This notebook outlines the modeling workflow I used to solve this challenge

1 EDA

From the given EDA (notebooks/eda1.ipynb), I got a sense of the features in the dataset. Since the series were not of the same length, it was essential to take a model which could handle that. Each store-dept combination would have different amount of data to learn from

The sales were skewed, with some weeks having very huge sale numbers (the sales were also skewed for certain departments) This made me create a configurable option to log transform the weekly_sales

The ACF plot indicated there is +ve correlation upto 5 weeks, but the weeks after are not significant. Temperature, Fuel_price, Marrkdown 2,3,4 were removed as they were not highly correlated with fuel price. However, correlation only measures a linear relationship, so the flexibility to add these features was given through a CONFIG.yaml file.

Markdown 4 is highly correlated with markdown 1, so it was dropped.

Store size was highly correlated with sales, so it seems that is an important feature (understandably,larger stores have more sales)

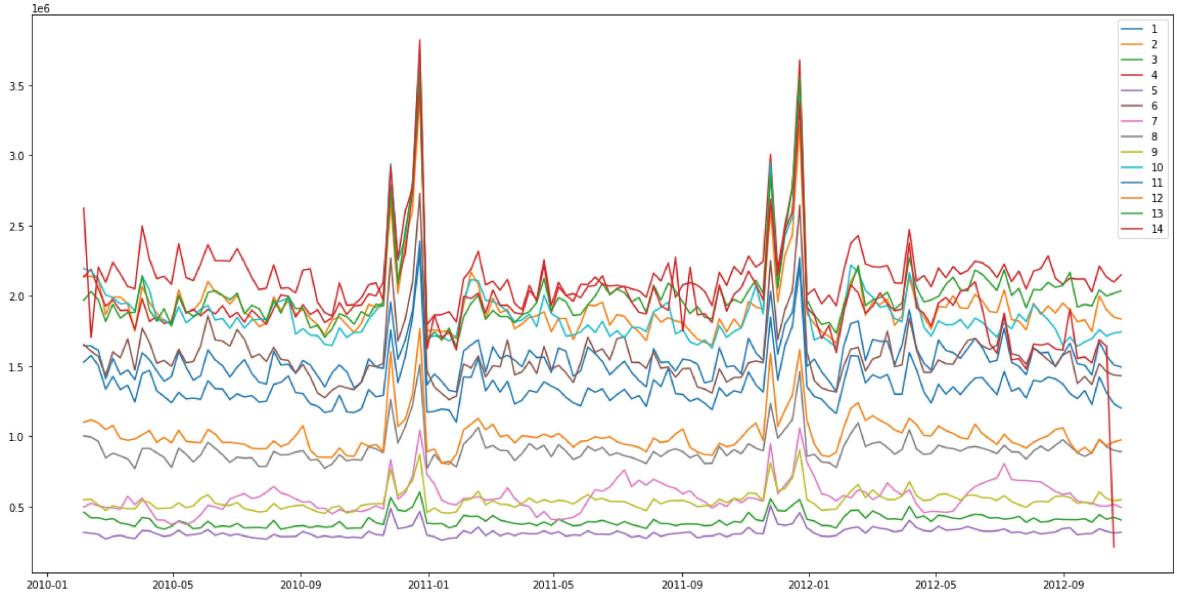
I wanted to do some extra exploratory analysis to identify data quality. It was seen that 0.23 % of sales are negative. I converted those to 1 so they could log transform to zero. The markdowns had a lot of missing values, so I imputed them with zero (to nullify their effect). But this imputation would be more informed if given the implication of the markdown column.

