A PROJECT REPORT ON
Forecasting the unit sales of Walmart retail goods
[M5 Forecasting - Accuracy]

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

Under

Applied Roots with the University of Hyderabad

Project submitted by

Kishankumar Mistri

Under the Guidance of

Mentor: Saugata Paul

Approved by: Saugata Paul (Mentor)

University of Hyderabad

DECLARATION OF AUTHORSHIP

We hereby declare that this thesis titled “Walmart Unit Sales Forecasting” and the work presented by the undersigned candidate is part of a 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: Kishankumar Mistri

Thesis Title: Walmart Unit Sales Forecasting

CERTIFICATE OF RECOMMENDATION

We hereby recommend that the thesis entitled “Walmart Unit Sales Forecasting” prepared under my supervision and guidance by Kishankumar Mistri be accepted in fulfilment of the requirement for awarding the degree of Diploma in AI & ML Under applied roots with the 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, Kishankumar Mistri, a student of applied roots, am extremely grateful to my mentors for the confidence bestowed on me and for entrusting my project entitled "Walmart Unit Sales Forecasting" with special reference.

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

Name: Kishankumar Mistri

Contents

Abstract:

The sales forecasting of a particular product indicates how much quantity of a specific product is most likely to be sold in a specified future period (a day, a week, a month, a quarter, etc) in a specified market at a specified price. The ability to forecast accurately can enable business leaders to monitor and facilitate strategic planning to run day-to-day operations efficiently. With the rise in competition, it is not only advantageous but also a necessity to forecast sales to replenish resources and handle workloads efficiently meeting supply and demand. As it has a direct impact on the monetary & marketing resources allocation of every business, it is important to accurately forecast sales within the business's short-term and long-term purposes.[1] Sales forecasting is a complex process due to it being dependent on internal and external factors like shifts in trends on the factors which affect the sales like product changes, economic conditions, advertising, online promotions, season changes, market changes, etc; which impact to forecast of sales. Traditional forecasting methods include intuition-based forecasting, quantitative methods and qualitative statistical modelling which are inefficient to a certain extent due to a variety of factors. With the rise in data collection, we can put this data to forecast sales numbers more accurately by applying machine learning techniques. Machine learning techniques are used in every field from small to large-scale businesses.

1. Literature Review

1.1 Business problem Introduction:

Sales forecasting is one of the most important cornerstones of the decision-making process for every business. It is more important for large conglomerates, especially as their margin increase by a single decimal, which will lead to a massive profit due to the scale it has been implemented and the otherwise is also true. We will be seeing the data for one of such retail conglomerates Walmart with this case study.

The primary objective of this study is to forecast/predict sales accurately for the item-unit deals for Walmart based on store sales data provided for three US states (California, Texas, and Wisconsin). In arrange to perform expectations on different items that are sold in Walmart, machine learning methods have been actualized at the side the conventional strategies to extend the exactness. Three diverse machine learning models are utilized to figure out every day deals for taking after **28 days**. The primary problem at hand is to predict the price from historical data.

1.2 Challenges:

As sales forecasting is traditionally done by a gut feeling, the adoption of technology was a major challenge for enterprises. However, day by day it is making

progress. However, even today it is managed by sales tools that may not be working to their full potential. Why? Because of the dynamic nature of the retail and lack of visibility on the cycle as a whole. There are factors like changes in product mix, price & quantity changes made by competitors, and demographic changes around the target market. Seasonality and Trends-based changes. Slight changes can lead to many SKUs and ultimately leads to not appropriate data for ideal forecasting situations. The changes in the sub-category of points above make each forecasting makes difficult. However, as time goes by we tend to understand the situation and be smart about the data collection which matters. Also, the adoption of machine learning algorithms helps us with many complex relationships and forecasts accurately. For example, weather changes are an external feature affecting sales, which is one of the goals of this study.
Due to such an undefined number of factors, we will have to incorporate all factors (which may seem independent) data that need to be used to project/forecast sales more accurately.

Note: The basic assumption made by forecasting unit sales is the environment it was conducted previously will be repeated or will be improved based on suggested changes to meet the target.

1.3. Earlier methods:

From the traditional method standpoint, we have qualitative, quantitative & casual models. You can find more details in the Harvard Business Review post.[5] However, building upon these fundamentals time series approach looks for seasonal fluctuations, behavioural patterns and cyclic behaviour, we can’t forecast it accurately.

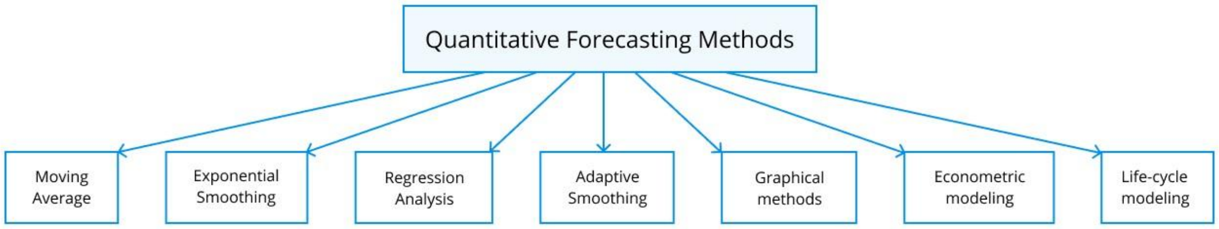


Fig: 1: Few quantitative forecasting methods image by Mobidev[7]

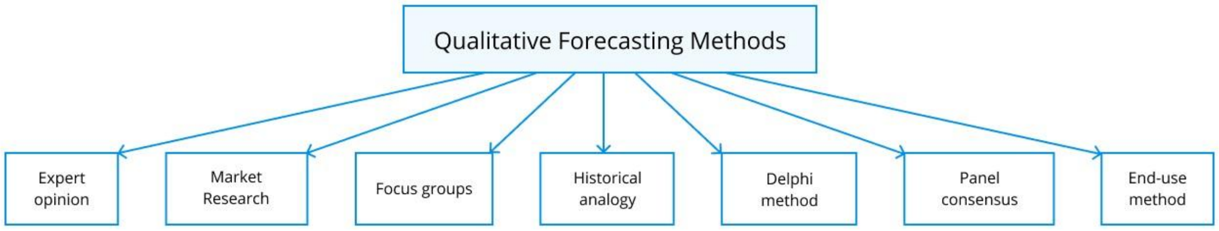


Fig: 2: Few qualitative forecasting methods image by Mobidev[7]

Most of the forecasting problems are solved using one of the below or a set of below methods nowadays.

- Regressive Models:
    - Linear Regression
    - Random Forest
    - XGBoost
- Long Short-Term Memory (LSTM): This neural network model works well on sequential data predictions that we will be using for rolling forecasting. Chen, I. and Lu, C. J. (2016).
- ARIMA/SARIMA: This model mainly works to get the autocorrelation of the time series data for the short-term (generally <1 yrs). The SARIMA stands for Seasonal ARIMA model, which includes the data which backshifts seasonal data. We will provide seasonal order to model for making adjustments around predictions. ARIMA models have been proven to be more accurate for predicting monthly and weekly data as it is making fewer errors in the majority of cases.[8]
- Finally, ensembling different models to get a collective representation of prediction/forecasting.
- In the Kaggle competition, the top position users used the LightGBM model (With and without Tweedie objective function), a recursive method model which works in one-day prediction at a time and predicts the next day using a predicted value for the required span.

1.4. Dataset

Challenges in obtaining dataset:

The dataset analyzed for the problem is provided by Walmart on the Kaggle stage here. As the dataset is present on Kaggle. It is clean and we do not need to use other means of getting this dataset. There are no challenges at this point. The dataset also contains weather data on Kaggle. In general, we can also get those weather features from government climate data sources.[4] However, it will take additional time and methods to get the data in the desired state.

Dataset properties and challenges:

The M5 dataset provided in the Kagle is generously made available by **Walmart** and involves the unit sales of various products sold in the USA, organized in the form of **grouped time series**. More specifically, the dataset involves

- the unit sales of **3,049 products**,

- classified into **3 product categories** (Hobbies, Foods, and Household) and **7 product departments**,
- Located in **three states** (CA, TX, and WI).
- Sold across **ten stores**
    - CA_1, CA_2, CA_3, CA_4,
    - TX_1,TX_2, TX_3
    - W1_1, WI_2, WI_3

Aggregation levels of the dataset provided are in the below grouped/aggregated format:

Level id	Aggregation Level	No of series
1	Unit sales of all products, aggregated for all stores/states	1
2	Unit sales of all products, aggregated for each State	3
3	Unit sales of all products, aggregated for each store	10
4	Unit sales of all products, aggregated for each category	3
5	Unit sales of all products, aggregated for each department	7
6	Unit sales of all products, aggregated for each State and category	9
7	Unit sales of all products, aggregated for each State and department	21
8	Unit sales of all products, aggregated for each store and category	30
9	Unit sales of all products, aggregated for each store and department	70
10	Unit sales of product *x*, aggregated for all stores/states	3,049
11	Unit sales of product *x*, aggregated for each State	9,147
12	Unit sales of product *x*, aggregated for each store	30,490
Total		**42,840**

In this respect, product-store unit sales can be mapped across either product categories or geographical regions, below graph, shows the same relationship. [3]

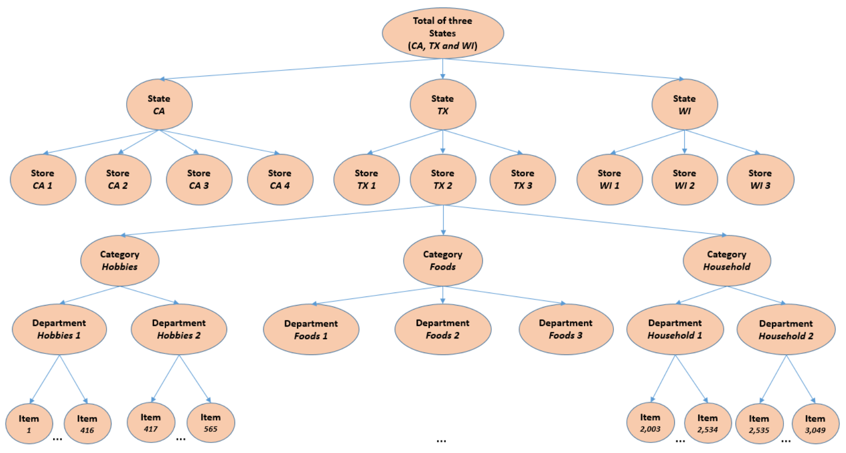


Fig 3: An overview of how the M5 series is organized.

Files level details in brief:

- “sell_prices.csv”: Contains information about the price of the products sold per store and date, which could indicate the promotion activity. The selling price of each product is marked weekly.
    - store_id: ID of the store
    - item_id: Product ID
    - wm_yr_wk: Week ID of specific year
    - sell_price: The price point 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 weekly, they may change over time (both training and test set).]

- “*calendar.csv”:* Contains information about the dates on which the products are sold and special events information, like national vacations and religious memorial days.
    - *date*: The date is in the format of “YYYY-MM-DD”.
    - *wm_yr_wk*: WEEK id of a particular year.
    - *weekday*: Extracted weekday type of the day (Saturday, Sunday, …, Friday).
    - *wday*: The weekday id, starting from Saturday.
    - *month*: The month of the date.
    - *year*: The year of the date.
    - *event_name_1*: If the date includes an event, the name of this event.
    - *event_type_1*: If the date includes an event, the type of this event.
    - *event_name_2*: If the date includes a second event, the name of this event.
    - *event_type_2*: If the date includes a second event, the type of this event.
    - *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.

Tools (Pandas, SQL, Spark, etc) that you will use to process this data.

- Python 3.7+ & JupyterLab/Jupyter Notebook
- The implementation-specific library used: Sklearn, Keras, TensorFlow

1.5. Loss functions and metrics:

The metrics for assessing the regression models are mean squared error (MSE), Root mean Square Error(RMSE) which is an extension of MSE, Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), pinball loss and many more including custom metrics [2]. The result from RMSE seems back to the expressed theory and helps the trade examiner to progress in arranging diverse perspectives of the trade level, for the occurrence of stock dispersion, dispersion administration, stock capacity arrangements, item fulfilment, etc. We generally prefer to use MAPE if the data is overall continuous. However, from the provided data some of the products are not been sold for a few weeks so it would be zero for those. Because of that reason, the overall MAPE value is affected so we can't choose it for our use case.
However, the sales value are including lots of zero as a sales, which will bias to making the overall value of vanilla RMSE near zero value. To mitigate the effect of the scale issues of RMSE, we will use the Root Mean Squared Scaled Error (RMSSE) referred to in *"Another look at measures of forecast Accurately by Hyndman and Koehler"* (2006)[6].

The measure penalizes positive and negative forecast errors, as well as large and small forecasts, equally, thus being symmetric. This can be used to find errors for a single product, in that case, if you want to use it for multiple products collectively then Weighting needs to be done. Why? As weight if we consider margin then because sales forecasting on the different products has different margins so directly averaging out will mislead the judgement. We have a small number of transactions resulting in high sales due to its price and for the same sales amount of another product will need to have a larger quantity to be sold. So to balance it out WRMSSE is used. **The lower WRMSSE score the better.[3]**

$$Numerator\ (L) = \sqrt{(}\frac{1}{h} * \sum_{i=1}^{h}(y_i - \hat{y}_1)^2)$$

h = Number of test points

y_i = Actual Values

$\hat{y}_h$ = Predicted Y values

$$Denominator\ (S) = \sqrt{(}\frac{1}{n-1} * \sum_{i=2}^{n}(y_i - y_{i-1})^2)$$

y_i = Actual Values

n = Number of training point

$$RMSSE = \frac{L}{S}$$

$$WRMSSE\ of\ a\ product\ family\ U(P) \\ = \sum_{p \in P} weight_p * RMSSE(p)$$

Where weight p is given time-series weight. Note that the weight of each series will be computed based on the last 28 observations of the training sample of the dataset(sum of units sold multiplied by their respective price).
Note that this metric is already provided for the competition to measure the accuracy of the outcome expected.

When does this metric fail?:

When the data is having more distortion/noise/outliers as it is having averaging equations.

