INTERNSHIP: PROJECT REPORT

Name Of The Student	Anagha Isal
Internship Project Title	RIO-125: Forecasting System - Project Demand of Products at a Retail Outlet Based on Historical Data
Name of the Company	TCS iON
Name of the Industry Mentor	Himalaya Ashish
Name of the Institute	ICT Academy of Kerala

Start Date	End Date	Total Effort (hrs.)	Project Environment	Tools used
19/02/2021	30/03/2021	142	Python	Jupyter Notebook /Google colab

Table of Contents

Acknowledgements

I would like to express my sincere gratitude to several individuals and organizations for supporting me throughout my 30 days internship in TCS iON. First, I wish to express my sincere gratitude to my industry mentor and institute mentor for giving proper guidance for the project. Also, I would like to thank the instructors who replied to all our quires posted in discussion room which helped us in getting a clear understanding on how to take up the project. I also wish to express my sincere thanks to ICT Academy of Kerala for accepting me into the Data Science and Analytics Program and giving me opportunity to do a virtual internship in TCS iON.

Problem Statement

The problem is quite straightforward. Data from Walmart stores across the US is given, and it is up to us to forecast their weekly sales. We need to build a model to train data that is able to forecast those weeks sales as accurately as possible.

Aim and Objective

The objective of this project is to build a regression model which predicts the Walmart store sales based on the historical data. With this model, Walmart authorities should be able to decide their future plans which is very important for arranging stocks, calculating revenue and deciding to make new investment or not.

With the accurate prediction company should be able to:

- Determine seasonal demands and take action for this
- Protect from money loss because achieving sales targets can have a positive effect on stock prices and investors' perceptions
- Forecast revenue easily and accurately
- Manage inventories
- Do more effective campaigns

Dataset

The data is obtained from Kaggle competition. There are mainly weekly sales for 45 stores and 81 departments of Walmart in different areas.

Following is the link to the dataset:

https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting/data

1. Introduction

Predicting future sales for a company is one of the most important aspects of strategic planning. In that way, Sales forecasting or predicting the future is very important for every business. It is used for companies to make plans for high revenue, keep costs lower and high efficiency. Companies made short-term and long-term future planning as per forecasting data. Based on past data with some assumptions which predict future trends and draw their budget accordingly.

In this project, I use the dataset of Walmart sales to forecast future sales. The data collected ranges from 2010 to 2012, where 45 Walmart stores across the country were included in this analysis. Each store contains several departments, and the task will be to predict the department-wide sales for each store. It is important to note that we also have external data available like CPI, Unemployment Rate and Fuel Prices in the region of each store which, hopefully, helps us to make a more detailed analysis.

With the accurate prediction company can;

- Determine seasonal demands and take action for this
- Protect from money loss because achieving sales targets can have a positive effect on stock prices and investors' perceptions
- Forecast revenue easily and accurately
- Manage inventories
- Do more effective campaigns

Data:

The data is obtained from Kaggle competition. There are mainly weekly sales for 45 stores and 81 departments of Walmart in different areas.

Plan:

1. Understanding, Cleaning and Exploring Data
2. Preparing Data to Modelling
3. Creating the models and find the model with highest accuracy

1.1 Dataset Overview

For this project, we have used the dataset available from 'Walmart Store Sales Forecasting' project that was available on Kaggle. In this dataset, we have weekly sales data for 45 stores and 81 departments for a period of 3 years. In addition, we had store and geography specific information such as store size, unemployment rate, temperature, promotional markdowns etc.

Stores:
Store: The store number. Range from 1–45.
Type: Three types of stores 'A', 'B' or 'C'.
Size: Sets the size of a Store would be calculated by the no. of products available in the particular store ranging from 34875 to 219622.

Features*:*
Temperature: Temperature of the region during that week.
Fuel Price: Fuel Price in that region during that week.
MarkDown1:5: Represents the Type of markdown and what quantity was available during that week.
*CPI***:** Consumer Price Index during that week.
Unemployment: The unemployment rate during that week in the region of the store.

Sales*:*
Date: The date of the week where this observation was taken.
Weekly Sales: The sales recorded during that Week.
Dept: One of 1–99 that shows the department.
Is Holiday: a Boolean value (True/False) representing a holiday week or not.

2. Data Mining

Data mining is the process that helps in extracting information from the given dataset to identify trends, patterns, and useful data. The objective of using data mining is to make data-supported decisions from enormous data.

2.1 Regression

Regression is a data mining technique used to predict a range of numeric values (also called continuous values), given a particular dataset. For example, regression might be used to predict the cost of a product or service, given other variables. Few data mining algorithms that have been used for regression problems include Linear and Polynomial Regression, Neural Networks, Decision tree, K-nearest neighbor (KNN), Random Forest , Gradient boosting etc.

2.1.1 Linear and Polynomial Regression

Single Variable Linear Regression is a technique used to model the relationship between a single input independent variable (feature variable) and an output dependent variable using a linear model i.e a line. The more general case is Multi Variable Linear Regression where a model is created for the relationship between multiple independent input variables (feature variables) and an output dependent variable. The model remains linear in that the output is a linear combination of the input variables. There is a third most general case called Polynomial Regression where the model now becomes a non-linear combination of the feature variables i.e there can be exponential variables, sine and cosine, etc. This however requires knowledge of how the data relates to the output

2.1.2 Decision Tree Algorithm

A Decision Tree is an intuitive model where by one traverses down the branches of the tree and selects the next branch to go down based on a decision at a node. Tree induction is the task of taking a set of training instances as input, deciding which attributes are best to split on, splitting the dataset, and recurring on the resulting split datasets until all training instances are categorized. While building the tree, the goal is to split on the attributes which create the purest child nodes possible, which would keep to a minimum the number of splits that would need to be made in order to classify all instances in our dataset. Purity is measured by the concept of information gain, which relates to how much would need to be known about a previously-unseen instance in order for it to be properly classified.

2.1.3 Random Forest Algorithm

Random forests or random decision forests is an ensemble learning method for classification and regression problems. Random forest, as the name says it is a combination of number of decision trees and an ensemble classification model. Random forest model collects trained data from all the tree nodes and separates the weaker nodes training data to get better predictions. Both classification and regression problems are solved using RF model.

2.1.4 K-nearest neighbor (KNN) Algorithm

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. The KNN algorithm assumes that similar things exist in close proximity. K-NN was the parameter free method which uses a Euclidean distance measure. This algorithm expressed in terms of the resemblance measure among the pair of data in N-dimension, the number of nearest data that are trained for classification, and vector of training data applied by the classifiers.

2.1.5 Gradient boosting

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function. XGBoost is a special implementation of the Gradient Boosting and XGBoost stands for Extreme Gradient Boosting, XGBoost uses more accurate approximations by employing second-order gradients and advanced regularization. The Objective function is solved by using Second-order Taylor polynomial approximation and simply put XgBoost tries to find the optimal output value for a tree ft in an iteration t that is added to minimize the above loss function across all data point

2.1.6 ExtraTreesRegressor

This class implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

2.2 Model Evaluation Technique

In regression model, the most commonly known evaluation metrics include:

1. R-squared (R^2), which is the proportion of variation in the outcome that is explained by the predictor variables. In multiple regression models, R^2corresponds to the squared correlation between the observed outcome values and the predicted values by the model. The Higher the R-squared, the better the model.

2. Root Mean Squared Error (RMSE), which measures the average error performed by the model in predicting the outcome for an observation. Mathematically, the RMSE is the square root of the mean squared error (MSE), which is the average squared difference between the observed actual outcome values and the values predicted by the model. So, MSE = mean((observed - predicted)^2) and RMSE = sqrt(MSE). The lower the RMSE, the better the model.

The problem with the above metrics, is that they are sensible to the inclusion of additional variables in the model, even if those variables don't have significant contribution in explaining the outcome. Put in other words, including additional variables in the model will always increase the $R^2$and reduce the RMSE. So, we need a more robust metric to guide the model choice.

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score
from math import sqrt
```

3. Approach / Methodology

3.1 Loading the dataset and importing the libraries

Importing the libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rcParams
import calendar # To get month
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score
from math import sqrt
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.preprocessing import StandardScaler,MinMaxScaler,RobustScaler
```

3.2 Merging and Understanding the Dataset

Feature dataset:

```
[ ] features_df.head()
```

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106	False
1	1	2010-02-12	38.51	2.548	NaN	NaN	NaN	NaN	NaN	211.242170	8.106	True
2	1	2010-02-19	39.93	2.514	NaN	NaN	NaN	NaN	NaN	211.289143	8.106	False
3	1	2010-02-26	46.63	2.561	NaN	NaN	NaN	NaN	NaN	211.319643	8.106	False
4	1	2010-03-05	46.50	2.625	NaN	NaN	NaN	NaN	NaN	211.350143	8.106	False

- Store - the store number.
- Date - the week
- Temperature - average temperature in the region
- Fuel Price - cost of fuel in the region
- MarkDown1-5 - 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.

Store dataset:

```
stores_df.head()
```

	Store	Type	Size
0	1	A	151315
1	2	A	202307
2	3	B	37392
3	4	A	205863
4	5	B	34875

Sales Dataset:

```
train_df.head()
```

	Store	Dept	Date	Weekly_Sales	IsHoliday
0	1	1	2010-02-05	24924.50	False
1	1	1	2010-02-12	46039.49	True
2	1	1	2010-02-19	41595.55	False
3	1	1	2010-02-26	19403.54	False
4	1	1	2010-03-05	21827.90	False

This is the training data, which covers to 2010-02-05 to 2012-11-01.

