NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

SC4000/CE4041/CZ4041 Machine Learning

Group 23: Project Report

Kaggle: Store Item Demand Forecasting Challenge

Group Member	Matriculation Number	Contributions
Ahmad Aleena	U2023405K	EDA, ARIMA
Jarel Tan Zhi Wei	U2121582H	LightGBM, Meta Model
Koh Zi En	U2122351K	SARIMAX, GAM
Lam Ting En	U2021128D	Prophet, Meta Model
Weng Michelle	N2303578B	XGBoost, Video Editing

Table of Contents

Problem Statement

In this Kaggle competition, given 5 years of store-item sales data, we are asked to predict 3 months of sales for 50 different items at 10 different stores. The motivation is to find the best forecasting model to predict the store-item demand.

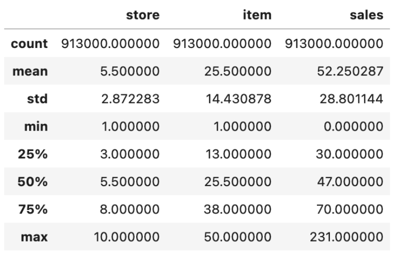
$$\mathrm{SMAPE} = \frac{100}{n} \sum_{t=1}^{n} \frac{|F_t - A_t|}{(|A_t| + |F_t|)/2}$$

Figure 1. SMAPE formula

Submissions for this competition are evaluated using Symmetric Mean Absolute Percentage Error (SMAPE), as seen in Figure 1. SMAPE is an accuracy measure based on percentage errors between the actual and forecast values, where *At* is the actual values and *Ft* is the forecast values.

Data Processing and Exploratory Data Analysis

The dataset is extremely large with 913,000 data points and relatively simple and clean with only 4 variables: ‘date’, ‘store’, ‘item’ and ‘sales’. There are no null or missing values.



	store	item	sales
count	913000.000000	913000.000000	913000.000000
mean	5.500000	25.500000	52.250287
std	2.872283	14.430878	28.801144
min	1.000000	1.000000	0.000000
25%	3.000000	13.000000	30.000000
50%	5.500000	25.500000	47.000000
75%	8.000000	38.000000	70.000000
max	10.000000	50.000000	231.000000

Figure 2. Statistics of train data

Now we visualize the daily sales of items per store of all stores to see if we notice any trend or pattern.

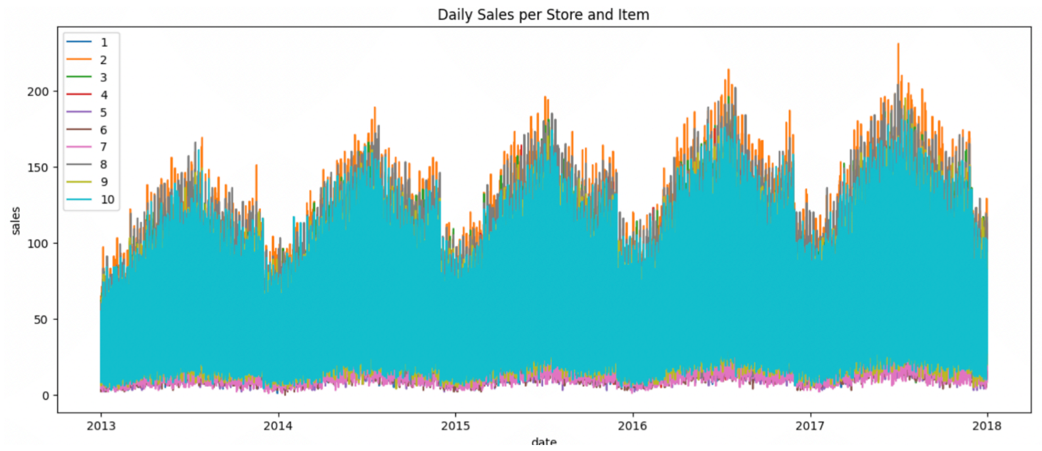


Figure 3. Daily sales per store and item

We observe a consistent pattern emerging both at the item and store levels and this pattern repeats itself yearly.

Weekly Sales Trends
A weekly analysis reveals a distinct trend in sales across all stores with sales being the lowest on Monday and the highest on Friday. This pattern suggests fluctuating customer buying trends, possibly due to work schedules, weekly planning, etc.

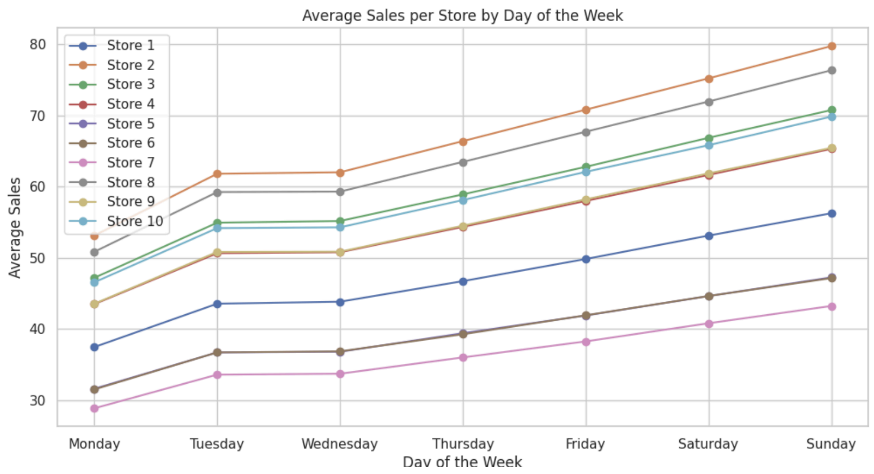


Figure 4. Average sales per store in a week

Monthly Sales Trends:
On a monthly scale, the sales data exhibits a notable pattern. Each store reaches its highest sales in July, a trend that can be owing to various factors like peak tourism seasons, vacations etc. Conversely, during the beginning of the year(January-February) and end of the year (November-December), sales are at their lowest.

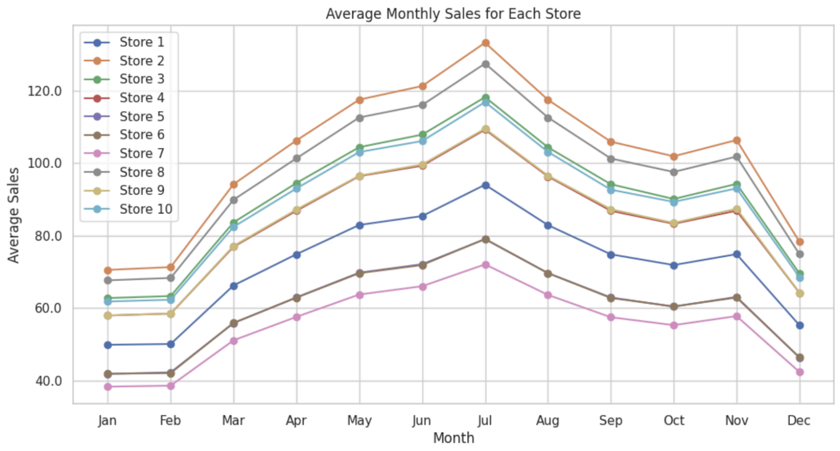


Figure 5. Average monthly sales per store

Yearly Sales Trends:
When examining the data on a yearly basis, we observe an overall sales growth yearly.

