WALMART SALES PREDICTION

Project Overview

With gaining competitive advantage becoming ever more difficult for retail stores, timely, accurate, and relevant forecasting has become a point of focus for increasing revenue and decreasing costs. It is of increasing pertinence during the holiday seasons for retail stores such as Walmart. With effective sales/demand forecasts, inventory and staffing requirements can be forecasted, allowing revenue maximization, possible cost reduction, or increased customer service.

With the ability to forecast which products, product categories, and store departments will have the greatest demands, understocking or stock-outs can be prevented to increase revenue, and losses can be minimized by preventing over-stocking of perishables. Adequate staffing based on demand can contribute to greater customer service.

Problem Statement

Weekly sales data for 45 stores and their individual departments have been made available. The data contains the size and type of each store, highlighted holiday weeks, and price mark-down data. Additional macro indicators have been included: CPI, Unemployment Rate, Fuel price, etc. Our task is to department-wide sales for each store.

Business Objectives and Constraints:

- Predict the department-wide sales for each store using machine learning.
- No strict latency constraints.

Data Overview:

The dataset is from Kaggle https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting/data and it is freely available for public use. The data field contains a total of 4 datasets: stores.csv, train.csv, test.csv, and features.csv.

Data Description

Walmart has provided historical weekly sales data for 45 Walmart stores located in various regions. In addition to historical sales data, they have also provided data on various aspects of their stores, details of which are presented in later sections. The data is presented in three CSV files: “stores.csv, “test.csv”, and “features.csv”.

Stores.csv:

This file provides information on store type and size of each of the 45 stores. The information has been anonymized. The variables and their descriptions are as follows:

Variable	Description
Store	Primary key used for identifying the store
Type	Nominal variable that classifies stores into three categories: ‘A’, ‘B’, ‘C’
Size	Ordinal variable that distinguishes each store by physical size

Train.csv:

Historical training data (sales) for Walmart’s stores and departments is provided within this file. It covers historical data from the time period *2010-02-05* to *2012-11-01.* The variables and their descriptions are as follows:

Variable	Description
Store	Primary key used for identifying the store
Dept	The department number used for identifying departments in each store
Date	The week
Weekly_sales	Sales amount for each department in each store (TARGET VARIABLE)
IsHolidy	Binary variable indicating whether the week is a holiday week or not

Features.csv:

This file provides covariate data regarding each store, department, and region for each week. The variables and their descriptions are as follows:

Variable	Description
Store	Primary key used for identifying the store
Date	The week
Temperature	Average temperature during the week
Fuel price	Cost of fuel in each region
Markdown 1.5	Promotional markdown information
CPI	Consumer price index
Unemployment	Unemployment rate
IsHolidy	Binary variable indicating whether the week is a holiday week or not

Machine Learning Object

With the goal of predicting weekly sales, our machine learning objective is to build prediction models. To further realize the models that can be utilized, we first conducted thorough data exploration and analysis. Data preprocessing was performed as needed.

Data Preprocessing

Various preprocessing steps were taken prior to conducting data exploratory analysis. They are as follows:

Null Values:

Each csv file was analyzed for null values, with the following results showing “features.csv” with multiple null values:

```
#Check for null values in stores.
stores.isnull().any()
```

```
Store    False
Type     False
Size     False
dtype: bool
```

```
#Check for null values in train
train.isnull().any()
```

```
Store           False
Dept            False
Date            False
Weekly_Sales    False
IsHoliday       False
Temperature     False
Fuel_Price      False
MarkDown1       False
MarkDown2       False
MarkDown3       False
MarkDown4       False
MarkDown5       False
CPI             False
Unemployment    False
Type            False
Size            False
dtype: bool
```

```
#Check for null values in features.
features.isnull().any()
```

```
Store           False
Date            False
Temperature     False
Fuel_Price      False
MarkDown1        True
MarkDown2        True
MarkDown3        True
MarkDown4        True
MarkDown5        True
CPI              True
Unemployment     True
IsHoliday       False
dtype: bool
```

```
#Count of null values present
features.isnull().sum()
```

```
Store              0
Date               0
Temperature        0
Fuel_Price         0
MarkDown1       4158
MarkDown2       5269
MarkDown3       4577
MarkDown4       4726
MarkDown5       4140
CPI              585
Unemployment     585
IsHoliday          0
dtype: int64
```

Data Merge & Imputation:

The three csv files were merged, duplicate columns were removed, and rows with missing values were removed.

```
# Removing rows with null values in all columns
data.dropna(axis=0, how="all", inplace=True)
```

```
data.shape
```

```
(421570, 16)
```

```
# Removing all rows with null values in all rows
data.dropna(axis=1, how="all", inplace=True)
```

```
# Fill missing values with 0
data=data.fillna(0)
```

```
data.isna().sum()
```

```
Store           0
Dept            0
Date            0
Weekly_Sales    0
IsHoliday       0
Temperature     0
Fuel_Price      0
MarkDown1       0
MarkDown2       0
MarkDown3       0
MarkDown4       0
MarkDown5       0
CPI             0
Unemployment    0
Type            0
Size            0
dtype: int64
```

Data Validity:

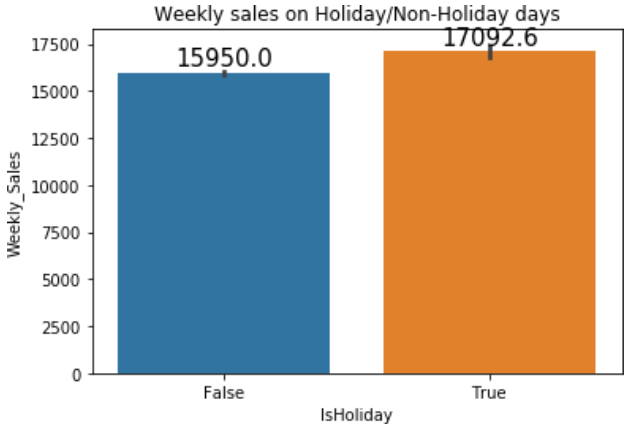
We have found that there are negative values for 'Weekly_sales. Given that sales cannot be negative, we have skipped the rows containing negative values.

