Project Report on Walmart sales prediction

A project report submitted in fulfilment for the Diploma Degree in AI & ML
Under
Applied Roots with University of Hyderabad

M5 Forecasting -Accuracy
Estimate the Unit Sales of Walmart Retail Goods

Project submitted by

Manaswini Palutla

Under the Guidance of

Mentor: Saugata Paul

Contents

Abstract

A keyway to increase a retail company's profit is by retaining its customer base and a healthy cost structure. Maintaining the customer base requires knowing the behaviour patterns of the customers. The cost structure is more related to proper inventory management. Both these factors are directly related to demand. However, today's customer preferences and expectations are more complex than ever. Retailers are finding it hard to make decisions related to fetching the right inventory to the right location. From grocery to apparel, all retailers rely on forecasts in their demand planning.

This challenge comes under the concept of retail demand forecasting. Demand forecasting in retail is fundamentally the process of developing an estimate of the future customer demand. It includes the analysis of numerous internal and external variables that impact demand - from seasonality to promotions, inventory levels to market trends.

This thesis will help to identify the critical features that influence sales and an experiment is performed to find the best suitable machine learning algorithm/model for sales forecasting .

Introduction

Walmart Inc. is an American multinational retail corporate that operates a chain of hypermarkets, discount department stores, and grocery stores. We have hierarchical sales data from Walmart, the world's largest company by revenue. The data, covers stores in three US States (California, Texas, and Wisconsin) and includes item level, department, product categories, and store details. We are also provided with data such as price, promotions, day of the week, and special events to know the impact of these factors on sales.

Objective: To forecast daily sales for the next 28 days using the historical sales data.

Accurate forecasting helps a retailer in below ways:

- Make informed buying decisions
- Minimize wastage by optimizing inventory
- Make better pricing decisions
- Improve customer experience
- Fine tuning logistics planning
- Designing relevant marketing campaigns

The above factors would directly help in maximizing the profits. Forecasting demand successfully helps retailers, where customers are demanding, competitors are fierce, and supply chains are complex. Forecasts at all levels of granularity - hourly, daily, weekly, or monthly – can be extremely valuable for businesses that aim to meet customer demand, gain competitive advantage, and increase profits.

Mapping to an ML problem:

The model needs to generate future predictions based on past events. So, this problem comes under the concept of time series forecasting. The goal of forecasting is to project the underlying trend or pattern of the time series into the future as the most likely values for the data.

Since we need point estimates of demand of various products at various levels, it can be solved using a Machine Learning Regression model as input variables are generated features and items sold on a particular date as output variable which belongs to a real number.

I would like to experiment with different types of machine learning models and choose the one with best performance.

Source of the data:

The dataset is provided by Walmart in Kaggle website. The dataset involves the unit sales of 3,049 products. The M5 dataset, generously made available by Walmart, involves the unit sales of various products sold in the USA, organized in the form of grouped time series. The historical data range from 2011-01-29 to 2016-06-19. Thus, the products have a (maximum) selling history of 1,941 days.

More specifically, the dataset involves the unit sales of 3,049 products, classified in 3 product categories (Hobbies, Foods, and Household) and 7 product departments, in which the above-mentioned categories are disaggregated. The products are sold across ten stores, located in three States (CA, TX, and WI).

The dataset consists of 3 files:

- calendar.csv
- sales_train_evaluation.csv
- sell_prices.csv

Metrics

Business Metrics:

- Accuracy – The accuracy of the model is significant. The forecast should be as close as possible to the actual market demand so that required quantities could be made available for the market. Inaccurate forecast may cost huge to the firm. It may create over or under production thereby effecting the profit. The forecasts made should also be accurate across all hierarchical and granularity levels.
- Interpretable – An interpretable model is preferred as it gives an insight into why specific product sales are predicted to increase or decrease based on the various factors like geography, special events, day of the week etc.
- Low-latency – There is no requirement for low – latency. As long as the model takes reasonable amount of time, there wouldn't be any issue.

Performance metrics:

Many regression models rely on distance metrics to determine the convergence to the best result. Even the definition of a "best" result needs to be explained quantitatively by some metric. To measure the performance of the model I have used RMSE – Root Mean Squared Error. It's the square root of the average of squared

differences between prediction and actual observation. RMSE is calculated as below:

$$\boldsymbol{RMSE} = \sqrt{\sum_{i=1}^{n} \frac{(y_t - \hat{y_t})^2}{n}}$$

Where, Y_t - Actual future value of the examined time series at point t
$\widehat{Y_t}$ - the generated forecast
n - the length of the training sample.

Code implementation:

```
import numpy as np
def calculate_rmse(pred,actual):

    mse = np.square(np.subtract(actual,pred)).mean()
    rmse = np.sqrt(mse)

    return rmse
```

✓ 0.2s

Requirements

Hardware Requirements

- Computer system
- 16 GB Ram
- Intel i3 and above processor
- Server Overhead
- Internet connectivity

Software Requirements

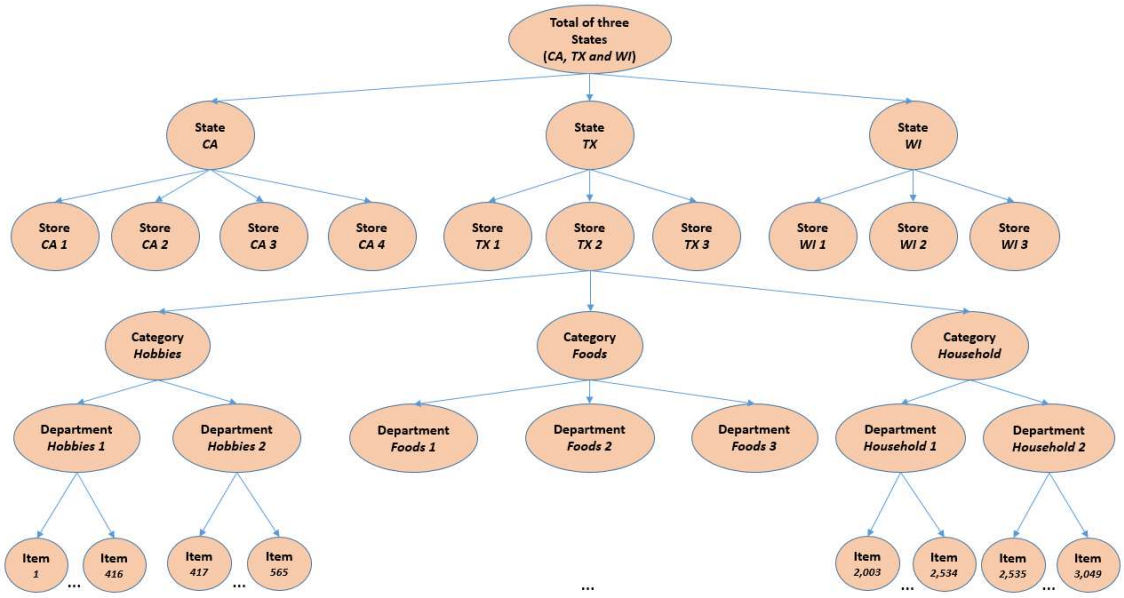
- Notepad++ or any Code Editor
- Anaconda Navigator , Jupyter Notebook
- Flask
- MS-Excel
- Python 3
- Python libraries like pandas, NumPy etc.
- Scikit-learn
- Client environment may be Windows or Linux
- Web Browser

Dataset Description

The dataset is provided by Walmart in Kaggle website. The dataset involves the unit sales of 3,049 products. The M5 dataset, generously made available by Walmart, involves the unit sales of various products sold in the USA, organized in the form of grouped time series. The historical data range from 2011-01-29 to 2016-06-19. Thus, the products have a (maximum) selling history of 1,941 days.

More specifically, the dataset involves the unit sales of 3,049 products, classified in 3 product categories (Hobbies, Foods, and Household) and 7 product departments, in which the above-mentioned categories are disaggregated. The products are sold across ten stores, located in three States (CA, TX, and WI).

Overview of data organization:



There are four csv files provided by Kaggle.

Calendar.csv:

It contains details on dates the projects have been sold. It has information regarding any special events occurred on that particular date. The table has details of whether SNAP purchase was allowed on given date or not indicated by three different columns one for each state using binary variable.

	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
0	2011-01-29	11101	Saturday	1	1	2011	d_1	NaN	NaN	NaN	NaN	0	0	0
1	2011-01-30	11101	Sunday	2	1	2011	d_2	NaN	NaN	NaN	NaN	0	0	0
2	2011-01-31	11101	Monday	3	1	2011	d_3	NaN	NaN	NaN	NaN	0	0	0

Description of columns:

- date: The date in a 'YYYY-MM-DD' format.
- wm_yr_wk: The id of the week the date belongs to.
- weekday: The day of the week of the date
- wday: The id of the weekday, starting from Saturday. Saturday --> 1 , sunday --> 2 and so on
- 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.

