School of Computer Science and Engineering

Semester 1 2021/2022

CZ/CE 4041 – Machine Learning

Kaggle Competition Project

Done By:

Aaron Yap Jia Cheng	U1922048H
Nickson Ng	U1921476E
Ryan Lee	U1922245G
Teo Jia Sheng	U1921884J

Contribution

Aaron Yap Jia Cheng	Research + Models Discovery + Prophet
Nickson Ng	Research + Models Discovery + Ensemble Learning
Ryan Lee	Research + Models Discovery + Random Forest
Teo Jia Sheng	EDA + Research + Models Discovery + Arima

Content

1. Introduction

This document is a report written for the team's Course Project for CZ/CE4041 Machine Learning. We were tasked to select 1 kaggle competition and participate with our proposed solution to determine our ranking for the competition. Exploratory Data Analysis (EDA), Methodology of our proposed solution and competition results are included in this report. This report is accompanied by a Video Presentation as well as the source code with a readme file.

Youtube link to our video presentation: https://www.youtube.com/watch?v=UStm6WKxy58

1.1 Problem definition

In this kaggle competition named "Store Item Demand Forecasting Challenge", a 5 years worth of store-items sales data is provided. There are 10 stores with 50 items in each store. We are tasked to make a prediction of the sales of each item in each store for the next 3 months, making this a regression type predictive modeling problem, where we are forecasting a numerical quantity.

The competition uses SMAPE which stands for Symmetric mean absolute percentage error to evaluate the performance of the results. If the SMAPE is 0 means that the actual score and predicted score are the same. The aim is to achieve a SAMPE score that is as low as possible.

URL to competition: https://www.kaggle.com/c/demand-forecasting-kernels-only

1.2 Challenge

The group was relatively inexperienced in dealing with a time series forecasting problem at the start of the project. Problems with the time component makes their problems generally more difficult to handle than traditional machine learning problems. It had its own set of things to consider -

- Standard definitions of time series components, there are a lot of terms unique to time series data and forecasting, such as stationarity, trend, etc, that the team has to learn before understanding and working with the dataset.
- Preliminary Exploratory Analysis, the team has to be able to identify and understand the important attributes of time-series data to best interpret the historical information and ultimately help with the forecast, such as seasonality, trend of data and so on.
- Choosing and Fitting Models, the team had to learn and read up on the newly discovered time series models, which can be challenging at times to understand and choose from for this particular problem

2. Exploratory Data Analysis

The dataset provided is a store-item sales data with 913,000 records in the training data and 45,000 records in the test data provided.
The dataset consists of 4 columns - date, store, item and sales, making the dataset to be a time-series data, where data is collected at different points in time.

A concise summary produced from the columns using data.info() (from the pandas package) indicates that there are no null values in any of the columns and for datatype, only the column date is a datetime64 while the rest is int64.

2.1 Basic Data Exploration

In this section, the report will go through summary statistics and graphs to learn more about the nature of the data and the problem.

2.1.1 Descriptive Statistic

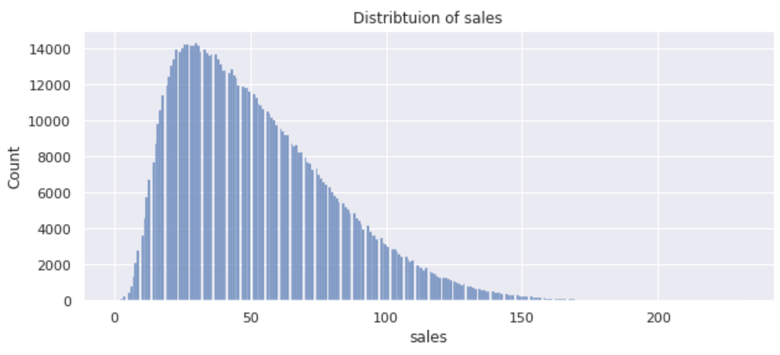
Looking at the boundaries of observed values, the time period of the train data dates back from 2013-01-01 to 2017-12-31 and the time period of the test data dates back 3 months after that, from 2018-01-01 to 2018-03-31. That translates to 5 years worth of training data and 3 months worth of testing data, which is sufficient to check for seasonality and trend in data. The report hypotheses that there will be trends and seasonality as it is frequently observed with data of the sales nature.

It is shown that every store sells all 50 items, from counting the unique ids from store-item combinations.

As there are no missing values for any columns, missing value analysis is not required.

2.1.2. Sales Distribution

The report aims to find how sales distribution varies against different attributes. First, looking at sales on it’s own, a histogram is plotted to visualize the frequency of sales number in the dataset. the distribution of sales at 10th, 30th, 50th, 70th, 80th, 90%, 95%, 99% and max percentiles.

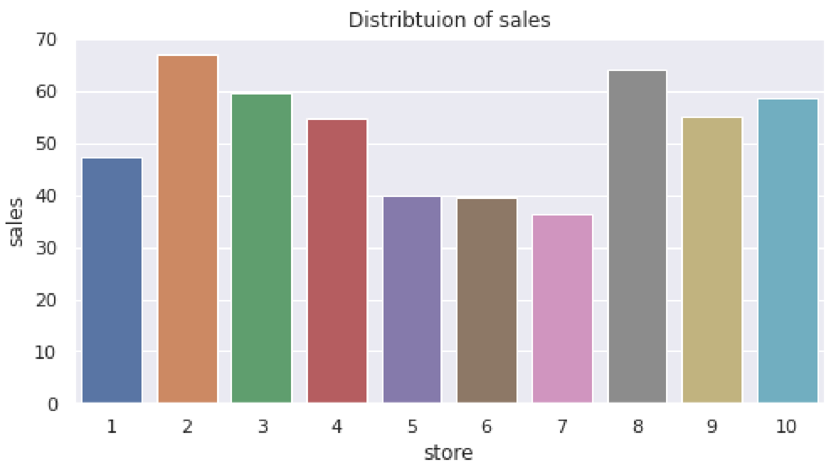


```
count    913000.000000
mean         52.250287
std          28.801144
min           0.000000
10%          20.000000
30%          33.000000
50%          47.000000
70%          64.000000
80%          76.000000
90%          93.000000
95%         107.000000
99%         135.000000
max         231.000000
```

Figure 1

Based on figure 1, sales follows a Positively Skewed Distribution - The distribution is skewed to the left, indicating that the majority of sales are on the lower end.

Next, the report seeks to find out the sales distribution with regards stores and items - find out which stores and items have the highest and lowest sales mean in the data.



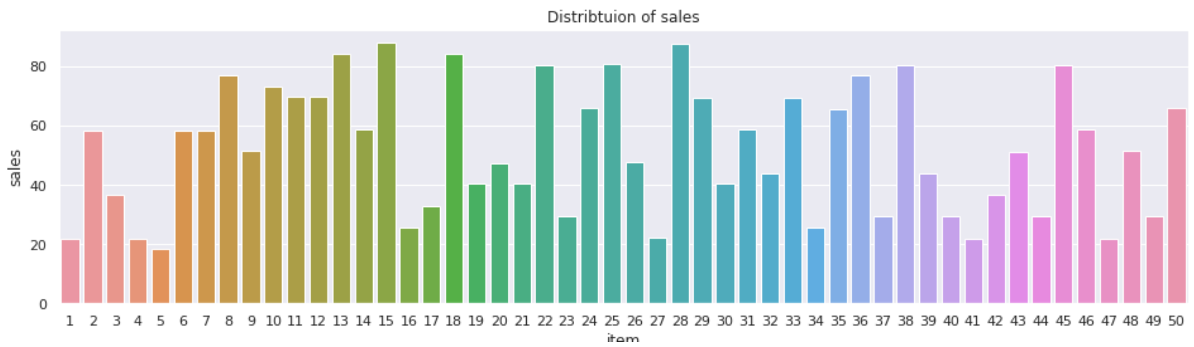


Figure 2

As observed in figure 2, store 2 has the highest mean sales while store 7 has the lowest mean sales and items 15 and 28 have the highest mean sales while item 5 has the lowest. This indicates that there is a noticeable effect that stores and items have with sales.

2.1.4 Store and Item Combination Sales Distribution

Next, The report wants to find the variability of stores and items, in other words, whether some stores sell more of a certain item. To visualize this, the dataset is first grouped by item and store with the total sales of each item is calculated. Each store, with their own sale value of the item, is a percentage of the total sales and the percentage is then plotted on a heat map shown in Figure 3.

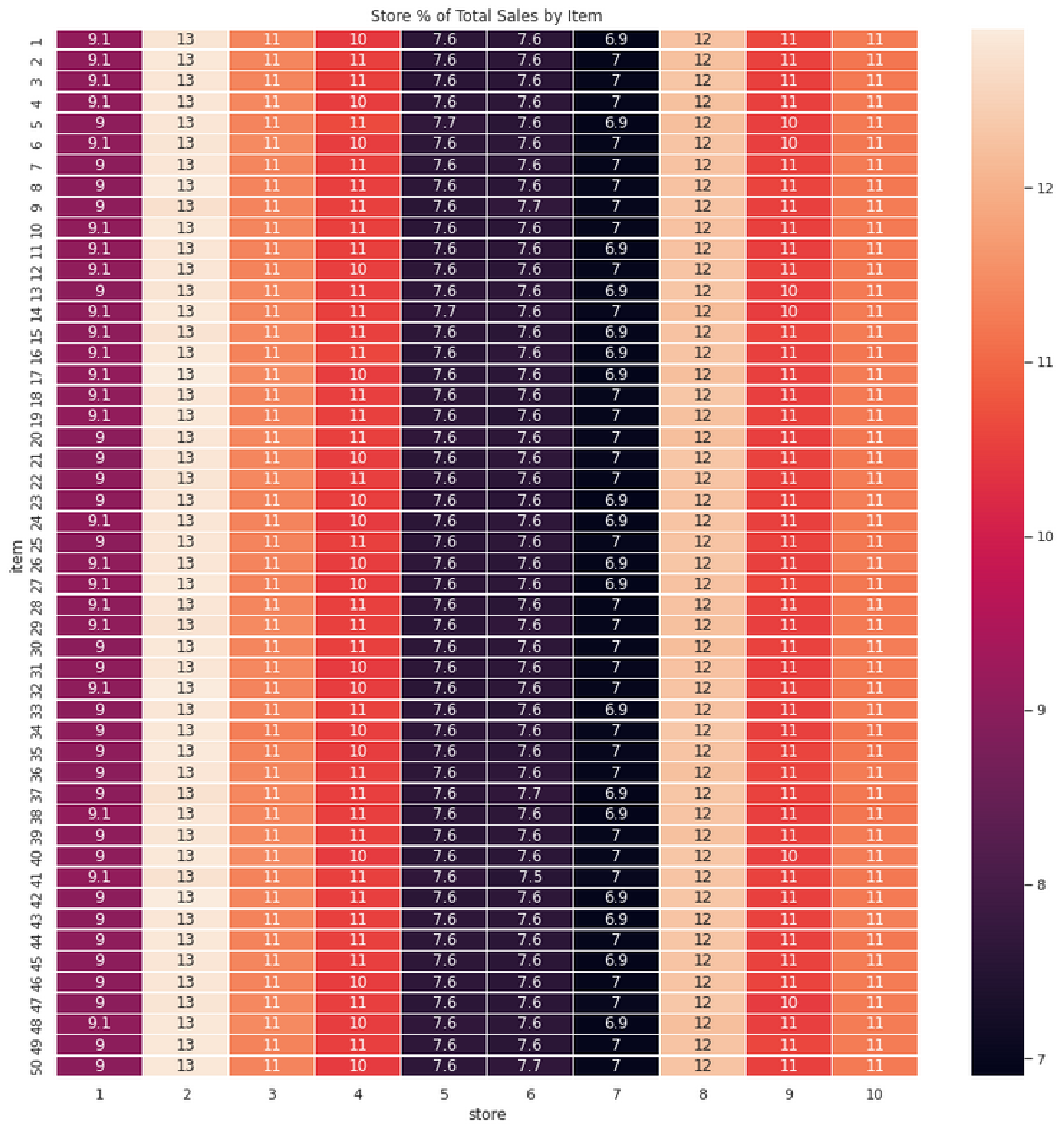


Figure 3

As observed in figure 3, all stores have sold generally the same percentage of each item, meaning that, there are no stores selling more or less of any items than other stores.

2.1.5 Characteristics of the Data

The dataset seems to be simulated data. The data is relatively clean with no empty or extreme values. Store & item combination sales distribution suggests that all stores sell a similar percentage of items consistently throughout 5 years, which is unlikely in a real store chain scenario. This rigidness will be taken into account when selecting a model, as some models, such as probabilistic models work better on truly random behavior while others, such as wiggly models work better on highly determined data.

