**A PROJECT REPORT ON**

**Walmart Sales Prediction**

A project report submitted in fulfillment for the Diploma Degree in AI & ML Under

Applied Roots with University of Hyderabad



Project submitted by

**Arun Kumar C S**

Under the Guidance of

Mentor: Saugata Paul

Approved by: Mentor: Saugata Paul



**University of Hyderabad**

***Declaration of Authorship***

We hereby declare that this thesis titled “Walmart Sales Prediction” and the work presented by the undersigned candidate, as part of Diploma Degree in AI & ML.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

**Name:** Arun Kumar C S

**Thesis Title:** Walmart Sales Prediction

**CERTIFICATE OF RECOMMENDATION**

We hereby recommend that the thesis entitled “Walmart Sales Prediction” prepared under my supervision and guidance by Arun Kumar C S be accepted in fulfillment of the requirement for awarding the degree of Diploma in AI & ML Under applied roots with University of Hyderabad. The project, in our opinion, is worthy of its acceptance.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Mentor: Saugata Paul

**Under Applied roots with**

****

**University of Hyderabad**

**ACKNOWLEDGEMENT**

Every project big or small is successful largely due to the effort of a number of wonderful people who have always given their valuable advice or lent a helping hand. I sincerely appreciate the inspiration; support and guidance of all those people who have been instrumental in making this project a success. I, Arun Kumar C S student of applied roots, is extremely grateful to mentors for the confidence bestowed in me and entrusting my project entitled “Walmart Sales Prediction” with special reference.

At this juncture, I express my sincere thanks to Mentor: Saugata Paul of applied roots for making the resources available at the right time and providing valuable insights leading to the successful completion of our project who even assisted me in completing the project.

**Name: Arun Kumar C S**

**Contents**

1. **Introduction**
2. **Literature Review**
3. **Data Description**
4. **Exploratory Data Analysis**
5. **Metric**
6. **Modeling and Error Analysis**
7. **Advanced Modeling and Feature Engineering**
8. **Discussion of Results**
9. **Conclusion**
10. **References**
11. **Productionizatino and Deployment**

**Abstract:**

Accurately forecasting the demand is essential for a retail company to deliver the products on time, and arrange well in advance for various inputs. It enables organizations to make better business decisions such as planning inventory, managing funds, deciding the price of the product, etc. In this project, we will use the power of Machine Learning to improve the forecasting accuracy of items sold in retail stores which can also handle varying demands based on the type of calendar events.

**1. INTRODUCTION**

Whether it be clearance sales in supermarkets or essential items going out of stock, it is a problem poorly forecasting the demand. If the companies know how many items are needed in advance accurately, they can make decisions on making them available to the customers. The aim of the case study is to accurately predict how many items are going to be sold at each store located at different geographical locations for a period of 28 days using the previous 5-year sales data. Solving this problem will improve the customer reputation and provide a platform that the customers can rely on. It reduces the wastage of resources, can plan on when to buy an item in bulk, etc. Bringing a better forecasting technique for this problem can be used to set up appropriate inventory levels, reduce wastage and prepare for the incoming demand which helps with risk management and generating profits for the company. The organization can also provide better prices to the customers while maintaining good availability of the product which overall improves customer satisfaction.

**2. LITERATURE REVIEW**

**1. Kaggle Fourth place solution [1]:**

@monsaraida’s tried a solution which is real world oriented, he used single LGBM model with tweedie as objective with time based cv. He didn’t used any post processing multipliers or recursive solution (to avoid error accumulation). He used a model split for each store for each week as shown in figure. The figure is detailed in explanation regarding the entire solution in terms of features, model and solution.

**2. Kaggle Fifth place solution [2]:**

@lahoudalan used LGBM with poisson as objective. I like his feature engineering, he used 30 features denoting 30 events where the value will be according to the remaining days for the event from that day (close to the event has higher value with max 30, distant events has low value with min 0). He decided to remove price of the item in the model, instead he used price difference between weeks as feature, so that rise in price would make more sense. Instead of using post processing magic multiplier of 1.03 to the entire model, he used magic multiplier for each store/department. He didn’t used it for each item to avoid overfitting.

**3. Kaggle First Place Solution [3]:**

Yeonjun In used Multiple LGBM models with tweedi.e as objective. He grouped data by stores, category and department to create multiple LGBM models. Cross validation was done using time based split with d1578-d1605, d1830-d1857, d1858-d1885, d1886-d1913. Final evaluation was done on d1914-d1941. He didn’t used any magic multipliers as suggested by @kyakovlev (who gave valuable suggestions throughout the competition, and came up with the idea of lightGBM with tweedie). He also used recursive and non-recursive solution. (I am not clear with the understanding of this, maybe it will become clear when I work with the codes.)

**4. Kaggle Third Place Solution [4]:**

Jeon used DeepAR which is a multiple LSTM model with tweedie as loss to solve this problem. He used features such as Lag 1 value, moving averages of 7, 28 days, Normalized calendar features [-0.5,0.5], event name and type using embedding, SNAP feature, raw/normalized by time/normalized within department, category and continuous zero sale days until today. They tried using CNN, Transformers but LSTM based DeepAR worked better than other DL techniques.

**3. DATA DESCRIPTION**

The dataset contains 3049 different products classified into 3 product categories (Food, Household, and Hobbies) and 7 product departments. The data covers products sold across 10 stores in the US from the state of California, Texas, and Wisconsin. The features include item level, product category, department, and store details along with the time series data.

The time-series data has 1941 days of data from **2011-01-29** to **2016-06-19**.

**3.1 Source of the dataset:**

The dataset is made available by Walmart to The Makridakis Open Forecasting Center (MOFC) at the University of Nicosia. The dataset is available to download at Kaggle in CSV format.

**3.2 Features**

Along with time-series data along with explanatory variables such as price, day of the week, promotions, and events like Christmas, Valentine’s day, etc.

1. The file “sell\_prices.csv” contains information about the price of the products sold per store and date.
   1. store\_id: The id of the store where the product is sold.
   2. item\_id: The id of the product.
   3. wm\_yr\_wk: The id of the week.
   4. sell\_price: The price of the product for the given week/store. The price is provided per week (average across seven days). If not available, this means that the product was not sold during the examined week. Note that although prices are constant a weekly basis, they may change over time (both training and test set).
2. The file **“calendar.csv”**Contains information about the dates the products are sold.
3. date: The date in a “y-m-d” format.
4. wm\_yr\_wk: The id of the week the date belongs to.
5. weekday: The type of the day (Saturday, Sunday, …, Friday).
6. wday: The id of the weekday, starting from Saturday.
7. month: The month of the date.
8. year: The year of the date.
   * 1. event\_name\_1: If the date includes an event, the name of this event.
     2. event\_type\_1: If the date includes an event, the type of this event.
     3. event\_name\_2: If the date includes a second event, the name of this event.
     4. event\_type\_2: If the date includes a second event, the type of this event.
     5. snap\_CA, snap\_TX, and snap\_WI: A binary variable (0 or 1) indicating whether the stores of CA, TX or WI allow SNAP purchases on the examined date. 1 indicates that SNAP purchases are allowed.

3. The file **“sales\_train\_validation.csv”** contains the historical daily unit sales data per product and store.

1. item\_id: The id of the product.
2. dept\_id: The id of the department the product belongs to.
3. cat\_id: The id of the category the product belongs to.
4. store\_id: The id of the store where the product is sold.
5. state\_id: The State where the store is located. d\_1, d\_2, …, d\_i, … **d\_1941**: The number of units sold at day i, starting from 2011-01-29.

4. The file **“sales\_train\_evaluation.csv”** has the same features but contains more days from d\_1 to d\_1941 for evaluation.

**3.3 Tools for data processing**

The data manipulation and pre-processing can be done using Pandas and Numpy.

**3.4 Data Challenges**

The entire dataset is less than 500 MB. Given that Google Collab provides up to 16 GB of RAM for the public, there seem no challenges to process the data. Also, we have sufficient data of 5 years from different stores at various locations. The dataset is self-sufficient to come up with a model.

**4. EXPLORATORY DATA ANALYSIS**

**4.1 High Level Overview**

The data available for us is in five separate csv files.

1. sell\_prices.csv
2. calendar.csv
3. sales\_train\_validation.csv
4. sales\_train\_evaluation.csv
5. sample\_submission.csv

We will import these files into memory as pandas.DataFrame object. The information on relevant files are summarized in Table 1.1.