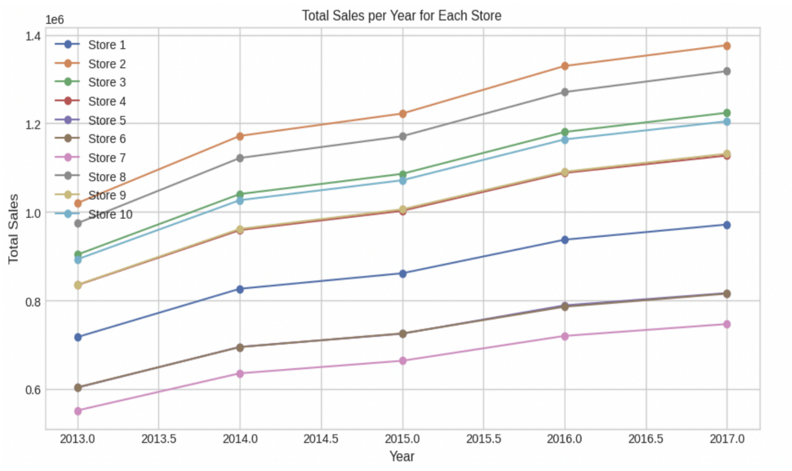


Figure 6. Total sales yearly per store

Due to consistent sales patterns across all stores and items daily,weekly,monthly and yearly, we move forward with aggregating data of stores (pooling) for daily frequency analysis and forecasting . This pooled approach simplifies analysis, enabling detailed insights and broader trend identification for effective forecasting across all stores and items.

Methodology

Approach 1: ARIMA

ARIMA stands for Auto-Regressive Integrated Moving Average and is one of the most common models used in forecasting. ARIMA consists of 3 main components, which are the Auto Regressive, Integrated, and Moving Average parts [1].

In an autoregression model, a variable is forecast using a linear combination of past values of the variable. The model formula is as seen in Figure 5, where εt represents the residual generated by the auto-regressive prediction errors, and ϕ represents the auto-regressive coefficient. This coefficient is denoted as p in ARIMA.

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t,$$

Figure 7. Auto-Regressive model formula

The integrated component represents the number of differencing we need to apply to the data to make it stationary, in the case of non-stationary data, as ARIMA assumes the stationarity of the data. This number of differencing is denoted as d in ARIMA.

Lastly, the moving average component uses past forecast errors in a regression-like model. The model formula is as seen in Figure 6, where *εt* represents white noise, and *θ* represents the moving average coefficient. This coefficient is denoted as q in ARIMA.

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q},$$

Figure 8. Moving Average model formula

These 3 components combine to form ARIMA(p, d, q), which we will use to predict the store item sales.

Implementation

In this time series analysis, we followed a structured approach to model a dataset exhibiting yearly seasonality with a peak in the middle of the year.

As ARIMA is sensitive to outliers, the topmost priority is to remove them from the train dataset and store the new data points in a copy. Moreover, since we have a very large training dataset, we perform subsampling to reduce the train dataset to only 1 store and item 1. This is done to reduce computational time and resources in addition to mitigating overfitting.

As this is a time-series dataset, the essential component to explore is the seasonality trends in the dataset for consideration in our forecasting models. On the subset of the dataset,we perform Seasonality and Trend decomposition using LOESS(STL) as a preprocessing method to understand and decompose the time-series dataset into 3 components: seasonality(recurring pattern), trend (general flow of data) and residual or noise components as shown in Figure 9.

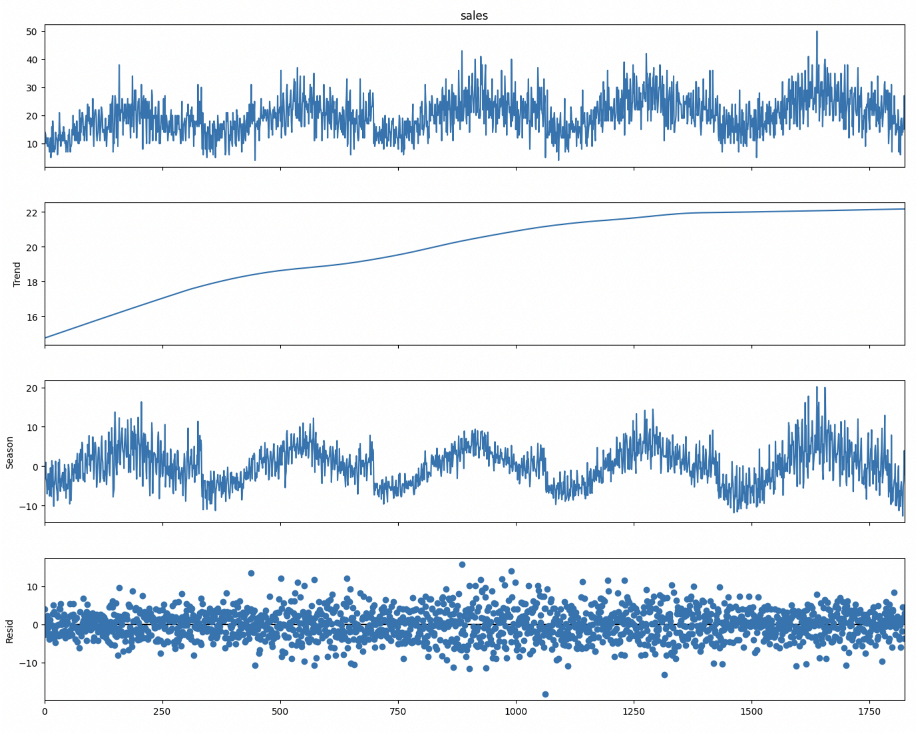


Figure 9. STL decomposition for store 1, item 1

To develop an ARIMA model for store 1, item 1, we first apply the Augmented Dickey-Fuller test, to check for stationarity of our input which is the residuals obtained from the STL decomposition. The p-value (7.827 x 10^-15) is near-zero p-value and the test statistic is highly negative indicating that the data is stationary and differencing is not needed. So d=0 for our ARIMA model.

```
The test statistics: -8.969403063862247
The p-value: 7.827809713352233e-15
```

Figure 10. Dicky-Fuller test results

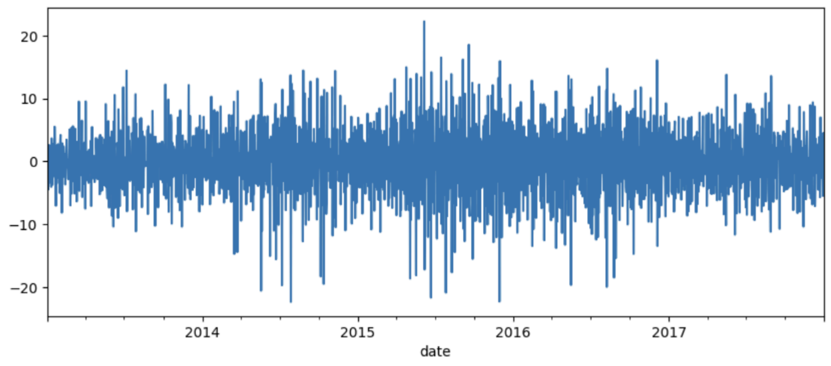


Figure 11. Residual data

To determine the optimal AR and MA orders (p and q) for our ARIMA model, we analyze the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots for 50 days or lags) on our data. We also utilize the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) to conduct a grid search to identify the optimal ARIMA model, focusing on achieving lower AIC and BIC scores. This approach helped us efficiently select parameters that best capture the data's underlying trends.

