specific Analysis
Feature Engineering
Model Creation
Multiple Model Enhancement
Final Model

Firstly, we will do essential data exploration to discover missing values and general trends and form certain ideas for feature engineering.

Secondly, specific descriptive statistical tests will be done, such as the augmented Dickey-Fuller test for stationarity, the Hodrick-Prescott filter to smooth data, and the Granger Causality test to determine causality between the 500 time series.

Thirdly, feature engineering based on the information from steps 1 and 2 will be conducted on the datasets. The models will then be trained on exogenous variables from our feature engineering. After which, the models are then tested against our validation set to observe discrepancies and compare SMAPE. Fine-tuning and re-optimisation will be done iteratively till the model no longer improves.

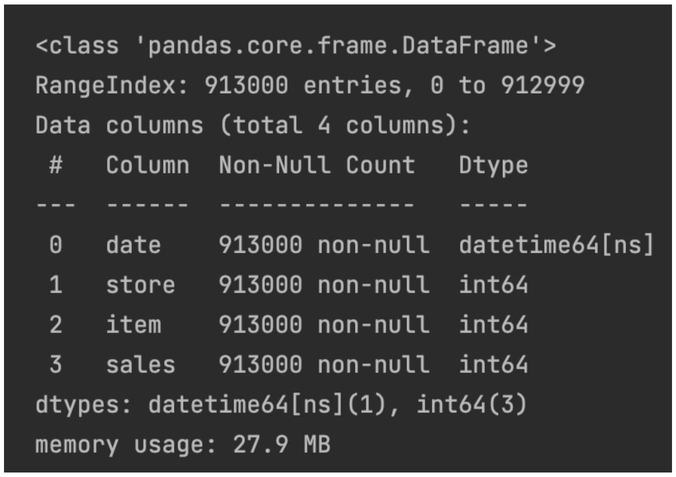
In step 5, we will attempt to improve our final model further by combining different created models from step 4 via ensemble learning.

Lastly, the final model will be chosen from all the models created previously.

3. Exploratory Data Analysis

The competition dataset consists of sales data across 10 stores, each with 50 items, totalling 913,000 entries in the training data and 45,000 in the test data.

There are four columns in the dataset: date, store, item, and sales. No null values are found in the dataset, and no missing data handling was required. All the data are integers (int64), except the date, a datetime object (datetime64).



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 913000 entries, 0 to 912999
Data columns (total 4 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   date    913000 non-null  datetime64[ns]
 1   store   913000 non-null  int64
 2   item    913000 non-null  int64
 3   sales   913000 non-null  int64
dtypes: datetime64[ns](1), int64(3)
memory usage: 27.9 MB
```

Figure 1 Training Data Set Info

3.1. Basic Data Exploration

To understand the nature of the data, this section analyses descriptive statistics, with graphs used to visualise and illustrate the data.

3.1.1. Descriptive Statistic

For the training data set, the data covered from 2013-01-01 to 2017-12-31, and for the testing data set, the 'date' covered the back 3 months of the training data set, which is from 2018-01-01 to 2018-03-31. In addition, the training data set found no null values, so null value analysis is unnecessary.

3.1.2. Overall Sales Distribution

This section analysed sales distribution according to different attributes.

Plot | **Explanation**

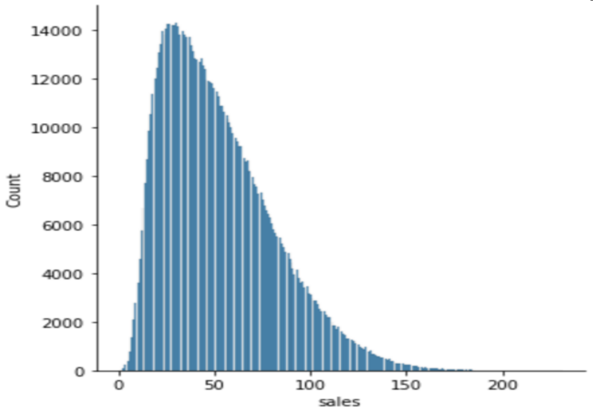
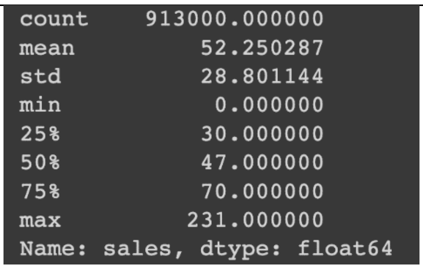


Figure 2 Sales Distribution

Firstly, sales distribution is analysed and visualised via a histogram. The distribution is skewed to the left, which means most sales are on the lower end.

To further analyse the details of the sales distribution, Panda describe() method is used to display the details like mean, std, distributions at 50% etc.



```
count     913000.000000
mean          52.250287
std           28.801144
min            0.000000
25%           30.000000
50%           47.000000
75%           70.000000
max          231.000000
Name: sales, dtype: float64
```

Figure 3 Statistical Details

Secondly, we analysed the sales distribution of each store. A mean of sales is plotted against different stores.

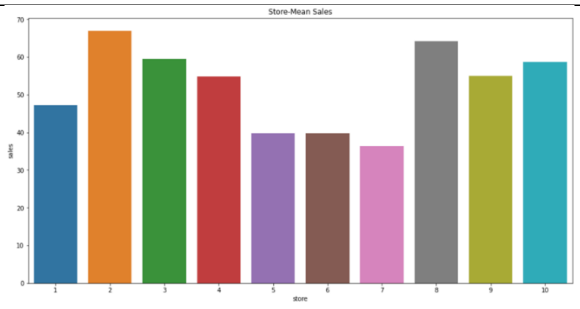


Figure 4 Sales Distribution of Stores

From figure 4, the store with the highest mean sales is Store 2, and Store 7 has the lowest mean sales. The mean sales of Store 2 and Store 8 are above 60. Store 5, 6 and 7 are the 3 stores with the lowest mean sales - below 40.

There does not appear to be a consistent sales trend across stores.

We analysed the sales distribution of each item. Below illustrates the mean sales plotted against different items.

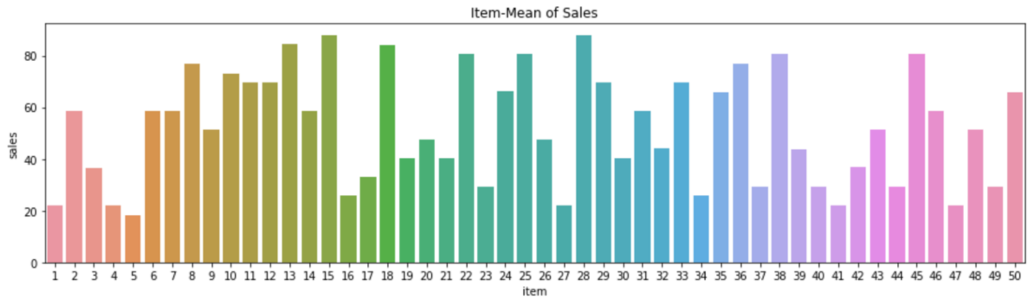


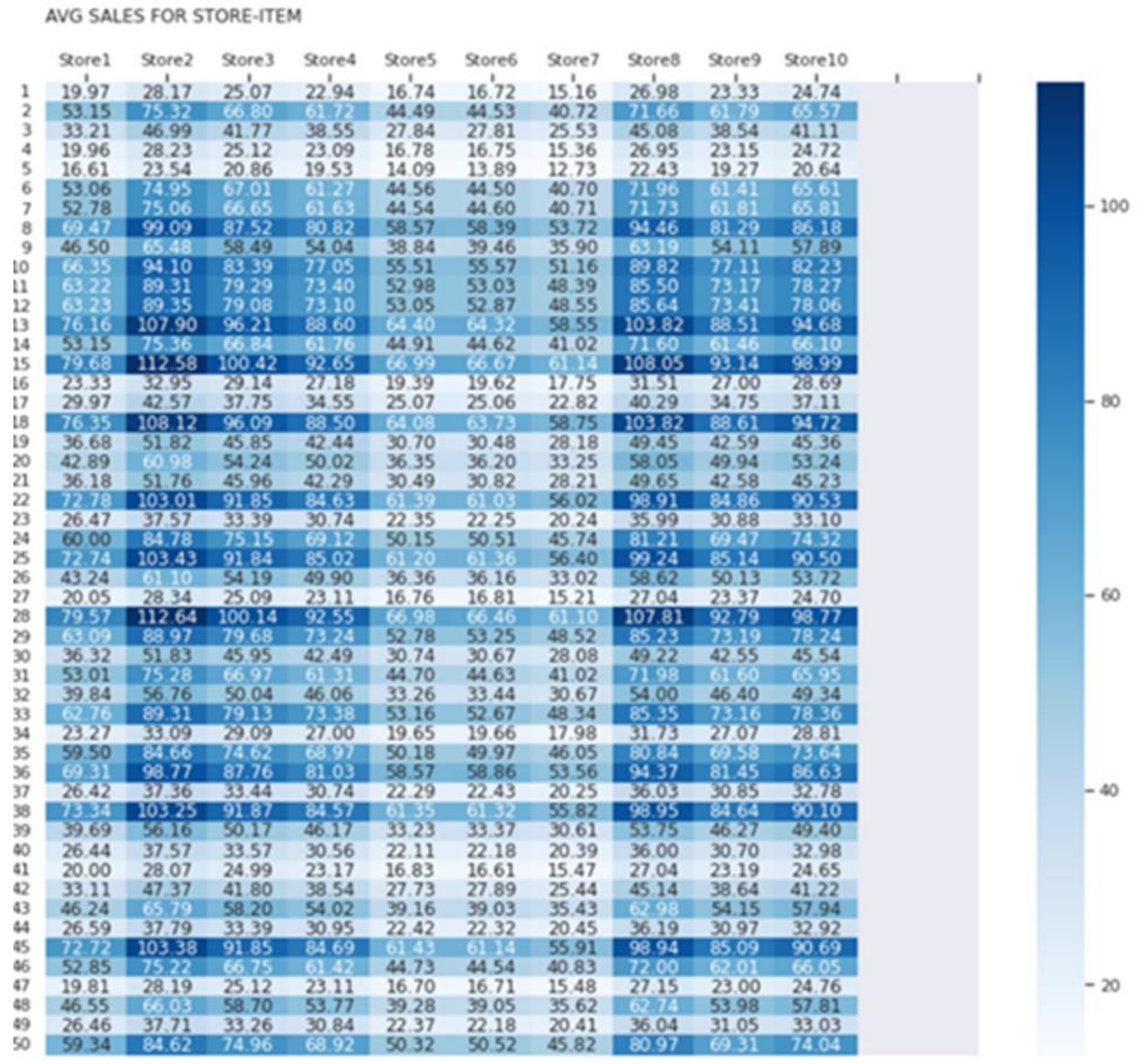
Figure 5 Sales Distribution of Items

From figure 5, the items with the higher mean sales are items 15, 28, 13 and 18, around 80. Item 5 has the lowest mean sales, which are below 20.

3.1.3. Sales Distribution of Store-Item

After analysing the sales distribution of stores and items, a deeper analysis is done across the distribution of store-item combinations. We wanted to observe relationships between the sales and store and items.

A heat map of average sales for each store's item is plotted. From figure 6, the sales of popular items in figure 5 account for a significant ratio of the store sales. In other words, popular items are popular in all stores. Conversely, unpopular items were all observed with low sales volume. This is useful for future analysis, as we can see a consistent trend between the same items across stores.



	Store1	Store2	Store3	Store4	Store5	Store6	Store7	Store8	Store9	Store10
1	19.97	28.17	25.07	22.94	16.74	16.72	15.16	26.98	23.33	24.74
2	53.15	75.32	66.80	61.72	44.49	44.53	40.72	71.66	61.79	65.57
3	33.21	46.99	41.77	38.55	27.84	27.81	25.53	45.08	38.54	41.11
4	19.96	28.23	25.12	23.09	16.78	16.75	15.36	26.95	23.15	24.72
5	16.61	23.54	20.86	19.53	14.09	13.89	12.73	22.43	19.27	20.64
6	53.06	74.95	67.01	61.27	44.56	44.50	40.70	71.96	61.41	65.61
7	52.78	75.06	66.65	61.63	44.54	44.60	40.71	71.73	61.81	65.81
8	69.47	99.09	87.52	80.82	58.57	58.39	53.72	94.46	81.29	86.18
9	46.50	65.48	58.49	54.04	38.84	39.46	35.90	63.19	54.11	57.89
10	66.35	94.10	83.39	77.05	55.51	55.57	51.16	89.82	77.11	82.23
11	63.22	89.31	79.29	73.40	52.98	53.03	48.39	85.50	73.17	78.27
12	63.23	89.35	79.08	73.10	53.05	52.87	48.55	85.64	73.41	78.06
13	76.16	107.90	96.21	88.60	64.40	64.32	58.55	103.82	88.51	94.68
14	53.15	75.36	66.84	61.76	44.91	44.62	41.02	71.60	61.46	66.10
15	79.68	112.58	100.42	92.65	66.99	66.67	61.14	108.05	93.14	98.99
16	23.33	32.95	29.14	27.18	19.39	19.62	17.75	31.51	27.00	28.69
17	29.97	42.57	37.75	34.55	25.07	25.06	22.82	40.29	34.75	37.11
18	76.35	108.12	96.09	88.50	64.08	63.73	58.75	103.82	88.61	94.72
19	36.68	51.82	45.85	42.44	30.70	30.48	28.18	49.45	42.59	45.36
20	42.89	60.98	54.24	50.02	36.35	36.20	33.25	58.05	49.94	53.24
21	36.18	51.76	45.96	42.29	30.49	30.82	28.21	49.65	42.58	45.23
22	72.78	103.01	91.85	84.63	61.39	61.03	56.02	98.91	84.86	90.53
23	26.47	37.57	33.39	30.74	22.35	22.25	20.24	35.99	30.88	33.10
24	60.00	84.78	75.15	69.12	50.15	50.51	45.74	81.21	69.47	74.32
25	72.74	103.43	91.84	85.02	61.20	61.36	56.40	99.24	85.14	90.50
26	43.24	61.10	54.19	49.90	36.36	36.16	33.02	58.62	50.13	53.72
27	20.05	28.34	25.09	23.11	16.76	16.81	15.21	27.04	23.37	24.70
28	79.57	112.64	100.14	92.55	66.98	66.46	61.10	107.81	92.79	98.77
29	63.09	88.97	79.68	73.24	52.78	53.25	48.52	85.23	73.19	78.24
30	36.32	51.83	45.95	42.49	30.74	30.67	28.08	49.22	42.55	45.54
31	53.01	75.28	66.97	61.31	44.70	44.63	41.02	71.98	61.60	65.95
32	39.84	56.76	50.04	46.06	33.26	33.44	30.67	54.00	46.40	49.34
33	62.76	89.31	79.13	73.38	53.16	52.67	48.34	85.35	73.16	78.36
34	23.27	33.09	29.09	27.00	19.65	19.66	17.98	31.73	27.07	28.81
35	59.50	84.66	74.62	68.97	50.18	49.97	46.05	80.84	69.58	73.64
36	69.31	98.77	87.76	81.03	58.57	58.86	53.56	94.37	81.45	86.63
37	26.42	37.36	33.44	30.74	22.29	22.43	20.25	36.03	30.85	32.78
38	73.34	103.25	91.87	84.57	61.35	61.32	55.82	98.95	84.64	90.10
39	39.69	56.16	50.17	46.17	33.23	33.37	30.61	53.75	46.27	49.40
40	26.44	37.57	33.57	30.56	22.11	22.18	20.39	36.00	30.70	32.98
41	20.00	28.07	24.99	23.17	16.83	16.61	15.47	27.04	23.19	24.65
42	33.11	47.37	41.80	38.54	27.73	27.89	25.44	45.14	38.64	41.22
43	46.24	65.79	58.20	54.02	39.16	39.03	35.43	62.98	54.15	57.94
44	26.59	37.79	33.39	30.95	22.42	22.32	20.45	36.19	30.97	32.92
45	72.72	103.38	91.85	84.69	61.43	61.14	55.91	98.94	85.09	90.69
46	52.85	75.22	66.75	61.42	44.73	44.54	40.83	72.00	62.01	66.05
47	19.81	28.19	25.12	23.11	16.70	16.71	15.48	27.15	23.00	24.76
48	46.55	66.03	58.70	53.77	39.28	39.05	35.62	62.74	53.98	57.81
49	26.46	37.71	33.26	30.84	22.37	22.18	20.41	36.04	31.05	33.03
50	59.34	84.62	74.96	68.92	50.32	50.52	45.82	80.97	69.31	74.04

Figure 6 Avg Sales for Store-Item

Next, to find out the relationship between the sales of store-item with the sales of items, a heatmap is potted with the ratio of <store-item sales>/<item sales>. From figure 7, all store-item sales almost have the same ratio as other items. In other words, each store has the same ratio of item sales for all items. It may allude to the fact that the training data set is machine-generated – and this would be useful to create models that picked up such trends.