	Store	Dept	Weekly_Sales	Te
count	421570.000000	421570.000000	421570.000000	4215
mean	22.200546	44.260317	15981.258123	
std	12.785297	30.492054	22711.183519	
min	1.000000	1.000000	-4988.940000	
25%	11.000000	18.000000	2079.650000	
50%	22.000000	37.000000	7612.030000	
75%	33.000000	74.000000	20205.852500	
max	45.000000	99.000000	693099.360000	1

Data Exploration

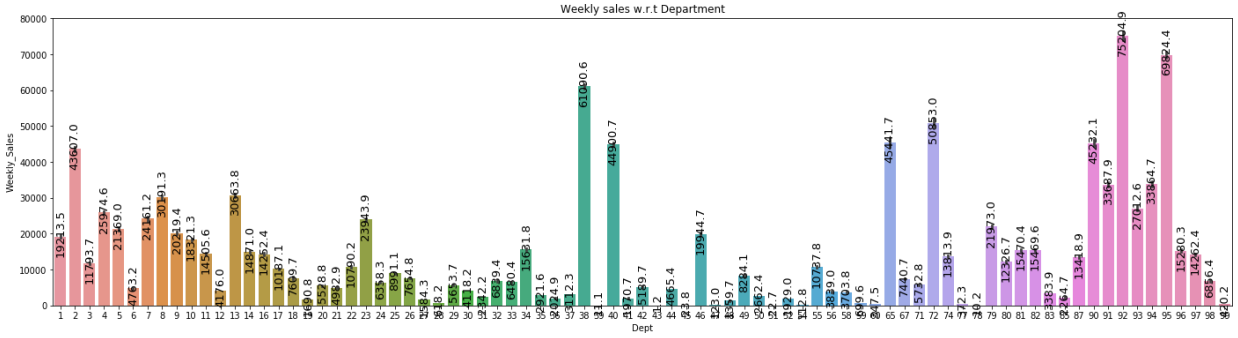
Holiday vs Non-holiday Weekly Sales:

We see in the following graph that weekly sales during holiday weeks are greater than during non-holiday weeks.



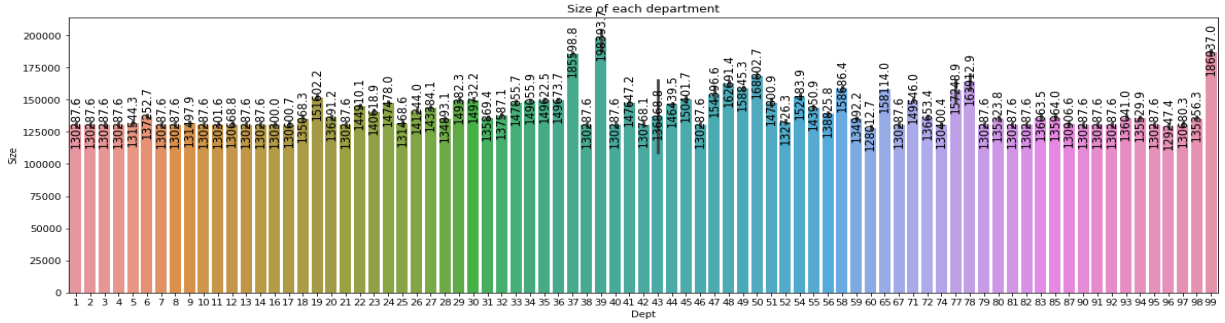
Weekly Sales by Department:

Department 92 has the highest recorder weekly sales with $483 million, followed by departments 95 and 38.



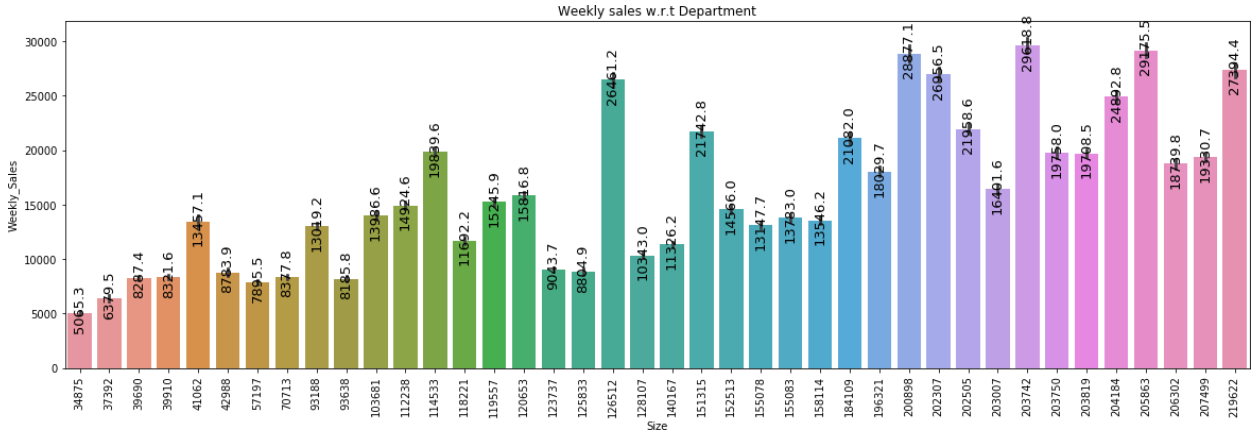
Weekly Sales by Department Size:

Departments are almost equal in size, however, dept 92, 95, and 38 have the highest sales, which may be due to price of items in these departments. We can therefore conclude that the department size is not proportional to weekly sales.