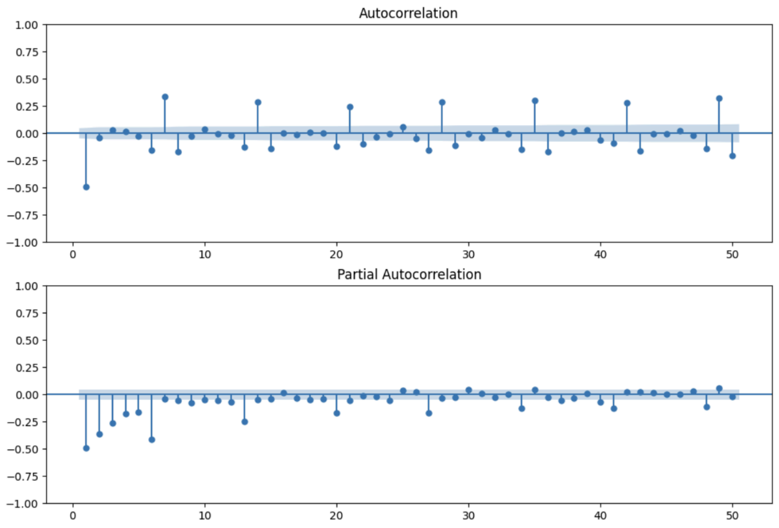


Figure 12. ACF & PACF plots

```
order_aic_bic = []

# Loop over p values from 0-10
for p in range(10):
    print('p:', p)

    # Loop over q values from 0-10
    for q in range(10):

        try:
            # create and fit ARMA(p,q) model
            model = sm.tsa.ARIMA(residuals, order=(p, 0, q))
            results = model.fit()

            # Print order and results
            order_aic_bic.append((p, q, results.aic, results.bic))
        except:
            print(p, q, None, None)

# Make DataFrame of model order and AIC/BIC scores
order_df = pd.DataFrame(order_aic_bic, columns=['p', 'q', 'aic','bic'])
```

Figure 13. Hyperparameter tuning to train the ARIMA model using grid search

After hyperparameter tuning, the optimal parameters p,d and q were chosen as **9,0,9** respectively.

Evaluation

To validate the model, we applied it to the last three months of the original train dataset subset of store 1, item 1, enabling an assessment of its performance on validation data.

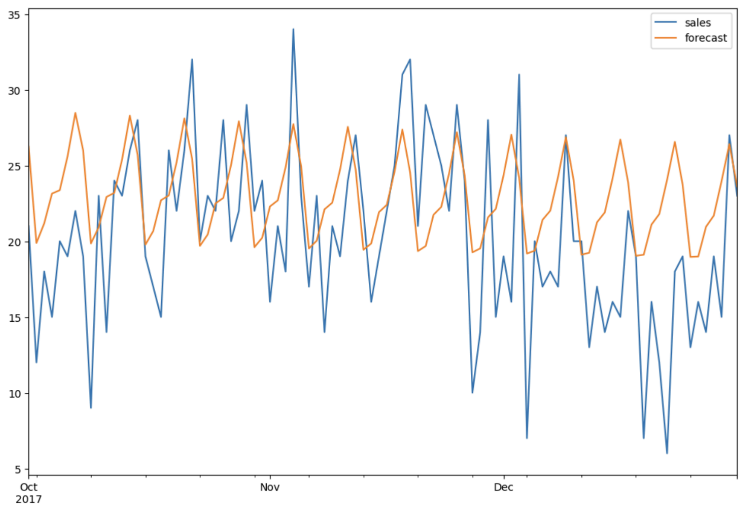


Figure 14. Forecasting of ARIMA on validation data of store 1, item 1

```
SMAPE(train_df[1734:1826]['sales'], train_df[1734:1826]['forecast'])
```

```
SMAPE:  24.405442977398597
```

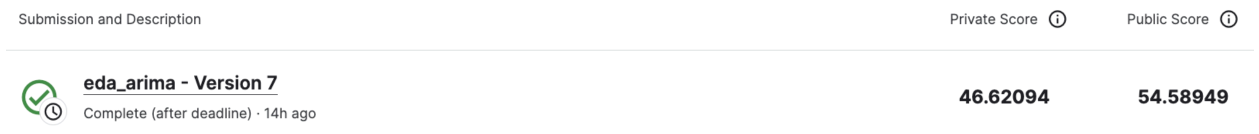
Figure 15. SMAPE for validation data for store 1 item 1

After calculating the SMAPE value to assess our model, we applied the model to each combination of store and item in our dataset following the same approach to calculate the average SMAPE of our ARIMA model across all store item combinations.

```
Total Average sMAPE for all store-item combinations: 32.73
```

Figure 15. Average SMAPE for validation data per store per item

Now when we test the model with test data and submit it to Kaggle



Submission and Description	Private Score	Public Score
eda_arima - Version 7 Complete (after deadline) · 14h ago	46.62094	54.58949

Figure 16. Performance for ARIMA model

We see that ARIMA is not performing well with this time-series data, which could be because of the following reasons:

1. ARIMA's linear nature is unable to capture non-linear relationships in the data
2. Outliers and insufficient data affect the forecasting abilities of ARIMA
3. ARIMA is not good for long-term forecasting because it uses past data and parameters influenced by human thinking

As the model is not robust enough to handle strong seasonality, we move on to Seasonal ARIMA (SARIMA) which is more suitable for data with prominent seasonal patterns such as in this dataset.

Approach 2: SARIMAX

SARIMA stands for Seasonal Auto-Regressive Integrated Moving Average, or Seasonal ARIMA. SARIMA is a common forecasting model used that improves upon the ARMIA model by accounting for seasonality trends as well.

As covered earlier, the ARIMA formula comprises the p, d, and q components for forecasting. SARIMA does two different ARIMA models at the same time, one with and one without fitting the seasonality, to understand the seasonal fluctuations. Hence, the SARIMA model consists of (p, d, q), which are the non-seasonal components, and (P, D, Q, s), which are the seasonal components [2].

For our case, we will be using SARIMAX, which is SARIMA that accounts for exogenous variables as well. Exogenous variables are external data that we will use in our forecast, such as extracted date features like the year or the day of the week. This is to ensure that we account as best as we can for the seasonality trends when forecasting.