Code:

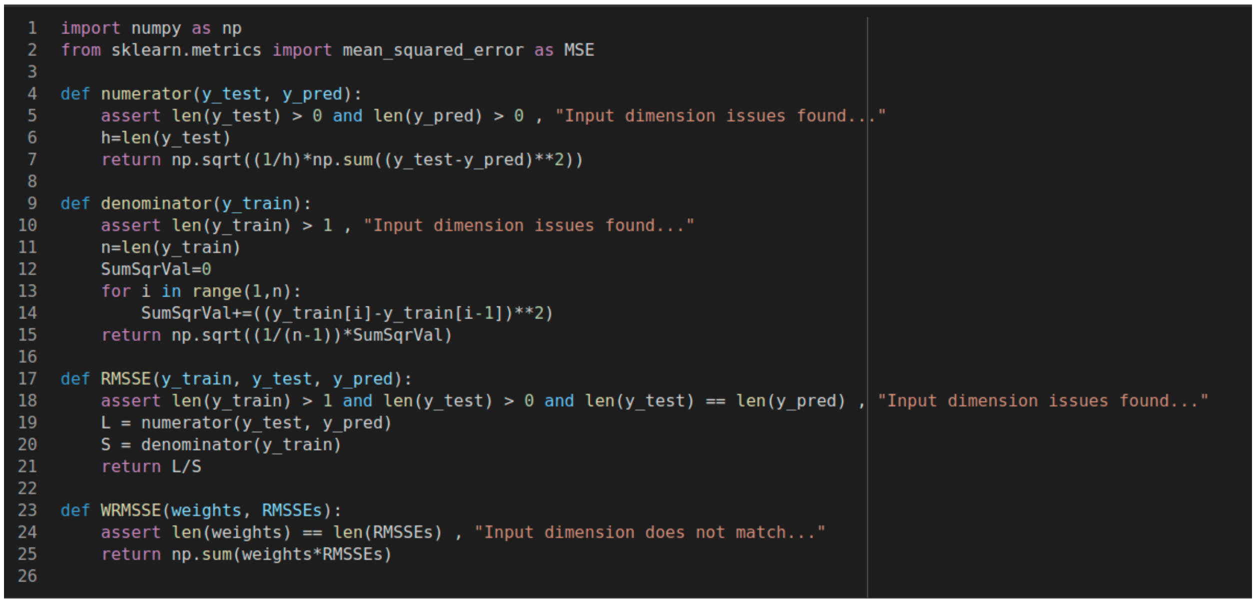
```
import numpy as np
from sklearn.metrics import mean_squared_error as MSE

def numerator(y_test, y_pred):
  assert len(y_test) > 0 and len(y_pred) > 0 , "Input dimension issues found..."
  h=len(y_test)
  return np.sqrt((1/h)*np.sum((y_test-y_pred)**2))

def denominator(y_train):
  assert len(y_train) > 1 , "Input dimension issues found..."
  n=len(y_train)
  SumSqrVal=0
  for i in range(1,n):
     SumSqrVal+=((y_train[i]-y_train[i-1])**2)
  return np.sqrt((1/(n-1))*SumSqrVal)

def RMSSE(y_train, y_test, y_pred):
  assert len(y_train) > 1 and len(y_test) > 0 and len(y_test) == len(y_pred) , "Input dimension
issues found..."
  L = numerator(y_test, y_pred)
  S = denominator(y_train)
  return L/S

def WRMSSE(weights, RMSSEs):
  assert len(weights) == len(RMSSEs) , "Input dimension does not match..."
  return np.sum(weights*RMSSEs)
```



```python
import numpy as np
from sklearn.metrics import mean_squared_error as MSE

def numerator(y_test, y_pred):
    assert len(y_test) > 0 and len(y_pred) > 0 , "Input dimension issues found..."
    h=len(y_test)
    return np.sqrt((1/h)*np.sum((y_test-y_pred)**2))

def denominator(y_train):
    assert len(y_train) > 1 , "Input dimension issues found..."
    n=len(y_train)
    SumSqrVal=0
    for i in range(1,n):
        SumSqrVal+=((y_train[i]-y_train[i-1])**2)
    return np.sqrt((1/(n-1))*SumSqrVal)

def RMSSE(y_train, y_test, y_pred):
    assert len(y_train) > 1 and len(y_test) > 0 and len(y_test) == len(y_pred) , "Input dimension issues found..."
    L = numerator(y_test, y_pred)
    S = denominator(y_train)
    return L/S

def WRMSSE(weights, RMSSEs):
    assert len(weights) == len(RMSSEs) , "Input dimension does not match..."
    return np.sum(weights*RMSSEs)
```

Alternative metric: Pinball Loss & Scaled Pinball Loss
When we want to say this is our predicted value at this time, we won't be having 100% certainty so in that case, we can have the confidence interval that we are 90% sure that we will have a value with the predicted value +- error we found.

Note: For experiment phases, we will use RMSE only as it is quickly calculated and reduces the workload. However, the results will also be uploaded to Kaggle to get WRMSSE results.

1.6. Real-world constraints and expectations

We have certain assumptions about the problem in place as well like:

- Our model is based on the assumption that the product sold by Walmart has patterns along with its timeline.
- Comparing stores and products is not optimal as there are many factors affecting sales.
- We do not have additional information about why the sales were down as it might be the case where the product is out of stock and this is very important as demand is there still due to unavailability it is impacting/suggesting the opposite that the product is not being sold on an average.
- Given the dataset, the sales of products might be impacted by the trends which create an anomaly “once in the blue moon situation”. But due to hashing out the product, we can’t be sure of it.

- The demand and supply of the products are also the following patterns that we'll be discovering with time.
- As discussed there are many factors affecting the sales, we'll be keeping in mind the bar/definition to be 65%. Anything above that should be considered an initial success. We will also consider Kaggle's scores as a reference. [10]

Latency requirements:

- As the forecasting is based on a 28-day rolling window, it will be modelled once a day. Once train models are in place the inferences will be made with the time complexity under 1-2ms.

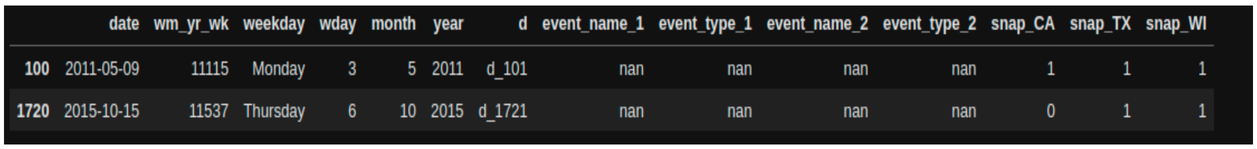
2. Exploratory Data Analysis

2.1 Dataset & Input-Output variable analysis

2.1.1 Dataset file Level analysis

Calender data:

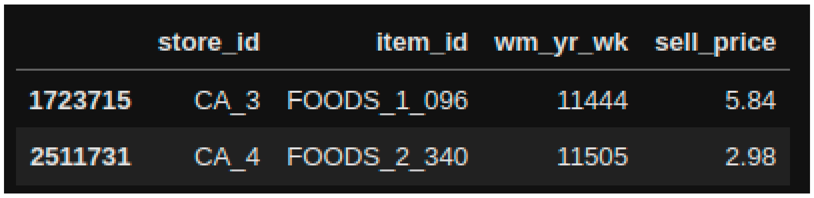
- We have 1969 rows which account for ~5.4 years of data (considering all unique dates) starting from Jan 2011 to June 2016
- We have features like a weekday and segregated date features. Apart from that if there were any special/specific day during the given date provided by columns with "event_type_" prefix
- Additional features like SNAP available at state with 0 being false and 1 being True.



	date	wm_yr_wk	weekday	wday	month	year	d	event_name_1	event_type_1	event_name_2	event_type_2	snap_CA	snap_TX	snap_WI
100	2011-05-09	11115	Monday	3	5	2011	d_101	nan	nan	nan	nan	1	1	1
1720	2015-10-15	11537	Thursday	6	10	2015	d_1721	nan	nan	nan	nan	0	1	1

Sales Price data

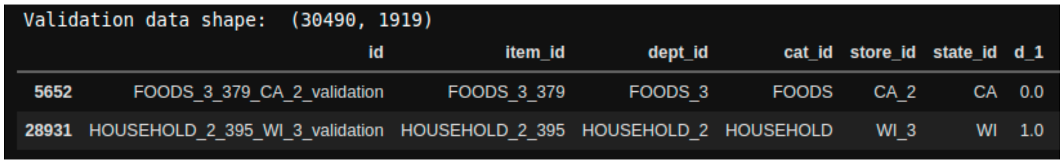
- In this data frame, we can see that we are having prices of each item at the store varying at a given weekid.
- We have 'store_id' & 'item_id', which can be used to merge the data with the calendar.
- Prices of the product range from 10 cents to $ 107.



	store_id	item_id	wm_yr_wk	sell_price
1723715	CA_3	FOODS_1_096	11444	5.84
2511731	CA_4	FOODS_2_340	11505	2.98

Sales Validation Data:

- Columns in validation data have "id" which is combination of CategoryID, DepartmentID, storeID & "_validation" as prefix.



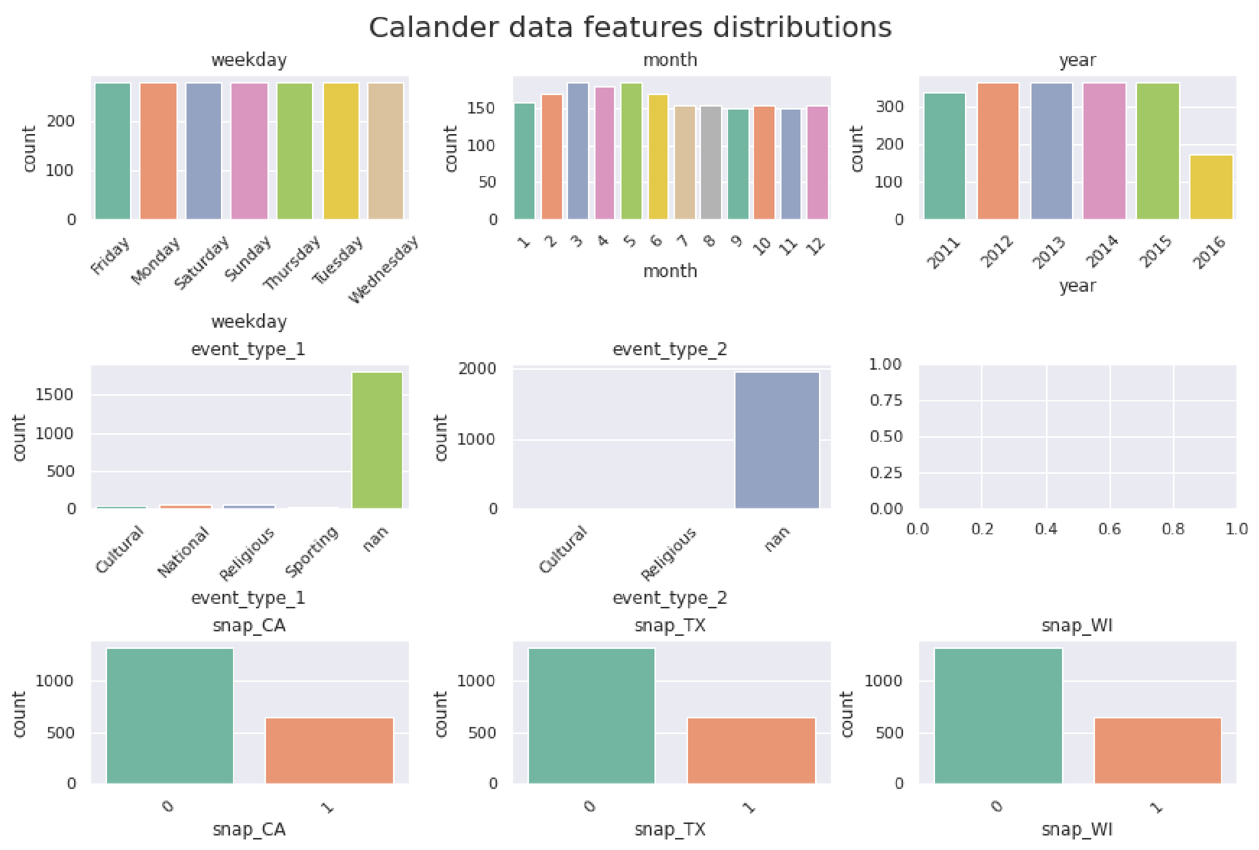
```
Validation data shape:  (30490, 1919)
```

	id	item_id	dept_id	cat_id	store_id	state_id	d_1
5652	FOODS_3_379_CA_2_validation	FOODS_3_379	FOODS_3	FOODS	CA_2	CA	0.0
28931	HOUSEHOLD_2_395_WI_3_validation	HOUSEHOLD_2_395	HOUSEHOLD_2	HOUSEHOLD	WI_3	WI	1.0

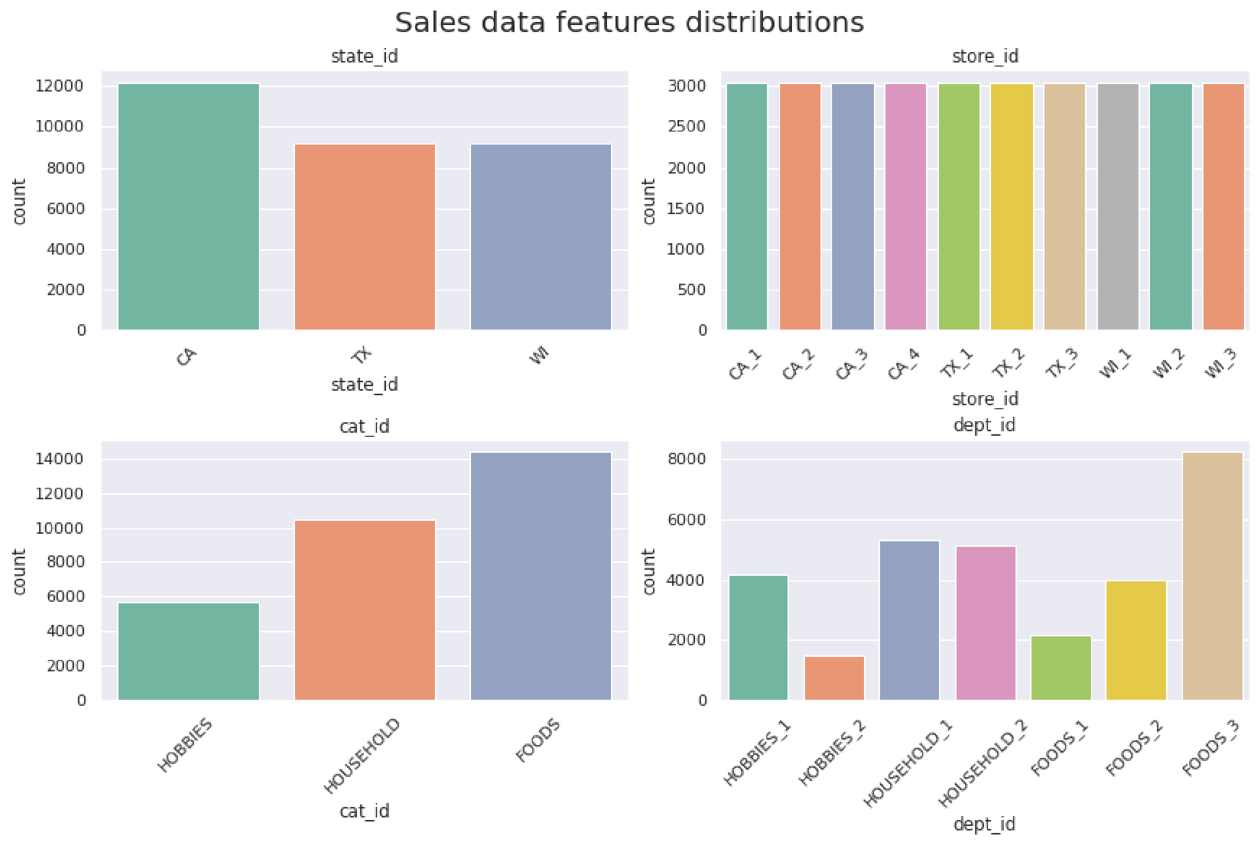
- The sale quantity is displayed as part of columns with the prefix "d_"
- The average number of units sold as zero is around 20730 per column out of all 30490 entries provided.
- We have 3049 unique items sold by every store from the data.
- We have all stores selling the same items. [as per the last cell output in ipynb]

2.1.2 Features-related observations

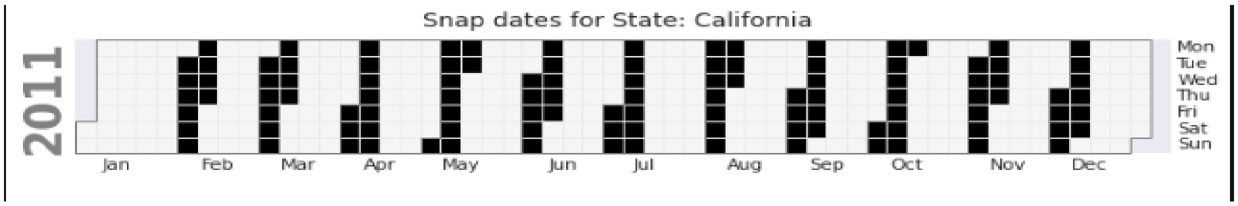
- For calendar features, it seems that weekdays and months are having nearly identical numbers of observations.
- Event type is having the most numbers of NaNs which are understandable as only 5-10% considering 16-17 public/religious/cultural holidays and another 20 odd days of different varieties make up a total of ~10% of the year. The remaining 90-95% are normal days with NaN values.