Summary statistics of numeric columns

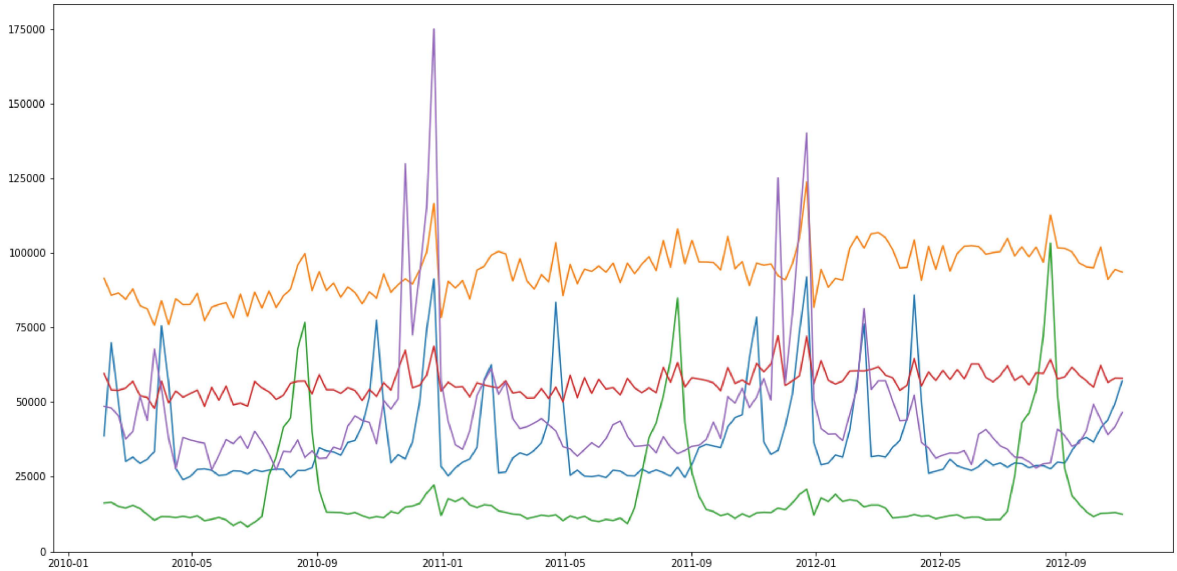
	Size	CPI	Unemployment	MarkDown1	MarkDown5	Weekly_Sales
count	137991.000000	137991.000000	137990.000000	49320.000000	49320.000000	137991.000000
mean	149063.943989	187.566431	7.603765	8127.484987	5122.983299	18811.313135
std	61269.553530	39.499884	1.889095	8627.341252	5344.725276	26596.725987
min	34875.000000	126.064000	3.879000	35.940000	135.160000	-1098.000000
25%	112238.000000	130.829533	6.425000	3093.390000	2314.580000	2732.245000
50%	155078.000000	211.894272	7.346000	6086.210000	3915.850000	8787.870000
75%	202505.000000	218.642470	8.257000	10192.490000	6147.500000	23474.870000
max	219622.000000	227.232807	14.313000	78124.500000	58068.140000	693099.360000

Since the sales are on different scales for each store and dept, I scaled them in a MinMax fashion. This helped me analyse their trends and seasonality. The data did not have a significant trend, but was very seasonal. The last month (december) recorded high sales across all years

