WALMART SALES FORECASTING

INDEX

PROJECT DESCRIPTION & SUMMARY

In a world running on generating and analyzing data, harnessing and utilizing it to create effective marketing strategies is extremely essential for organizations. For an establishment as big as Walmart, it is necessary to organize and analyze the large volumes of data generated to make sense of existing performance and identify growth potential. The main goal of this project is to understand how different factors affect the sales for Walmart and how these findings could be used to create more efficient plans and strategies directed at increasing revenue.

Time series data often arise when monitoring industrial processes or tracking corporate business metrics. The analysis of time series accounts for the fact that data points taken over time may have an internal structure (such as autocorrelation, trend or seasonal variation) that should be accounted for.

This report explores the performance of a subset of Walmart stores and forecasts future weekly sales for these stores based on several models including Ridge Regression, Random Forest, XGBoost, SARIMA and LSTM model. An exploratory data analysis has also been performed on the dataset to explore the effects of different factors like holidays, fuel price, and temperature on Walmart's weekly sales which provides an overview of the overall predicted sales.

Through the analysis, it was observed that the SARIMA model provided the most accurate sales predictions. Relationships were observed between factors like store size, holidays, unemployment, and weekly sales. Through the implementation of interaction effects, as part of the linear models, the relationship between a combination of variables like temperature, CPI, and unemployment was observed and had a direct impact on the sales for Walmart stores.

For this project, we used the "Walmart Sales Forecasting" Kaggle competition as our benchmark reference. The competition was posted 9 years ago and has over 688 participants. The competition is evaluated on a weighted mean absolute error (WMAE) as shown in the image below. Since the sales are higher during the holidays which constitutes only a very small fraction of the year, they are assigned higher weights. This contributes towards giving more weightage to the prediction of Sales during the holidays as opposed to off-seasonal sales.

$$WMAE = \frac{1}{\Sigma_i w_i} \Sigma_i w_i |y_i - \hat{y}_i|$$

where

n is the number of rows

$\hat{y}_i$ is the predicted sales

y_i is the actual sales

w_i are weights. w = 5 if the week is a holiday week, 1 otherwise

LITERATURE REVIEW

A company can make and stock products on demand according to sales forecasting. Future sales can be anticipated using historical sales data based on the examination of time series. The Seasonal Auto-Regressive Integrated Moving Average (SARIMA) approach along with an analysis of its use in predicting the monthly sales of a chemical company is presented in the study conducted by Heng Zhao (Heng Zhao, 2022). The company's sales data is used as the dataset for the SARIMA model. The time series is stabilized using the Difference Operation, and the best model among the five SARIMA models is chosen using the Box-Jenkins approach. Mean Square Error measures the model's forecasting accuracy. The forecast's outcomes are consistent with the data on actual sales, which attests to SAMIRA's success in accurately predicting sales for the chemical company. They have also employed the SARIMA model to project future sales for the company based on the sales data. [4]

Michael Crown (Crown, 2016) examined a comparable dataset but concentrated on using time series forecasting and non-seasonal ARIMA models to make his predictions. He used 2.75 years of sales data, including aspects of the store, department, date, weekly sales, and holiday data, to work on ARIMA modeling to produce one year's worth of weekly projections. The normalized root-mean-square error method was used to gauge performance (NRMSE). [2]

In 2020, Yi Niu (Yi Niu, 2020) combined XGBoost algorithm and feature engineering to process the same dataset provided by kaggle. Their XGBoost algorithm performed better than Logistic regression and Ridge algorithm. The RMSSE value of their XGBoost is 0.113 lower than Ridge and 0.141 lower than Logistic Regression. When it comes to feature engineering, they ranked the top 20 features based on the importance of each feature and have mentioned that a balance between accuracy of prediction and speed needs to be considered during model experiments to select features. [5].

The XGBoost algorithm adopts a more regularized model formulation to prevent over-fitting and improve its performance when compared to GBDT. It also has fast speed, good accuracy and anti-noise ability (Mesut Gumus, 2017). So, based on this literature review, we considered using the XGBoost algorithm for prediction. [3]

Recent advances in processing power and data accessibility have made deep neural network architectures quite effective at forecasting issues in a variety of fields. In the article, Bryan Lim and Stefan Zohren (Bryan Lim and Stefan Zohren, 2020) have surveyed the primary time-series forecasting architectures, emphasizing the fundamental components of neural network design. They have shown how temporal information can be used to make predictions one step ahead and discuss how they might be expanded to be used in multi-horizon forecasting. They have summarized two ways in which deep learning can be developed further to enhance decision support over time, with an emphasis on interpretability and counterfactual prediction techniques. Although several deep learning models have been created for time-series forecasting, there are still significant restrictions like, deep neural networks often need time series to be discretized at regular intervals, forecasting datasets with missing or erratic observations is challenging. [1]

DATA PROCESSING & SUMMARY STATISTICS

The dataset comes from an American retail organization, Walmart Inc. It comprises information from 45 Walmart division stores, from 2010 to 2012 primarily centered around their deals on a week after week premise. Each section has properties as takes after: the related store (recorded as a number), the comparing division (81 offices, each entered as a number), the date of the beginning day in that week, departmental week after week deals, the store measure, and a Boolean esteem indicating on the off chance that there's a major occasion within the week. The major occasions being one of Thanksgiving, Labor Day, Christmas or Easter. Together with the previously mentioned qualities may be a parallel set of highlights for each section counting Customer Cost List, unemployment rate, temperature, fuel cost, and special markdowns.

Data Set Description
There are four datasets provided by Walmart to build a predictive model.
Stores.csv - this file contains anonymized information about the 45 stores, indicating the type and size of the store:
Store: stores numbered from 1 to 45
Type: store type has been provided, there are 3 types — A, B and C.
Size: stores size has provided
Train.csv - this is the historical training data, which covers 2010–02–05 to 2012–11–01. Within this file you will find the following fields:
Store: the store number
Dept: the department number
Date: the week
Weekly_Sales: sales for the given department in the given store,
IsHoliday: whether the week is a special holiday week
Test.csv - This file is identical to train.csv, except we have withheld the weekly sales. We x§
Dept: the department number
Date: the week
IsHoliday: whether the week is a special holiday week
Features.csv - this file contains additional data related to the store, department, and regional activity for the given dates. It contains the following fields:
Store: the store number
Date: the week
Temperature: average temperature in the region
Fuel_Price: cost of fuel in the region
MarkDown 1–5: anonymized data related to promotional markdowns that Walmart is running. MarkDown data is only available after Nov 2011 and is not available for all stores all the time. Any missing value is marked with an NA.
CPI: the consumer price index
Unemployment: the unemployment rate
IsHoliday: whether the week is a special holiday week

Merging the data
"Merging" datasets are the process of bringing all datasets together into one, and aligning the rows from each based on common attributes or columns. In this case we will merge all three different sets ("Store", "Date"- from features and "Store" - from Store), will remove duplicated columns such as "IsHoliday_y".

Converting date variable into date-time object
Date variable was decomposed into three columns - date, month and year to analyze weekly and monthly sales.