2.2 Time Series Data Exploration

Time series data are composed of these components - trend, seasonality and residue. Trend represents the increasing or decreasing pattern observed over a period of time while seasonality represents a cyclic pattern - a similar pattern that repeats after a certain interval of time. Residual components are derived from the difference between the observed values and predicted values.

In this section, the report looks into the growth and trends of overall sales using data visualization and time series decomposition.

2.2.1 Overall Sales Trend

Overall sales were aggregated by days for the entire 5 years of train data and plotted on the graph in the figure below.

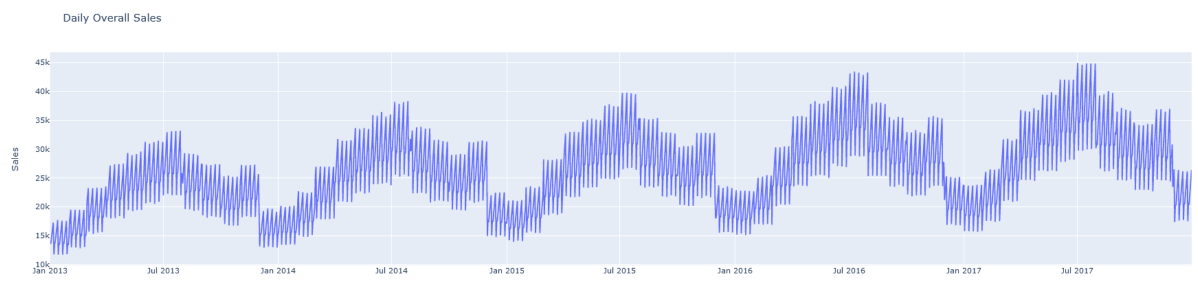


Figure 4

There is an obvious underlying increasing trend on the overall sales. This suggests that sales were increasing each year. A cyclic pattern is observed in the figure, with a low point at the beginning of the year and a high point nearing the middle of the year. This might be caused by the correlation festive seasons have with sales. The high points seem to be increasing, suggesting that this time-series is multiplicative. In addition, seasonality shows that the series is definitely non-stationary.

Time series decomposition is a method to derive more precise information from time series. It decomposes the series into its various components, overhead, trend, seasonal and residual. For this project, it is implemented in the project through the use of the seasonal_decompose() method from the statsmodel python library.

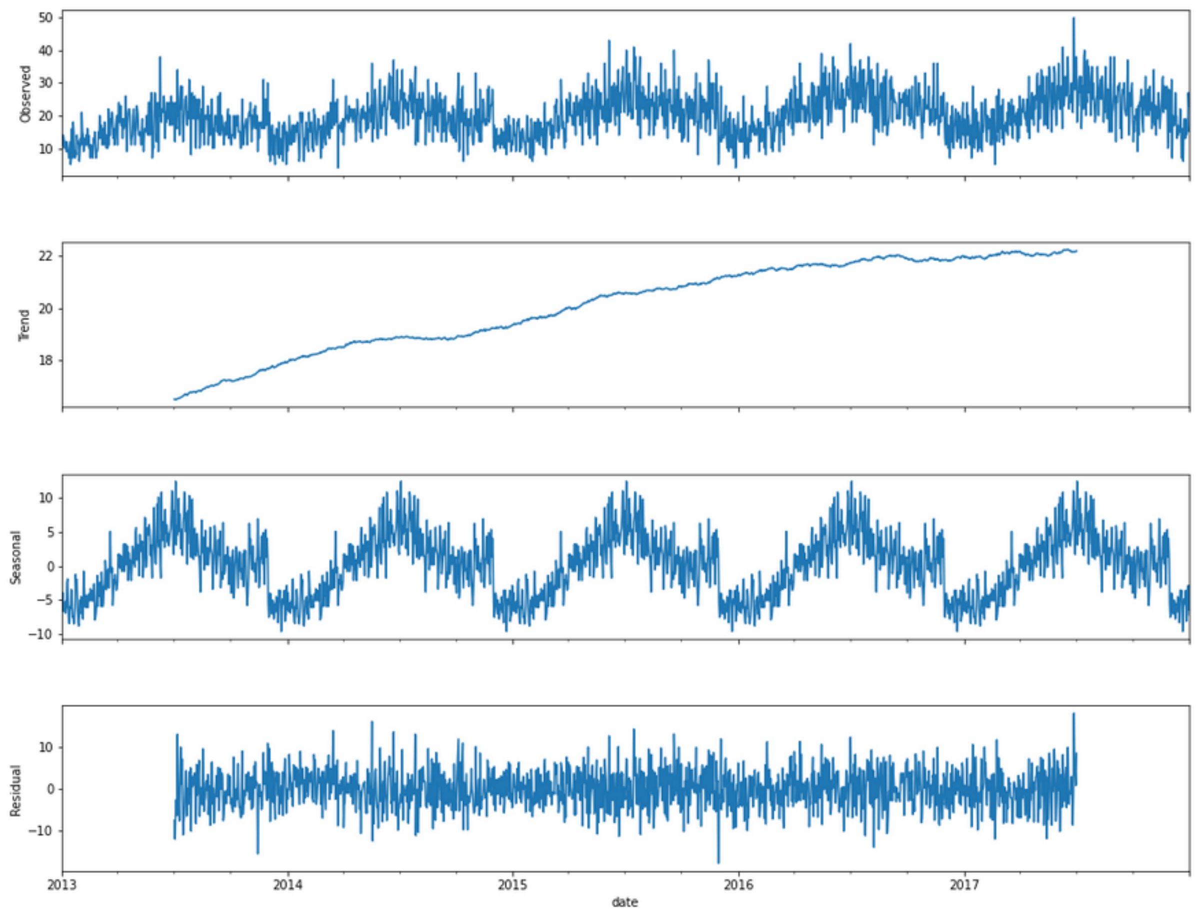


Figure 5

From figure 5, it is clear that the time series data is seasonal with a separated upward trend of data.

2.2.2 Store Trend

Sales were aggregated by store & date and plotted on the graph in the figure below. Two stores are shown in the figure, the stores that sell the most and the least to depict a clearer presentation of the store sales trend.

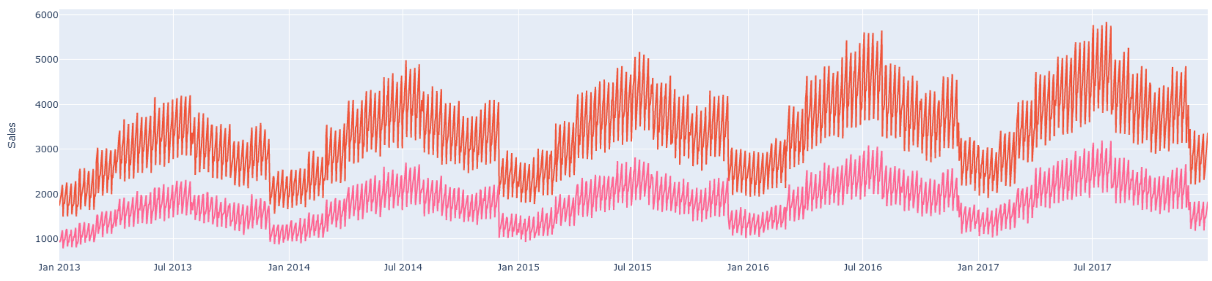


Figure 6

The same yearly trend is observed in both stores, which suggests that the data has a seasonality pattern.

2.2.3 Item Trend

Similarly, sales were aggregated by item & date and plotted on the graph in the figure below. Two items are shown in the figure, items with the highest and lowest mean, to depict a clearer presentation of the item's sales trend.

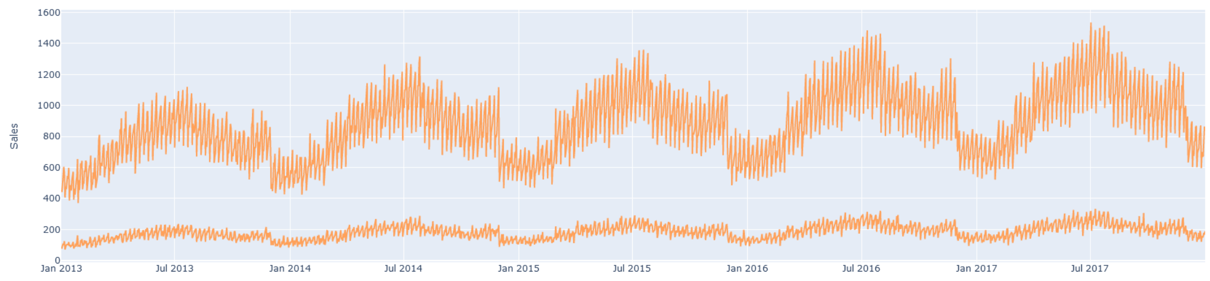


Figure 7

The same yearly trend is observed in both items, which suggests that the data has a seasonality pattern.

3. Feature Engineering

The goal of feature engineering is to generate new input features that have a relationship with the output feature, sales

3.1. Date-related Features

As the overall sales trend has shown, it reaches a low point nearing the start of the year and a high point nearing the middle of the year.

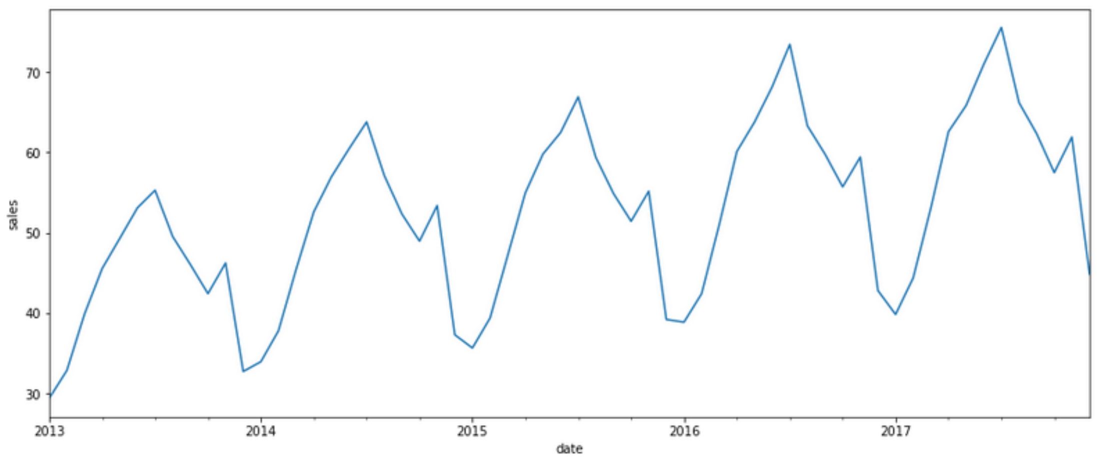


Figure 8

Since there seems to be a correlation with the date and sales, day, month and quarter were extracted from the date as new features.

As sales generally increase during weekends, a box plot was plotted based on the distribution of sales over the day of the week to see if it applied to this dataset.

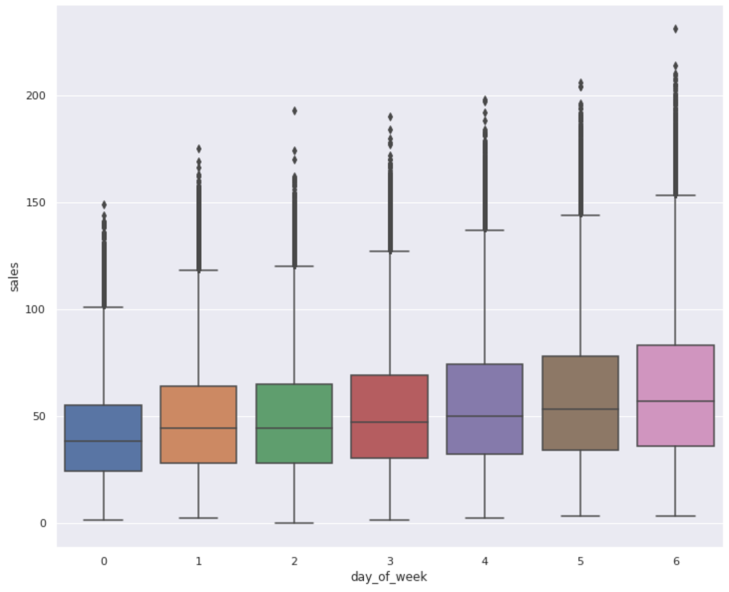


Figure 9

Weekends (Friday, Saturday and Sunday) do seem to have a higher amount of sales. Thus, a new feature, 'is_weekend' is generated, indicating if the date falls on the weekend. In addition, Monday seems to have the least amount of sales. Seeing that the day of the week might have a correlation with sales, the day_of_week feature is extracted from the date.

There is another assumption that holidays will incur more sales, so a bar graph was plotted over the mean sales holidays and non-holiday dates.