There are 14 columns in calendar table. The columns event_name_1,event_type_1 have data in 162 rows and event_name_2,event_type_2 have data in 5 rows and the values in rest of the rows are NULL. Other columns do not have any NULL values. We have replaced the NULL values with the value - 'no_event'.

There are 1969 rows in calendar dataframe where 1913 rows correspond to 29th Jan 2011 to 24th April 2016. We need to predict the sales of next 28 days i.e., 25th April 2016 to 22nd May 2016. The rest of the rows need not be used as they are specific to M5 competition held on Kaggle. Since this is a time series forecasting problem, we can use the 'date' column for splitting train, CV and test datasets

Sales_train_evaluation.csv:

This table contains the historical daily unit sales data per product and store. It has information details regarding the ID of the item, the category and department the item belongs to, the ID of store where the product is sold, the ID of the state the store is present in, the number of units sold per day for all 1941 days.

id	item_id	dept_id	cat_id	store_id	state_id	d_1	d_2	d_3	d_4	d_5
HOBBIES_	HOBBIES_	HOBBIES_	HOBBIES	CA_1	CA	0	0	0	0	0
HOBBIES_	HOBBIES_	HOBBIES_	HOBBIES	CA_1	CA	0	0	0	0	0
HOBBIES_	HOBBIES_	HOBBIES_	HOBBIES	CA_1	CA	0	0	0	0	0
HOBBIES_	HOBBIES_	HOBBIES_	HOBBIES	CA_1	CA	0	0	0	0	0
HOBBIES_	HOBBIES_	HOBBIES_	HOBBIES	CA_1	CA	0	0	0	0	0

Description of columns:

- item_id: The id of the product.
- dept_id: The id of the department the product belongs to.
- cat_id: The id of the category the product belongs to.
- store_id: The id of the store where the product is sold.
- 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.

There are 3049 items sold at 10 different stores 3049*10 = 30490 rows in sales_train_evaluation dataframe.

```
#Code to display the information on state,store,category,department and item columns
print('Items:',len(sales_train_eval['item_id'].unique()))
print('Department:',len(sales_train_eval['dept_id'].unique()))
print('Categories:',list(sales_train_eval['cat_id'].unique()))
print('Store:',list(sales_train_eval['store_id'].unique()))
print('State:',list(sales_train_eval['state_id'].unique()))
```

```
Items: 3049
Department: 7
Categories: ['HOBBIES', 'HOUSEHOLD', 'FOODS']
Store: ['CA_1', 'CA_2', 'CA_3', 'CA_4', 'TX_1', 'TX_2', 'TX_3', 'WI_1', 'WI_2', 'WI_3']
State: ['CA', 'TX', 'WI']
```

The data has sales information of stores in 3 states - California,Texas, Wiscosin. California has 4 stores and Texas; Wisconsin has 3 stores each.
There are 3 categories FOODS,HOUSEHOLD,HOBBIES. Each category is divided into departments.

Sell_prices.csv:

This table contains data of unique ID's of each store, unique ID's of each item, unique ID of the week and the price of the item during the week at the given store.

	store_id	item_id	wm_yr_wk	sell_price
0	CA_1	HOBBIES_1_001	11325	9.58
1	CA_1	HOBBIES_1_001	11326	9.58
2	CA_1	HOBBIES_1_001	11327	8.26
3	CA_1	HOBBIES_1_001	11328	8.26
4	CA_1	HOBBIES_1_001	11329	8.26

Description of columns:

- store_id: The id of the store where the product is sold.
- item_id: The id of the product.
- wm_yr_wk: The id of the week.
- sell_price: The average price of the product for the given week at the given store

There are 6841121 rows in the dataframe where each row corresponds to the average price of an item at a store for the given week. There are 4 columns with no NULL or Nan values.

Data Processing and Visualization

Before we start with the analysis it is better to have all of the required data in a single data frame. For this purpose, we would be merging calendar data with sales_train_eval dataset. We first pivot sales_train_eval data using pandas.melt() function. Pivoting is used to reshape a dataframe by converting the mentioned columns as data in rows. After pivoting, we merge with calendar data so that each column d_1 to d_1941 in sales_train_eval corresponds to the date in calendar data.

Code:

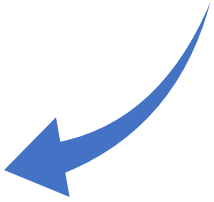
```
#dataframe is pivoted to have all the sales data under a single column
sales_final=sales_train_eval.melt(id_vars=['id', 'item_id', 'dept_id', 'cat_id',
                                           'store_id', 'state_id'], var_name='d',value_name='sales')
```

Pivoting sales_train table:

	id	item_id	dept_id	cat_id	store_id	state_id	d_1	d_2	d_3	d_4	...	d_1932	d_1933	d_1934	d_1935
0	HOBBIES_1_001_CA_1_evaluation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	2	4	0	0
1	HOBBIES_1_002_CA_1_evaluation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	0	1	2	1
2	HOBBIES_1_003_CA_1_evaluation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	1	0	2	0
3	HOBBIES_1_004_CA_1_evaluation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	1	1	0	4
4	HOBBIES_1_005_CA_1_evaluation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	0	0	0	2

5 rows × 1947 columns

	id	item_id	dept_id	cat_id	store_id	state_id	d	sales
0	HOBBIES_1_001_CA_1_evaluation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0
1	HOBBIES_1_002_CA_1_evaluation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0
2	HOBBIES_1_003_CA_1_evaluation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0
3	HOBBIES_1_004_CA_1_evaluation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0
4	HOBBIES_1_005_CA_1_evaluation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0



Merge on sales_train table and calendar dataframe:

Code:

```
#merging sales data with calendar data to plot total sales per day(in terms of date)
sales_final=sales_final.merge(calendar,on='d',how='left')
```

After merging with calendar dataframe:

id	item_id	dept_id	cat_id	store_id	state_id	d	sales	date	wm_yr_wk	weekday	wday	month	year	event_name_1	event_type_1	event_name_2	event_type_2	snap_CA	snap_TX	snap_WI
HOBBIES_1_001_CA_1_evaluation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0	2011-01-29	11101	Saturday	1	1	2011	no_event	no_event	no_event	no_event	0	0	0
HOBBIES_1_002_CA_1_evaluation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0	2011-01-29	11101	Saturday	1	1	2011	no_event	no_event	no_event	no_event	0	0	0
HOBBIES_1_003_CA_1_evaluation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0	2011-01-29	11101	Saturday	1	1	2011	no_event	no_event	no_event	no_event	0	0	0
HOBBIES_1_004_CA_1_evaluation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0	2011-01-29	11101	Saturday	1	1	2011	no_event	no_event	no_event	no_event	0	0	0
HOBBIES_1_005_CA_1_evaluation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0	2011-01-29	11101	Saturday	1	1	2011	no_event	no_event	no_event	no_event	0	0	0

There are 58 million rows in the above dataframe after merging. We sample 20 million rows for EDA to avoid memory issues.

```
#Sample 20 million rows out of 56 million to perform EDA
sales_final = sales_final.sample(n=20000000)
```

Merging sell_prices.csv with the above table to include price details.

```
#Merging with price dataframe
sales_final=sales_final.merge(price_data,on=['wm_yr_wk','item_id','store_id'],how='left')
sales_final.head()
```

There are no Nan or NULL values in sell_prices dataframe. But after we merged the sell_prices data frame with sales_final created above which contains calendar and sales_train_evaluation data, Nan values are present in dataframe.

This might be because the product is missing from the stores in that week at the given store. Out of 20 million rows which we sampled; 4 million rows have Nan values in ‘sell_price’ column.

```
#Sample 20 million rows out of 56 million to perform EDA
sales_final = sales_final.sample(n=20000000)
```

```
sales_final.isnull().values.any()
```

```
False
```

```
#Merging with price dataframe
sales_final=sales_final.merge(price_data,on=['wm_yr_wk','item_id','store_id'],how='left')
sales_final.isnull().values.any()
```

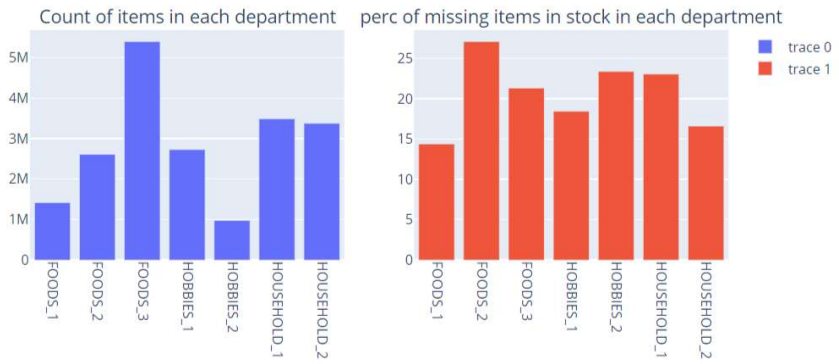
```
True
```

```
sales_final['sell_price'].isna().sum()
```

```
4155676
```

Analysis on missing items in stock:

Departments FOODS_3 and HOUSEHOLD_1 have highest number of products. Highest number of missing items belongs to FOODS_2 department. HOUSEHOLD and HOBBIES categories almost have same percentage of missing items in stock.