Data Cleaning
Checking for Missing values:

```
Store               0
Dept                0
Date                0
Weekly_Sales        0
IsHoliday           0
Temperature         0
Fuel_Price          0
MarkDown1      270889
MarkDown2      310322
MarkDown3      284479
MarkDown4      286603
MarkDown5      270138
CPI                 0
Unemployment        0
Type                0
Size                0
Year                0
Month               0
Day                 0
Week_of_year        0
dtype: int64
```

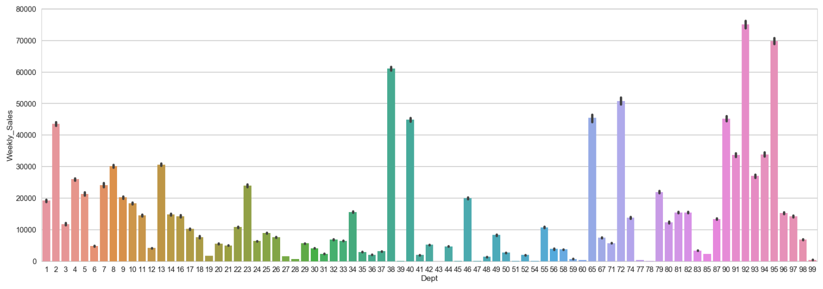
Except for 'MarkDown' columns, there are no missing values for all other colunms. The MarkDown columns represent Walmart clearance activities, which occurrs in different stores at different time. In this case, it is normal to see many NaN values in these columns. Later these NaN values are filled as 0.

Weekly sales cannot be less than or equal to zero. So we looked for such values and found 1358 rows having weekly sales value as 0 and less than 0. 1358 rows in 421570 rows means 0.3%, so we deleted and ignored these rows which contained wrong sales values.

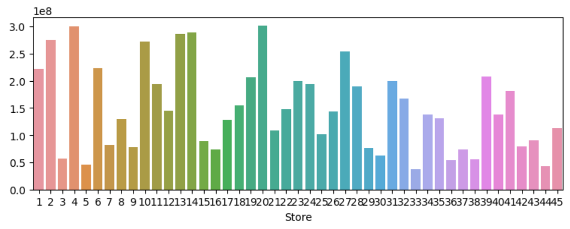
EXPLORATORY DATA ANALYSIS

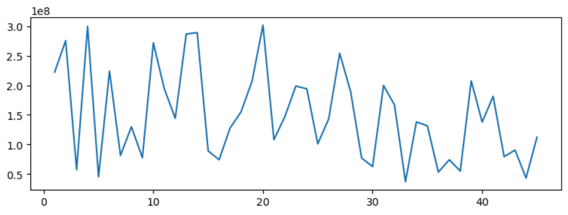
We performed EDA to analyze data sets and summerize their main characteristics.

- There are 45 stores and 81 department in data. Departments are not same in all stores.
- When we take the averages, it is seen that department 92 has higher average sales.

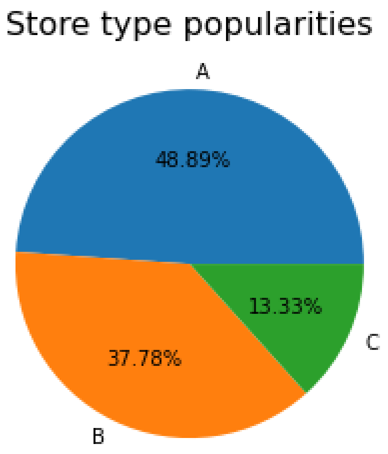


- In general, store 20 has the highest average weekly sales value followed by 4 and 14

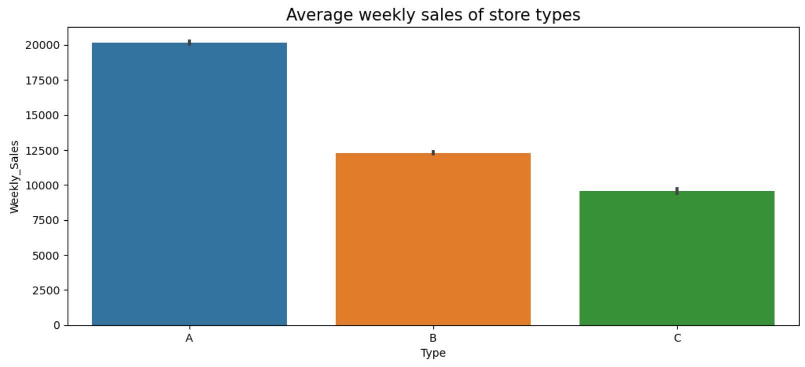




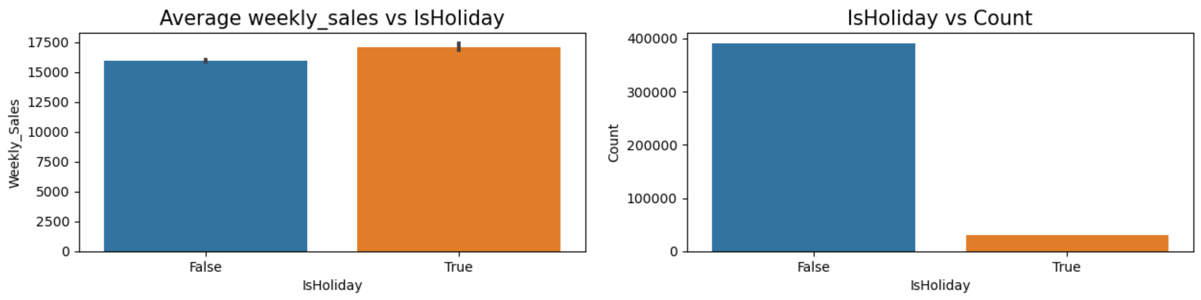
- Stores has 3 types as A, B and C according to their sizes. Almost half of the stores are bigger than 150000 and categorized as A. According to type, sales of the stores are changing.



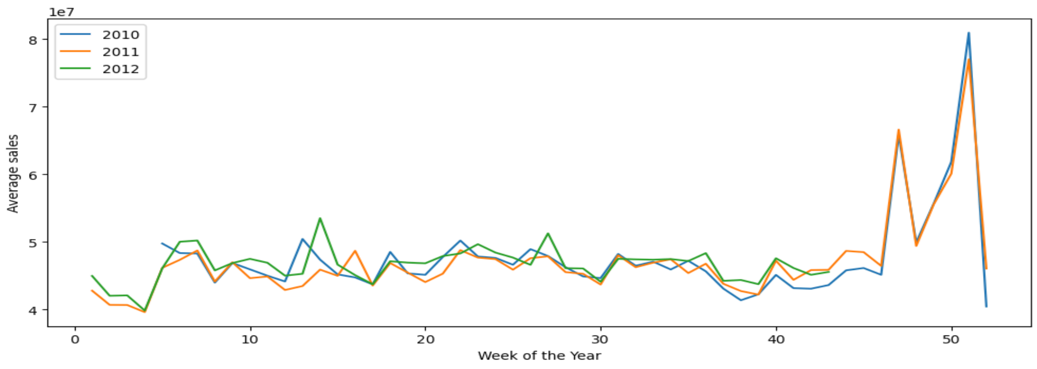
- From the plot below, we can infer that store-type A has the highest average weekly sales followed by B and C.



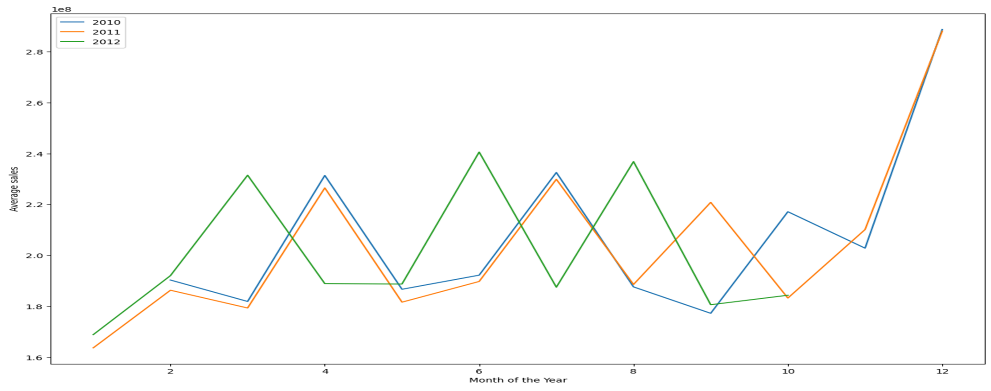
- There are 4 holiday values such as:
  Super Bowl: 12-Feb-10, 11-Feb-11, 10-Feb-12; Labor Day: 10-Sep-10, 9-Sep-11, 7-Sep-12; Thanksgiving: 26-Nov-10, 25-Nov-11; Christmas: 31-Dec-10, 30-Dec-11.
- It can be clearly seen that the average sales per week is much higher for holiday weeks although holidays accounts for very less portion of our data.