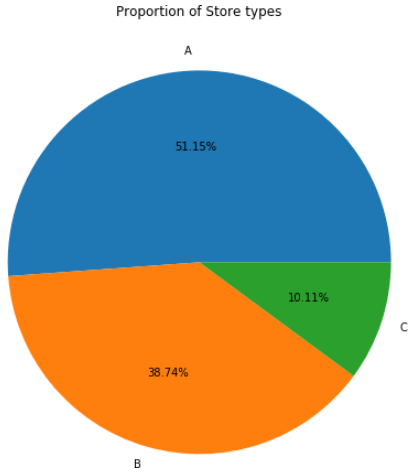
Weekly Sales by Store Size:

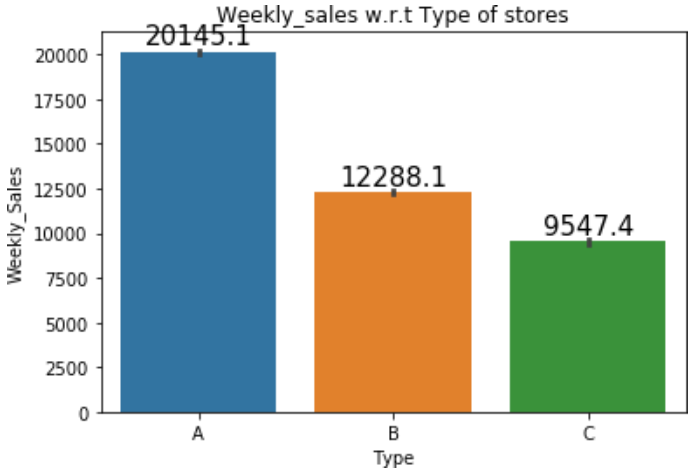
Weekly sales are higher for larger stores to a certain extent higher, however, it cannot be concluded that there is a relationship as some of the larger stores also had lower recorded weekly sales.

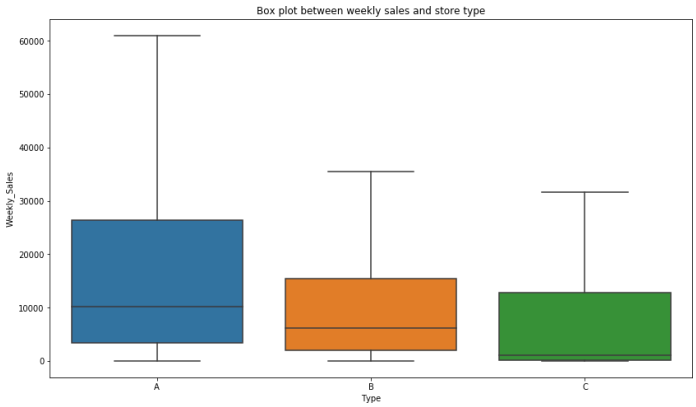


Distribution of Store Types & Sales:

There are three types of stores, ‘A’, ‘B’, and ‘C’. Type ‘A’ being the largest type, square footage wise. The following are their distributions, with type ‘A’ accounting for 51.15% of the stores. We also see that type ‘A’ stores have the largest volume of sales and a greater median of sales.

