TITLE: Walmart Store’s Sales Forecasting

Team Members:

Name	**SapID**
Dhruv Bohara	60004180021
Nidhi Panchal	60013180013
Rut Vyas	60002180085
Sarvesh Navare	60004180096
Shreya Nayak	60002180107
Uditi Namdev	60004180111

University Name: DJ Sanghvi College of Engineering

Problem Statement:

This is an interesting data science problem that involves forecasting future sales across various departments within different Walmart outlets. The challenging aspect of this data science project is to forecast the sales on 4 major holidays- Labor Day, Christmas, Thanksgiving and Super Bowl. The selected holiday markdown events are the ones when Walmart makes the highest sales and by forecasting sales for these events they want to ensure that there is sufficient product supply to meet the demand. The dataset contains various details like markdown discounts, consumer price index, whether the week was a holiday, temperature, store size, store type and unemployment rate.

Solution Approach:

Strategic planning demands to forecast future sales of a company. In this project we are working on market supergiant Walmart’s dataset. By analyzing approximately 45 stores’ data between 2010 and 2012, we plan to roughly predict the sales on 4 major holidays of the United States- Labor Day, Christmas,Thanksgiving and Super Bowl. Every Departmental store chain like Walmart wants to predict the store sales in the nearby future so that inventory planning can be done. Along with that, sales prediction helps to increase/decrease store staff based on the rush (More sales can mean more customers are coming to the stores). Also, it is always a good idea to do sales and revenue forecasting to better understand the company's cash-flows and overall growth.We plan to use various machine learning algorithms to predict the value of the new data points and then plot them using several techniques of visual representation like tableau and matplotlib modules.Exploratory analysis is done using tableau and python will be used to preprocess the data and deploy models to get accurate results for each model.

Tools and Technologies Used:

1. **Python libraries:**

- Numpy:NumPy is a Python library used for working with arrays.It also has functions for working in the domain of linear algebra, fourier transform, and matrices.
- Matplotlib:Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- Seaborn: Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- Pandas: pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool,built on top of the Python programming language.
- Sklearn: It is a machine learning library that features various algorithms like support vector machines, random forests, and k-neighbours, and it also supports Python numericals and scientific libraries like NumPy and SciPy .

2. **Tableau**

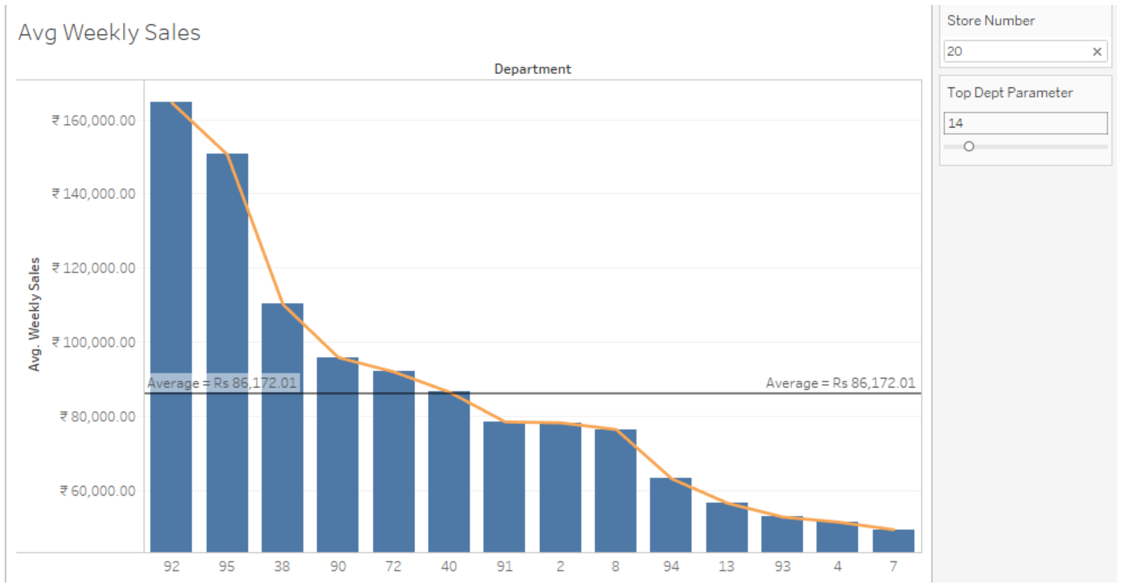
Dataset Used:

test.csv features.csv stores.csv

1.) Exploratory Data Analysis

Exploring the Data to get some inferences:

A]

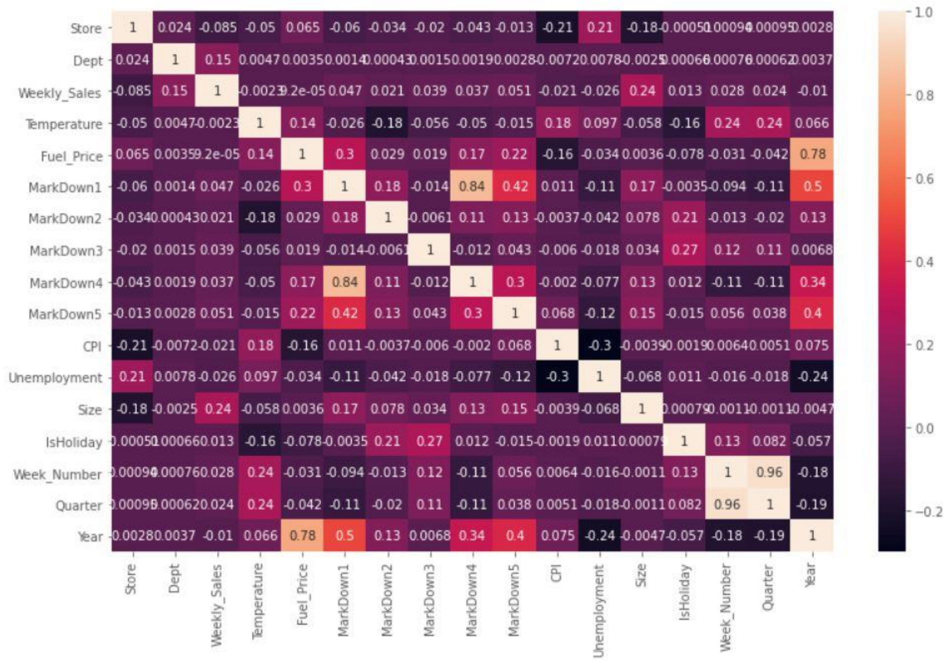


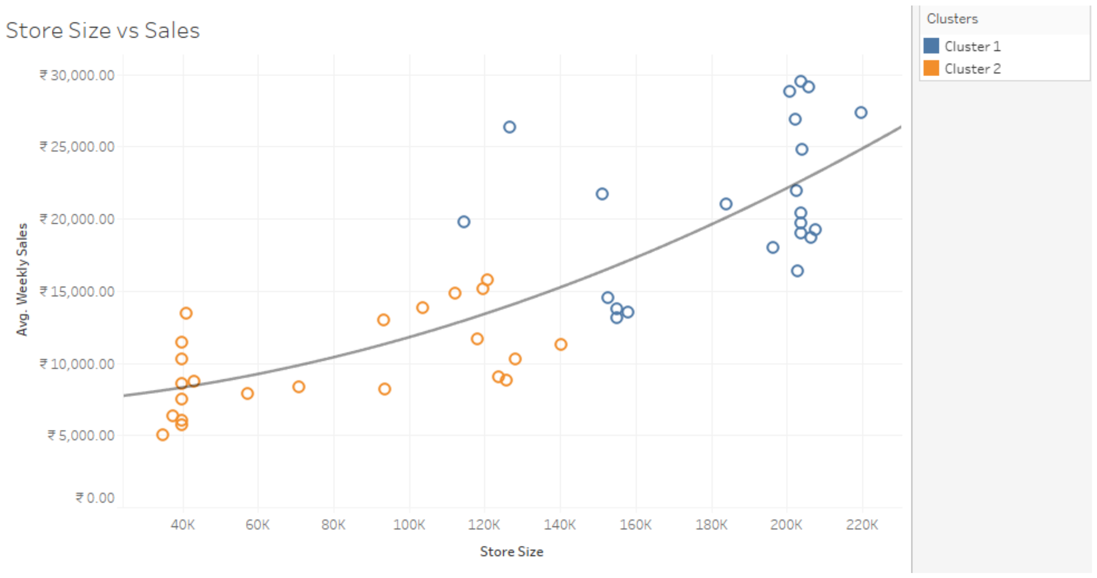
Store Number
20

Top Dept Parameter
14

The Top department parameter helps us to shortlist the departments in a particular store (which can be selected by entering the store number) having the highest average weekly sales. This is useful as it gives us information about which departments have products that are popular among people and fast selling.