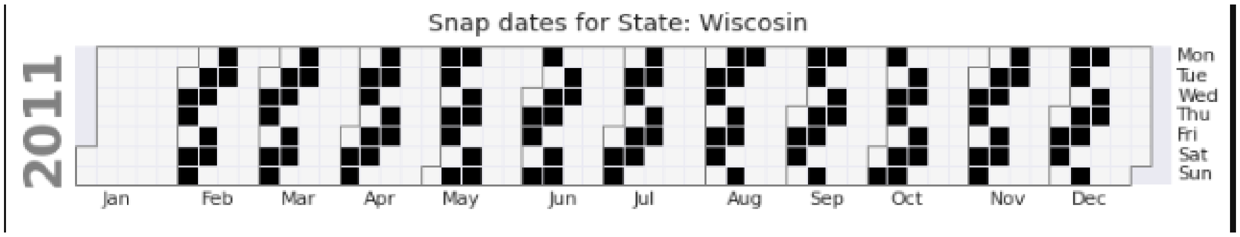


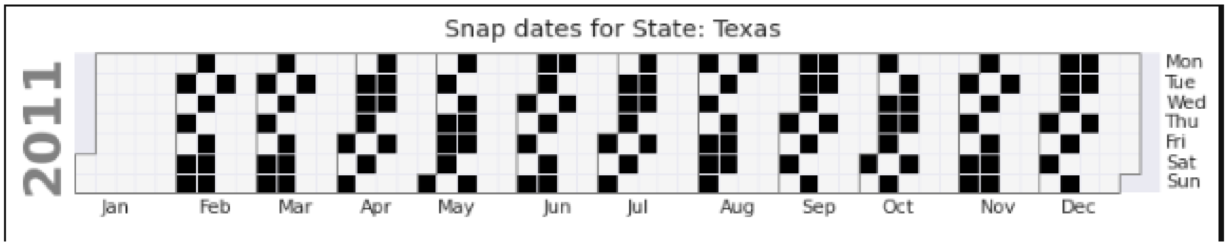
- California has more sales points. And Wisconsin & Texas have the same number of points.
- The increasing order of count of categories is Hobbies, HouseHold, Food.



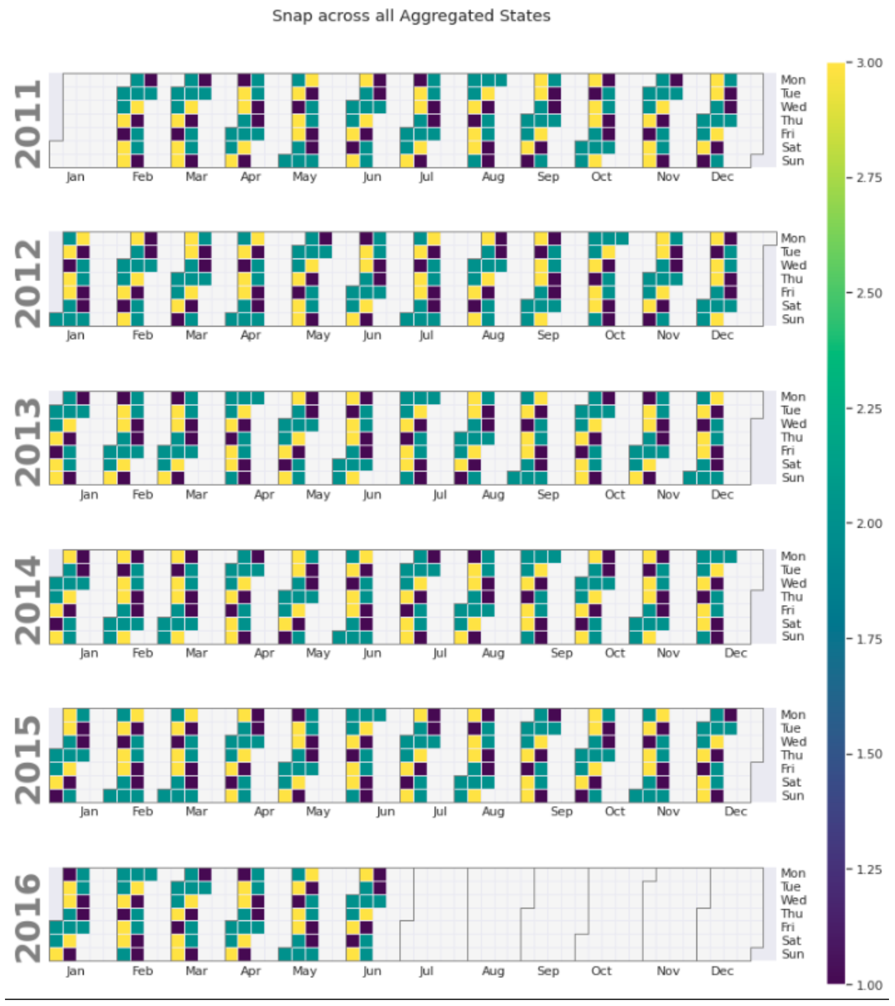
- SNAP stands for Supplemental Nutrition Assistance Program. From the SNAP website description: “The Supplemental Nutrition Assistance Program (SNAP) is the largest federal nutrition assistance program. SNAP provides benefits to eligible low-income individuals and families via an Electronic Benefits Transfer card. This card can be used like a debit card to purchase eligible food in authorized retail food stores.”
- California has continuous 10 SNAP days within the first half of the month every time. Wisconsin has 2 days following a non-Snap day and the same way within the first half of the month. Texas does not have a specific pattern for selected SNAP days. [Single-year data for reference.]







- All the Snap dates are in the first half of the month for all states. This certainly helps to measure their impact and make predictions more robust.

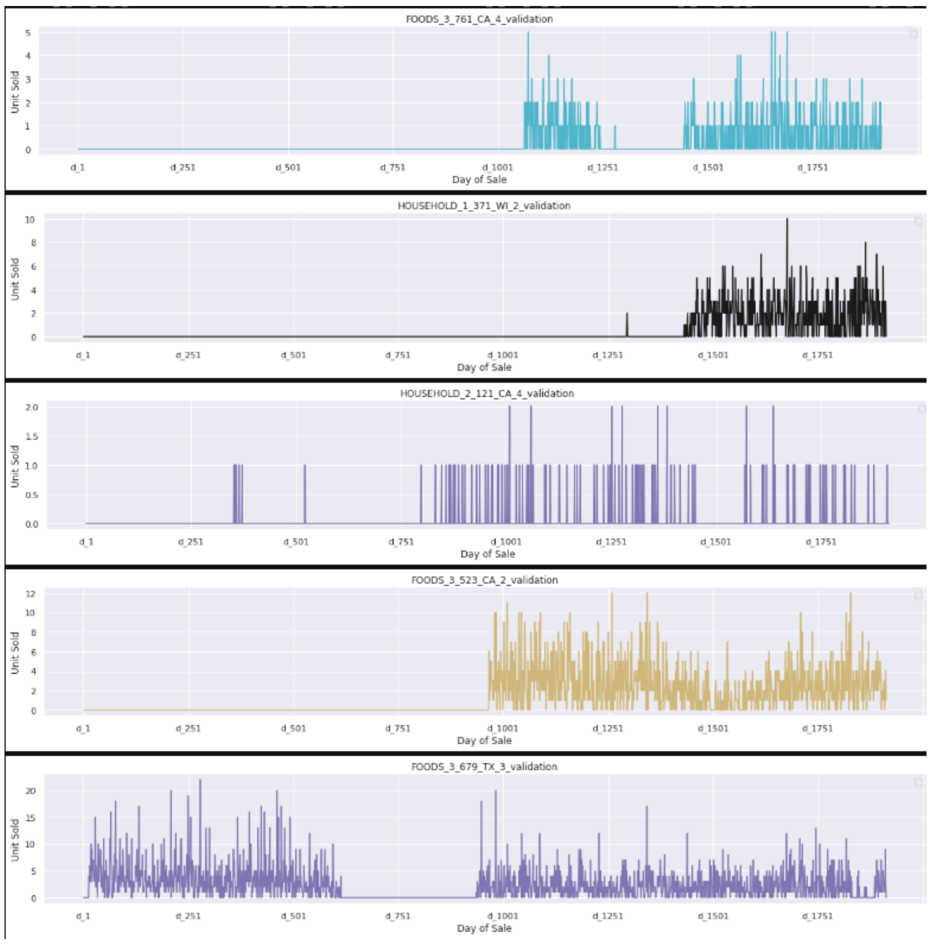


2.2 Time Series Exploratory Analysis

2.2.1 Univariate Time-Series

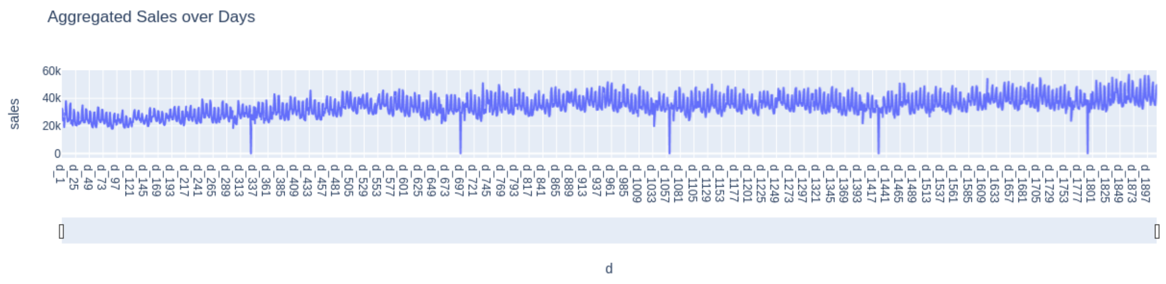
2.2.1.1 Item Level

- As there are 3049 products, we can't visualize them all in a single graph as it will be noisier. So here I have chosen random sample[, which can be changed but changing just variable "number_of_samples".] Here X-axis denotes the day while Y-axis denotes units sold. X shares the same scale, however, Y can't for this because for a few products there are mostly zero units or 1 unit sold.
- There are irregular patterns as there valleys/gaps present ~40-50% time (0 sales values) in most of the items. Which might be a data collection issue or problem of its own.
- No particular trend was found for items from this.
- We can use Correlational & partial correlation charts to check how many days we get patterns for individual products/categories.



2.2.1.2 Aggregated Sales per day

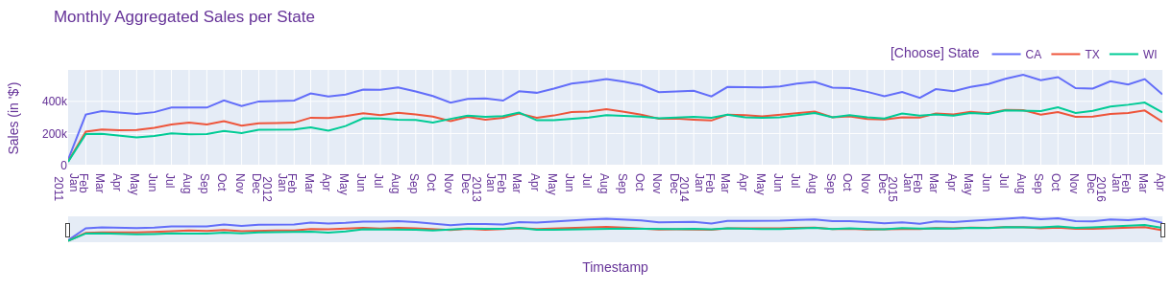
- The sales are increasing by each year/day goes. Which is a good indication from the sample collected.
- The sales in the latest year, increase at a pace greater than the previous year. At least that can be confirmed from the data available.
- There are yearly, monthly & weekly patterns which we will see in detail during the next analysis. [Try the day scroller to look for patterns manually.]
- There are 5 drops nearly valued 0, are Chrismas Holiday.



2.2.2 Multi-Variate Time-Series

2.2.2.1 Sales by Each State

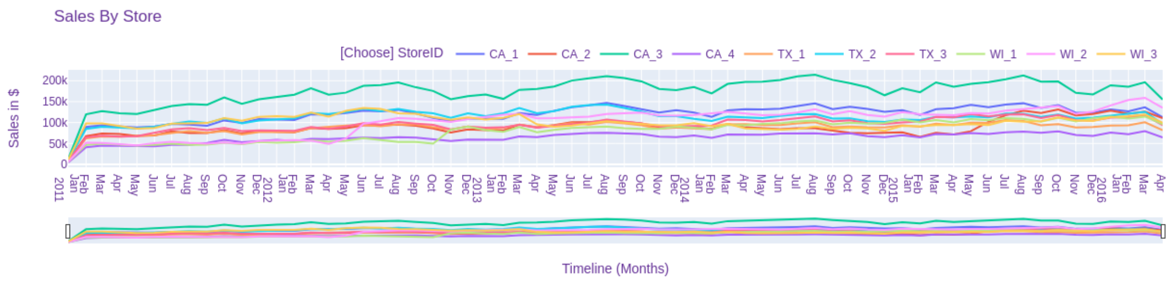
- Sales are increasing in all 3 states.
- All three mostly follow the same increase and decreasing patterns. But the scale is different for each of them
- California [CA] is having the highest number of sales every time. The main reason might be there are 4 stores in California while the other 2 states have 3 stores each. Wisconsin started lower than Texas but by the end of our training data, it overtook the Texas sales.