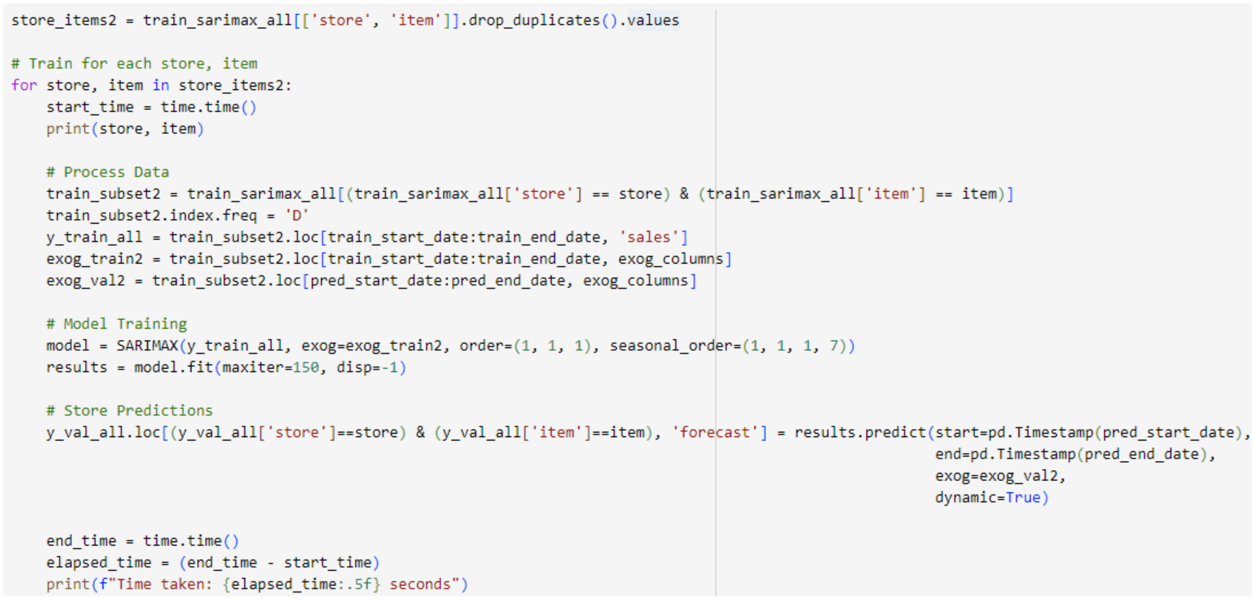
Implementation

Firstly, date and seasonality features are extracted and added to the training dataset, namely the year, day_of_week, and month_cos. These 3 features will be used as exogenous variables in training our model as they give the best model result after experimenting with other features.

Secondly, the last 3 months of the training dataset are split and kept as the validation dataset.

Thirdly, the Akaike Information Criterion (AIC) is used with the pmdarima library to loop through and select the best parameters for the SARIMAX model, as AIC considers the goodness of fit and the complexity of the model. The best parameters selected are (1, 1, 1) and (1, 1, 1, 7) for (p, d, q) and (P, D, Q, s) respectively.

Fourthly, the SARIMAX model is trained using the exogenous variables and model parameters discussed in the first and third steps, as seen in Figure 17.

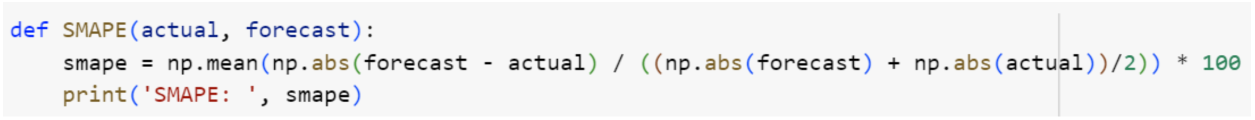


```
store_items2 = train_sarimax_all[['store', 'item']].drop_duplicates().values

# Train for each store, item
for store, item in store_items2:
    start_time = time.time()
    print(store, item)

    # Process Data
    train_subset2 = train_sarimax_all[(train_sarimax_all['store'] == store) & (train_sarimax_all['item'] == item)]
    train_subset2.index.freq = 'D'
    y_train_all = train_subset2.loc[train_start_date:train_end_date, 'sales']
    exog_train2 = train_subset2.loc[train_start_date:train_end_date, exog_columns]
    exog_val2 = train_subset2.loc[pred_start_date:pred_end_date, exog_columns]

    # Model Training
    model = SARIMAX(y_train_all, exog=exog_train2, order=(1, 1, 1), seasonal_order=(1, 1, 1, 7))
    results = model.fit(maxiter=150, disp=-1)

    # Store Predictions
    y_val_all.loc[(y_val_all['store']==store) & (y_val_all['item']==item), 'forecast'] = results.predict(start=pd.Timestamp(pred_start_date),
                                                                                                        end=pd.Timestamp(pred_end_date),
                                                                                                        exog=exog_val2,
                                                                                                        dynamic=True)

    end_time = time.time()
    elapsed_time = (end_time - start_time)
    print(f"Time taken: {elapsed_time:.5f} seconds")
```

Figure 17. Training SARIMAX Model

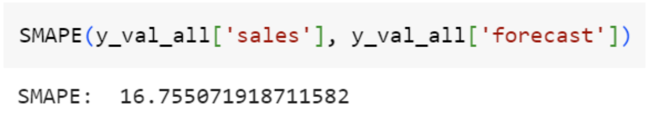
Finally, the SARIMAX model trained is used to predict the sales for the store items in the validation dataset. The SMAPE is then calculated using the actual and forecasted sales values for the store items in the validation dataset.

Evaluation



```
def SMAPE(actual, forecast):
    smape = np.mean(np.abs(forecast - actual) / ((np.abs(forecast) + np.abs(actual))/2)) * 100
    print('SMAPE: ', smape)
```

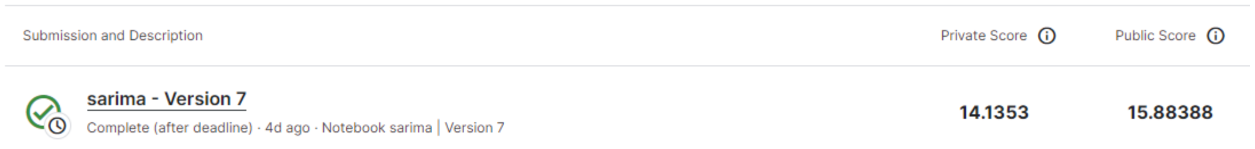
Figure 18. SMAPE function



```
SMAPE(y_val_all['sales'], y_val_all['forecast'])

SMAPE:  16.755071918711582
```

Figure 19. SMAPE score for SARIMAX model

After training the SARIMAX model against all store items and forecasting the sales for the last 3 months of the store items as validation, the SMAPE score obtained is 16.755, as seen in Figure X. However, the model seems to perform better against the test dataset, with a SMAPE score of 15.884, as seen in Figure 18.