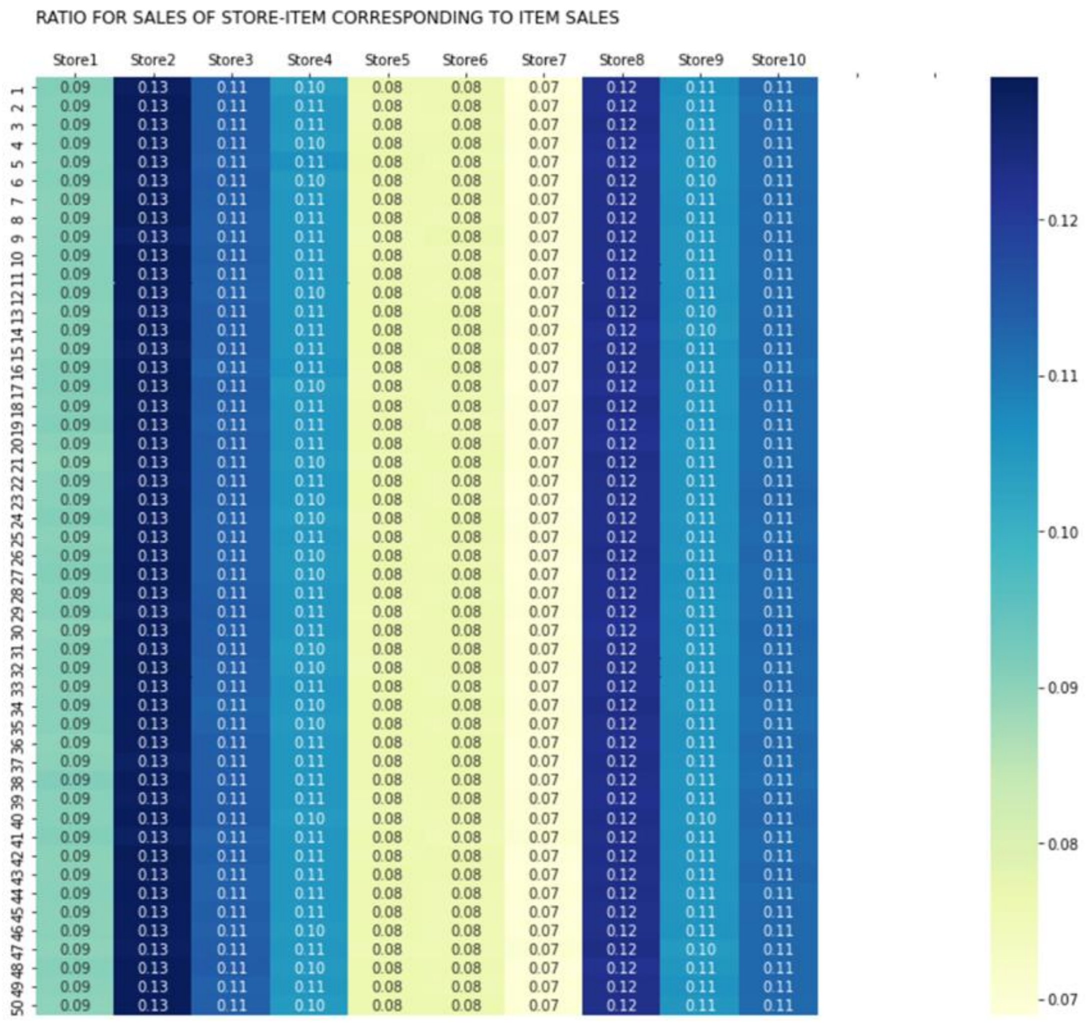


Figure 7 Store-Item Sales / Item Sales

3.2. Time Series Data Exploratory

The components of time series analysis consist of trend, seasonality, cyclical and irregularity [1].

- **Trend**:
  Any divergence within the given dataset is a continuous timeline with no predetermined interval. It could be a Positive, Negative, or Null Trend.
- **Seasonality**:
  Defined as a continuous timeline with regular or fixed interval shifts within the dataset.
- **Cyclical**:
  No defined interval, and movement and pattern are unpredictable.
- **Irregularity**:
  Situations/events/scenarios and spikes in a short time range are examples of irregularity.

The stationarity of the time series is another important term that needs to be analysed because many useful analytical tools and models rely on it. For example, autoregressive moving average models require stationary data to forecast well – differencing to make the data stationary and an inverse differencing to undo the change. Hence, these four time series components and data stationarity are analysed in this section.

3.2.1. *Data Stationarity*

Visualisation of the plotted graph is the most basic and commonly used method to validate stationarity. In this section, datetime aggregated by month against sales is plotted to help observe the trend.

From figure 8, the trend is increasing as sales are increased each year. Obviously, there is a cyclical pattern in the graph. The pattern shows high peak sales in the middle of the year and minor peak sales at the end of the year. Overall sales are regular and with seasonality. Hence, we can conclude that the time series data set is non-stationarity because the graph shows an increasing trend, seasonality, cyclical and regular pattern.

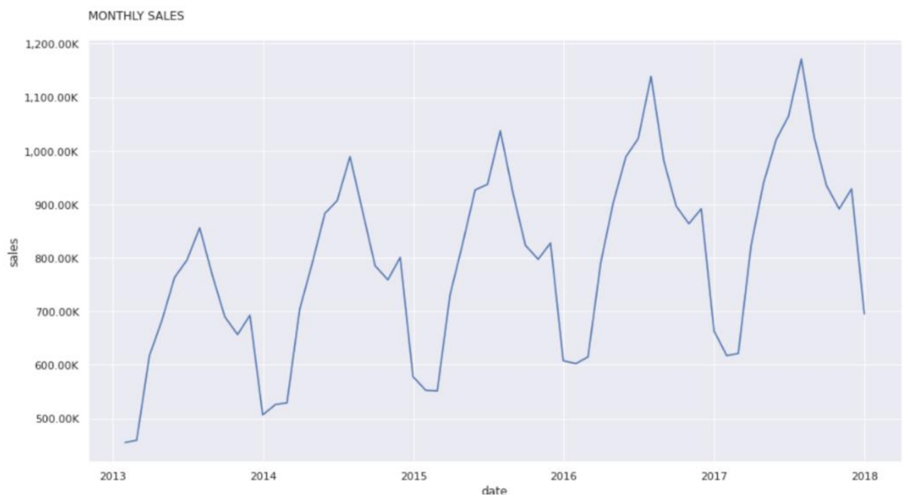


Figure 8 Overall Monthly Sale0073

3.2.2. Store Trend & Item Trend

In this section, monthly sales are grouped by the store and item accordingly to analyse if all stores have an increasing trend. The pattern shown in figure 9 is the same as the overall monthly sales. It is seen that they still follow the same trend across stores and items.

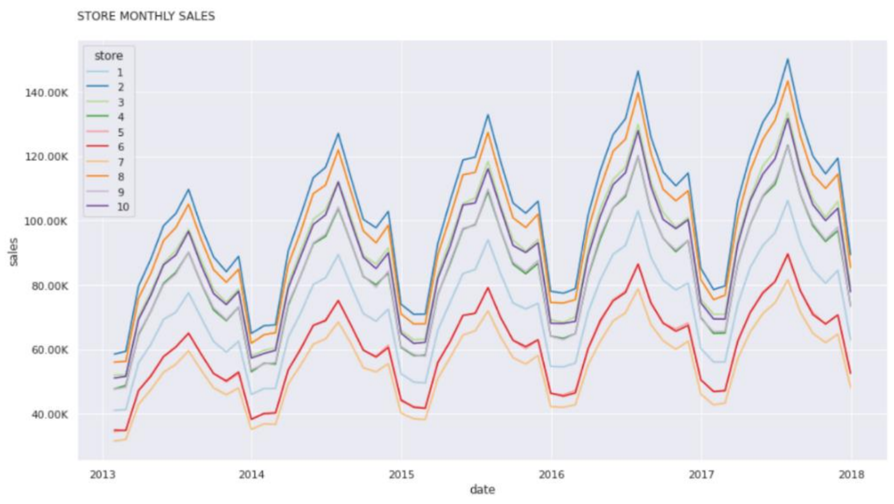


Figure 9 Monthly Sales of Each Store

Figure 10 below plots the monthly sales of items, and the pattern also follows the overall monthly sales.