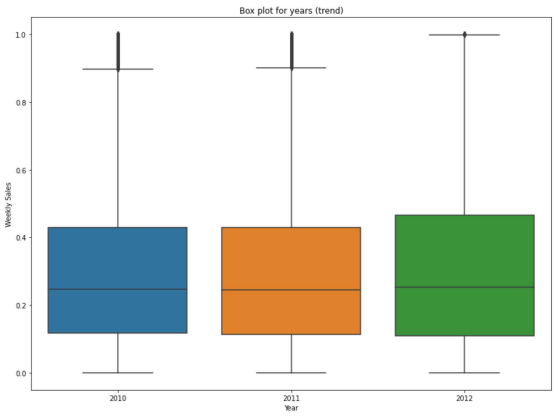
Store sales

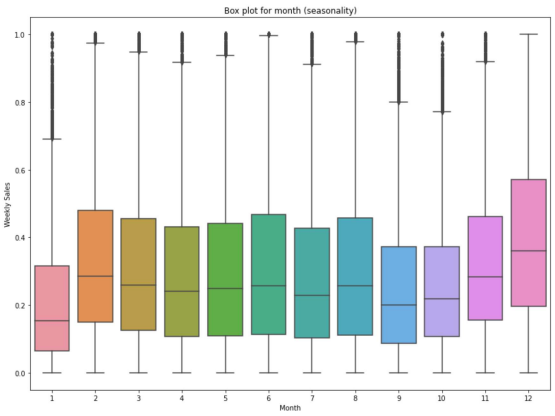


Department sales

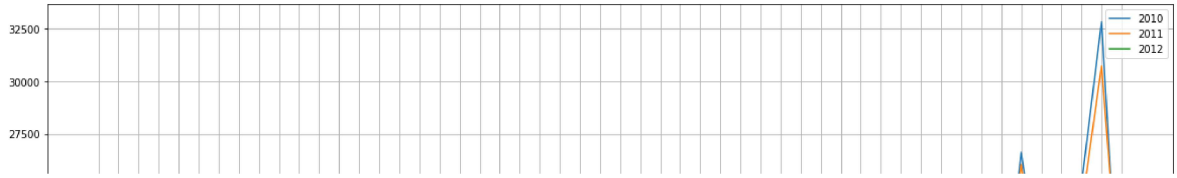


Trends and Seasonality





Sales per year



From the above plots, it seems while all stores have a similar pattern, each department is very unique. So department would be very important to forecast the sales

2 Data preparation

Data is prepared sequentially, with data being preprocessed, then creating various features and splitting the dataset into train and test. The numeric columns are on different scales, so they are standardized ensuring that there is no data leakage between train and test.

3 Feature engineering

Here, I mainly focused on encoding the features which were categorical. I created time based features like year, month, week number. I also added lag features to the model so it can be used in a supervised setting (user can control number of lags from config file)

However, this would require me to recursively add lags at each predicted time step (Current prediction would be a lag for the next prediction). I left that due to time constraints

4 Model and evaluation

I unfortunately was not able to experiment as much as I'd have liked. I saw the data would be need to be processed in different ways to use models like VARIMA, but it is not as powerful (I was not sure if the generated time series followed the ARIMA process, or if there are linear dependencies) and I wanted to use the continuous and categorical features (exogenous). Hiearchical forecasting was also on my list to try, but I was not sure about the results or implementation. Rather than try models which may or may not work, I focused on XGBoost which would be able to handle the differing scales and types of features. It was also robust to seasonality I added the time based features to convert it to a supervised problem.

I also experimented with DeepAR from PyTorch Forecasting, and was able to get it running. However, the results were not that strong (see implementation in notebooks/deepar.py) and I did not have time to investigate. However, I believe it is a strong candidate model for this problem as it is designed for multiple time series and used recurrent neural nets

For evaluation, I used MAPE, Symmetric MAPE and WMAPE. The reason for using them are for their advantages outlined in this post (https://medium.com/@vinitkothari.24/time-series-evaluation-metrics-mape-vs-wmape-vs-smape-which-one-to-use-why-and-when-part1-32d3852b4779)

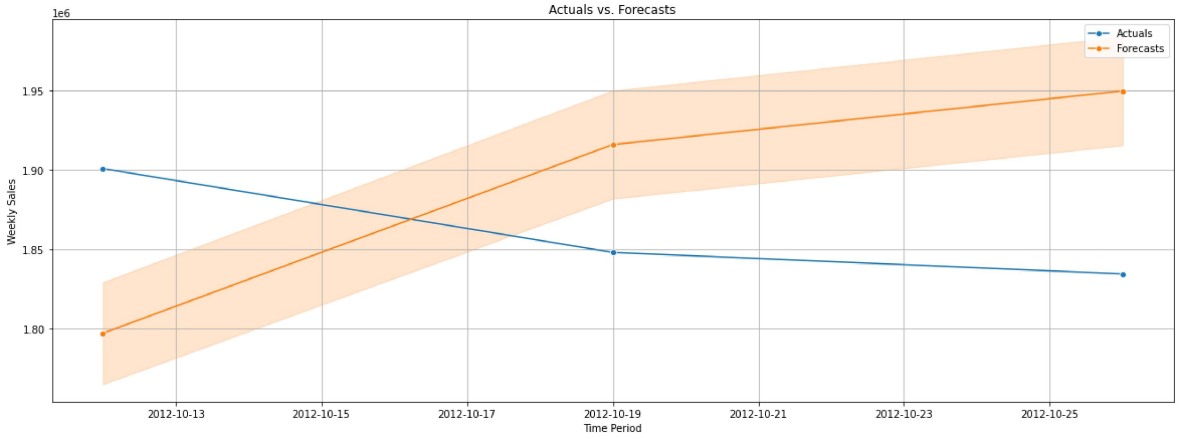
The results of my hyperparamter tuning are in notebooks/hyperparameter.ipynb. Squared error was used as the objective function for the model.