B]

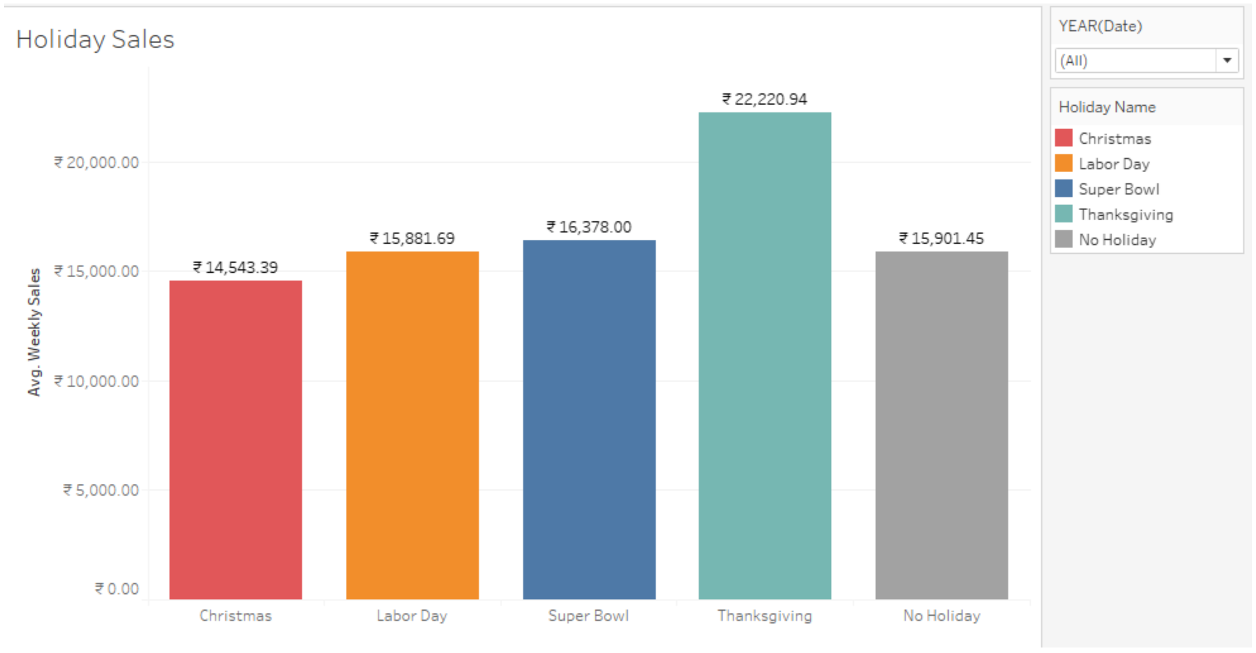




A correlation matrix was made to find out the correlation between different variables of the dataset. From this, we infer that the Weekly Sales had the most correlation of 0.24 with the size of the store. To further check how they are correlated, we plotted a trend line for the same.

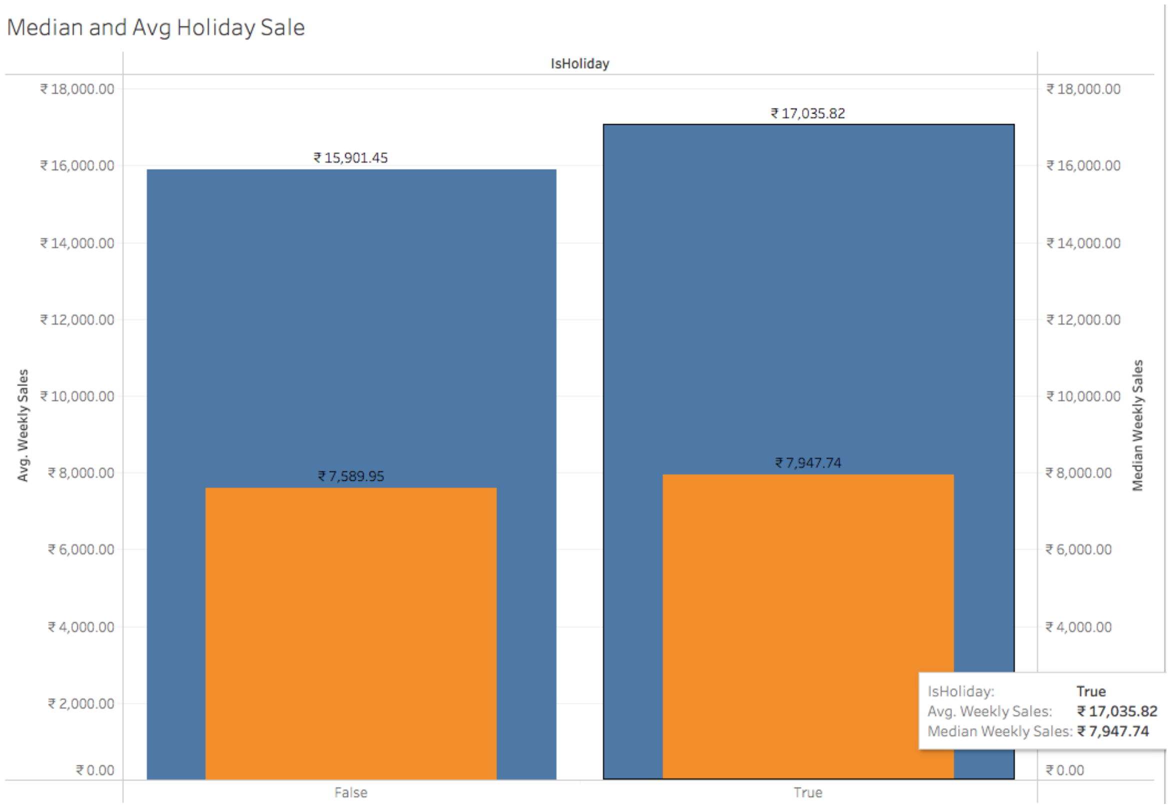
The dependent variable is placed on the y-axis (average weekly sales) and the independent variable is placed on the x-axis (Store Size). It can be observed that the average weekly sales increases with the increase in the store size. Here the different stores of Walmart vary in size, which is found to be an important factor in contributing to the sales.