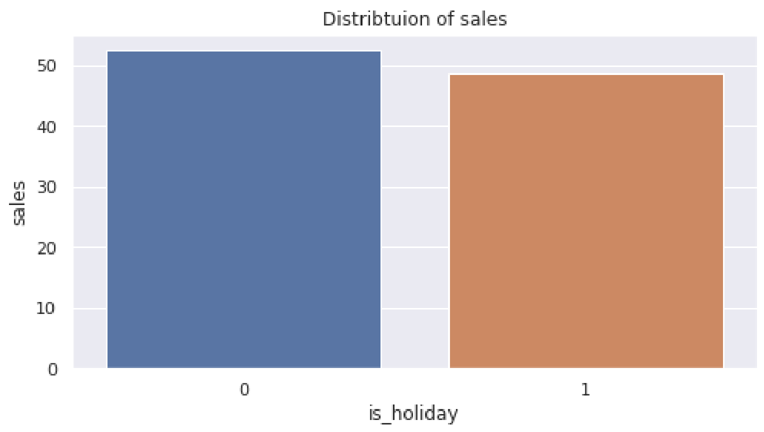


Figure 10

Although sales does not seem to necessarily increase over the holidays, there still might be unforeseen correlation with holidays and sales, and 'is_holiday' feature is generated.

3.2 Lag Features

The general approach here is to use sales to predict the value at time t + 1, given the value at previous time, t -1. Selecting the lag value will depend on the correlation of individual values with it's past values.

From the overall sales data, It seems like there is a strong yearly trend of having the lowest point be at the start of the year and highest point be somewhere around the middle of the year. As there seems to be a strong yearly trend, lag values of a year and a quarter are considered for this dataset. This project uses the shift() function from the pandas library to generate these lag features from the dataset.

Another approach to determine lag correlation is significant is to use the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots. ACF plot is a measure of the correlation between the time series and the lagged version of itself, while PACF plot is a measure of the correlation between time series with a lagged version of itself but after eliminating the variations already explained by the intervening comparisons.

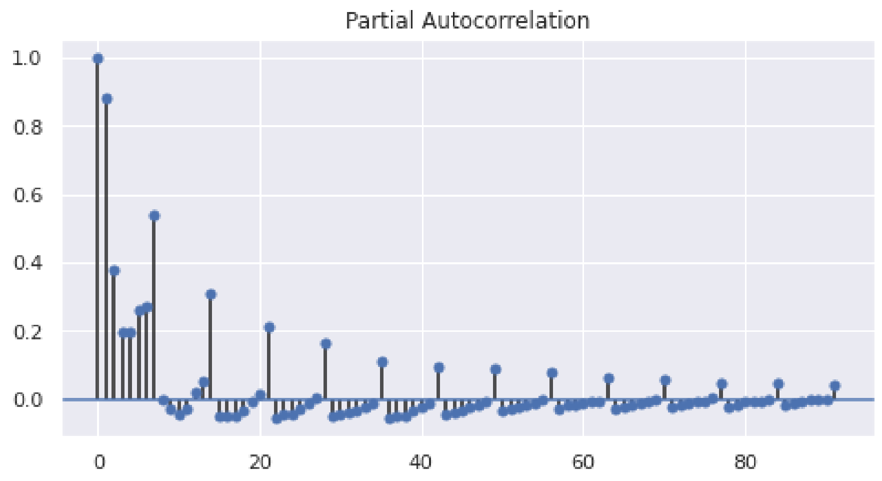


Figure 11

The partial autocorrelation function shows a high correlation with the first two and 7th lags. Therefore, since there is a high correlation between the value and the 7th lag, it is too considered as a feature for the dataset.

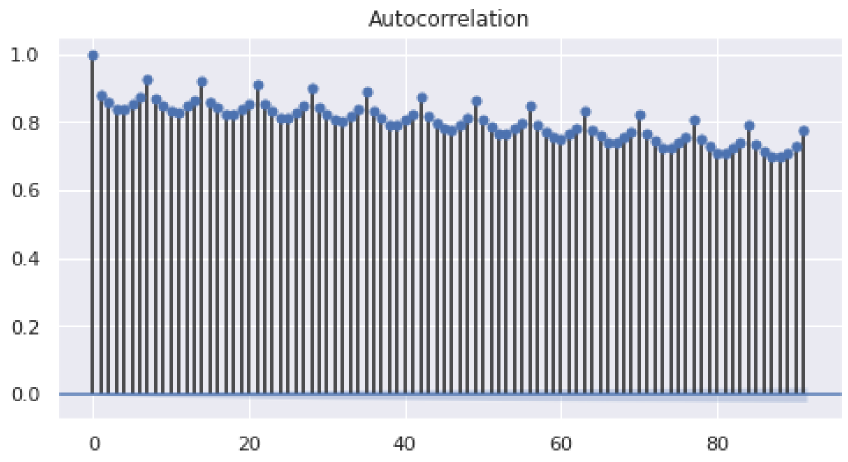


Figure 12

The auto correction shows a slow decay, which means that the future values have a very high correlation with it's past values.

3.3 Rolling Window Feature

Rolling window features are statistical values based on past values, like a summary of value at previous time steps.

A window size of 7 is selected as there were weekly correlations as observed in the PACF. Recency is another important factor in time series, generally, values closer to the current date would hold more information, so we will keep the window size low stay 7. The rolling dataset is created and the mean value is calculated each window of 7 values using the rolling() function from the pandas library.

In conclusion these are the products of feature engineering - day, month, day of week, is_weekend, is_holiday, prev_week_sales, prev_quarter_sales and prev_year_sales, rolling_mean.

4. Models

In this section we will be exploring the use of 3 different models for this problem:

1. SARIMAX
2. Random forest
3. Prophet

4.1 SARIMAX

4.1.1 Overview of SARIMAX

One of the models commonly used in time series analysis and estimation is Autoregressive integrated moving average (ARIMA). ARIMA models are statistical analysis models that use time-series data to predict future trends. It can be understood by outlining each component as follows:

Autoregression (AR) - refers to a model that forecasts the variable using a linear combination of past values of the variable.

Integrated(I) - Integration here means differencing the data, taking the difference between data points and it’s lagged version.

Moving Average (MA) - A moving average model uses past forecast errors in a regression-like model (as opposed to using past values of the forecast variable).

As our time-series data has a seasonal component and uses exogenous variables, the project will be using the SARIMAX model. Unlike traditional ARIMA models, SARIMAX has the capability to account for seasonality and exogenous variables. Application wise, with AR accounting for prior values and MA accounting for prior errors, the model is able to rapidly adjust for sudden changes in trend, resulting in more accurate forecasts. The exogenous component allows the model to take into account external variables and generally ARIMAX models perform better than the ARIMA models in terms of cumulative forecast errors, as stated in their official paper.

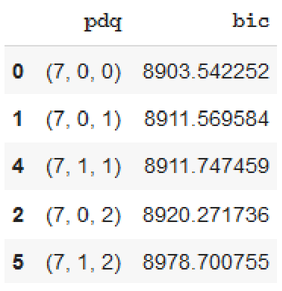
4.1.2 SARIMAX Model Used

The SARIMAX model takes into account various parameters, the notation for SARIMAX models is:

$$\text{SARIMAX}\,(p, d, q) \times (P, D, Q, S)$$

where *p* = non-seasonal autoregressive (AR) order, *d* = non-seasonal differencing, *q*= non-seasonal moving average (MA) order, *P* = seasonal AR order, *D* = seasonal differencing, *Q* = seasonal MA order, and *S* = length of repeating seasonal pattern. Finding the optimal SARIMAX model will require finding the optimal values for these parameters. From the PACF and ACF in figure 11 and 12, it is clear that AR is significant every 7 lags, which means the value of 7, can be used for p.

The rest of the values are sought out through a grid search, every combination of these values are fitted to a model and evaluated with Bayesian information criterion (BIC) as the main selection criterion. BIC selects the model that explains the greatest amount of variation using the fewest possible independent variable.



	pdq	bic
0	(7, 0, 0)	8903.542252
1	(7, 0, 1)	8911.569584
4	(7, 1, 1)	8911.747459
2	(7, 0, 2)	8920.271736
5	(7, 1, 2)	8978.700755

Figure 13

After the grid search, it is observed that Model (7,0,0) had the best BIC score and was selected for the subsequent experiments.

4.1.3 Prediction Result

The model was then evaluated with the last 3 months of the train data. The model scored a SMAPE score of 21.618. The predicted values were then plotted alongside the actual values in Figure 14

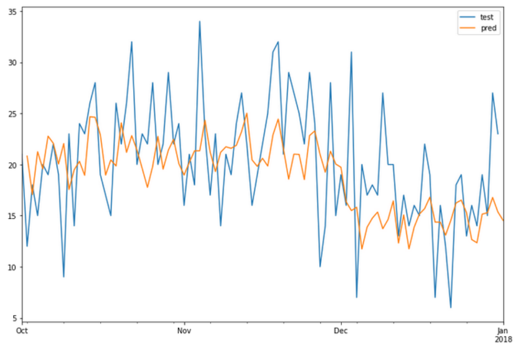
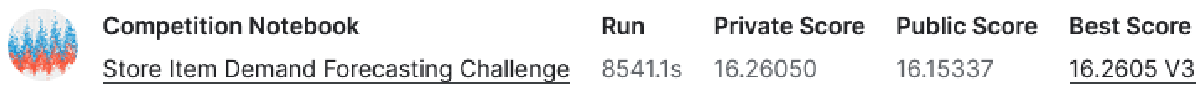


Figure 14

4.1.4 Kaggle Submission Result

The model is then submitted to the kaggle challenge, which returns 16.26 to be the private score and 16.15 to be the public score, as shown in Figure 15.



Competition Notebook	Run	Private Score	Public Score	Best Score
Store Item Demand Forecasting Challenge	8541.1s	16.26050	16.15337	16.2605 V3

Figure 15

4.2 Random Forest

4.2.1 Overview of Random Forest

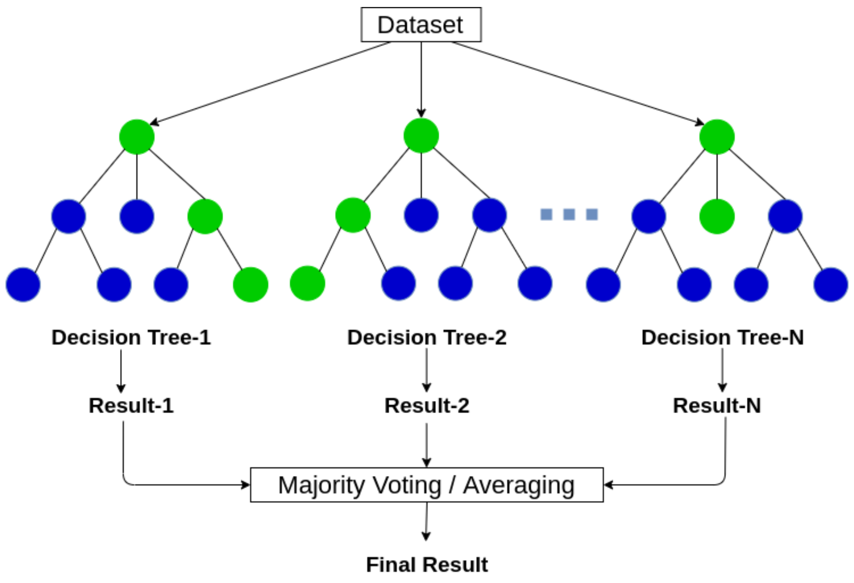
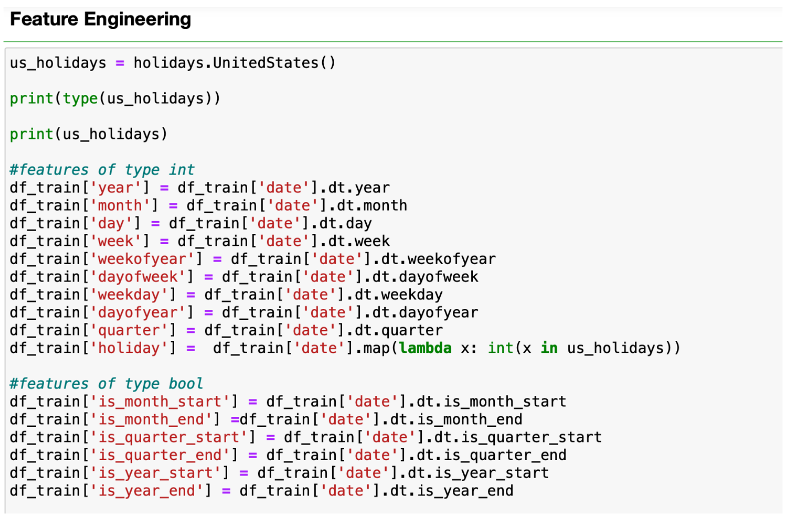


Figure 16

Random forest is a supervised training algorithm where a training set is used to teach a model to yield the desired output. Random forest is a class of ensemble methods where multiple decision tree base classifiers are combined to make a more accurate prediction.