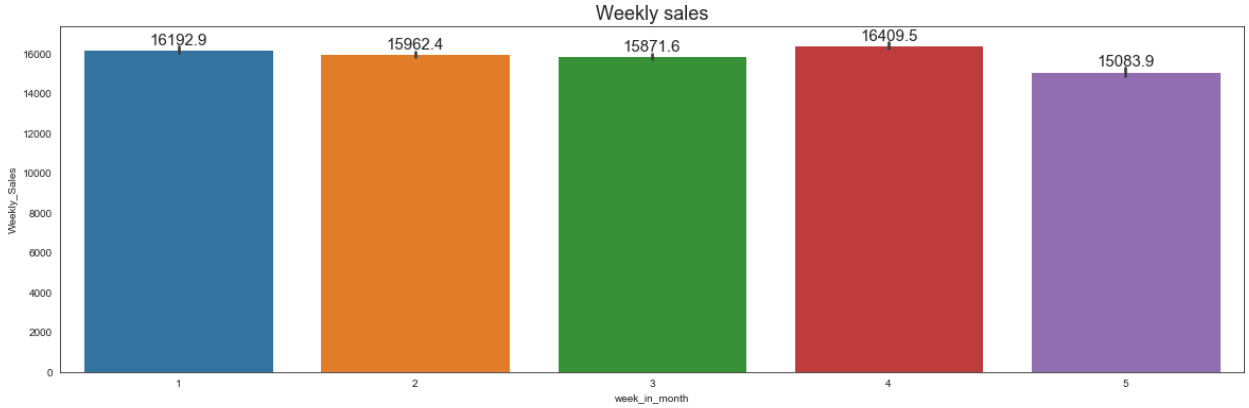






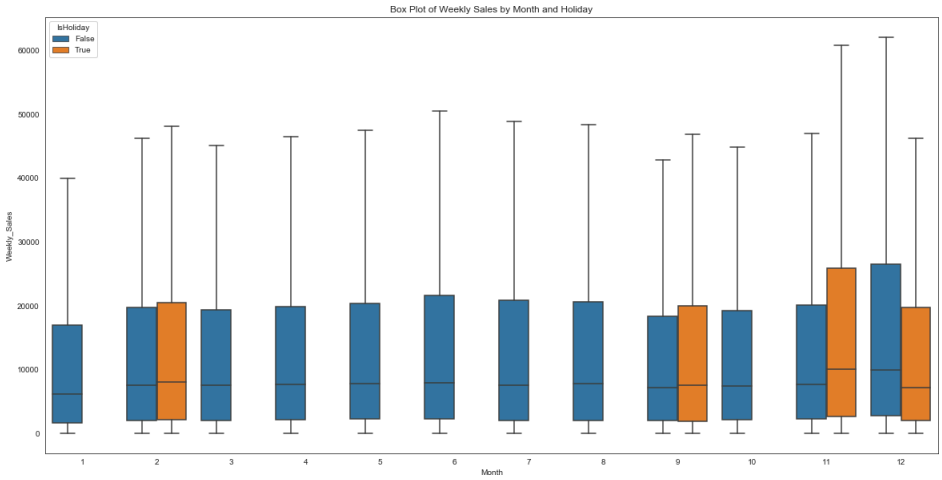
Sales by Week of the Month:

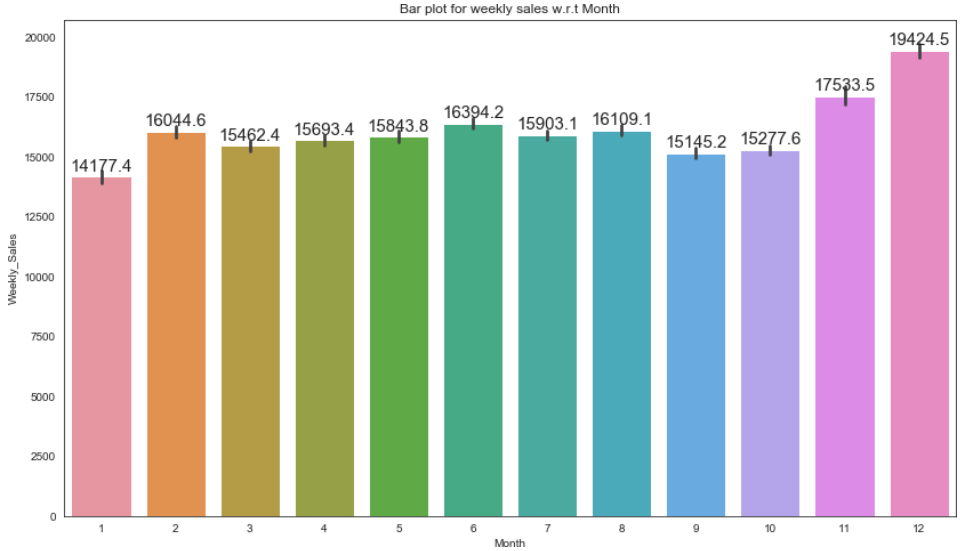
Weekly sales are on average higher in the 4th week but not significantly. Indicating that weekly sales are higher at the end of each month than the beginning.



Sales by Month of the Year:

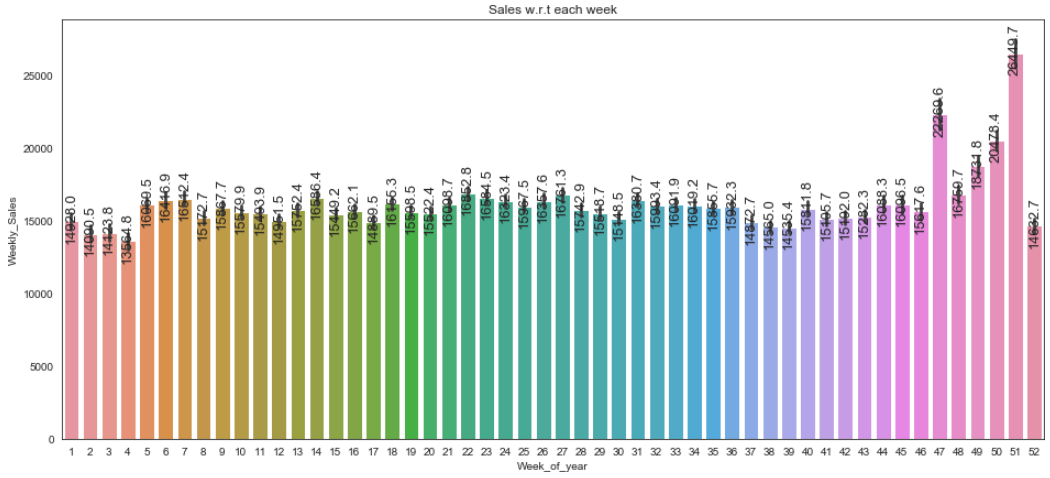
December and November, on average, have slightly higher sales; this may be due to the holiday seasons, Thanksgiving and Christmas.