C]



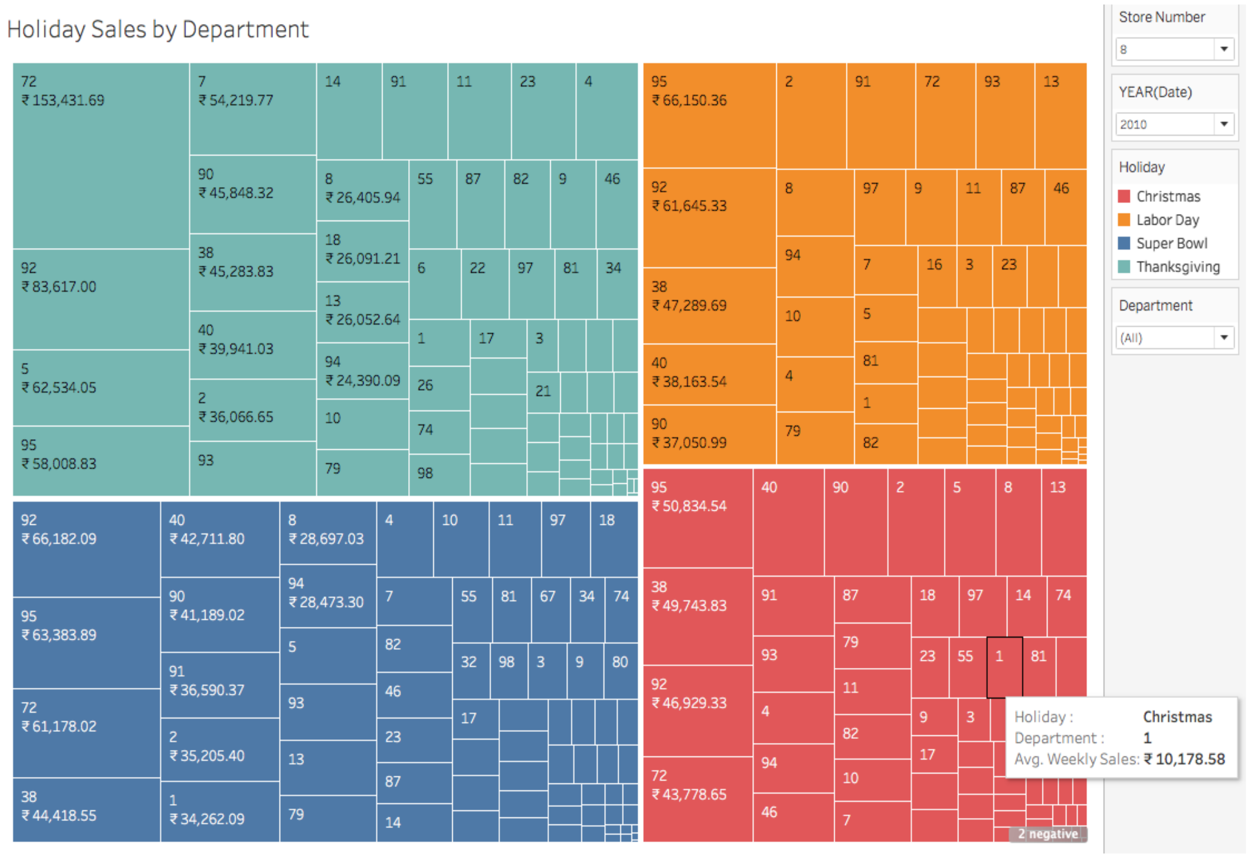
The Holiday Sales are useful as we can infer on which holiday there were maximum sales. On Thanksgiving the average weekly sales were maximum. This could also be due to the fact that there are many Sales during Thanksgiving such as the Black Friday Sales which attract a lot of customers to the stores.

D]



The huge difference of almost 10k between the median (in orange) and mean(in blue) tells us that some stores and their departments are doing considerably better than others which has caused this rift between the average and the median value of sales. Hence, we need a more detailed view to understand the sales in the upper and lower quartiles.

E]



Store Number: 8

YEAR(Date): 2010

Holiday

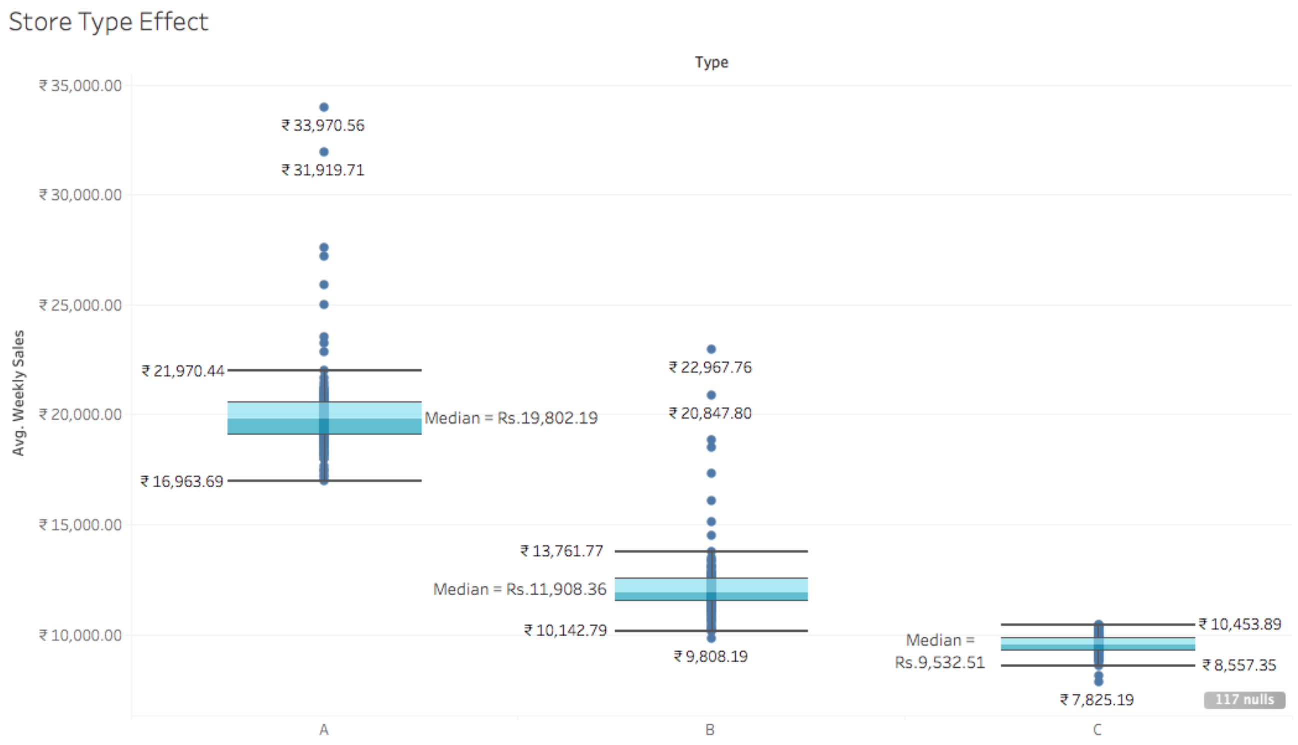
Department: (All)

Holiday : Christmas
Department : 1
Avg. Weekly Sales: ₹ 10,178.58

For store 8

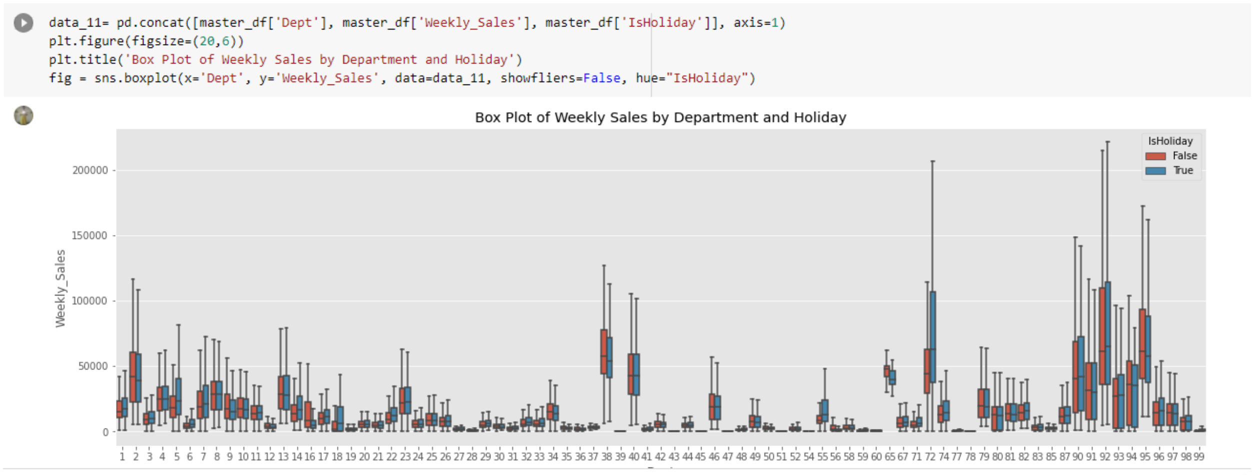
On selecting the store number, we can see the sales of the store for all the four holidays department wise. These holiday sales are highlighted in different colours for every holiday. This helps us to simultaneously compare the sales across all holidays for different departments and understand the stocking needs of the particular store at that time in the future.