*Table 1.1: Details of Files*

| **Filename** | **No. of Rows** | **No. of Columns** | **Memory Usage of dataframe** | **No. of NaN values** |
| --- | --- | --- | --- | --- |
| sell\_prices.csv | 6841121 | 4 | 208.77 MB | 0 |
| sales\_train\_evaluation.csv | 30490 | 1947 | 452.91 MB | 0 |
| calendar.csv | 1969 | 14 | 0.21 MB | 7542 |

The *sales\_train\_validation.csv* contains the same data as *sales\_train\_evaluation.csv* but has less number of days. Also, the *sample\_submission.csv* file contains the submission format for kaggle competition held as M5 Accuracy competition. Detailed information of the files in Table 1.1 is given in Table 1.2, 1.3 and 1.4

*Table 1.2 : Columns in sell\_prices.csv*

| **column** | **datatype** | **# Unique** | **# Null** | **Max Value** | **Min Value** | **Mean** | **Median** | **S.D** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| wm\_yr\_wk | int64 | 282 | 0 | 11621 | 11101 | 11382.9 | 11411 | 148.61 |
| sell\_price | float64 | 1048 | 0 | 107.32 | 0.01 | 4.41095 | 3.47 | 3.40881 |
| store\_id | object | 10 | 0 | N/A | N/A | N/A | N/A | N/A |
| item\_id | object | 3049 | 0 | N/A | N/A | N/A | N/A | N/A |

This dataframe contains the sell price of each item in a particular store at a given week.

*Table 1.3 : Columns in sales\_trian\_evaluation.csv*

| **column** | **datatype** | **# Unique** | **# Null** | **Max** | **Min** | **Mean** | **Median** | **S.D** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| id | object | 30490 | 0 | N/A | N/A | N/A | N/A | N/A |
| item\_id | object | 3049 | 0 | N/A | N/A | N/A | N/A | N/A |
| dept\_id | object | 7 | 0 | N/A | N/A | N/A | N/A | N/A |
| cat\_id | object | 3 | 0 | N/A | N/A | N/A | N/A | N/A |
| store\_id | object | 10 | 0 | N/A | N/A | N/A | N/A | N/A |
| state\_id | object | 3 | 0 | N/A | N/A | N/A | N/A | N/A |
| d\_1 | int64 | 84 | 0 | 360 | 0 | 1.07022 | 0 | 5.12669 |
| d\_2 | int64 | 82 | 0 | 436 | 0 | 1.04129 | 0 | 5.36547 |
| d\_3 | int64 | 72 | 0 | 207 | 0 | 0.780026 | 0 | 3.66745 |
| … | … | … | … | … | … | …. | … | 4…. |
| d\_1939 | int64 | 64 | 0 | 110 | 0 | 1.39561 | 0 | 3.51432 |
| d\_1940 | int64 | 67 | 0 | 156 | 0 | 1.68967 | 1 | 4.08921 |
| d\_1941 | int64 | 76 | 0 | 117 | 0 | 1.78216 | 1 | 4.28436 |

This dataframe contains sales information from d\_1 to d\_1941 with ids of different items, department, categories, stores and state.

*Table 1.4 : Columns in calendar.csv*

| **column** | **datatype** | **# Unique** | **# Null** | **Max** | **Min** | **Mean** | **Median** | **S.D** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| date | object | 1969 | 0 | N/A | N/A | N/A | N/A | N/A |
| wm\_yr\_wk | int64 | 282 | 0 | 11621 | 11101 | 11347.1 | 11337 | 155.277 |
| weekday | object | 7 | 0 | N/A | N/A | N/A | N/A | N/A |
| wday | int64 | 7 | 0 | 7 | 1 | 3.99746 | 4 | 2.00114 |
| month | int64 | 12 | 0 | 12 | 1 | 6.32555 | 6 | 3.41686 |
| year | int64 | 6 | 0 | 2016 | 2011 | 2013.29 | 2013 | 1.5802 |
| d | object | 1969 | 0 | N/A | N/A | N/A | N/A | N/A |
| event\_name\_1 | object | 31 | 1807 | N/A | N/A | N/A | N/A | N/A |
| event\_type\_1 | object | 5 | 1807 | N/A | N/A | N/A | N/A | N/A |
| event\_name\_2 | object | 5 | 1964 | N/A | N/A | N/A | N/A | N/A |
| event\_type\_2 | object | 3 | 1964 | N/A | N/A | N/A | N/A | N/A |
| snap\_CA | int64 | 2 | 0 | 1 | 0 | 0.330117 | 0 | 0.470374 |
| snap\_TX | int64 | 2 | 0 | 1 | 0 | 0.330117 | 0 | 0.470374 |
| snap\_WI | int64 | 2 | 0 | 1 | 0 | 0.330117 | 0 | 0.470374 |

This dataframe contains calendar information such as date, week numbers, event names, event types and whether there is a snap day on that particular date for a state.**4.2 Transforming the data and Feature Extraction**

Inorder to aid our EDA and modeling process, we can transform the data to a convenient format where all the features are captured.

To do this firstly we will ‘unpivot’ our data using dataframe.melt() method. Then we will merge this data with the calendar dataframe on ‘d’ (days d\_1, d\_2 etc..). Finally we will merge sell\_prices on this merged dataframe using multiple keys ('store\_id', 'item\_id', 'wm\_yr\_wk'). The final dataframe will have 59,181,090 rows and 8 columns.

From this data we will combine our snap\_CA, snap\_TX and snap\_WI features into a single ‘snap’ feature as each row contains only one store information on a particular day. Note that type conversions were done in order to format the dataframe better. For example, the date column was converted from string data type to numpy.datetime64 datatype.

There were missing values in sell\_price columns, this is because the products were not available in the store yet (see Figure 3.3) We can use this information for EDA. This information is captured as another column is\_product\_available - A value of 0 indicates unavailability (Nan value of sell\_price column) and a value of 1 indicates the product is available.

Additionally, Label encoding was done on item\_id, dept\_id, cat\_id, store\_id, state\_id, event\_name\_1, event\_type\_1, event\_name\_2, event\_type\_2. An example of encoding is given in Table 2.1. The details of the final dataframe is summarized in Table 2.2.

*Table 2.1: Label encoding on store\_id*

| *store\_id* | *CA\_1* | *CA\_2* | *CA\_3* | *CA\_4* | *TX\_1* | *TX\_2* | *TX\_3* | *WI\_1* | *WI\_2* | *WI\_3* |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *encoded* | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* |

*Table 2.2: Combined dataframe after label encoding*

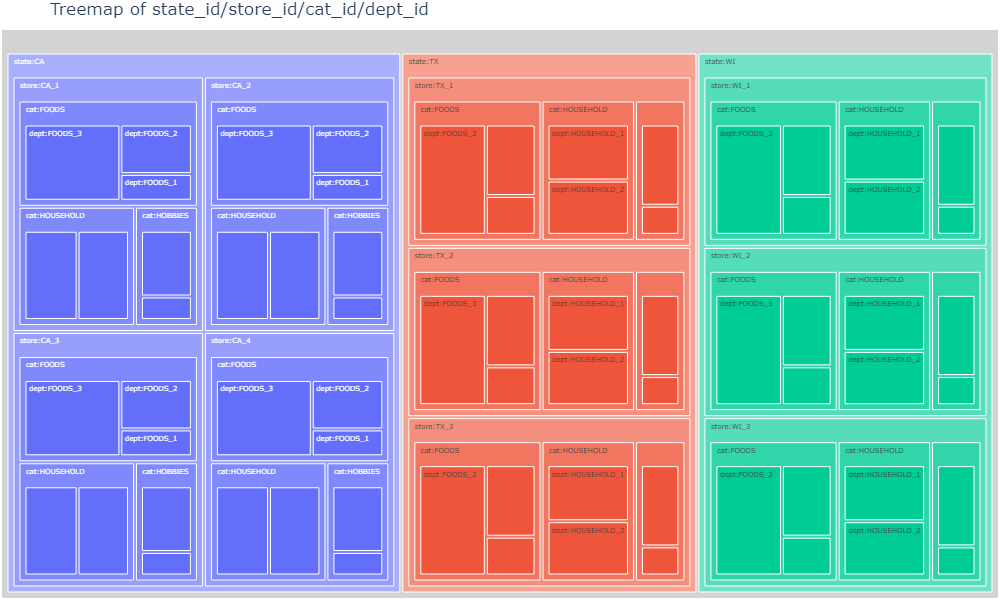
| **column** | **datatype** | **# Unique** | **# Null** | **Max** | **Min** |
| --- | --- | --- | --- | --- | --- |
| id | object | 30490 | 0 | N/A | N/A |
| item\_id | int16 | 3049 | 0 | 3048 | 0 |
| dept\_id | int8 | 7 | 0 | 6 | 0 |
| cat\_id | int8 | 3 | 0 | 2 | 0 |
| store\_id | int8 | 10 | 0 | 9 | 0 |
| state\_id | int8 | 3 | 0 | 2 | 0 |
| d | int16 | 1941 | 0 | 1941 | 1 |
| sales | int16 | 419 | 0 | 763 | 0 |
| date | object | 1941 | 0 | N/A | N/A |
| wm\_yr\_wk | int16 | 278 | 0 | 11617 | 11101 |
| wday | int8 | 7 | 0 | 7 | 1 |
| event\_name\_1 | int8 | 31 | 0 | 30 | 0 |
| event\_type\_1 | int8 | 5 | 0 | 4 | 0 |
| event\_name\_2 | int8 | 5 | 0 | 4 | 0 |
| event\_type\_2 | int8 | 3 | 0 | 2 | 0 |
| sell\_price | float32 | 1037 | 0 | 107.32 | 0 |
| snap | int8 | 2 | 0 | 1 | 0 |
| is\_product\_available | int8 | 2 | 0 | 1 | 0 |

Memory usage of data was reduced from ~7.8 GB to ~2.6 GB after label encoding and data type conversion. The transformed data and encoded labels are saved as pickle files to save time.