- 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

Merging the three datasets into one with the Store value

Combining the feature, store and sales files

```
master_df = train_df.merge(stores_df, on='Store', how='left')
master_df = master_df.merge(features_df, on=['Store', 'Date'], how='left')
```

After combining the dataset, next step was to perform the preliminary analysis:

Dataset Dimension:

```
master_df.columns
```

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

```
master_df.shape
```

```
(421570, 17)
```

Variables and its data type:

```
master_df.dtypes
```

```
Store             int64
Dept              int64
Date             object
Weekly_Sales    float64
IsHoliday_x        bool
Type             object
Size              int64
Temperature     float64
Fuel_Price      float64
MarkDown1       float64
MarkDown2       float64
MarkDown3       float64
MarkDown4       float64
MarkDown5       float64
CPI             float64
Unemployment    float64
IsHoliday_y        bool
dtype: object
```

3.3 Checking for Null Values in the Dataset

Missing Data can occur in the dataset when no information is provided for one or more items or for a whole unit. isna() function from pandas library gives us the empty cells in the dataset.

```
#Checking for null values
master_df.isna().sum()
```

```
Store                0
Dept                 0
Date                 0
Weekly_Sales         0
IsHoliday_x          0
Type                 0
Size                 0
Temperature          0
Fuel_Price           0
MarkDown1       270889
MarkDown2       310322
MarkDown3       284479
MarkDown4       286603
MarkDown5       270138
CPI                  0
Unemployment         0
IsHoliday_y          0
dtype: int64
```

```
percentage = (master_df.isna().sum()/master_df.count()*100).sort_values(ascending=False)
percentage
```

```
MarkDown2       278.946138
MarkDown4       212.350426
MarkDown3       207.511069
MarkDown1       179.776481
MarkDown5       178.388980
IsHoliday_y       0.000000
Type              0.000000
Dept              0.000000
Date              0.000000
Weekly_Sales      0.000000
IsHoliday_x       0.000000
Fuel_Price        0.000000
Size              0.000000
Temperature       0.000000
Unemployment      0.000000
CPI               0.000000
Store             0.000000
dtype: float64
```

In our dataset, we have missing values in column – MarkDown1, MarkDown2, MarkDown3, MarkDown4 and MarkDown5. All the missing values in the dataset is filled with value 0.

```
[ ]  #Fill the missing values in markdown columns with 0
     master_df['MarkDown1'] = master_df['MarkDown1'].fillna(0)
     master_df['MarkDown2'] = master_df['MarkDown2'].fillna(0)
     master_df['MarkDown3'] = master_df['MarkDown3'].fillna(0)
     master_df['MarkDown4'] = master_df['MarkDown4'].fillna(0)
     master_df['MarkDown5'] = master_df['MarkDown5'].fillna(0)
```

3.4. Variable Analysis:

Store:

On analyzing the store variable, we can see that there are 45 unique values in the store column ranging from 1 to 45.

```
[90] #Store
     print("Number of Store:",master_df['Store'].nunique())
     print("Store: ",master_df['Store'].unique())
```

```
Number of Store: 45
Store:  [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45]
```

Dept:

On analyzing the department variable, we can see that there are 81 unique values in the store column ranging from 1 to 99.

```
#Dept
print("Number of Dept:",master_df['Dept'].nunique())
print("Dept: ",master_df['Dept'].unique())
```

```
Number of Dept: 81
Dept:  [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 16 17 18 19 20 21 22 23 24 25
 26 27 28 29 30 31 32 33 34 35 36 37 38 40 41 42 44 45 46 47 48 49 51 52
 54 55 56 58 59 60 67 71 72 74 77 78 79 80 81 82 83 85 87 90 91 92 93 94
 95 96 97 98 99 39 50 43 65]
```

Holiday

On analyzing the holiday variable, we can see that we have the data for 10 holiday weeks and 133 non-holiday weeks.

```
[92] #Holiday
     df_holiday = master_df.loc[master_df['isHoliday']==True]
     df_holiday['Date'].unique()
```

```
array(['2010-02-12', '2010-09-10', '2010-11-26', '2010-12-31',
       '2011-02-11', '2011-09-09', '2011-11-25', '2011-12-30',
       '2012-02-10', '2012-09-07'], dtype=object)
```

```
[94] df_holiday['Date'].nunique()
```

```
10
```

```
df_not_holiday = master_df.loc[master_df['isHoliday']==False]
df_not_holiday['Date'].nunique()
```

```
133
```

Thus there are 133 weeks for non-holiday and 10 weeks for holiday.

Size:

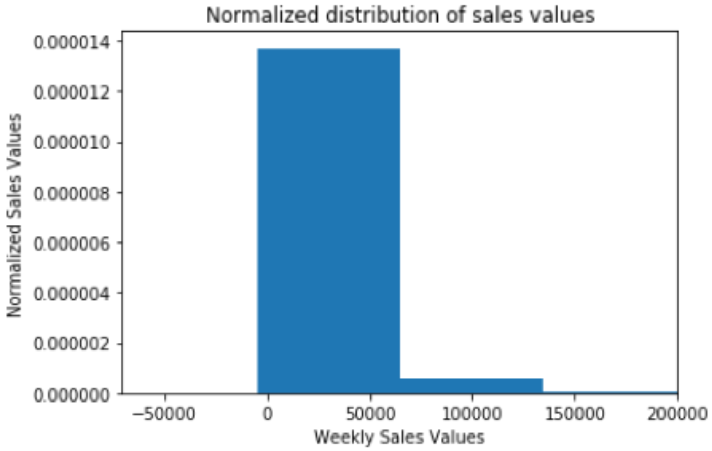
On analyzing the Size variable, we can see that number of products available in the 45 Walmart stores ranges from 34875 to 219622.

```
[146] #size
      print("Minimum Weekly_Sales value:",master_df['Size'].min())
      print("Maximum Weekly_Sales value:",master_df['Size'].max())
```

```
Minimum Weekly_Sales value: 34875
Maximum Weekly_Sales value: 219622
```

Weekly Sales:

On analyzing the weekly sales variable, we can see that the weekly sales value in the 45 Walmart stores goes upto 693099.36.



Analyzing the weekly Sales data, we can some negative values and zero values in the weekly sales value column. To confirm and revalidate the same I plotted a normalized histogram where I found there are some values which lie on the negative side of zero indicating negative sales. Since sales can never be a negative value or a zero value, we are excluding the negative and zero values in the sales column.

```
[102] #Weekly_Sales
      print("Minimum Weekly_Sales value:",master_df['Weekly_Sales'].min())
      print("Maximum Weekly_Sales value:",master_df['Weekly_Sales'].max())

      Minimum Weekly_Sales value: -4988.94
      Maximum Weekly_Sales value: 693099.36
```

From the pivot table, it is obviously seen that there are some wrong values such as there are 0 and minus values for weekly sales. But sales amount can not be minus. Also, it is impossible for one department not to sell anything whole week. So, I will change this values.

```
[104]  #keeping only positive values in weekly sales
      master_df = master_df.loc[master_df['Weekly_Sales'] > 0]
```

3.5 Feature Engineering

After variable analysis, next step was feature engineering. I tried to create few columns out of the date filed provided in the dataset. New features created from the date filed are Week Number, Month, Quarter, Year and Season.

Week Number – The dataset consists of data for 52 weeks from 2010-12.

Month – This filed consider the month value of each date

```
import calendar # To get month
master_df['Month'] = master_df['Date'].dt.month.apply(lambda x: calendar.month_name[x])
master_df['Month'].value_counts()
```

```
April        41332
July         40980
March        38451
October      38362
September    38339
August       38169
June         38137
February     35526
May          35314
December     29802
November     23613
January      23545
Name: Month, dtype: int64
```

Following is the distribution of average sales for each month:

```
sales_month = master_df.groupby(by=master_df['Month'])['Weekly_Sales','Month'].mean().sort_values(by='Weekly_Sales', ascending = False) # to see the best months for s
sales_month
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of keys) will be deprecate
  """Entry point for launching an IPython kernel.
```

Month	Weekly_Sales
December	19355.702141
November	17491.031424
June	16326.137002
August	16062.516933
February	16008.779217
July	15861.419650
May	15776.337202
April	15650.338357
March	15416.657597
October	15243.855576
September	15095.886154
January	14126.075111

Maximun sales is obtained in the month of December

This means that maximum sales is obtained in the month of December. This could be because we have Christmas in December and next highest sales happened in November. This could be again because Black Friday is in November.

Quarter – This filed consider the quarter of each date

```
master_df['Week_Number'] = master_df['Date'].dt.week
master_df['Quarter'] = master_df['Date'].dt.quarter
master_df['Quarter'].value_counts()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Series.dt.weekofyear and Series.dt.week
  """Entry point for launching an IPython kernel.
3    117488
2    114783
1     97522
4     91777
Name: Quarter, dtype: int64
```

Year – This filed explains us about the data we have for each year

```
[ ]  master_df["Year"] = master_df["Date"].dt.year
     master_df['Year'].value_counts()
```

```
2011    153453
2010    140679
2012    127438
Name: Year, dtype: int64
```

Season – With the month data that we obtained from the date field, I tried to obtain the season for each of the month. We have data for season Spring and Winter.

```
[ ]  #mapping season with each month
     seasons_dict = {
         1:"Winter",
         2:"Winter",
         3:"Spring",
         4:"Spring",
         5:"Spring",
         6:"Summer",
         7:"Summer",
         8:"Summer",
         9:"Fall",
         10:"Fall",
         11:"Fall",
         12:"Winter"
     }
     master_df['Season'] = (master_df['Date'].apply(lambda dt: (dt.month%12 + 3)//3)).map(seasons_dict)
     master_df['Season'].value_counts()
```

```
Spring    217600
Winter    203970
Name: Season, dtype: int64
```

3.6 Encoding the categorical Variables

The categorical and Boolean variables such as Season, Store type, Quarter and isHoliday values are encoded using the get_dummies() function in Pandas. The pd. get_dummies function when applied to a column of categories where we have one category per observation will produce a new column (variable) for each unique categorical value. It will place a one in the column corresponding to the categorical value present for that observation. This is equivalent to one hot encoding.

```
[ ]  master_df = master_df.join(pd.get_dummies(master_df['Quarter'], prefix='Quarter'))
```

```
[ ]  master_df['Season'].value_counts()
```

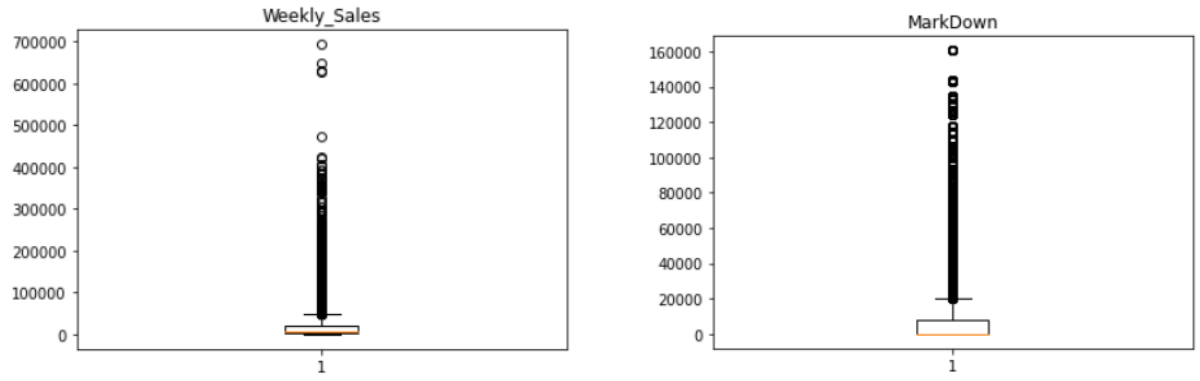
```
Spring    216924
Winter    203288
Name: Season, dtype: int64
```

```
[ ]  master_df = master_df.join(pd.get_dummies(master_df['Season'], prefix='Season'))
```

```
[ ]  master_df = master_df.join(pd.get_dummies(master_df['Type'], prefix='Type'))
```

3.7 Outlier Detection

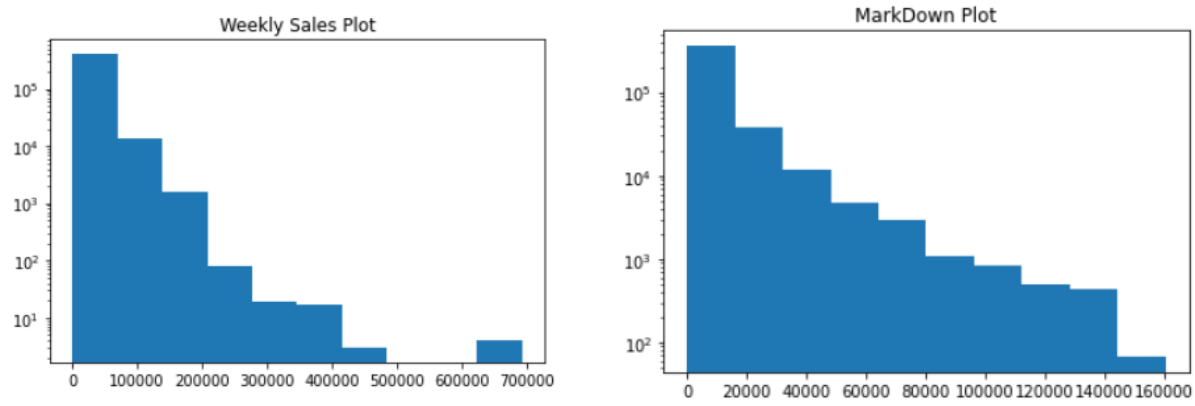
An outlier is a data point in a dataset that lies outside the overall distribution of the dataset. They are also datapoints which has to flagged and investigated if they are conducive for including in the analysis.