F]



Based on the three store types A, B and C we can infer the overall weekly sales of each store type. We can clearly see that the means of Store A, B and C are in decreasing order respectively. This tells us that for whatever reason a store which is opened as Type A will definitely do better than Type B and C stores and a store opened as Type B will most probably do better than Type C store.

G]



```
data_11= pd.concat([master_df['Dept'], master_df['Weekly_Sales'], master_df['IsHoliday']], axis=1)
plt.figure(figsize=(20,6))
plt.title('Box Plot of Weekly Sales by Department and Holiday')
fig = sns.boxplot(x='Dept', y='Weekly_Sales', data=data_11, showfliers=False, hue="IsHoliday")
```

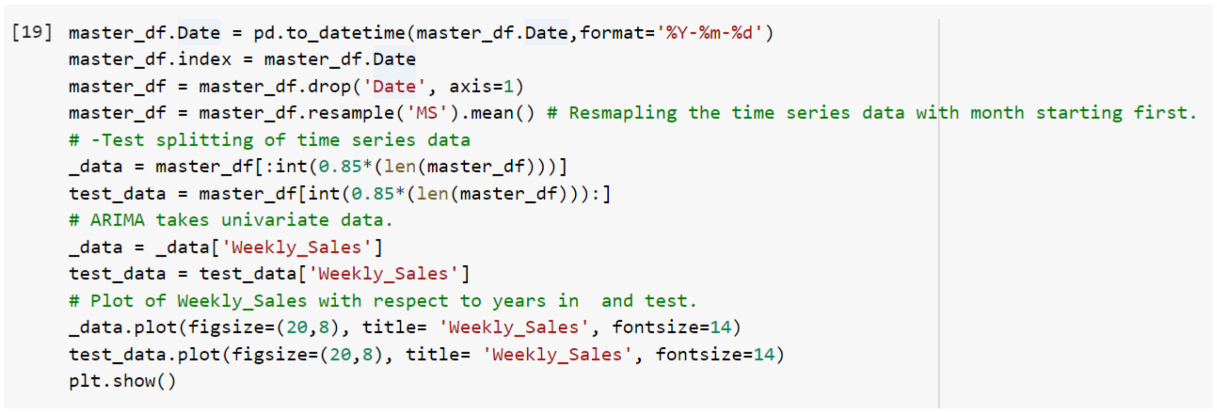
This chart shows us the comparative sales of each department on holidays and non-holidays.

2.) Forecasting and Prediction of Sales

Sales Forecasting using different Models:

1) ARIMA Model

The data has been divided into test_data (85%) and train_data (15%) for the ARIMA model.



```
[19] master_df.Date = pd.to_datetime(master_df.Date,format='%Y-%m-%d')
     master_df.index = master_df.Date
     master_df = master_df.drop('Date', axis=1)
     master_df = master_df.resample('MS').mean() # Resmapling the time series data with month starting first.
     # -Test splitting of time series data
     _data = master_df[:int(0.85*(len(master_df)))]
     test_data = master_df[int(0.85*(len(master_df))):]
     # ARIMA takes univariate data.
     _data = _data['Weekly_Sales']
     test_data = test_data['Weekly_Sales']
     # Plot of Weekly_Sales with respect to years in  and test.
     _data.plot(figsize=(20,8), title= 'Weekly_Sales', fontsize=14)
     test_data.plot(figsize=(20,8), title= 'Weekly_Sales', fontsize=14)
     plt.show()
```

ARIMA, short for 'Auto Regressive Integrated Moving Average' is actually a class of models that 'explains' a given time series based on its own past values, that is, its own lags and the lagged forecast errors, so that equation can be used to forecast future values. Any 'non-seasonal' time series that exhibits patterns and is not a random white noise can be modeled with ARIMA models.

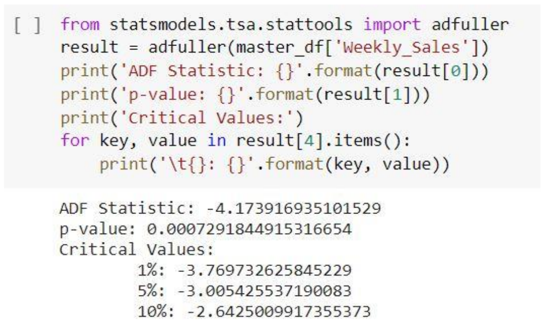
An ARIMA model is characterized by 3 terms: p, d, q where,

'p' is the order of the 'Auto Regressive' (AR) term. It refers to the number of lags of Y to be used as predictors.

And 'q' is the order of the 'Moving Average' (MA) term. It refers to the number of lagged forecast errors that should go into the ARIMA Model.

d is the number of differencing required to make the time series stationary. The value of d, therefore, is the minimum number of differencing needed to make the series stationary. And if the time series is already stationary, then d = 0.

To check if the series is stationary using the Augmented Dickey Fuller test (adfuller()), from the statsmodels package.

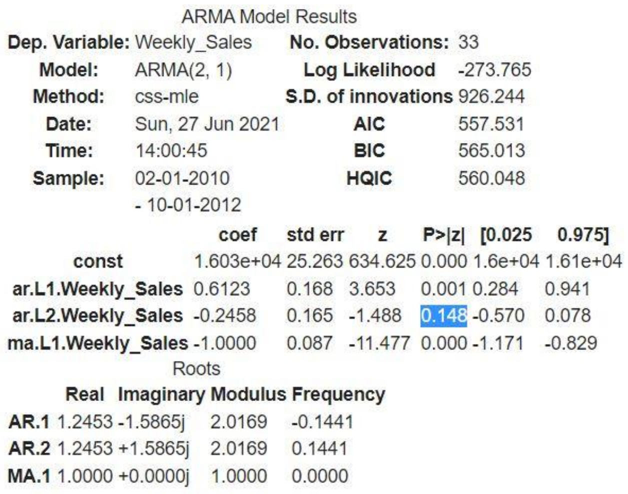


```
[ ] from statsmodels.tsa.stattools import adfuller
    result = adfuller(master_df['Weekly_Sales'])
    print('ADF Statistic: {}'.format(result[0]))
    print('p-value: {}'.format(result[1]))
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t{}: {}'.format(key, value))
```

```
ADF Statistic: -4.173916935101529
p-value: 0.0007291844915316654
Critical Values:
        1%: -3.769732625845229
        5%: -3.005425537190083
        10%: -2.6425009917355373
```

From this we can interpret,

Since the ADF statistic is less than 5% and the p-value < 0.05, we can reject the null hypothesis and infer that the time series is stationary.