Each decision tree is generated based on a random subset of features and is built on a bootstrap sample of the training data. Training among a subset of features is faster than searching the complete set, improving the efficiency of the algorithm. Bootstrap aggregating and generating random subset of features incorporates diversity in the base classifiers, granting each classifier independent of the other. The final prediction is a more accurate result taken from the average across all the predicted values from each base classifier.

4.2.2 Random Forest Application on Dataset



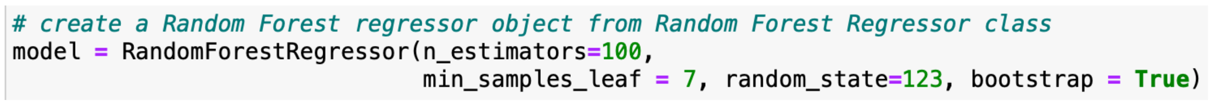
Feature Engineering

```
us_holidays = holidays.UnitedStates()

print(type(us_holidays))

print(us_holidays)

#features of type int
df_train['year'] = df_train['date'].dt.year
df_train['month'] = df_train['date'].dt.month
df_train['day'] = df_train['date'].dt.day
df_train['week'] = df_train['date'].dt.week
df_train['weekofyear'] = df_train['date'].dt.weekofyear
df_train['dayofweek'] = df_train['date'].dt.dayofweek
df_train['weekday'] = df_train['date'].dt.weekday
df_train['dayofyear'] = df_train['date'].dt.dayofyear
df_train['quarter'] = df_train['date'].dt.quarter
df_train['holiday'] =  df_train['date'].map(lambda x: int(x in us_holidays))

#features of type bool
df_train['is_month_start'] = df_train['date'].dt.is_month_start
df_train['is_month_end'] =df_train['date'].dt.is_month_end
df_train['is_quarter_start'] = df_train['date'].dt.is_quarter_start
df_train['is_quarter_end'] = df_train['date'].dt.is_quarter_end
df_train['is_year_start'] = df_train['date'].dt.is_year_start
df_train['is_year_end'] = df_train['date'].dt.is_year_end
```

Figure 17

The dataset contains three features, the date, store ID and item ID for prediction of the sales value. From the exploratory data analysis of the dataset, feature engineering extracts more important features that seem to have a correlation with sales. Gathering more data through feature engineering tends to improve the performance of the random forest regression model.

4.2.3 Random Forest Model Used



```
# create a Random Forest regressor object from Random Forest Regressor class
model = RandomForestRegressor(n_estimators=100,
                              min_samples_leaf = 7, random_state=123, bootstrap = True)
```

Figure 18

For the random forest regressor model used, the bootstrap parameter is set to true to engage bagging of the training data. Since bagging repeatedly samples the training data with replacement for a uniform probability distribution, certain instances of the training data may not be sampled if the number of trees are too small. The number of trees (n_estimators) in our model is tuned to the default value of 100 which gives a good result when tuning the hyperparameters of the model.

For the purpose of this project to ensure that the same random forest model is the same each time it is built, the random_state value is set to a random seed integer value. This ensures that the random results of bagging can be reproduced for the random forest model to be constant.

Min_samples_leaf is the minimum number of samples required to be at the leaf node such that a split point in each decision tree will be considered if it has this number of training samples in each of the left and right branches. While performing hyperparameter tuning, the value of 7 is found to give a good result for the prediction.

4.2.4 Evaluation of Random Forest Model

The model was then evaluated with the actual values from the training dataset against the predicted values from the training dataset. The model scored a SMAPE score of 11.021. The model also scored a R-squared score for 0.9506. The predicted values are plotted alongside the actual values in Figure 19.

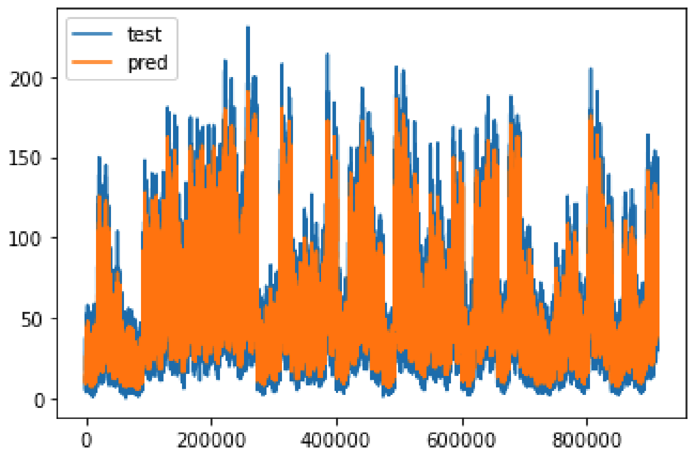
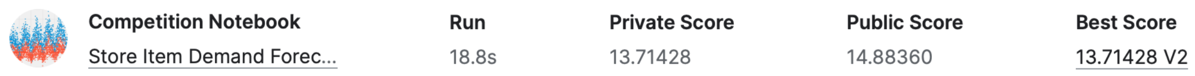


Figure 19

4.2.5 Kaggle Submission Result

The model is then submitted to the kaggle challenge, which returns 13.71 to be the private score and 14.88 to be the public score, as shown in Figure 20.



Competition Notebook	Run	Private Score	Public Score	Best Score
Store Item Demand Forec...	18.8s	13.71428	14.88360	13.71428 V2

Figure 20

4.3 Prophet

4.3.1 Overview of Prophet

The Prophet library is an open-source library designed by Facebook. Its purpose is making forecasts for univariate time series datasets. It is easy to use and designed to automatically find a good set of hyperparameters for the model in an effort to make skillful predictions for data that have strong seasonal effects and several seasons of historical data.

4.3.2 Parameters of Prophet

Trend Parameters

- **growth:** linear' or 'logistic' to specify a linear or logistic trend
- **changepoints:** List of dates at which to include potential changepoints (automatic if not specified)
- **N_changepoints:** If changepoints in not supplied, you may provide the number of changepoints to be automatically included
- **changepoint_prior_scale:** Parameter for changing flexibility of automatic changepoint selection
- **changepoint_range:** The proportion of the history in which the trend is allowed to change

The trend parameters are automatically set. Prophet will automatically detect changepoints and allow the trend of the dataset to adapt. These parameters are only required if a finer control is needed such as overfitting or underfitting. In most cases, the default parameters will work.

For parameter tuning, changepoint_prior_scale can be tuned due to its impact. It determines the flexibility of the trend. In particular, how much the trend changes at the trend changepoints. If the changepoint is too small, the trend will be underfit and variance that should have been modeled with trend changes will instead end up being handled with the noise term. If it is too large, the trend will overfit and in the most extreme case, it can end up with the trend capturing yearly seasonality. The default is set at 0.05, and can be tuned at [0.001, 0.5]. Though not shown in the report, the experiments we carried out showed that changepoint_prior_scale tuning does not have significant improvement to the accuracy of the model.

Seasonality & Holiday Parameters

- **daily_seasonality:** Fit daily seasonality. Set to true if it exists. Otherwise false
- **weekly_seasonality:** Fit weekly seasonality. Set to true if it exists. Otherwise false
- **yearly_seasonality:** Fit yearly seasonality. Set to true if it exists. Otherwise false
- **holidays:** Feed dataframe containing holiday name and date
- **seasonality_prior_scale:** Parameter for changing strength of seasonality model
- **holiday_prior_scale:** Parameter for changing strength of holiday model
- **seasonality_mode:** additive or multiplicative

The EDA looked into yearly seasons. For this time series model, we will need to dive deeper into the seasonality of the dataset: yearly -> monthly -> weekly. Daily is not applicable here as time is not provided in the dataset.

Weekly Seasonality:

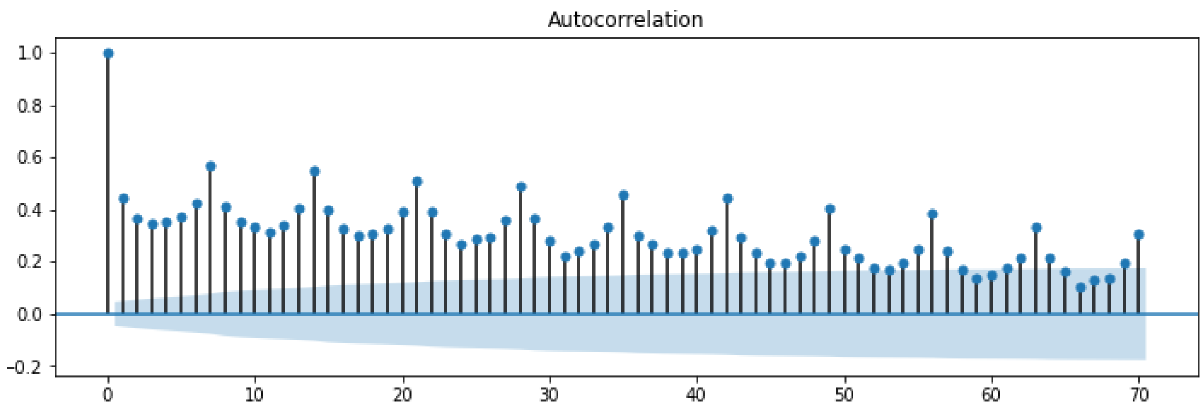


Figure 21

1 "stick" refers to 1 date. There are 70 lags (10 weeks) in figure 21. From figure 21, it can be seen that there is a pattern of weekly seasonality. The 1st, 2nd and 7th lag are of much higher correlation which supports EDA's discovery of weekends being correlated with sales. There exists a weekly seasonality.

Monthly Seasonality:

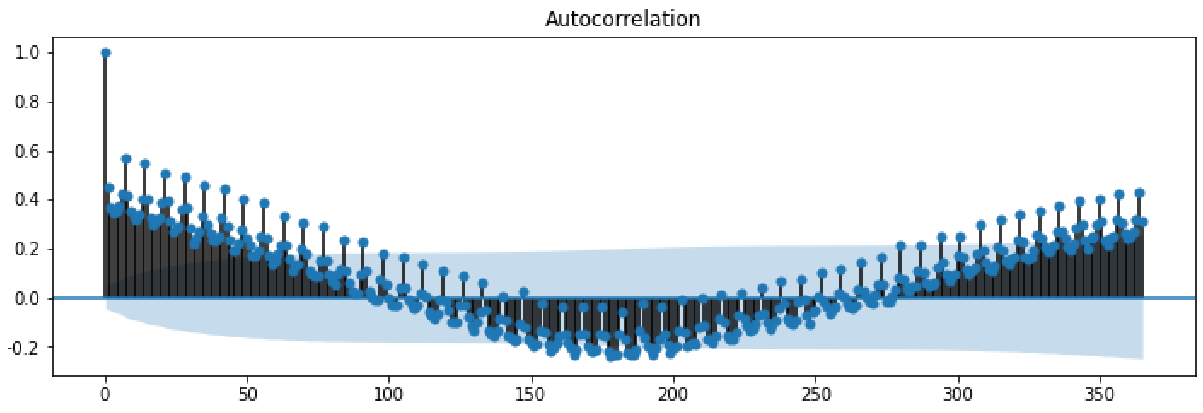


Figure 22

There are 365 lags in figure 22. In figure 22, there is 1 year worth of consecutive dates. It can be seen that there is a dip in sales in the middle of the year, followed by a rebound during the end of the year. The trend seems to be repeated each year. There exists a monthly seasonality.

Yearly Seasonality:

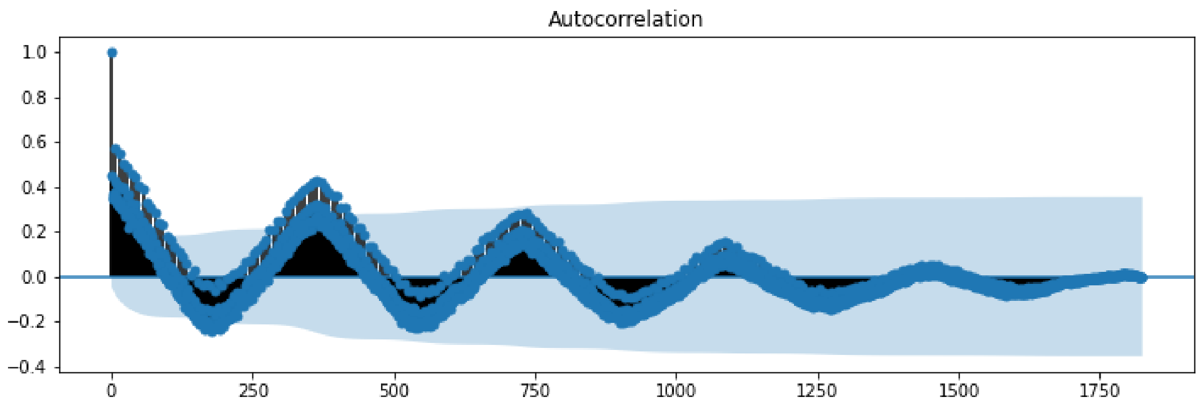


Figure 23

From figure 23, there are 5 years worth of consecutive dates. There is a decreasing trend of yearly seasonality.