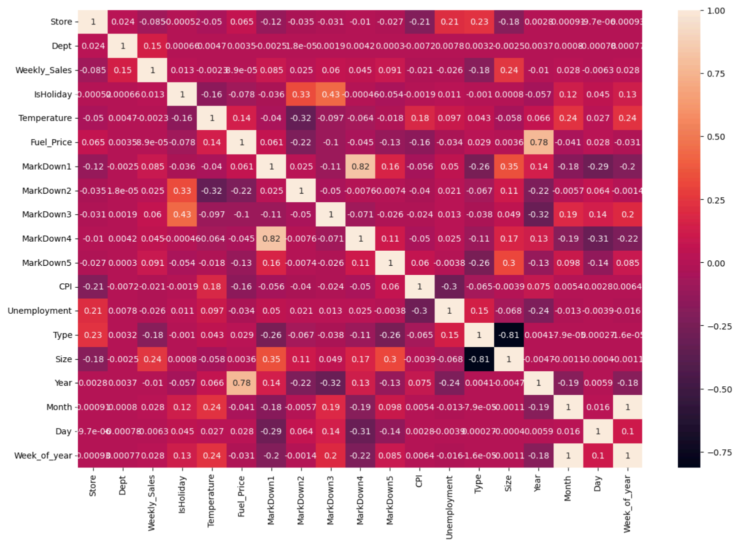
- The average sales trend is similar for each year. We can see that the week which had a holiday in it has higher sales in that particular region. For the Holiday Weeks i.e 5,35,46,51 we can see a sudden spike in the graph.



- Since our data contains Holidays on weekly basis, the Sales by Months graph does not provide much information as the Sales by weeks graph. However we can see that the highest sales is seen at the end of each year.



- Checking the correlation between our variables using heatmap.



The Tempeature, Fuel price, CPI, Unemployment have very low correlation with Weekly sales so can be removed. Also the month, day columns can also be dropped because of high correlation with week. Also, Markdown 4 and 5 highly correlated with Markdown 1 and can be dropped as well.

MACHINE LEARNING MODELS

The following models were trained on the dataset :

- SARIMA
- Ridge Regression
- Random Forest Regressor
- XGBoost Regressor
- LSTM
- Vanilla/ Stacked LSTM
- Multilayer Perceptron

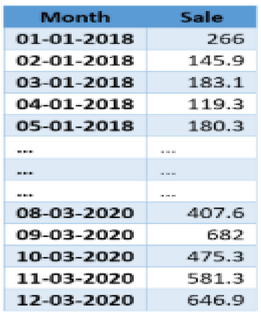
The error metric used to evaluate all the models is weighted mean absolute error (WMAE). We define this error using this function:

```python
def WMAE(data, prediction, actual):
    weights = data['IsHoliday'].apply(lambda x: 5 if x==1 else 1)
    return  np.round(np.sum(weights*abs(prediction-actual))/weights.sum(), 2)
```

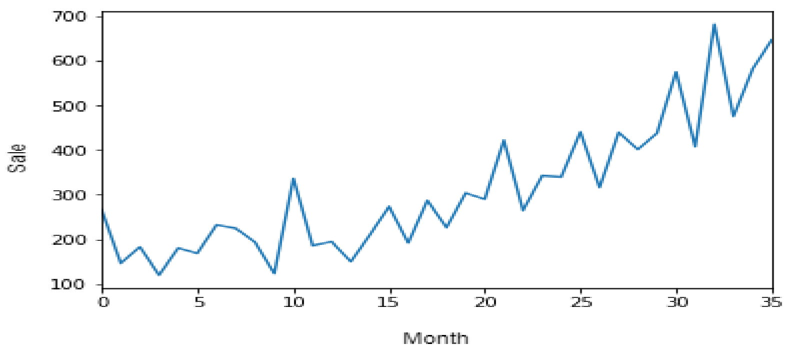
Time Series Analysis

What is Time Series Analysis?
> Time Series analysis comprises methods for analyzing time series data in order to extract meaningful statistics and other characteristics of the data. Objective of time series analysis is to understand how change in time affects the dependent variables and accordingly predict values for future time intervals.

What is Time Series Data?
> Any data recorded with some fixed interval of time is called time series data. This fixed interval can be hourly, daily, monthly or yearly. In time series data, time will always be an independent variable and there can be one or many dependent variables.



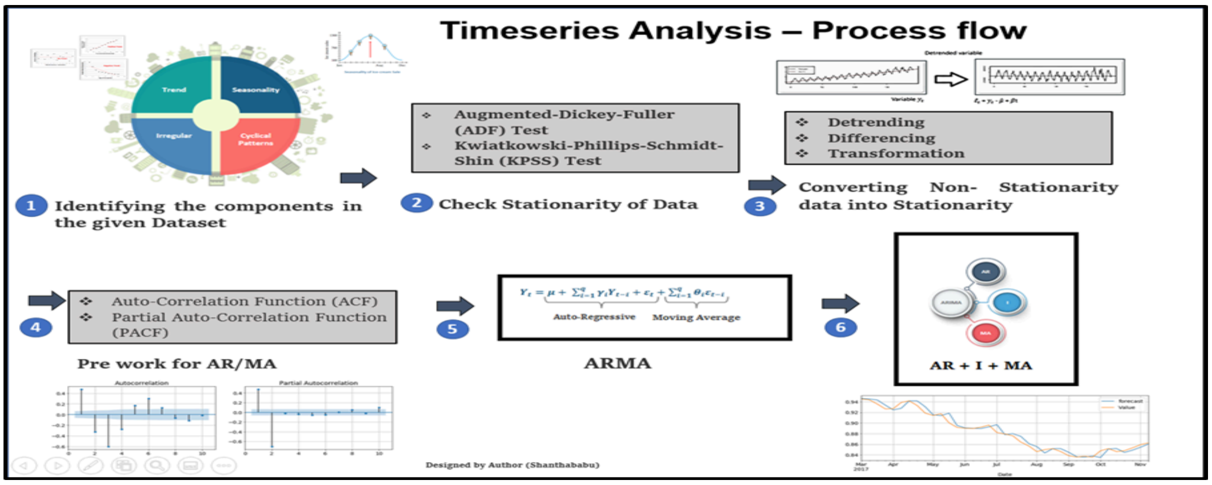
Month	Sale
01-01-2018	266
02-01-2018	145.9
03-01-2018	183.1
04-01-2018	119.3
05-01-2018	180.3
...	...
...	...
...	...
08-03-2020	407.6
09-03-2020	682
10-03-2020	475.3
11-03-2020	581.3
12-03-2020	646.9



What are the steps involved in Time Series Analysis?
> The general process involved in Time Series Modelling is as follows:
1. Visualize the Time Series Data- Plot the series to check for outliers
2. Apply Statistical tests to check if the series is stationary i.e. check if it has trend and seasonality.
3. If non-stationary, make it stationary

4. Choose the best parameters for fitting the model
5. Construct the model using these parameters
6. Use the model to make predictions
7. Extract insights from these predictions