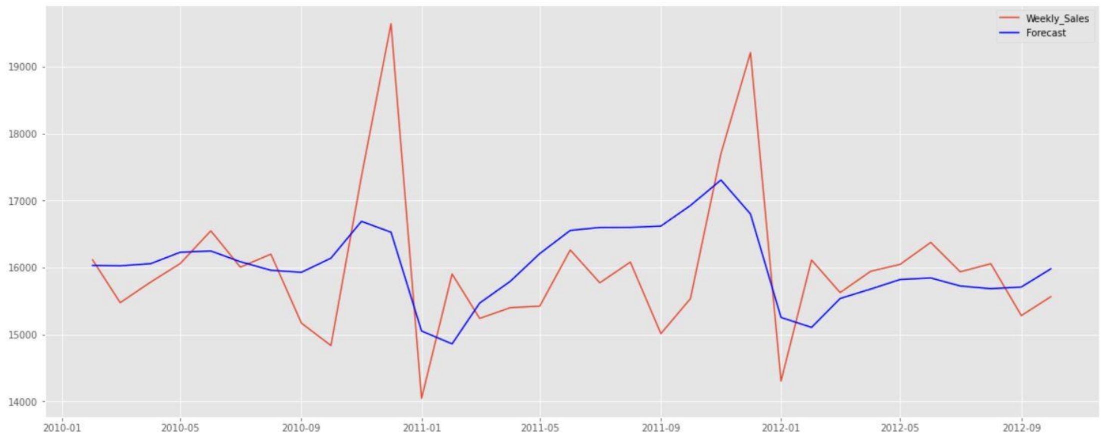
ARMA Model Results

Dep. Variable:	Weekly_Sales	**No. Observations:**	33
Model:	ARMA(2, 1)	**Log Likelihood**	-273.765
Method:	css-mle	**S.D. of innovations**	926.244
Date:	Sun, 27 Jun 2021	**AIC**	557.531
Time:	14:00:45	**BIC**	565.013
Sample:	02-01-2010	**HQIC**	560.048
	- 10-01-2012		

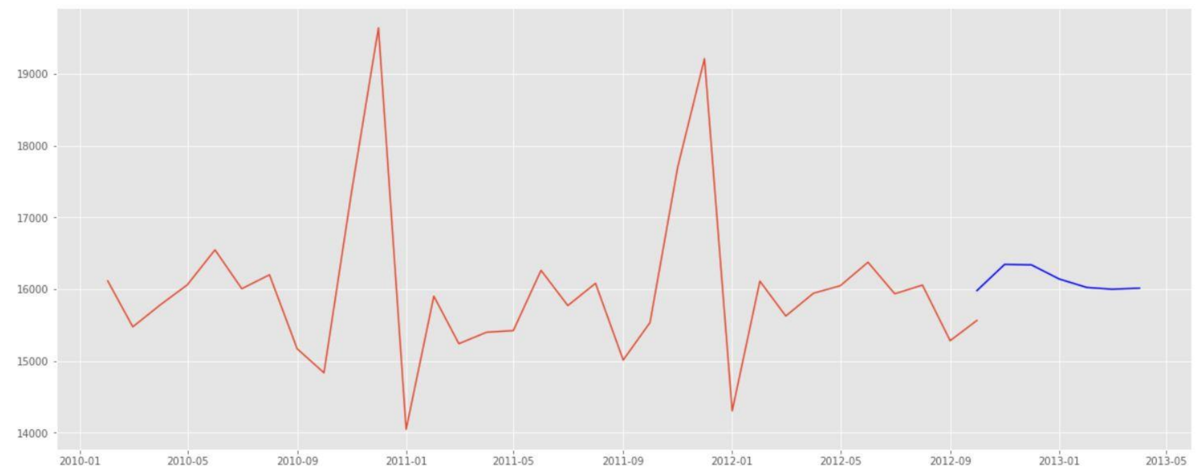
	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	1.603e+04	25.263	634.625	0.000	1.6e+04	1.61e+04
ar.L1.Weekly_Sales	0.6123	0.168	3.653	0.001	0.284	0.941
ar.L2.Weekly_Sales	-0.2458	0.165	-1.488	0.148	-0.570	0.078
ma.L1.Weekly_Sales	-1.0000	0.087	-11.477	0.000	-1.171	-0.829

Roots

	Real	**Imaginary**	**Modulus**	**Frequency**
AR.1	1.2453	-1.5865j	2.0169	-0.1441
AR.2	1.2453	+1.5865j	2.0169	0.1441
MA.1	1.0000	+0.0000j	1.0000	0.0000



Forecasting using ARIMA model:

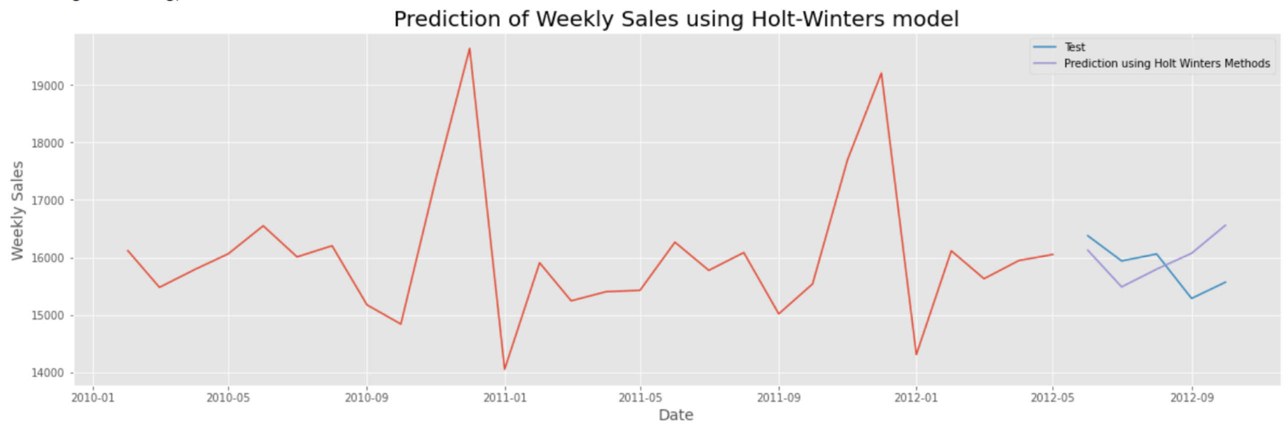


From this we can conclude that the ARIMA model does not accurately detect seasonality in the dataset. Hence, the forecast made by the ARIMA model may not be accurate as it might have some peaks which the model failed to identify due to the presence of seasonality in our data.

2) Holt-Winter Model

Holt-Winters forecasting is a way to model and predict the behavior of a sequence of values over time-a time series.

The three aspects of the time series behavior—value, trend, and seasonality—are expressed as three types of exponential smoothing, so Holt-Winters is called triple exponential smoothing. The model predicts a current or future value by computing the combined effects of these three influences.



The sales are predicted using Holt-Winters model. The errors for Holt-Winters models were calculated to test the accuracy of the prediction against the test data.

```
Mean Squared Error (MSE) of Holt-Winters:  388409.6112769923
Root Mean Squared Error (RMSE) of Holt-Winters:  623.2251690015354
Mean Absolute Deviation (MAD) of Holt-Winters:  549.8500311065163
```

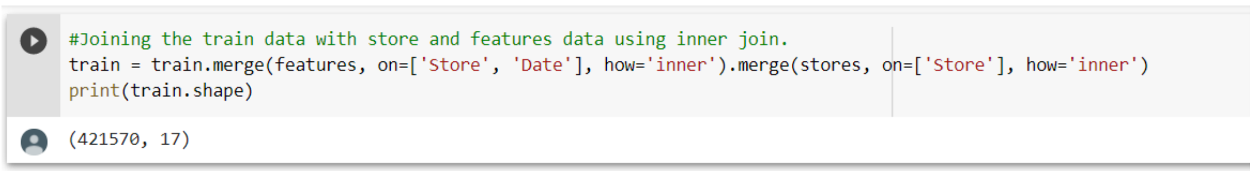
Observing the errors calculated, we can infer that the model does not provide an accurate representation of actual sales and hence the model is not very accurate for forecasting the future sales.

Prediction of Sales on Holidays

Thus after analysing the dataset, finding out trends and seasonality, we turn to predicting the sales on the 4 holidays, namely, Labor Day, Christmas, Thanksgiving and Super Bowl. We chose a predictor model that gives the least WMAE (Weighted Mean Absolute Error). Out of the analysed predictor models, we found that Random Forest Regression model has the lowest WMAE and thus is the best choice of predictor model in this case.