4.3.3 Model Tuning

The following tests different sets of parameters of the prophet model. SMAPE is used to judge the accuracy of the model. This test is done on a single pair of stores and items so the generation can be much quicker. Once the parameter settings are tested, the best set of parameters will be used in the final model to predict the 3 months sales results for all stores and all items.

The seasonality_mode is set to additive as the trend is of gradual increases instead of huge surge.

<u>Model 1: Default Model with No Parameters</u>

Model parameters used:

```
model = Prophet(daily_seasonality=False,
  weekly_seasonality=False,
  yearly_seasonality=False,
  seasonality_mode='additive',)
```

The following graph shows the Predicted versus Actual Sales over 5 years for model 1:

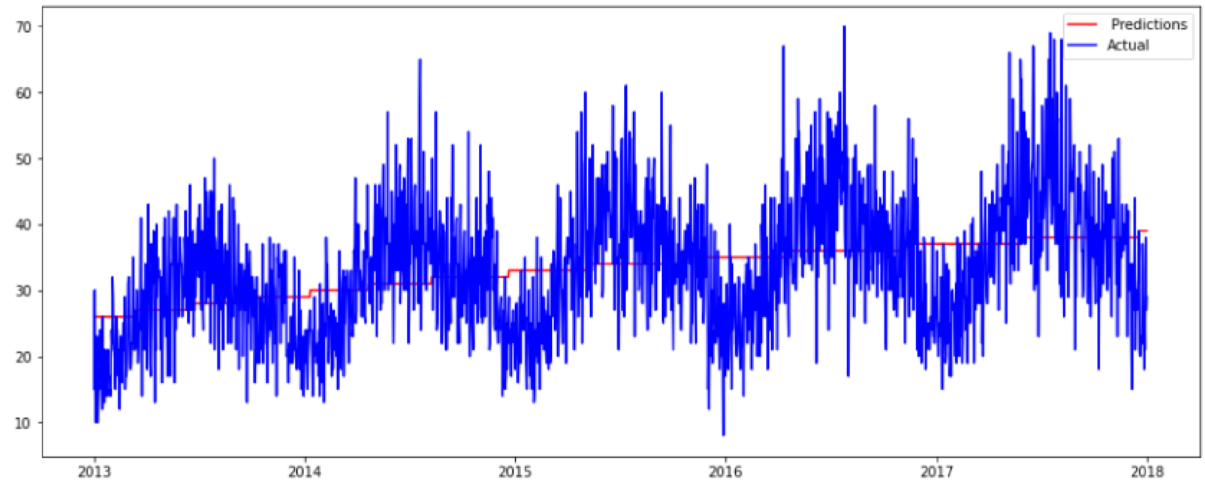


Figure 24

SAMPE of Model: 23.26947342589741

From figure 24, it can be seen that the model without any parameters, generates a prediction that is very far from the actual sales result. This model is not accurate and should be improved further.,

Model 2: Model with Seasonality

Model parameters used:

```
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)
```

The following graph shows the Predicted versus Actual Sales over 5 years for model 2:

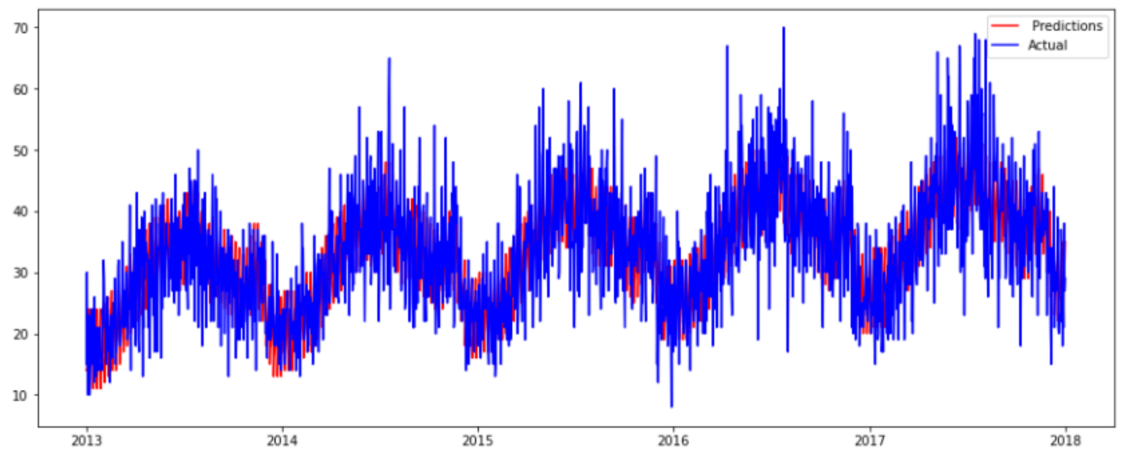


Figure 25

SAMPE of Model: 14.549194123058971

From our seasonality analysis, it can be seen that there was weekly season and yearly season in the dataset thus they are set to True. It can be seen that the SAMPE was improved greatly and from figure 25, the prediction is much more accurate compared to the previous model.

Model 3: Model with Seasonality + Holidays

Model parameters used:

```
model = Prophet(daily_seasonality=False,
  weekly_seasonality=True,
  yearly_seasonality=True,
  holidays = holidays, seasonality_mode='additive',)
model.add_seasonality(name='monthly', period=30.5, fourier_order=5)
```

The following graph shows the Predicted versus Actual Sales over 5 years for model 3:

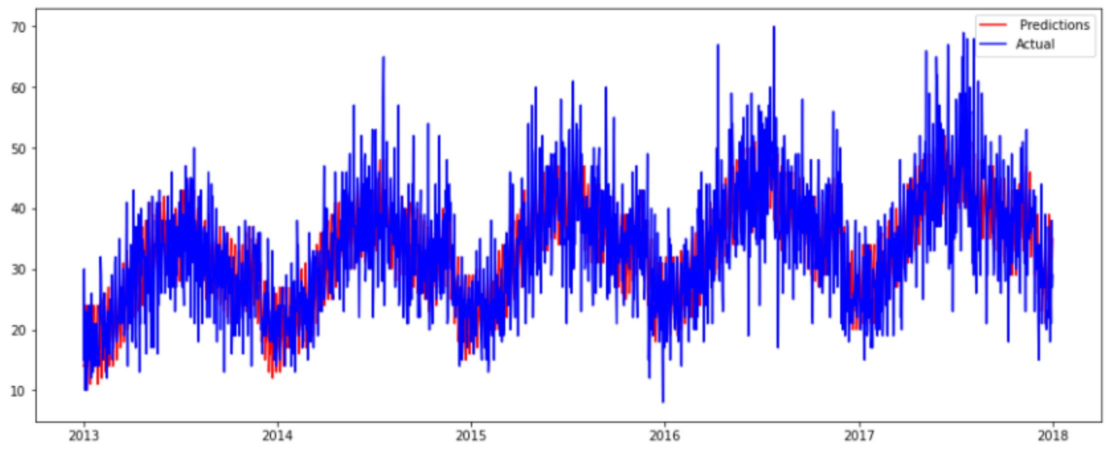


Figure 26

SAMPE of Model: 14.459655890540562

In the EDA, it was unsure if holidays have any hidden correlation to sales. It may be due to the amount of holidays being too little compared to non holidays thus the visualization is not as clear on the bar chart. It is much easier to test the effect directly on the model. Though the improvement in terms of SAMPE is not much, it is still a significant improvement.

Model 4: Model with Seasonality + Holidays + Holiday Effect Scale

Model parameters used:

```
model = Prophet(daily_seasonality=False,
  weekly_seasonality=True,
  yearly_seasonality=True,
  holidays = holidays,
seasonality_mode='additive',holidays_prior_scale=0.7,)
model.add_seasonality(name='monthly', period=30.5, fourier_order=5)
```

The following graph shows the Predicted versus Actual Sales over 5 years for model 4:

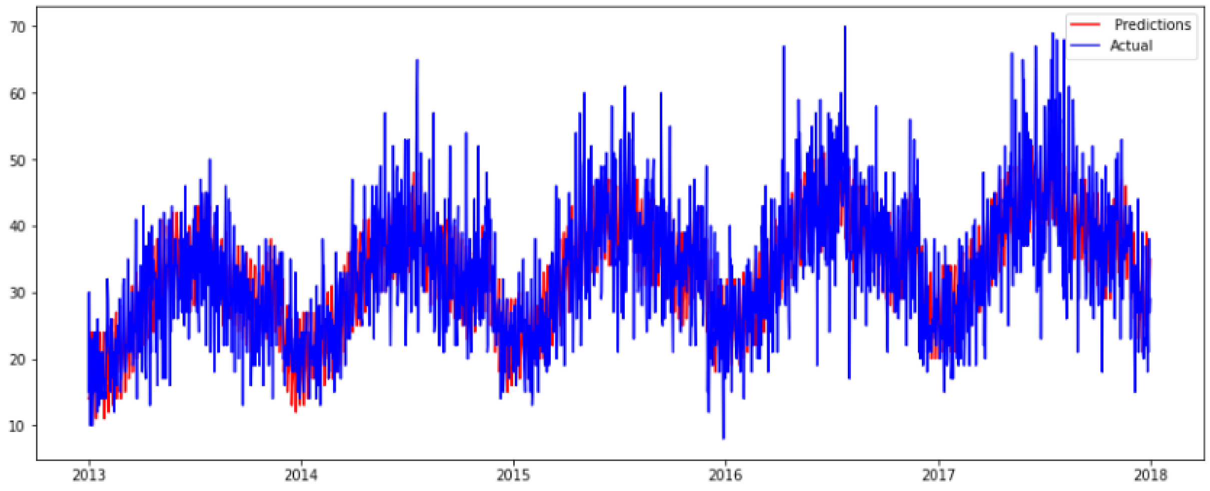


Figure 27

SAMPE of Model: 14.470509043295529

From EDA, holidays do not have a clear correlation with sales. In this model, we try to scale the effect of holidays on the model down. The SAMPE decreased slightly. It is better not to scale it down.

Model 5: Model with Seasonality + Holidays + Log Transform

Model parameters used:

```
df["sales"] = np.log1p(df["sales"])
model = Prophet(daily_seasonality=False,
  weekly_seasonality=True,
  yearly_seasonality=True,
  holidays = holidays, seasonality_mode='additive',)
```

```
model.add_seasonality(name='monthly', period=30.5, fourier_order=5)
forecast['yhat'] =  np.expm1(forecast['yhat'])
```

The following graph shows the Predicted versus Actual Sales over 5 years for model 5:

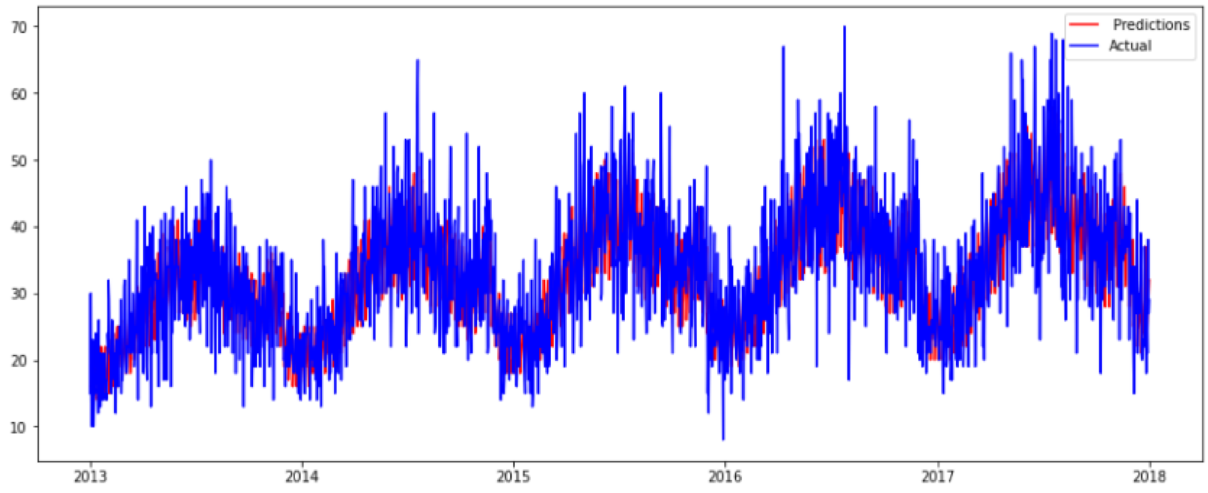


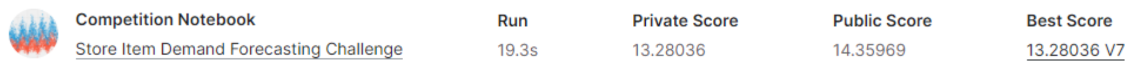
Figure 28

SAMPE of Model: 14.239401665798605

In this model, the parameters used are the same as model 3, but the dataframe fitted into the model is log transformed to stabilize the variance. There is a great improvement in terms of SAMPE. This set of settings and parameters will be used for the final model which will loop through every store-item pair to generate the required competition output.