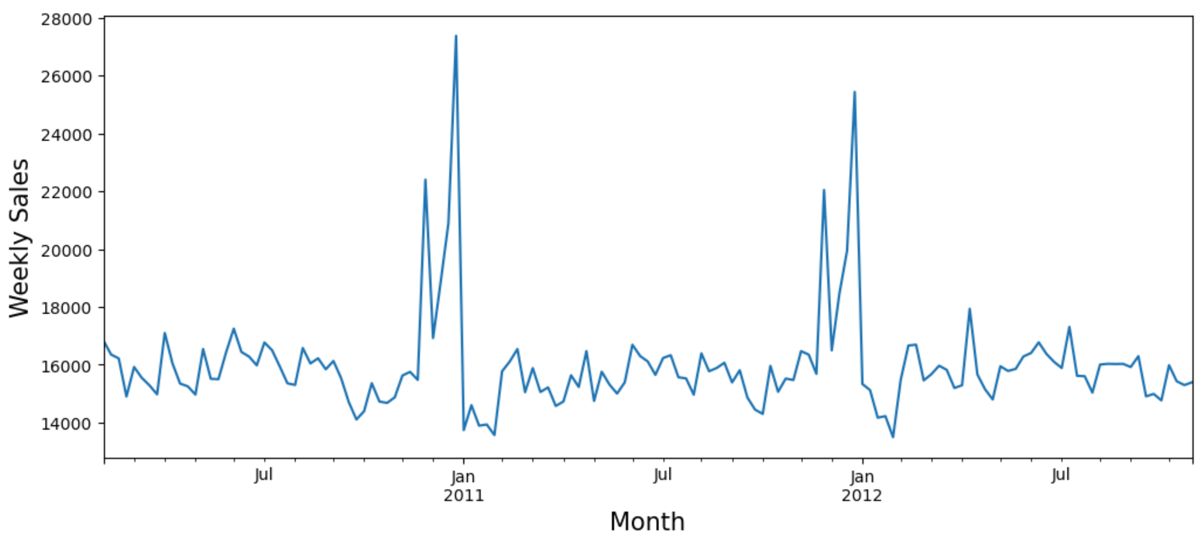
More detailed description on the flow of Time Series Analysis can be found in the below flow-chart



Timeseries Analysis – Process flow
Trend
Seasonality
Irregular
Cyclical Patterns
1 Identifying the components in the given Dataset
Augmented-Dickey-Fuller (ADF) Test
Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test
2 Check Stationarity of Data
Detrending
Differencing
Transformation
3 Converting Non- Stationarity data into Stationarity
4 Auto-Correlation Function (ACF)
Partial Auto-Correlation Function (PACF)
Pre work for AR/MA
Autocorrelation
Partial Autocorrelation
5 $Y_t = \mu + \sum_{i=1}^{p} \gamma_i Y_{t-i} + \epsilon_t + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}$
Auto-Regressive
Moving Average
ARMA
6 AR + I + MA
Designed by Author (Shanthababu)

Visualizing the Data

We plot our weekly sales data for visualization



We can see that mean and variance is nearly constant throughout the data. However, there is some sort of seasonality associated with the weekly sales i.e in the months of Nov and Dec the sales are very high due to holidays like Christmas and Thanksgiving. This clearly tells its a Time Series problem and it will be interesting to look more into it.

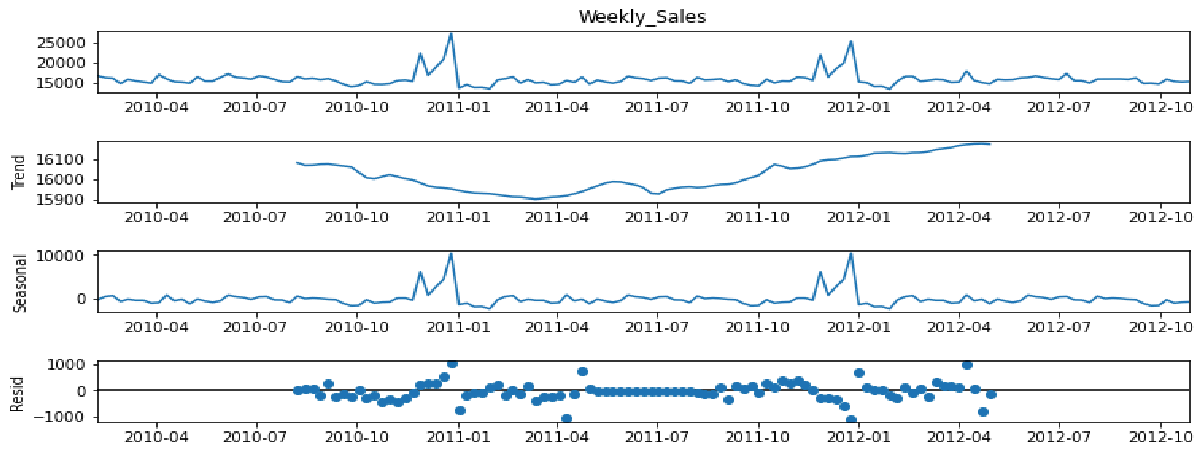
Decomposing the Time-Series to Identify Components like Trend, Seasonality, Noise

From the plot above, the mean and variance of the time series data remains nearly constant throughout the data. Hence, there is no need to transform the data. We now proceed to check the trend and seasonal components of the data. Each time series can be decomposed into 3 components –

- Trend
- Seasonality
- Noise

We can decompose our data into these components using the following code:

```
decomposition = seasonal_decompose(df_ts['Weekly_Sales'], period=52)
residual= decomposition.resid
```



We can see that there is no significant trend in our data but there is an annual seasonality. Let's confirm this with some statistical tests.

Checking if the data is Non-Stationary by determining the rolling statistics and performing the Dickey-Fuller test

The following function calculates the rolling statistics- Rolling Mean and Standard Deviation and also performs the Augmented Dickey-Fuller Test for a given time series. Based on the results, we decide if our data is Stationary or not.

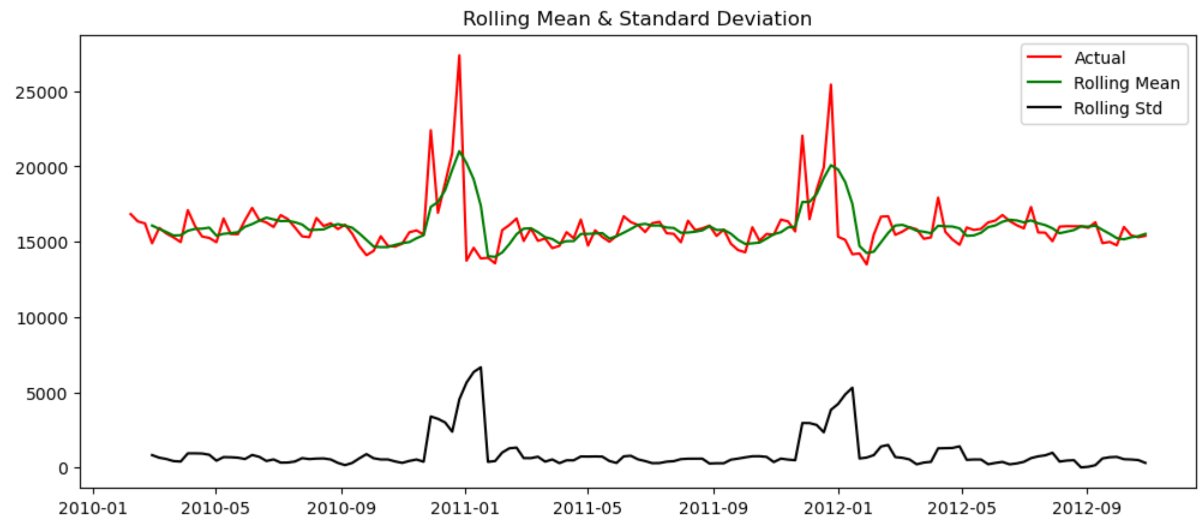
```
def stationarity(tmseries):
    #determining the rolling statistics
    rolmean=tmseries.rolling(window=4).mean()
    rolstd=tmseries.rolling(window=4).std()

    plt.figure(figsize=(12,5), dpi=100)
    actual=plt.plot(tmseries, color='red', label='Actual')
    mean_6=plt.plot(rolmean, color='green', label='Rolling Mean')
    std_6=plt.plot(rolstd, color='black', label='Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    #performing the Dickey-Fuller Test
    print('Dickey-Fuller Test: ')
    dftest=adfuller(tmseries, autolag='AIC')
    dfoutput=pd.Series(dftest[0:4], index=['Test
Statistic','p-value','Lags Used','No. of Obs'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
```