From this plot, it is clear that there are a lot of outliers in weekly sales filed and in MarkDown filed. But since weekly sales in our target variable and sales can increase drastically in the seasonally period, I did not remove the outliers present in the Weekly Sales column. And since MarkDown feature does not show much correlation with the target variable, i.e. weekly sales, I thought of removing this feature from the training dataset instead of treating the outliers.

3.8 Checking the Skewness

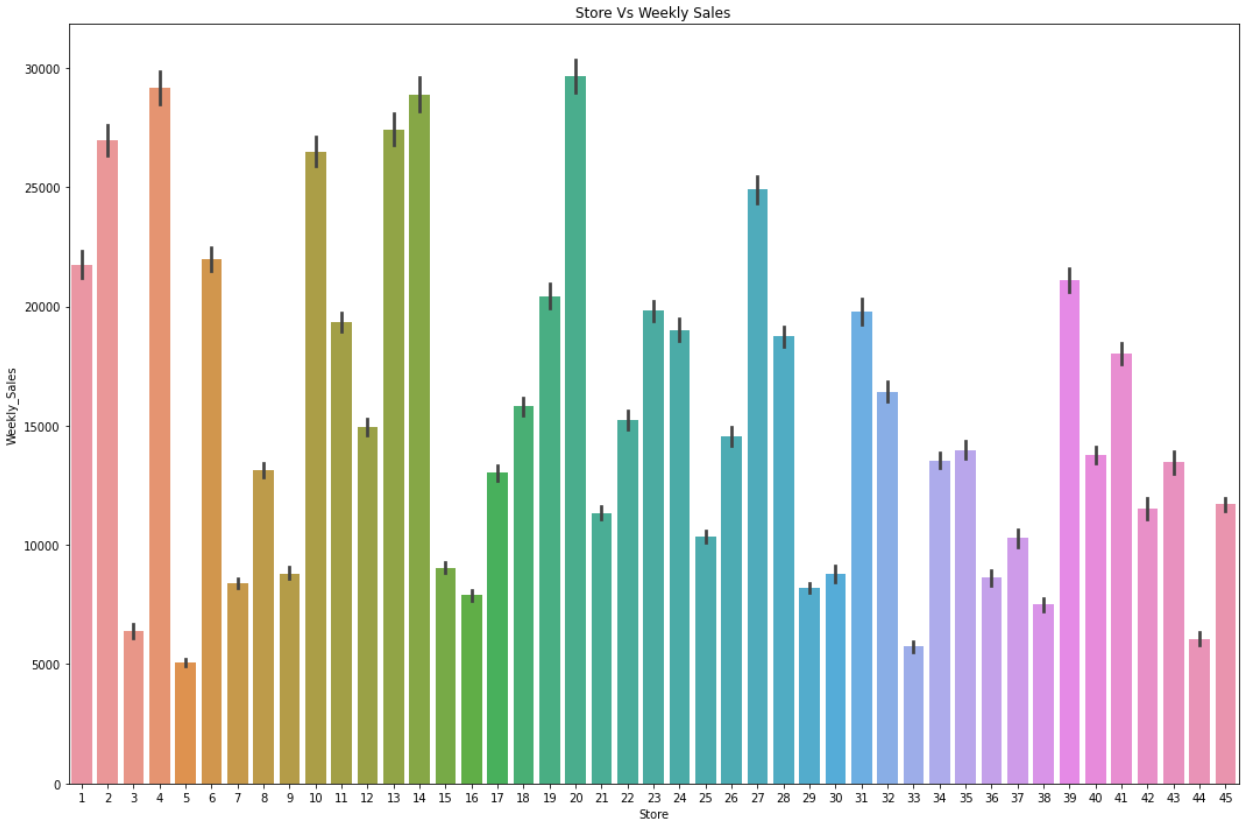
Skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean. The skewness value can be positive, zero, negative, or undefined.



From the above plots, we can say that both Weekly sales and MarkDown fields are right skewed where the skewness value will be positive.

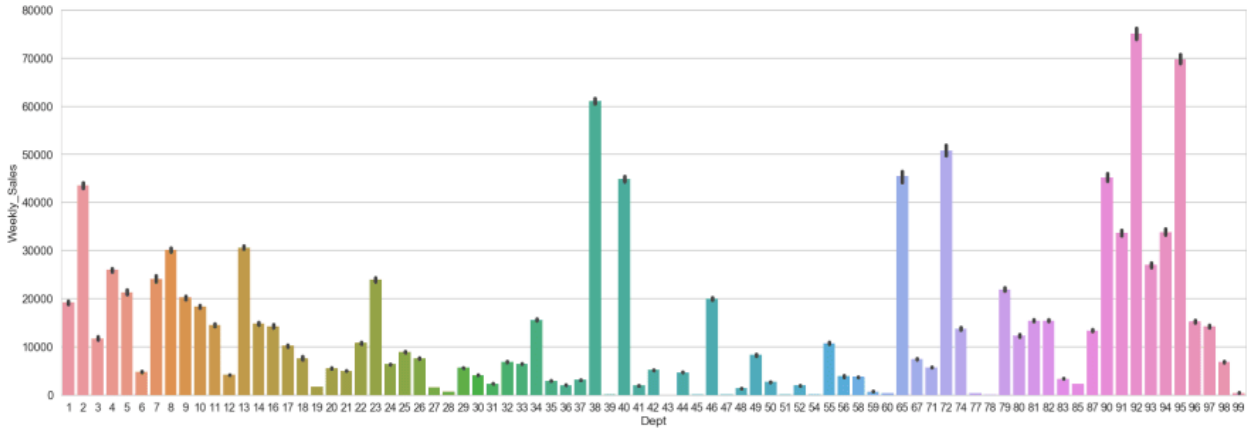
3.9 Data Visualizations

i) Weekly Sales Vs Store



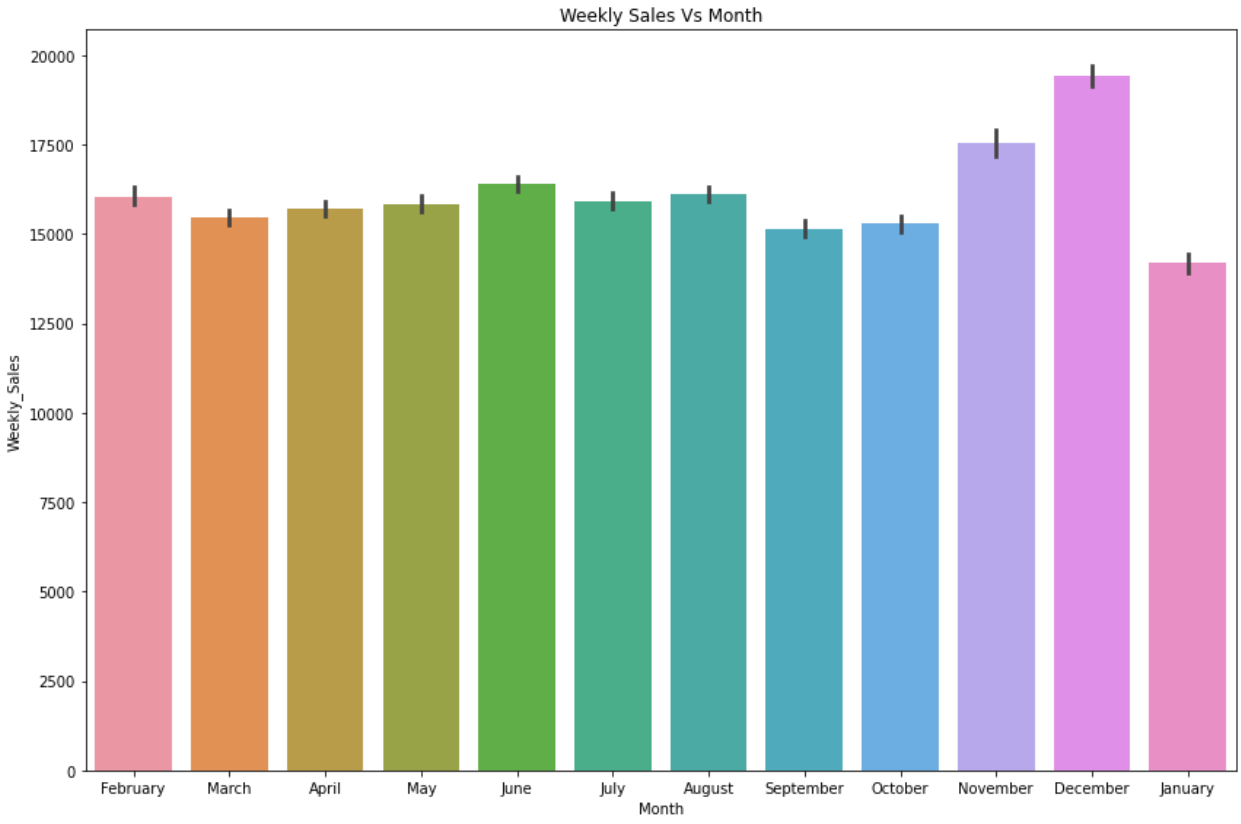
From the above plot, we can say that sales varies for each store. I could be because of the products available in each store, the physical location of each store etc. Out of the 45 Walmart Stores, store no. 20 shows the maximum sales.