**4.3 Exploration of Time Series Data**

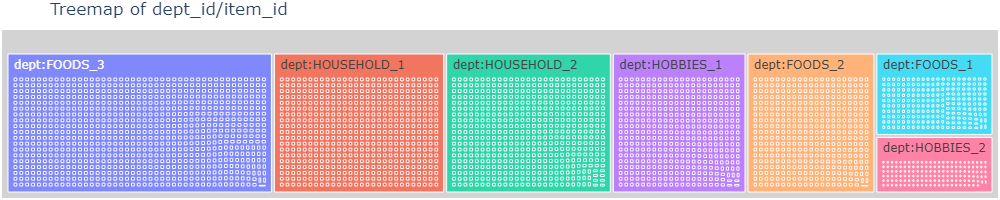
**Exploring the hierarchy**

The hierarchy can be visualized in the data using plotly treemaps. The visualization is presented below:



*Figure 3.1: Treemap of hierarchical categorization of data from state level to dept level.*

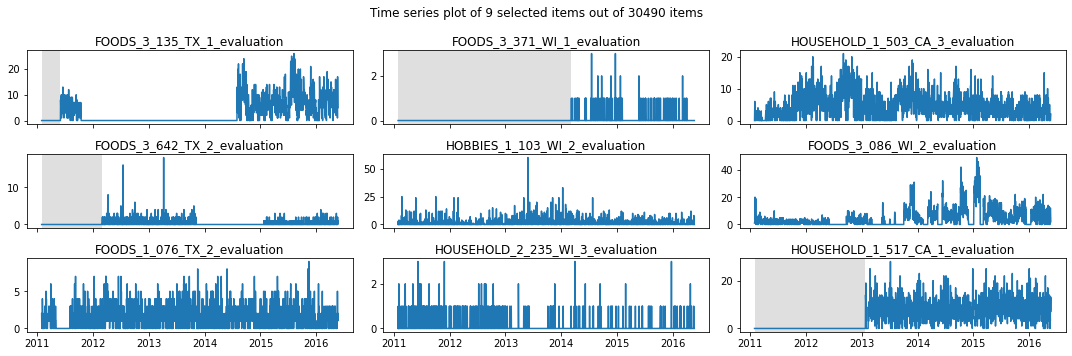
Each small block represents each department across various categories in various stores in California, Texas, and Wisconsin. Each category is furtther subdivided into various items.



*Figure: 3.2: Treemap of hierarchical categorization of data on dept level*

**Exploring time series**

The following plot is explored by selecting random time series from 30490 series and observing various patterns, some of the interesting patterns we can observed through the data is visualized in Figure 3.3

****

*Figure 3.3: Subplot of selected items (gray area indicates product is not available)*

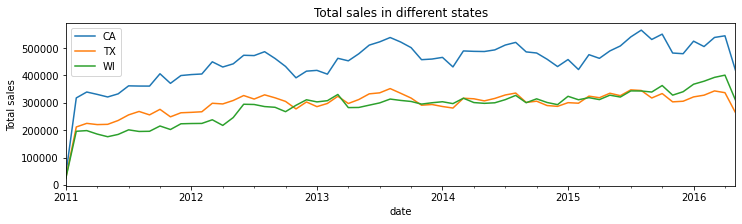
Some of the observations from our data are:

* The individual time series are non stationary.
* Even for products having sales, sometimes sales drop to zero. This could be because of our product going out of stock, but not necessarily as this product is sold again after months or even after years after having lots of zero sales. For example: FOODS\_3\_135\_TX\_1\_evaluation, FOODS\_3\_642\_TX\_2\_evaluation, FOODS\_1\_076\_TX\_2\_evaluation
* Some products are not available from d\_1 itself, so a lot of values in the beginning are zero, maybe because these products are not made available in store.
* Some low selling products are often sold 0-2 times in a day, forecasting the sales in this case can be a challenge. eg: HOUSEHOLD\_2\_235\_WI\_3\_evaluation (row-3, column-2)
* Unpredictable irregularities in the sales. eg: FOODS\_3\_086\_WI\_2\_evaluation (row-2,column-3)
* The gray region indicates the sell prices were null on these dates. This may be because there were no sell\_price since there were no products. This feature could act as a mask if we need to get the launch date of the product in a particular store.
* - We could also use the null information to separate out of stock dates from the rest of the dates for high selling products if needed.

**4.4 Sales Analysis**

**Total monthly sales by State**

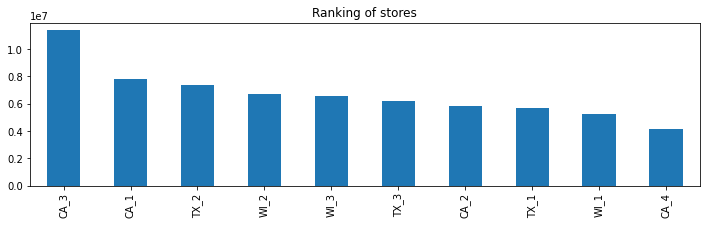
California has more sales throughout the years, this is also due to the fact that california has more stores. And generally there is an increasing trend in the number of sales. Gradually over the years, the Wisconsin store overtook Texas store in terms of sales.



*Figure 3.4: Total sales in different states*

Taking average sales will not give accurate pictures as at a certain point in time items didn’t exist. So if we divide by total number of items, the average sale calculation will be inaccurate.

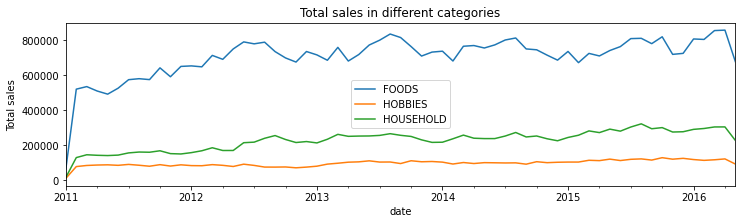
By looking at the stores themselves, CA\_3 and CA\_1 are the best performing stores among all while CA\_4 is the least performing one.



*Figure 3.5: Ranking of stores based on total sales*

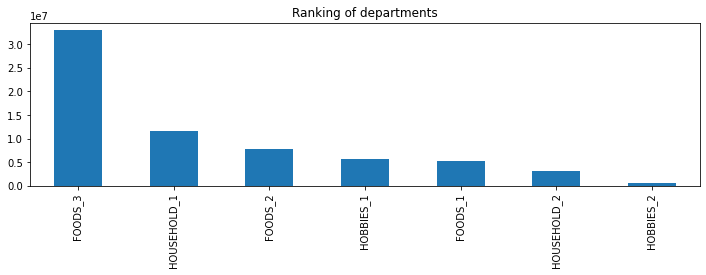
**Total monthly sales by Category**

Generally food items are sold most in terms of quantity and hobby items are the least sold ones. The sales of food and household items are increasing gradually over the years compared to hobby items.

****

*Figure 3.6: Total sales in different categories*

FOODS\_3 and HOUSEHOLD\_1 are the best performing sales items, while the least sold ones are HOUSEHOLD\_2 and HOBBIES\_2 items.

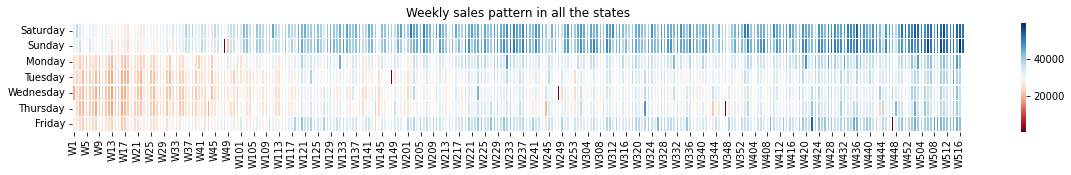
****

*Figure 3.7: Ranking of department based on total number of sales*

**4.5 Factors affecting Sales**

**4.1 Impact of weekdays**

Sales are common on weekends, the following heatmap shows how number of sales changed over since day 1 on a weekly basis.