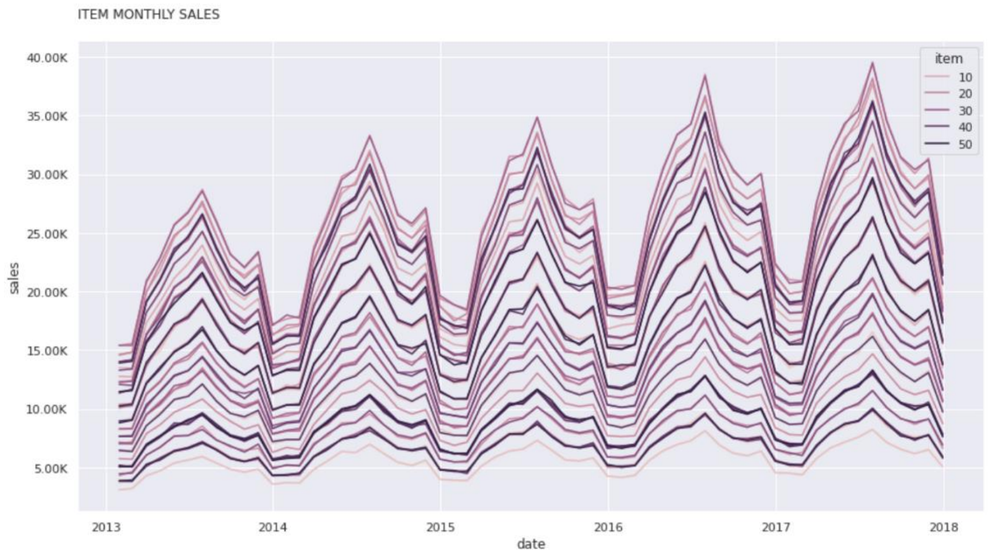
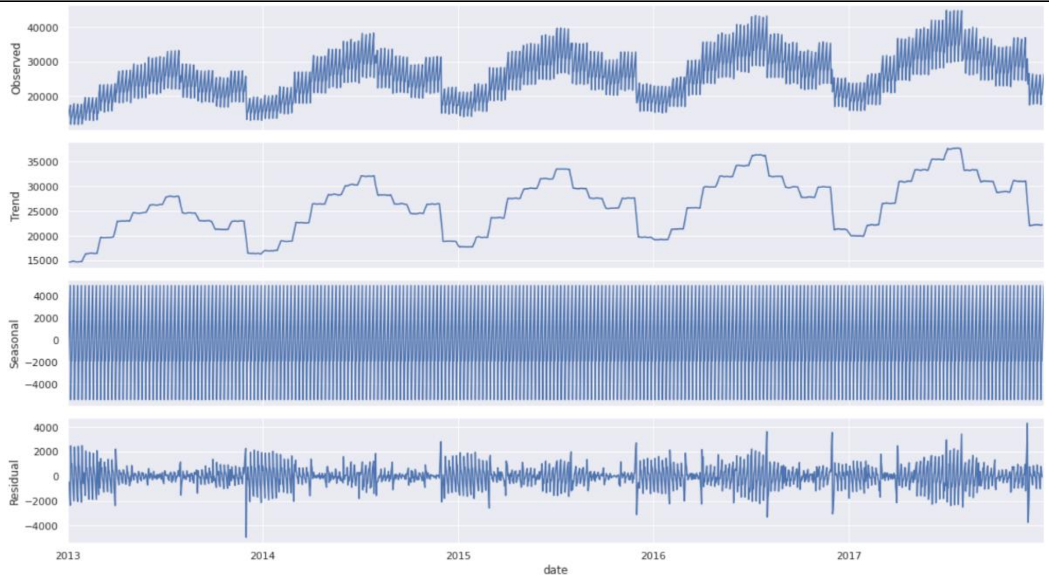
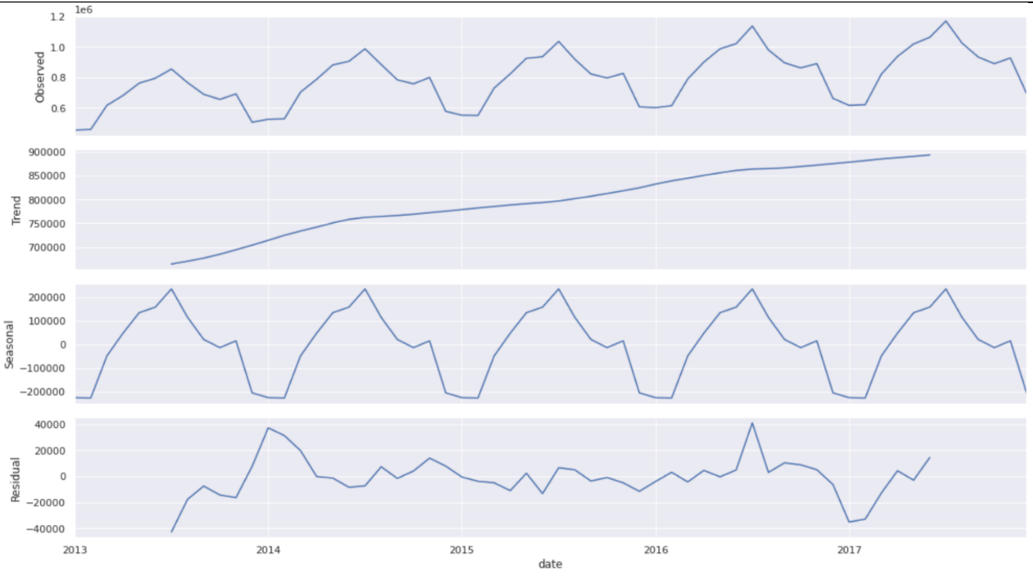


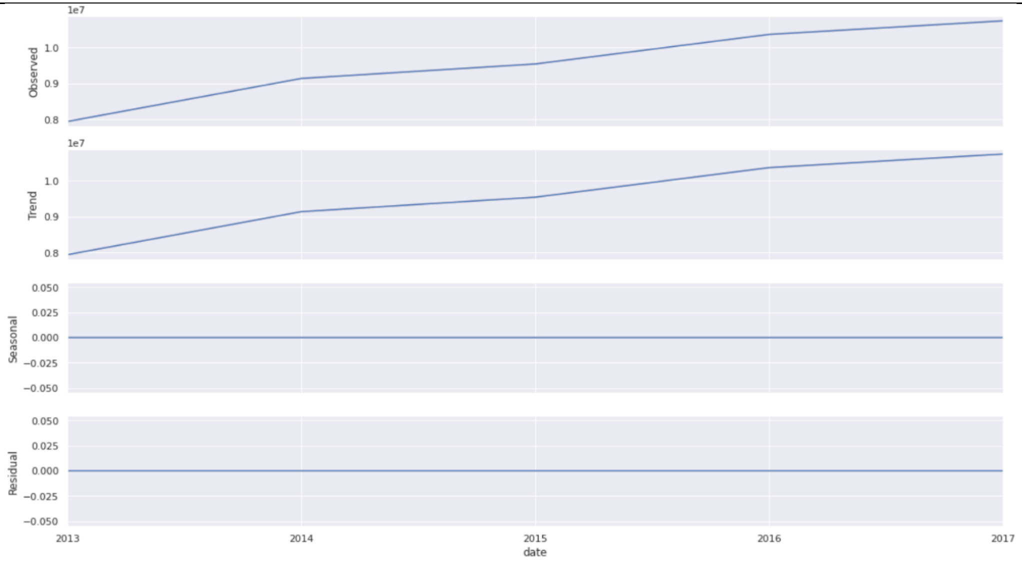
Figure 10 Monthly Sales of Each Item

In conclusion, all the store's monthly and item's monthly sales showed an increasing trend.

3.2.3. Decompose Time Series

This section decomposes the training time-series data set to see various time-series components. Python module named statmodels provides an easy-to-use utility that can get an individual component of a time series and then visualise it. Decomposed daily sales, monthly sales and yearly sales are plotted as below.

Type	Plot
Day	 *Figure 11 Decomposed Daily Sales*
Month	 *Figure 12 Decomposed Monthly Sales*

Year	 *Figure 13 Decomposed Yearly Sales*

In conclusion, from the day, month and year decomposition graph, time series components clearly show the training data set is non-stationary, especially from the decomposition monthly sales graph.

4. Feature Engineering

In this section, we will explain the feature engineering we have done on the dataset. Feature engineering is a technique to manipulate the existing features using domain knowledge to improve performance. Those features we generated here will be used as inputs for our prediction models later.

Overall, we did our feature engineering by categories of data, where 3 relevant types of features are covered:

1) Date-related features
2) Lag/shifting features
3) Smoothing features

We will explain in detail below how we generated the features for each category, followed by a section to cover "feature encoding" to explain how we deal with the multi-categorical features we have.

4.1. Date-related Features

Date-related features was needed because we found that there is a relationship between the dates and the sales (i.e., there is a seasonality element to it). For example, we found that there is a repeated weekly sales pattern, as shown below (data for Jan-2013, where a similar trend was observed during the 4 weeks within the month):

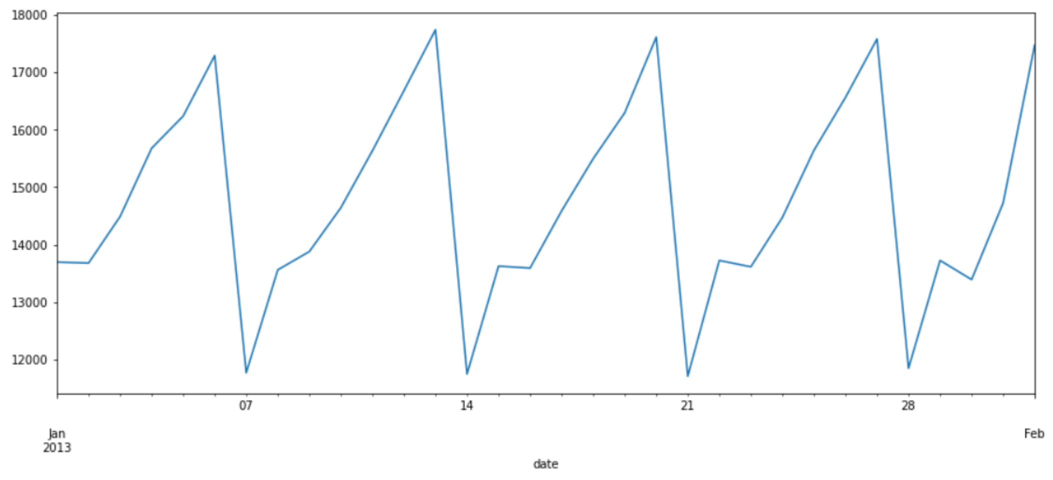


Figure 14, Jan 2013 Sales data, showing a repeated weekly pattern

This suggests that being a specific "day" within a week could have different impacts on the sales value – and this is a similar case such as being a specific "week" in a month, being a specific "month" in a year, etc.

However, in the original dataset, the "date" is presented as datetime strings format, which cannot be used/interpreted in our desired way by models such as LightGBM (one of the models we will be using).

Therefore, it is necessary for us to find a way to **transform the "date' column into interpretable features**.

This was done by transforming the date into categorical variables, in terms of day, week, month, and quarter, so that the model is able to know which date is the specific line of data from. Specifically, we used a few datetime attributes functions to obtain the categorical value, examples as below:

Feature code	Feature generated
df['month'] = df.date.dt.month	Categorical "month" feature, where the values are integers ranging from 1 to 12. The integers represent the respective month the line of record belongs to.
df['day_in_week'] = df.date.dt.dayofweek	Categorical "day_in_week" feature, where the values are integers ranging from 1 to 7. The integers represent the respective day in a week (i.e., Monday, Tuesday, Wednesday etc.)
df['week_in_year'] = df.date.dt.weekofyear	Categorical "week_in_year" feature, where the values are integers ranging from 1 to 52. The integers represent which week the record line belongs to in a year.

In addition to those basic date features, we also added additional ones to recognize whether the day is mid-year or not, whether it is weekday or weekend (note: weekend is defined as Saturday & Sunday), whether it is a holiday or not. We did this because we discovered there is a distinction in sales for specific categories, as shown below:

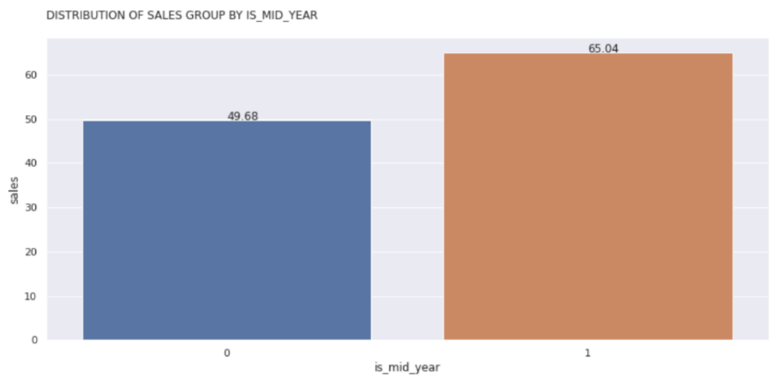


Figure 15, average sales of non-mid-year vs mid-year day (mid-year day has higher sales)

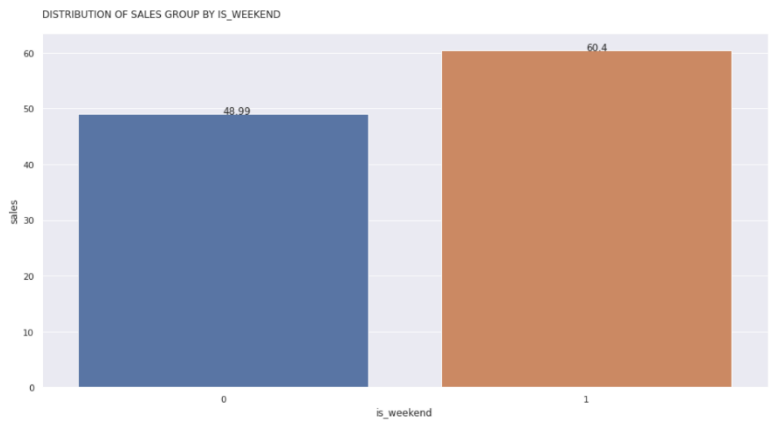


Figure 16, average sales of weekday vs weekends (weekends has higher sales)

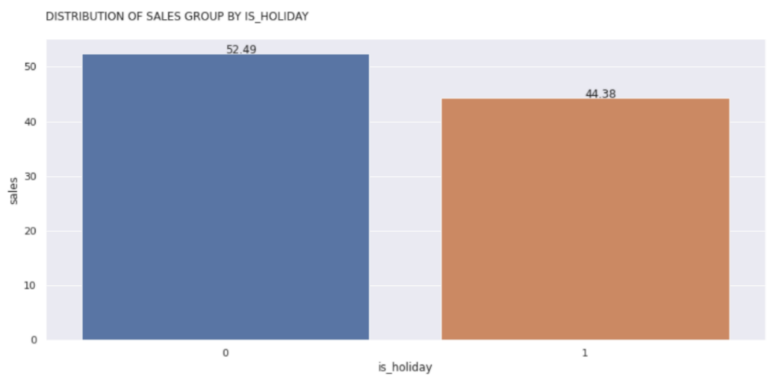


Figure 17, average sales during non-holiday vs non-holiday (unexpectedly, non-holiday has higher sales)

Therefore, the distinctions in sales seen in those categories imply that those categorical factors could be potential factors that contribute significantly to the sales value. Therefore, we found it helpful to include those additional date features as well (i.e., “is_weekend”, “is_mid_year”, “is_holiday”)

4.2. Smoothing features

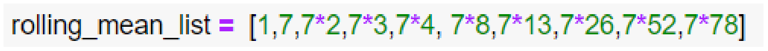
Typically, there will be many fluctuations for time series data if we look at the data in shorter-interval terms. Therefore, smoothing features are needed to reduce such short-term fluctuations and expose the overall patterns, which could be helpful features in a prediction model.

Here, to make our analysis more comprehensive, we constructed 3 different types of smoothing features. Their explanations are as below:

a. Rolling means (or Moving Averages)

This is the essential smoothing feature calculated by the moving average of sales over a specified rolling window value.

For our case, we used 7-day as a base moving average window (since it was the smallest unit where a complete “cycle” in sales was observed), and we constructed the rolling means for: 1 week, 2 weeks, 3 weeks, 1 month (or 4 weeks), 2 months (8 weeks), 1 quarter (13 weeks), half a year (26 weeks), 1 year (52 weeks) 1.5 year (78 weeks):



```
rolling_mean_list = [1,7,7*2,7*3,7*4, 7*8,7*13,7*26,7*52,7*78]
```

Figure 18. Rolling Mean Window List

b. Exponential Moving Average (EMAs)

This is a variation of the basic moving average method, where a more significant weightage is placed on the more recent data points when taking the averages. In other words, it assumes that the more recent data will have greater impacts on the future – which makes sense to a certain extent here since the sales value continues to grow across the years.

For example, to predict next week’s sales revenue, it will be more relevant for us to look at the previous week or previous month's data and estimate a sales level for next week, rather than using the data from a few years ago where the sales level was significantly lower back then.

The formula to calculate the EMA is as below:

$$EWA_{today} = (Value_{today} \times (\frac{Smothing}{1 + Days})) + EWA_{yesterday} \times (1 - (\frac{Smothing}{1 + Days}))$$

Fortunately, we do not need to manually calculate those features, as there is a python operation available (*.ewm()* function in pandas) which we can use directly.