Items in CA_2 store has price value missing which means more items are not in stock in CA_2 store. Other stores have approximately same number of items not present in stock.

→

The description of the column ‘sell_price’ shows that the data has outliers. There is a huge gap between 75th percentile and maximum value.

```
print(round(sales_final['sell_price'].describe(),2))
```

```
count    20000000.00
mean             NaN
std              0.00
min              0.01
25%              2.18
50%              3.42
75%              5.84
max            107.31
Name: sell_price, dtype: float64
```

Replacing Nan values with median of the product's category since there are outliers in dataset, we would consider median for imputation.

We have replaced based on product's category since there is a difference in median of price value between categories.

```
#Below code displays the data available for each state and different stores present in each state
grp_cat  = sales_final.groupby(['cat_id']).agg({'sell_price':['mean','min','max','median']})

print(grp_cat)
```

```
          sell_price
                mean       min         max    median
cat_id
FOODS       3.250000  0.010002   19.484375  2.679688
HOBBIES     5.332031  0.010002   30.984375  3.970703
HOUSEHOLD   5.464844  0.010002  107.312500  4.941406
```

```
#Median imputation
sales_final['sell_price'].fillna(sales_final.groupby(['cat_id','item_id'])['sell_price'].transform('median'),
                                 inplace=True)
```

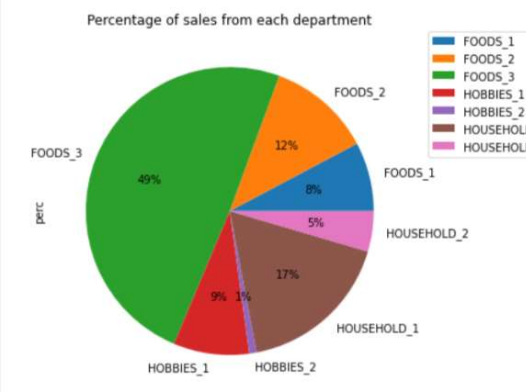
Added a new column 'revenue' for further analysis since the end goal of any store is to increase the revenue generated from a given product.

```
sales_final['revenue'] = sales_final['sell_price'] * sales_final['sales']
```

Exploratory Data Analysis

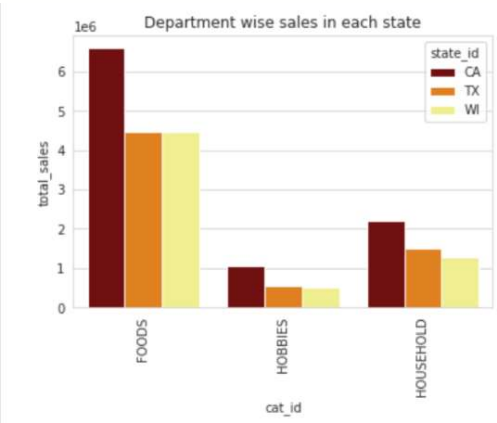
Basic analysis:

Sales across different departments and stores



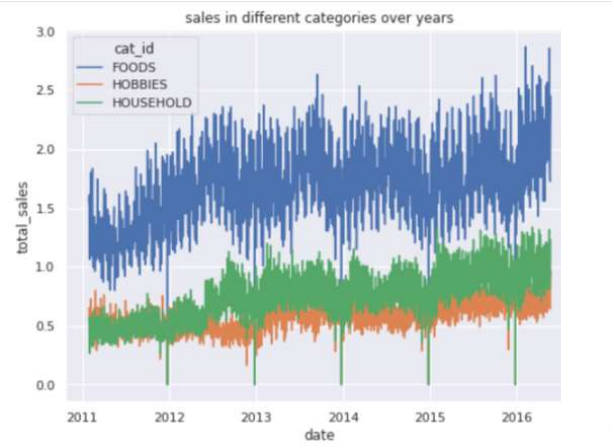
- Most of the sales are from items in FOODS_3 department
- 70% of the sales are from items in FOODS category.

→



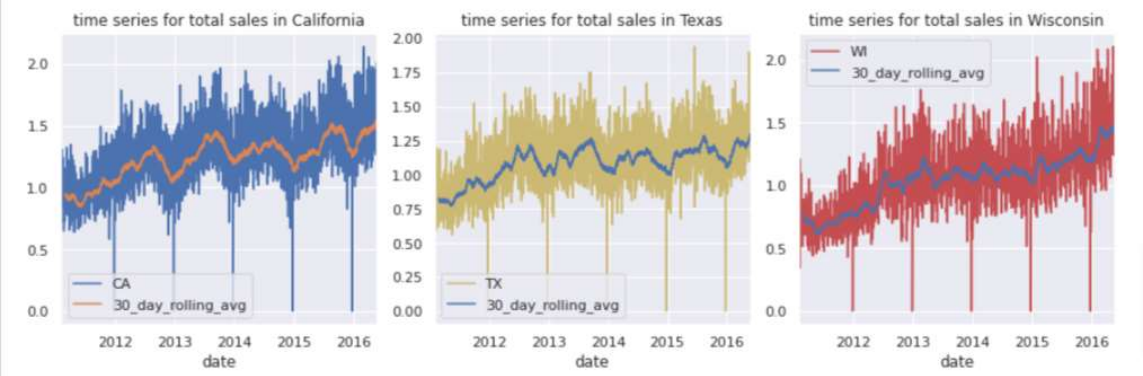
- Sales in FOODS department are highest in all the states.
- Items in hobbies have least number of sales across all states.
- California has highest sales in every department due to a greater number of stores.
- Sales in Texas and Wisconsin almost match across all departments.

→



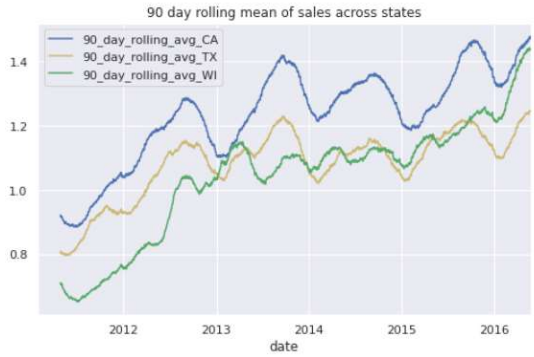
- Sales have a smooth upward trend through the years for items in Foods and Household category.
- Sales have drastically increased between 2011 to 2012.
- There is a cyclic pattern in sales within a year.
- Sales have increased marginally for items in HOBBIES.

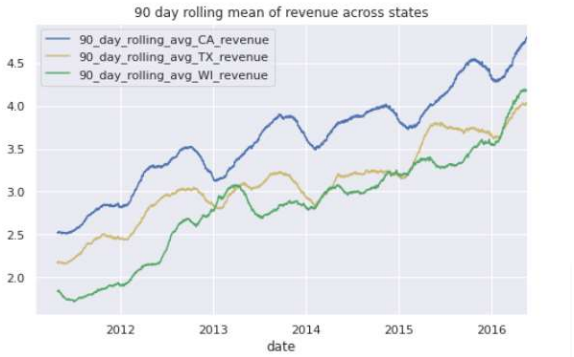
→



- There is a steady growth in sales in CA state as compared to TX and WI.
- The sales in Texas have a rough linear trend.
- The sales in WI were less in earlier years as compared to TX but have picked up fast by mid-2013.
- The total sales are almost zero on a particular day at the end of the year in all states.

→





Sales:

- CA sold the greatest number of items during the entire 5-year period.
- WI overtook TX in terms of the number of the items sold around July in 2015
- There's some seasonality at play here with sales reaching a local max every 300-400 days.
- The mean value has an upward linear trend. The sales are oscillating at higher frequency every few intervals.

Revenue:

- The revenue from CA state is significantly higher than other two states.
- Even though WI overtook TX in sales the revenue difference from both the states is minimal.
- The sales in TX have a rough linear trend but revenue has a slight upward trend.