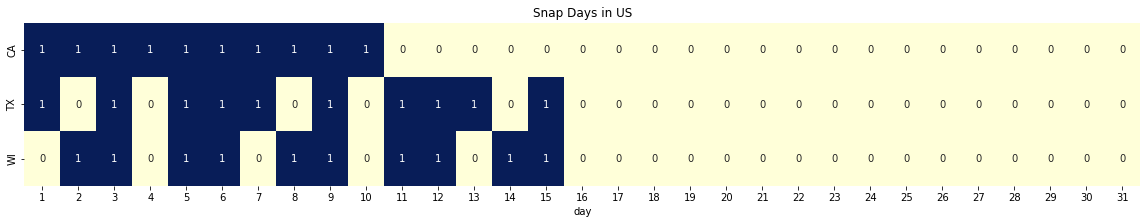
****

*Figure 4.1: Weekly change is sales*

We can see that the sales are high during saturday and sunday, then sales decline on monday, tuesday, wednesday, and thursday. The sales starts slightly increasing on friday again.

**4.2 Impact of SNAP days**

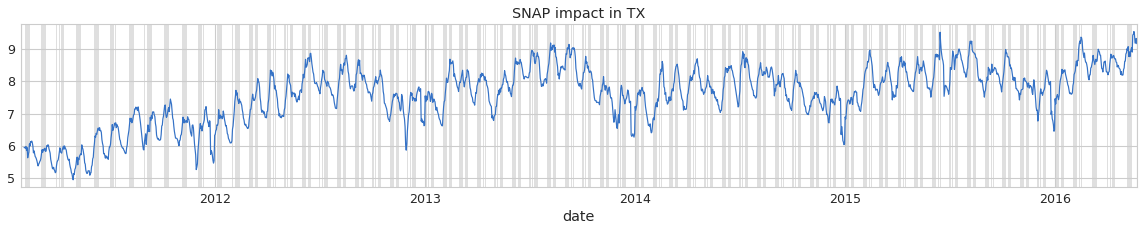
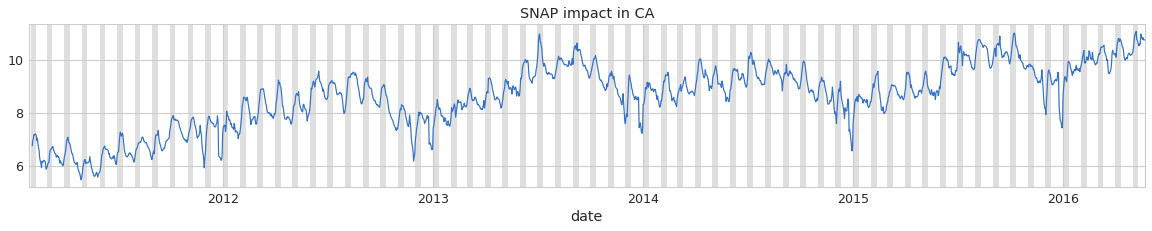
Supplemental Nutrition Assistance Program (SNAP) is a federal nutrition program provided by the United States of America. Each state has different snap days but according to the data, SNAP days happen in the first half of the month. During SNAP days, citizens can purchase food items with subsidy.

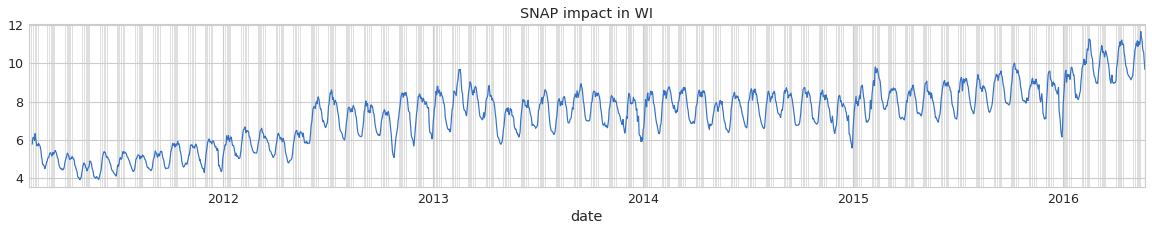


*Figure 4.2: Heatmap of SNAP days in different states (1 indicating SNAP day - blue)*

SNAP days are marked in blue for various states. In California SNAP days are continuous for the first 10 days of the month. In Texas and Wisconsin there are regular days between SNAP days and extend upto 15 th of the month.

There are only 10 SNAP days in a month, and on each of this day 1/10th of the population can avail benefits on food purchases.



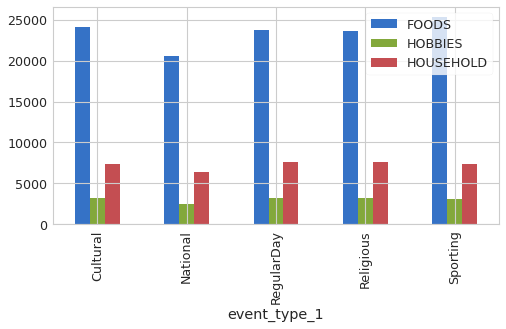


*Figure 4.3: Impact of SNAP days on sales*

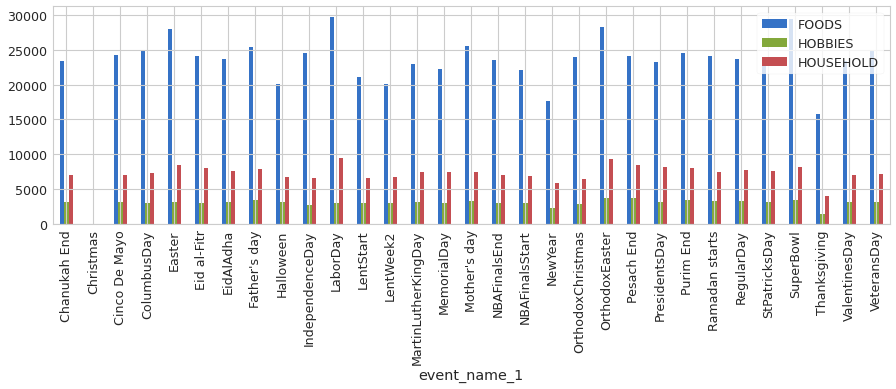
The gray markers indicate SNAP days in the states. We can observe that whenever there is a gray bar, the sales are increasing. So during SNAP days, we can expect higher sales.

**4.5 Impact of Calendar Events**

Calendar events are categorized into Cultural, National, Religious and Sporting event types. Sales can increase or decrease depending upon the holidays. The average sales during these event types are plotted below.

*Figure 4.4: Effect of types of events on sales*

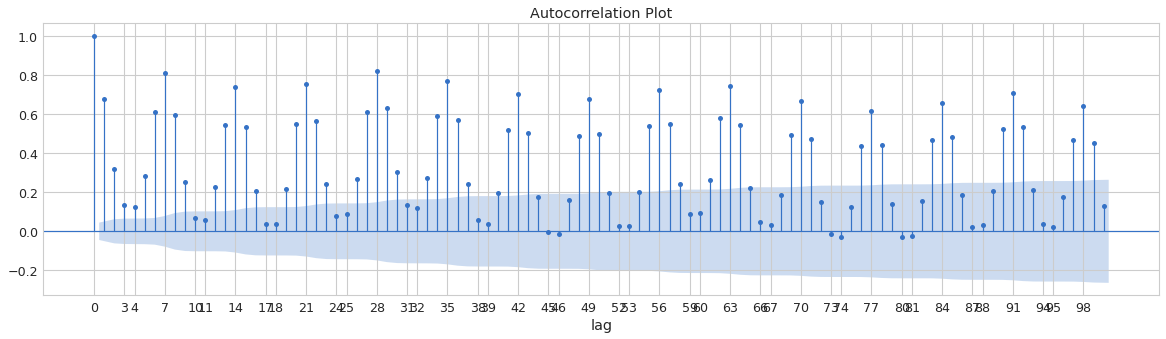
Sales are lower during national events and higher during sporting events. During Christmas the sales drop to zero, this could be due to store closure on Christmas. Also the average sales are very low during thanksgiving. Events such as LaborDay, Easter and Orthodox Easter have higher sales on average than regular days as shown in figure 4.5.

****

*Figure 4.5: Effect of calendar events on sales*

**4.6. Autocorrelation and partial autocorrelation plots**

From the Autocorrelation and Partial autocorrelation plots, we can see that there is strong weekly correlation. Certain days in the first 28 days, especially lags in multiple of 7 days have strong correlation. This insight can be used in and create Lag and Rolling features till the date where the correlation is strong.