c. Expanding window averages

The expanding window average is another variation of the moving average. In contrast to the simple moving average, which has a fixed rolling window size, the expanding window average only has a fixed starting point. The window for rolling will increase as the data points accumulate over time (and it will eventually cover the entire sample). In this way, the expanding window averages can reflect the average value of all historical data, which is more comprehensive than a simple moving average.

There is no need to specify the rolling window for expanding window averages. Instead, we want to see the expanding averages for different categories of data – e.g., the expanding mean for all Mondays/Tuesdays etc. or the expanding mean for all Januarys. This is because we have previously recognized that those different "categories" of dates come with a distinct in sales, so it will be valuable for us to look at the expanding mean category by category.

Here, we constructed the expanding window averages for: *day in a week, week in a month, week in a year, month in a year, quarter in a year,* and *day in a year*.

4.3. Lag features

Lag is a standard feature engineering method on time-series data, where the data values are shifted forward in time. Lags are very useful because of autocorrelation – which refers to the tendency for the values within a time series to be correlated with the previous copies of itself. In addition to that, lags and autocorrelation are also crucial to autoregressive forecasting models such as ARIMA (one of the models we plan to implement). Therefore, it would be relevant and helpful to construct some lags features here.

The next question is, how much should the lag duration be?

This should be determined by identifying the patterns within the time series – which can be obtained by the Auto-correlation Function (ACF) and Partial Auto-correlation Function (PACF) plots. Here to elaborate a bit more: ACF plots show how the present value correlates with the past, and PACF calculates the correlation between the time series and a delayed version after accounting for the differences described by the intervening comparisons.

From our plots, we observed a strong correlation at every 7 lags (which implies a weakly seasonality), and all lags are outside of the confidence interval (Figure 19). Therefore, we constructed a few lag features in terms of multiples of 7

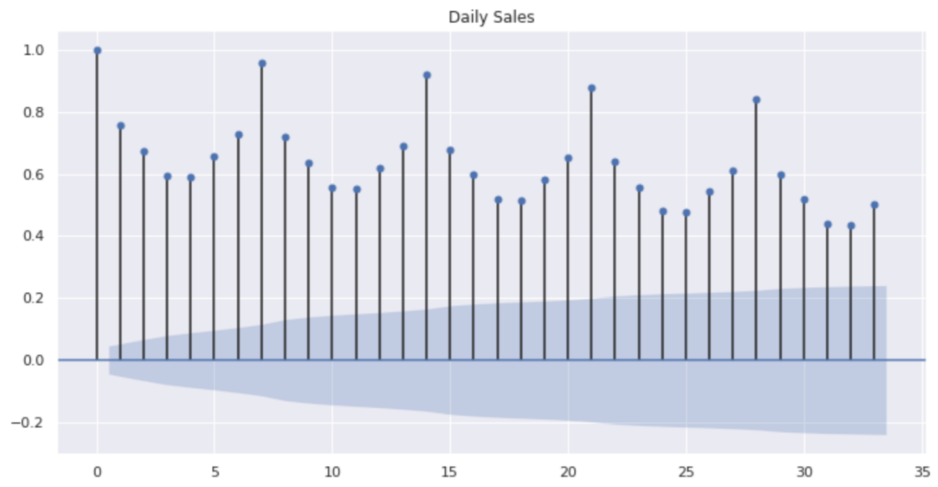


Figure 19, Daily ACF Plot

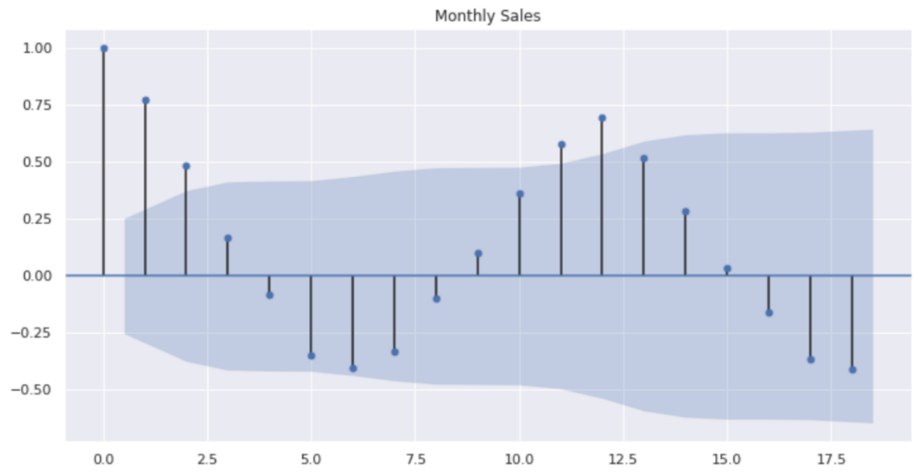


Figure 20, Monthly ACF Plot

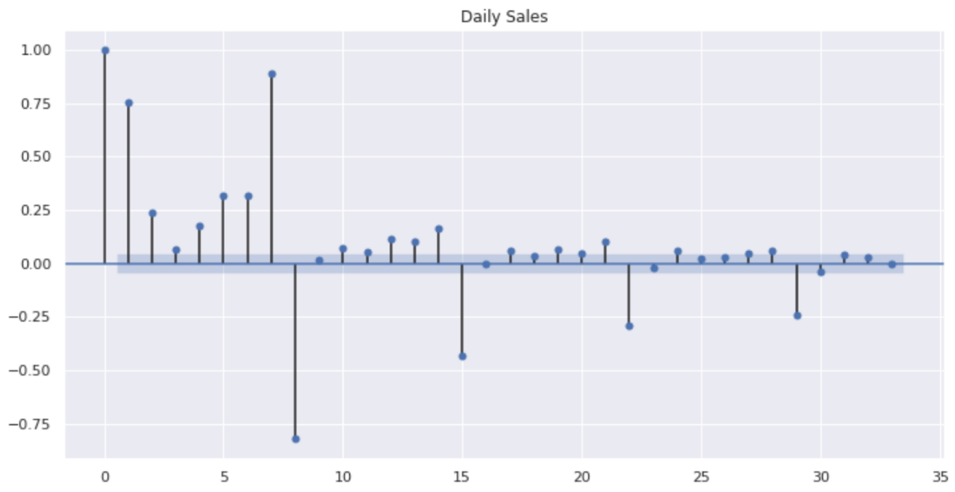


Figure 21, Daily PACF Plot

4.4. Feature Encoding

As the features we have generated contain mostly categorical features, for our models to process those inputs, we need to encode those features such that they are all binary categorical variables.

Typically, the encoding of categorical features was done in One-hot encoding, so our team has utilized the one-hot encoding method to process our multi-categorical features.

Nevertheless, during this process, we realized that some of the features (such as “day_of_year”, “week_of_year”) has a very large number of categories. If we were to use one-hot encoding for them, this would generate 365 additional columns for the day of year, for example, making our dataset and subsequent model complexity significantly larger. Hence, we explored other encoding methods that could potentially mitigate this problem.

Trigonometric Encoding of Variables (Experimental alternative to One-hot encoding)

One method we found was utilizing trigonometric functions like sines and cosines to process those categorical data - this method was proposed for seasonal variables that follow a trend and have some seasonality across the periods, which is suitable for our context. Therefore, we decided to implement this encoding method.

For the trigonometric encoding method, two columns - one sine and one cosine- will be used to represent the different encoded variables instead of categorical variables encoding. This drastically simplifies the model from x to 2-dimensionality. Below illustrates how this method works:

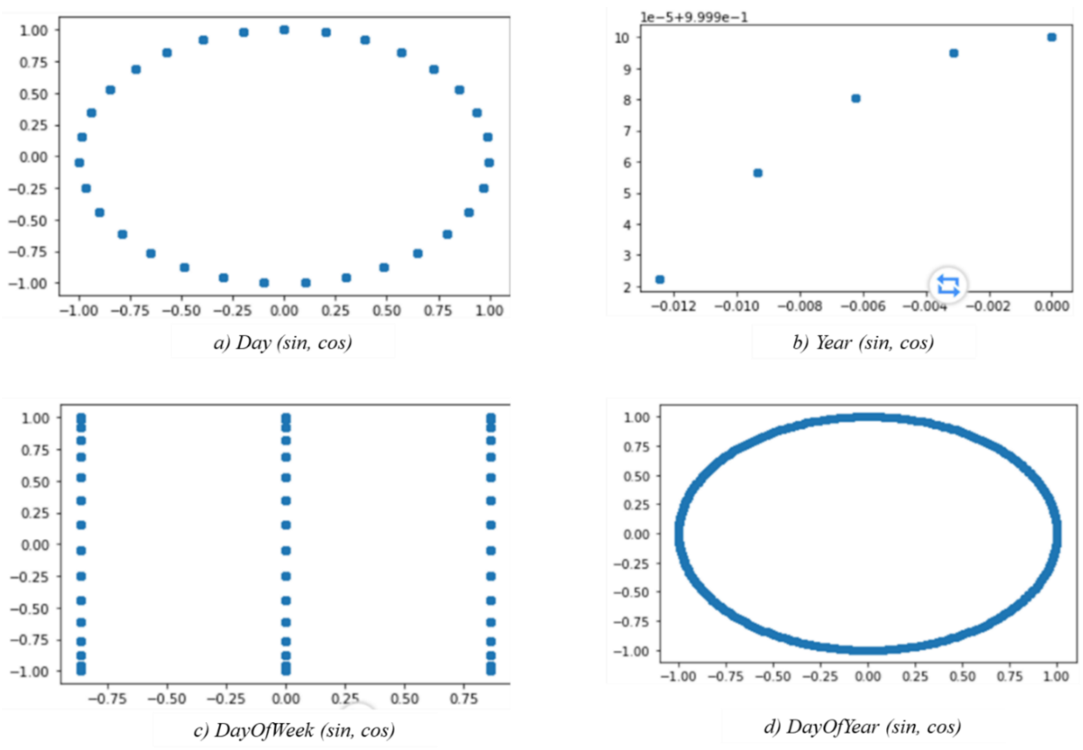


Figure 22. Feature Encoding via Trigonometric Functions

However, even though this method drastically reduced the size of our input dataset, when we used the encoded features to fit into our models, the model results were worsened by a lot compared to if we used the one-hot encoding method.

Hence, unfortunately, we could not use the features encoded through this method as inputs to our models (and we implemented one-hot encoding for all multi-categorical features). Nevertheless, we still think that this is an interesting method we have discovered through doing this project, and it is worth mentioning in our report.

Next, after we have finished the feature engineering part, let us move on to the model implementations in the next section.

5. Models

In this section, we will go into details of the models we have implemented. For each model, we will explain the following content (which may not be in sequence, but all will be covered):

- the rationale for choosing the model in our case (i.e., why it is suitable to use for our dataset/data problem)
- the details of the model's mechanism (i.e., how it works, the algorithms behind it)
- our implementation & the results we got

5.1. ARIMA/SARIMAX Model

a. Why did we choose ARIMA?

Our problem is a univariate time-series problem because the data contains only one time-dependent variable: sales. Variables "store" and "item" are not time-dependent. Therefore, we first deployed a widely used model in solving univariate time-series problems, Autoregressive Integrated Moving Average (ARIMA)[2].

If the data is correlated with its prior values, then ARIMA can be used to conduct regression analysis of the only time-dependent variable on its prior values. To determine whether this problem is suitable for ARIMA, we have plotted a lag plot to observe the relationship between the current values and their corresponding lags.

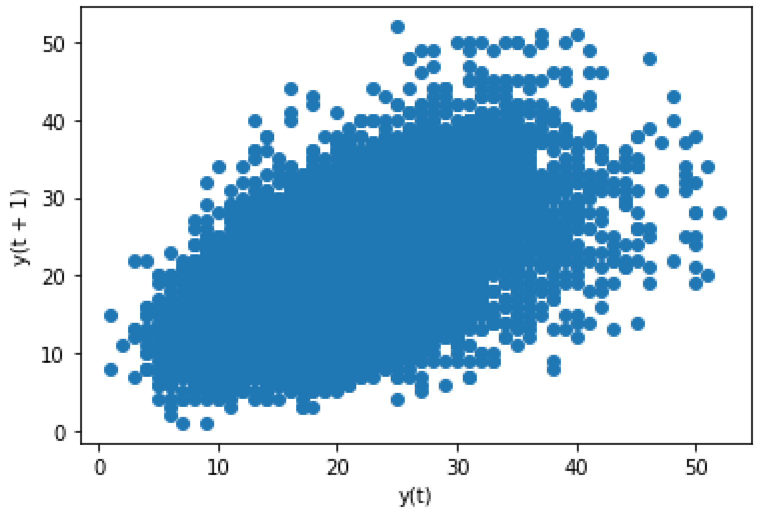


Figure 23: Lag plot of the data in different permutations of store and item

As we can observe from Figure X, there is a certain positive linear relationship between the current values and their corresponding lags. Therefore, we conclude that it is appropriate for us to use an autoregression model in this problem.

b. What is ARIMA? – Algorithm of ARIMA

ARIMA works by conducting regression analysis of the time-dependent variable on its own prior, or lagged, values. Also, one of the important steps in processing the data is differencing, which transforms non-stationary time-series data into stationary time-series data. Stationary time-series refer to time-series with a constant mean and variance over time. There also exist some parts of the time-series not being able to be explained by trends or seasonality, which is known as error. Hence, there are **three** hyperparameters [3] we must determine when deploying the ARIMA model. First being periods to lag for in conducting regression analysis on the past data, which we denote as ***P***. Second being the number of differencing transformations required to transform the data into a stationary time-series, which we denote as ***D***. Third hyperparameter is the order of the moving average model, which we denote as ***Q***.

c. Our Implementation

To determine **P**, we plotted out the Partial Autocorrelation Function plot (PACF) of the 500 time-series and randomly selected 5 of them to be printed on screen to check. The 5 graphs all appeared to be in similar shapes. Since we have observed a sharp cutoff after lag 7, we have determined **P=7**.