4.3.4 Kaggle Submission Result

The generated result achieved a private score of 13.28036 and public score of 14.35969 when submitted to the Kaggle competition as shown in figure 29.



Competition Notebook	Run	Private Score	Public Score	Best Score
Store Item Demand Forecasting Challenge	19.3s	13.28036	14.35969	13.28036 V7

Figure 29

5. Enhancement of Result

Based on previous model results, we can see that our ranking is still not as good.
In this section, we will be going in-depth on how we create our own algorithm to enhance our result for this competition.

5.1 Ensemble Learning with Thresholding

There are existing concepts for combining different models to achieve a better result, one of them is called ensemble learning. Ensemble learning can improve the performance in terms of the accuracy of the results through aggregating predictions of multiple models. It uses a based model that is generated from the training data to generate multiple sets of predictions using different parameters. The concept for classification is on majority voting where 3 different sets of results will vote and the majority votes will be the final prediction for the row. This is highly dependent on the accuracy of each of the generated models as if 2 out of 3 of them contains the wrong result, the row prediction will be wrong.

For our case, instead of just relying on one model to generate multiple results, we will use 3 of our model's results as the prediction. As our problem is a regression problem instead of a classification problem, comparing results from different rows to get the exact similarity is hard. Thus we can use a threshold for comparison between different results.

5.1.1 Ensemble Learning with Thresholding on 3 Models

We will use 3 different predicted results from our 3 models, SARIMAX, Random forest, and prophet.

These are the steps:
1. Set lowest SMAPE rate classifier results to be the default. This result will be the default results if there is no change to a row value.
2. Set a maximum difference threshold, if the maximum difference of any 2 model's results is above the threshold value, then we will take the average of 2 of the models that has the least amount of difference to be the predicted results for the row. Taking the least amount will modify the results slightly which hopes to maintain maximum accuracy.
3. If all 3 of the row results are below the threshold, we will take the default dataset's predicted result as the row's predicted result.

Example idea of how it works:
Predicted results of the dataset for 3 models

Max difference threshold: t = 6.8	Model A SMAPE =12.3 (Default) Predicted sales	Model B SMAPE =12.6 Predicted sales	Model C SMAPE = 14.1 Predicted sales	Final Predicted sales

Row1 sales 1.Check their difference: A to B : 23-20 = 3 A to C : 20-16 = 4 B to C:24-16=8 2.Max difference = 8, 8>t = True . 3. If the max is bigger than the threshold, pick the smallest difference and take their average as the final result. A to B has the smallest difference at 3 so we take avg of them. (23+20)/2 = 21.5	20	23	16	(23+20) /2 = 21.5
Row2 sales 1.Check their difference: A to B = 19-18 = 1 A to C = 20-18 = 2 B to C = 20-19=1 2. Max difference = 2, 2<t = False 3. Use the default model's value as the final value, 18.	18	19	20	18
Row3 sales 1.Check their difference: AtoB = 2-2 = 0 AtoC = 9-2 = 7 BtoC = 9-2=7 2. Max difference = 7, 7>t = True 3. As the max difference is smaller than the threshold, pick the smallest difference and take their average as the final result. A to B has the smallest difference at 0 so we take avg of them. (2+2)/2 = 2	2	2	9	(2+2) /2 = 2
...				

Table 1

Pseudo code:

```
Min_threshold = 2
Results = [cls1Results,cls2Results,cls3Results]
New_results = []
difference_arr = []
For each row in any df cause all same len:
        difference_arr.append(abs(cls1Results-cls2Results))
        difference_arr.append(abs(cls1Results-cls3Results))
        difference_arr.append(abs(cls2Results-cls3Results))
        max_value = max(difference_arr)
        If value>min_threshold:
                min_value = min(difference_arr)
                min_index = difference_arr.index(min_value)
                If min_index==0:
                        This row value = abs(cls1Results+cls2Results/2)
                If min_index==1: round(cls1Results+cls3Results/2)
                        This row value = abs(cls1Results+cls3Results/2)
                If min_index==2: round(cls2Results+cls3Results/2)
                        This row value = abs(cls2Results+cls3Results/2)
                New_results.append(this row value)
        Else:
                New_results.append(default df value for this row)
```

Apply algorithm on actual predictions:

We can run through the datasets and take the maximum sum difference between each of the model's results to determine the number of times a swap is needed to replace the default dataframe value.

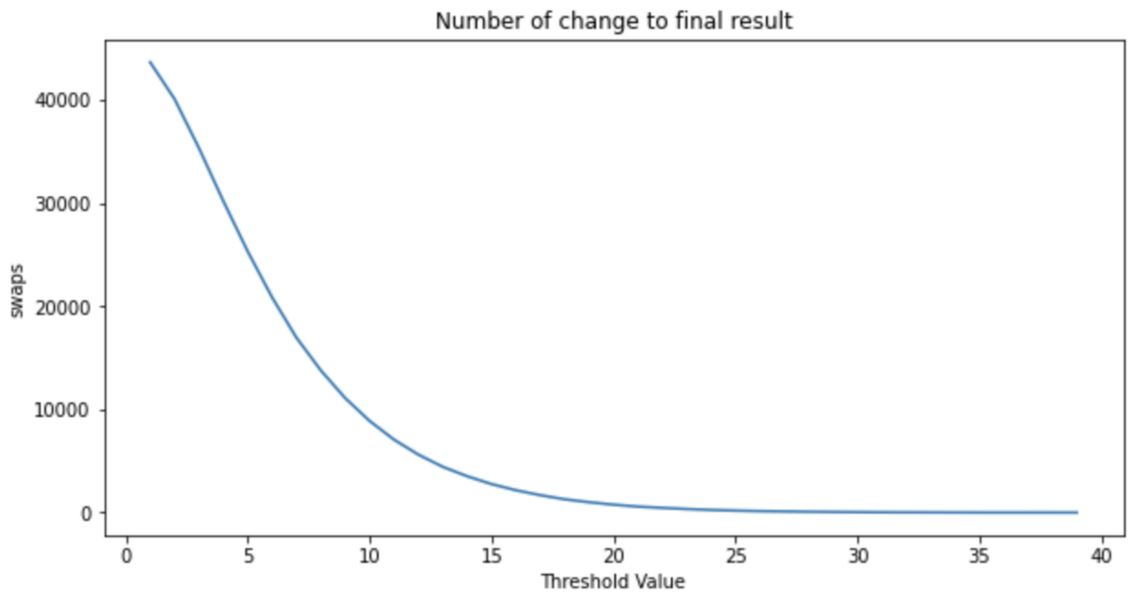


Figure 29

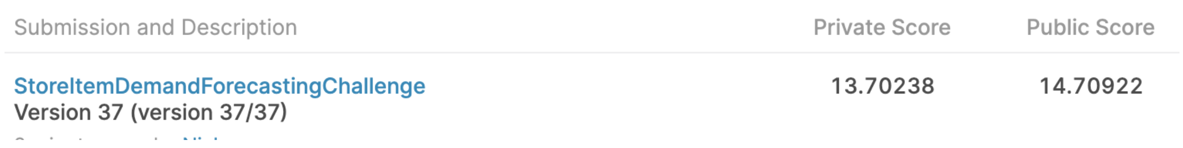
This graph shows how the threshold value affects the number of averaging that needs to be done. As lower threshold values show a huge change, the gradient of the curve starts to become less steep as the threshold value increases, this shows that this algorithm can only improve up to a certain point above a threshold value of 20, the accuracy difference would not make much difference.

Here are some of the experimental results of using different threshold values:

Max difference threshold	Private Score	Public Score	Private Score Ranking	Private Score Ranking Percentage
1	13.70238	14.70922	277	60.3%
3	13.65904	14.71648	276	60.1%
9	13.55765	14.66714	271	59%
12	13.52065	14.64202	270	58.8%
15	13.52218	14.60324	270	58.8%
22	13.50065	14.60474	266	57.9%

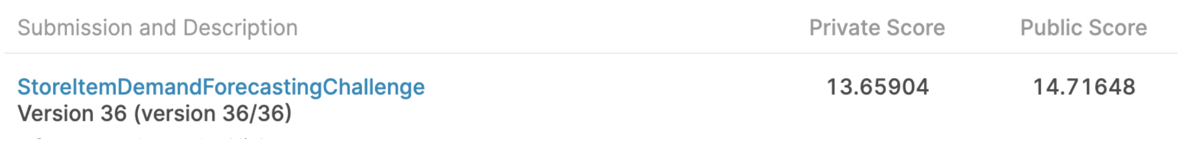
Table 2

Max difference threshold = 1



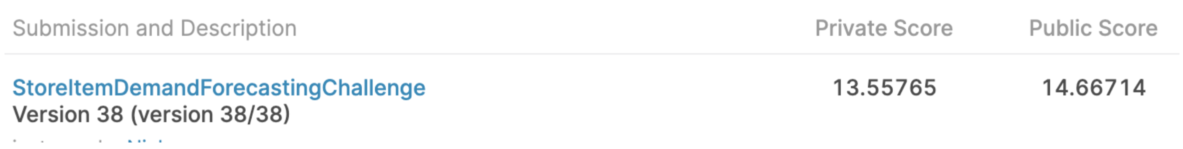
Submission and Description	Private Score	Public Score
StoreItemDemandForecastingChallenge Version 37 (version 37/37)	13.70238	14.70922

Max difference threshold = 3



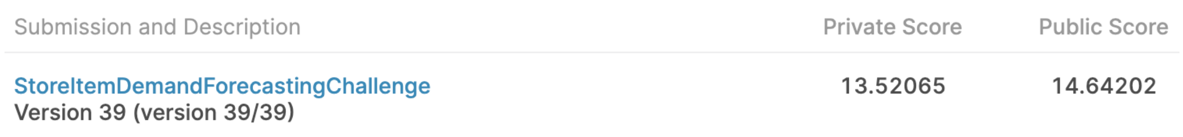
Submission and Description	Private Score	Public Score
StoreItemDemandForecastingChallenge Version 36 (version 36/36)	13.65904	14.71648

Max difference threshold = 9



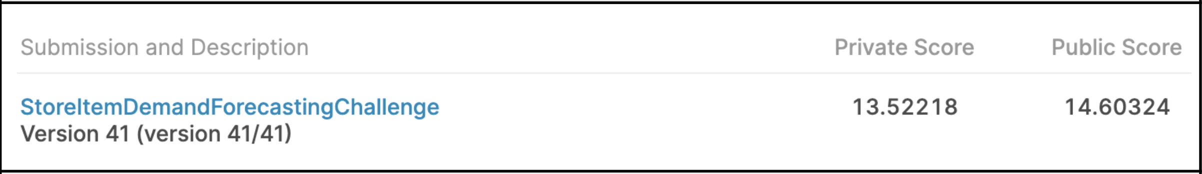
Submission and Description	Private Score	Public Score
StoreItemDemandForecastingChallenge Version 38 (version 38/38)	13.55765	14.66714

Max difference threshold = 12



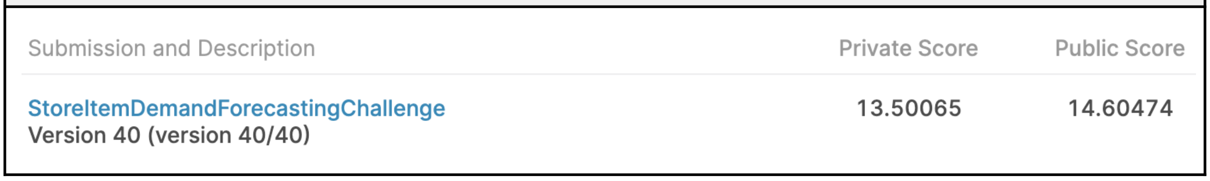
Submission and Description	Private Score	Public Score
StoreItemDemandForecastingChallenge Version 39 (version 39/39)	13.52065	14.64202

Max difference threshold = 15



Submission and Description	Private Score	Public Score
StoreItemDemandForecastingChallenge Version 41 (version 41/41)	13.52218	14.60324