*Figure 5.1: Autocorrelation plot*

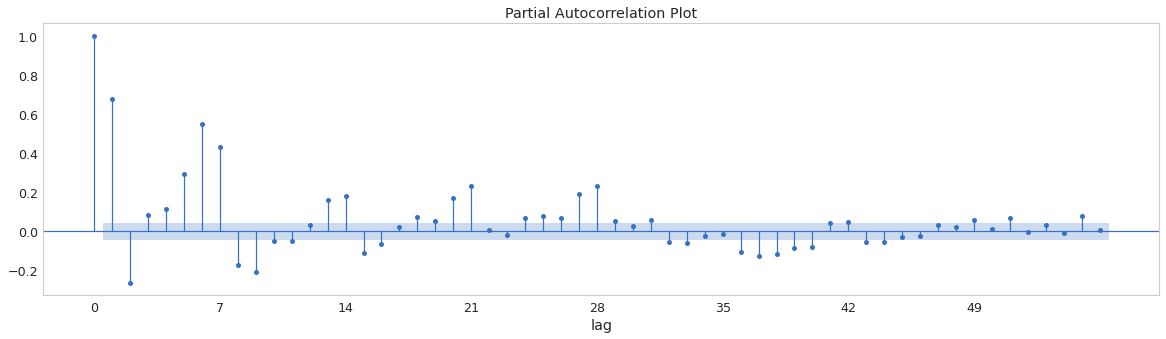


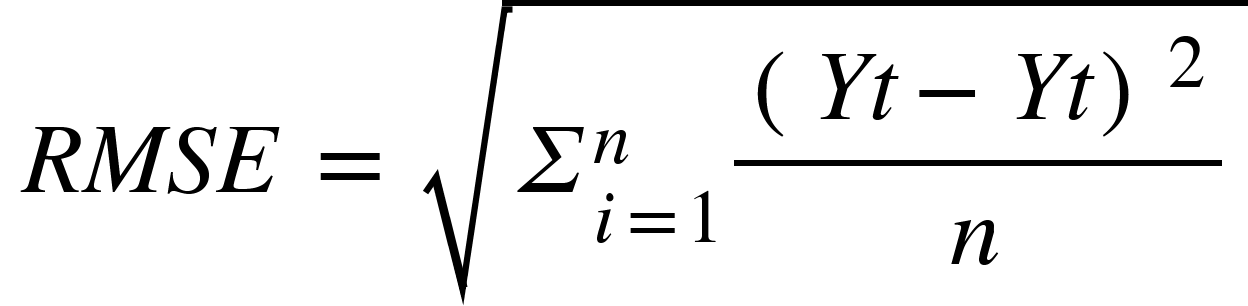
Figure 5.2: Partial Autocorrelation plot

**5. Metric**

Metrics of regression can be used to solve the problem. We will use Root Mean Squared Error (RMSE) to solve this problem.

**3.1 RMSE - Root Mean Squared Error**

Root Mean Squared Error is a commonly used error for a regression problem. But since we have different categories of products, for comparison it might not be the best option.



Where,

- Actual future value at time *t*

- Generated forecast

*n -* number of historical observations

**6. Modeling and Error Analysis**

**6.1 Baseline Models and Metrics**

**Mean model**

We are using root mean squared error as the evaluation metric. To compare models, we need a simple baseline model to understand whether our model is performing better or worse.

We will use a simple mean model for this purpose. Mean models of different window lengths were tried, however using the entire training data gave the best result of RMSE value 6.31.

Now our baseline is established, we will try different models and see how the rmse is improving.

**ETS Model**

ETS Model uses trend and seasonality components for forecasting. Using simple ETS model, the rmse score was 5.04.

**Exponential Smoothing**

Exponential Smoothing on the time series data gave an rmse of 4.87

Out of these simple to forecast models, we got the best RMSE score of 4.87. This will be used as the baseline score for our regression models.

The results of these models given below are done with a seasonal periodicity of 7 days. Changing this periodicity by a factor of 7 didn’t made significant change in the RMSE scores.

| Model | RMSE |
| --- | --- |
| Simple Mean Model | 6.31 |
| ETS Model | 5.04 |
| Exponential Smooting | 4.87 |

**6.2 Regression Models**

**Features**

We used lag and rolling features of the last 7, 14, 21 and 28 days with respective sliding windows for featurizing the models.

The following were the featurization used for the regression model.

1. wm\_yr\_wk
2. wday
3. snap\_CA
4. snap\_TX
5. snap\_WI
6. sell\_price
7. lag\_7
8. lag\_14
9. lag\_21
10. lag\_28
11. rolling\_mean\_7
12. rolling\_std\_7
13. rolling\_mean\_14
14. rolling\_std\_14
15. rolling\_mean\_21
16. rolling\_std\_21
17. rolling\_mean\_28
18. rolling\_std\_28
19. event\_name\_1
20. event\_type\_1
21. event\_name\_2
22. event\_type\_2
23. id
24. item\_id
25. dept\_id
26. cat\_id
27. store\_id
28. state\_id

**Approach**

We will use sliding window methods to forecast further into the future using the predicted values. So for simplicity and computational advantage, it was best to go with predicting the next day. Then experimenting with different algorithms.

Once we are confident in a set of algorithms, we can try different feature engineering to make the model better. Once a model is created well, we could use the sliding window to recursively predict the next 28 days of sales for each category of items.

**Cross Validation**

In this regression approach, we will have time series data till t days, then we need to forecast the number of sales for the next t+1 to t+28 days. So a realistic train - cross validation - test split would be based on the timestamp. We will use 1 to t dates of sales for training. t +1 to t + 28 for cross validation to tune our model. For evaluation we will use data from t+29 to t+56. In this way, using the dates till now, we can forecast for the next 28 days.

**Hyperparameter Tuning**

Hyperparameter tuning is done using the basics for loops using a custom random search. This will help to find the optimum hyperparameters faster than grid search.

Since tuning a model is expensive, it is done in two stage

1. Broad search - Where a broad range of values are taken randomly and searching for lower values of minima.
2. Narrow search - Near the minima, we will narrow the range of search so as to get a much more tuned result.

**6.3 Regression Algorithms**

**Linear Regression with L2 regularization**

This is mainly used as a baseline regression model to compare our models' performance.

Table: Hyperparameter Tuning Result - Linear Regression with L2 Regularization

| **L2 Regularization** | **Train Error** | **CV Error** |
| --- | --- | --- |
| 0.01 | 2.446404 | 2.178166 |
| 0.0001 | 2.447 | 2.178695 |
| 0.000001 | 2.446991 | 2.178708 |
| 0.00001 | 2.446991 | 2.17873 |
| 0.001 | 2.446901 | 2.178846 |
| 0.1 | 2.470704 | 2.206977 |
| 1 | 2.54564 | 2.287303 |
| 10 | 3.064857 | 2.801523 |
| 100 | 3.378298 | 3.123945 |
| 100 | 3.378298 | 3.123945 |
| 10000 | 3.401201 | 3.1441 |

**Gaussian Naive Bayes**

Gaussian Naive Bayes performed poorly on our regression data.

Table: Hyperparameter Tuning Result - Gaussian Naive Bayes

| **var\_smoothing** | **train\_error** | **cv\_error** |
| --- | --- | --- |
| 10 | 6.074611 | 3.249828 |
| 100 | 3.562972 | 3.398215 |
| 1000 | 3.556178 | 3.398215 |
| 10000 | 3.556178 | 3.398215 |
| 1 | 10.412797 | 6.447999 |
| 0.1 | 13.181332 | 8.780273 |
| 0.01 | 16.99767 | 12.024037 |
| 0.001 | 22.145534 | 18.799591 |
| 0.0001 | 29.829064 | 28.107998 |

**Support Vector Regressors**

Support Vector Regression with linear kernel was giving similar results to that of linear regression. So due to the computational intensity of working with millions of rows were not feasible, there was no significant reason to try out Support Vector Regressors. Due to the inherent nature of the dataset which itself is a tree, trying out tree based regressors and it’s ensembles made more sense.

**Tweedie Regression**

Some resources suggested tweedie as a good loss function to get much more accurate results. So TweedieRegression from sklearn was tried out. But the results weren’t promising.

Table: Hyperparameter Tuning Result - Tweedie Regression

| Alpha | Power | Train Error | CV Error |
| --- | --- | --- | --- |
| 1 | 1.1 | 12.950694 | 2.719709 |
| 10 | 1.1 | 10.60614 | 2.790567 |
| 10 | 1.3 | 30.479093 | 2.794545 |
| 10 | 1.4 | 51.387356 | 2.826686 |
| 100 | 1.1 | 3.303286 | 3.056747 |
| 100 | 1.3 | 3.325098 | 3.078801 |
| 100 | 1.4 | 3.332904 | 3.085991 |
| 0.1 | 1.1 | 9.038981 | 3.114711 |
| 1000 | 1.4 | 3.400491 | 3.1441 |
| 1000 | 1.1 | 3.400119 | 3.1441 |
| 1000 | 1.3 | 3.400394 | 3.1441 |
| 1 | 1.3 | 69.353221 | 3.164844 |
| 0.01 | 1.1 | 8.060455 | 3.486667 |
| 0.001 | 1.1 | 7.993603 | 3.529907 |
| 1 | 1.4 | 240.991861 | 4.23283 |
| 0.1 | 1.3 | 40.793371 | 4.678297 |
| 0.01 | 1.3 | 32.58404 | 5.801989 |
| 0.001 | 1.3 | 31.825274 | 5.934 |
| 0.1 | 1.4 | 131.197509 | 7.153914 |
| 0.01 | 1.4 | 98.407607 | 9.111949 |
| 0.001 | 1.4 | 95.192894 | 9.350677 |

**Decision Tree Regressors**

Out of the regression algorithms, decision trees worked better than any other base models. This could be due to the hierarchical nature of the problem itself.