ii) Average Weekly Sales for each Dept

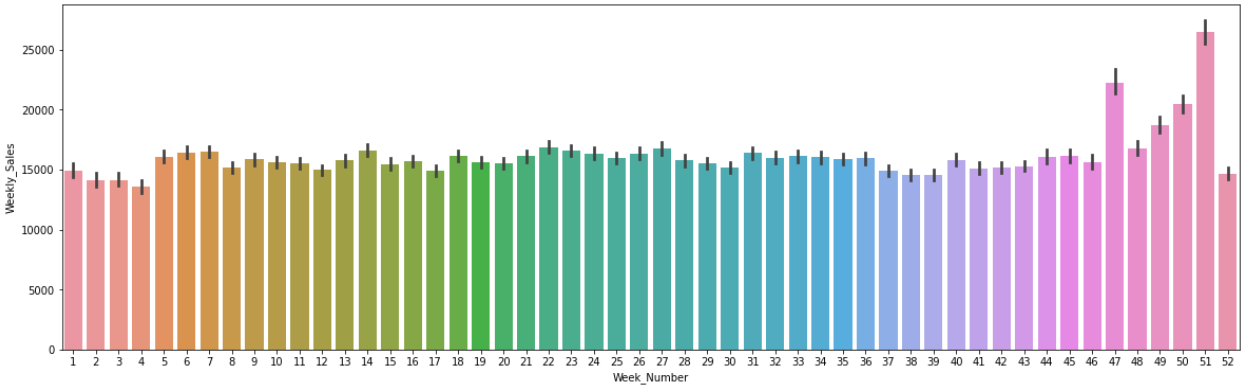


From the above plot, we can see that Dept 92 and 95 have highest average sales when compare to other departments.

iii) Weekly Sales Vs Month

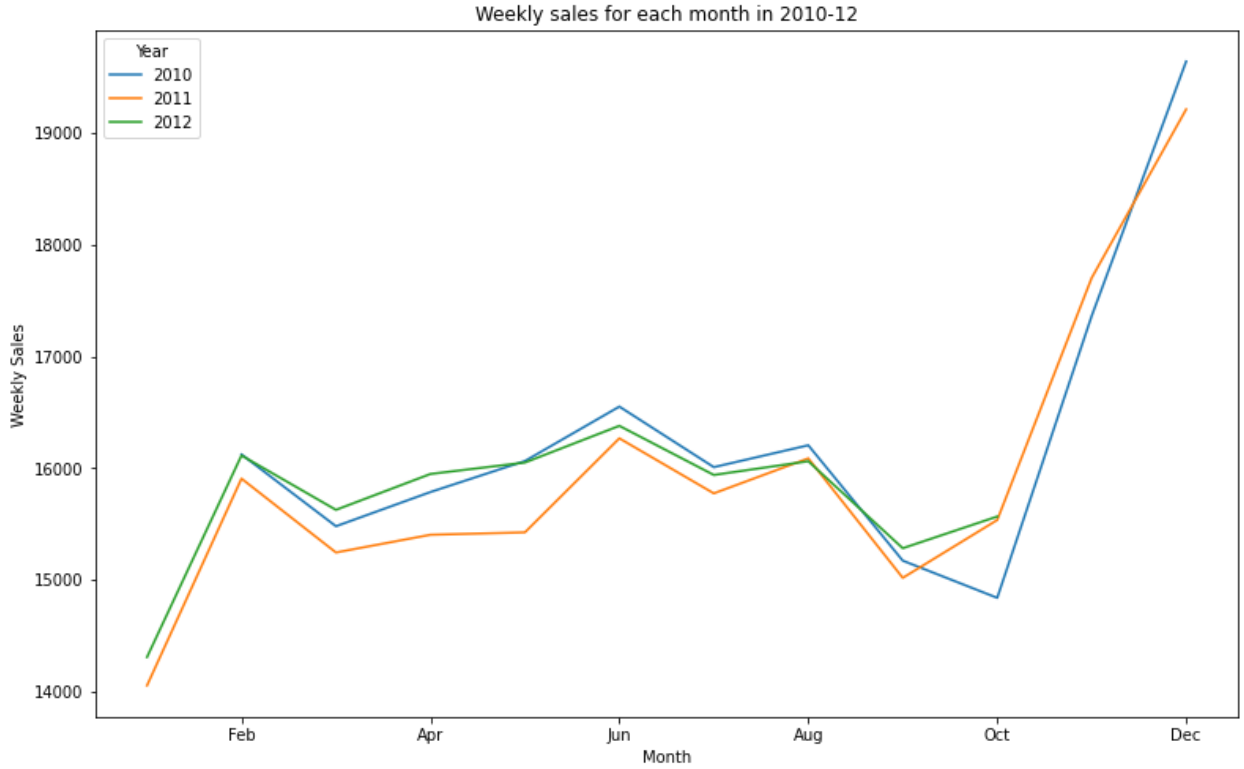


When we look at the average sales monthly, it is obviously seen that November and December have higher sales. To make deep analysis, I looked at the weekly sales with corresponding week numbers.



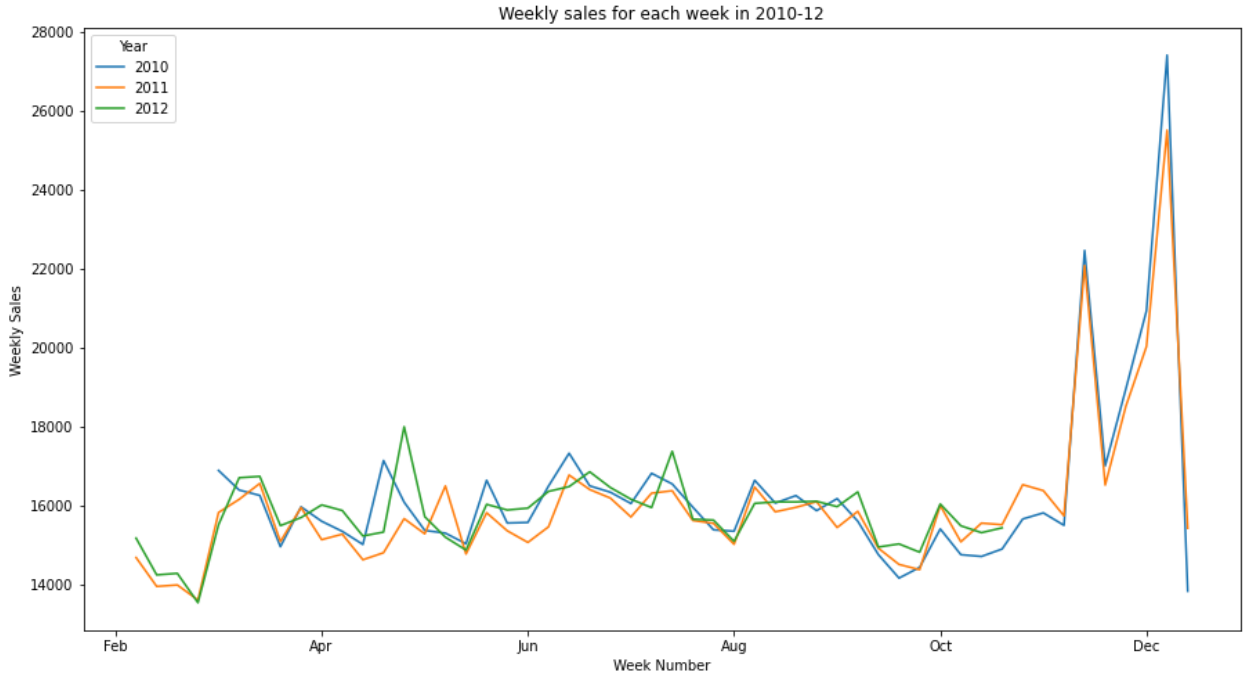
From the graph, we can understand easily 51st and 47th weeks have higher sales. It means that higher demands in whole year belong to Christmas, and Black Friday seasons. Although during other months sales are closed to each other's, January has the least sales average. This is the result of November and December high sales. After two high sales month, people prefer to pay less on January.

iv) Weekly Sales for each month in 2010-12



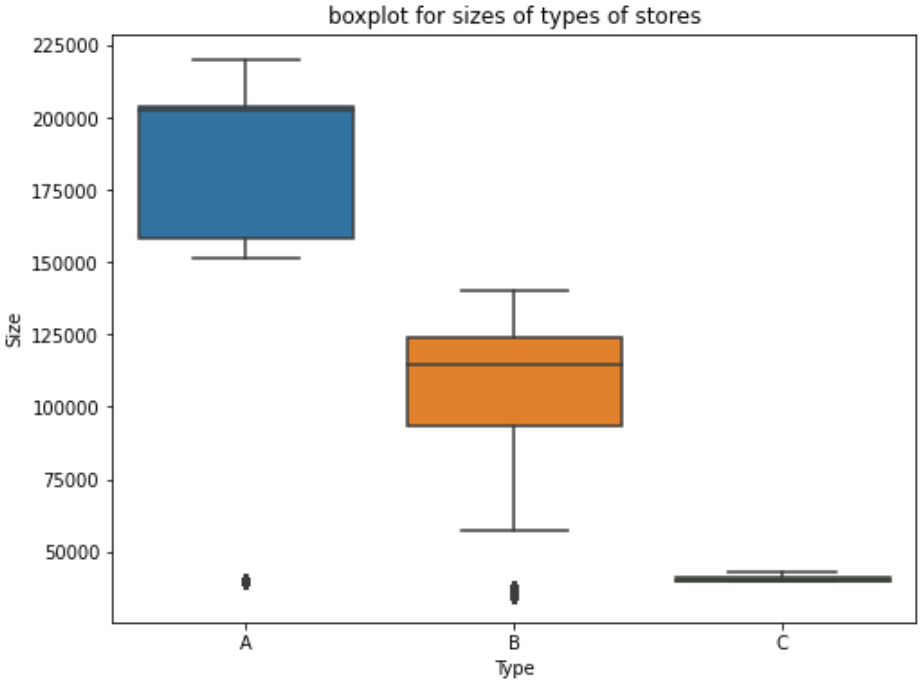
From the graph, it is seen that 2011 has lower sales than 2010 generally. When we look at the mean sales it is seen that 2010 has higher values, but 2012 has no information about November and December which have higher sales. Despite of 2012 has no last two months sales, it's mean is near to 2010. Most probably, it will take the first place if we get 2012 results and add them.

v) Weekly sales for each week in 2010-12



From graphs, it is seen that 51th week and 47th weeks have significantly higher averages as Christmas and Black Friday effects.