Submission and Description	Private Score	Public Score
sarima - Version 7 Complete (after deadline) · 4d ago · Notebook sarima | Version 7	14.1353	15.88388

Figure 20. Performance for SARIMAX model

Approach 3: GAM

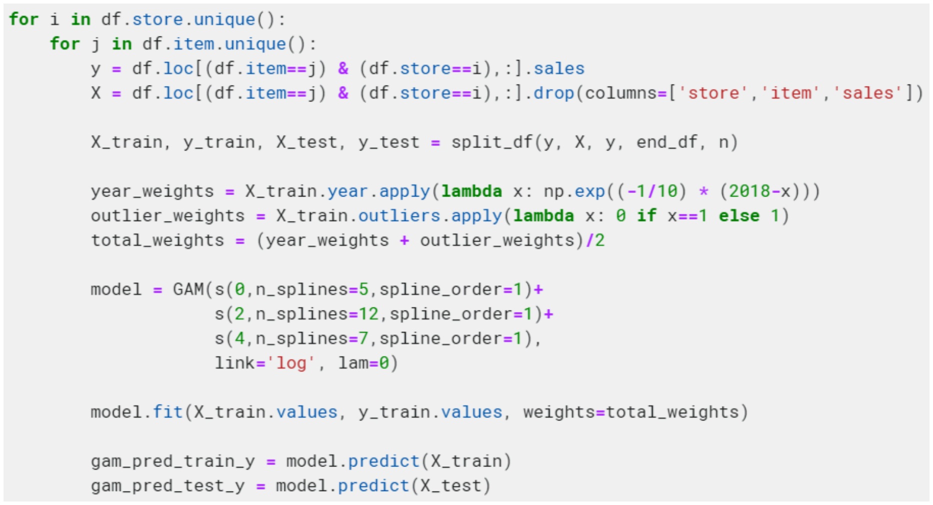
A Generalised Additive Model (GAM) is a linear model that can also learn non-linear features. Instead of using a weighted sum for the relationship, GAM uses the sum of flexible functions, known as splines, that model non-linear relationships for each feature [3]. As seen in our exploratory data analysis, the relationship of the store item demand generally follows a linear trend with non-linear seasonality. Consequently, GAM is a highly flexible model that we can use to predict the store item demand.

Implementation

Firstly, after doing exploratory data analysis, date features, namely year, month, and day_of_week are extracted and added to the training and test dataset to account for seasonality.

Next, the dataset is split into train and test datasets, with the training dataset used to train the GAM for each store item and the test dataset used for prediction.

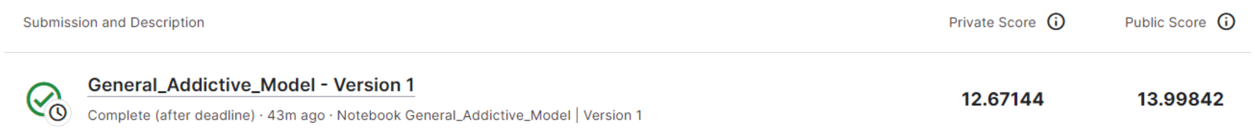
As the years 2013-2017 are used for training and the year 2018 is used for testing, an exponential weight cost function is applied to the GAM to give more weight to the recent years. Weights are also given to detected outliers to penalise the GAM. As seen in Figure 21, the features used in the GAM equation 0, 2, and 4 correspond to the year, month, and day_of_week features respectively.



```
for i in df.store.unique():
    for j in df.item.unique():
        y = df.loc[(df.item==j) & (df.store==i),:].sales
        X = df.loc[(df.item==j) & (df.store==i),:].drop(columns=['store','item','sales'])

        X_train, y_train, X_test, y_test = split_df(y, X, y, end_df, n)

        year_weights = X_train.year.apply(lambda x: np.exp((-1/10) * (2018-x)))
        outlier_weights = X_train.outliers.apply(lambda x: 0 if x==1 else 1)
        total_weights = (year_weights + outlier_weights)/2

        model = GAM(s(0,n_splines=5,spline_order=1)+
                    s(2,n_splines=12,spline_order=1)+
                    s(4,n_splines=7,spline_order=1),
                    link='log', lam=0)

        model.fit(X_train.values, y_train.values, weights=total_weights)

        gam_pred_train_y = model.predict(X_train)
        gam_pred_test_y = model.predict(X_test)
```

Figure 21. Implementing GAM

Evaluation
After using the GAM to predict the store item demand for the test data, the predictions are stored and rounded, before submitting. The resulting SMAPE for the public score is 13.998, as seen in Figure 20, and is in the top 30% of the leaderboard.



Submission and Description	Private Score	Public Score
General_Addictive_Model - Version 1 Complete (after deadline) · 43m ago · Notebook General_Addictive_Model | Version 1	12.67144	13.99842

Figure 22. Performance for GAM

Approach 4: XGBoost

XGBoost, which stands for Extreme Gradient Boosting, is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning algorithm under the Gradient Boosting framework [4].

XGBoost builds upon supervised machine learning, decision trees, ensemble learning, and gradient boosting. Gradient boosting is a powerful machine learning algorithm used to achieve state-of-the-art accuracy on a variety of tasks such as regression, classification and ranking.

The term "gradient boosting" comes from the idea of "boosting" or improving a single weak model by combining it with several other weak models to generate a collectively strong model. Gradient boosting is an extension of boosting where the process of additively generating weak models is formalized as a gradient descent algorithm over an objective function [5].

Implementation
The model is implemented using the Python library *xgboost* and the *XGBRegressor* method. In the parameters, we specified that we want 1000 trees to be generated (*n_estimators*), *early_stopping_rounds*=50 meaning if the test set does not improve after 50 trees then stop,

learning_rate=0.01 make sure it doesn't overfit too quickly, makes splits up to the *max_depth* param and then starts pruning the tree backwards and removing splits beyond which there is no positive gain, and the rest of the parameters are kept at default as seen in Figure 23.