Table: Hyperparameter Tuning Result - Decision Tree Regression

| Max depth | Min sample split | Min sample leaf | Train error | CV error |
| --- | --- | --- | --- | --- |
| 11 | 3 | 5 | 2.259054 | 2.192753 |
| 10 | 24 | 16 | 2.343793 | 2.195772 |
| 9 | 19 | 27 | 2.381579 | 2.197011 |
| 8 | 4 | 2 | 2.36153 | 2.199432 |
| 11 | 26 | 14 | 2.310163 | 2.203802 |
| 11 | 9 | 14 | 2.310163 | 2.203802 |
| 6 | 16 | 23 | 2.463273 | 2.231982 |
| 5 | 12 | 11 | 2.502923 | 2.255038 |
| 16 | 23 | 25 | 2.24775 | 2.271997 |
| 19 | 27 | 28 | 2.230523 | 2.274726 |
| 18 | 3 | 23 | 2.213961 | 2.296256 |
| 3 | 22 | 6 | 2.656797 | 2.406704 |
| 3 | 23 | 5 | 2.656797 | 2.406704 |
| 16 | 7 | 7 | 2.075503 | 2.454331 |
| 2 | 5 | 5 | 2.767814 | 2.490598 |
| 2 | 15 | 20 | 2.767814 | 2.490598 |
| 2 | 3 | 7 | 2.767814 | 2.490598 |
| 2 | 7 | 12 | 2.767814 | 2.490598 |
| 19 | 21 | 5 | 2.007605 | 2.502255 |
| 19 | 17 | 8 | 2.003085 | 2.523757 |

**Results**

Out of all the models, LightGBM performed best with RMSE of 2.096811. So we will improve the LightGBM model with additional featurizations.

Table: Tuned Results

| **Tuned Models** | **Best CV RMSE** |
| --- | --- |
| LightGBM regressor | 2.096811 |
| linear regression (L2 regularization) | 2.178166 |
| DecisionTree regressor | 2.192753 |
| Tweedie regressor | 2.719709 |
| Gaussian Naive Bayes | 3.249828 |
| Exponential Smoothing | 4.873292 |
| ETS Model | 5.039345 |
| Simple mean model | 6.315841 |

The evaluation result of the top 3 models from the experiment is given below.

Table - Evaluation Result

| Models | Train RMSE | CV RMSE | Test RMSE |
| --- | --- | --- | --- |
| Linear Regression with L2 Regularization | 2.429733 | 2.157479 | 2.26867 |
| DecisionTree regressor | 2.241267 | 2.172427 | 2.379114 |

The performance of the model can be improved by adding more days as well as more types of features.

**7. Advanced Modeling and Feature Engineering**

**7.1 Using Ensembles**

Using ensembles of decision trees to improve the model performance on the existing featurization. Ensembles takes a lot of computational time since they have to train a lot of base learners. Ensembles like XGBoost, GBDT, RandomForest and CAT Boost were not computing in a reasonable time. From the literatures, highly efficient Gradient Boosted Decision Trees like LightGBM are preferred for hierarchical data and they show best results compared to other Ensembles.

**LightGBM Regression**

LightGBM with “tweedie” loss showed the best results out of all the regressors. It gave the lowest RMSE. According to wikipedia, a Tweedie distribution is a special case of exponential dispersion models and are often used as distributions for generalized linear models.

One thing to note when tuning LightGBM for time series models is to have large num\_leaves parameter.

Table: Hyperparameter Tuning Result - Light GBM regression

| n\_estimators | max\_depth | learning\_rate | num\_leaves | reg\_lambda | train\_error | cv\_error |
| --- | --- | --- | --- | --- | --- | --- |
| 1000 | 10 | 0.01 | 100000 | 0.3 | 1.932674 | 2.096811 |
| 1000 | 5 | 0.01 | 100000 | 0.3 | 2.329517 | 2.132988 |
| 100 | 3 | 0.3 | 100000 | 0.5 | 2.360733 | 2.135046 |
| 100 | 12 | 0.1 | 100000 | 0.5 | 1.707952 | 2.135951 |
| 100 | 7 | 0.3 | 100000 | 0.9 | 2.06261 | 2.285918 |
| 1000 | 7 | 0.3 | 100000 | 0.3 | 1.498984 | 2.287496 |
| 1000 | 8 | 0.3 | 100000 | 0.5 | 1.299552 | 2.32315 |
| 100 | 11 | 0.01 | 100000 | 0.3 | 2.521621 | 2.380228 |
| 100 | 7 | 0.01 | 100000 | 0.5 | 2.642418 | 2.398913 |
| 1000 | 11 | 0.3 | 100000 | 0.9 | 0.737117 | 2.470182 |

**Evaluation Result of LightGBM Regressor:**

The LightGBM Regressor was evaluated on test data and these are the results.

Table: Evaluation result of LightGBM Model

| Train RMSE | 1.912675 |
| --- | --- |
| CV RMSE | 2.074397 |
| Test RMSE | 2.205613 |

**7.2 Advanced Feature Engineering**

In the previous models, we used lag and rolling models for a time period up to 28 days. In advanced feature engineering, we try to add and remove features based on various experiments.

Here is the new features.

**Lag Features**

Previously we use lag features for 7, 14, 21 and 28 days. Now we are increasing the number of lag features to that week plus the previous weekdays. Also the closest days to the same day last week was also added by offsetting by 8 days with a step count of 7 days. In this way, given today is sunday, we will also capture the effects of last sunday as well as saturday. This information could potentially improve the model performance.

| lag\_1  lag\_2  lag\_3  lag\_4  lag\_5  lag\_6  lag\_7 | lag\_14  lag\_21  lag\_28  lag\_35  lag\_42  lag\_49 | lag\_56  lag\_63  lag\_70  lag\_77  lag\_84 |
| --- | --- | --- |

**Rolling Features.**

More previous dates are added to the rolling feature. In addition to that features like rolling skew and kurtosis are also added to see if there is any performance improvement.

| rolling\_kurt\_14  rolling\_kurt\_21  rolling\_kurt\_28  rolling\_kurt\_35  rolling\_kurt\_42  rolling\_kurt\_49  rolling\_kurt\_56  rolling\_kurt\_63  rolling\_kurt\_7  rolling\_kurt\_70  rolling\_kurt\_77  rolling\_kurt\_84  rolling\_max\_14  rolling\_max\_21  rolling\_max\_28  rolling\_max\_35  rolling\_max\_42  rolling\_max\_49  rolling\_max\_56  rolling\_max\_63  rolling\_max\_7  rolling\_max\_70  rolling\_max\_77  rolling\_skew\_14  rolling\_skew\_21  rolling\_skew\_28  rolling\_skew\_35  rolling\_skew\_42  rolling\_skew\_49 | rolling\_skew\_56  rolling\_skew\_63  rolling\_skew\_7  rolling\_skew\_70  rolling\_skew\_77  rolling\_skew\_84  rolling\_std\_14  rolling\_std\_15  rolling\_std\_21  rolling\_std\_22  rolling\_std\_28  rolling\_std\_29  rolling\_std\_35  rolling\_std\_36  rolling\_std\_42  rolling\_std\_43  rolling\_std\_49  rolling\_std\_50  rolling\_std\_56  rolling\_std\_57  rolling\_std\_63  rolling\_std\_64  rolling\_std\_7  rolling\_std\_70  rolling\_std\_71  rolling\_std\_77  rolling\_std\_78  rolling\_std\_8  rolling\_std\_84 | rolling\_max\_84  rolling\_mean\_14  rolling\_mean\_15  rolling\_mean\_21  rolling\_mean\_22  rolling\_mean\_28  rolling\_mean\_29  rolling\_mean\_35  rolling\_mean\_36  rolling\_mean\_42  rolling\_mean\_43  rolling\_mean\_49  rolling\_mean\_50  rolling\_mean\_56  rolling\_mean\_57  rolling\_mean\_63  rolling\_mean\_64  rolling\_mean\_7  rolling\_mean\_70  rolling\_mean\_71  rolling\_mean\_77  rolling\_mean\_78  rolling\_mean\_8  rolling\_mean\_84 | rolling\_median\_14  rolling\_median\_15  rolling\_median\_21  rolling\_median\_22  rolling\_median\_28  rolling\_median\_29  rolling\_median\_35  rolling\_median\_36  rolling\_median\_42  rolling\_median\_43  rolling\_median\_49  rolling\_median\_50  rolling\_median\_56  rolling\_median\_57  rolling\_median\_63  rolling\_median\_64  rolling\_median\_7  rolling\_median\_70  rolling\_median\_71  rolling\_median\_77  rolling\_median\_78  rolling\_median\_8  rolling\_median\_84 |
| --- | --- | --- | --- |