Let us run the function for our time series data and see the results.

```
stationarity(df_ts['Weekly_Sales'])
```

We get the following output:

Rolling Statistics:



Augmented Dickey-Fuller (ADF Test):

```
Dickey-Fuller Test:
Test Statistic          -5.930803e+00
p-value                  2.383227e-07
Lags Used                4.000000e+00
No. of Obs               1.380000e+02
Critical Value (1%)     -3.478648e+00
Critical Value (5%)     -2.882722e+00
Critical Value (10%)    -2.578065e+00
dtype: float64
```

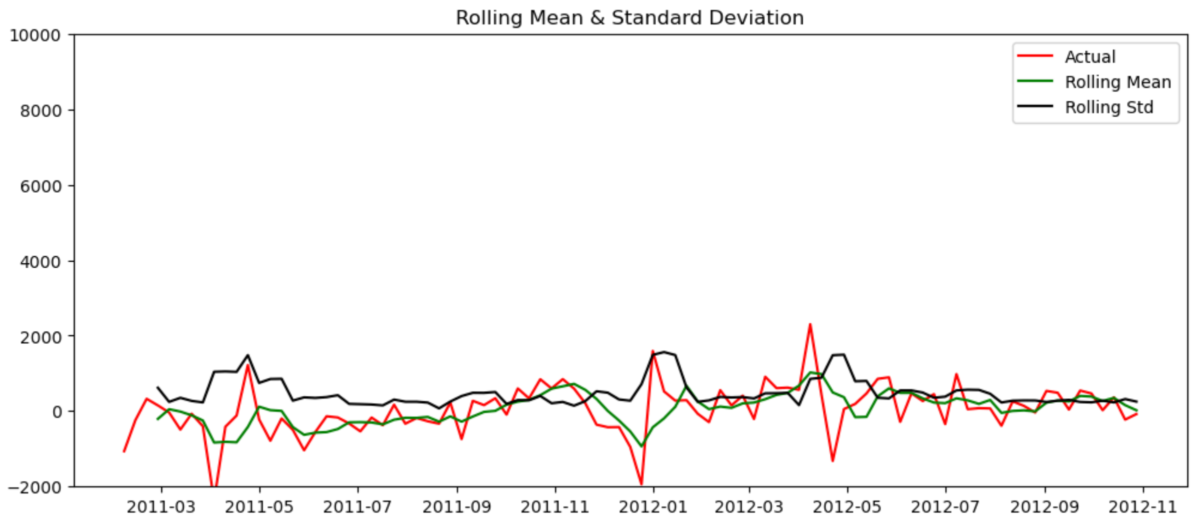
Observations:
We can see that the mean and variance are approximately consistent over the weeks. Also, this is confirmed by the Dickey-Fuller test where we get p-value less than 0.05.
Hence, we can reject the null hypothesis and conclude that the data is stationary.

But from the above rolling statistics plots, we can see that there is an annual seasonality in our data.

Our next step would be to perform seasonal differencing at lag= 52 (weeks i.e. yearly) to remove the seasonality.
Performing Seasonal differencing to remove the Seasonality component from our data and again checking the rolling statistics:

```
df_s = pd.DataFrame({'Weekly_Sales': df_ts['Weekly_Sales'] -
df_ts['Weekly_Sales'].shift(52)})
df_s['IsHoliday']= df_ts['IsHoliday']
df_s = df_s.dropna()
stationarity(df_s['Weekly_Sales'])
```

We get the following results:



```
Dickey-Fuller Test:
Test Statistic          -7.398813e+00
p-value                  7.651130e-11
Lags Used                0.000000e+00
No. of Obs               9.000000e+01
Critical Value (1%)     -3.505190e+00
Critical Value (5%)     -2.894232e+00
Critical Value (10%)    -2.584210e+00
dtype: float64
```

Now that our data is completely stationary i.e. free of trend and seasonality, we can proceed to the modelling part.

Implementation of SARIMA Model

Seasonal Autoregressive Integrated Moving Average (SARIMA) is an extension to ARIMA that supports the direct modeling of the seasonal component of the series. Since our data has an annual seasonal component we use SARIMA.

It consists of three major hyperparameters to specify the autoregression (AR), differencing (I) and moving average (MA) for both seasonal and non-seasonal components of the series, as well as an additional parameter for the period of the seasonality.

The details of the parameters that the model takes is as follows:
p and seasonal P: indicate the number of AR terms (lags of the stationary series)

d and seasonal D: indicate differencing that must be done to stationary series

q and seasonal Q: indicate the number of MA terms (lags of the forecast errors)

lag: indicates the seasonal length in the data

Finally the SARIMA model equation looks like:

SARIMA (p, d, q) x (P, D, Q, Lag)

To find these parameters we do the following process :

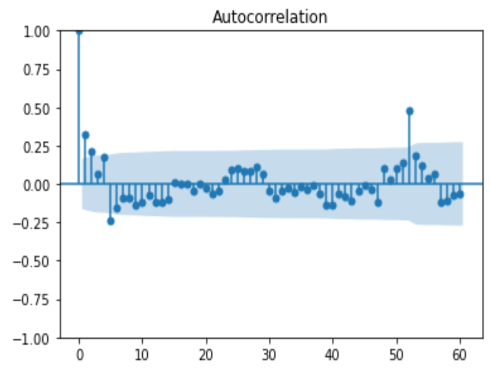
1. Plot the Auto-Correlation Function (ACF) and Partial Auto- Correlation Function (PACF) graphs to get a rough estimation of our parameters.
2. Perform a grid search by taking these estimations to get the best combination of the values based on the AIC scores.

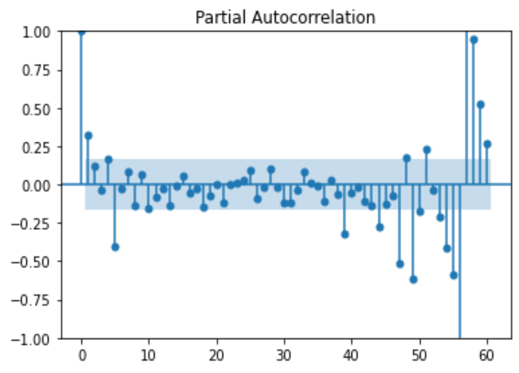
By now we have identifies 3 out of 7 parameters and they are:
d = 0 (As we did not take any differencing because our data was already stationary)
D = 1 (As we performed a seasonal differencing to remove the seasonality
Lag = 52 (weeks i.e. yearly)

Estimating the Modelling Parameters from ACF and PACF plots

We use the following code to generate these plots:

```
plot_acf(df_ts['Weekly_Sales'],lags=60)
matplotlib.pyplot.show()
plot_pacf(df_ts['Weekly_Sales'],lags=60)
matplotlib.pyplot.show()
```





From the ACF plot, we have initial spikes at lag = 1 and seasonal spikes at lag = 52, which means a probable AR order of 3 or 4 and seasonal AR order of 1

From the PACF plot, we have initial spikes at lag = 1 and a lot of spikes around 50 indicating seasonality. So we can take a probable MA order of 2 to 4 and seasonal MA order of 1

Now we will perform a grid search with the list of possible values around our estimated parameters p , d, P, Q. We will then pick the model with the least AIC.