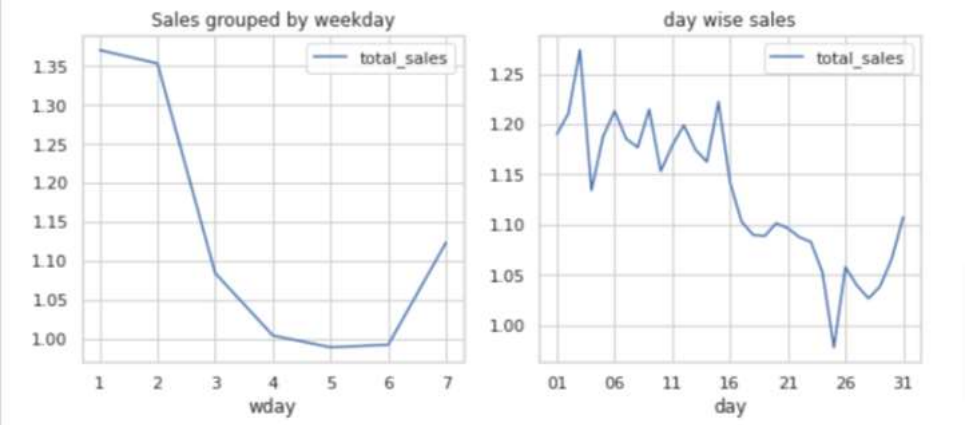
→

- CA_3 store has highest sales followed by CA_1
- CA_4 has least amount of sales

→

- Stores WI_1 and WI_2 have a drastic increase in sales in the year 2012. WI_3 store has a dip after 2012.
- The CA stores are well separated in terms of sales. The sales in CA_1 and CA_3 almost have same pattern. CA_2 store has dip in sales in the year 2014 but recovered quickly by 2015. Sales in CA_3 is always high as compared to other stores.
- The sales of stores in Texas are quite close to each other. Around beginning of 2014 store TX_3 sales have increased.

→



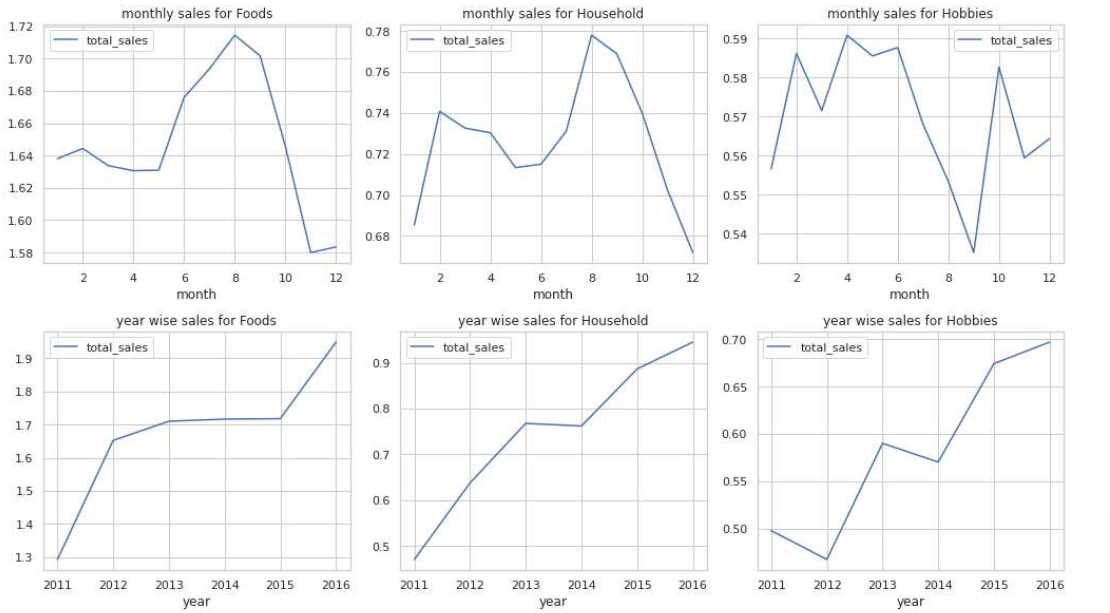
Sales as per weekday:

- The sales are highest on Saturday followed by sunday and Friday.
- There is a sudden dip in sales after sunday which continued till Thursday

Sales as per day of month:

- The sales are high for first half of month as compared to the second half.
- The sales are least around 20th to 25th days and have slowly picked up by 28th.
- The reason of this behaviour might be because of general salary cycle.

→



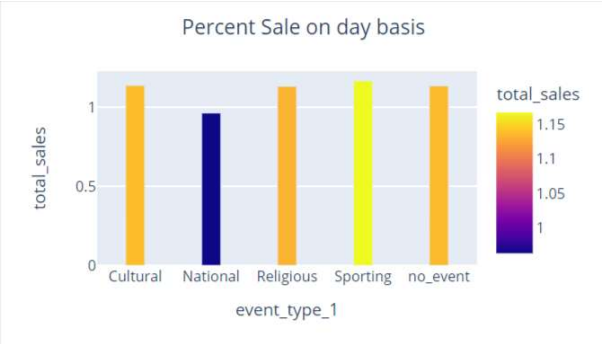
Month wise sales

- The sales for items in foods and household have steadily increased from May and are high in august. This might be because of summer vacation in these months at US.
- The sales in Hobbies have decreased the same time the sales in Foods and Hobbies have increased i.e., around May to August.

Year wise sales

- There is a steady increase in sales across years for all categories
- There is no evident growth in sales between the years 2013 to 2014.The sales of items in Hobbies department has gone down from 2013 to 2014. The sale of items in Household and Foods have a linear trend between these years.

→

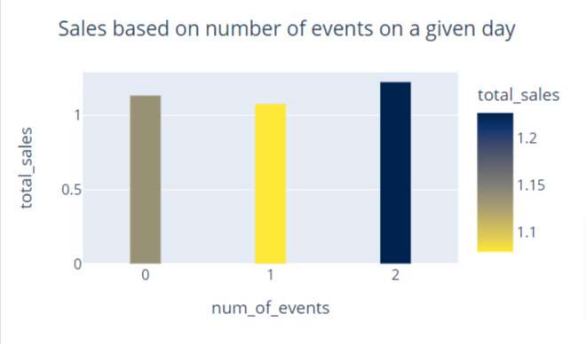


- Sales are slightly higher for sporting event than on days with no events.
- Religious events and cultural events do not seem to have much impact on sales.
- National event days have least number of sales.

→

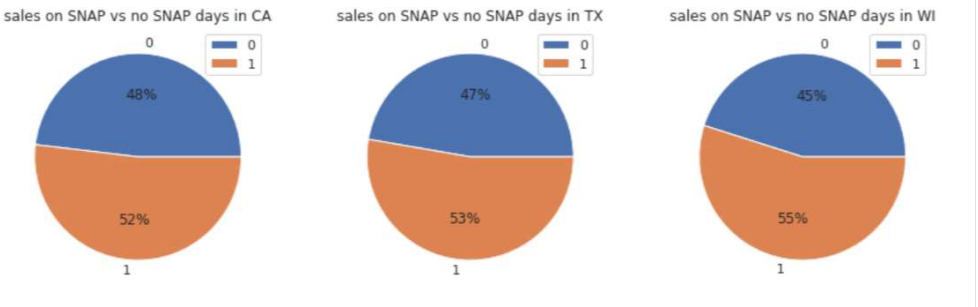
We have created a new column – 'num_of_events' which depicts the number of events on a given day.

Average sales are highest when there are 2 events.



→

In the United States, SNAP - Supplemental Nutrition Assistance Program, formerly known as the Food Stamp Program, is a federal program that provides food-purchasing assistance for low and no-income people.



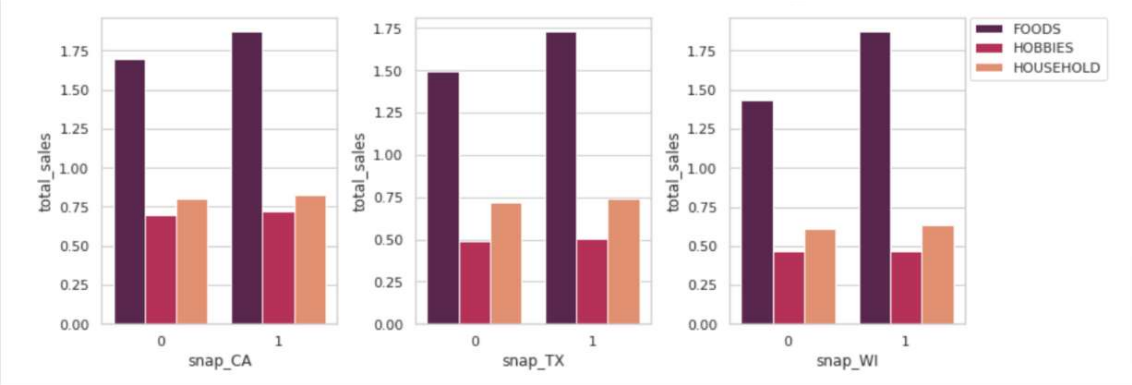
- Sales on SNAP days are slightly higher than non-SNAP days
- SNAP purchases have higher impact in WI state than TX and CA

→

```
SNAP days in a month in California state ['01', '02', '03', '04', '05', '06', '07', '08', '09', '10']
SNAP days in a month in Texas state ['01', '03', '05', '06', '07', '09', '11', '12', '13', '15']
SNAP days in a month in Wisconsin state ['02', '03', '05', '06', '08', '09', '11', '12', '14', '15']
```

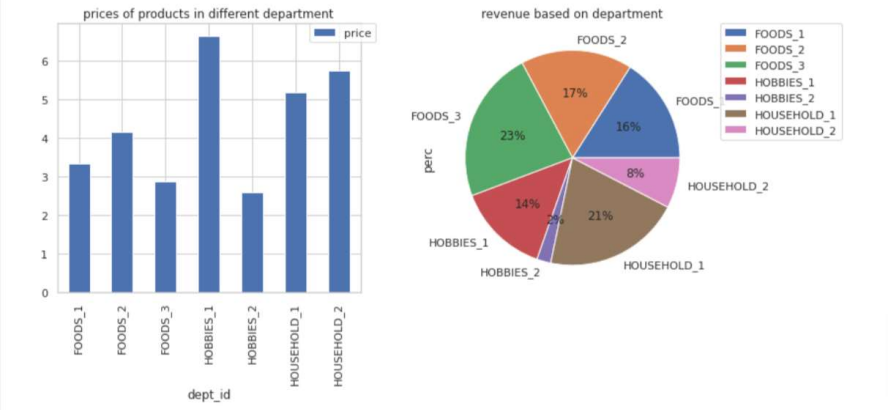
- SNAP days in all three states are present only in first 15 days of the month. This might be one of the reasons for higher sales during first half of month.
- Only 10 particular days in first 15 days are SNAP days.