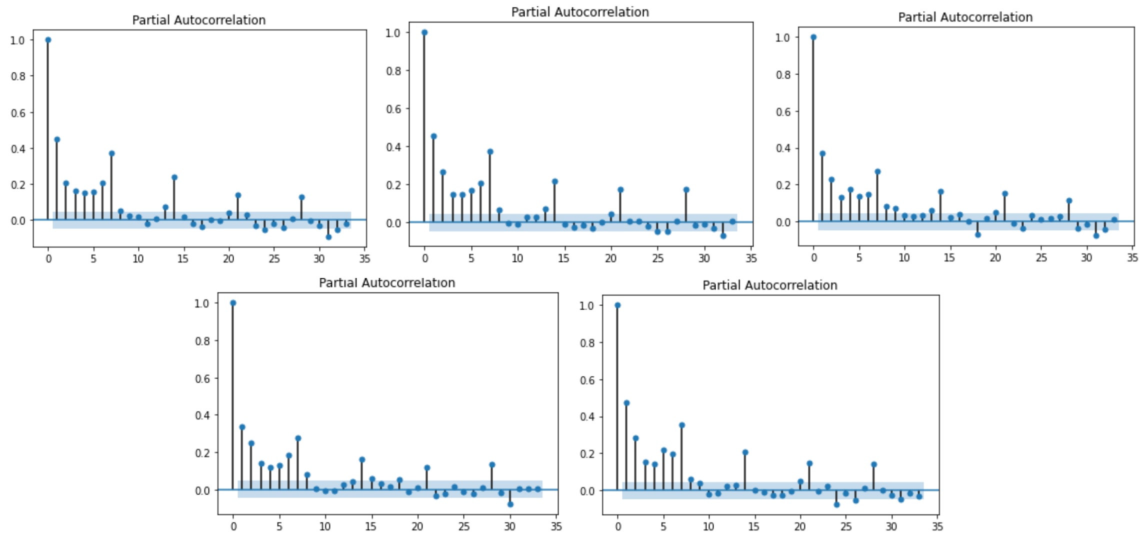


Figure 24: 5 randomly selected PACF plots with similar patterns

To determine D, the minimum differencing transformations required to get a near stationary series with the Autocorrelation Function Plot (ACF) reaching zero quickly, we used statsmodels module available in Python. This module provided us with a function to use the augmented Dickey-Fuller Test to determine the stationarity of the time-series. After passing all the 500 time-series (50 items in 10 stores) into this function, we have found that before we differentiate them, there already exist 434 (out of 500) stationary series and 66 (out of 500) non-stationary series. With an overwhelming 86.8% of data being stationary, we believe that there is no need to differentiate our data, hence D=0.

To determine **Q**, we first tried to use ACF Plot to determine the number of MA (Moving Average) terms. Hence, we have plotted out the Autocorrelation Function plot (ACF) of the 500 time-series and randomly selected 5 of them to be printed on the screen to check. The 5 graphs were all in similar shapes. We have observed a sharp cutoff after lag 0, but all the following points remained above the significant level and decayed at a slow rate.

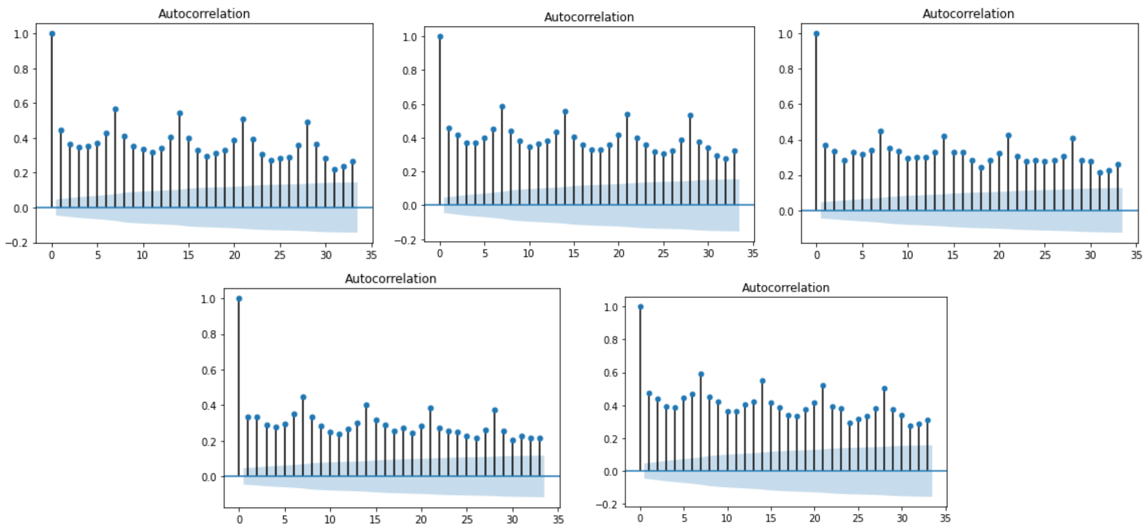


Figure 25: 5 randomly selected ACF plots with similar patterns

Hence, we have done research online on how to determine **Q** in this special case. According to a website published by Robert Nau from Faqua School of Business, Duke University, "<u>If the PACF displays a sharp cutoff while the ACF decays more slowly (i.e., has significant spikes at higher lags), we say that the stationarized series displays an "AR signature," meaning that the autocorrelation pattern can be explained more easily by adding AR terms than by adding MA terms."</u> [4]. The explanation was followed by a recommendation that we should try **Q=0 or Q=1**. As such, we have decided to set the hyperparameter **Q=1**.

After training the data, we did not split the training data into training and validation set, instead we tested the model directly on the final testing set on Kaggle. The two scores we have obtained from using this most basic model of ARIMA were **18.02099** (private, used in ranking) and 20.73437 (public, not used in ranking). Since the scores were calculated based on SMAPE (Symmetric Mean Absolute Percentage Error), the lower the score, the better the performance of the model.

Since we could observe clear seasonality in the daily sales as shown in Figure 11, we will now move on to incorporate seasonality into our ARIMA to improve the performance of the model. As such, we will use the SARIMA model instead of the most basic form, the ARIMA model.

The main difference between SARIMA and ARIMA is that we will need to include another hyperparameter in our model, the seasonality factor, which we denote as **S,** over and above the 3 parameters P, D, and Q which we had determined in the previous part. We did not change the values determined for these parameters in our SARIMA model. Even though we have data for the daily frequency, it is infeasible to use 365 for daily data as the running process is extremely long and requires a lot of memory. Using the supercomputer resources assigned to us on Google Collab environment, it took 30 minutes to just finish one iteration. As such, instead of using seasonality, we will use lag value to simulate seasonality to a certain extent in feature engineering.

We have also explored the inclusion of exogenous variables. An exogenous variable is a variable that is not affected by other variables in the system [5]. Exogenous variables stand for the "X" component in SARIMAX and ARIMAX models, which are also extensions to the basic ARIMA model. As mentioned above, seasonality components cannot be included in our model due to the sheer need for huge computing power. Therefore we will use the ARIMAX model instead of the SARIMAX model.

Five exogenous variables were used in our model, namely day, month, year, day of week and day of year. The traditional way of representing categorical variables in numeric values is to transform a single column of variables into multiple columns using one-hot-encoding. In our model, it is infeasible to do so as we have this variable day of year, so if we went ahead with the traditional method of one-hot encoding, we would have generated 365 extra columns. Since we have 500 time-series to work with in our model, the sharp increase in the number of columns to work with in the database is something that cannot be neglected. Hence, we have found an innovative way online to encode categorical variables using just 2 columns for each variable, regardless of how many values the categorical variable can take.

This method suggested the use of mathematical formulation and trigonometry to represent time-based factors such as the day of week, day of month and day of year, this is because these factors have a common cyclic nature. Taking the sin and cosine of the feature values will create 2 dimensionality features. As such all our 5 exogenous variables were represented by 2 columns each and the increase in dimensionality of our model was reduced greatly.

However, the inclusion of exogenous variables worsened the performance of our ARIMA model instead of improving it. The two scores we have obtained from using ARIMAX were **29.72134** (private, used in ranking, increased from 18.02099) and 38.23257 (public, not used in ranking, increased from 20.73437).

In the next stage of exploration, we have decided to use traditional one-hot-encoding on selected exogenous variables as listed below:

1. Day of Week (Monday, Tuesday, ...)
2. Month of Year
3. Week of Year
4. Week of Month
5. Is Weekend (Boolean)
6. Is Month Start (Boolean)
7. Is Month End (Boolean)
8. Quarter

We have excluded the factor "Day of the year" as it will significantly increase the complexity of our model. After the inclusion of the exogenous variables listed above, the accuracy of the model improved significantly. The two scores we have obtained from using ARIMAX with selected exogenous variables were **13.77655** (private, used in ranking, decreased from 18.02099 obtained from the original ARIMA model) and 16.50861 (public, not used in ranking, decreased from 20.73437 obtained from the original ARIMA model):

Submission
Version 47 (version 47/48)
3 minutes ago by Aloysius Chow
arima

13.77655 16.50861

5.2. LightGBM

The second model we fit onto our dataset is the LightGBM model. In this section, we will first explain what a LightGBM model is, and justify why we chose this model to fit onto our dataset; after that, we will present our implementations of the model & the results generated and end the section with an evaluation of our model.

a. What is LightGBM?

Light Gradient Boosting model (LightGBM) is a gradient boosting model which is based on a decision tree algorithm.

Here, **"boosting"** is an ensemble technique, which means that the model (i.e., LightGBM in our case) is an ensemble of weak learners (decision trees), and the final output of the model is a form of aggregation/combination of the outputs from the individual decision trees.

"gradient" boosting is a type of boosting technique. In gradient boosting. The model will first initialize a constant value (e.g., it could be the average of Y values), then it will compute the pseudo residuals between the actual Y value and the initialized constant. With that, the model will then fit a weak learner to predict the pseudo-residuals and update the model value with the residual error (this is an iterative process). A formal explanation of this gradient boosting process (Hastie, T.; Tibshirani, R.; Friedman, J. H., 2009) is as below:

1) Initialize constant value, $F_0(x) = argmin \sum_{i=1}^{n} L(y_I, \gamma)$

For M iterations:

2) Compute residual errors,

$$r_{im} = -\left[\partial \frac{L\big(y_i, F(x_i)\big)}{F(x_i)}\right]_{F(x)=F_{m-1}(x)},$$

$$for\ i = 1, \ldots, n;\ L(y, F(x))\ is\ an\ input\ loss\ function$$

3) Fit a decision tree $h_m(x)$ to learn the pseudo residuals (i.e., a decision tree in our case)
4) Compute multiplier $\gamma_m = argmin \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma_m h_m(x))$
5) Update the model, where $F_m(x) = F_{m-1}(x) + v \cdot \gamma_m h_m(x),\ where\ 0 < v \leq 1$

Here, we see that within each iteration, the value is updated by incrementing the predicted residual error*a factor v (also known as the learning rate, which is used to avoid model overfitting). The overall error is reduced with each additional round of iteration:

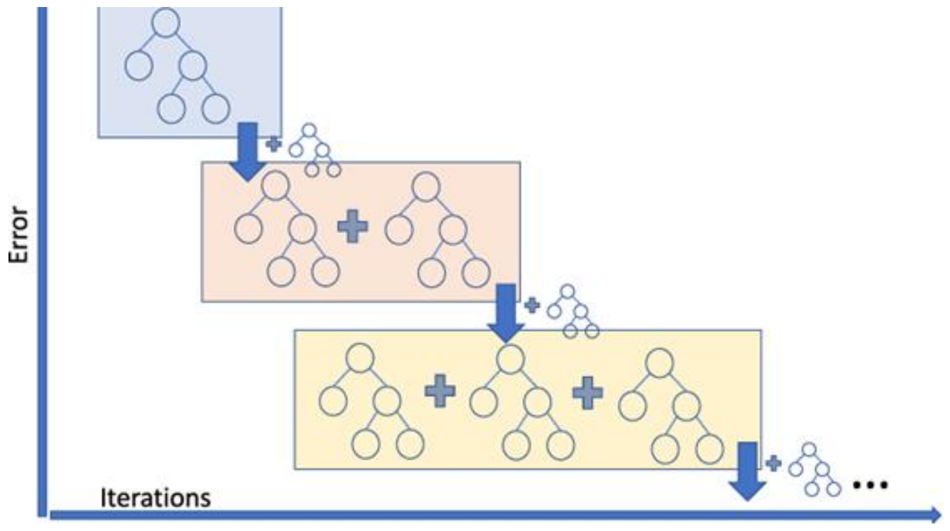


Figure 26. Relationship between error and number of iterations[6]

The final model output will be the initial constant value + the sum of predicted residual errors*learning rate, *v*.

With the above concepts established, the **"light"** gradient boosting simply refers to a gradient boosting method where the individual decision trees (in step 3 above) grow **"leaf-wise"** (best-first method) instead of level-wise (or depth-wise). The leaf that the model chooses to grow is the one with the maximum delta loss. When there is the same number of leaves, the leaf-wise algorithm will tend to have lower loss compared to the level-wise algorithm.

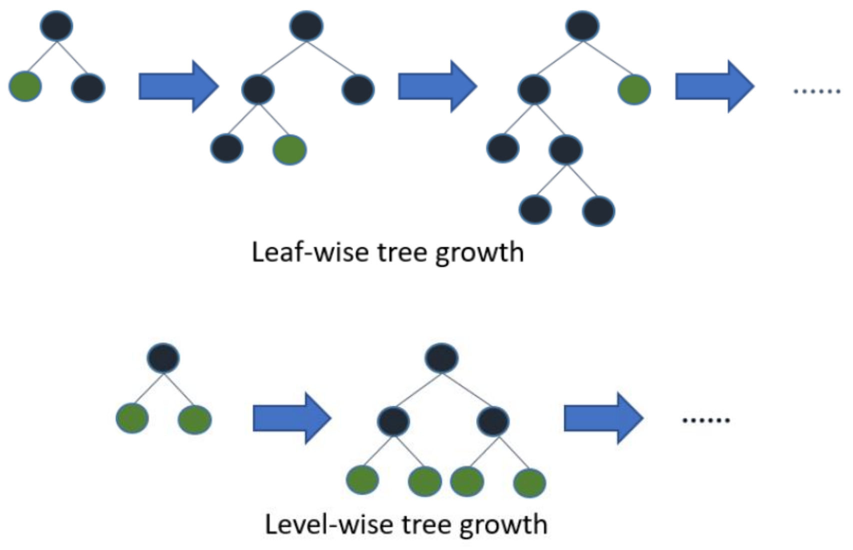


Figure 27, Leaf-wise tree growth vs Level-wise tree growth[7]

b. Why did we choose it?