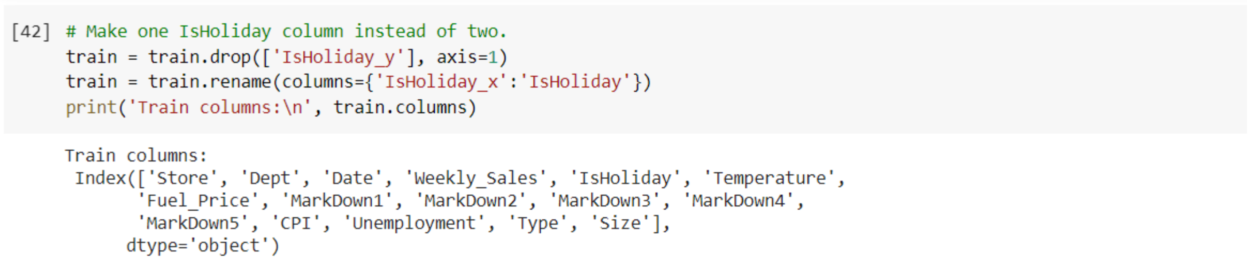
Random Forest predictor is trained using the dataset present, but before that the data is preprocessed (cleaned, integrated and transformed into suitable format). The dataset present is divided into 70:30 ratio as training and testing data respectively and fed into the Random Forest Regressor model to train the model. Then a test file (test.csv) is imported to test the model and weekly sales for the dates present in the test file are generated using our predictor model. The final result including store number, department , Date and its weekly sales is stored in a separate csv file.

- Preprocessing



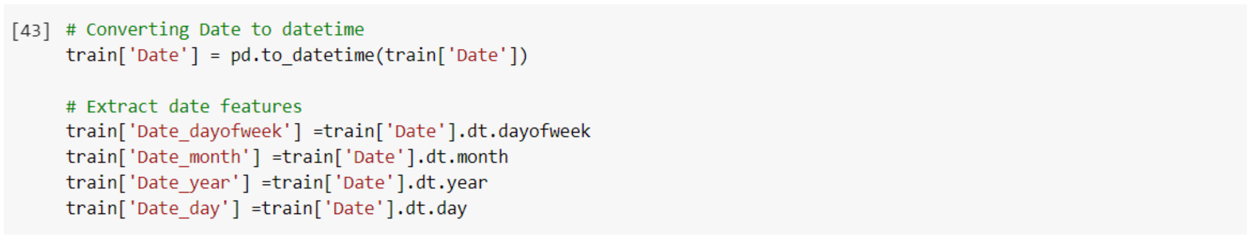
```
#Joining the train data with store and features data using inner join.
train = train.merge(features, on=['Store', 'Date'], how='inner').merge(stores, on=['Store'], how='inner')
print(train.shape)
```

```
(421570, 17)
```

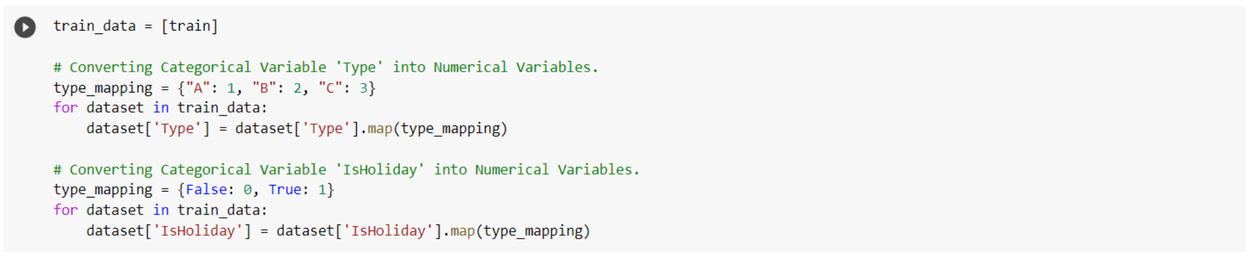


```
[42] # Make one IsHoliday column instead of two.
     train = train.drop(['IsHoliday_y'], axis=1)
     train = train.rename(columns={'IsHoliday_x':'IsHoliday'})
     print('Train columns:\n', train.columns)
```

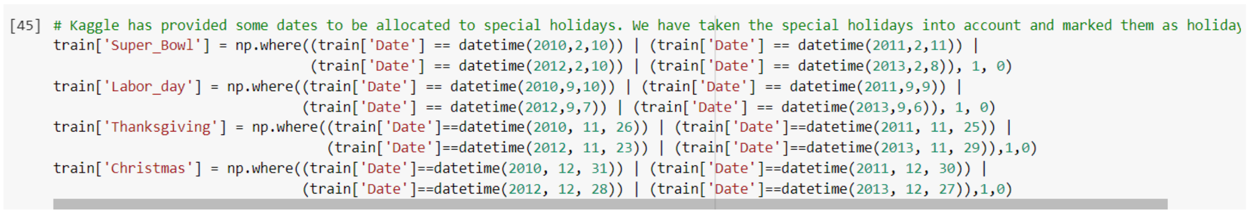
```
Train columns:
 Index(['Store', 'Dept', 'Date', 'Weekly_Sales', 'IsHoliday', 'Temperature',
       'Fuel_Price', 'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4',
       'MarkDown5', 'CPI', 'Unemployment', 'Type', 'Size'],
      dtype='object')
```



```
[43] # Converting Date to datetime
     train['Date'] = pd.to_datetime(train['Date'])

     # Extract date features
     train['Date_dayofweek'] =train['Date'].dt.dayofweek
     train['Date_month'] =train['Date'].dt.month
     train['Date_year'] =train['Date'].dt.year
     train['Date_day'] =train['Date'].dt.day
```

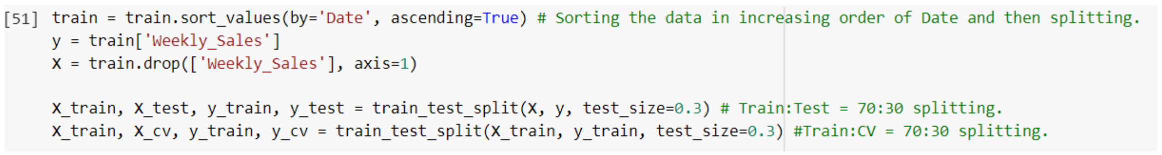


```
train_data = [train]

# Converting Categorical Variable 'Type' into Numerical Variables.
type_mapping = {"A": 1, "B": 2, "C": 3}
for dataset in train_data:
    dataset['Type'] = dataset['Type'].map(type_mapping)

# Converting Categorical Variable 'IsHoliday' into Numerical Variables.
type_mapping = {False: 0, True: 1}
for dataset in train_data:
    dataset['IsHoliday'] = dataset['IsHoliday'].map(type_mapping)
```



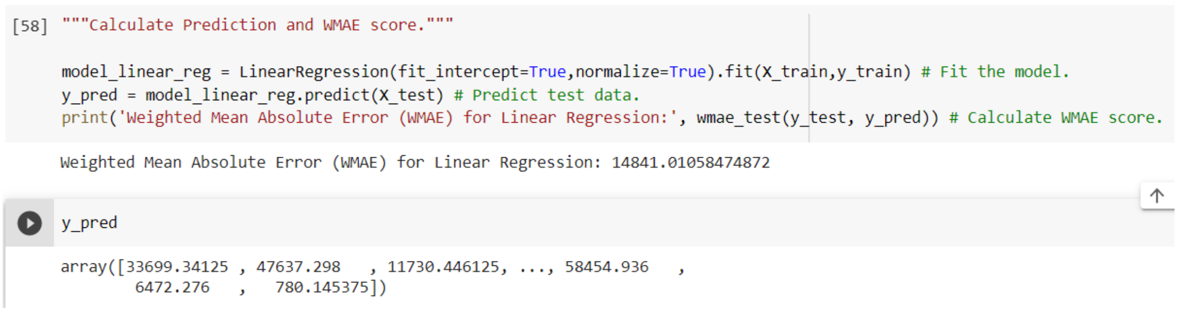
```
[45] # Kaggle has provided some dates to be allocated to special holidays. We have taken the special holidays into account and marked them as holiday
     train['Super_Bowl'] = np.where((train['Date'] == datetime(2010,2,10)) | (train['Date'] == datetime(2011,2,11)) |
                                    (train['Date'] == datetime(2012,2,10)) | (train['Date'] == datetime(2013,2,8)), 1, 0)
     train['Labor_day'] = np.where((train['Date'] == datetime(2010,9,10)) | (train['Date'] == datetime(2011,9,9)) |
                                   (train['Date'] == datetime(2012,9,7)) | (train['Date'] == datetime(2013,9,6)), 1, 0)
     train['Thanksgiving'] = np.where((train['Date']==datetime(2010, 11, 26)) | (train['Date']==datetime(2011, 11, 25)) |
                                     (train['Date']==datetime(2012, 11, 23)) | (train['Date']==datetime(2013, 11, 29)),1,0)
     train['Christmas'] = np.where((train['Date']==datetime(2010, 12, 31)) | (train['Date']==datetime(2011, 12, 30)) |
                                   (train['Date']==datetime(2012, 12, 28)) | (train['Date']==datetime(2013, 12, 27)),1,0)
```