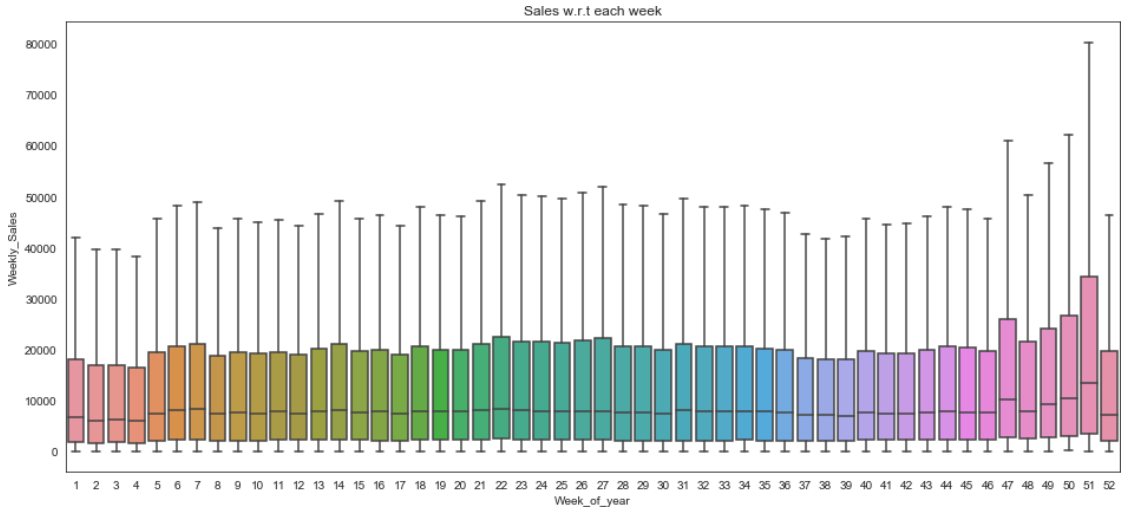




Sales by Each Week of the Year:

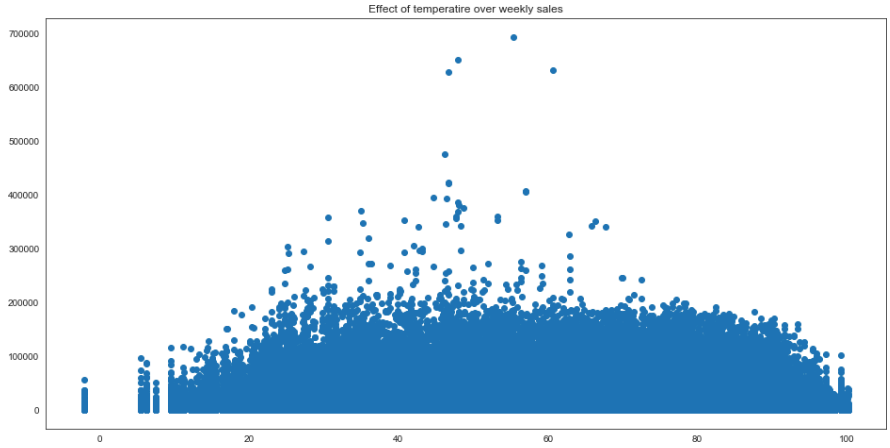
Sales during weeks 51 and 47 are significantly higher. Which may be due to the holidays, Thanksgiving and Christmas.





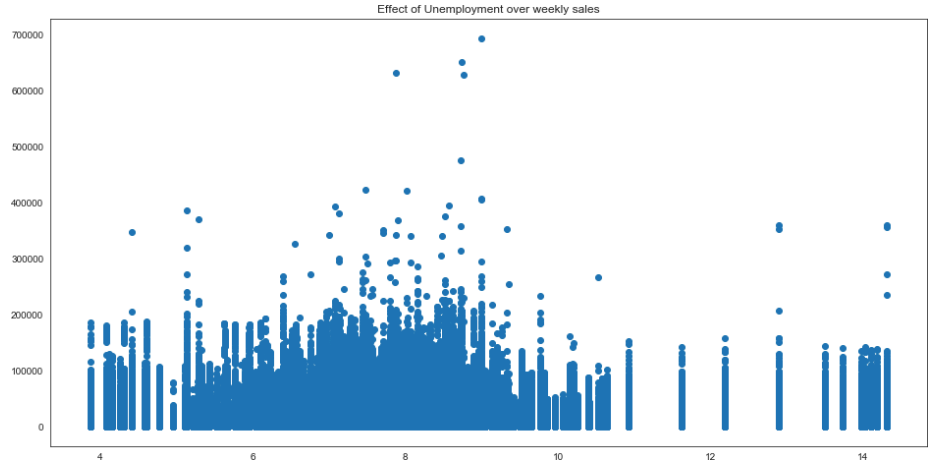
Temperature & Weekly Sales:

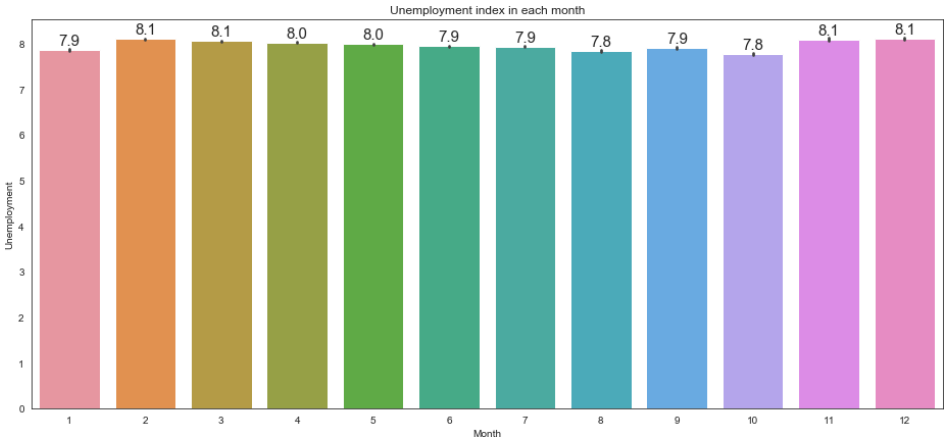
Average temperature range is between 40- and 60-degrees Fahrenheit, with greater sales occurring within this mean range.