Max difference threshold = 22



Submission and Description	Private Score	Public Score
StoreItemDemandForecastingChallenge Version 40 (version 40/40)	13.50065	14.60474

Table 3

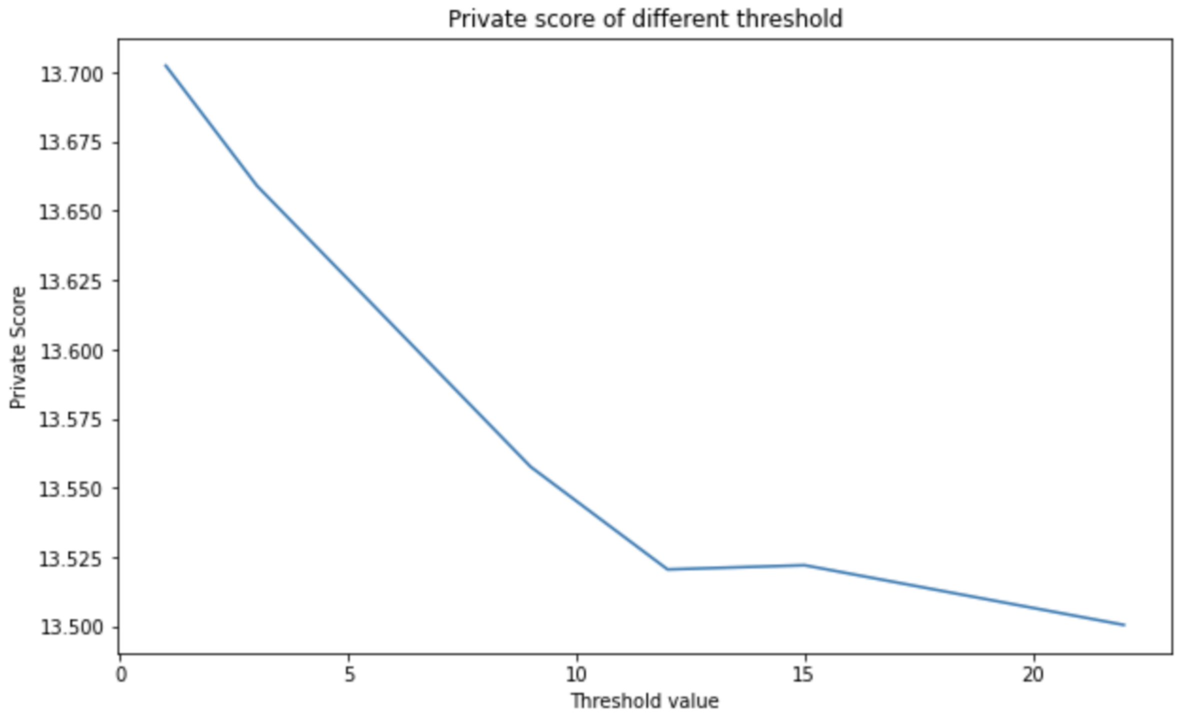


Figure 30

As the threshold increases, the ranking decreases. This shows that when there are a lot of changes to the default dataset is needed due to the high deviation between them. This also tells us that our results are not very good as it requires a big deviation threshold.

Although the private score is decreasing when the threshold value increases, the above data shows that there is no improvement in the score against our best model prophet, scoring 13.28 for private, even with ensemble learning.

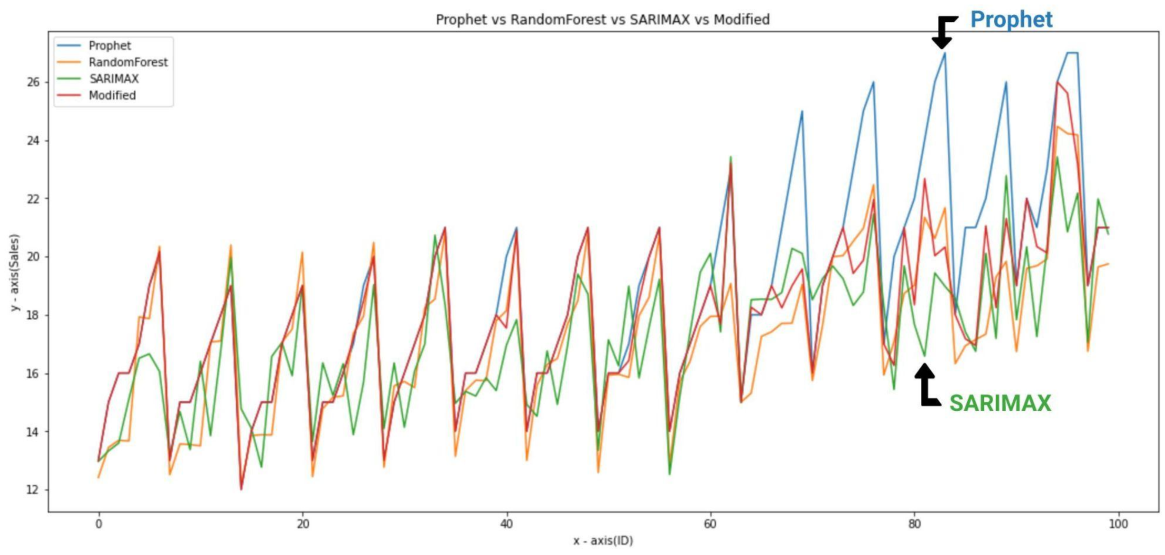


Figure 31

Above are the plotted results of the first 100 predicted results of random forest, prophet, SARIMAX, and the modified final results. When we plot their results side by side, we can see that the similarity of these models deviates after id 60 and above. One indicator can be seen at around id of 80 where SARIMAX's result points downwards while prophet and random forest points upwards. The difference between the lowest score and the highest score is too huge which also resulted in a huge threshold value.

The results correlate with the assumption of ensemble learning where the model accuracy should be similar to each other. SARIMAX's accuracy performance may be too far apart from prophet and random forest, thus SARIMAX may be the cause of pulling down the final results.

5.1.2 Ensemble Learning with Thresholding on 2 Models

In this algorithm, we will remove SARIMAX and use prophet and random forest as they have similar SMAPE scores. The algorithm is similar to the 3 model ensemble learning from above but with a slight difference, instead of taking the minimum difference from the lowest 2 models, we will just take the average of the existing 2 models since there are only 2 of them.

The steps are as such:

1. Set lowest SMAPE rate classifier results to be the default. This result will be the default result if there is no change to a row value.
2. Set a maximum difference threshold, if the difference of the 2 model's result is above the threshold value, then we will take the average of 2 to be the predicted results for the row.
3. If all 3 of the row results are the same, we will take the default dataset's predicted result as the row's predicted result.

Pseudo code:

```
Min_threshold = 2
Results = [cls1Results,cls2Results]
New_results = []

For each row in any df cause all same len:
        difference = 0
        difference=abs(cls1Results-cls2Results))

        If difference>min_threshold:
                New_results.append(abs(cls1Results+cls2Results/2))
        Else:
                New_results.append(default df value for this row)
```

Let us visualize how different thresholds will affect the results by the number of times it needs to replace the default value.

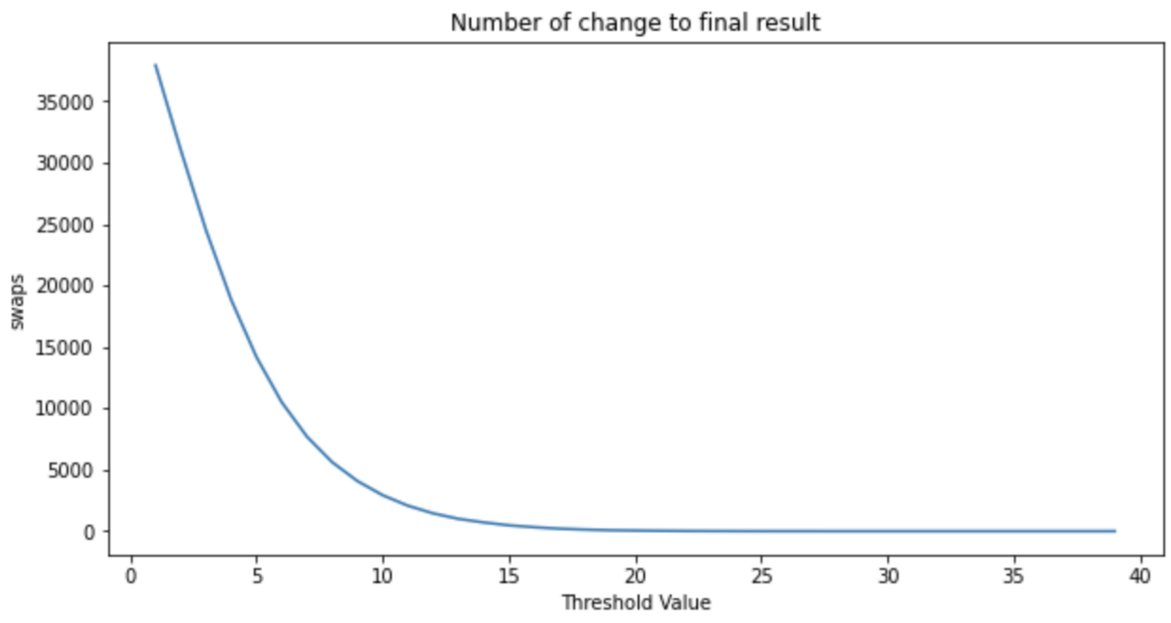


Figure 32

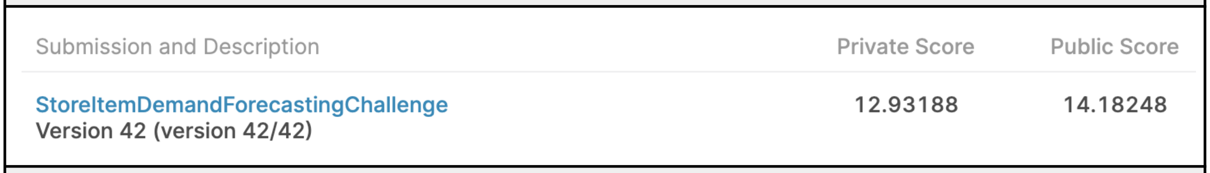
As we can see that the maximum number of swaps on the lower end of the threshold value has decreased as compared to one with 3 models in the previous section. This is due to the accuracy of the 2 models being similar, thus the difference between them is reduced. This chart also shows the same result as the threshold value drops to around 15, the changes to the result might be small.

Results:

Max difference threshold	Private Score	Public Score	Private Score Ranking	Private Score Ranking Percentage
1	12.93188	14.18248	158	34.6%
3	12.95155	14.27606	169	36.8%
9	13.25903	14.55452	244	53.1%
12	13.36897	14.59192	259	56.4%
15	13.44469	14.60214	265	57.7%
22	13.49095	14.60713	265	57.7%

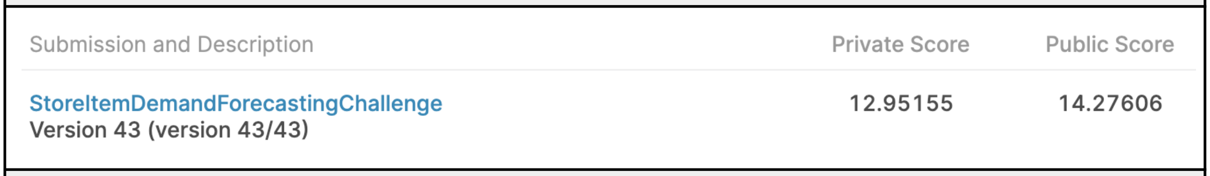
Table 4

Max difference threshold = 1



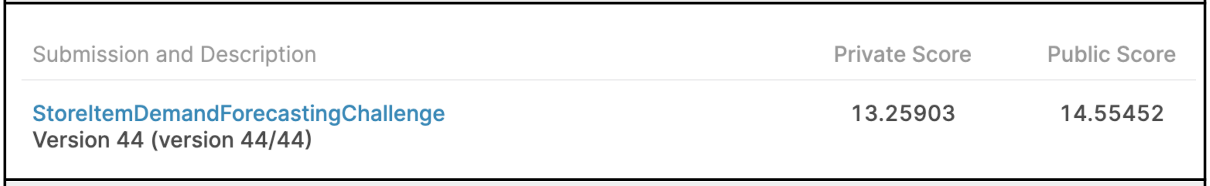
Submission and Description	Private Score	Public Score
StoreItemDemandForecastingChallenge Version 42 (version 42/42)	12.93188	14.18248