Even though LightGBM is not a typical time series modelling method (which usually refers to models like ARIMA and VAR), we decided to implement the LightGBM model as an attempt for performance enhancement. This is because as seen from section 5.1, the performance of our ARIMA/SARIMAX model was not good enough in terms of the overall accuracy (SMAPE), so we want to implement another model that can potentially bring in greater accuracy – which could be typically done by a gradient boosting technique. There were a few options such as the XGBoost, CatBoost, and LightGBM. Among those, LightGBM stands out with the following advantages: [8][9][10][11]

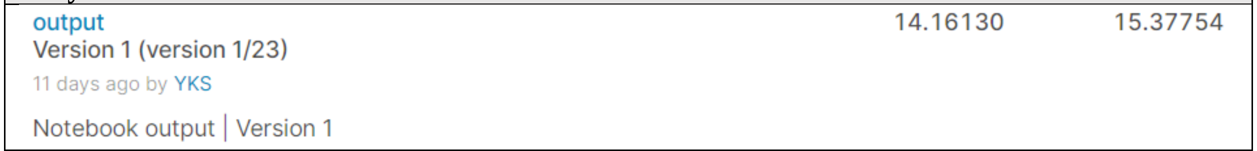
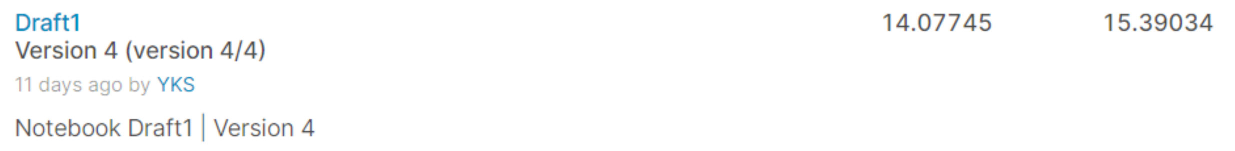
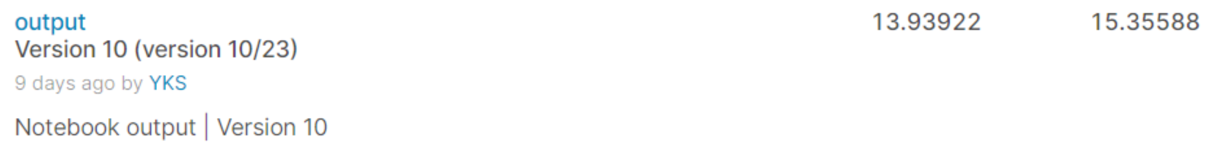
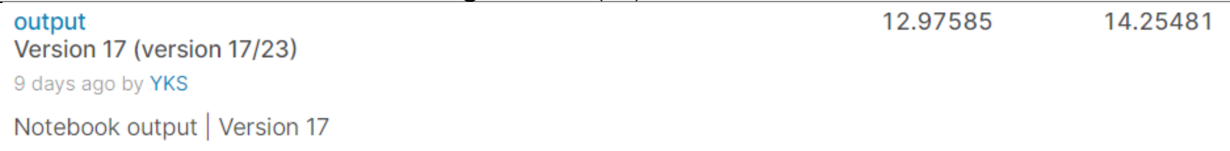
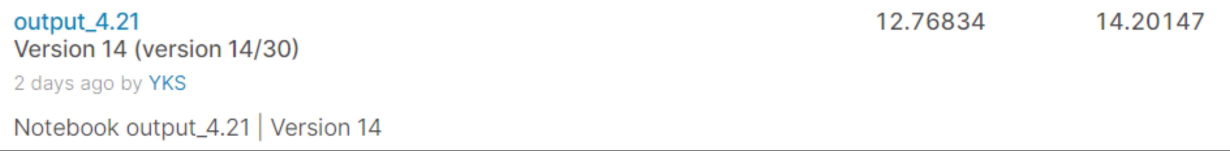
- Faster training speed and greater efficiency
- Lower memory usage
- Better accuracy compared to other boosting algorithms
- Compatibility with large datasets

Considering the dataset we have is relatively large (close to 1000k rows with more than 200 features) and we have limited computation power and memory for all our laptops, LightGBM would be the most suitable option for us to use in our case.

c. Our Model Implementation

The implementation of LightGBM is fairly straightforward here. We first utilize the features we generated as described in section 4. Feature Engineering as the model inputs and obtain the model performance, then proceed to optimize the model hyperparameters.

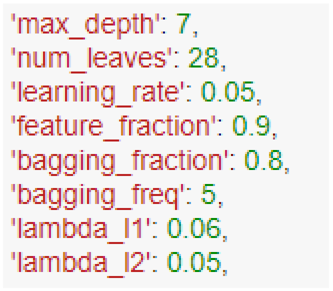
For model inputs, we tried a few variations and observed that the model performs the best when we utilize all categories of inputs, as shown below:

Only use date-related features		
 output Version 1 (version 1/23) 11 days ago by YKS Notebook output | Version 1	14.16130	15.37754
Use date-related features + smoothing features (rolling mean only)		
 Draft1 Version 4 (version 4/4) 11 days ago by YKS Notebook Draft1 | Version 4	14.07745	15.39034
Use date-related features + smoothing features (rolling mean & exponential weighted mean)		
 output Version 10 (version 10/23) 9 days ago by YKS Notebook output | Version 10	13.93922	15.35588
Use date-related features + smoothing features (all)		
 output Version 17 (version 17/23) 9 days ago by YKS Notebook output | Version 17	12.97585	14.25481
(Best version) Use all features, date-related + smoothing + lag/shift features		
 output_4.21 Version 14 (version 14/30) 2 days ago by YKS Notebook output_4.21 | Version 14	12.76834	14.20147

With the optimal set of inputs derived, we then proceed to optimize the hyperparameters, which includes the following:

max_depth, num_leaves, task, objective, learning_rate, feature_fraction, bagging_fraction, bagging_Freq, lambda_l1, and lambda_l2

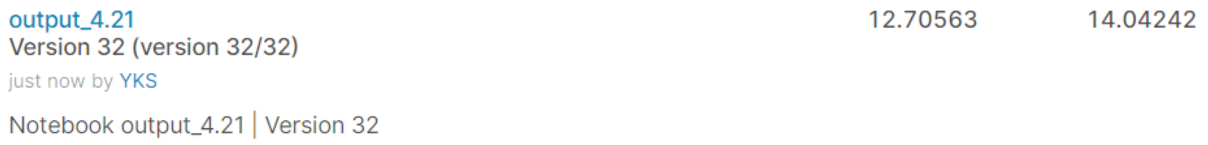
Though using a traditional grid search method would be optimal, due to the limiting computing power and time we have, we opt to use the RandomizedSearchCV method from sklearn package, and obtained the following as the optimal set of hyperparameters:



```
'max_depth': 7,
'num_leaves': 28,
'learning_rate': 0.05,
'feature_fraction': 0.9,
'bagging_fraction': 0.8,
'bagging_freq': 5,
'lambda_l1': 0.06,
'lambda_l2': 0.05,
```

Figure 28. Optimal set of Hyperparameters for LightGBM

With that, we obtained the optimal result from the LightGBM prediction model on our dataset, where we achieved a private score of 12.70563, and a public score of 14.04242:



```
output_4.21                                         12.70563        14.04242
Version 32 (version 32/32)
just now by YKS
Notebook output_4.21 | Version 32
```

***d.	Evaluation***

We have already mentioned the advantages of LightGBM, which is that it is relatively faster, more efficient, and more accurate compared to other gradient boosting models. We do recognize that there are a few shortcomings in our context:

Firstly, though the model is relatively faster, it is still very slow as running one single model takes up at least 3 hours along with high computer memory usage. This greatly limited the number of variations (e.g., different combinations of features) that we could test, which resulted in our final model may not be the most optimal (it is only the best one among all variations that we have tried)

Secondly, similar to the first point, on hyperparameter tuning, we are only able to try for a small set of parameter values using a RandomizedSearchCV, which might not guarantee an optimal. We actually also explored other optimization frameworks such as *Optuna* and *FLAML*. However, similar issues were encountered regarding the time taken for the optimization to run, so we were unable to obtain the most optimal set of hyperparameters in an objective sense due to the practical constraints.

Lastly, we also recognize that LightGBM has inherent problems such as overfitting. However, the issue of overfitting is only likely to occur when the dataset is small. Considering our very large dataset used in this case, as well as the reasonable test set performance, we do not think the issue of overfitting is a concern for our case.

5.3. Prophet Model

a. What is Prophet?

Prophet is an open-source library procedure for forecasting provided by Meta and works well with time series that have strong seasonality and holiday effects.

Prophet is based on an additive regression decomposable model with 3 main components: trend, seasonality, and holiday. The model equation as below:

$$y(t) = g(t) + h(t) + s(t) + et$$

where **g(t)** is the trend factor, there are 2 trend models: the Logistic Growth Model (non-linear) and the Piecewise Linear Model (linear). There are 2 components in trend, **saturating growth** and **changepoints**. Saturating growth means the expected target to grow or fall for the entire forecast interval. And changepoints are the point of time that the growth rate is changed, which can be Christmas sales or Black Friday sales for our case. Prophet allows user to feed in the changepoints manually if it is necessary.

h(t) is the holiday component that represents the effect of holidays that users provide. h(t) can use the Python holidays package or create a holiday data frame that only contains 'ds' and 'holiday'.

s(t) is the seasonality component representing periodic changes such as weekly seasonality and yearly seasonality etc. It is estimated using partial Fourier Sums.

$$s(t) = \sum_{n=1}^{N} \left(a_n \cos \frac{2\pi nt}{P} + b_n \sin \frac{2\pi nt}{P} \right)$$

Where **P** is the period. (ie. 7 for weekly data, 365.25 for yearly data)

N is the Fourier order that represents how quickly the seasonality changes. Faster-changing cycles should set a high Fourier order. But if the high-frequency components are believed to be noise, then the order should be set to a low value to prevent overfitting.

The last term **et** is the error term that accounts for any unusual changes not accommodated by the model.

b. Why did we choose it?

In the Time Series Data Exploratory section, yearly seasonality is observed. Hence, we assume this dataset has a strong correlation with seasonality and, as Prophet works well on forecasting with solid seasonality and holiday effect data set.

It makes time series predictions with satisfactory accuracy with a simple, intuitive parameter that can be tuned easily. Even someone who lacks expertise in forecasting models can use Prophet to make a meaningful prediction in a business scenario.

We would like try Prophet and see if the accuracy of the prediction is improved with the model. And to compare the result with ARIMA/SARIMAX and LightGBM.

c. Our Implementation

In the Time Series Data Exploratory section and Feature Engineering, based on the overall sales distribution graph and ACF plot, we assume there are yearly seasonality and weekly seasonality in this data set. We plot ACF graphs weekly, monthly, and yearly to find an optimal model to find all possible seasonality.

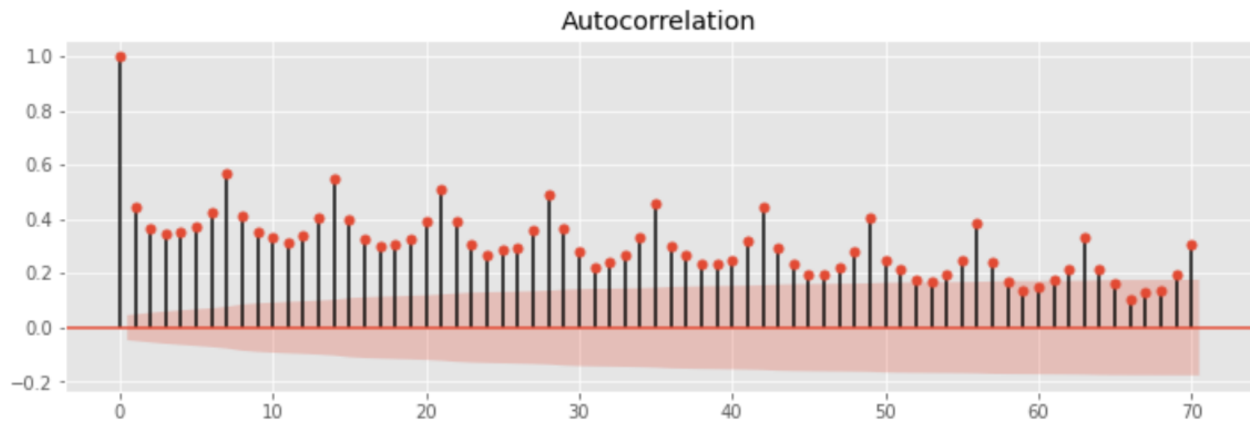


Figure 29: ACF Plot with lags 70

Figure 29 shows that there is a pattern of weekly seasonality. The first, second, and seventh lags had a substantially stronger connection, supporting Feature Engineering's discovery of the weekend connected with sales. And there is no daily seasonality shown in the graph. Hence, we set daily_seasonality to false and weekly_seasonality to true.

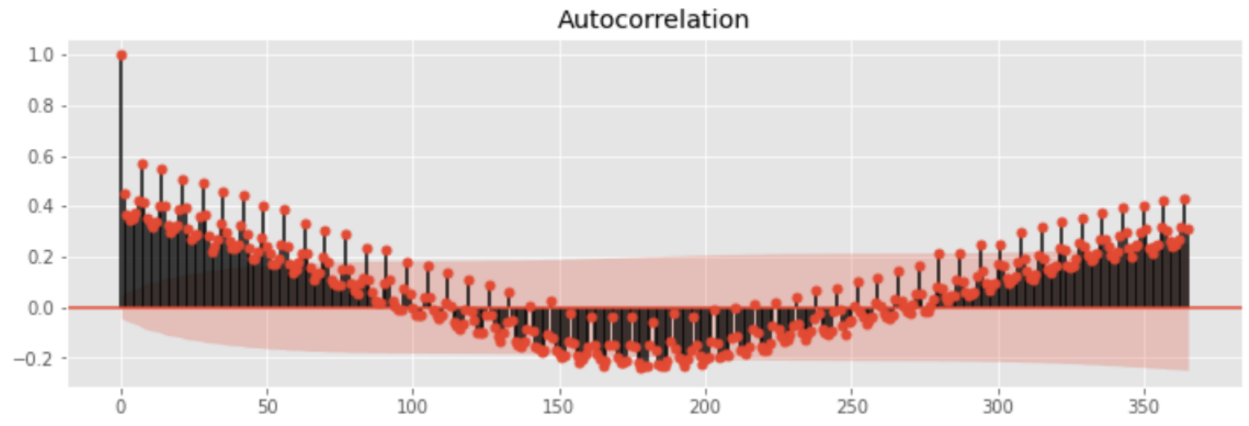


Figure 30: ACF Plot with lags 365

Figure 30 shows autocorrelation over 1 year of consecutive dates. From lag 100 to lag 300, the lags are inside the confidence interval, which can be due to the seasonality of 3 months. Hence, we set monthly_seasonality to true as well.