→



- FOODS item sales are high on SNAP days as compared to hobbies and household. This is because SNAP is applicable only on purchase of food items.
- FOODS items sales are impacted more in Wisconsin state as compared to California and Texas.

→



- Even though the price of items in FOODS category are less, more than 50% of the revenue is from FOODS department.
- Household items have high price values as compared to items in FOODS category and HOBBIES_2. They contribute around 30% for the revenue
- Even though HOBBIES_1 has less amount of sales, the revenue is comparatively higher this might be because of cost of items in HOBBIES_1.

Basic Modelling – First cut solution

Classical time series models

❖ Simple moving average

The moving average model is probably the naivest approach to time series modelling. This model simply states that the next observation is the mean of all past observations. Although simple, this model might be surprisingly good and it represents a good starting point.

$$\hat{y_{t+1}} = \frac{1}{m} \sum_{t-m}^{t} y_i$$ where m is the window size.

Code:

```
def SMA(df, forecast, wind_size):

    preds = []
    for i in range(forecast):
        if i == 0:
            preds.append(np.mean(df[df.columns[-wind_size:]].values, axis=1))
        if i < forecast and i > 0:
            pred = 0.5 * (np.mean(df[df.columns[-wind_size + i:]].values, axis=1) + \
                          np.mean(preds[:i], axis=0))
            preds.append(0.5 * (np.mean(df[df.columns[-wind_size + i:]].values, axis=1) + \
                          np.mean(preds[:i], axis=0)))

    return preds
```

```
df = pd.DataFrame()
predictions = SMA(X_train, forecast, 28)
for d, i in enumerate(range(1914, 1942)):
    df['F_' + str(i)] = predictions[d]

print("RMSE of SMA is:", math.sqrt(mse(X_test,df)))
```

```
RMSE of SMA is: 2.233064272884162
```

With window size = 28, the RMSE value of SMA model is 2.23.

Using the average method, all future forecasts are equal to a simple average of the observed data. Hence, the average method assumes that all observations are of equal importance and gives them equal weights when generating forecasts.

❖ Exponential weighted moving average model

Forecasts produced using exponential smoothing methods are weighted averages of past observations, with the weights decaying exponentially as the observations get

older. In other words, the more recent the observation the higher the associated weight.

It may be sensible to attach larger weights to more recent observations than to observations from the distant past. This is exactly the concept behind simple exponential smoothing. Forecasts are calculated using weighted averages, where the weights decrease exponentially as observations come from further in the past — the smallest weights are associated with the oldest observations

$\hat{y_{t+1}} = \alpha y_t + (1-\propto)\hat{y_t}$ where $0 \leq \alpha \leq 1$

Code:

```
def EWMA(X_train,X_test,forecast,alpha):

    preds = []
    alpha = alpha
    for i in range(forecast):
        if i == 0:
            pred = (X_train.iloc[:,-1])
            preds.append(pred)
        if i < forecast and i > 0:
            pred = (alpha * X_test.iloc[:,i-1]+ (1-alpha)*preds[i-1])
            preds.append(pred)
    return preds
```

✓ 0.3s

RMSE: 2.16

Comparing performance of all models:

```
Performance of models without any featurization
+----+-----------------------------------+------------+
| No |                Model              | RMSE SCORE |
+----+-----------------------------------+------------+
| 1  |        Simple Moving Average      |    2.23    |
| 2  | Exponential Weighted Moving Average |    2.16    |
| 3  |          Linear regression        |   3.579    |
| 4  |            Decision Tree          |   3.293    |
| 5  |            Random Forest          |   2.711    |
+----+-----------------------------------+------------+
```

The RMSE is quite high for all the models. The classical time series models have performed better than machine learning models and even the ensemble model. One thing to note here is that the machine learning models are implemented without any sort of featurization and hyper parameter tuning. This might be one of the reasons of poor performance in machine learning models.

Generating other meaningful features which seemed to be useful in EDA can help us to improve the performance of models. We can also use time series featurization techniques and generate new features.

Feature Engineering

As discussed in data preparation chapter, the three datasets calendar.csv, sales_train_evaluation.csv and sell_prices.csv are combined using operations like pivot and merge to form a single dataframe.

❖ Reducing Memory used by dataframe:

The final dataframe has 58 million rows. To avoid any memory related issues, we need to reduce the memory used by the dataframe. To downcast the tables, I have used reduce function from downcast library. This package is used the reduce the dataframe size without affecting the values. It finds the max and min value in dataframe columns, based on these values it down casts the datatypes of that columns.

```
cal_bfr = calendar.memory_usage(deep=True).sum()
calendar=reduce(calendar)
cal_aftr = calendar.memory_usage(deep=True).sum()
sales_bfr = sales_train_eval.memory_usage(deep=True).sum()
sales_train_eval=reduce(sales_train_eval)
sales_aftr = sales_train_eval.memory_usage(deep=True).sum()
price_bfr = sell_price.memory_usage(deep=True).sum()
sell_price=reduce(sell_price)
price_aftr = sell_price.memory_usage(deep=True).sum()

print("memory usage of calendar dataframe reduced by",cal_aftr/cal_bfr * 100,"percent")
print("memory usage of sales dataframe reduced by",sales_aftr/sales_bfr * 100,"percent")
print("memory usage of price dataframe reduced by",price_aftr/price_bfr * 100,"percent")
```

✓ 3m 40.1s

```
memory usage of calendar dataframe reduced by 25.33067651400899 percent
memory usage of sales dataframe reduced by 21.31192592896515 percent
memory usage of price dataframe reduced by 4.797418193536785 percent
```

❖ Handling Missing values:

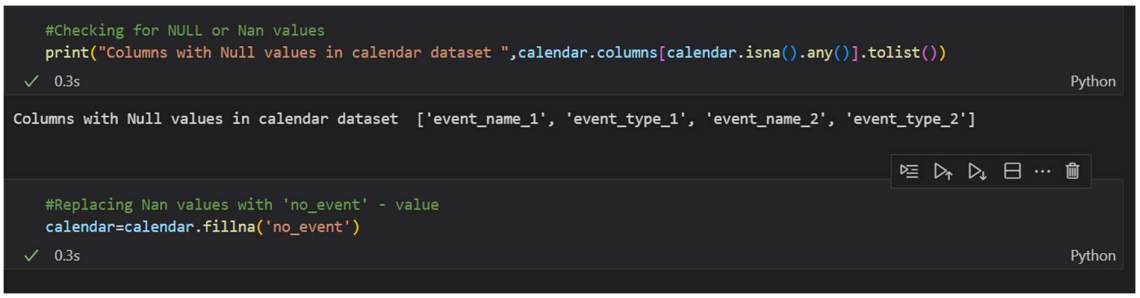
The problem of missing value is quite common in many real-life datasets. Missing value can bias the results of the machine learning models and/or reduce the

accuracy of the model. Many machine learning algorithms fail if the dataset contains missing values so it is important to handle the missing values appropriately.

There are missing values in below columns of calendar table

- event_name_1
- event_type_1
- event_name_2
- event_type_2

These columns are used to represent if there is any event on the given day. Before merging sales_train_evaluation and calendar tables, the Nan or NULL values in these columns are replaced with ‘no_event’.



→

The column ‘sell_price’ has Nan or NULL values in the final created dataframe. One thing to note here is that there are no missing values in sell_prices table, however, the column has missing values only after the dataframe is merged. A NULL value in ‘sell_price’ column could indicate the absence of the given item in stock at a given store on a particular day.

The Nan value in sell_price column is replaced by imputing with median value. The median value is calculated by grouping the ‘item_id’ and ‘store_id’.