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, early_stopping_rounds=50, gamma=0,
       learning_rate=0.01, max_delta_step=0, max_depth=3,
       min_child_weight=1, missing=None, n_estimators=1000, n_jobs=1,
       nthread=None, objective='reg:linear', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)
```

Figure 23. XGBRegressor Parameters

Feature extraction is the most important step in the process. We first extracted the date features such as day of week, and day of year, then used those calculations to tune and extract other features. We charted the features and found sales were most correlated with the mean store item sale in that week as seen in Figure 24 and the code used to generate these features in Figure 25.

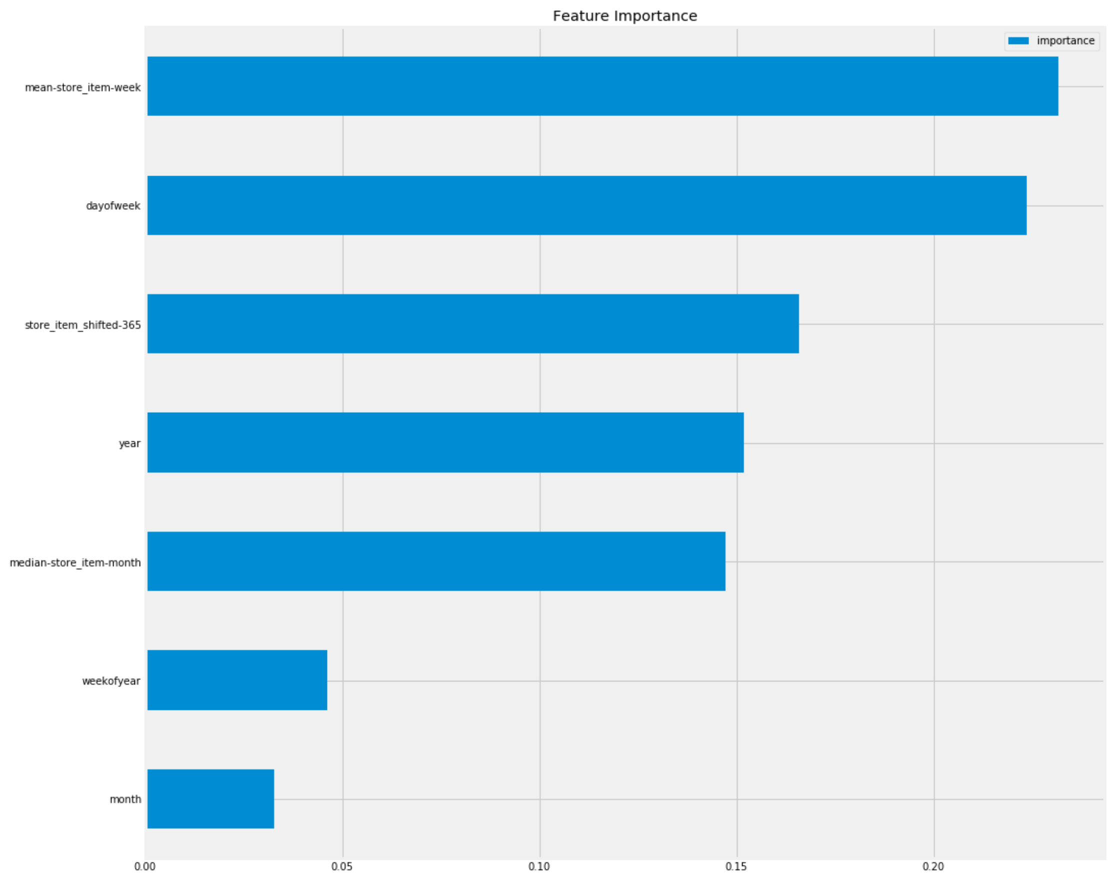
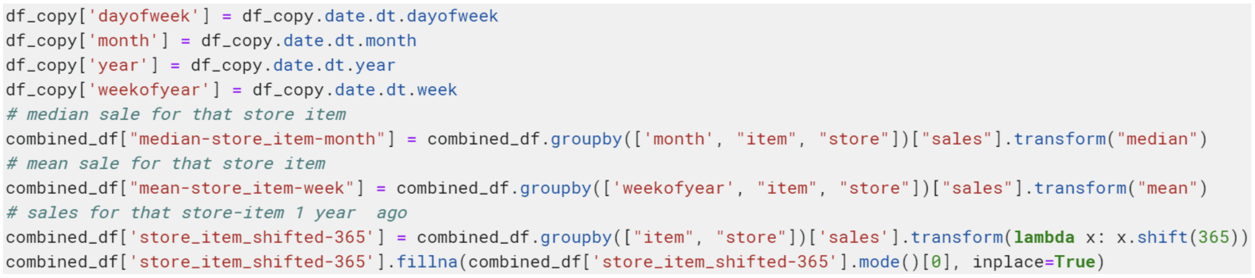


Figure 24. Feature Importance



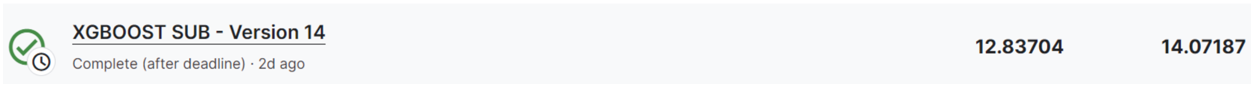
```
df_copy['dayofweek'] = df_copy.date.dt.dayofweek
df_copy['month'] = df_copy.date.dt.month
df_copy['year'] = df_copy.date.dt.year
df_copy['weekofyear'] = df_copy.date.dt.week
# median sale for that store item
combined_df["median-store_item-month"] = combined_df.groupby(['month', "item", "store"])["sales"].transform("median")
# mean sale for that store item
combined_df["mean-store_item-week"] = combined_df.groupby(['weekofyear', "item", "store"])["sales"].transform("mean")
# sales for that store-item 1 year  ago
combined_df['store_item_shifted-365'] = combined_df.groupby(["item", "store"])['sales'].transform(lambda x: x.shift(365))
combined_df['store_item_shifted-365'].fillna(combined_df['store_item_shifted-365'].mode()[0], inplace=True)
```

Figure 25. Creating Features

Split the training data into a test and training set to fit to the regressor, then calculate these features for the testing data given and then use the regression to predict the potential sales.

Evaluation

After training the XGBoost model against all store items and forecasting against the test dataset, with a SMAPE score of 14.57. Afterwards, we tuned the XGBRegressor parameters by increasing the max_depth of trees to 5. Then round up the double value for sales return after running through the trained model. The resulting SMAPE and leaderboard position was very good with 14.086 but it was even better after increasing max_depth to 7 resulting in 14.071 being in the top 35% (161 out of 456).



XGBOOST SUB - Version 14
Complete (after deadline) · 2d ago
12.83704
14.07187

Figure 26. Performance for XGBoost Model

Approach 5: LightGBM

LightGBM is a gradient-boosting framework that uses tree-based learning algorithms and is designed for distributed and efficient training, particularly on large datasets. It stands out for its use of a histogram-based algorithm that groups continuous feature values into discrete bins, which speeds up training and reduces memory usage. LightGBM supports parallel and GPU learning and has an efficient implementation of the leaf-wise growth strategy for decision trees, which often results in better performance with less memory consumption compared to depth-wise growth strategies. Its high performance and fast execution speeds make it a viable model to predict the store item demand.

Implementation

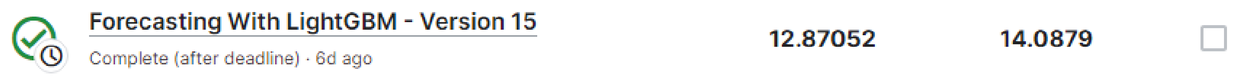
Firstly, to prepare the data for the model we conduct some Exploratory Data Analysis (EDA). Mainly analyzing the Data Structure, Sales Analysis, Visualization and Hypothesis testing.

Next, we perform some Feature Engineering, creating about 200 new features under the categories: Time-Related Features, Lagged Features, Moving Average Features, Hypothesis Testing Similarity Features and Exponentially Weighted Mean (EWM) Features.

Then, we perform an analysis of Feature Importance by running the first model and using tools, such as SHAP, to rank the features based on importance to the LightGBM Model. Following this, we run a Second Model using Feature Selection with LGBM Feature Importance.

Lastly, we have two rounds of Hyperparameter Tuning. The 1st optimization round consists of finding each hyperparameter's optimal values using RandomSearchedCV and the 2nd optimization round includes early stopping to determine the best iteration number, preventing overfitting.

Evaluation
After using the LightGBM to predict the store item demand for the test data, the predictions are stored and rounded, before submitting. The resulting SMAPE for the public score is 14.0879, as seen in Figure 27, and is in the top 35% of the leaderboard.



Forecasting With LightGBM - Version 15
Complete (after deadline) · 6d ago
12.87052   14.0879

Figure 27. Performance for LightGBM

Approach 6: Prophet

The Prophet model is a tool designed for making forecasts for time series data with strong seasonal effects and several seasons of historical data [6]. This is appropriate for our dataset which spans 5 years.

Prophet operates by decomposing the time series into three main components: Trends, Seasonality and holidays.

- The trend component captures underlying growth or decline patterns.
- Seasonality addresses regular, periodic changes.
- The holiday component accounts for predictable irregular occurrences, which does not apply to our project as the dataset excludes holiday effects or store closures.

Implementation