Max difference threshold = 3



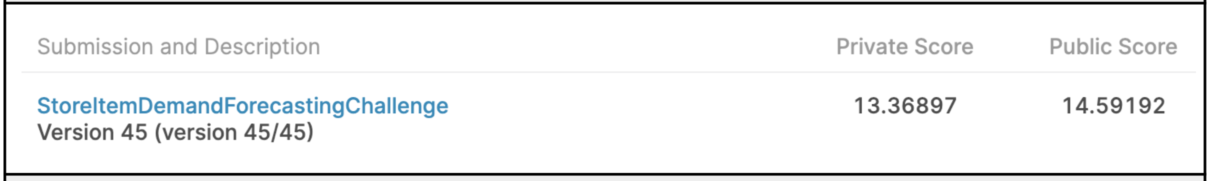
Submission and Description	Private Score	Public Score
StoreItemDemandForecastingChallenge Version 43 (version 43/43)	12.95155	14.27606

Max difference threshold = 9



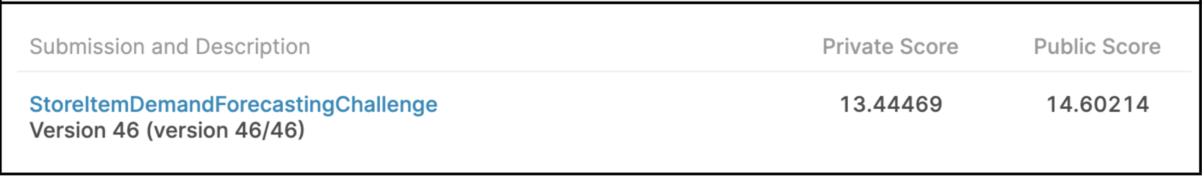
Submission and Description	Private Score	Public Score
StoreItemDemandForecastingChallenge Version 44 (version 44/44)	13.25903	14.55452

Max difference threshold = 12

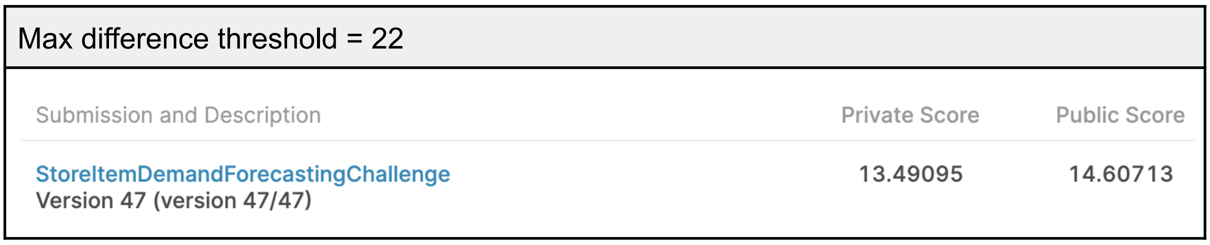


Submission and Description	Private Score	Public Score
StoreItemDemandForecastingChallenge Version 45 (version 45/45)	13.36897	14.59192

Max difference threshold = 15



Submission and Description	Private Score	Public Score
StoreItemDemandForecastingChallenge Version 46 (version 46/46)	13.44469	14.60214



Max difference threshold = 22

Submission and Description	Private Score	Public Score
StoreItemDemandForecastingChallenge Version 47 (version 47/47)	13.49095	14.60713

Table 5

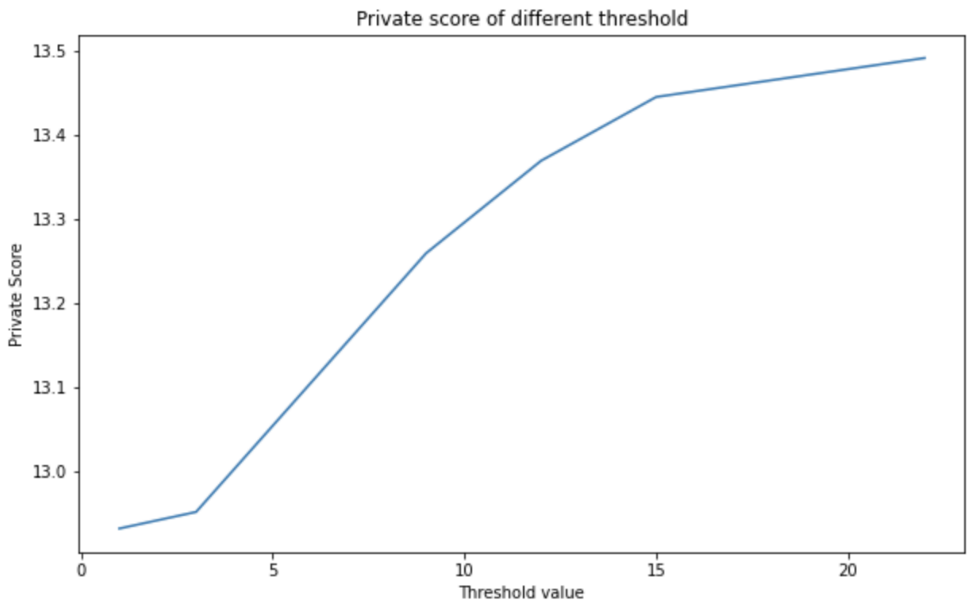


Figure 32

We can see that as the 2 models are similar in terms of their SMAPE score, means they are more related to each other. The results of the different thresholds above show a different trend from the one with SARIMAX in it. When the threshold increases, the performance of the results decreases. This shows that the 2 models are actually quite close to the actual results from Kaggle and there is not much that is needed to change.

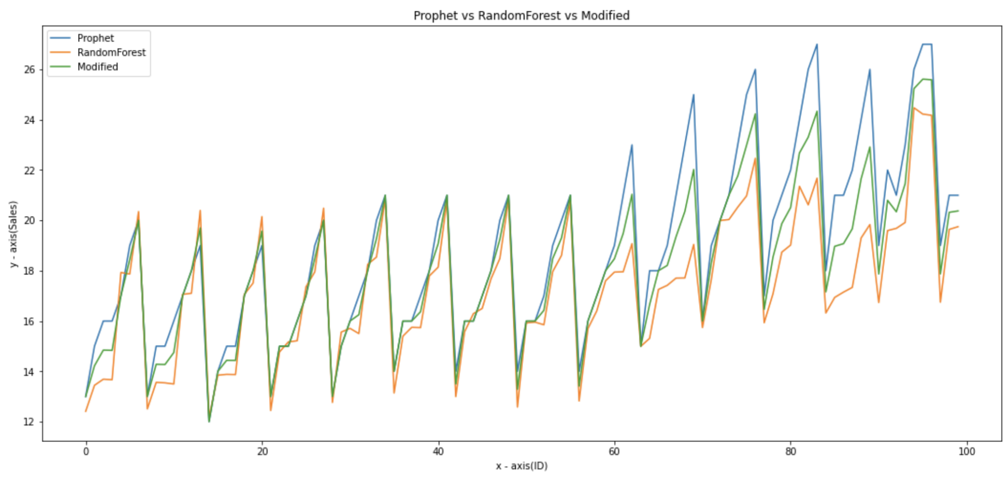


Figure 32

When we plot the first 100 rows of the final predicted results with prophet and random forest, we can see that the trends are the same. At row id 0 to around 58, the results are similar and only some parts where we have to take the average to minimize the difference. For row id after 60, we can see that the ensemble learning thresholding technique actually helps to minimize the difference a lot thus this is why we are able to achieve better results than prophet.

6. Conclusion

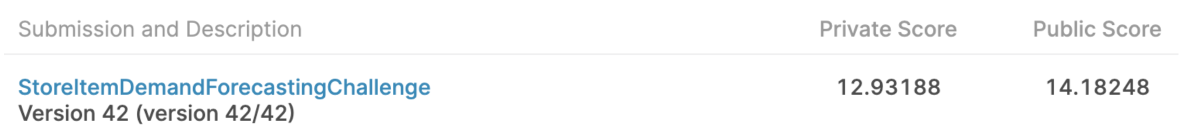
	SARIMAX	Random Forest	Prophet	Ensemble
Private Score	16.26050	13.71428	13.28036	12.93188
Private Ranking	339/459	277/459	251/459	159/459

Table 6

From table 6, it can be seen that each model's ranking is around 55-75% ranking. Though the private score improvement seems to be minimal, the ensemble learning was able to bring our ranking up by a significant number to 34.6%.

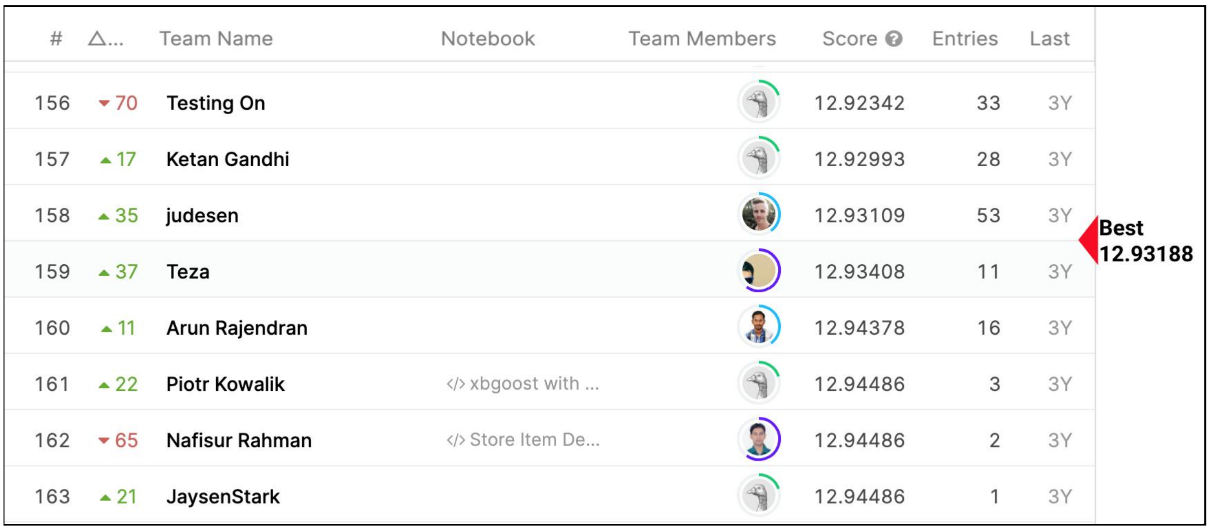
6.1 Final Leaderboard Ranking

The final submission csv file is named submission_best.csv and is included in the submission folder.



Submission and Description	Private Score	Public Score
StoreItemDemandForecastingChallenge Version 42 (version 42/42)	12.93188	14.18248

Private Score: 12.93188
Rank: 159/459 = 34.6%



#	△...	Team Name	Notebook	Team Members	Score	Entries	Last
156	▾70	Testing On			12.92342	33	3Y
157	▴17	Ketan Gandhi			12.92993	28	3Y
158	▴35	judesen			12.93109	53	3Y
159	▴37	Teza			12.93408	11	3Y
160	▴11	Arun Rajendran			12.94378	16	3Y
161	▴22	Piotr Kowalik	</> xbgoost with ...		12.94486	3	3Y
162	▾65	Nafisur Rahman	</> Store Item De...		12.94486	2	3Y
163	▴21	JaysenStark			12.94486	1	3Y

Figure 33. Results of submission_best.csv

6.2 What Was Learnt

Initial
The dataset provided has a minimal amount of features. It seems to be a straightforward dataset. However, what we learnt is that this may not be a good sign as the prediction will be limited by the number of features. Through this project, we understood that datasets with

limited features and features engineering can be applied to expand the dataset to get more insight for predictions.

The deep analysis between the correlation of the data trends allowed us to discover new insights on what affects sales which enabled us to come up with new features. This is a great addition to the dataset.

Models

With the knowledge gained from EDA and new features available from feature engineering, we were able to start modelling. However, we realised that each model has its own requirement of parameters and settings. The information discovered in EDA as well as the additional features may not be utilised fully as some cannot fit into the model. An example is seasonality and holidays are not useful for Random Forest, but are very useful and much needed in Prophet.

The project provided an opportunity to apply theory and learn new prediction models. We have tried using different models to make predictions and found out that traditional models do not perform as well as time series models in this case. The surprising thing is that with the correct attributes and parameters, random forest is able to outperform some of the time series models like SARIMAX. With much time spent on the project, we were able to come by the many different methods for time series predictions and understood its real world importance and applications. It provided a clearer understanding of how the theories learnt are applicable in real applications.

Enhancement

As the team had multiple models on hand, and each individual model's private score was not very satisfactory, we looked into applying ensemble learning which was taught in class to combine all the models to hopefully get better scores. The first application of ensemble learning was a challenge as we had 3 models that were of very different scores, causing the score to worsen further. Thus to get the most accurate results, we have to do experiments through plot visualisation and modify parameters to tweak the results.