❖ Label Encoding:

There are many categorical features in the dataframe. To make these features more understandable for the model, it is often labeled using Label encoding. Label Encoding is a technique of converting the labels into numeric form so that it could be ingested to a machine learning model. It is an important step in data preprocessing for supervised learning techniques.

We have label encoded below categorical features using LabelEncoder class in sklearn.

- id
- item_id
- dept_id
- cat_id
- store_id
- state_id
- event_name_1
- event_type_1
- event_name_2
- event_type_2

```
column = ['id','item_id', 'dept_id', 'cat_id', 'store_id', 'state_id', 'event_name_1', 'event_type_1', 'event_name_2', 'event_type_2']
for feature in column:
    encoder = LabelEncoder()
    sales_final[feature] = encoder.fit_transform(sales_final[feature])
```

✓ 33.4s

❖ Time-based features:

Lag features: The lag features are basically the target variable but shifted with a period of time, it is used to know the behavior of our target value in the past, maybe a day before, a week or a month. Added lag of 1,7,28 and 30 days since farther out in the time horizon, the less accurate a forecast is.

```
#Adding lag features
lags = [1,7,28,30]
for lag in tqdm(lags):
    sales_final['lag_'+str(lag)] = sales_final.groupby(['id'],as_index=False)['sales'].shift(lag).astype(np.float16)
```

✓ 7.1s

```
100%|██████████| 4/4 [00:06<00:00,  1.70s/it]
```

Rolling window features: Rolling window calculates a statistic for a fixed contiguous block of prior observation. It is more like applying your function for a fixed set of continuous prior observations repeatedly to the sub-data in your full data set.

```
#Adding rolling window features
window = [7,14,28,35,42]
for i in tqdm(window):
    func = lambda x: x.rolling(i).median()
    sales_final['rolling_median_'+str(i)] = sales_final.groupby(['id'],as_index=False)['sales'].transform(func)
```

✓ 2m 16.4s

```
100%|██████████| 5/5 [02:16<00:00, 27.22s/it]
```

```
sales_final['rolling_sales_mean'] = sales_final.groupby(['item_id','dept_id',
                'cat_id','store_id','state_id'])['sales'].transform(lambda x: x.rolling(window=7).mean()).astype(np.float16)
```

✓ 23.4s

→

Featurization on columns in calendar table:

As sales are high on weekend, I have created a new column ‘is_Weekend’ which has value 1 for Saturday and Sunday and 0 for other days.

Dropped ‘weekday’ as we have the same information in ‘wday’ column.

```
#Removing weekday column and adding is_weekend feature

sales_final["is_Weekend"]=sales_final['wday'].map(lambda x: 1 if x in [1,2] else 0)
sales_final=sales_final.drop("weekday",axis=1)
```

✓ 7.2s

As sales are high during first half of the month as compared to second half, I have added a new column ‘day_of_month’ which contains the value of the day in month.

```
#adding day of month column
sales_final['day_of_month'] =  sales_final['date'].dt.strftime("%d")
sales_final['day_of_month'] = sales_final['day_of_month'].astype('int')
```

✓ 1m 22.5s

SNAP days are almost common in all states so combined the columns SNAP_CA, SNAP_TX, SNAP_WI to a single column SNAP.

```
# Converting snap_CA,snap_WI,snap_TX into one feature named snap

sales_final.loc[sales_final['state_id'] == 'CA', 'snap'] = sales_final.loc[sales_final['state_id'] == 'CA']['snap_CA']
sales_final.loc[sales_final['state_id'] == 'TX', 'snap'] = sales_final.loc[sales_final['state_id'] == 'TX']['snap_TX']
sales_final.loc[sales_final['state_id'] == 'WI', 'snap'] = sales_final.loc[sales_final['state_id'] == 'WI']['snap_WI']
sales_final.drop(['snap_CA','snap_TX','snap_WI'],axis=1,inplace=True)
```

✓ 2.9s

The day column has days in the form of ‘d_i’ where i represents the day. So, removed the string ‘d_’ to represent the actual day number.

```
#Extract number from string
sales_final['d'] = sales_final['d'].str.extract(r"(\d+)").astype(np.int16)
```

Date value is captured using ‘day_of_month’, ’month’, ’year’ columns. Dropped columns ‘wm_yr_wk’ and ‘date’ of calendar dataframe.

```
#dropping wm_yr_wk column
sales_final.drop('wm_yr_wk',axis=1,inplace=True)
#dropping date column
sales_final = sales_final.drop('date',axis=1)
sales_final = sales_final.reset_index(drop=True)
```

Below is the final data frame created on which model will be trained and tested. There are 29 columns in the data frame and there are approximately 15.5 million rows of data.

```
sales_final.head()
```
✓ 0.3s

	id	item_id	dept_id	cat_id	store_id	state_id	d	sales	wday	month	...	lag_1	lag_7	lag_28	lag_30	rolling_median_7	rolling_median_14
0	14370	1437	3	1	0	0	1434	0	6	1	...	0.0	0.0	1.0	3.0	1.0	0.5
1	14380	1438	3	1	0	0	1434	0	6	1	...	0.0	0.0	0.0	0.0	0.0	0.0
2	14390	1439	3	1	0	0	1434	0	6	1	...	0.0	0.0	0.0	0.0	0.0	0.0
3	14400	1440	3	1	0	0	1434	2	6	1	...	0.0	0.0	1.0	1.0	2.0	1.0
4	14410	1441	3	1	0	0	1434	3	6	1	...	0.0	0.0	0.0	0.0	0.0	2.0

5 rows × 29 columns

→

Splitting the dataframe into train, valid and test datasets:

Due to memory constraints, I have considered data from 1st Jan 2015. It did not affect the performance of the model since recent data is more relevant than past data in time series problem.

Performed train, validation and test split based on temporal basis. The last 28 days i.e., days between 1914 to 1941 are considered as test dataset. Days between 1885 to 1913 are considered as validation dataset. The days before 1885 are considered as train dataset.

Splitting the dataframe into train and test datasets

Since this is a time series problem, dataset is split on temporal basis instead of random splitting using sklearn

```
train_set =  sales_final[(sales_final['d'] <= 1885)]
valid_set = sales_final[(sales_final['d'] > 1885) & (sales_final['d'] <= 1913)]
test_set = sales_final[(sales_final['d'] > 1913)]
```
[41]

```
y_train = X_train['sales']
y_cv = X_cv['sales']
y_test = X_test['sales']

# We are drpping the features which are not required.
X_train.drop(['sales'],axis = 1,inplace = True)
X_cv.drop(['sales'],axis = 1,inplace = True)
X_test.drop(['sales'],axis = 1,inplace = True)

print(X_train.shape,y_train.shape)
print(X_cv.shape,y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(13781480, 23) (13781480,)
(853720, 23) (853720,)
(853720, 23) (853720,)
```

Modelling

Linear Models

Linear Regression is the basic form of regression analysis. It assumes that there is a linear relationship between the dependent variable and the predictor(s). In regression, we try to calculate the best fit line, which describes the relationship between the predictors and predictive/dependent variables.

Linear regression is also called as Ordinary Least Squares and Linear least Squares. The optimization problem in linear regression is

$(W^*, W_0) = \text{argmin}_{W,W0} \sum_{i=1}^{n} \{y_i - (W^T x_i + w_0)\}^2$

To avoid overfitting the data, we add regularization term to the above loss function.

- Lasso regression:

Linear regression with L1 regularization is called as Lasso regression. Lasso regression is what is called the Penalized regression method. Along with minimizing the Ordinary Least Squares, the model also tries to minimize the absolute sum of the coefficients. The LASSO imposes a constraint on the sum of the absolute values of the model parameters, where the sum has a specified constant as an upper bound. This constraint causes regression coefficients for some variables to shrink towards zero. This is the shrinkage process. The shrinkage process allows for better interpretation of the model and identifies the variables most strongly associated with the target corresponds variable.

The loss function in Lasso regression is

$(W^*, W0) = \text{argmin}_{W,W0} \sum_{i=1}^{n} \{y_i - (W^T x_i + w_0)\}^2 + \lambda ||W||$ where $0 \leq \lambda \leq 1$

$||W|| = |w|_1 + |w|_2 + |w|_3 + \cdots + |w|_n$

Here λ is the hyperparameter. Lambda is applied to the regression model to control the strength of the penalty. It is the penalty term that denotes the amount of shrinkage (or constraint) that will be implemented in the equation. Bias increases and variance decreases as lambda increases.

Code:

```
# Re-training the model with the best alpha value.
model1 = linear_model.Lasso(alpha=0.001)
model1.fit(X_train, y_train)
y_pred = model1.predict(X_test)
y_pred_train = model1.predict(X_train)
rmse = math.sqrt(mse(y_train,y_pred_train))
print(f"For alpha value 0.001, the train RMSE score is {rmse}")
rmse = math.sqrt(mse(y_test,y_pred))
print(f"For alpha value 0.001, the test RMSE score is {rmse}")
```

```
For alpha value 0.001, the train RMSE score is 1.8585271745634224
For alpha value 0.001, the test RMSE score is 1.811625828895387
```

The model performed better than classical time series models. Train error is higher than test error. There might be some bias in the model.