2.2.2.2 Sales by Each store

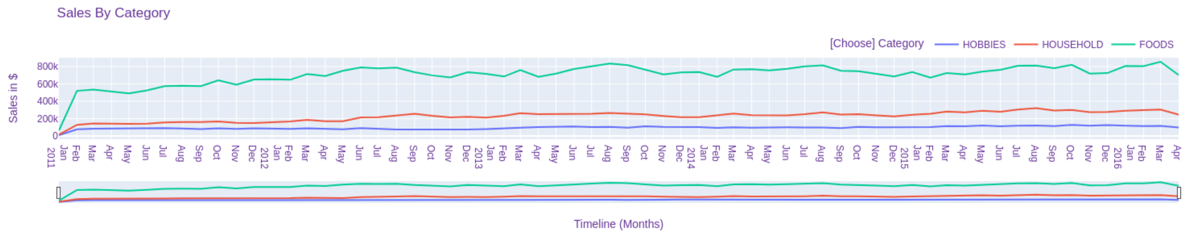
- The Texas stores are quite close together in sales; with “TX_3” rising from the levels of “TX_1” to the level of “TX_2” over the time of our training data.
- The Wisconsin stores “WI_1” and “WI_2” showed a drastic increase in sales in 2012, while “WI_3” shows a drop over several years.

- The California stores are relatively well-separated in-store volume with "CA_3" being at the top. Note “CA_2”, which declines to the “CA_4” level in 2015, only to recover and jump up to “CA_1” sales later in the year.



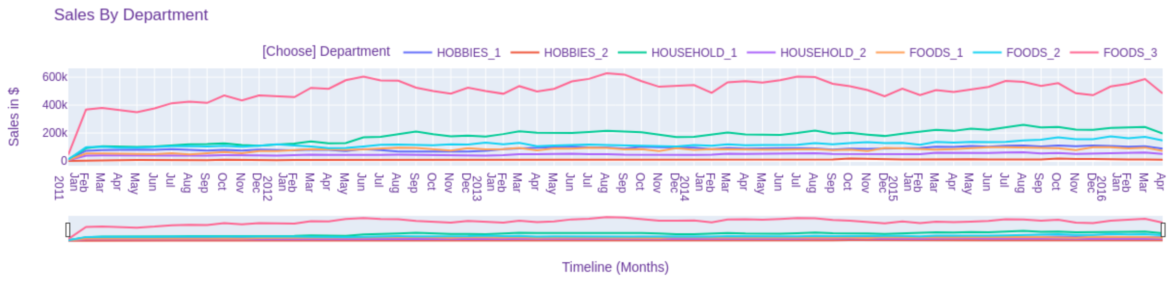
2.2.2.3 Sales by Each Category Type

- As seen previously, the Foods category has the most number of datapoint/rows in the validation set.
- Also sales figures are also in the same order, FOODS > Household > Hobbies



2.2.2.4 Sales by Department Type

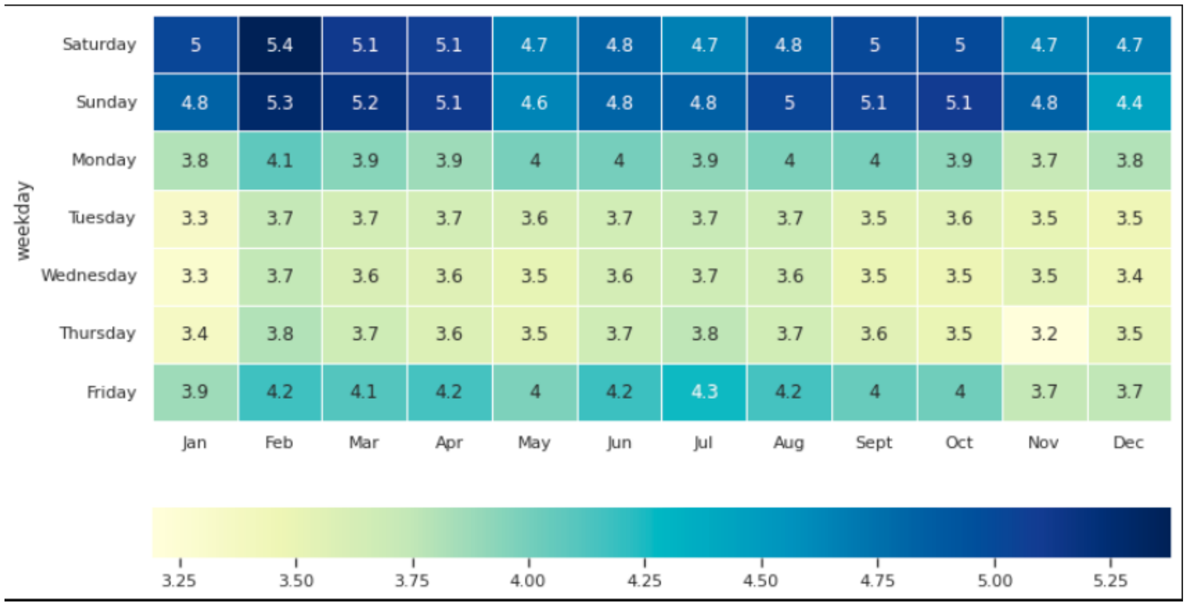
- Majority of the sales are driven by 2 departments: 1) FOOD3 & 2) HouseHold1
- Hoobies2 & HouseHold2 departments are majorly steady all the time. HOBBIES2 is not having so much impact on sales over this timespan suggesting that it is inefficient.
- Household2 has similar steadiness however, the scale/sales are 4x of Hobbies2.



2.3. Multi-variate effects on Sales

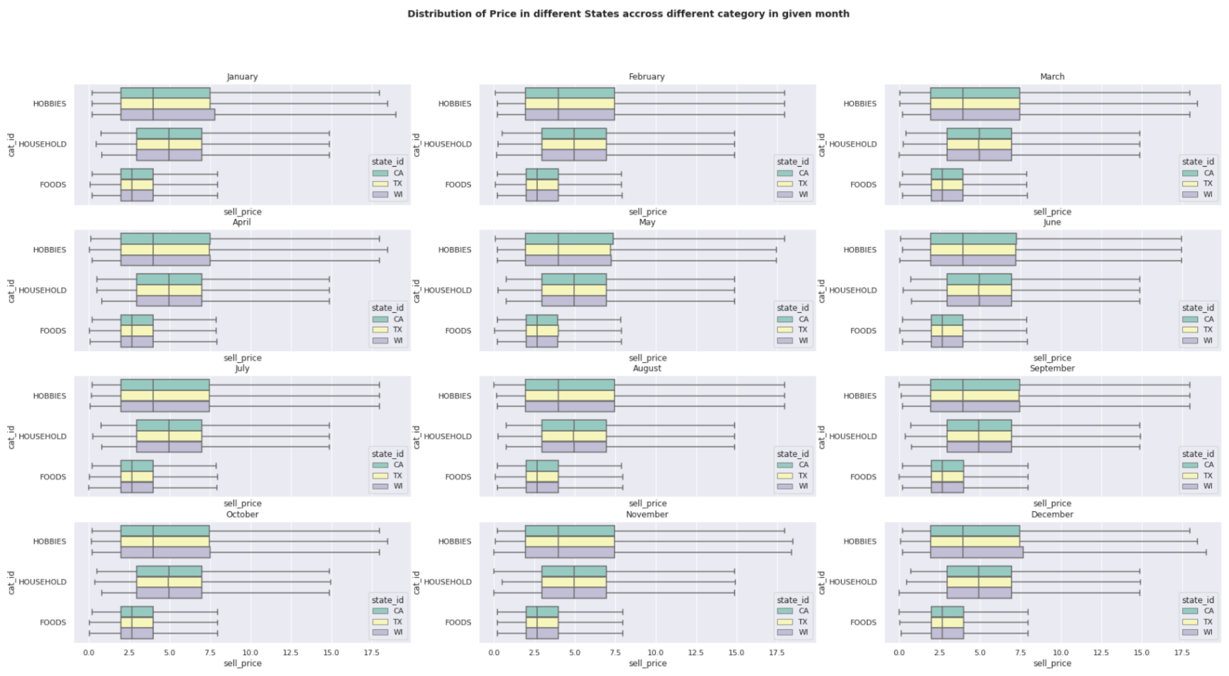
2.3.1 Effect on average price Weekday effect over year

- Sales pattern with the weekly repetition shows a strong relationship. Especially Saturday & Sunday are having strong sales.
- Friday and Monday are having effects as well on weekend sales.
- November and December are having a dip in sales numbers. There is another dip around May as well. Feb, March & Apr are having strong weekly sales patterns.



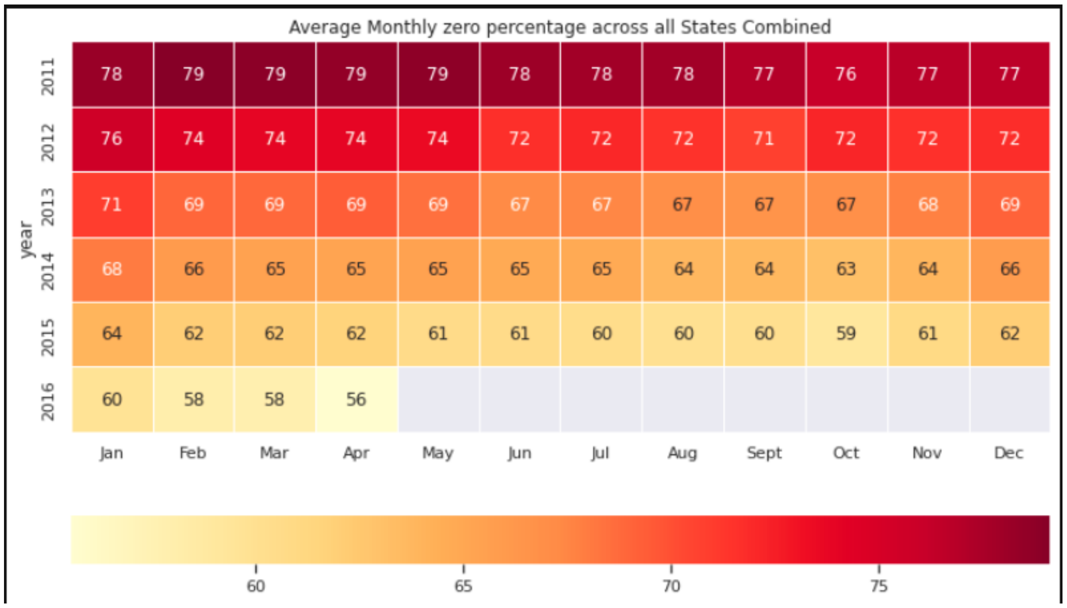
2.3.2 Price distributions across different states for the same product categories

- Prices are provided on average prices on a monthly basis.
- Price distribution across different categories has many outliers, so in this case, I have extended the whisker length to 2 IQR instead of 1.5 IQR to get the most out of our data and didn't show the remaining outliers.
- As you can see from the box plot, we have consistent sales prices across states for all categories. So we can confirm the price distributions is the same so the sales/revenue dominated based on the unit sold. Which is necessary to confirm for same price level products.
- Same goes another way around for clarification in the second boxplot.



2.3.3 Monthly averaged percentage zeroes for product units

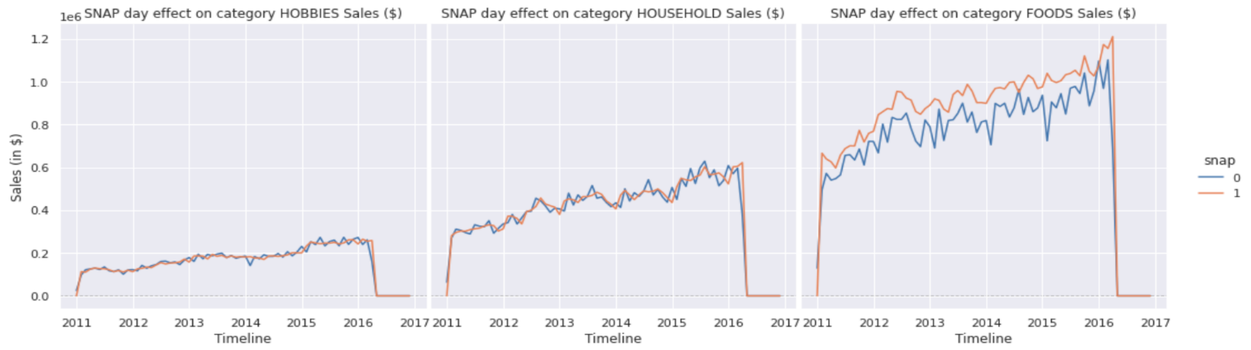
- Below heatmap calculates the number of product units sold averaging it to the monthly level.
- As zero unit sold count decreases, the percentage Monthly average product sales count is increasing continuously we can conclude. This is also a signal that sales of all products are happening and fewer products are not staying in the shops only.



2.3.4 How Snap affects sales

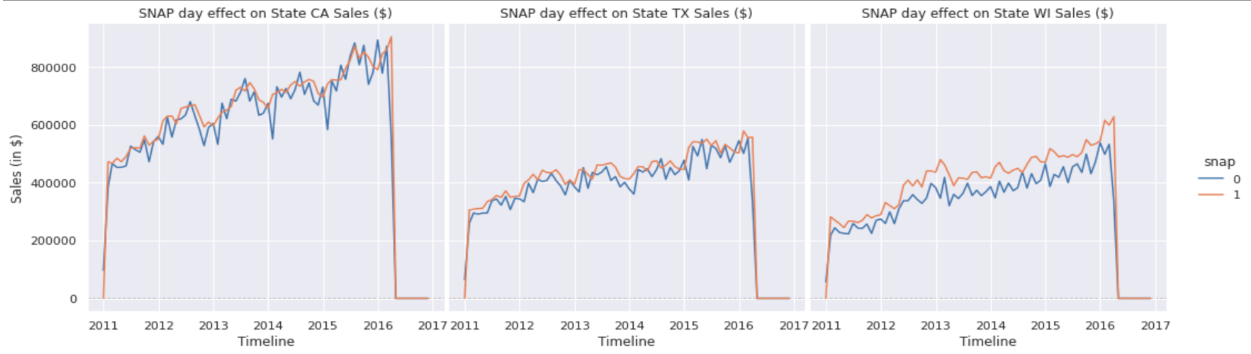
2.3.4.1 How Snap affects sales by Categories

- The maximum impact of snap can be seen in the FOODS categories. The other 2 categories have more or less the same number of sales on the SNAP included.
- Only SNAP days are considered here, the sales price can be affected by Holiday or Special event day along with SNAP inclusive day.
- Note: These sales are cumulative SNAP across categories so the state and its own SNAP days give a clear idea. However, for simplicity, this cumulative graph shows a global view of the SNAP effect across categories.



2.3.4.2 How Snap affects sales by State

- SNAP has a greater effect on WI (Wisconsin) state, where SNAP-inclusive days were ALWAYS driving more sales than NON-SNAP ones.
- For Texas as well the effect is visible in the graph where MOST of the days, it is driving more sales.
- For California, it is having an effect on fewer parts of the timeline. However, the graph shows the YEARLY SNAP sales pattern to be consistent except for 2016 due to partial year data.