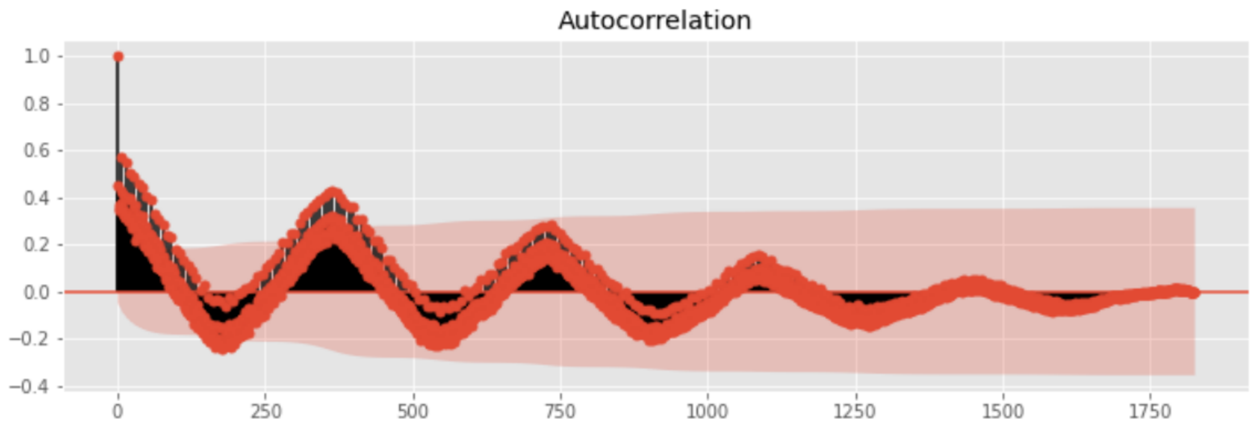
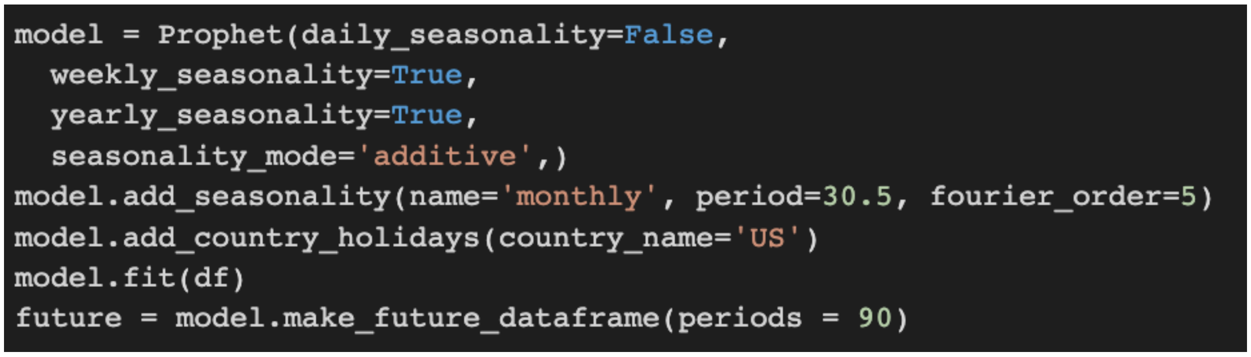


Figure 31: ACF Plot with lags 5 years

To get more practical information we plot the ACF with around 5 years' lags as above graph. After lag 375 (roughly 1 year later), lags are all inside the confidence interval, suggesting that sales very likely a correlation and not a statistical fluke. ACF also exponentially decays to 0 as the lag increases. Hence, we set yearly_seasonality to true.

In Prophet model implementation, weekly, monthly (default Fourier order = 5 is used) and yearly seasonality is set, and holiday component was added by a built-in collection of country-specific holidays method. Here, we set the country to US.



```
model = Prophet(daily_seasonality=False,
  weekly_seasonality=True,
  yearly_seasonality=True,
  seasonality_mode='additive',)
model.add_seasonality(name='monthly', period=30.5, fourier_order=5)
model.add_country_holidays(country_name='US')
model.fit(df)
future = model.make_future_dataframe(periods = 90)
```

Figure 32: Screenshot of Prophet model components set up

d. Evaluation

To validate the performance of Prophet, we use the model and put back the training set and plot the prediction and actual sales data. The predictions are highly similar to the actual data but with less noise.

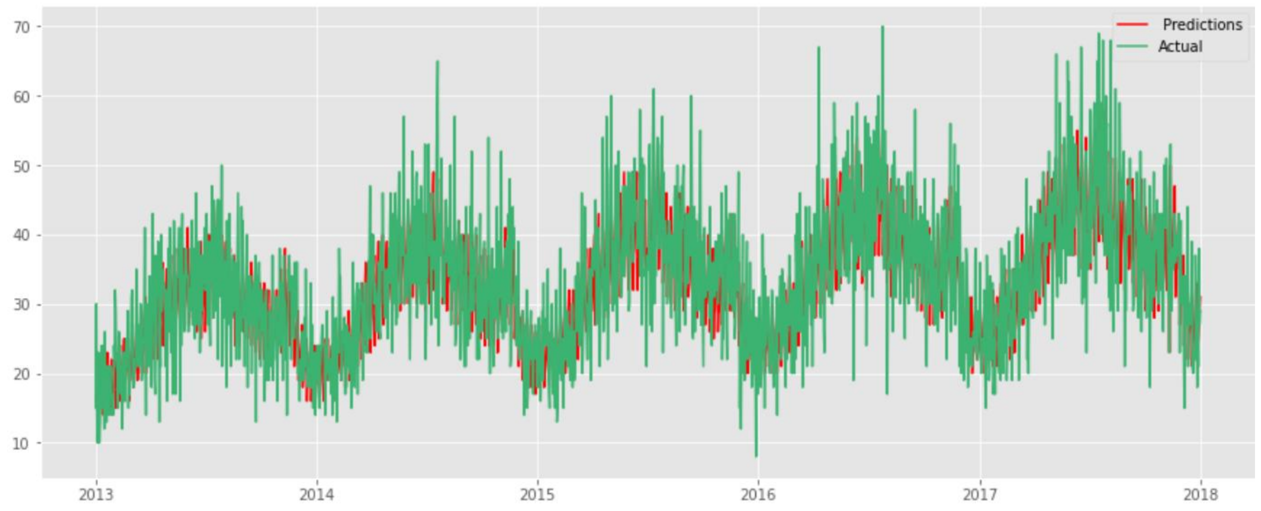


Figure 33: Predictions vs Actual sales over 5 years

The score we get using Prophet model is **13.31264** that is significantly improved compared to ARIMA/SARIMAX, but still worse than LightGBM (12.70563).

output_4.21 13.31264 14.52440
Version 30 (version 30/34)
2 days ago by YKS
Notebook output_4.21 | Version 30

Figure 34: Result of Prophet <13.31>

a. Periodic Factor Method

<u>Interesting observations</u>

After creating 6 exogenous variables (week, month, year, day of week, day of month and day of year) from the date variable, we found interesting observations.

Firstly, when we plot relative sales (relative to the yearly average sales) against the date of the year across stores, we have obtained this graph, which shows that the trends across all different years of observations are identical across stores. Note that a relative sales value of 1.2 implies that the sales on that day are 1.2 x the average daily sales of the year.

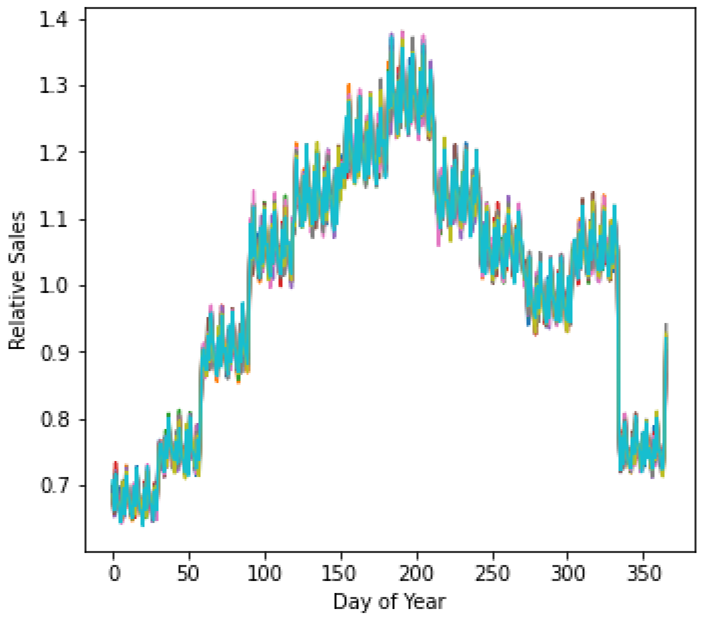


Figure 35: Relationship of relative sales against the day of year

Secondly, when we plot relative sales against the day of the week across stores, we have obtained this graph below, which shows that the trends across all weeks in the database were identical. The Same observation holds true for the variables “Month of year,” and “Week of year”.

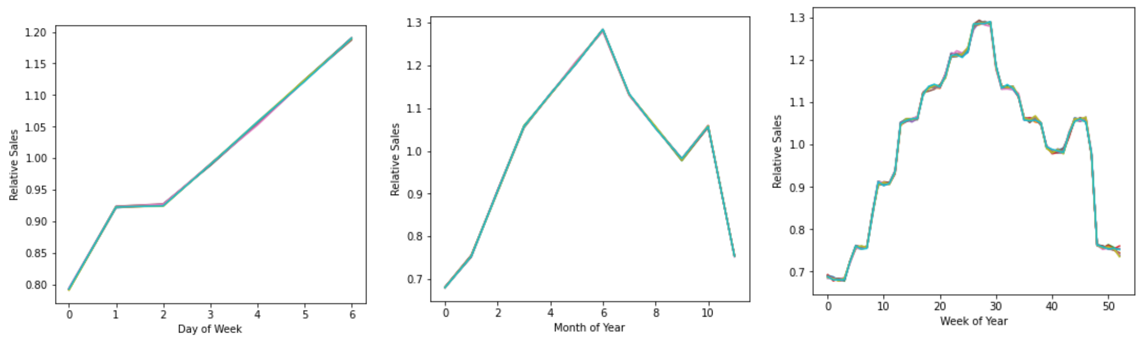


Figure 36: Relationship of relative sales against the day of year, month of year and week of year

As seen from the relationship shown above, there are unique relationships between sales and certain exogeneous variables. However, this unique relationship cannot be represented by simple linear or quadratic functions. In fact, the identical relative sales trends imply that the data is artificially generated with certain variations according to those exogenous variables we have identified. As such, instead of using a general time series forecasting algorithm with exogenous variables to forecast trends, it would be more precise to use a table of relative sales against the exogenous variables to determine multipliers to be applied during prediction.

After obtaining a table of 10 rows (for 10 stores) with 50 columns (50 items) of multipliers, we still need to find out the yearly growth in sales. After plotting the average sales by year, we have 2 options, to adopt a linear growth model, which suggests that the relative sales growth is 1.2132 by year, or to adopt a quadratic growth model, with suggests that the relative sales growth is 1.1509 by year. In the end, we have chosen the quadratic growth model.

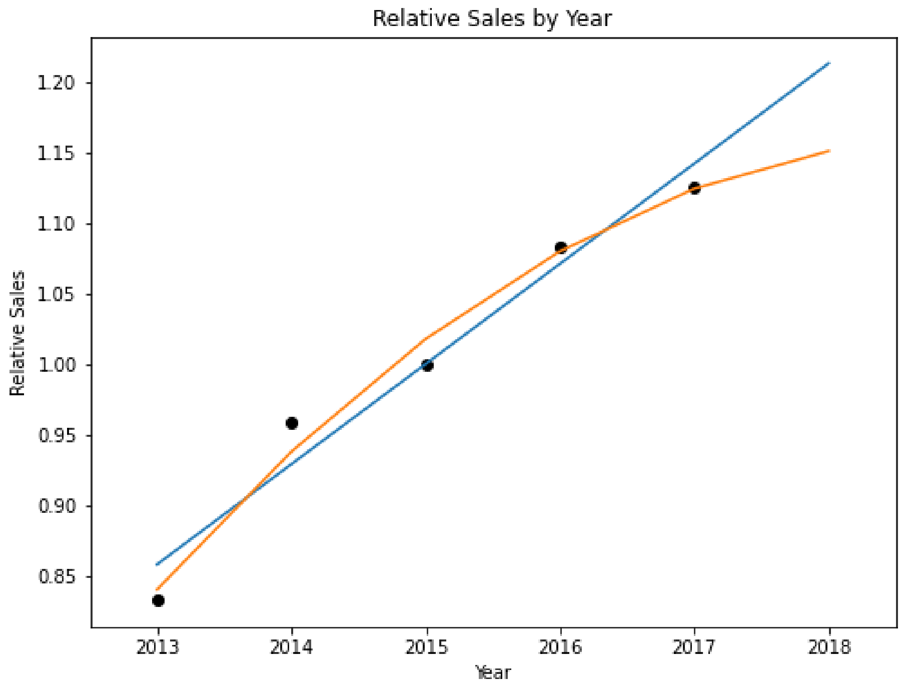


Figure 37: Relative sales growth across 5 years of training data

The model works by first getting the average sales price given an item and store. Note that since each item and store are their own time series, it was better to separate them into their own average sales. After which, we went to multiply the multiplication effects given the exogeneous variables. Lastly, we multiplied it by the growth rate, as observed in the trends across years.

We did an iterative approach to the model and removed exogeneous variables which had the most noise one by one. When we included all values, we got a private and public score of 39.30698 and 75.03988 respectively.

We did an iterative process of removing the least correlated variables sequentially, until the model achieved the best results. We removed day of year (since it appeared to have the most fluctuation). We then got 22.92815 and 39.52614. Finally, we observed the best result when using day of week and month, at **12.62071** and 13.88569.