Firstly, to prepare the data, the date and sales columns are renamed to 'ds' and 'y' respectively, which is the required format for Prophet.

Next, a log transformation is applied to the sales data ('y') to help stabilize the variance to capture the multiplicative nature of seasonal effects better.

The Prophet model is instantiated and fitted to historical sales data from 2013 to 2017. After training, data from 2018 is used as test data to make predictions for the sales variable. The predictions are transformed back to the log scale to the original scale by applying the exponential function.

Evaluation

Overall, the model is trained separately for each store-item combination, which allows for capturing unique seasonal patterns for each combination. The resulting SMAPE for the public score is 14.198, as seen in Figure 28, and is in the top % of the leaderboard.

Prophet - Version 1
Complete (after deadline) · 14d ago · prophet 7/11

Prophet - Version 1	13.21898	14.1975

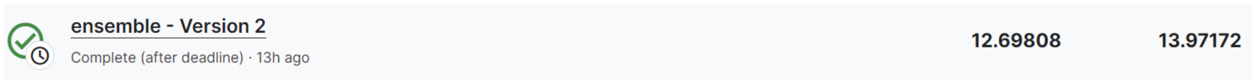
Figure 28. Performance for Prophet

Approach 7: Model Ensembling

Ensemble methods in machine learning are techniques that combine the predictions of multiple models to improve the overall performance and accuracy of the model. The key idea is that by aggregating the predictions of multiple models, the strengths of one model can compensate for the weaknesses of another, leading to more robust and accurate predictions [7].

7.1 Averaging

We combined the results from our 4 best performing models: XGBoost, GAM, LightGBM, and Prophet, by taking the mean of the predicted sales for each ID and the resulting prediction is slightly better than our best model (GAM), showing that with more tweaks this could be a viable way to slightly improve our model even more once we have trained various models.



ensemble - Version 2
Complete (after deadline) · 13h ago

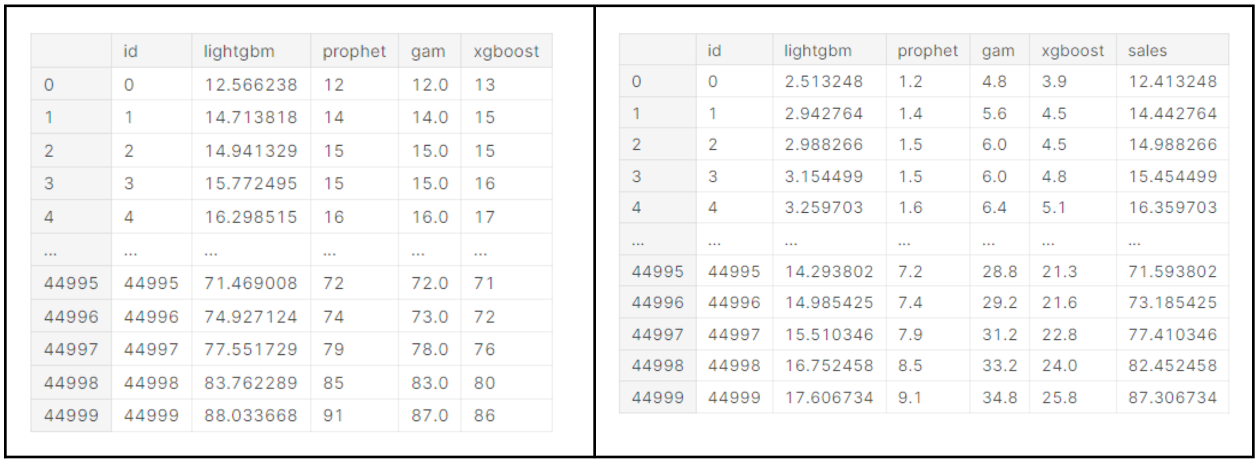
ensemble - Version 2	12.69808	13.97172

Figure 29. Performance for Ensemble by Mean

7.2 Rank Averaging

Similar to averaging, but instead of giving every model an equal weight, we rank each model based on its performance and calculate its weights based on its ranks [8]. In our case, our models' performance ranks from GAM, XGBoost, LightGBM, and Prophet in terms of best-performing to worst-performing. As we have 4 models in our ranking, their weights neatly come out to be 0.4, 0.3, 0.2, and 0.1 respectively. We applied these weights to the models' predictions and obtained a slightly better score of 13.964.

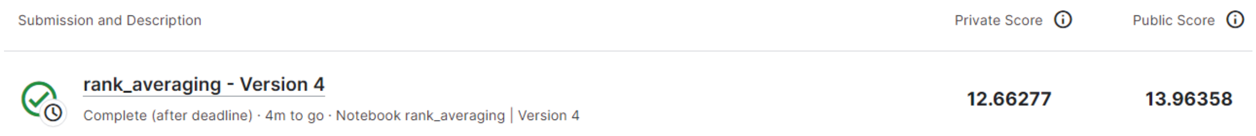
Before rank averaging	After rank averaging



	id	lightgbm	prophet	gam	xgboost
0	0	12.566238	12	12.0	13
1	1	14.713818	14	14.0	15
2	2	14.941329	15	15.0	15
3	3	15.772495	15	15.0	16
4	4	16.298515	16	16.0	17
...	...	...	...	...	...
44995	44995	71.469008	72	72.0	71
44996	44996	74.927124	74	73.0	72
44997	44997	77.551729	79	78.0	76
44998	44998	83.762289	85	83.0	80
44999	44999	88.033668	91	87.0	86

	id	lightgbm	prophet	gam	xgboost	sales
0	0	2.513248	1.2	4.8	3.9	12.413248
1	1	2.942764	1.4	5.6	4.5	14.442764
2	2	2.988266	1.5	6.0	4.5	14.988266
3	3	3.154499	1.5	6.0	4.8	15.454499
4	4	3.259703	1.6	6.4	5.1	16.359703
...	...	...	...	...	...	...
44995	44995	14.293802	7.2	28.8	21.3	71.593802
44996	44996	14.985425	7.4	29.2	21.6	73.185425
44997	44997	15.510346	7.9	31.2	22.8	77.410346
44998	44998	16.752458	8.5	33.2	24.0	82.452458
44999	44999	17.606734	9.1	34.8	25.8	87.306734

Figure 30. Before and After Rank Averaging



Submission and Description	Private Score	Public Score
rank_averaging - Version 4 Complete (after deadline) · 4m to go · Notebook rank_averaging | Version 4	12.66277	13.96358

Figure 31. Performance for Ensemble by Rank Averaging

7.3 Stacking - Meta Model

Stacking is an advanced ensemble learning technique which combines multiple prediction models to improve the overall performance of a predictive task. The stacking process involves using the predictions of our four distinct forecasting models (Prophet, LightGBM, GAM, and XGB) as input features for a higher-level, meta-model. LightGBM was chosen as our meta model due to its high efficiency, speed and inclusion of regularisation.

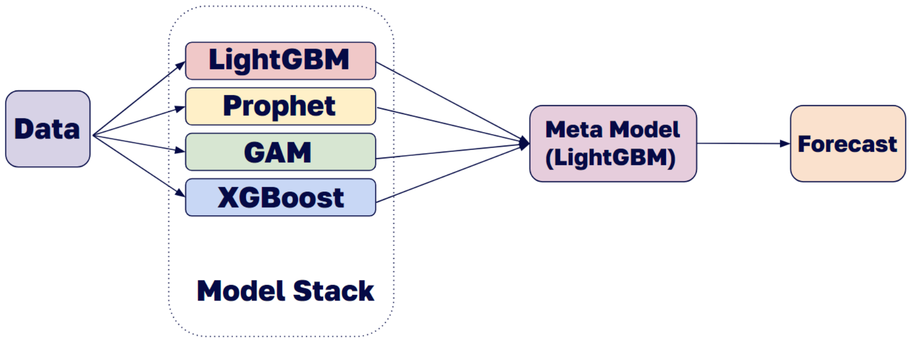


Figure 32. Stacking

This approach leverages the combined strengths of these models to enhance prediction accuracy in time series forecasting. The process includes data preprocessing, feature creation, hyperparameter tuning using a randomized search as well as early stopping.

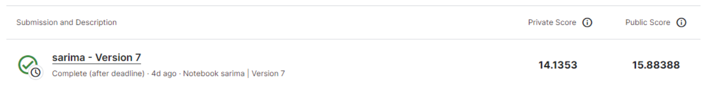
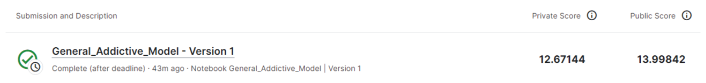
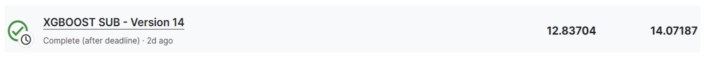
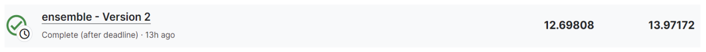
The individual models were trained on data from 2013-2016 and predicted on the validation dataset consisting of data from 2017. The predictions from the validation dataset were used as features to train the meta model. Finally, the meta model is used to make the final sales prediction on the test dataset with dates in 2018 and a SMAPE public score of 14.10244 was obtained.

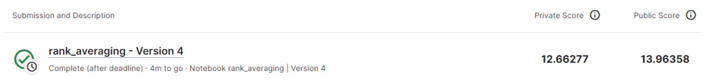
However, our results were not as significant, and we concluded that this method of model ensembling has overtrained the model due to the complexity of the base models and will not be as effective as forecasting compared to our base models.