Miscellaneous/If digging deeper

- Autocorrelation plots
- Rolling averages for smoothened line chart data
- Price Change of (sample)products years against Sales of those(sample) products

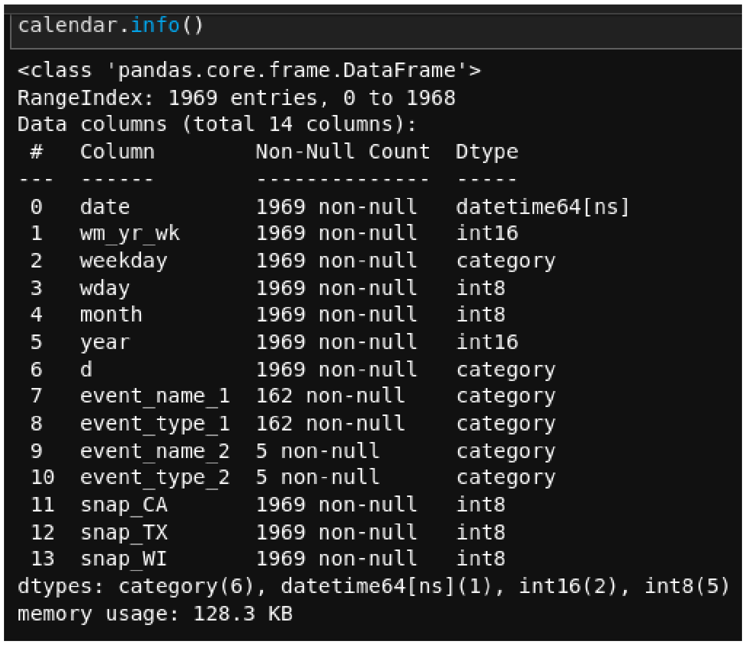
3. Data Wrangling & FE

3.1 Data Processing

3.1.1 Missing Data Handling

We have nearly ~12.3 Million missing values after melting so we have to keep this data by imputing appropriate values and average/mean values for that product was selected.

We had nearly ~92% event-related column values NaNs, however, it was mitigated by creating the OHE and dropping the first values which were NaNs. So the combinations of EventType columns give an idea about missing values along with Encoding in place.

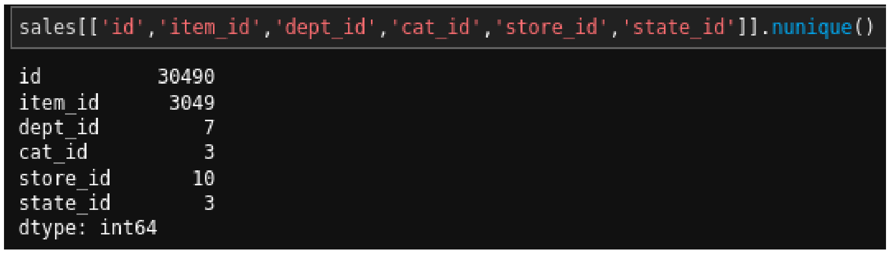


```
calendar.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1969 entries, 0 to 1968
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   date          1969 non-null   datetime64[ns]
 1   wm_yr_wk      1969 non-null   int16
 2   weekday       1969 non-null   category
 3   wday          1969 non-null   int8
 4   month         1969 non-null   int8
 5   year          1969 non-null   int16
 6   d             1969 non-null   category
 7   event_name_1  162 non-null    category
 8   event_type_1  162 non-null    category
 9   event_name_2  5 non-null      category
 10  event_type_2  5 non-null      category
 11  snap_CA       1969 non-null   int8
 12  snap_TX       1969 non-null   int8
 13  snap_WI       1969 non-null   int8
dtypes: category(6), datetime64[ns](1), int16(2), int8(5)
memory usage: 128.3 KB
```

3.1.2 LabelEncoding

We have categorical columns such as (product) id, store_id, cat_id, dept_id, state_id, etc. We suppose to convert them into Nominal feature columns however, there is an issue with it being extremely high dimensional. This will affect the distance-based algorithm hard to distinguish the point in high dimensions. Along with this, it will increase the storage limits to push. So for simplicity, it needs to be converted to a mapped object. So here we have to convert them to mapped objects.

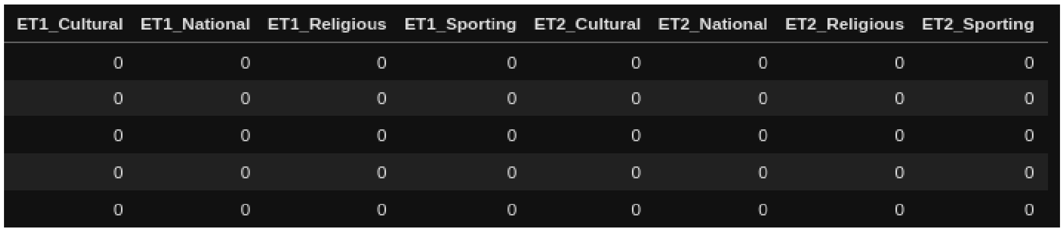


```
sales[['id','item_id','dept_id','cat_id','store_id','state_id']].nunique()

id          30490
item_id      3049
dept_id         7
cat_id          3
store_id       10
state_id        3
dtype: int64
```

3.1.3 One-Hot Encoding

For event type, I have verified that it has a limited number (ET1 - 4 & ET2 - 4) of categories and so we can use One Hot Encoding, which will convert them to individual categories to columns. This way model can relate which type of events will have more effect on unit sales. These columns encode data from event names columns so we can omit/drop them as they became redundant.



ET1_Cultural	ET1_National	ET1_Religious	ET1_Sporting	ET2_Cultural	ET2_National	ET2_Religious	ET2_Sporting
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

3.2 Feature Engineering

3.2.1 Simple feature extraction data

We have fields which can be interpreted by humans easily however, we need to convert them into machine-useful features. Like from field

- “Date”: ‘month’, ‘year’, ‘Day_Of_Month’
- “wday” or “weekday”: whether it is “isWeekend” or not
- Eventtype OHE columns: How many total events were there on that day.
- After merging Calendar data with Sales, we have units_sold and the price of the product so we can create price * units => sell price for that product on that day.

3.2.2 Rolling Mean / Median / Standard Deviation

Rolling is an extremely important feature as it calculates statistics for a given **fixed** frame size. As far as stats are concerned, it needs to be experimented with.
Through experimentations of frame size ranging from 2 to 180, I found out that more recent rolling features are more accurate so I have added them to mean rolling features.

Note: due to RAM they needed to be done separately so it is of relative importance.

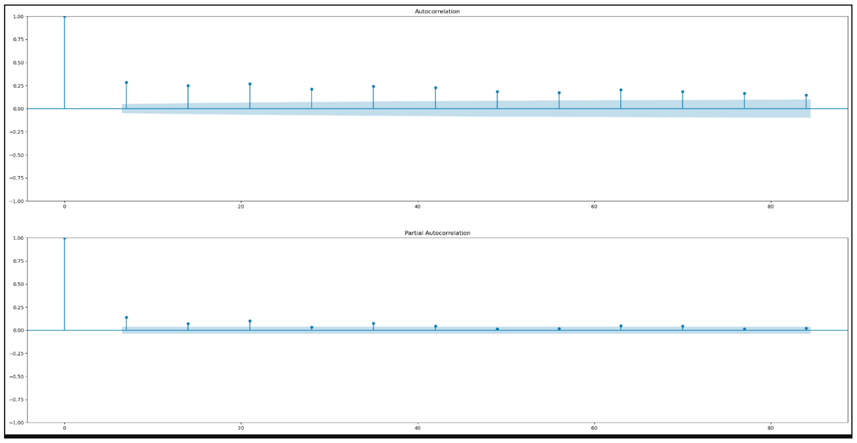
3.2.3 Introducing Lags

Value of k day is the time gap being considered and is called the lag. For example lag - 1 compares the current with 1 (unit) day value, which will help us get an idea of how many days' gap/shift is related to the current value.
Randomly sampled productids and find the most common lags that can be used as a feature with a 90% confidence interval. 95% was given very few so broaden the confidence interval to get an appropriate number of lags. For this, we are using the stats model library's plot_acf and plot_pacf functions.
Interpretation: Partial correlation:

It is the correlation between an independent variable and a dependent variable after controlling for the influence of other variables on both the independent and dependent variables.

Confidence intervals are drawn as a blue cone. By default, this is set to a 95% (in our case it is 90%) confidence interval, suggesting that correlation values outside of this code are very likely a correlation and not a statistical fluke.

Example:



3.2.4 Expanding Window Stats (Mean, Median, Deviation)

It is similar to a rolling window however, the size of the window increases its size one at a time and aggregated window size, we will apply our stats function (i.e. Mean, Median, Standard Deviation).

3.2.5 Mean Encoding

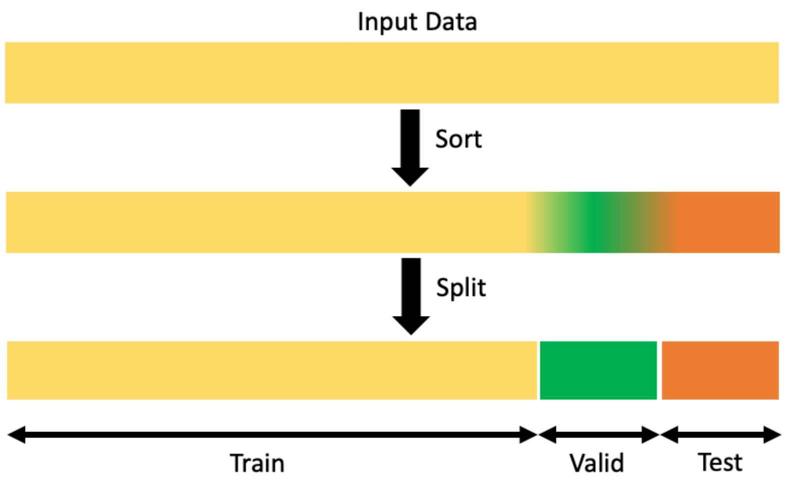
As we have many categorical variables in our data set we can create as many features by grouping them in different combinations and getting mean/std stats of target variable unit sold. For our use case, we have used 12 different grouping combinations.

3.2.6 Selling Trend feature

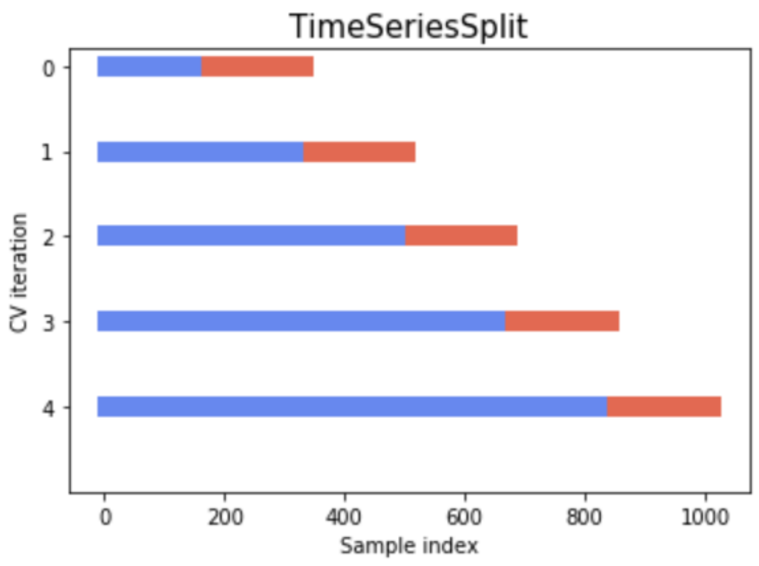
Selling trend determines how the selling of a particular product is done. For that, we will determine the total sold units for a given product on a given date. Then subtracting the overall mean sales values for a given date will result in a trend point for those days. Plotting will show the trend.

3.2.7 Data Split strategies employed

As we have a time-series problem, I have first split the data in Train-Validation-Test with 2 steps (Data => (Train-Val) + Test & (Train-Val) => Train + validation).



Furthermore for cross-validation purposes in time-series is a little bit different so I have used Scikit-learn's TimeBasedSplit and split the Train data among 3 Folds.



4. Models Experiment and Results

It has been also noted that even if we consider zero units sold for the future, our WRMSSE score is **5.8307046792717365.** This can be found in the Time-Series & Weight Generation step of the notebook.

4.1 Statistical Model

4.1.1 Simple Moving Average:

It calculates the simple mean over the window size specified. Pandas data frame has inbuilt functions for moving averages. The larger the size of the window, the smoother the curve will be (if plotted).

$$\begin{aligned} SMA_k &= \frac{p_{n-k+1} + p_{n-k+2} \cdots + p_n}{k} \\ &= \frac{1}{k} \sum_{i=n-k+1}^{n} p_i \end{aligned}$$

4.1.2 Cumulative Moving Average

Averaging out the whole previous data along with the current value (not the fixed window size from the current day).

$$CMA_t = \frac{x_1+x_2+x_3+...+x_t}{t}$$

4.1.3 Exponential Moving Average

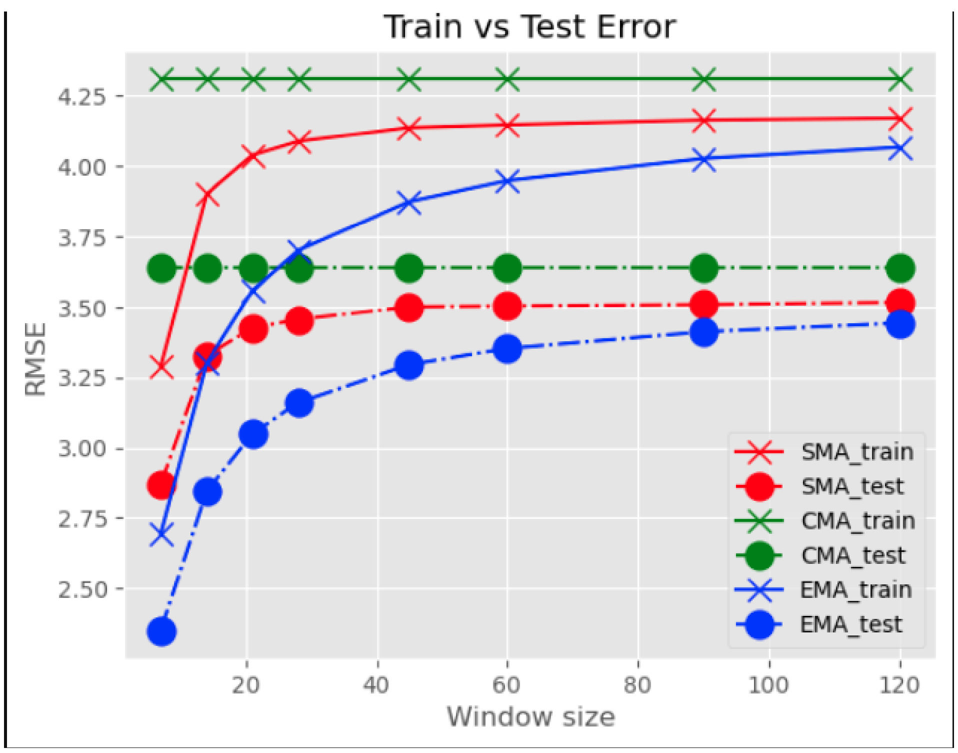
Here alpha is setting the weighing mechanism for k day span. Weighing more to recent data, which is generally seen in the Time-Series plot.

$$EMA_t = \begin{cases} x_0 & t = 0 \\ \alpha x_t + (1-\alpha)EMA_{t-1} & t > 0 \end{cases}$$

4.1.4 Comparison of MA

After trying different span/window sizes [7,14,21,28,45,60,90,120]. We found out that recent data give more accurate results. So selected a 7-day span.
From the curve, we can see that the RMSE increases drastically when we increase the frame size for MA.