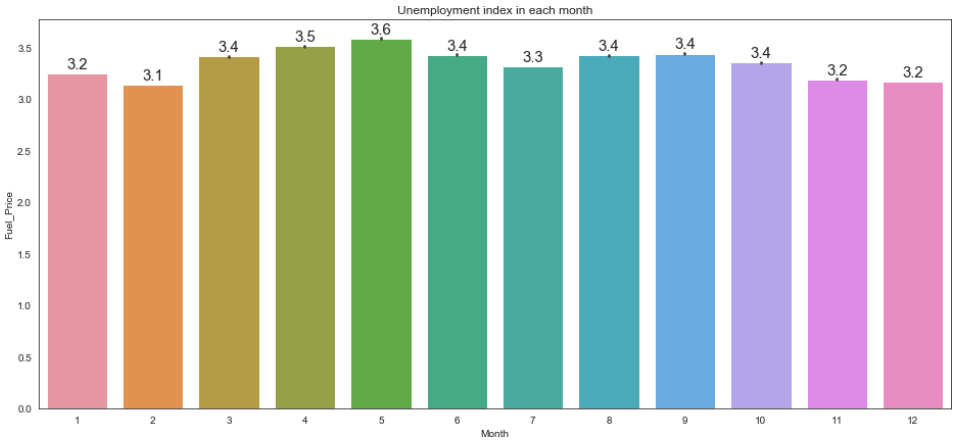


Unemployment Rate & Weekly Sales:

Although unemployment rises from 7.8 to 8.1 percent, there is still an increase in sales, indicating that there may not be a relationship.

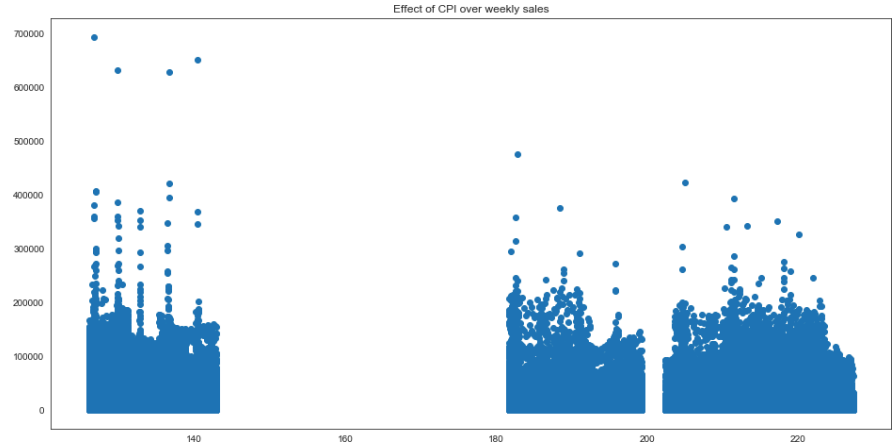






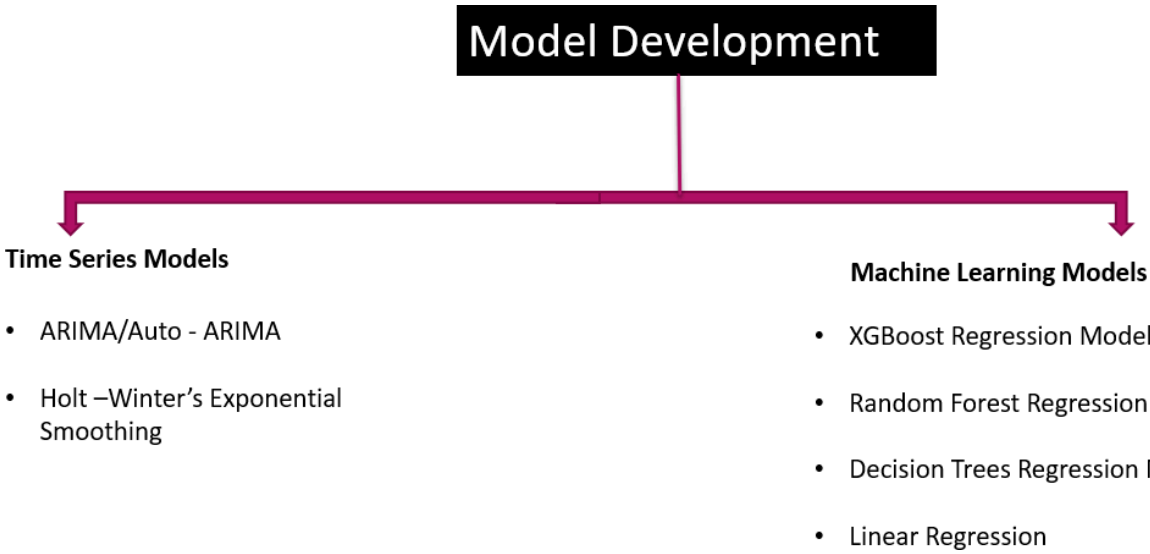
Consumer Price Index & Weekly Sales:

The data for CPI is incomplete.