Ridge regression:

Linear regression with L2 regularization is also referred to as Ridge Regression. Similar to the lasso regression, ridge regression puts a similar constraint on the coefficients by introducing a penalty factor. However, while lasso regression takes the magnitude of the coefficients, ridge regression takes the square. L2 shrinks all the coefficient by the same proportions but eliminates none, while L1 can shrink some coefficients to zero, performing variable selection.

The loss function in Ridge regression is

$(W^*, W_0^*) = argmin_{W,W_0} \sum_{i=1}^{n} \{y_i - (W^T x_i + W_0)\}^2 + \lambda ||W||^2$ where $0 \leq \lambda \leq 1$

$||W||^2 = \sqrt{(w_1^2 + w_2^2 + \cdots + w_n^2)}$

The penalty term has the effect of therefore shrinking our coefficients towards 0. Not exactly 0 like lasso, but the higher the coefficient is the more the penalty is so therefore we want to be reducing the size of those coefficients. This is going to impose bias on the model but also reduce variance.

Code:

```
# Re-training our model with the best alpha value.
model2 = linear_model.Ridge(alpha=1)
model2.fit(X_train, y_train)
y_pred_train = model2.predict(X_train)
rmse = math.sqrt(mse(y_train,y_pred_train))
print(f"For alpha value 0.001, the train RMSE score is {rmse}")
y_pred = model2.predict(X_test)
rmse = math.sqrt(mse(y_test,y_pred))
print(f"For alpha value 0.001, the test RMSE score is {rmse}")
```

```
For alpha value 0.001, the train RMSE score is 1.8585033136584053
For alpha value 0.001, the test RMSE score is 1.8114981019749543
```

The model performed better than classical machine learning models. Similar to Lasso regression, ridge regression has train error higher than test error indicating bias in the model.

Tree based model

Unlike linear models, decision trees support non-linearity. Decision tree regressor observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Decision trees work by basically dividing the features space using axes parallel hyperplanes.

Decision Tree use loss functions that evaluate the split based on the purity of the resulting nodes. In regression, the function used is mean squared error for split.

Code:

```python
max_depth=176
min_samples_split=141
max_leaf_nodes = 329
model4= DecisionTreeRegressor(max_depth=max_depth,min_samples_split=min_samples_split,max_leaf_nodes=max_leaf_node
model4.fit(X_train,y_train)
y_pred_train = model4.predict(X_train)
rmse = math.sqrt(mse(y_train,y_pred_train))
print(f"With max_depth: 159, min_samples_split:275, max_leaf_nodes: 367 the train RMSE score is {rmse}")
y_pred = model4.predict(X_test)
rmse = math.sqrt(mse(y_test,y_pred))
print(f"With max_depth: 159, min_samples_split:275, max_leaf_nodes: 367 the test RMSE score is {rmse}")
```

Python

```
With max_depth: 159, min_samples_split:275, max_leaf_nodes: 367 the train RMSE score is 1.7767201875722742
With max_depth: 159, min_samples_split:275, max_leaf_nodes: 367 the test RMSE score is 1.776710057516806
```

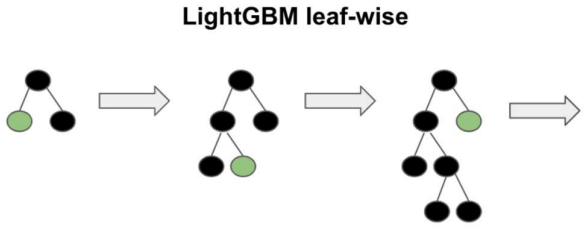
The error is less as compared to linear models. The train error and test error are approximately equal unlike linear models. This could be because decision trees by nature have low bias.

Ensemble model - Light GBM

GBDT is an ensemble model of decision trees which learns the decision trees by finding the best split points. Finding the best split points while learning a decision tree is supposed to be a time-consuming issue. Due to lack of computational resources, we have not trained on XGboost and CatBoost takes a lot of time and memory to train. This issue can be overcome by implementing Light GBM.

Light GBM is a gradient boosting framework that uses tree-based learning algorithm. Light GBM grows tree vertically while other algorithm grows trees horizontally meaning that Light GBM grows tree leaf-wise while other algorithms grow level-wise.

The size of data is increasing day by day and it is becoming difficult for traditional data science algorithms to give faster results. Light GBM is prefixed as ‘Light’ because of its high speed. Light GBM can handle the large size of data and takes lower memory to run.



Code:

```
#after checking mutliple parameters below values have given best results
learn_rate = 0.034
num_leaves = 78
min_data_in_leaf = 67
model6 = LGBMRegressor(learning_rate=learn_rate ,num_leaves=num_leaves , min_data_in_leaf=min_data_in_leaf)
model6.fit(X_train,y_train)
y_pred_train = model6.predict(X_train)
y_pred = model6.predict(X_test)
rmse = math.sqrt(mse(y_train,y_pred_train))
print(f"Train error with learn_rate: {learn_rate}, num_leaves:{num_leaves}, min_data_in_leaf: {min_data_in_leaf} =
rmse = math.sqrt(mse(y_test,y_pred))
print(f"Test error with learn_rate: {learn_rate}, num_leaves:{num_leaves}, min_data_in_leaf: {min_data_in_leaf} =
print('*'*100)
```

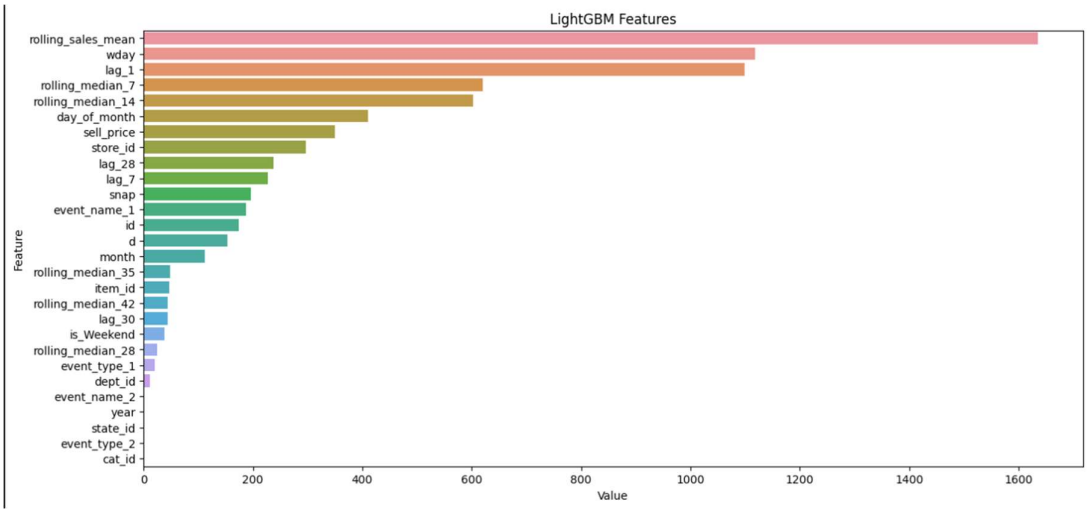
✓ 1m 16.4s Python

```
[LightGBM] [Warning] min_data_in_leaf is set=67, min_child_samples=20 will be ignored. Current value:
min_data_in_leaf=67
Train error with learn_rate: 0.034, num_leaves:78, min_data_in_leaf: 67 = 1.7558997825731877
Test error with learn_rate: 0.034, num_leaves:78, min_data_in_leaf: 67 = 1.7497339806829757
****************************************************************************************************
```

Conclusion

Feature Importance:

Lag features and rolling features have more importance as compared to other features. Simple features like ‘wday’ and ‘day_of_month’ have had better impact as compared to other features.



Performance of different models:

The performance of models has increased after featurization.

The RMSE score of linear models is high for training data as compared to test data. This indicates bias in the model. Decision tree model does not have this difference between train error and test error. Decision trees are prone to overfit, so they are not inherently biased. Light GBM model performed better than all the models.

Performance of models after featurization and hyper parameter tuning

No	Model	RMSE SCORE
3	Lasso regression	1.811
3	Ridge regression	1.811
4	Decision Tree	1.776
5	LGBM	1.75

Future work:

- Since this is a recursive time series problem where we need to predict data for next 28 days, I would like to implement LSTM and see how it works.
- As per Kaggle competition, we need to implement WRMSSE which is Weighted Root Mean Square Scaled Error for this problem. I would like to implement WRMSSE metric along with RMSE.