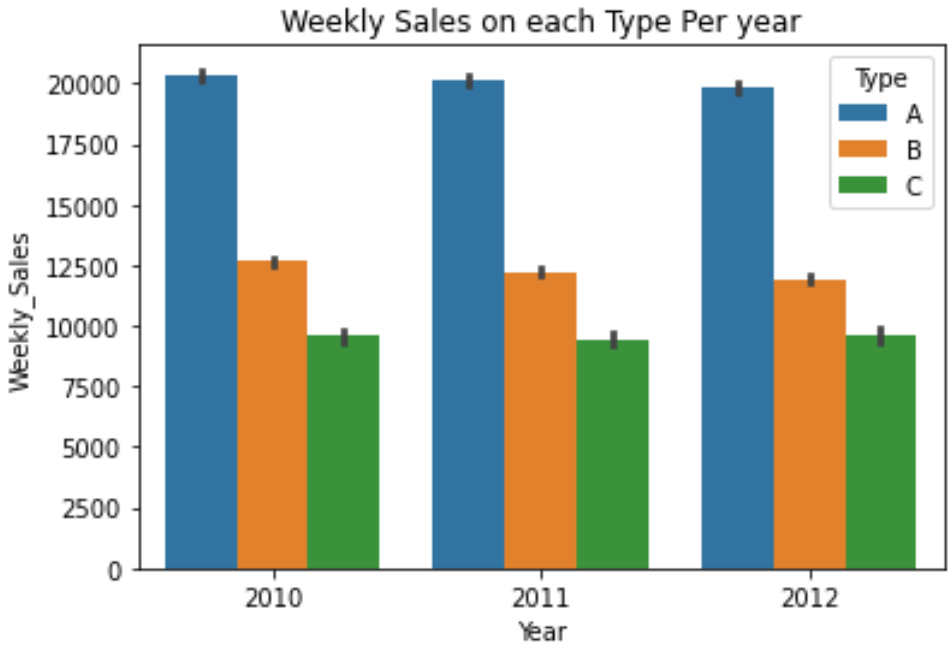
vi) Sizes of types of stores



From the box plot above, we can understand that there are mainly 3 types of Walmart stores in the data and Walmart categorized them according to sizes. Type A is the biggest store, B is

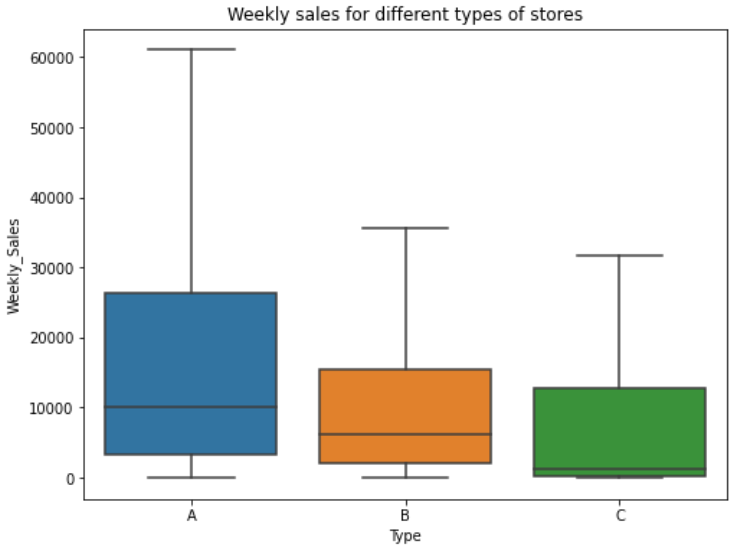
medium and C is the smallest. And, nearly half of the stores in the data are Type A stores which are biggest.

vii) Weekly sales for all the tree types (A,B,C) from 2010-12



The above plot says that in all the three years, maximum sales happened in Type A store and least sales happened in Type C Store.

viii) Weekly sales in the three different type of stores

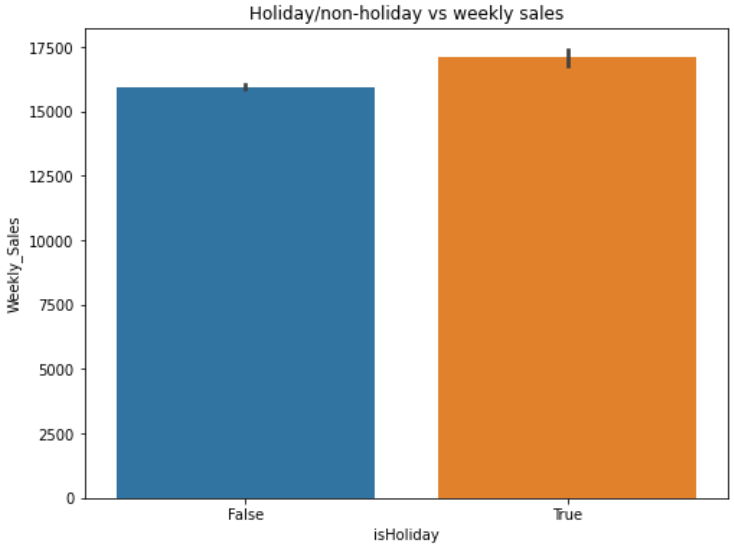


From the above three plots, we can say that since type A store are larger stores with more products, the sales also became maximum in type A store when compared to other stores. Also, our dataset does not have equal number of sales data from all the three store which can also result in the inconsistency.

```
#Type
master_df['Type'].value_counts()
```

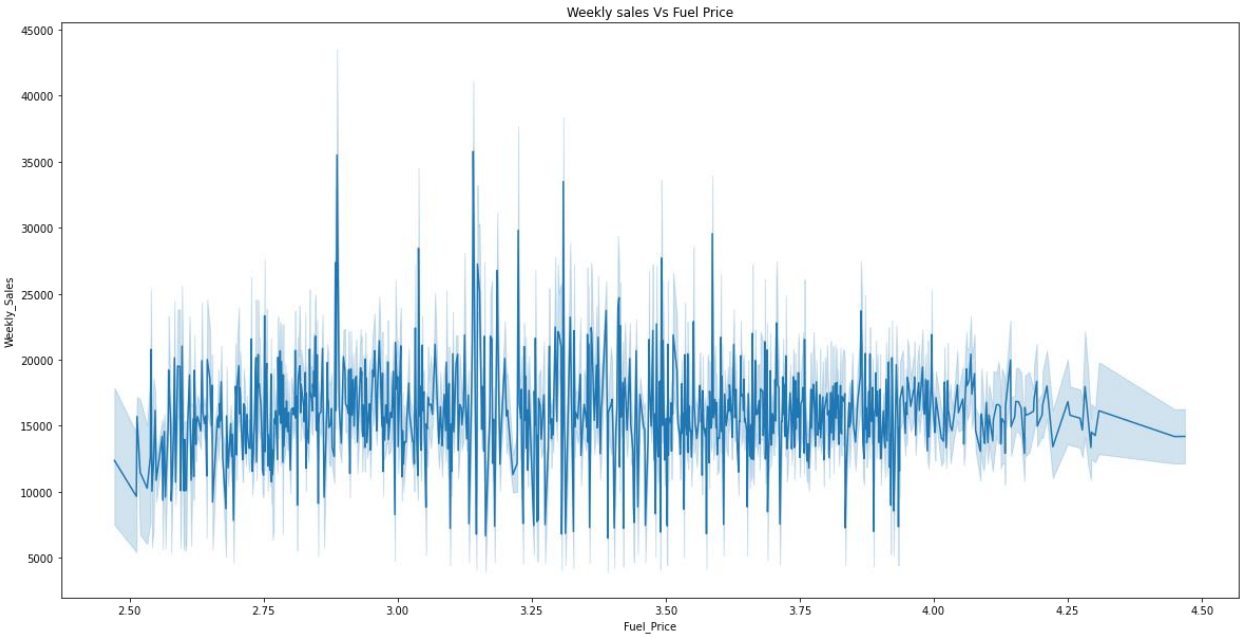
```
A    215478
B    163495
C     42597
Name: Type, dtype: int64
```

ix) Weekly sales on Holiday/Non-holiday



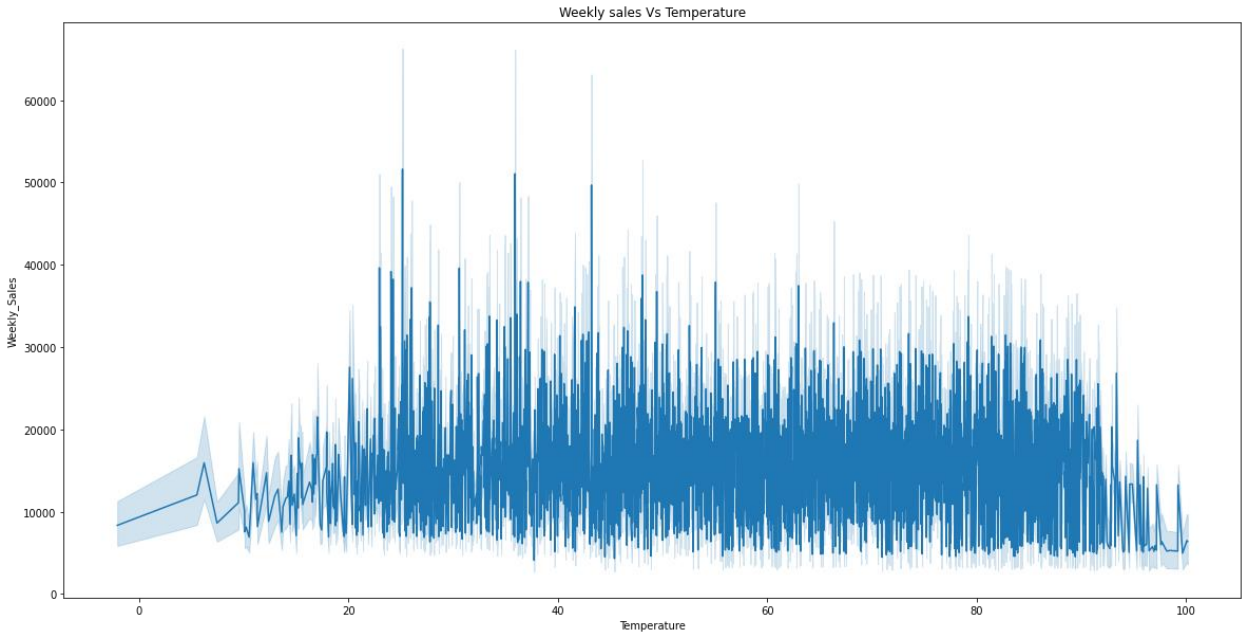
From the above plot, we can say that sales on holiday is a little bit more than sales in not-holiday.

x) Weekly Sales Vs Fuel price:



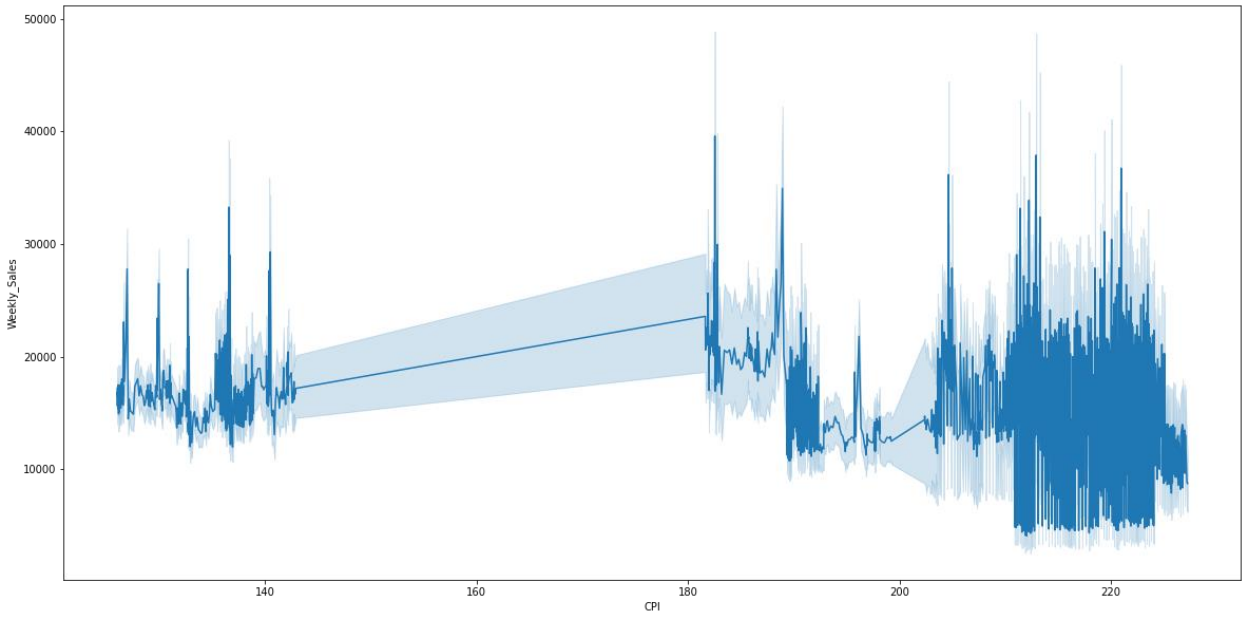
From above plot, it is seen that there are no significant patterns between fuel price and weekly sales.

xi) Weekly Sales Vs Temperature



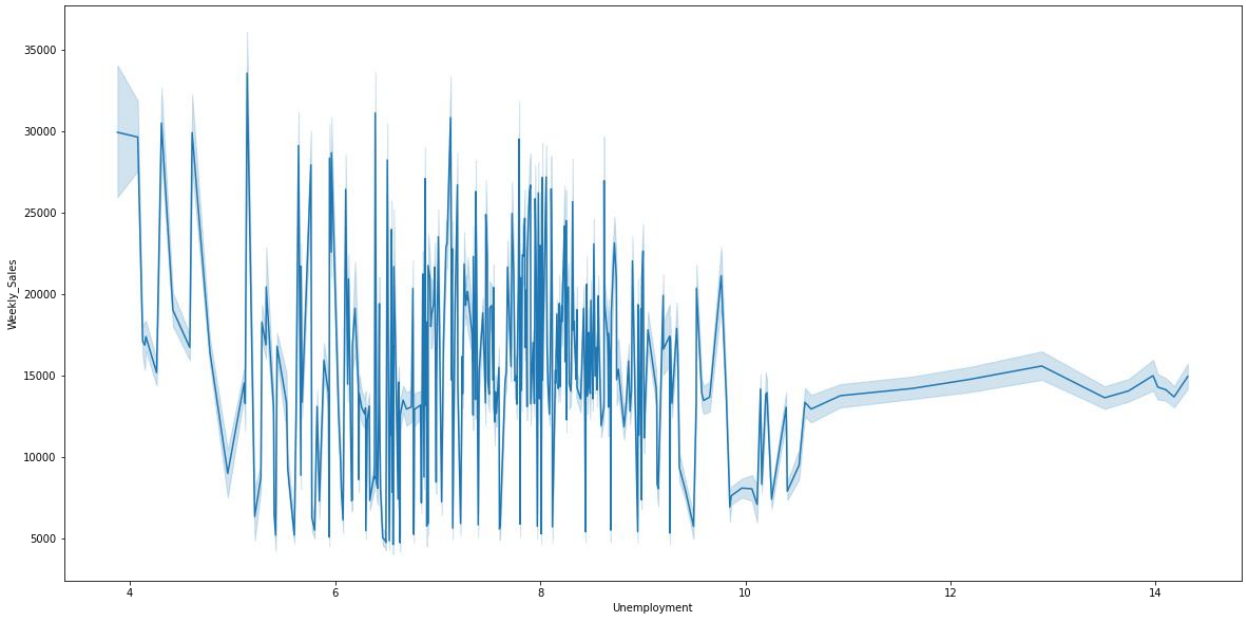
From above plot, it is seen that there are no significant patterns between Temperature and weekly sales.

xii) Weekly Sales Vs CPI



From above plot, it is seen that there are no significant patterns between CPI and weekly sales.

xiii) Weekly Sales Vs Unemployment



From above plot, it is seen that there are no significant patterns between Unemployment and weekly sales.

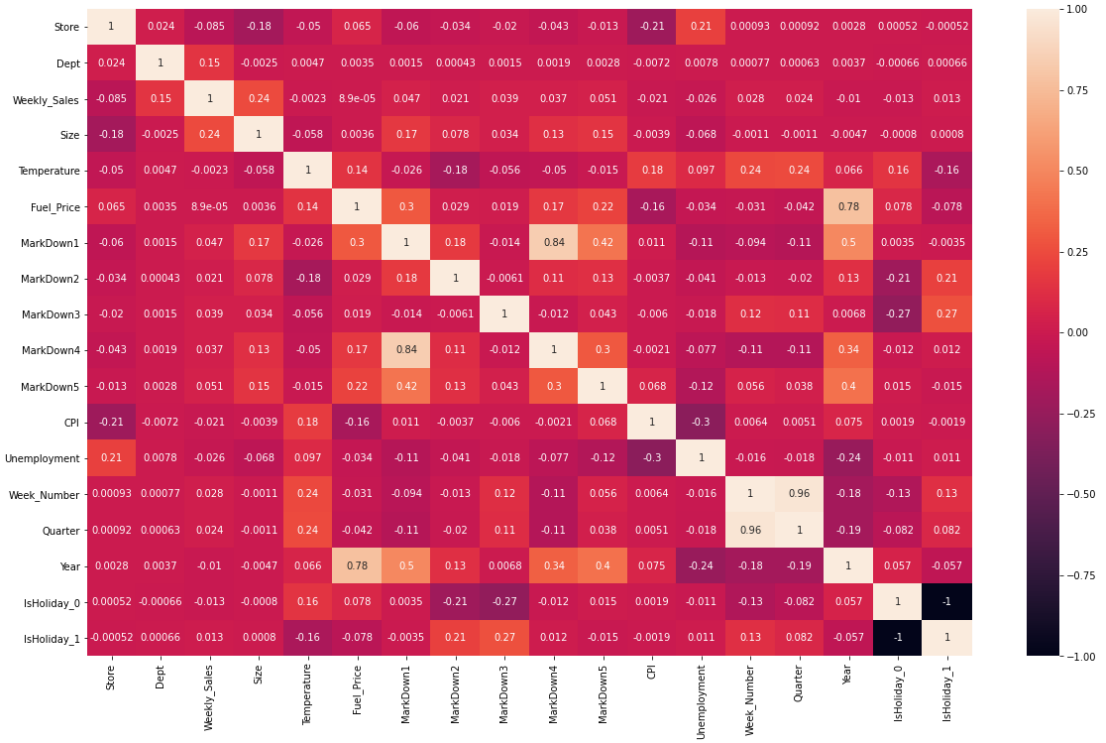
3.10 Correlation

Correlation is a bivariate analysis that measures the strength of association between two variables and the direction of the relationship. In terms of the strength of relationship, the value of the correlation coefficient varies between +1 and -1.

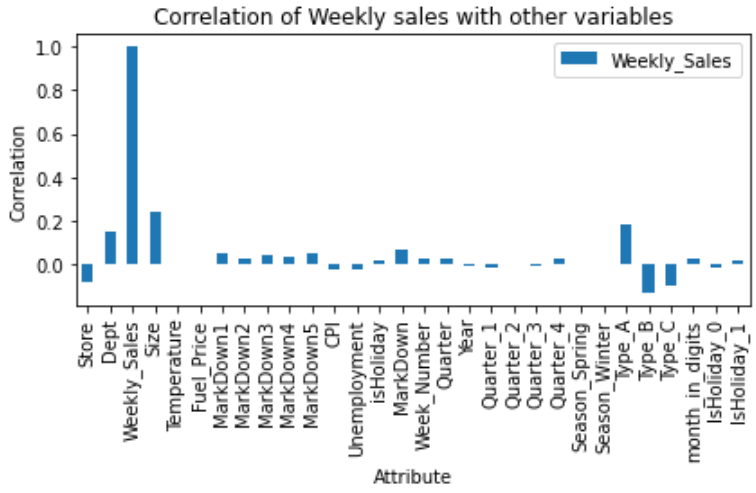
Before splitting the dataset into feature set and target variable, it is important to check the correlation between the target variable and the features.

A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. A correlation matrix is used to summarize data, as an input into a more advanced analysis, and as a diagnostic for advanced analyses.

Following is the correlation matrix for the master data:



In order to understand more on the correlation of Weekly sales variable with other features, I tried to plot a bar graph that shows correlation:



This means that Weekly sales have higher correlation with Dept, Store size, Store and Store Type.

3.11 Scaling

In most cases, the numerical features of the dataset do not have a certain range and they differ from each other. In real life, it is nonsense to expect age and income columns to have the same range. But from the machine learning point of view, we can compare these columns using Scaling. The continuous features become identical in terms of the range, after a scaling process. Here I have used three scaling techniques to standardize the variables in the dataset.

1. Standard Scaling: This assumes data is normally distributed within each feature and scales them such that the distribution centered around 0, with a standard deviation of 1.
2. MinMax Scaling: This is the simplest method and consists in rescaling the range of features to scale the range in [0, 1] or [−1, 1]. Selecting the target range depends on the nature of the data.
3. Robust Scaling: This technique ignores the outliers and scales the variables based on the interquartile range.

```
[104] from sklearn.preprocessing import StandardScaler,MinMaxScaler,RobustScaler
```

3.12 Splitting the dataset:

Here I have separated the dataset into train set and test set with the test set size as 20% of the dataset. Now the machine learning models can be trained with the 80% of the data and can test the model with the 20% data.