Selecting the optimal value of frame size = 7 for the model. More recent data provides more accurate results for the near future. Selecting less amount of older data will also benefit the model training capabilities.



4.2 Machine learning Models

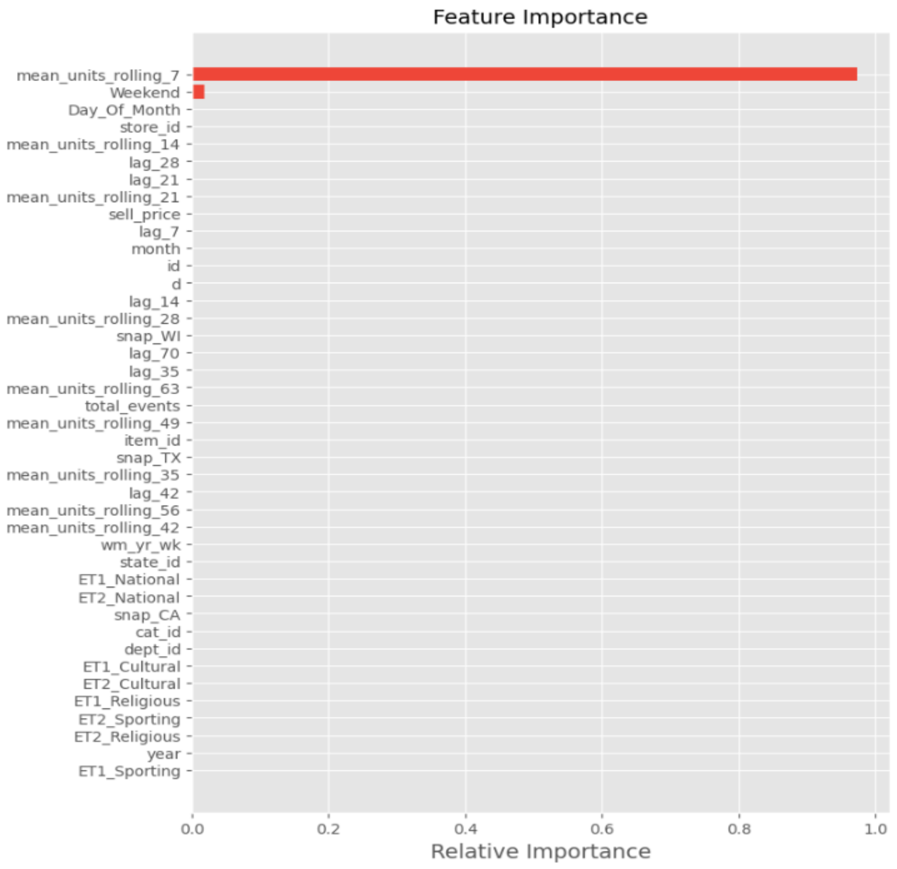
For hyperparameter tuning we are using RandomizedSearchCV class of sklearn. Where the CV parameter is handled with Time Series train-test split indexes rather than normal/randomized selection of train data points. The function can be found in Utility as most of the algorithms are going to be needed, Hyper-Parameter Tuning.

4.2.1 Tree-Based Model - Decision Tree

Decision trees are go to when you have a large dataset to train, efficient in runtime and you have no idea about what is important and what is not. It is easy to understand and has no scaling constrain. It is also able to handle both Numerical and Categorical data.

However, there are a few things to consider when building a Decision Tree model. We find a better fit with DT for the given dataset but it is not Optimal as finding it is itself an NP-hard problem. It is easy to overfit as it will create a tree node to all training nodes while training which is remembering the training set or overfitting. We need to stop the tree from creating those leaf nodes as training points purely and the below parameters are generally tuned to mitigate Overfitting.

Best Hyper-Parameters	{'min_samples_split': 379, 'min_samples_leaf': 461, 'max_depth': 94}
Test set RMSE	1.784
WRMSEE	0.57237



4.2.2 Linear Model - Stochastic Gradient Descent

Linear Models like Ridge, Lasso, or ElasticNet, work on small amounts of data. This is where SGD plays a vital role. SGDRegressor has different loss functions, and penalization options to fit data well. And especially with the last training samples (>10k).

Primarily using this model helps us with many tunable parameters that help our goal of achieving better results while not overfitting.

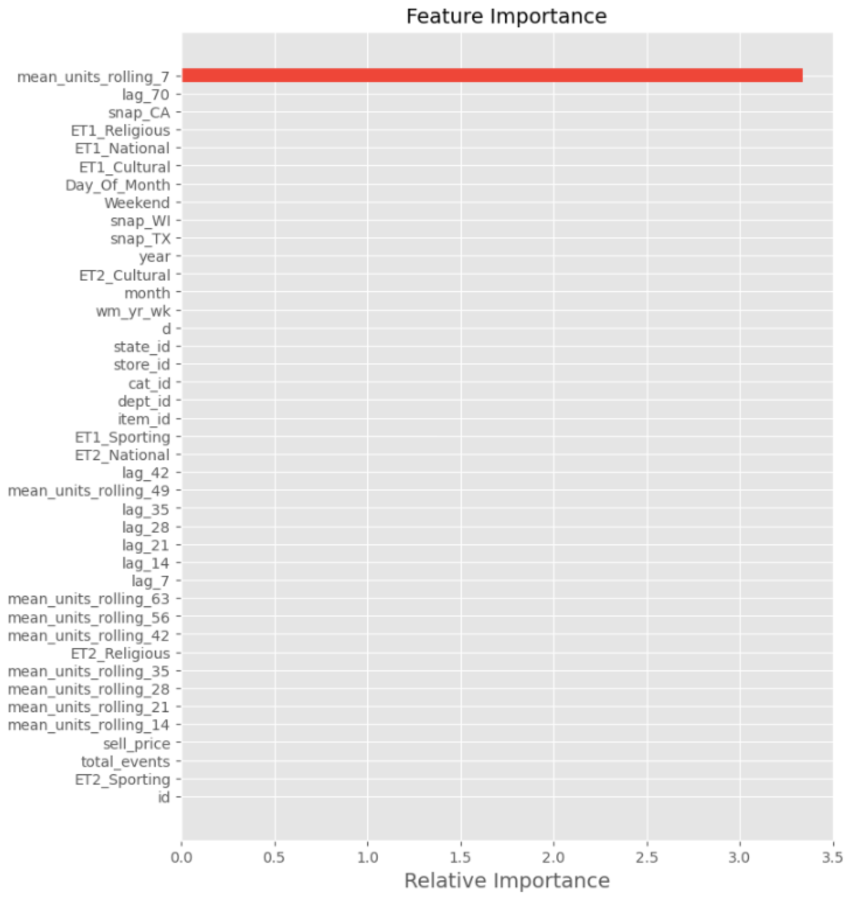
The major advantage of SGD is its efficiency, which is basically linear in the number of training examples. If X is a matrix of size (n, p) training has a cost of, where k is the number of iterations (epochs) and is the average number of non-zero attributes per sample.

The limitation/additional step with SGD Scaling features is necessary as it is susceptible to feature values.

The hyper-parameter tuning can be early stopped if the selected loss is not within tolerance criteria (0.001) or the number of iterations completes.

Here alpha is a multiplier of the Regularization term.

Best Hyper-Parameters	{'penalty': 'l1', 'loss': 'squared_error', 'alpha': 0.8}
Test set RMSE	1.913
WRMSEE	2.02



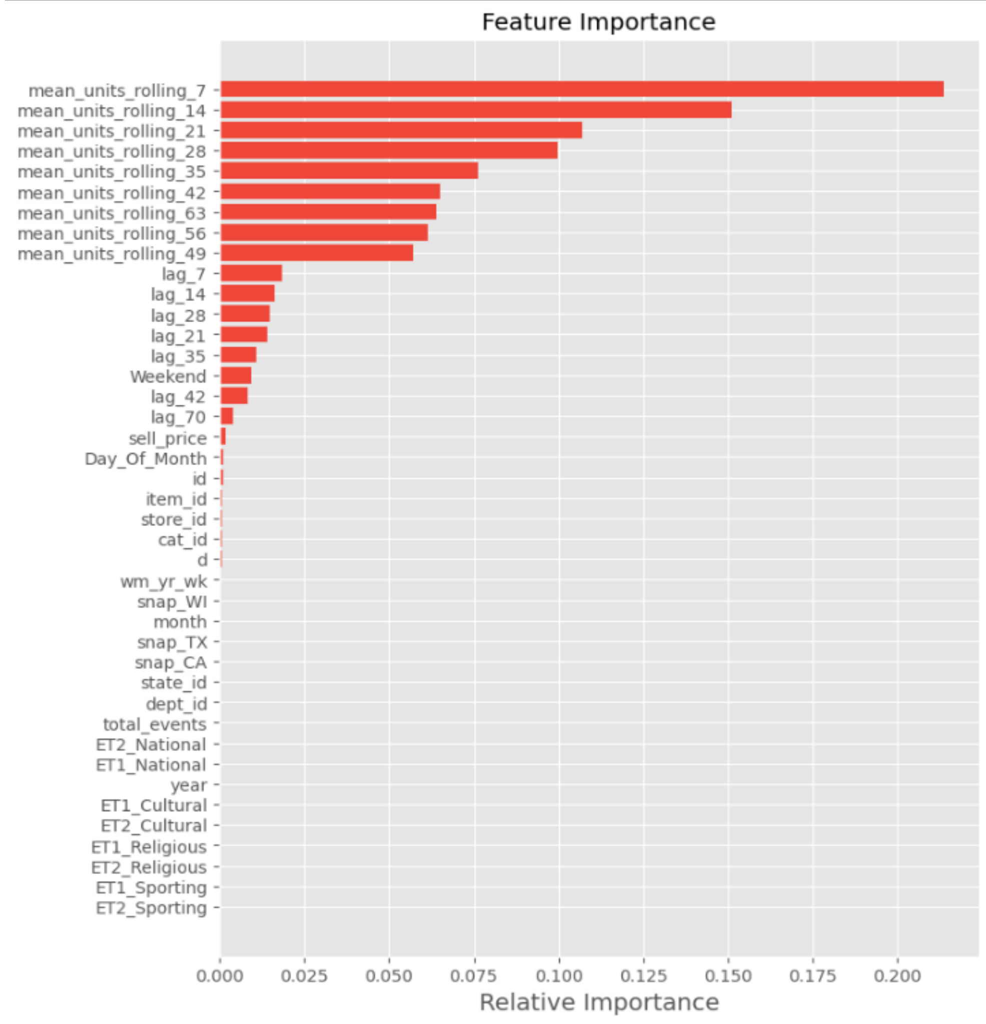
4.2.3 Ensemble Model - Random Forest Regressor

Random forest is decision tree model only, however, instead of creating single tree we will create multiple trees (n_estimators). These DTs will be working with a subset of rows and columns of training data. This is making individual trees weak learners but the collection of weak learners makes a model more generalizable. We are creating 70%

of rows and X of columns [Hyper-Parameter] for individual DT training which will decrease the variance. This makes a more robust model for overfitting.

These individual models are not dependent on each other making them parallelizable. However, this technique is quite computationally intensive.

Best Hyper-Parameters	{'n_estimators': 95, 'min_samples_split': 175, 'min_samples_leaf': 129, 'max_features': 'sqrt', 'max_depth': 95}
Test set RMSE	1.79
WRMSEE	0.60348



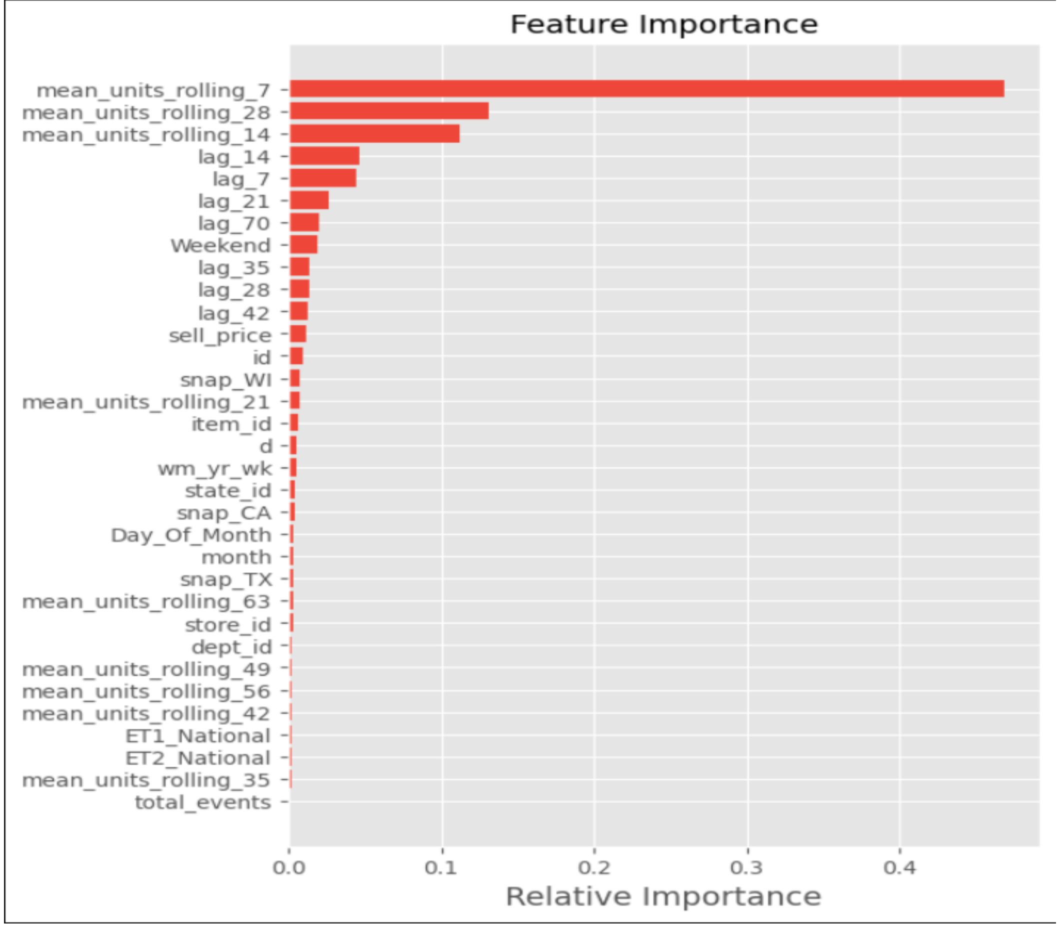
4.2.4 Ensemble Model - XGBoost

XGBoost has become default choice in ensemble tree-based model due to its speed. It has a linear model solver and a tree-based learning algorithm. This makes it parallelizable, hence fast in execution.