**Difference Features**

Difference features were also added. The difference is sales of consecutive days as well as consecutive weekdays were added so that the model could learn from this new information.

| diff1\_1  diff1\_14  diff1\_2  diff1\_21  diff1\_28  diff1\_3  diff1\_35  diff1\_4  diff1\_42  diff1\_49  diff1\_5  diff1\_56 | diff1\_6  diff1\_63  diff1\_7  diff1\_70  diff1\_77  diff1\_84  diff7\_1  diff7\_14  diff7\_2  diff7\_21  diff7\_28  diff7\_3 | diff7\_35  Diff7\_4  diff7\_42  diff7\_49  diff7\_5  diff7\_56  diff7\_6  diff7\_63  diff7\_7  diff7\_70  diff7\_77  diff7\_84 |
| --- | --- | --- |

**Feature Importance**

Feature importance by DecisionTree Regressor were computed and these are the results. Once we do that we can remove features and try out model performance.

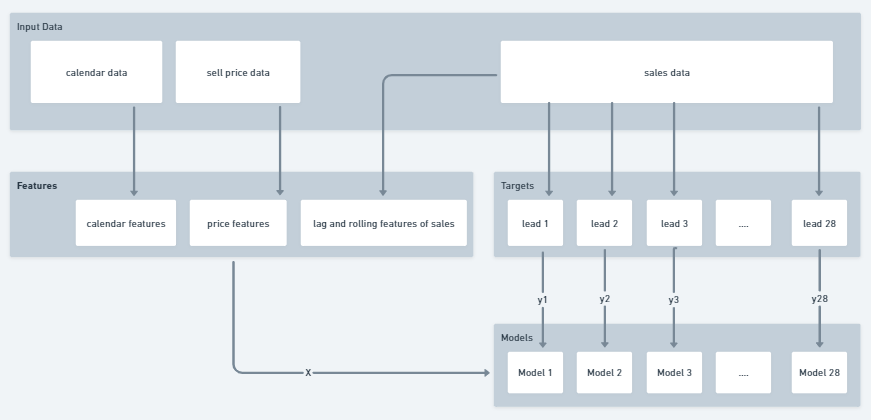
The top important features are:

| **Features** | **Feature Importance** |
| --- | --- |
| lag\_1 | 0.508546 |
| lag\_7 | 0.196271 |
| lag\_6 | 0.051359 |
| lag\_2 | 0.05006 |
| lag\_28 | 0.018872 |
| lag\_21 | 0.013458 |
| lag\_3 | 0.010191 |
| lag\_14 | 0.008389 |
| lag\_4 | 0.0069 |
| diff1\_7 | 0.00465 |
| lag\_5 | 0.004512 |
| rolling\_mean\_56 | 0.004155 |
| lag\_35 | 0.004014 |
| lag\_63 | 0.003297 |
| lag\_56 | 0.003115 |
| rolling\_mean\_28 | 0.003063 |
| rolling\_mean\_49 | 0.003032 |
| diff7\_21 | 0.002732 |
| wday | 0.002721 |
| diff1\_1 | 0.00269 |

**Multi Output Strategy**

To solve the business problem, we will extend this to multiple days. For that we will train different models for different days.

With some experimentation, it is found that lag and rolling features were best for 28 days prediction throughout. It is possible to use different featurization for different models, but it is a costly process to do so.



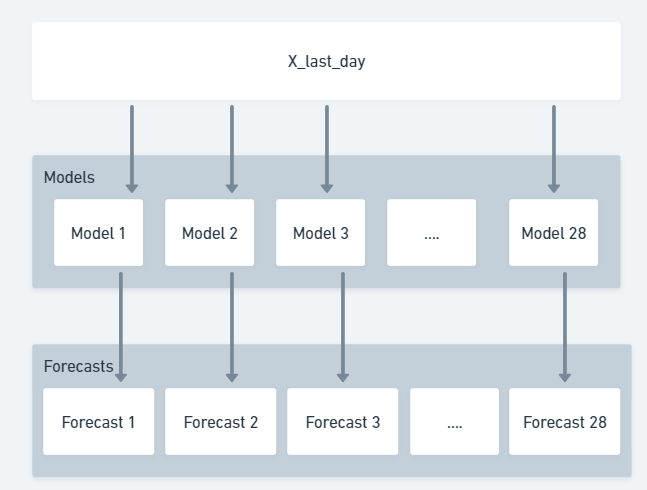
The Multi Output Strategy used here has the following features:

| item\_id  dept\_id  cat\_id  store\_id  state\_id  d  sales  wm\_yr\_wk  wday  month  year  event\_name\_1  event\_type\_1  event\_name\_2  event\_type\_2  snap\_CA  snap\_TX  snap\_WI  sell\_price  weekly\_price\_change | lag\_1  lag\_2  lag\_3  lag\_4  lag\_5  lag\_6  lag\_7  lag\_8  lag\_9  lag\_10  lag\_11  lag\_12  lag\_13  lag\_14  lag\_15  lag\_16  lag\_17  lag\_18  lag\_19  lag\_20 | lag\_21  Lag\_22  lag\_23  lag\_24  lag\_25  lag\_26  lag\_27  lag\_28  lag\_29  rolling\_mean\_7  rolling\_mean\_14  rolling\_mean\_21  rolling\_mean\_28  rolling\_mean\_30  rolling\_std\_7  rolling\_std\_14  rolling\_std\_21  rolling\_std\_28  rolling\_std\_30 |
| --- | --- | --- |

Targets were created for the next 28 days, and individual models were trained for each day.

Once we pass the last day features, these 28 models will predict the demand for the next 28 days.

Training Pipeline for 28 days models. Once we train these 28 days models. We give a single row of feature X of the last day in the dataset.



The X\_last\_day input is fed to all the 28 models which are trained to predict for forecast 1, forecast 2, upto forecast 28.

**8. DISCUSSION OF RESULTS**

We have tried various regression models from simple forecasters to advanced regression algorithms. The performance of the Tree based ensembles were the best out of all.

| Models | Forecast 1 - RMSE |
| --- | --- |
| Exponential Smoothing | 4.873292 |
| ETS Model | 5.039345 |
| Simple mean model | 6.315841 |
| linear regression (L2 regularization) | 2.178166 |
| DecisionTree regressor | 2.192753 |
| Tweedie regressor | 2.719709 |
| Gaussian Naive Bayes | 3.249828 |
| LightGBM | 2.074397 |

Since Forecast 1 RMSE of lightgbm was the highest, it was used for making the 28 models for 28 days. It is expected to see a decline in performance as the forecasting horizon increases.

The Mean RMSE of the LightGBM model for the 28 days prediction is 2.35.

**9. CONCLUSION**

We tried various methods for predicting the sales data from simple statistical models to different regression algorithms. Univariate and statistical methods were unable to capture information between the stores and hierarchy that exists within the data. So regression methods are apt since we can use a family of machine learning algorithms to solve the problem. Feature engineering and tuning the models played an important role. Tree based ensemble regressors proved to work better with time series data than traditional methods, however lightgbm model performed better than other ensembles, and is very fast to train. Forecasting for walmart sales data is completed. We can improve the model with a few more techniques. Separating the sales data based on the volume of sales and training separate models for that could improve the forecasting accuracy for wide range of products.