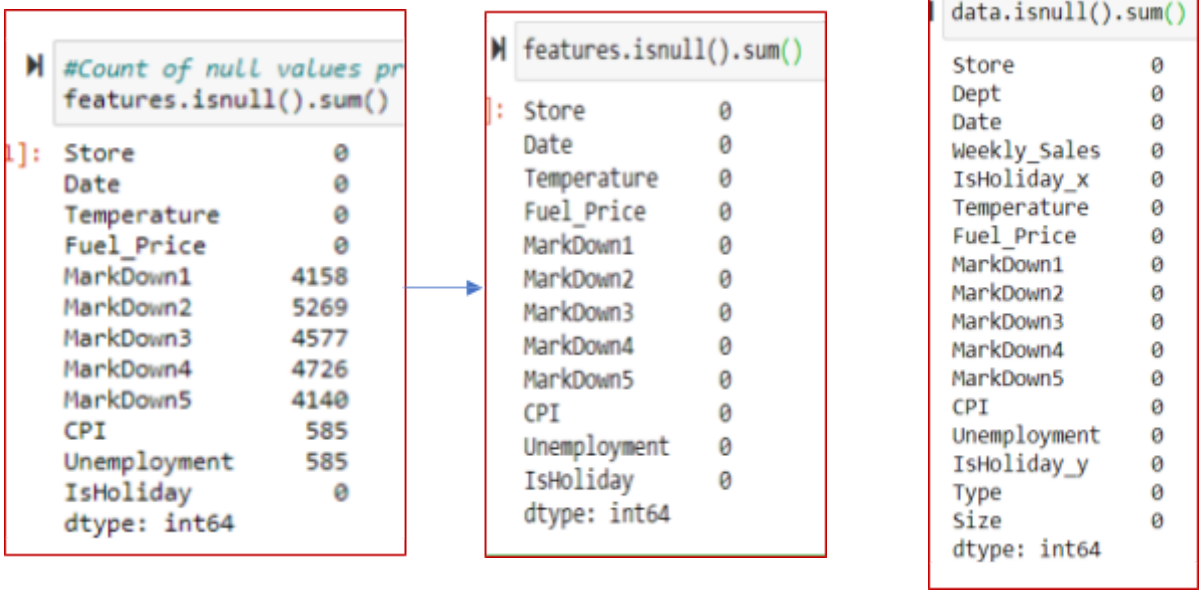
Model Development:

Below are the models that we used in our analysis. In total there were six models among which two of them were time series models and remaining were machine learning models.



Data Pre-processing for Time Series Analysis:

- Before applying any model on our data, we did some data preparation for our analysis.
- Among the three CSV files we have used, features.csv file has some missing values present. Missing values as in there were some NAN values present in some of the columns so we imputed it with 0. Below is a screenshot for reference.



```
#Count of null values pr
features.isnull().sum()

Store             0
Date              0
Temperature       0
Fuel_Price        0
MarkDown1      4158
MarkDown2      5269
MarkDown3      4577
MarkDown4      4726
MarkDown5      4140
CPI             585
Unemployment    585
IsHoliday         0
dtype: int64
```

```
features.isnull().sum()

Store           0
Date            0
Temperature     0
Fuel_Price      0
MarkDown1       0
MarkDown2       0
MarkDown3       0
MarkDown4       0
MarkDown5       0
CPI             0
Unemployment    0
IsHoliday       0
dtype: int64
```

```
data.isnull().sum()

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

- Next, we merged three csv files using inner join based on Store and date column.

Merge files

```
#Merging the three csv files using inner join.
data = train.merge(features, on=['Store', 'Date'], how='inner').merge(stores, on=['Store'], how='inner')
print(data.shape)
```

```
(421570, 17)
```

❖ Sample dataset after merging:

Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	Type	Size
1	1	2010-02-05	24924.50	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	8.106	A	151315
1	2	2010-02-05	50605.27	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	8.106	A	151315
1	3	2010-02-05	13740.12	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	8.106	A	151315
1	4	2010-02-05	39954.04	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	8.106	A	151315
1	5	2010-02-05	32229.38	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	8.106	A	151315

- ❖ It is necessary to have the date columns present in the dataset to be available in the datetime format as in the ARIMA model it is required to see the sales values in date-wise fashion. Hence, we converted the string formatted Date into datetime format. Then resampled the time series data with month starting first.
- ❖ Next, the data was split into a training set (70%) and test set (30%). A training set was used to fit the model and a test was used to evaluate the best model to get an estimation of generalization error.

```
data.Date = pd.to_datetime(data.Date,format='%Y-%m-%d')
data.index = data.Date
data = data.drop('Date', axis=1)
```

```
data = data.resample('MS').mean() # Resmapling the time series data with month starting first
```

```
# Train-Test splitting of time series data
train_data = data[:int(0.7*(len(data)))]
test_data = data[int(0.7*(len(data))):]