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2, random_state=42)
print("x_train :",x_train.shape)
print("y_train :",y_train.shape)
print("x_test :",x_test.shape)
print("y_test :",y_test.shape)
```

```
x_train : (336169, 18)
y_train : (336169, 1)
x_test : (84043, 18)
y_test : (84043, 1)
```

3.13 Machine Learning Models

Various machine learning models that I used to predict the Weekly Sales include:

1.Linear Regression

R^2 and Root Mean Squared Error for Linear Regression Model before scaling is:

```
Model 1: LinearRegression
```

```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(x_train,y_train)
y_pred = lm.predict(x_test)
```

```
[90] print("Mean squared error: ", mse(y_test, y_pred))
     print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
     print("R-squared: ", r2_score(y_test, y_pred))
```

```
Mean squared error:  462911059.17540187
Root mean squared error:  21515.367976760284
R-squared:  0.09376278345609113
```

R^2 and Root Mean Squared Error for Linear Regression Model after standard scaling is:

Model 1: LinearRegression After Standard Scaling

```
[106] lr_model = LinearRegression()
      lr_model.fit(x_train,y_train)
      y_pred=lr_model.predict(x_test)
      print("Score:",lr_model.score(x_test,y_test))
      print("Mean squared error: ", mse(y_test, y_pred))
      print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
      print("R-squared: ", r2_score(y_test, y_pred))
```

```
Score: 0.09375914904149718
Mean squared error:  462912915.65478545
Root mean squared error:  21515.411119817938
R-squared:  0.09375914904149718
```

R^2 and Root Mean Squared Error for Linear Regression Model after MinMax scaling is:

Model 1: LinearRegression after MinMax Scaling

```
[113] lr_model = LinearRegression()
      lr_model.fit(x_train,y_train)
      y_pred=lr_model.predict(x_test)
      print("Score:",lr_model.score(x_test,y_test))
      print("Mean squared error: ", mse(y_test, y_pred))
      print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
      print("R-squared: ", r2_score(y_test, y_pred))
```

```
Score: 0.0937627834560899
Mean squared error:  462911059.17540246
Root mean squared error:  21515.367976760295
R-squared:  0.0937627834560899
```

R^2 and Root Mean Squared Error for Linear Regression Model after Robust scaling is:

Model 1: LinearRegression after Robust Scaling

```
[120] lr_model = LinearRegression()
      lr_model.fit(x_train,y_train)
      y_pred=lr_model.predict(x_test)
      print("Score:",lr_model.score(x_test,y_test))
      print("Mean squared error: ", mse(y_test, y_pred))
      print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
      print("R-squared: ", r2_score(y_test, y_pred))
```

```
Score: 0.09376278345609146
Mean squared error:  462911059.1754016
Root mean squared error:  21515.367976760277
R-squared:  0.09376278345609146
```

2.K-Nearest Neighbors

R^2 and Root Mean Squared Error for K-Nearest Neighbors Model before scaling is:

```
[93] r2_value.index(max(r2_value))
```

```
3
```

```
[94] knn = KNeighborsRegressor(n_neighbors=r2_value.index(max(r2_value)),metric = 'minkowski')
     knn.fit(x_train,y_train)
     y_pred = knn.predict(x_test)
```

```
print("Score:",knn.score(x_test,y_test))
print("Mean squared error: ", mse(y_test, y_pred))
print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
print("R-squared: ", r2_score(y_test, y_pred))
```

```
Score: 0.44565612933067306
Mean squared error:  283161962.04958755
Root mean squared error:  16827.416974972348
R-squared:  0.445656129330673
```

R^2 and Root Mean Squared Error for K-Nearest Neighbors Model after standard scaling is:

Model 2: K-Nearest Neighbors After Standard Scaling

```
knn = KNeighborsRegressor(n_neighbors=r2_value.index(max(r2_value)),metric = 'minkowski')
knn.fit(x_train,y_train)
y_pred = knn.predict(x_test)
print("Score:",knn.score(x_test,y_test))
print("Mean squared error: ", mse(y_test, y_pred))
print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
print("R-squared: ", r2_score(y_test, y_pred))
```

```
Score: 0.30857920905361946
Mean squared error:  353181622.6809918
Root mean squared error:  18793.12700646148
R-squared:  0.30857920905361946
```

R^2 and Root Mean Squared Error for K-Nearest Neighbors Model after MinMax scaling is:

Model 2: K-Nearest Neighbors after MinMax Scaling

```
knn = KNeighborsRegressor(n_neighbors=r2_value.index(max(r2_value)),metric = 'minkowski')
knn.fit(x_train,y_train)
y_pred = knn.predict(x_test)
print("Score:",knn.score(x_test,y_test))
print("Mean squared error: ", mse(y_test, y_pred))
print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
print("R-squared: ", r2_score(y_test, y_pred))
```

```
Score: 0.46508207720072753
Mean squared error:  273239078.7913745
Root mean squared error:  16529.944911928003
R-squared:  0.4650820772007276
```

R^2 and Root Mean Squared Error for K-Nearest Neighbors Model after Robust scaling is:

Model 2: K-Nearest Neighbors after Robust Scaling

```
knn = KNeighborsRegressor(n_neighbors=r2_value.index(max(r2_value)),metric = 'minkowski')
knn.fit(x_train,y_train)
y_pred = knn.predict(x_test)
print("Score:",knn.score(x_test,y_test))
print("Mean squared error: ", mse(y_test, y_pred))
print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
print("R-squared: ", r2_score(y_test, y_pred))
```

```
Score: 0.3087149200605085
Mean squared error:  353112300.7365893
Root mean squared error:  18791.28257295359
R-squared:  0.3087149200605085
```

3.DecisionTree Algorithm

R^2 and Root Mean Squared Error for DecisionTree Model before scaling is:

Model 3 : DecisionTree Algorithm

```
[96] from sklearn.tree import DecisionTreeRegressor
     dtr_model = DecisionTreeRegressor()
     dtr_model.fit(x_train,y_train)
     y_pred=dtr_model.predict(x_test)
```

```
print("Score:",dtr_model.score(x_test,y_test))
print("Mean squared error: ", mse(y_test, y_pred))
print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
print("R-squared: ", r2_score(y_test, y_pred))
```

```
Score: 0.9629828917825534
Mean squared error:  18908546.746621467
Root mean squared error:  4348.395882003094
R-squared:  0.9629828917825535
```

R^2 and Root Mean Squared Error for DecisionTree Model after standard scaling is:

Model 3 : DecisionTree Algorithm After Standard Scaling

```
dtr_model = DecisionTreeRegressor()
dtr_model.fit(x_train,y_train)
y_pred=dtr_model.predict(x_test)
print("Score:",dtr_model.score(x_test,y_test))
print("Mean squared error: ", mse(y_test, y_pred))
print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
print("R-squared: ", r2_score(y_test, y_pred))
```

```
Score: 0.9597437095120767
Mean squared error:  20563139.239972193
Root mean squared error:  4534.6597711374325
R-squared:  0.9597437095120767
```

R^2 and Root Mean Squared Error for DecisionTree Model after MinMax scaling is:

Model 3 : DecisionTree Algorithm after MinMax Scaling

```
dtr_model = DecisionTreeRegressor()
dtr_model.fit(x_train,y_train)
y_pred=dtr_model.predict(x_test)
print("Score:",dtr_model.score(x_test,y_test))
print("Mean squared error: ", mse(y_test, y_pred))
print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
print("R-squared: ", r2_score(y_test, y_pred))
```

```
Score: 0.9616960806554798
Mean squared error:  19565857.096403994
Root mean squared error:  4423.3309955738105
R-squared:  0.9616960806554797
```

R^2 and Root Mean Squared Error for DecisionTree Model after Robust scaling is:

Model 3 : DecisionTree Algorithm after Robust Scaling

```
[122] dtr_model = DecisionTreeRegressor()
      dtr_model.fit(x_train,y_train)
      y_pred=dtr_model.predict(x_test)
      print("Score:",dtr_model.score(x_test,y_test))
      print("Mean squared error: ", mse(y_test, y_pred))
      print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
      print("R-squared: ", r2_score(y_test, y_pred))
```

```
Score: 0.9628970967170445
Mean squared error:  18952371.347864725
Root mean squared error:  4353.432134289534
R-squared:  0.9628970967170445
```

4.Random Forest Algorithm

R^2 and Root Mean Squared Error for Random Forest Model before scaling is:

Model 4 : RandomForestRegressor

```
[98] from sklearn.ensemble import RandomForestRegressor
     rfr_model = RandomForestRegressor()
     rfr_model.fit(x_train,y_train)
     y_pred=rfr_model.predict(x_test)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected
  This is separate from the ipykernel package so we can avoid doing imports until
```

```
print("Score:",rfr_model.score(x_test,y_test))
print("Mean squared error: ", mse(y_test, y_pred))
print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
print("R-squared: ", r2_score(y_test, y_pred))
```

```
Score: 0.9780850808664608
Mean squared error:  11194263.756390749
Root mean squared error:  3345.782981065979
R-squared:  0.9780850808664608
```

R^2 and Root Mean Squared Error for Random Forest Model after standard scaling is:

Model 4 : RandomForestRegressor After Standard Scaling

```
[109] rfr_model = RandomForestRegressor(n_estimators = 400,max_depth=15,n_jobs=4)
      rfr_model.fit(x_train,y_train)
      y_pred=rfr_model.predict(x_test)
      print("Score:",rfr_model.score(x_test,y_test))
      print("Mean squared error: ", mse(y_test, y_pred))
      print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
      print("R-squared: ", r2_score(y_test, y_pred))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DataConversionWarning: A column-vector y was passed whe

Score: 0.9655071957263505
Mean squared error:  17619118.116929844
Root mean squared error:  4197.513325402297
R-squared:  0.9655071957263506
```

R^2 and Root Mean Squared Error for Random Forest Model after MinMax scaling is:

Model 4 : RandomForestRegressor after MinMax Scaling

```
rfr_model = RandomForestRegressor(n_estimators = 400,max_depth=15,n_jobs=4)
rfr_model.fit(x_train,y_train)
y_pred=rfr_model.predict(x_test)
print("Score:",rfr_model.score(x_test,y_test))
print("Mean squared error: ", mse(y_test, y_pred))
print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
print("R-squared: ", r2_score(y_test, y_pred))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DataConversionWarning: A column-vector y was passed when a 1d arra

Score: 0.9656106222746017
Mean squared error:  17566287.25529259
Root mean squared error:  4191.215486621106
R-squared:  0.9656106222746017
```

R^2 and Root Mean Squared Error for Random Forest Model after Robust scaling is:

Model 4 : RandomForestRegressor after Robust Scaling

```
[123] rfr_model = RandomForestRegressor(n_estimators = 400,max_depth=15,n_jobs=4)
      rfr_model.fit(x_train,y_train)
      y_pred=rfr_model.predict(x_test)
      print("Score:",rfr_model.score(x_test,y_test))
      print("Mean squared error: ", mse(y_test, y_pred))
      print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
      print("R-squared: ", r2_score(y_test, y_pred))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DataConversionWarning: A column-vector y was passed when a 1d a

Score: 0.9659777060880188
Mean squared error:  17378778.782044094
Root mean squared error:  4168.786248063589
R-squared:  0.9659777060880189
```

5.XGB Regressor

R^2 and Root Mean Squared Error for XGB Regressor Model before scaling is:

Model 5 : XGBRegressor Model

```
[100] from xgboost import XGBRegressor
      xgb_model = XGBRegressor(objective='reg:linear', nthread= 4, n_estimators= 500, max_depth= 6, learning_rate= 0.5)
      xgb_model.fit(x_train,y_train)
      y_pred=xgb_model.predict(x_test)
```

```
[18:23:33] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

```
print("Score:",xgb_model.score(x_test,y_test))
print("Mean squared error: ", mse(y_test, y_pred))
print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
print("R-squared: ", r2_score(y_test, y_pred))
```

```
Score: 0.9761926653192577
Mean squared error:  12160920.244740054
Root mean squared error:  3487.2511014752085
R-squared:  0.9761926653192577
```