References:

https://www.appliedroots.com

https://mofc.unic.ac.cy/m5-competition/

https://www.kaggle.com/c/m5-forecasting-accuracy/discussion/138881

https://machinelearningmastery.com/basic-feature-engineering-time-series-data-python/

https://www.kaggle.com/tarunpaparaju/m5-competition-eda-models

https://www.kaggle.com/c/m5-forecasting-accuracy/discussion/163216

https://www.kaggle.com/c/m5-forecasting-accuracy/discussion/163684

https://medium.com/thecyphy/m5-forecasting-accuracy-af6c45fb7d58

https://www.kaggle.com/code/headsortails/back-to-predict-the-future-interactive-m5-eda/report

https://www.analyticsvidhya.com/blog/2019/12/6-powerful-feature-engineering-techniques-time-series/

https://www.kaggle.com/competitions/m5-forecasting-accuracy/discussion/147425

https://dipanshurana.medium.com/m5-forecasting-accuracy-1b5a10218fcf

https://medium.com/@shantanuekhande19/m5-forecasting-accuracy-73343a873685

https://www.coursera.org/lecture/python-project-for-ai-application-development/creating-web-applications-using-flask-QdaLa

Deployment

Deployment of machine learning models is the process of making your models available to the end users or systems. Flask, as we saw earlier, is a Microframework for creating web application quickly and easily with python.

Below are the files required for deployment:

- model.pkl – This file has the end model used to predict the sales. The trained model is saved to disk using pickle library.
- app.py – This file contains the flask web framework to handle the requests.
- index.html – We use this file to collect the data like ITEM_ID and STORE_ID from the end user.
- predict.html – This file has the html code to display the sales predicted for next 28 days.

The files predict.html and index.html are placed in a folder called template. The application uses templates to render HTML which will display in the user's browser.

Since my laptop has only 4GB of RAM, I have deployed the model using flask framework in google colab. Google Colab provides a VM(virtual machine) so we cannot access the localhost(all it does it route it to our local machine's localhost) as we do on our local machine when running a local web server. We can use ngrok for this purpose.

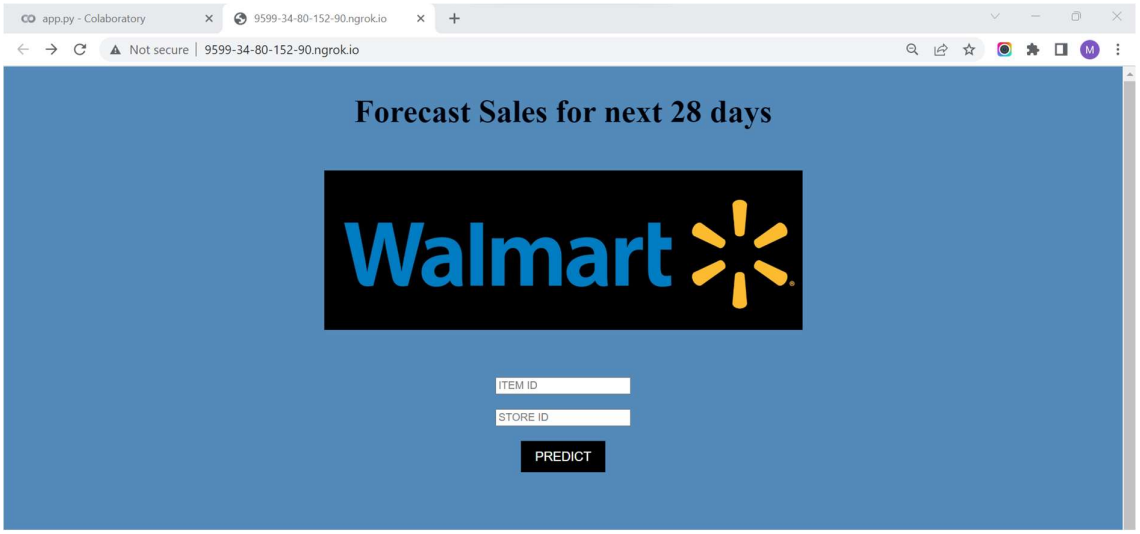
'ngrok' is a cross-platform application that enables developers to expose a local development server to the Internet with minimal effort. The software makes your locally-hosted web server appear to be hosted on a subdomain of ngrok.com, meaning that no public IP or domain name on the local machine is needed.

Application screenshots:

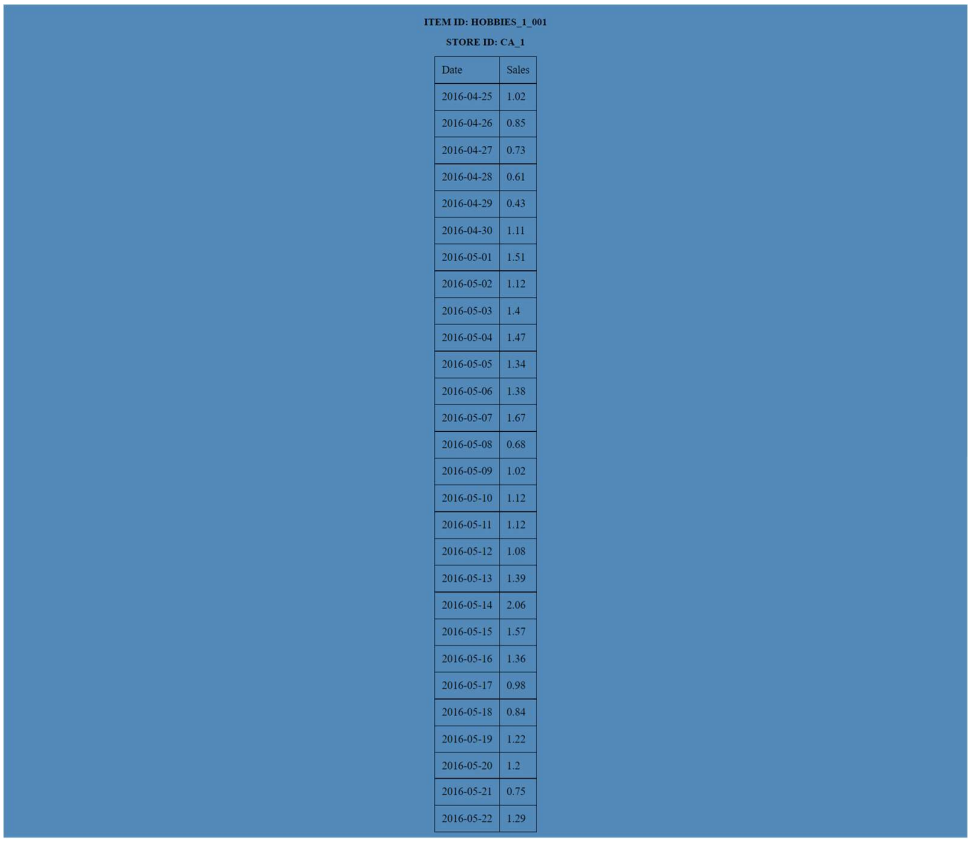
A URL is generated once we run app.py as shown below:

```
...   * Serving Flask app "__main__" (lazy loading)
      * Environment: production
        WARNING: This is a development server. Do not use it in a production deployment.
        Use a production WSGI server instead.
      * Debug mode: off
    INFO:werkzeug: * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
      * Running on http://9599-34-80-152-90.ngrok.io
      * Traffic stats available on http://127.0.0.1:4040
    INFO:werkzeug:127.0.0.1 - - [15/Oct/2022 16:28:02] "GET / HTTP/1.1" 200 -
    INFO:werkzeug:127.0.0.1 - - [15/Oct/2022 16:28:02] "GET /static/walmart.jpg HTTP/1.1" 200 -
    INFO:werkzeug:127.0.0.1 - - [15/Oct/2022 16:28:04] "GET /favicon.ico HTTP/1.1" 404 -
    100%|██████████| 4/4 [00:00<00:00, 492.91it/s]
    100%|██████████| 5/5 [00:00<00:00, 187.40it/s]
    INFO:werkzeug:127.0.0.1 - - [15/Oct/2022 16:28:19] "POST /predict HTTP/1.1" 200 -
```

Copy and open the URL on web browser. Below is the screenshot of user interface.



Once we enter ITEM_ID and STORE_ID, below output is generated:



ITEM ID: HOBBIES_1_001

STORE ID: CA_1

Date	Sales
2016-04-25	1.02
2016-04-26	0.85
2016-04-27	0.73
2016-04-28	0.61
2016-04-29	0.43
2016-04-30	1.11
2016-05-01	1.51
2016-05-02	1.12
2016-05-03	1.4
2016-05-04	1.47
2016-05-05	1.34
2016-05-06	1.38
2016-05-07	1.67
2016-05-08	0.68
2016-05-09	1.02
2016-05-10	1.12
2016-05-11	1.12
2016-05-12	1.08
2016-05-13	1.39
2016-05-14	2.06
2016-05-15	1.57
2016-05-16	1.36
2016-05-17	0.98
2016-05-18	0.84
2016-05-19	1.22
2016-05-20	1.2
2016-05-21	0.75
2016-05-22	1.29