Final results for evaluation - {'mape': 12.54, 'smape': 0.103, 'wmape': 0.113}

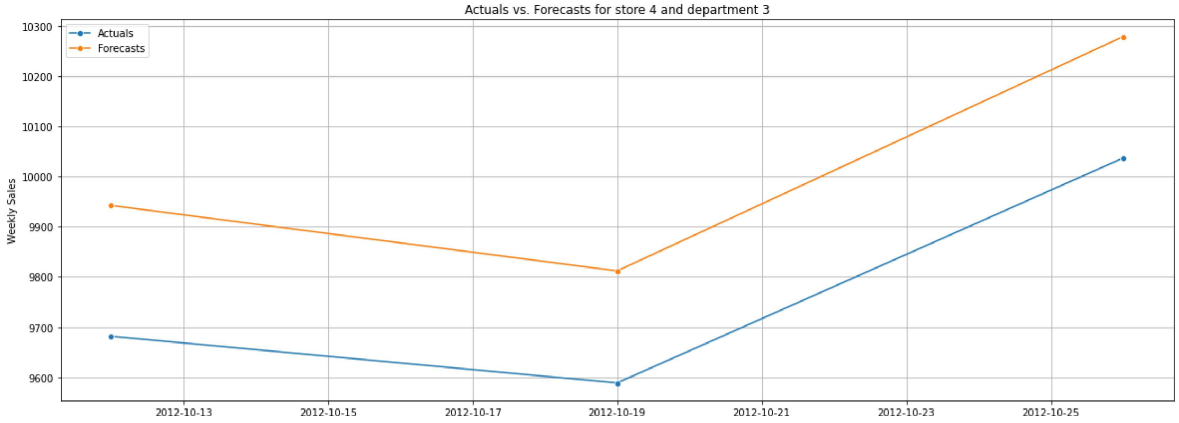
Qualitative evaluation

Summed forecasts for all stores and departments

Forecast for store 1



Forecast for store 4 department 3



5 Pipeline

The pipeline is very streamlined, it prints and saves the forecasts for the 3 week period by running main.py. The store and department number can be passed as command line arguments to print specific forecasts.

All the flexibility is included in CONFIG.yaml where the features, lags, train_split_date, forecast_create_date, transform_target etc can be controlled.

I also was not sure how to include forecast_create_date, but in general the test set would be created to include dates greater than train_split_date and would be less than or equal to forecast_create_date.

There are many ways this could be optimized but I refrained due to time constraints

1. Create a checkpoint folder for version control and save model params as metadata
2. Create a cmd argument to use a specific model for the forecasts
3. Use MLFlow tracking to record experiments
4. Automatically pick parameters form the best model rather than model_config.yaml

5. Create an endpoint for serving the model predictions given data. This could include a feature store which would take in the provided features (like store, department), calculate the others from databases (size, type of store) and obtain others from different apis (CPI, unemployment etc.)

As for well engineered code, I missed out on documenting and including asserts/tests for all functions. But this would be done ideally.

6 Final Comments

All in all, I had a lot of fun in this assesment. I did my best given the time, but I could definitely identify areas of improvement. It was a challenging problem to think and implement, so it helped me understand the complexities of the role.