R^2 and Root Mean Squared Error for XGB Regressor Model after standard scaling is:

Model 5 : XGBRegressor Model After Standard Scaling

```
[110] xgb_model = XGBRegressor(objective='reg:linear', nthread= 4, n_estimators= 500, max_depth= 6, learning_rate= 0.5)
      xgb_model.fit(x_train,y_train)
      y_pred=xgb_model.predict(x_test)
      print("Score:",xgb_model.score(x_test,y_test))
      print("Mean squared error: ", mse(y_test, y_pred))
      print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
      print("R-squared: ", r2_score(y_test, y_pred))
```

```
[18:45:41] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Score: 0.9756786716369836
Mean squared error:  12423471.103971377
Root mean squared error:  3524.6944695918505
R-squared:  0.9756786716369835
```

R^2 and Root Mean Squared Error for XGB Regressor Model after MinMax scaling is:

Model 5 : XGBRegressor Model after MinMax Scaling

```
xgb_model = XGBRegressor(objective='reg:linear', nthread= 4, n_estimators= 500, max_depth= 6, learning_rate= 0.5)
xgb_model.fit(x_train,y_train)
y_pred=xgb_model.predict(x_test)
print("Score:",xgb_model.score(x_test,y_test))
print("Mean squared error: ", mse(y_test, y_pred))
print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
print("R-squared: ", r2_score(y_test, y_pred))
```

```
[19:03:10] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Score: 0.9756800353968016
Mean squared error:  12422774.48779811
Root mean squared error:  3524.5956488366305
R-squared:  0.9756800353968016
```

R^2 and Root Mean Squared Error for XGB Regressor Model after Robust scaling is:

Model 5 : XGBRegressor Model after Robust Scaling

```
[124] xgb_model = XGBRegressor(objective='reg:linear', nthread= 4, n_estimators= 500, max_depth= 6, learning_rate= 0.5)
      xgb_model.fit(x_train,y_train)
      y_pred=xgb_model.predict(x_test)
      print("Score:",xgb_model.score(x_test,y_test))
      print("Mean squared error: ", mse(y_test, y_pred))
      print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
      print("R-squared: ", r2_score(y_test, y_pred))
```

```
[19:23:21] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Score: 0.9756880004445168
Mean squared error:  12418705.896697616
Root mean squared error:  3524.0184302437488
R-squared:  0.9756880004445166
```

6.ExtraTreesRegressor

R^2 and Root Mean Squared Error for ExtraTreesRegressor Model before scaling is:

Model 6: ExtraTreesRegressor Model

```
[102] from sklearn.ensemble import ExtraTreesRegressor
      xtr_model = ExtraTreesRegressor()
      xtr_model.fit(x_train,y_train)
      y_pred=xtr_model.predict(x_test)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DataConversionWarning: A column-vector y was passed wh
  This is separate from the ipykernel package so we can avoid doing imports until
```

```
print("Score:",xgb_model.score(x_test,y_test))
print("Mean squared error: ", mse(y_test, y_pred))
print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
print("R-squared: ", r2_score(y_test, y_pred))
```

```
Score: 0.9761926653192577
Mean squared error:  12911861.990276553
Root mean squared error:  3593.3079453724185
R-squared:  0.9747225527700482
```

R^2 and Root Mean Squared Error for ExtraTreesRegressor Model after standard scaling is:

Model 6: ExtraTreesRegressor Model After Standard Scaling

```
xtr_model = ExtraTreesRegressor(n_estimators=30,n_jobs=4)
xtr_model.fit(x_train,y_train)
y_pred=xtr_model.predict(x_test)
print("Score:",xgb_model.score(x_test,y_test))
print("Mean squared error: ", mse(y_test, y_pred))
print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
print("R-squared: ", r2_score(y_test, y_pred))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DataConversionWarning: A column-vector y was passed when a 1d array was expected.

Score: 0.9756786716369836
Mean squared error:  12686668.380413407
Root mean squared error:  3561.834973775934
R-squared:  0.9751634124689921
```

R^2 and Root Mean Squared Error for ExtraTreesRegressor Model after MinMax scaling is:

Model 6: ExtraTreesRegressor Model after MinMax Scaling

```
xtr_model = ExtraTreesRegressor(n_estimators=30,n_jobs=4)
xtr_model.fit(x_train,y_train)
y_pred=xtr_model.predict(x_test)
print("Score:",xgb_model.score(x_test,y_test))
print("Mean squared error: ", mse(y_test, y_pred))
print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
print("R-squared: ", r2_score(y_test, y_pred))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DataConversionWarning: A column-vector y was passed when a 1d array was expe

Score: 0.9756800353968016
Mean squared error:  13506513.550309706
Root mean squared error:  3675.1208892102727
R-squared:  0.9735584082461781
```

R^2 and Root Mean Squared Error for ExtraTreesRegressor Model after Robust scaling is:

Model 6: ExtraTreesRegressor Model after Robust Scaling

```
[125] xtr_model = ExtraTreesRegressor(n_estimators=30,n_jobs=4)
      xtr_model.fit(x_train,y_train)
      y_pred=xtr_model.predict(x_test)
      print("Score:",xgb_model.score(x_test,y_test))
      print("Mean squared error: ", mse(y_test, y_pred))
      print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
      print("R-squared: ", r2_score(y_test, y_pred))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DataConversionWarning: A column-vector y was passed when a 1d array was expec

Score: 0.9756880004445168
Mean squared error:  13038692.306918414
Root mean squared error:  3610.9129464608272
R-squared:  0.974474258090436
```

4.Choosing the Best Model

4.1 Comparing the R2 values of Machine Learning models

Model	Without Scaling	Standard Scaling	MinMax Scaling	Robust Scaling
Linear Regression model	0.0937627	0.0937591490	0.093762783	0.093762783
KNN Model	0.4456561	0.3085792090	0.465082077	0.308714920
Decision Tree Algorithm	0.9628832	0.960603511	0.962981842	0.960156163
Random Forest Algorithm	0.9776500	0.9657048454	0.965754850	0.965703674
XGB Regressor	0.97619266	0.9756786716	0.975688000	0.975688000
ExtraTreesRegressor	0.9753077	0.9743729184	0.973855148	0.97577245

The most common interpretation of R^2 is how well the regression model fits the observed data. For example, an R^2 of 60% reveals that 60% of the data fit the regression model. Generally, a higher R^2 indicates a better fit for the model. Here, we have four models with R2 value more than 95%. They are DecisionTreeRegressor, RandomForestRegressor, XGBRegressor and ExtraTreesRegressor.

4.2 Comparing the RMSE values of Machine Learning models

Model	Without Scaling	Standard Scaling	MinMax Scaling	Robust Scaling
Linear Regression model	21515.367976	21515.411119	21515.36797	21515.36797
KNN Model	16827.416974	18793.127006	16529.94491	18791.28257
Decision Tree Algorithm	4354.2469330	4485.9722960	4348.457487	4511.369553
Random Forest Algorithm	3378.8311039	4185.4698127	4182.4173149	4185.541254
XGB Regressor	3487.2511014	3524.6944695	3524.018430	3524.018430
ExtraTreesRegressor	3551.4703432	3618.0736839	3654.4407282	3517.891912

Here we have taken 4 models as their accuracies are more than 95%. The models are DecisionTreeRegressor, RandomForestRegressor, XGBRegressor and ExtraTreesRegressor.

On comparing the r2 score value and RMSE value, we can see that XGB Regressor has highest R2 score value and lowest RMSE value. With this we can conclude that XGB Regressor is the model that gives the maximum accurate value in predicting weekly sales.

5.Explorations & Findings

- There are 45 stores and 81 department in data. Departments are not same in all stores.
- Although department 72 has higher weekly sales values, on average department 92 is the best. It shows us, some departments have higher values as seasonal like Christmas and black Friday. It is consistent when we look at the top 5 sales in data, all of them belongs to 72th department at a holiday time.
- Although stores 10 and 35 have higher weekly sales values sometimes, in general average store 20 and store 4 are on the first and second rank. It means that some areas have higher seasonal sales.
- 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.
- As expected, holiday average sales are higher than normal dates.
- Year 2010 has higher sales than 2011 and 2012. But, November and December sales are not in the data for 2012. Even without highest sale months, 2012 is not significantly less than 2010, so after adding last two months, it can be first.
- It is obviously seen that week 51 and 47 have higher values and 50-48 weeks follow them. Interestingly, 5th top sales belong to 22th week of the year. This results show that Christmas and Black Friday are very important than other weeks for sales and 5th important time is 22th week of the year and it is end of the May, when schools are closed. Most probably, people are preparing for holiday at the end of the May.
- January sales are significantly less than other months. This is the result of November and December high sales. After two high sales months, people prefer to pay less on January.
- CPI, temperature, unemployment rate and fuel price have no pattern on weekly sales.

6. Conclusion

As a conclusion, following are the few insights drawn from analysis of the sales data of the Walmart. Walmart can maintain its store size between 100000 f. to 150000 ft to maintain optimum sales and in the month of August as many states in United States offer a tax free on sale, the sales shoot up though there were no markdowns in that month. The analysis shows that in the November and December. store types A & B shows a very high sale because of the special days like Good Friday and Christmas. High markdowns are not helping the sales, the ideal range of Markdown2 is (0 to 18000), markdown2 is (0 to 10000), markdown3 is (0 to 500), Markdown4 is (0 to 6000) and markdown5 is (0 to 12500). The relationship between the sales and temperature was varying with each and every department.

This paper dealt with the implementation of six algorithms namely, Linear Regression, KNN Model, Decision Tree, Random Forest, Gradient Boosting, and Extra Trees, on the Walmart dataset and a comparative analysis was carried out to determine the best algorithm. We find that XGB Regressor equation shows maximum accuracy (97.6% accuracy) in predicting the weekly sales. Walmart can use it to forecast the sales better. They need to focus on the inventory planning of key departments like 38,92 and 95. They need to overhaul the Markdowns that are given currently as they are not having the intended impact on sales. They need to focus on the year-end inventory as week 51 and 52 play a crucial part in predicting sales.

7. Enhancement Scope

Future work would include the fine-tuning of the hyperparameters of the models to improve the accuracy of prediction. Future work could also entail combining the models to produce an ensemble training model that could represent even the tiniest details present in the data. This work shows that there are highly efficient algorithms to forecast sales in big, medium or small organizations, and their use would be beneficial in providing valuable insight, thus leading to better decision-making.

8. Link to code and executable file

The source code along with the dataset is uploaded into GitHub. Following is the link to my GitHub repository for Walmart-Stores-Sales-Forecasting project –

https://github.com/anaghaisal/Walmart-Stores-Sales-Forecasting

This consist of:

1. Three dataset which I worked on
2. Source code
3. Project report

Loom link explaining the project –

https://www.loom.com/share/6bc4a91afc7e4bd7acbe5a28dce23806