- Splitting of data



```
[51] train = train.sort_values(by='Date', ascending=True) # Sorting the data in increasing order of Date and then splitting.
     y = train['Weekly_Sales']
     X = train.drop(['Weekly_Sales'], axis=1)

     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # Train:Test = 70:30 splitting.
     X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.3) #Train:CV = 70:30 splitting.
```

1) Prediction using Linear Regression model



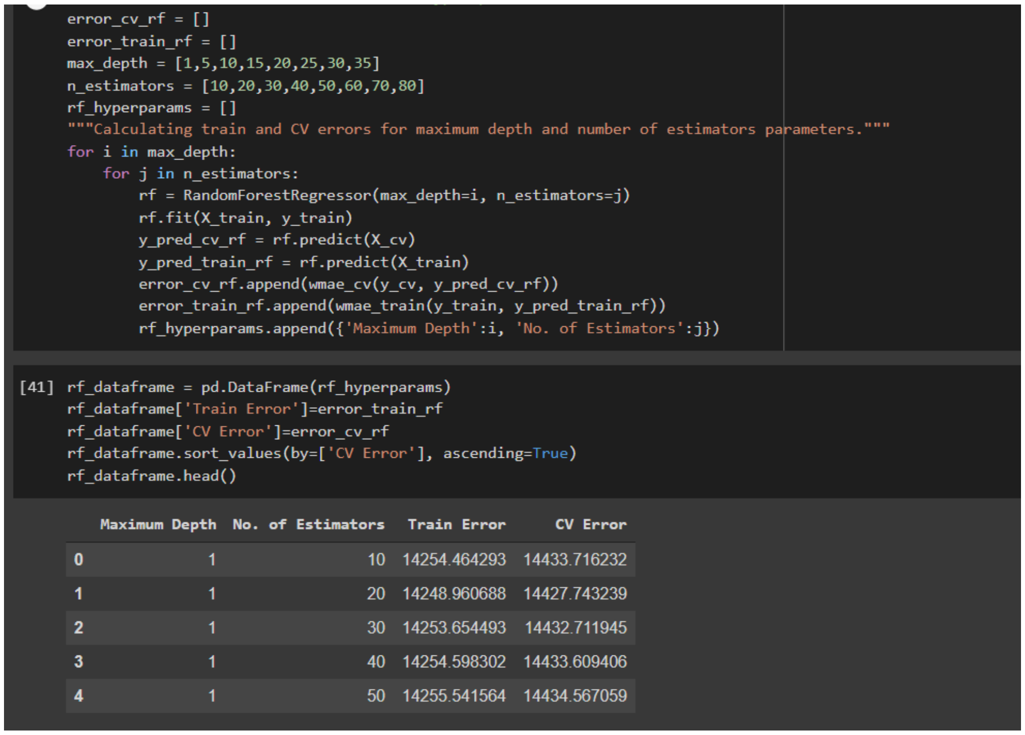
```
[58] """Calculate Prediction and WMAE score."""

     model_linear_reg = LinearRegression(fit_intercept=True,normalize=True).fit(X_train,y_train) # Fit the model.
     y_pred = model_linear_reg.predict(X_test) # Predict test data.
     print('Weighted Mean Absolute Error (WMAE) for Linear Regression:', wmae_test(y_test, y_pred)) # Calculate WMAE score.

     Weighted Mean Absolute Error (WMAE) for Linear Regression: 14841.01058474872
```

```
y_pred

array([33699.34125 , 47637.298   , 11730.446125, ..., 58454.936   ,
        6472.276   ,   780.145375])
```

We observe that the WMAE value for linear regression model is very high and Random Forest Regression model has the lowest WMAE value.

2) Prediction using Random Forest Regression Model

The best hyper parameter values were found by running the following code and by calculating the training and cross validation errors for maximum depth and n_estimators parameters:

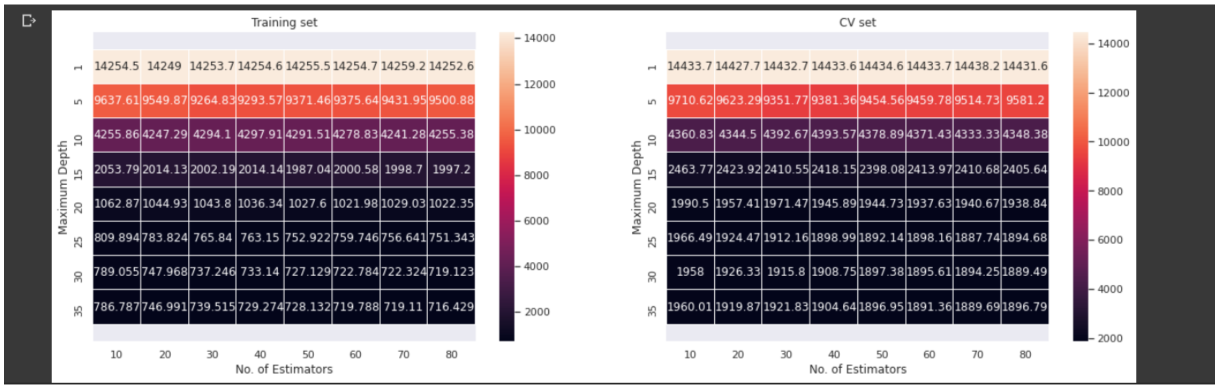


```python
error_cv_rf = []
error_train_rf = []
max_depth = [1,5,10,15,20,25,30,35]
n_estimators = [10,20,30,40,50,60,70,80]
rf_hyperparams = []
"""Calculating train and CV errors for maximum depth and number of estimators parameters."""
for i in max_depth:
    for j in n_estimators:
        rf = RandomForestRegressor(max_depth=i, n_estimators=j)
        rf.fit(X_train, y_train)
        y_pred_cv_rf = rf.predict(X_cv)
        y_pred_train_rf = rf.predict(X_train)
        error_cv_rf.append(wmae_cv(y_cv, y_pred_cv_rf))
        error_train_rf.append(wmae_train(y_train, y_pred_train_rf))
        rf_hyperparams.append({'Maximum Depth':i, 'No. of Estimators':j})
```

```python
[41] rf_dataframe = pd.DataFrame(rf_hyperparams)
     rf_dataframe['Train Error']=error_train_rf
     rf_dataframe['CV Error']=error_cv_rf
     rf_dataframe.sort_values(by=['CV Error'], ascending=True)
     rf_dataframe.head()
```

	Maximum Depth	No. of Estimators	Train Error	CV Error
0	1	10	14254.464293	14433.716232
1	1	20	14248.960688	14427.743239
2	1	30	14253.654493	14432.711945
3	1	40	14254.598302	14433.609406
4	1	50	14255.541564	14434.567059

Then, we plotted the heat maps for each (max depth, no. of estimators) and found that the least error is found for (35,80) for the training set. Thus, these were the chosen hyper parameters for the Random Forest Regressor.