Meta Model - Version 8
Complete (after deadline) · now
12.84963 14.10244

Figure 33. Performance for Ensemble by Meta Model

Results

Method	Performance	Public Rank (out of 459)
ARIMA	eda_arima - Version 7 Complete (after deadline) · now 46.62094 54.58949 Public Score: 54.58949	446
SARIMAX	 Submission and Description Private Score Public Score sarima - Version 7 Complete (after deadline) · 4d ago · Notebook sarima | Version 7 14.1353 15.88388 Public Score: 15.88388	310
GAM	 Submission and Description Private Score Public Score General_Addictive_Model - Version 1 Complete (after deadline) · 43m ago · Notebook General_Addictive_Model | Version 1 12.67144 13.99842 Public Score: 13.99842	137
XGBoost	 XGBOOST SUB - Version 14 Complete (after deadline) · 2d ago 12.83704 14.07187 Public Score: 14.07187	161
LightGBM	Forecasting With LightGBM - Version 15 Complete (after deadline) · 6d ago 12.87052 14.0879 Public Score: 14.0879	164
Prophet	Prophet - Version 1 Complete (after deadline) · 14d ago · prophet 7/11 13.21898 14.1975 Public Score: 14.1975	197
Averaging	 ensemble - Version 2 Complete (after deadline) · 13h ago 12.69808 13.97172 Public Score: 13.971772	132

Rank Averaging	 Submission and Description · Private Score · Public Score rank_averaging - Version 4 Complete (after deadline) · 4m to go · Notebook rank_averaging | Version 4 12.66277 · 13.96358 **Public Score: 13.96358**	**130**
Stacking (Meta Model)	Meta Model - Version 8 Complete (after deadline) · now 12.84963 · 14.10244 Public Score: 14.10244	171

Conclusion

Throughout this project, we deepened our understanding and skills in several key areas:

- **Exposure to a variety of models**
  We explored a range of models, from traditional time series models like ARIMA and SARIMAX to more advanced machine learning algorithms like XGBoost and LightGBM. Beyond the 6 models in our methodology, we also researched other models not included in our project. This exposure allowed us to appreciate the unique strengths and weaknesses of different modelling approaches in forecasting contexts.

- **Data Preprocessing for models**
  Each model we used required its specific form of data preprocessing, where we had to learn to adapt our dataset to fit the requirements of these models.

- **Balancing complexity and performance**
  We learnt to balance model complexity with the quality of predictions by tuning hyperparameters to optimise each model's performance. Fine-tuning of hyperparameters played a pivotal role in enhancing model accuracy and preventing overfitting, thereby striking a balance between complexity and predictive efficacy.

- **Exploring Ensemble learning**
  By combining the predictions of our 4 best performing models through stacking, using LightGBM as a meta model, we leveraged the collective strengths of multiple algorithms. This strategy of ensemble learning improved predictive accuracy and provided insights into the power of ensemble techniques for forecasting tasks.

Even with a simple and clean dataset, utilising different machine learning techniques will yield different results. Although ARIMA and SARIMAX are the more common forecasting models, they performed generally worse overall than more complex models such as XGBoost and LightGBM. Hence, there is no single best model to solve the problem. The performance of different models varies largely based on the method of implementation and such. Therefore, we believe that when implementing various machine learning models, the best approach is to combine these models to leverage their strengths for overall performance.

References

[1] V. Xavier, ‘ARIMA/SARIMA — Demand Forecasting Tutorial’, Medium. Accessed: Nov. 18, 2023. [Online]. Available: https://medium.com/@vitor_xavier14/arima-sarima-demand-forecasting-tutorial-81670107b692https://medium.com/analytics-vidhya/simple-weighted-average-ensemble-machine-learning-777824852426

[2] B. Artley, ‘Time Series Forecasting with ARIMA , SARIMA and SARIMAX’, Medium. Accessed: Nov. 18, 2023. [Online]. Available: https://towardsdatascience.com/time-series-forecasting-with-arima-sarima-and-sarimax-ee61099e78f6

[3] A. Shafi, ‘What is a Generalised Additive Model?’, Medium. Accessed: Nov. 18, 2023. [Online]. Available: https://towardsdatascience.com/generalised-additive-models-6dfbedf1350a

[4] ‘What is XGBoost?’, NVIDIA Data Science Glossary. Accessed: Nov. 18, 2023. [Online]. Available: https://www.nvidia.com/en-us/glossary/data-science/xgboost/

[5] A. Says, ‘Gradient Boosting, Decision Trees and XGBoost with CUDA’, NVIDIA Technical Blog. Accessed: Nov. 18, 2023. [Online]. Available: https://developer.nvidia.com/blog/gradient-boosting-decision-trees-xgboost-cuda/

[6] ‘Welcome to Prophet — Prophet 0.1.0 documentation’. Accessed: Nov. 21, 2023. [Online]. Available: https://prophet.readthedocs.io/en/latest/

[7] H. Singh, ‘Basic Ensemble Techniques in Machine Learning’, Analytics Vidhya. Accessed: Nov. 18, 2023. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/03/basic-ensemble-technique-in-machine-learning/

[8] J. Jiang, ‘Simple Weighted Average Ensemble | Machine Learning’, Analytics Vidhya. Accessed: Nov. 18, 2023. [Online]. Available: https://medium.com/analytics-vidhya/simple-weighted-average-ensemble-machine-learning-777824852426