We use the following code to generate a grid of estimated parameters:

```
# Define the p, d and q parameters
p = q =  range(0,5)
d = [0,1]
# Generate all different combinations of p, d and q triplets
pdq = list(itertools.product(p, d, q))
# Generate all different combinations of seasonal p, d and q triplets
seasonal_pdq = [(x[0], x[1], x[2], 52) for x in
list(itertools.product(p, d, q))]
```

Splitting the dataset into train and test data

```
ts_train = df_ts[:int(0.5*(len(df_ts)))]
ts_test =  df_ts[int(0.5*(len(df_ts))):]
```

Checking the combination with least AIC score:

```
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(ts_train['Weekly_Sales'],
                  order=param, seasonal_order=param_seasonal)
            results = mod.fit(method = 'powell')
            print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal,
results.aic))
        except:
            continue
```

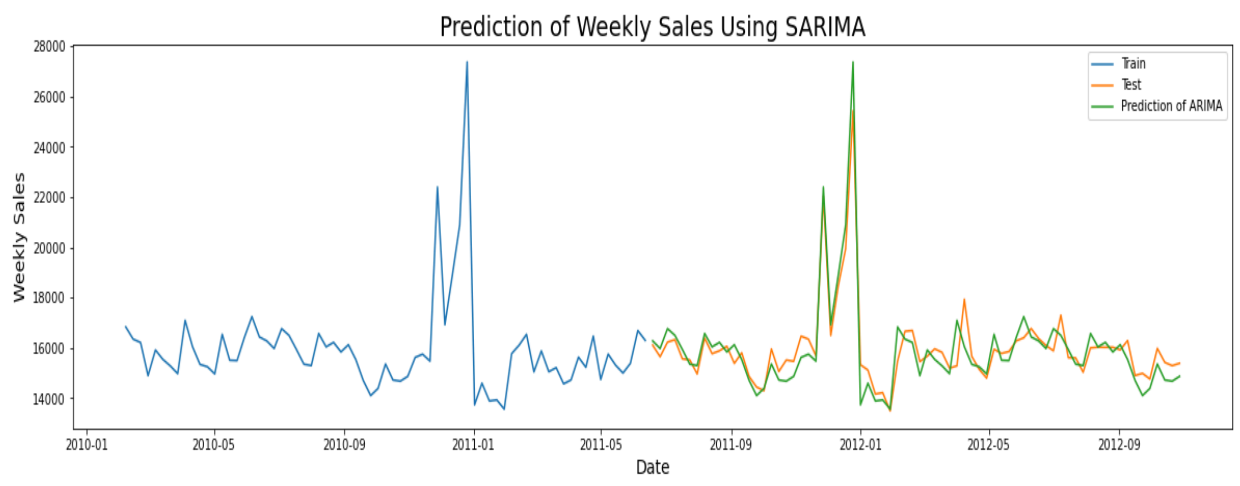
After running the above code, I got the least AIC value for
order = (3, 0, 4) and seasonal_order = (1, 1, 1, 52)

Now we fit our SARIMA model using these parameters and record the WMAE (Weighted Mean Average Error)

```
import statsmodels.api as sm
mod = sm.tsa.statespace.SARIMAX(ts_train['Weekly_Sales'],
                                order=(3, 0, 4),
                                seasonal_order=(1, 1, 1, 52))
results = mod.fit()
```

```
print(results.summary().tables[1])
```

Plotting the results and calculating the Error:



```
WMAE(ts_test,y_pred['Prediction'],ts_test['Weekly_Sales'])
```

```
Output: 438.62
```

In conclusion, our model does a very good job in predicting the future sales which is evident from the plot and also from the WMAE error.

Ridge Regression

We implemented the Ridge regression model from sklearn.linear_model and calculated the WMAE error using the above defined function.

```
Output:
The WMAE loss for the training set is 14785.88.
The WMAE loss for the validation set is 14829.79.
```

The WMAE error was found to be very high for Ridge Regression: 14829.79.

Random Forest Regressor

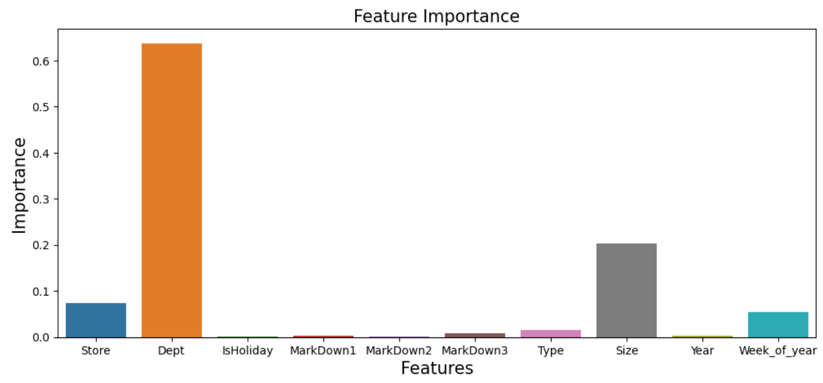
The random forest regression works by creating numerous distinct regression decision trees during the training phase. On the basis of the criteria it chose, each tree forecasts a choice. Following that, the random forest calculates a prediction by averaging individual predictions.

We started to fit the model with random values for parameters:

```
rf_model = RandomForestRegressor(n_estimators=50, random_state=432,
n_jobs=-1, max_depth=35, max_features = 'auto',min_samples_split =
5).fit(X_train, y_train)
```

WMAE on training and testing data was found to be 858.85 and 1657.81 respectively.

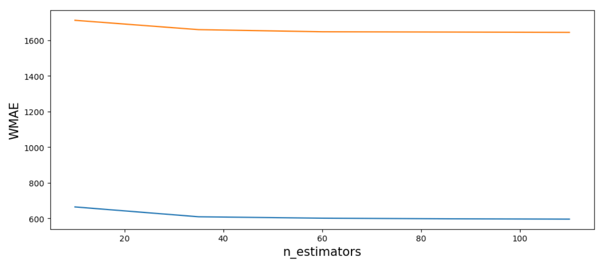
Next, we found important features in the data for random forest model:

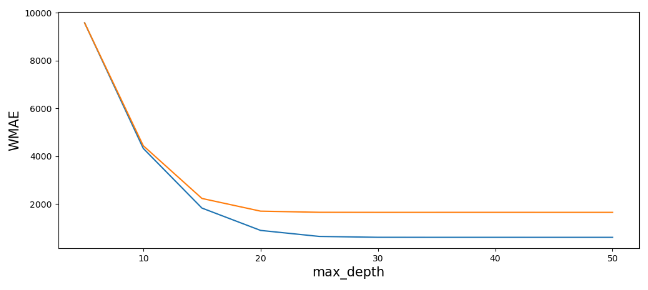


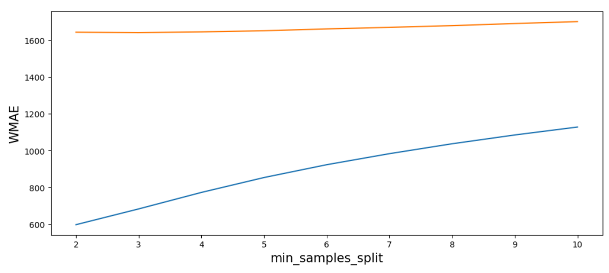
We dropped a few least important features like Markdown1, Markdown2 and Year and re-ran the model to check if it reduces the WMAE. But it increased the error to 1841.44. This means our model is learning these features. So we went forward with the previous model.