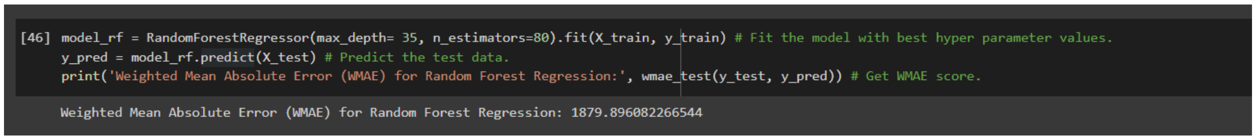
Training set

Maximum Depth	10	20	30	40	50	60	70	80
1	14254.5	14249	14253.7	14254.6	14255.5	14254.7	14259.2	14252.6
5	9637.61	9549.87	9264.83	9293.57	9371.46	9375.64	9431.95	9500.88
10	4255.86	4247.29	4294.1	4297.91	4291.51	4278.83	4241.28	4255.38
15	2053.79	2014.13	2002.19	2014.14	1987.04	2000.58	1998.7	1997.2
20	1062.87	1044.93	1043.8	1036.34	1027.6	1021.98	1029.03	1022.35
25	809.894	783.824	765.84	763.15	752.922	759.746	756.641	751.343
30	789.055	747.968	737.246	733.14	727.129	722.784	722.324	719.123
35	786.787	746.991	739.515	729.274	728.132	719.788	719.11	716.429

CV set

Maximum Depth	10	20	30	40	50	60	70	80
1	14433.7	14427.7	14432.7	14433.6	14434.6	14433.7	14438.2	14431.6
5	9710.62	9623.29	9351.77	9381.36	9454.56	9459.78	9514.73	9581.2
10	4360.83	4344.5	4392.67	4393.57	4378.89	4371.43	4333.33	4348.38
15	2463.77	2423.92	2410.55	2418.15	2398.08	2413.97	2410.68	2405.64
20	1990.5	1957.41	1971.47	1945.89	1944.73	1937.63	1940.67	1938.84
25	1966.49	1924.47	1912.16	1898.99	1892.14	1898.16	1887.74	1894.68
30	1958	1926.33	1915.8	1908.75	1897.38	1895.61	1894.25	1889.49
35	1960.01	1919.87	1921.83	1904.64	1896.95	1891.36	1889.69	1896.79

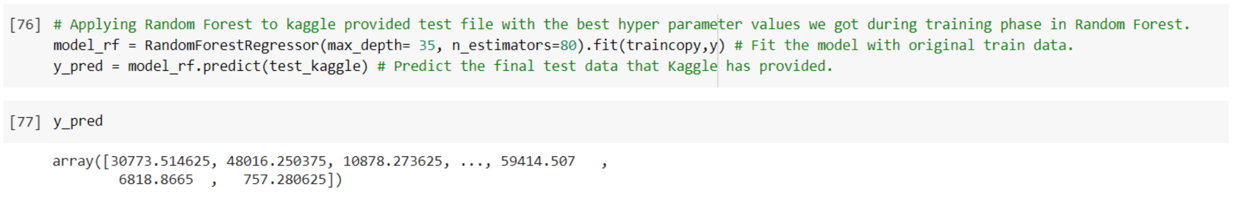
The WMAE of Random Forest regression comes out to be **1879.9** and is the least among all predictor models and thus it was chosen for prediction of weekly sales values.



```python
[46] model_rf = RandomForestRegressor(max_depth= 35, n_estimators=80).fit(X_train, y_train) # Fit the model with best hyper parameter values.
     y_pred = model_rf.predict(X_test) # Predict the test data.
     print('Weighted Mean Absolute Error (WMAE) for Random Forest Regression:', wmae_test(y_test, y_pred)) # Get WMAE score.

     Weighted Mean Absolute Error (WMAE) for Random Forest Regression: 1879.896082266544
```

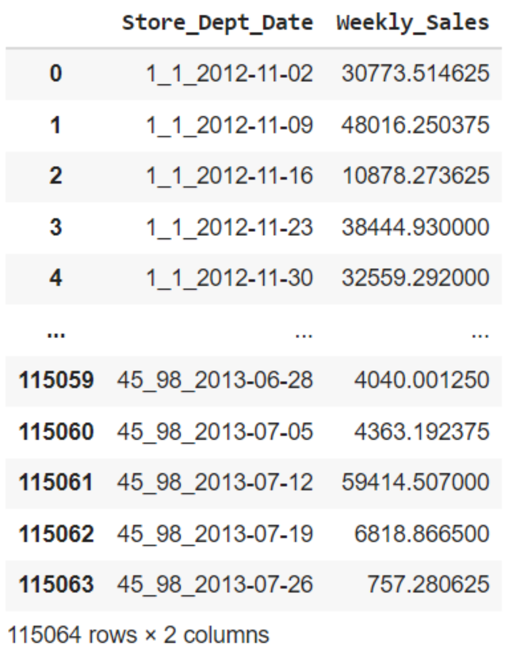
3) Predicting weekly sales values for holidays

Test.csv, which was stored in test_kaggle, was used to test our model and the y_pred variable was used to hold the weekly sales values predicted by our model. Weekly sales were then merged in a single dataframe with records of test_kaggle and the final dataframe was exported as Weekly Sales Prediction.csv.



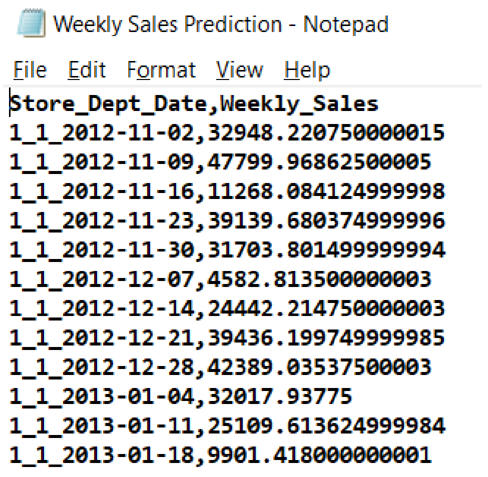
```
[76] # Applying Random Forest to kaggle provided test file with the best hyper parameter values we got during training phase in Random Forest.
     model_rf = RandomForestRegressor(max_depth= 35, n_estimators=80).fit(traincopy,y) # Fit the model with original train data.
     y_pred = model_rf.predict(test_kaggle) # Predict the final test data that Kaggle has provided.

[77] y_pred

     array([30773.514625, 48016.250375, 10878.273625, ..., 59414.507    ,
             6818.8665  ,   757.280625])
```

Hence, the weekly sales predicted came out to be:



	Store_Dept_Date	Weekly_Sales
0	1_1_2012-11-02	30773.514625
1	1_1_2012-11-09	48016.250375
2	1_1_2012-11-16	10878.273625
3	1_1_2012-11-23	38444.930000
4	1_1_2012-11-30	32559.292000
...	...	...
115059	45_98_2013-06-28	4040.001250
115060	45_98_2013-07-05	4363.192375
115061	45_98_2013-07-12	59414.507000
115062	45_98_2013-07-19	6818.866500
115063	45_98_2013-07-26	757.280625

115064 rows × 2 columns



```
Store_Dept_Date,Weekly_Sales
1_1_2012-11-02,32948.220750000015
1_1_2012-11-09,47799.96862500005
1_1_2012-11-16,11268.084124999998
1_1_2012-11-23,39139.680374999996
1_1_2012-11-30,31703.801499999994
1_1_2012-12-07,4582.813500000003
1_1_2012-12-14,24442.214750000003
1_1_2012-12-21,39436.199749999985
1_1_2012-12-28,42389.03537500003
1_1_2013-01-04,32017.93775
1_1_2013-01-11,25109.613624999984
1_1_2013-01-18,9901.418000000001
```

Result of Prediction of Sales:

Weekly Sales Prediction.csv

Holiday_Sales_Predict ion.csv

Predicted Sales values for the holidays: Superbowl, Thanksgiving and Christmas for the year 2012. This prediction is done for all the 98 departments in all the 45 stores of Walmart

Implementation in Python:

Python_Implementati
on.pdf

References

[1] ARIMA Model - Complete Guide to Time Series Forecasting in Python | ML+ (machinelearningplus.com)

[2] Walmart Recruiting - Store Sales Forecasting | Kaggle (Dataset files)

[3] pandas documentation — pandas 1.2.5 documentation (pydata.org)

[4] scikit-learn Tutorials — scikit-learn 0.24.2 documentation (https://scikit-learn.org/stable/tutorial/index.html)