XGB gives more importance to Functional Space than hyperparameter space (RF). Random Forest has to wait for all trees to complete to collect results while XGB needs the gradient of data. XGB is boosting type of ensemble while RF is bagged. Tree-based algorithms generally tend to overfit if not taken care of hyperparameters.

From the guidelines, we have tuned the model to mitigate overfitting.

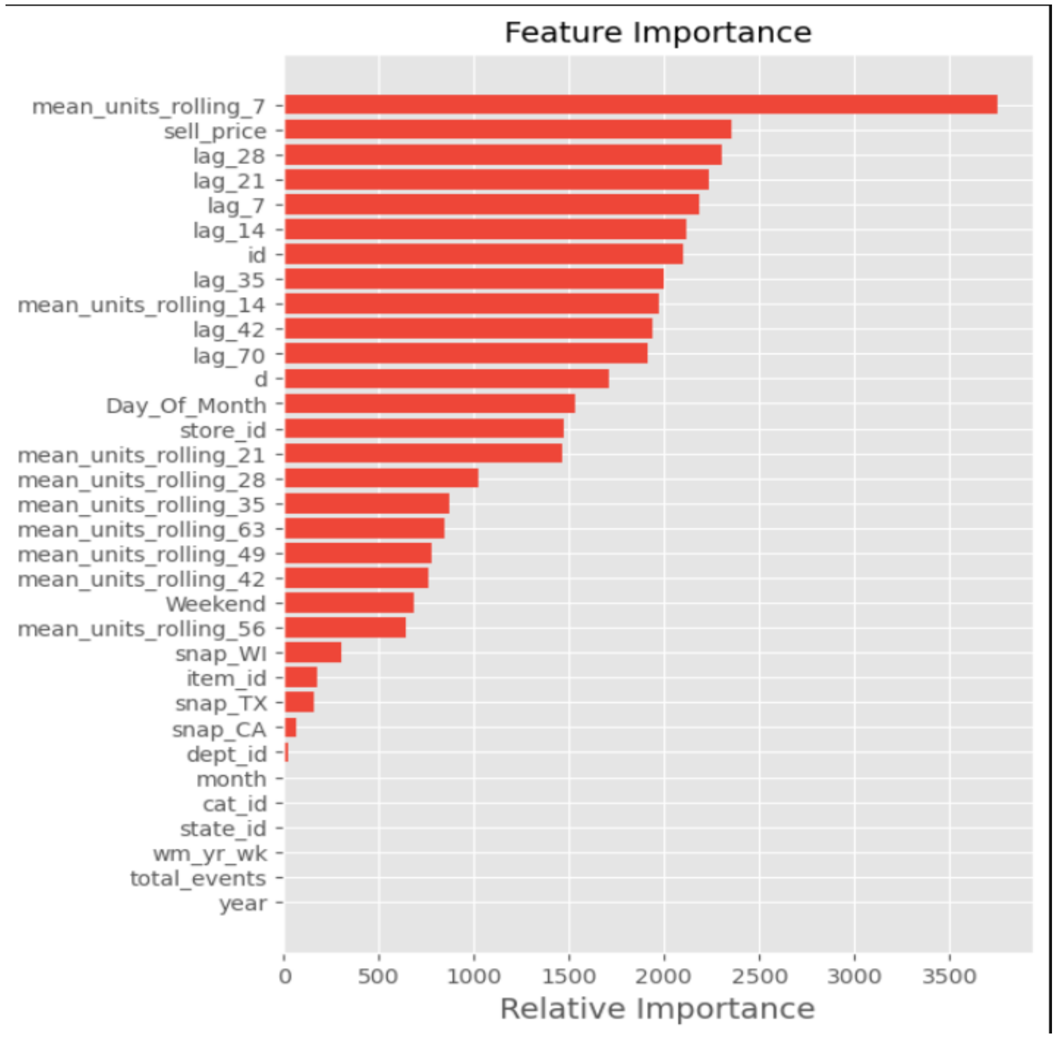
Best Hyper-Parameters	{'learning_rate' : 0.031, 'max_leaves' : 139, 'min_child_weight' : 99}
Test set RMSE	1.827
WRMSEE	0.62749



4.2.5 LightGBM

LightGBM is based on the leaf-wise tree growth algorithm, but other tools use depth-wise tree growth. Compared with depth-wise growth, the leaf-wise algorithm can converge much faster. However, the leaf-wise growth may be overfitting if not used with the appropriate parameters. Using these guidelines for parameter tuning and effects [12], we have tuned the model.

Best Hyper-Parameters	{ 'learning_rate' : 0.035, 'max_depth' : 148, 'Num_leaves' : 375, 'Lambda_l2' : 0.05}
Test set RMSE	1.783
WRMSEE	

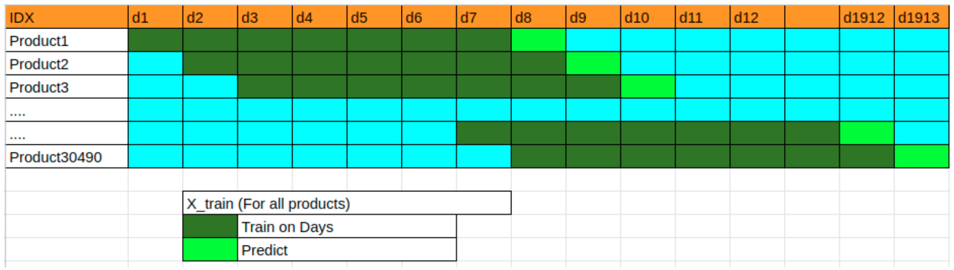


4.3 DL and other Advance models

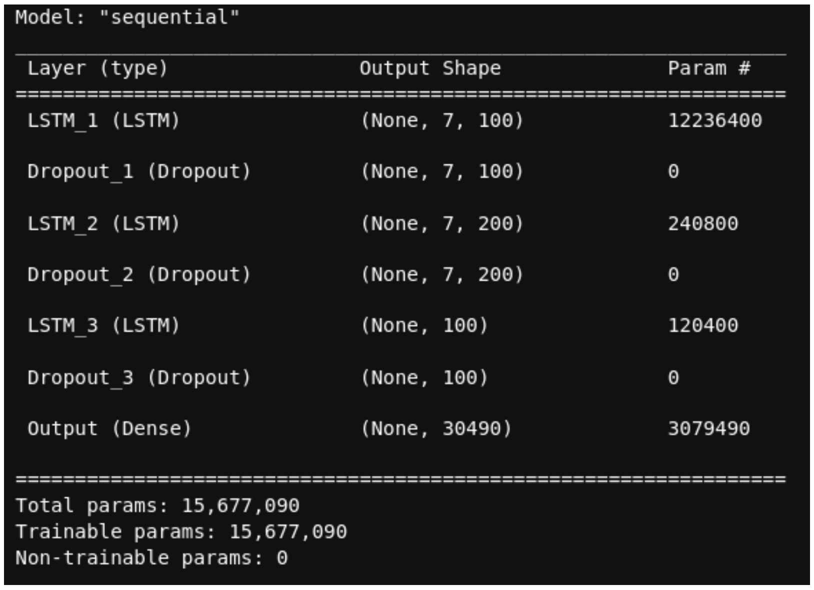
4.3.1 LSTM

Neural Network/Deep Learning can solve large data problem more effectively. But it tent to overfit data very fast and computationally expensive. We can create RNN with the arbitrary designed network with trial and error. However, we face issues of Exploding and Vanishing gradients. These problems are also can be solved using the regularization methods and Neural Network best practices of introducing Batch Normalization, DropOut layer. As we are solving a time-series problem, past data has an effect on current/latest data. So that's something that can be found in what LSTM offers. Rather than updating whole weights, it works internally to make additive changes. Based on our EDA, we found out that more recent data provides better insights into data, which can be utilised by LSTM layers.

I have tried keeping the window of the last [7,14,21,28] days and measuring the loss minimization with a set of HyperParameters. [For a few, experiments, I faced issues with RAM issues which negated that set of parameters with a larger history window]



Model Summary:



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 LSTM_1 (LSTM)               (None, 7, 100)            12236400

 Dropout_1 (Dropout)         (None, 7, 100)            0

 LSTM_2 (LSTM)               (None, 7, 200)            240800

 Dropout_2 (Dropout)         (None, 7, 200)            0

 LSTM_3 (LSTM)               (None, 100)               120400

 Dropout_3 (Dropout)         (None, 100)               0

 Output (Dense)              (None, 30490)             3079490

=================================================================
Total params: 15,677,090
Trainable params: 15,677,090
Non-trainable params: 0
```

Model Created using Keras Sequential API. This makes adding layers very easy just like Scikit-Learn API.

Arch Params	{'L1_cells': 100, 'L1_DropOut': 0.5, 'L2_cells': 200, 'L2_DropOut': 0.3, 'L3_cells': 100, 'L3_DropOut': 0.2,}
Tuning Param	{ 'n_epoch' : 30, 'batch_size' : 100}
WRMSEE	5.11691

Without any feature engineering, vanilla version of it doesn't give any promising solution. Implementing the LSTM-related RNN model is easy but tending it to our needs extra care. Memory requirements increase drastically even if we have a large window [HistoryLookup parameter] of data for predicting a single extra day.

4.4 Comparing the performance of the model

From the tuned model, Tree-based models perform better with this set of datasets. A single decision tree model will not work for all cases as it seems to overfit data quickly and will not be stable. By means of stability, when the subsample of data changes slightly, the tree structure will change drastically.

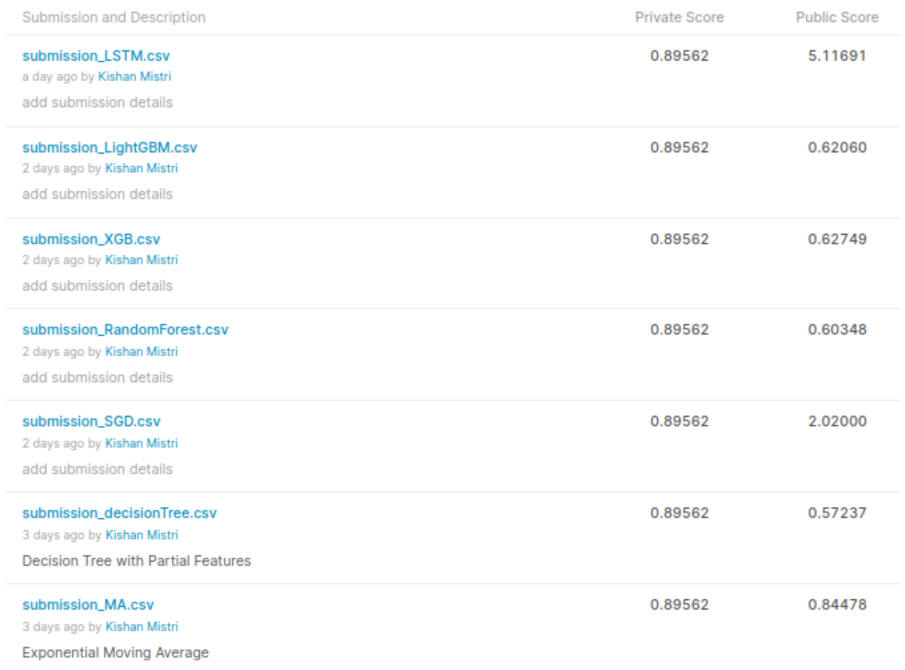
Model	RMSE	WRMSSE (Kaggle)
EMA	2.351	0.84478
Decision Tree	1.784	0.57237
SGD	1.913	2.02
RF	1.79	0.60348
XGB	1.827	0.62749
LightGBM	1.783	0.6206

We see that Random Forest Regressor is having 2nd best results for RMSE & WRMSSE. However, the training of the Random Forest regressor is slower compared to LightGBM Regressor. Also, the model size has drastically different.

If this model needs to be used in production on the data stream, using the most accurate model might not work every time as it has its limitations with particular dataset segments.

We see that Random Forest Regressor is having 2nd best results for RMSE & WRMSSE. However, the training of the Random Forest regressor is slower compared to

LightGBM Regressor. Also, the model size has drastically different. You can find how other models predict accurately from [19].



Submission and Description	Private Score	Public Score
submission_LSTM.csv a day ago by Kishan Mistri add submission details	0.89562	5.11691
submission_LightGBM.csv 2 days ago by Kishan Mistri add submission details	0.89562	0.62060
submission_XGB.csv 2 days ago by Kishan Mistri add submission details	0.89562	0.62749
submission_RandomForest.csv 2 days ago by Kishan Mistri add submission details	0.89562	0.60348
submission_SGD.csv 2 days ago by Kishan Mistri add submission details	0.89562	2.02000
submission_decisionTree.csv 3 days ago by Kishan Mistri Decision Tree with Partial Features	0.89562	0.57237
submission_MA.csv 3 days ago by Kishan Mistri Exponential Moving Average	0.89562	0.84478

If this model needs to be used in production on the data stream, using the most accurate model might not work every time as it has its limitations with particular dataset segments.

5 Conclusion

As shown in the model comparison, after experiments with different models and sets of parameters Moving Average models, Decision Tree model, and SGD model from the Linear model class. Random Forest Regressor and LGBM models from Ensemble model class, Long Short Term Memory (LSTM) from Neural network part, the most accurate ones are Random Forest, LGBM and Exponential Moving Average for predicting/forecasting 28 days unit sales for our dataset. LSTM models are more accurate when having more data and are tuned properly because of their memory-carrying capability, which is advantageous to time series problems. IAs part of future work we can create separate models for each category. There are other advanced models available. There were interesting ideas in the Kaggle to create separate models for each store, clustering-based solution, using Facebook's in-house tool prophet (fbProphet). Kaggle gives insights on solving specific solutions with different approaches from different backgrounds.

Overall, this case study significantly improved my/our understanding of different models from the separate classification of machine learning models working with time series data and I am looking forward to exploring future possibilities with this work!

6 References

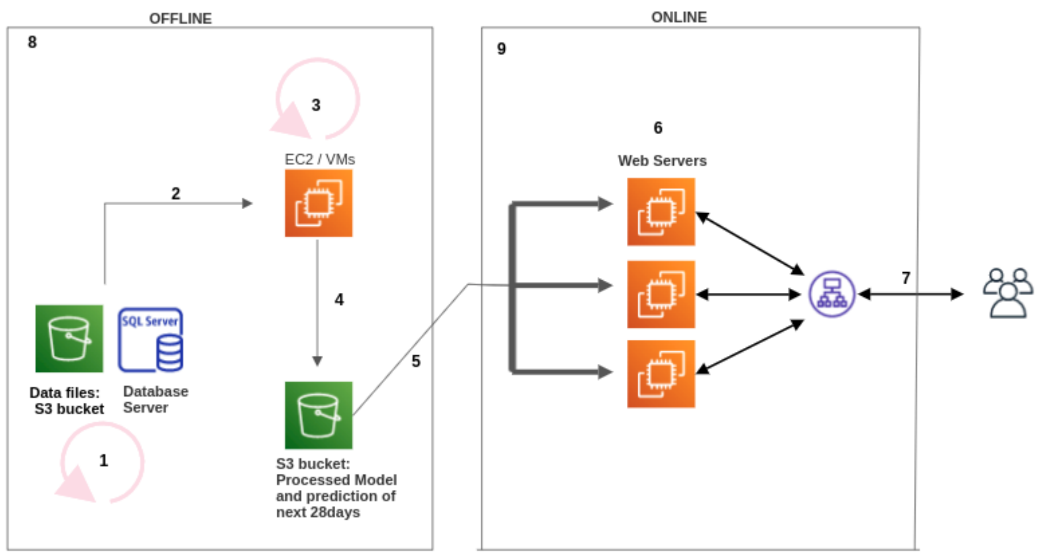
1. https://www.chargebee.com/blog/importance-of-sales-forecasting/
2. https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics
3. https://mofc.unic.ac.cy/m5-competition/
4. https://www.climate.gov/maps-data/dataset/past-weather-zip-code-data-table
5. https://hbr.org/1971/07/how-to-choose-the-right-forecasting-technique
6. https://robjhyndman.com/papers/mase.pdf
7. https://mobidev.biz/blog/machine-learning-methods-demand-forecasting-retail
8. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0262009
9. https://michelbaudin.com/2021/07/16/evaluating-sales-forecasts/
10. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
11. https://scikit-learn.org/stable/modules/ensemble.html#forests-of-randomized-trees
12. https://xgboost.readthedocs.io/en/stable/parameter.html
13. https://xgboost.readthedocs.io/en/stable/tutorials/param_tuning.html#control-overfitting
14. https://xgboost.readthedocs.io/en/stable/parameter.html
15. https://lightgbm.readthedocs.io/en/v3.3.2/Features.html
16. https://lightgbm.readthedocs.io/en/v3.3.2/Parameters-Tuning.html
17. https://github.com/microsoft/LightGBM
18. https://analyticsindiamag.com/when-to-apply-l1-or-l2-regularization-to-neural-network-weights/#:~:text=But%20it%20is%20suggested%20to,to%20have%20the%20same%20scale.
19. https://keras.io/api/layers/recurrent_layers/lstm/
20. https://www.nbi.dk/~petersen/Teaching/ML2020/FinalProject/FinalProject2_MariaAndyEmilMads_WalmartKaggle.pdf
21. https://arxiv.org/pdf/1703.04691.pdf
22. https://stackoverflow.blog/2020/10/12/how-to-put-machine-learning-models-into-production/
23. https://docs.streamlit.io/library

7. Deployment

The deployment of the machine learning model into production efficiently reduces human workloads. Due to that reason, there are many frameworks in place and used by the majority of businesses. However, to understand the whole cycle of deployment we will be creating architecture based on our needs and improving upon it. As stated earlier, we will have static data so we won't be able to showcase every functionality. But the opportunity details will be discussed briefly here. We have created modular architecture, so different tasks can be decoupled and handled in case of troubleshooting.

This application aims to show end-users the 28-day forecasted values of a particular item along with historical data in graph format through web UI. Our data is updated daily so we can re-train the model once a day during offline hours. The used Business metric is WRMSSE, as it will help the management stakeholder to take action. However, from a modelling perspective, we won't minimise the individual differences in predictions hence choosing RMSE here.

Production Deployment Flowchart



Actions

1: Database gets updated by third party services (in production), we have static data here.

2: EC2 pulls the latest required data from S3 data bucket, processes it and trains the model.
As the data is being updated once a day, live training is not needed.

3: RAM intensive & multi core machine: As the latest data is available, Our selected model notebook will process it once based on crontab set and model training will be done. It will also creates the prediction files for next 28 days for charts and saves them. Once

4: Saved model and file will be saved in S3 separate bucket for dash-boarding, validation, model enhancement.

5: From the saved output csv files, we will it will be pulled by our web server. Web server request have ttl set.

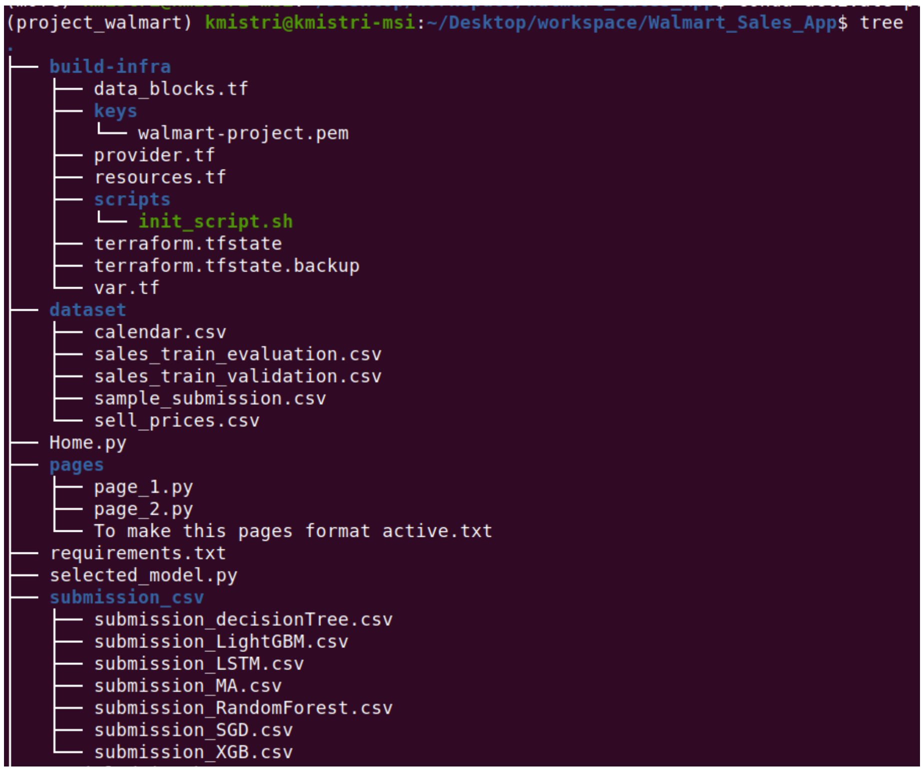
6: We will have light weight web servers, handled by ASG and behind load balancer. Our UI component, Streamlit app will be set here.

7: Users behind load balancer will hit load balancer endpoint and load will be distributed accordingly.

8: Our data processing pipeline will be offline and will be trigger once a day.

9: Our webservers will be active online all the time.

Directory structure:


```
(project_walmart) kmistri@kmistri-msi:~/Desktop/workspace/Walmart_Sales_App$ tree
.
├── build-infra
│   ├── data_blocks.tf
│   ├── keys
│   │   └── walmart-project.pem
│   ├── provider.tf
│   ├── resources.tf
│   ├── scripts
│   │   └── init_script.sh
│   ├── terraform.tfstate
│   ├── terraform.tfstate.backup
│   └── var.tf
├── dataset
│   ├── calendar.csv
│   ├── sales_train_evaluation.csv
│   ├── sales_train_validation.csv
│   ├── sample_submission.csv
│   └── sell_prices.csv
├── Home.py
├── pages
│   ├── page_1.py
│   ├── page_2.py
│   └── To make this pages format active.txt
├── requirements.txt
├── selected_model.py
├── submission_csv
│   ├── submission_decisionTree.csv
│   ├── submission_LightGBM.csv
│   ├── submission_LSTM.csv
│   ├── submission_MA.csv
│   ├── submission_RandomForest.csv
│   ├── submission_SGD.csv
│   └── submission_XGB.csv
```

For my deployment infrastructure creation, I have created **Terraform** script package named "build-infra" which will create AWS resources for the deployment. VM Resource(s) will satisfy the primary library & file structure requirements through the Shell script(s) passed as user data and remote execution code.

Python script "selected_model.ipynb", will be responsible for pulling data, training and generating resulting files (model and forecasting CSV). I have created/added crontab. As a part of project preparation.

```
# run the model notebook with the papermill process at 1 AM every day
0 1 * * * papermill ~/.Walmart_Sales_Deployment/selected_model.ipynb ~/.Walmart_Sales_Deployment/results.ipynb
```

The exported data will be used by Streamlit "Home.py" to process and create the visualization. The architecture diagram gives a detail description of the flow.

I have to choose the split methodology for 1) training and 2) dashboarding as

- Training requires high-specification machines and it is costly to keep them running just for web servers.
- Decoupling the deployment makes it modular, and easy to understand and manage.
- Target audience can be part of different teams and they do not need all accesses in one place.

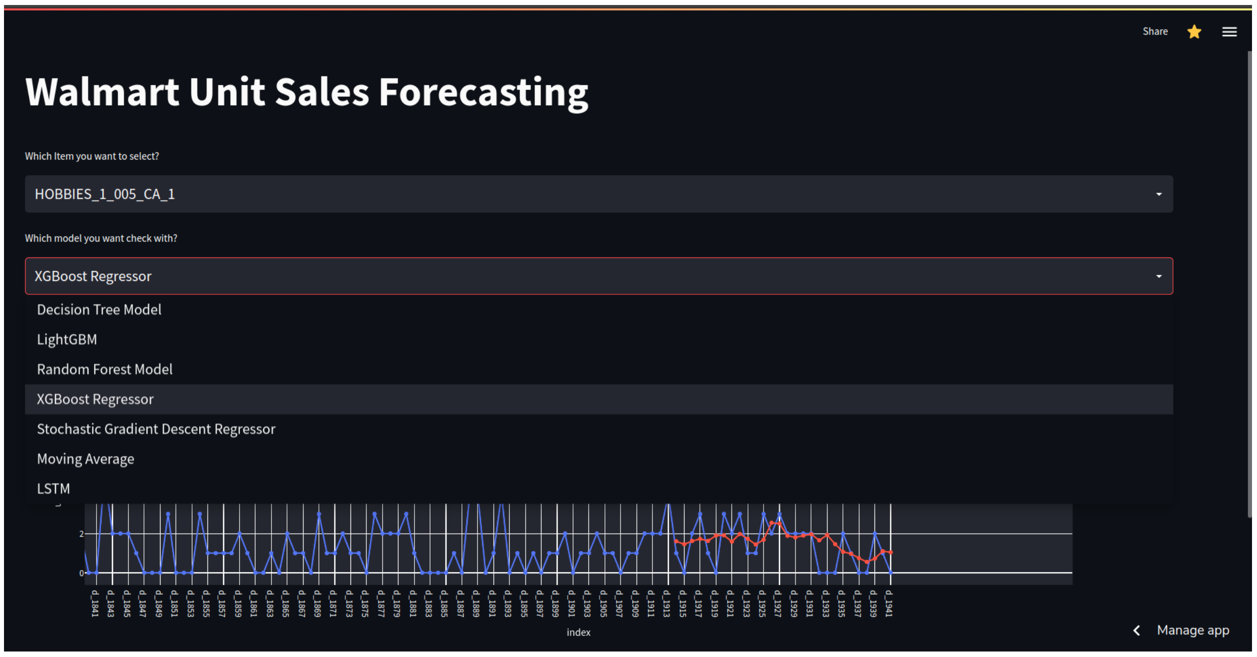
Deploying web servers can be accommodated within the Free Tier of any cloud provider, however, data-related tiers and the training/modelling phase will require high-spec VMs.

We have used the below libraries for deployment tasks:

- Terraform: Creating infrastructure as code, which makes them more maintainable, reproducible and keeping track of changes.
- Papermill: This is to run IPython Notebook with crontab.
- Streamlit: It is for ease of creating web applications, especially with Data Science libraries. It has integrations of many frameworks.

Scalability can be achieved using Auto Scaling group and load balancers for Web App, which will not have any downtime unless in maintenance. There is a limitation with the scalability of model training as the code doesn't support distributed computing. Dask can be used here for the next improvements. There are no latency issues as the data is already fetched to web servers. However, in production environments, it will have a separate database and warehousing in place. However, it will increase code complexity. It is the trade-off that we need to consider while designing.

Project screens:
Dashboarding Demo: kmistri-Walmart-sales-deployment-dashboard
Code: https://github.com/KishanMistri/Walmart_Sales_Deployment

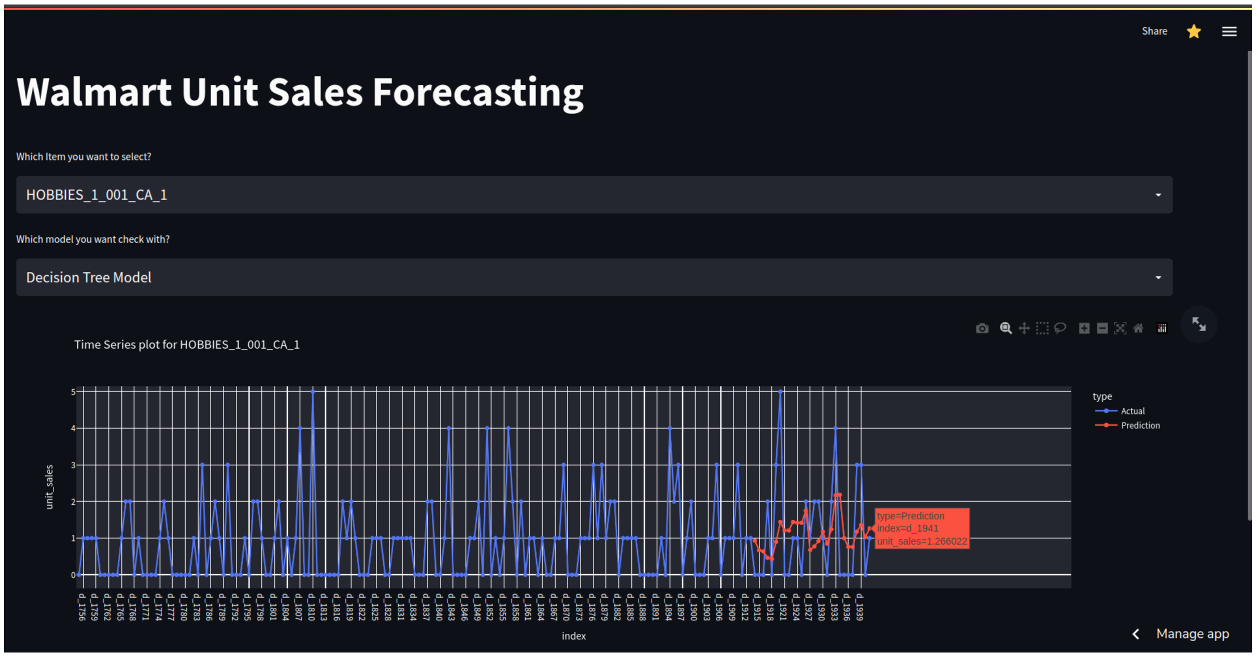

Walmart Unit Sales Forecasting

Which Item you want to select?

HOBBIES_1_005_CA_1

Which model you want check with?

XGBoost Regressor



Share

Walmart Unit Sales Forecasting

Which Item you want to select?

HOBBIES_1_001_CA_1

Which model you want check with?

Decision Tree Model

Time Series plot for HOBBIES_1_001_CA_1