**10. REFERENCES:**

1. Kaggle Fourth place solution: <https://www.kaggle.com/c/m5-forecasting-accuracy/discussion/163216>
2. Kaggle Fifth place solution: <https://www.kaggle.com/c/m5-forecasting-accuracy/discussion/163916>
3. First Place Solution: <https://www.kaggle.com/c/m5-forecasting-accuracy/discussion/163684>
4. Third place solution: <https://www.kaggle.com/c/m5-forecasting-accuracy/discussion/164374>
5. Top 3% simple solution : <https://www.kaggle.com/c/m5-forecasting-accuracy/discussion/163154>
6. M5 Competitors Guide: <https://mofc.unic.ac.cy/wp-content/uploads/2020/03/M5-Competitors-Guide-Final-10-March-2020.docx>
7. M5 Forecasting Accuracy - Kaggle: <https://www.kaggle.com/competitions/m5-forecasting-accuracy/overview>
8. Michel Baudin’s Blog: <https://michelbaudin.com/2021/07/16/evaluating-sales-forecasts/>
9. Researchgate: <https://www.researchgate.net/publication/351458556_The_M5_competition_Background_organization_and_implementation>
10. 5 Machine Learning Techniques for Sales Forecasting - Molly Liebeskind: <https://towardsdatascience.com/5-machine-learning-techniques-for-sales-forecasting-598e4984b109>
11. Demand forecast with different data science approaches - Andrii Shchur : <https://towardsdatascience.com/demand-forecast-with-different-data-science-approaches-ba3703a0afb6>
12. Tweedie Distribution Wikipedia: <https://en.wikipedia.org/wiki/Tweedie_distribution>
13. M4 Competition forecasting methods: <https://www.sciencedirect.com/science/article/pii/S0169207019301128>
14. Multi Output Regression Forecast - Jason Brownlee : <https://machinelearningmastery.com/multi-output-regression-models-with-python/>

**11. Productionisation and Deployment**

We will deploy our machine learning model into production along with the pipeline to transform the data into required format. Here, the user has to upload data of sales, sell prices and calendar in csv format, the model will predict for the next 28 days.

Our best performing model lightgbm is used for the production. Each of the 28 models only have 200 base learners, only hyperparameters such as learning rate, regularization, max leaves etc.. are changing.

**App Development**

Streamlit is used for developing the front end and deployment of the app. The end user has to upload the csv file of sales data. The featurized data from the calendar is applied to the model.

**Serializing the model**

We use pickle library to serialize the trained model and dump it to the lgbm.pkl file. For deployment in streamlit we need the following files.

1. F1.pkl, F2.pkl,.... F28.pkl - This pickle file contains the trained LightGBM Regressor from lightgbm library in F<forecast\_days>.pkl format.
2. calendar\_label\_encoders.pkl - This file contains the dictionary of LabelEncoder() objects from scikit learn. This is used to transform the categorical columns.
3. sales\_label\_encoders.pkl - This file contains the dictionary of LabelEncoder() object from scikit learn. This is used to transform the categorical columns.
4. standard\_scaler.pkl - This file contains the standardization used in training. This is used to transform the incoming data to proper scale for the ML model.
5. requirements.txt - This is a text document of versions of python packages like pandas, lightgbm etc.
6. app.py - This is our streamlit app backend code. This python file loads the pickle files and generates forecasts.
7. Dockerfile - This file contains the code for Dokerising the application in Cloud.

We can run the streamlit app during development using the code:

*streamlit run app.py*

**Deployment on Google Cloud Run**

Deploying the model directly on streamlit was not possible because of the resource requirement. The deployment was done in Google Compute Platform using Google Cloud Run, it helps us to run containerised applications.

**Dockerfile**

Inorder to do this, we need a docker file that specifies the base image, runs necessary commands, and expose ports so that anyone can access it.

FROM python:3

WORKDIR /app

COPY . .

RUN pip3 install --no-cache-dir -r /app/requirements.txt

EXPOSE 8080

ENTRYPOINT ["streamlit", "run", "app.py", "--server.port=8080", "--server.enableCORS=false"]

Once the docker file is made, install Google Cloud SDK and run the following commands.

gcloud config set project <project\_name>

This will set our project as the default project.

gcloud builds submit --tag gcr.io/<project\_name>/walmart\_sales\_pred --project=<project\_name>

This will upload our files from the directory and builds a container using the docker file we created.

gcloud run deploy --image gcr.io/<project\_name>/walmart\_sales\_pred --platform managed --project=<project\_name>

Now our app is successfully hosted to us. To make it available to all the users, instead we need to specify “--allow-unauthenticated” to the google cloud shell.

gcloud run deploy --image gcr.io/<project\_name>/walmart\_sales\_pred --platform managed --project=<project\_name> --allow-unauthenticated

Deployment can also be done with a streamlit share feature. This is done as follows.

1. Upload the app.py, requirements.txt and pickle files to a new github repository.
2. Create a streamlit account and click on new app.
3. You can mention the python version and github repository.
4. In a while the app will be hosted.

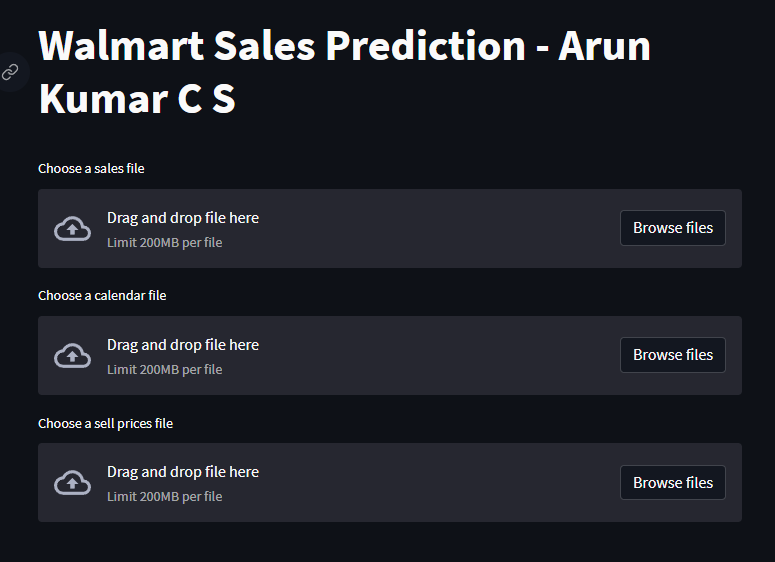
Streamlit installs all the necessary packages from the requirement.txt file. Once it is done, it will open app.py to deploy our model.

**Demo**

Hosted Application can be found at:

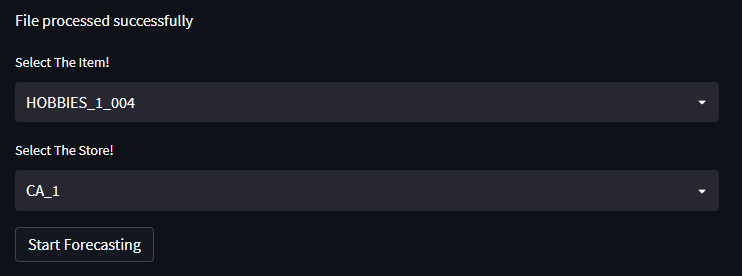
<https://arun-kumar-c-s-walmart-sales-prediction-app-tjpfki.streamlitapp.com/>

<https://walmartsalespred-fgrpzob4ia-wn.a.run.app/>

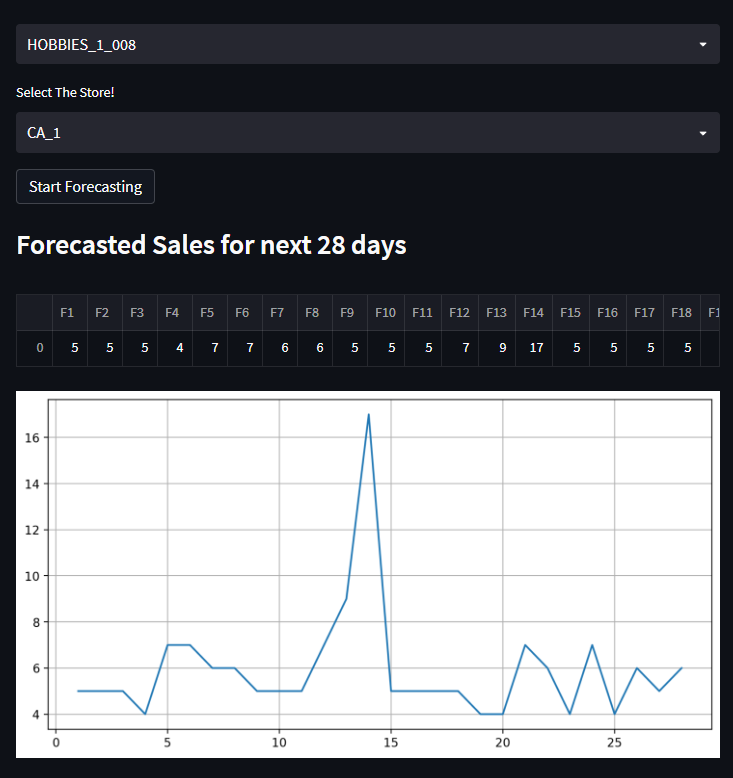


On this page we can upload 3 files. The sales information of at least 60 days, a calendar file containing events for the next 28 days and sell price information.

The application reads the items in the data and shows which item to forecast. You can select the item and store from the drop down option.



Click Start Forecasting to generate the forecast for the next 28 days.



You can see F1 to F28 which are the forecasted value for the ‘item\_id’ from ‘store\_id’. A graph of the sales is also shown below for better visualization of the forecast.