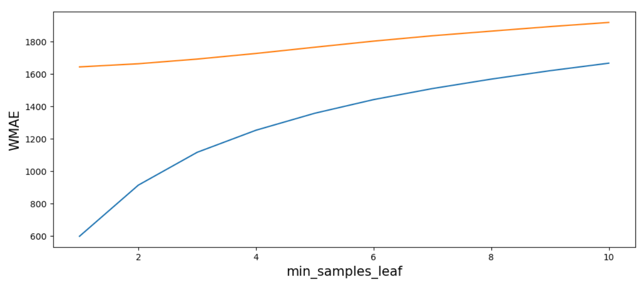
Hyperparameter tuning for Random Forest Regressor
Parameters chosen for tuning are 'n_estimators', 'max_depth', 'min_samples_split', 'min_samples_leaf', 'max_features', 'max_samples'.

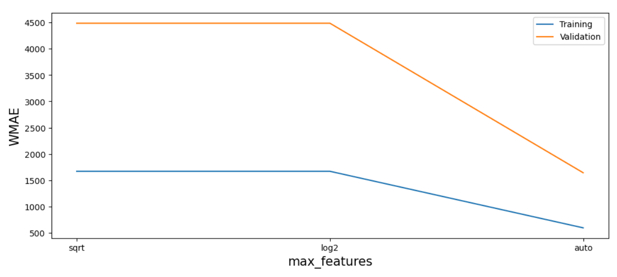
We fitted the model with different range of values for each chosen tuning parameters:

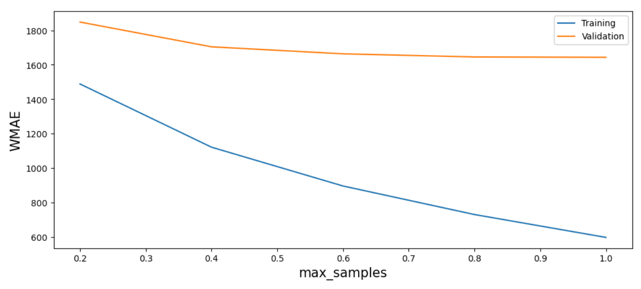












The following parameters are chosen: n_estimators: 100, max_depth: 50, min_samples_split: 2, min_samples_leaf: 1, max_features: 'auto', max_samples: 0.9 to fit the model and WMAE on testing data was found to be 1642.51

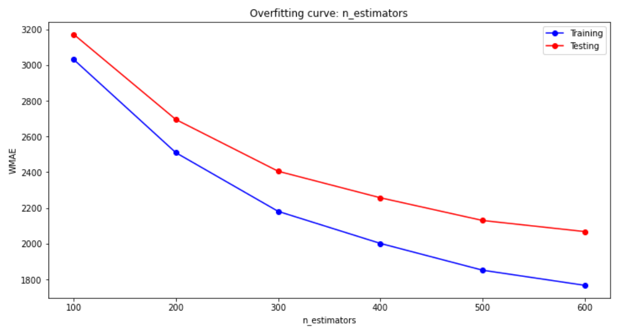
XGBoost Regressor

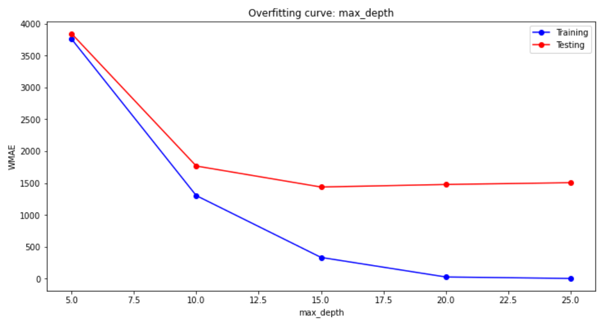
Using the XGBRegressor, the training and testing datasets were used to create a basic gradient boosting model and the WMAE was calculated.

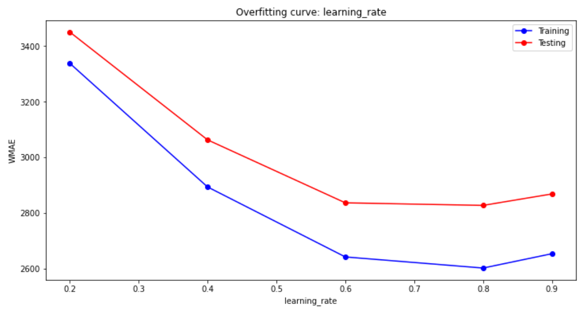
```
from xgboost import XGBRegressor
#Starting with default parameters of the model
gbm = XGBRegressor(random_state = 432, n_jobs = -1)
#Fitting the model
gbm.fit(X_train, y_train)
```

With the above model, the WMAE loss for training and testing data was found to be 3031.18 and 3171.92. We checked the important features and dropped the least important features to check if WMAE reduces. But, it increased the WMAE and we continued with the previous model.

Hyperparameter tuning of XGBRegressor:
The parameters chosen for tuning are ‘n_estimators’, ‘max_depth’ and ‘learning_rate’.
We fitted the model with different range of values for each chosen tuning parameters:







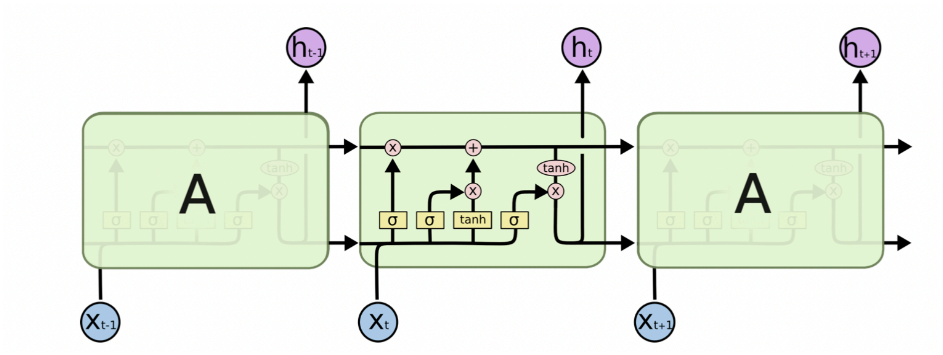
From the above testing of parameters, the best parameters which minimizes the error are found to be n_estimators = 600, max_depth = 20, learning_rate = 0.8. These parameters are varied individually but now, it should be varied simultaneously for further tuning of the model by the best combination of the values which minimizes the error. With trial and error method, best combination of the values were found to be n_estimators = 400, max_depth = 15, learning_rate = 0.35.
The model was fitted and WMAE loss for testing set was found to be 1456.36.

Long Short Term Memory (LSTM) Networks

Similar to how as you read this report, you understand each word based on your understanding of the previous words, Recurrent neural networks are neural networks with loops in them allowing the information they learn to persist.

A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of a neural network to use for such data.

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. LSTMs also have the chain-like structures seen in all RNNs, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a way as seen below. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate.



We treated the *Weekly Sales* of our data as a univariate time series forecasting problem and modeled two different LSTM models to compare the results. Before implementing our models, we had to prepare our data to make it compliant with these Deep Learning models.

Data Preparation

The LSTM model will learn a function that maps a sequence of past observations as input to an output observation. As such, the sequence of observations of the Weekly Sales must be transformed into multiple examples from which the LSTM can learn.

We divided the sequence into multiple input/output patterns called samples, where 25 steps were used as input and one time step was used as output for the one-step prediction that is being learned. The split_sequence() function implements this behavior and will split a given univariate sequence into multiple samples where each sample has a specified number of time steps and the output is a single time step.