Adding Constant and Multiplication Factor to Growth

We tried to improve on the quadratic growth model by introducing additional variables to account for tuning effect. In our training phase we tried to use an additional additive and multiplication factor to negate an observed under-forecast in our model. We used an iterative approach and finally decided on a value of 1.009. We then used the improvements on the quadratic fit method mentioned above to fit the result even more accurately. We managed to get a final score of **12.59735** (private) and 13.87908 (public), placing 18th place in the competition out of 461, top 4%:

Submission
Version 48 (version 48/48)
just now by Aloysius Chow
12.59735
13.87908

6. Enhancements of Models – Ensemble Learning

6.1. Our Methodology

We will be working with the 3 output files from 3 machine learning models (ARIMAX, Prophet and Light GBM). Firstly, we will assign trust scores to each of the outputs generated by the three different models, which are inversely proportional to their private score on Kaggle. Then we will take the weighted average of the predicted values from the three models with the trust score as the weight and produce it as the final output. However, we will also consider the standard deviation among the 3 values. If the standard deviation exceeds a predefined threshold, we will boost the trust scores in such a way that the model with the best performance will be assigned an even larger weight and eventually exert a bigger influence on the final output. We have decided to work with 2 types of architecture, and test out which is the better architecture, judging by the private score on Kaggle

6.2. Single Step Boosting

In this architecture, we have explored 2 ways of boosting trust score. One way is to multiply the original trust score of the best performing model (LGBM) by 2 when the standard deviation exceeds the cut-off percentile. The second way is to multiply the trust scores by themselves (square operation) when the standard deviation exceeds the cut-off percentile. We will be running the model on 9 different cut-off percentiles and decide on the best architecture based on the 18 private scores generated.

Cut-off percentile	Final private score (boost x2)	Final private score (boost ^2)
10	12.82309	12.83937
20	12.82401	12.83959
30	12.82397	12.84009
40	12.83368	12.83756
50	12.82157	12.83619
60	**12.81967**	12.83608
70	12.83688	12.83631
80	12.83763	12.83671
90	12.84671	12.84005

We can see from the result table that even though boostx2 with cut off at 60th percentile performed the best; it is not significantly higher than the rest of the results. There also exists no clear trend or relationship between the cut-off percentile and performance.

6.3. Two-Step Boosting

We first analysed the range of standard deviations among the 3 results from the different models and obtained the following results:

Percentile	Standard deviation value
10	0.558929308496788
20	0.8624277011948513
30	1.1566565975427094
40	1.4714463075018631
50	1.818234519257317
60	2.2187512746639673
70	2.700453707939753
80	3.3517303433423176
90	4.335033811947681

From the table above, we can see that the distribution of standard deviation values is very evenly spread out, and the range is limited from 0 to 5. As a result, we have decided to fix the 2 boosting threshold values at the 40th percentile and 70th percentile and test out 2 different boosting methods, the same as what was tested in Single-step boosting.

The result is that we have achieved an improved private score of 12.79664 for boost^2 model and 12.81474 for boostx2 model. Therefore, we can conclude that in the two-step boosting architecture, boost^2 model worked better.

However, even though the score of 12.79664 was much better than scores achieved by ARIMAX and Prophet models, it is still worse than the LGBM model. To understand why we cannot achieve a better score than the best performing LGBM model, we explored further into improvements for our ensemble learning model.

6.4. Improvements to the model

We have plotted the mean predicted sales against the private scores generated by the 3 different models and realized one interesting finding. The private scores are inversely proportional to the mean predicted sales generated by the models. That means, for a better performing model with a lower private score, it had higher mean predicted sales. Therefore, we suspect that all three models were underpredicting the sales. If this were the case, it could explain why we could never achieve a better score than LGBM model because the final output will always be lower than the highest predicted values among the three outputs. As a result, the ensemble learning will always be underpredicting and the final model will never perform better than the best performing model among the three.

Model	Private Score (lower is better)	Mean predicted sales
LGBM	12.70	48.56623844444465
Prophet	13.31	47.71120967542589
ARIMAX	13.78	47.43252228728302

Therefore, we made improvements to our best performing two-step boosting model with boost^2 policy. We assigned different growth factors to our best performing model (LGBM), hopefully making it an overpredicting model and used the two other underpredicting models to balance out this overpredicting effect. We tried 3 different growth factors and attained three different private scores with the best performing one when we artificially increased the mean predicted sales of LGBM model to 50. As a result, the ensemble learning model outperformed the LGBM model. (12.69872<12.70)

Target mean	Growth factor	Private score
49.0	49.0/48.566	12.76225
50.0	**50.0/48.566**	**12.69872**
51.0	51.0/48.566	12.74571

Final output on Kaggle:

output_4.21 12.69872 14.08590
Version 26 (version 26/34)
2 days ago by YKS
Notebook output_4.21 | Version 26

7. Final Model & Evaluation

We used different techniques in our modelling, such as gradient-boosted decision trees, an autoregressive plus moving average model ARIMA, an additive regressive model Prophet, a periodic factor method, and ensemble learning to combine different models. Below illustrates the performance of the final models:

Model	**Private Score**	**Placing (Out of 459)**	**Percentage**
Periodic Factor Method	12.59735	18	3.92%
Ensemble Learning (LGBM, Prophet, ARIMAX)	12.69872	95	20.70%
LGBM	12.70563	95	20.70%
Prophet	13.31264	254	55.34%
ARIMAX	13.77655	280	61.00%

While the Ensemble Learning Model managed to come close (within 0.1 score), the Periodic Factor Method was the best performing model, placing 18 out of 459 participants and achieving top 3.92%.

Pros:

We believed that this method performed very well due to the extremely high correlation between certain time periods such as weeks, even across different stores and items. As such, there is a strong possibility that the sales dataset was machine-generated. Given this correlation, we are certain that forecasting future sales on this data is likely to be very accurate in nature.

Additionally, this model has very low complexity and run time, finishing in less than 30 seconds versus our other models which take many hours.

Cons:

While the model appears to be very consistent, we are uncertain of the performance on non-machine-generated data. As time series in reality do not follow such consistent trends, this model may lead to overfitting and result in worsened performance.

Honorable Mentions

We believe that our ensemble learning model is very flexible and adaptive to data. It currently pales to our periodic factor method by only a score of 0.1. By combining 3 different techniques and models, together with a 2-step boosting ensemble method, we are certain that while our model may not have performed the best for this dataset, it is versatile and will easily adapt to other time series data.

8. Conclusion

In this section, we would like to first give a quick summary of what are the major contents we have covered in our report. What we have covered are as follows:

- Introduction (Problem statement & challenges, as well as our approach to the problem)
- Data Exploration Analysis
- Feature Engineering (include date-related features, 3 types of smoothing features, lag features, as well as the encoding methods we have attempted such as one-hot encoding and trigonometry-encoding)
- Model details (4 models in total)
  - ARIMA/ARIMAX
  - LightGBM
  - Prophet
  - Periodic Prediction Model

    **For each model, we explained in detail about our rationale for choosing the model (i.e., why it is suitable), our implementation procedures, as well as the model performance
- Model enhancement – Ensemble Learning
  - We self-designed an ensemble technique to create a blended model with ARIMA, LightGBM, and Prophet models
- Final model chosen & our evaluation

With the above contents established, for this project, our team strives to:

1. **Solve the problem with the most suitable approach/method, so as to be convincing**

   Where we designed a comprehensive approach to solve the problem, and have a valid justification for every step we took, every feature we created, and every model we implemented
2. **Be innovative and creative**

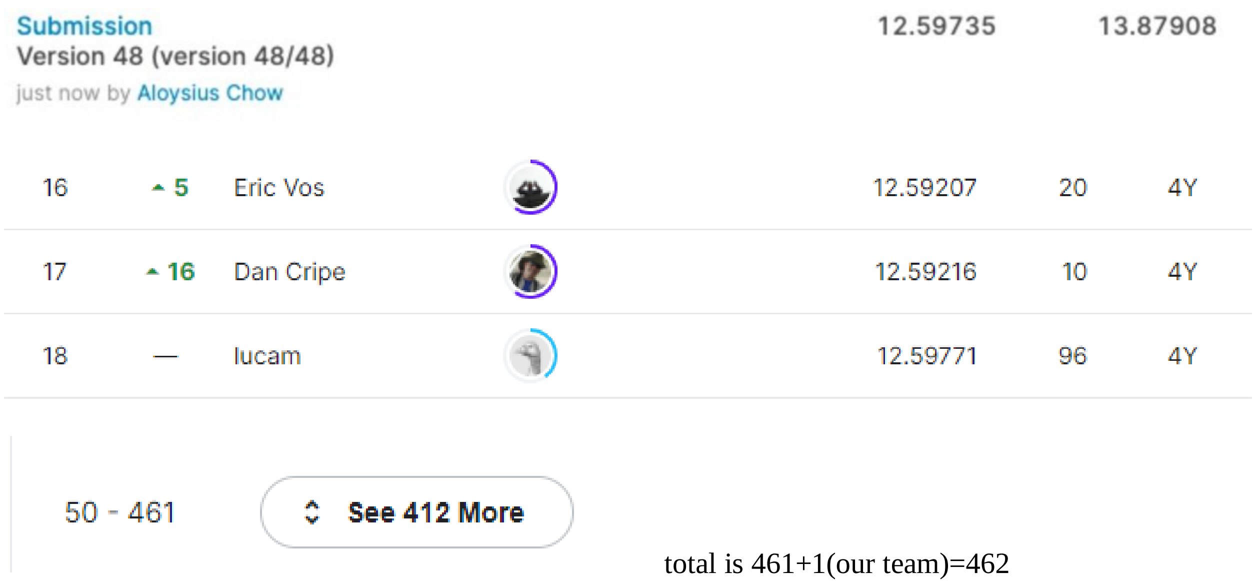
   Where we designed our own ensemble learning method to fit into the context, and also try to incorporate different methods to solve the problem such as the 4th model, periodic table prediction
3. **Keep an open mind, and take it as a learning process**

   While this project is part of the compulsory deliverables for this module, our team does not only to just complete it for the sake of grading, but also want to take it as an opportunity to get practical experiences in implementing what we have learnt in class to solve real-data problems.

   In addition to that, we tried to study lots of online resources to understand the model mechanisms (e.g., LightGBM, Prophet) which was not taught in class, as well as try out for new tools such as the trigonometry. We believe that keeping an open mind and embrace new knowledge will help us improve better.

9. Kaggle Leaderboard Placing

As part of the project requirement, below is a screenshot of our final scores and leaderboard placing, where we have achieved 18/452 placing (top 4%).



Submission
Version 48 (version 48/48)
just now by Aloysius Chow

12.59735 13.87908

16	▲ 5	Eric Vos	12.59207	20	4Y
17	▲ 16	Dan Cripe	12.59216	10	4Y
18	—	lucam	12.59771	96	4Y

50 - 461

total is 461+1(our team)=462

References

[1] “A comprehensive guide to time series analysis,” Analytics Vidhya, 12-Nov-2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-to-time-series-analysis/. [Accessed: 15-Apr-2022].

[2] X. Chen, “A multivariate time series modeling and forecasting guide with python machine learning client for SAP HANA,” *SAP Blogs*, 12-Jul-2021. [Online]. Available: https://blogs.sap.com/2021/05/06/a-multivariate-time-series-modeling-and-forecasting-guide-with-python-machine-learning-client-for-sap-hana/#:~:text=A%20Multivariate%20Time%20Series%20consist,some%20dependency%20on%20other%20variables. [Accessed: 24-Apr-2022].

[3] Sangarshanan, “Time series forecasting - arima models,” *Medium*, 07-Apr-2019. [Online]. Available: https://towardsdatascience.com/time-series-forecasting-arima-models-7f221e9eee06. [Accessed: 24-Apr-2022].

[4] F. S. of Business, *Identifying the orders of Ar and MA terms in an Arima model*. [Online]. Available: https://people.duke.edu/~rnau/411arim3.htm. [Accessed: 24-Apr-2022].

[5] Stephanie, “Endogenous variable and exogenous variable: Definition and classifying,” *Statistics How To*, 05-Apr-2021. [Online]. Available: https://www.statisticshowto.com/endogenous-variable/. [Accessed: 24-Apr-2022].

[6] A. Choudhury, “What is gradient boosting? how is it different from Ada Boost?,” *Medium*, 24-Dec-2020. [Online]. Available: https://medium.com/analytics-vidhya/what-is-gradient-boosting-how-is-it-different-from-ada-boost-2d5ff5767cb2. [Accessed: 24-Apr-2022].

[7] Microsoft, “LightGBM/Features.rst at master · Microsoft/Lightgbm,” *GitHub*, 04-May-2021. [Online]. Available: https://github.com/microsoft/LightGBM/blob/master/docs/Features.rst. [Accessed: 24-Apr-2022].

[8] M. Pace, “Lightgbm for timeseries forecasting,” *Medium*, 19-Jan-2022. [Online]. Available: https://medium.com/data-reply-it-datatech/lightgbm-for-timeseries-forecasting-408971289a12. [Accessed: 24-Apr-2022].

[9] A. Swalin, “CatBoost vs. light GBM vs. XGBoost,” *Medium*, 11-Jun-2019. [Online]. Available: https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db. [Accessed: 24-Apr-2022].

[10] pranavkotak40@pranavkotak40, “Lightgbm vs XGBOOST - which algorithm is Better,” *GeeksforGeeks*, 12-Feb-2021. [Online]. Available: https://www.geeksforgeeks.org/lightgbm-vs-xgboost-which-algorithm-is-better/. [Accessed: 24-Apr-2022].

[11] “What are the pros and cons of LGB model?: Data Science and Machine Learning,” *Kaggle*. [Online]. Available: https://www.kaggle.com/questions-and-answers/103834. [Accessed: 24-Apr-2022].

S. Kumar, “Stop One-hot encoding your time-based features,” Medium, 21-Aug-2021. [Online]. Available: https://towardsdatascience.com/stop-one-hot-encoding-your-time-based-features-24c699face2f. [Accessed: 20-Apr-2022].

Hong.css-1cd9gw4{margin-left:.3em;}生命在于折腾 BINGO, “时间序列预测方法总结,” 知乎专栏. [Online]. Available: https://zhuanlan.zhihu.com/p/67832773. [Accessed: 20-Apr-2022].