Vanilla LSTM

A Vanilla LSTM is an LSTM model that has a single hidden layer of LSTM units, and an output layer used to make a prediction. We implemented a Vanilla LSTM model with 50 LSTM units in the hidden layer and an output layer that predicts a single numerical value. The model was fit using the efficient

```
...
# define model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

Adam version of stochastic gradient descent and optimized using the mean squared error, or ‘mse‘ loss function.

Once the model is defined, we fit it on the training dataset and computed the WMAE which came out to be around **4572.83**.

Stacked LSTM

We realized that one of the main drawbacks of LSTM models is that both larger models and hierarchical models (stacked LSTMs) are better to automatically learn (or “remember”) a larger temporal dependence. Larger models can learn more. Hence we considered using Stacked LSTM models.

Multiple hidden LSTM layers can be stacked one on top of another in what is referred to as a Stacked LSTM model. We also increased the number of hidden layers and the number of epochs. This led to lowering the WAME to **2384.63.**

However, the LSTM models came with many drawbacks. It mainly demanded a lot of computational capacity. LSTMs take a long time to learn complex dependencies. Although a much larger hidden layer (more memory units) and a much deeper network (stacked LSTMs) would be better suited for the given data, they need high memory-bandwidth because of linear layers present in each cell which the system usually fails to provide for. Thus, hardware-wise, LSTMs become quite inefficient.

Hence, we fit a time window based MLP to check if it outperforms the LSTM pure-AR approach.

MULTILAYER PERCEPTRON MODEL

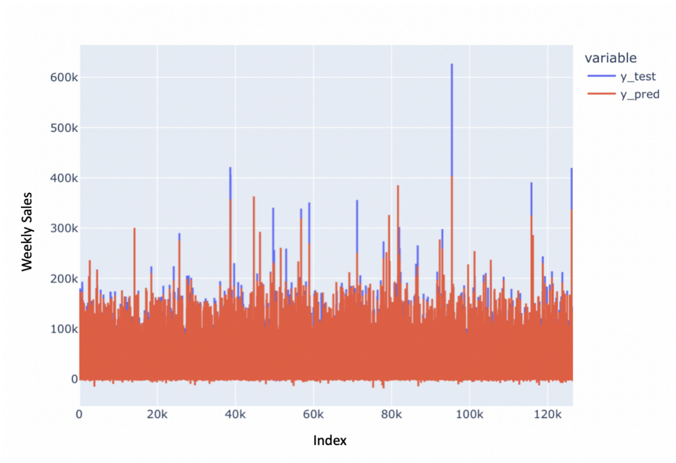
Multilayer Perceptrons, or MLPs for short, can be applied to time series forecasting. A challenge with using MLPs for time series forecasting is in the preparation of the data. Specifically, lag observations must be flattened into feature vectors. Before we model a univariate series for the Weekly Sales of the data, we need to prepare the dataset.

```
# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
 X, y = list(), list()
 for i in range(len(sequence)):
 # find the end of this pattern
 end_ix = i + n_steps
 # check if we are beyond the sequence
 if end_ix > len(sequence)-1:
 break
 # gather input and output parts of the pattern
 seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
 X.append(seq_x)
 y.append(seq_y)
 return array(X), array(y)
```

The MLP model will learn a function that maps a sequence of past observations as input to an output observation. As such, the sequence of observations must be transformed into multiple examples from which the model can learn. We define a split_sequence() function that implements this behavior and will split a given univariate sequence into multiple samples where each sample has a specified number of time steps and the output is a single time step.

We then implemented a simple Multiplayer Perceptron Model. The model is fit using the efficient Adam version of stochastic gradient descent and optimized using the mean squared error, or ‘mse‘, loss function.

Once the model is defined, we fit it on the training dataset and get a WMAE of **936.74** which is a significant drop as compared to the LSTM models. Below is a plot of the Weekly Sales of the test data and its prediction by the MLP model.



CONCLUSION

Working on the Walmart Sales dataset serves as a platform for us to review the various techniques and models present to tackle the time series forecasting. Below is a summary table of all the models we implemented and their WMAE scores.

We noticed that our dataset followed an autoregressive pattern, i.e., the Weekly Sales of the Walmart dataset at a given timestamp shows strong dependence on the weekly sales at previous timestamps. This is probably why the autoregressive models worked better on the dataset as opposed to traditional ML models/ the deep learning models.

Regression models like the Ridge regressor, random forest regressor and XGBoost regressor provided comparatively good WMAE scores on the test set. However, they performed badly when extrapolated (predicting outside the range of the data). Hence, we get a high error.

The Deep learning models like the LSTM and the Multilayer perceptron models posed difficulties in terms of their high memory bandwidth and computational demand. Deep learning models like these require a much larger hidden layer (more memory units) and a much deeper network (stacked LSTMs) for the given data, they need high memory-bandwidth because of linear layers present in each cell which most systems usually fails to provide for. Thus, hardware-wise, LSTMs and MLPs become quite an inefficient option.

The models that performed the best were the Autoregressive models, especially SARIMA. Since we were working with autoregressive data over a period of about three years, the data showed a cyclic (repetitive pattern) with respect to its sales. Cyclic patterns like these are picked up well with models like SARIMA to detect seasonality in the data. This is probably why the model performed the best for our dataset.

Model	**WMAE**
Ridge Regressor	14829.79
Random Forest Regressor	1642.51
XGBoost Regressor	1456.36
SARIMA	**438.62**
Vanilla LSTM	4572.83

Stacked LSTM	2384.63
Multilayer Perceptron Model	936.74

Kaggle Competition:

For this project, we used the "Walmart Sales Forecasting" Kaggle competition as our benchmark reference. The competition was posted 9 years ago and has over 688 participants. The competition is evaluated on a weighted mean absolute error (WMAE). The average WMAEs submitted by the top 15 submissions is around the 500 level mark. Our SARIMA model, aided by the data cleaning and preparation done, gave us a WMAE of 438.62 which is comparable to the top submissions posted over the last 9 years.

To conclude, we this sales forecasting we can also answer various business questions, forecast revenue, determine seasonal demands and manage inventories accordingly, and also protect from money loss by achieving the sales targets.

CHALLENGES FACED & FUTURE SCOPE

The following are the challenges we faced while training the model:

1. Hyperparameter tuning takes a lot of time to run and requires very high computational capabilities so essential parameters were tuned individually.
2. Identifying the seasonality in the data and choosing the best possible method to eliminate it.
3. Identifying key features in prediction to avoid noise and overfitting.

When it comes to future scope, we can combine external information like the dollar index, oil price and prominent news about Walmart might have had an impact on departmental sales. We can also extract data from platforms like Twitter or Google Trends. This gives us a larger dataset and implementing the same SARIMA model for a larger dataset will give more accurate predictions, because our dataset consisted of 2.75 years of data, with limited seasonality and no trends that might exist.

REFERENCES

1. Bryan Lim and Stefan Zohren (2020), “Time-series forecasting with deep learning: a survey”, Oxford-Man Institute for Quantitative Finance, Department of Engineering Science, University of Oxford
2. Crown, M. (2016). “Weekly sales forecasts using non-seasonal arima models”, http : / /mxcrown.com/walmart-sales-forecasting/
3. Heng Zhao, Xumin Zuo and Peisong Lin, “Sales forecasting for Chemical Products by Using SARIMA Model”, ICBDE '22: Proceedings of the 5th International Conference on Big Data and Education
4. Mesut Gumus and Mustafa S. Kiran, “Crude Oil Price Forecasting Using XGBoost", 2nd International Conference on Computer Science and Engineering
5. Yi Niu (2020). “Walmart Sales Forecasting using XGBoost algorithm and Feature engineering”, 2020 International Conference on Big Data & Artificial Intelligence & Software Engineering
6. Brownlee, Jason. Deep learning for time series forecasting: predict the future with MLPs, CNNs and LSTMs in Python. Machine Learning Mastery, 2018.