print('Train:', train_data.shape)
print('Test:', test_data.shape)
```

```
Train: (23, 20)
Test: (10, 20)
```

Time Series Analysis:

A time series is a sequence of numerical data points in successive order. In investing, a time series tracks the movement of the chosen data points, such as a security's price, over a specified period of time with data points recorded at regular intervals. There is no minimum or maximum amount of time that must be included, allowing the data to be gathered in a way that provides the information being sought by the investor or analyst examining the activity.

❖ Applications:

The usage of time series models is twofold:

- ❖ Obtain an understanding of the underlying forces and structure that produced the observed data
- ❖ Fit a model and proceed to forecasting, monitoring or even feedback and feedforward control.

Time Series Analysis is used for many applications such as:

- ❖ Economic Forecasting
- ❖ Sales Forecasting
- ❖ Budgetary Analysis
- ❖ Stock Market Analysis
- ❖ Yield Projections
- ❖ Process and Quality Control
- ❖ Inventory Studies
- ❖ Workload Projections

❖ Techniques:

The fitting of time series models can be an ambitious undertaking. There are many methods of model fitting including the following:

Box-Jenkins ARIMA model:

ARIMA stands for Autoregressive Integrated Moving Average model. Univariate (single vector) ARIMA is a forecasting technique that projects the future values of a series based entirely on its own inertia. Below steps will compute forecasting values of the input univariate data using ARIMA.

❖ Auto-Regressive (AR):

AR implies that there may be a degree of present and future time findings from past time points. The ARIMA model takes into account lagged observations to arrive at predicted observations. However, a weight is applied to past remarks; the weight can vary depending on how recent the past comments are. The more recent, the greater weight is given to the current, past observation.

❖ Integrated (I):

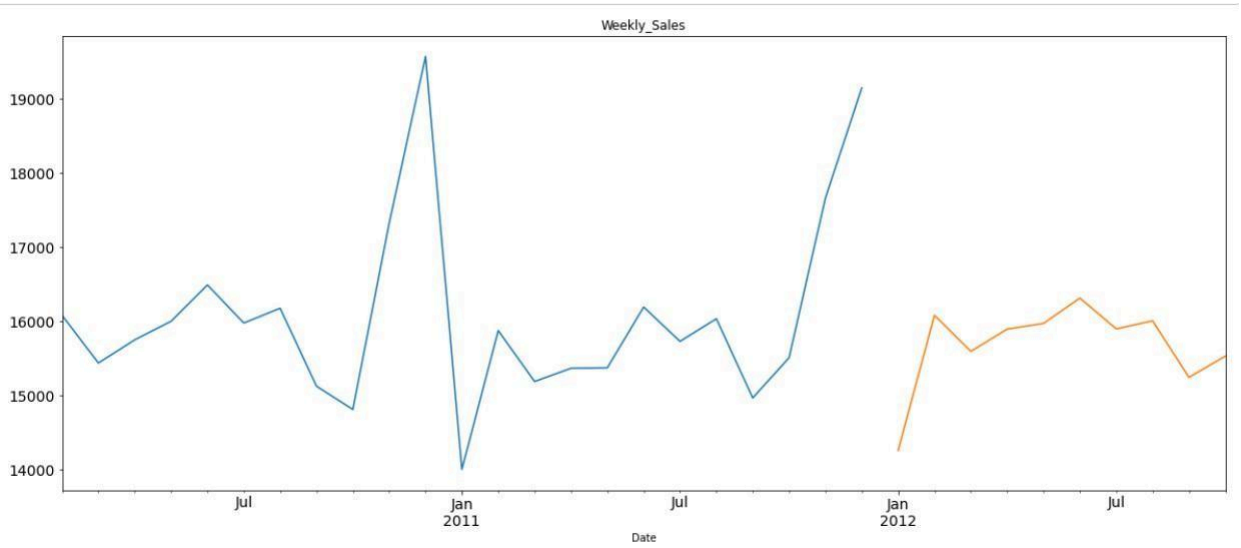
When there are persistent patterns in past price movement, it is most definitely a non-stationary indication that seasonality exists in past price movement. Integrated eliminates our dataset's seasonality process in case clear trends suggest this is the case. The degree of differentiation available in ARIMA models removes the trend in seasonality.

❖ Moving Average (MA):

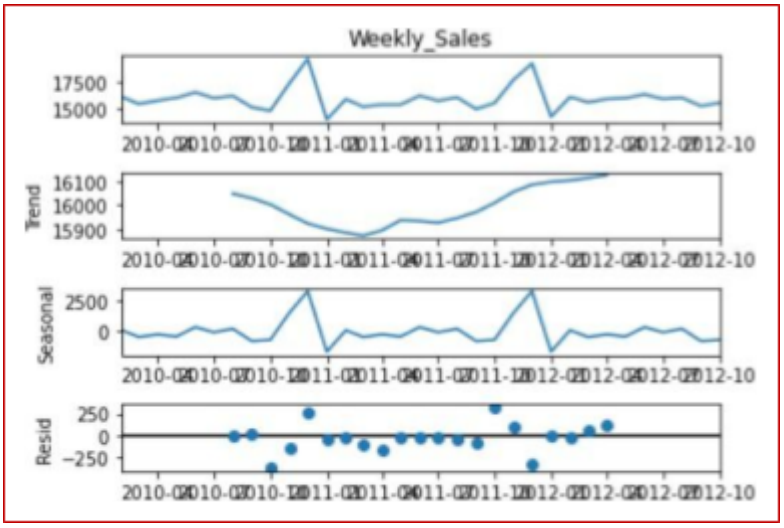
In our case, moving average helps to reduce the effect of spontaneous weekly sales price movements. If there was an unusual event that led to a rise in prices for weekly sales, moving average would help us "smooth things up" and our model time series would not be vulnerable to such fluctuations.

Below are the steps used for forecasting values of the input univariate data using ARIMA.

- ❖ First, we loaded the data.
- ❖ Visualized the available univariate data in a timely fashion. Below is the plot of Weekly_Sales with respect to years in train and test. The blue line indicates the train data and the orange line used for test data.



- ❖ Saw the seasonal decomposition parameter to find out for seasonality, trend etc. available or not. Below figure shows the trend of weekly sales is such that it first decreases then elevates. Given the dataset is seasonal since it is repeating the same pattern in the month of Nov-Dec.



- Performed a test of stationarity using the Augmented Dicky Fuller Test. If the time series is non- stationary, we have to make it stationary else no need to make it stationary.

❖ Stationarity:

A stationary time series is one whose statistical properties such as mean, variance, and autocovariance are all constant and not a function of time.

❖ Augmented Dicky Fuller Test:

The test is a type of statistical test which is called a root unit. The theory behind a root-unit test is that it decides how strongly a pattern describes a time series.

Null Hypothesis (H0): The test's null hypothesis is that a unit root that isn't stationary will represent the time series.

Alternative Hypothesis (H1): The test's alternative Hypothesis is that the time series is stationary.

Interpreting the value p:

P value > 0.05: Accepts the Null Hypothesis (H0), has the root unit and is non-stationary.

P value < = 0.05: Null hypothesis (H0) is denied, data stationary

As we can see in the output below, our p-value is definitely less than 0.05 and is even less than 0.01 so we can say with pretty good confidence that we can reject the null and can assume our data is stationary. Additionally, our ADF is much less than our 1% confidence value of –3.76, so we have another confirmation that we can reject the null.

```
ADF Statistic: -4.23944720683554
p-value: 0.0005645989175856225
Critical Values:
        1%: -3.769732625845229
        5%: -3.005425537190083
        10%: -2.6425009917355373
```

- The most important step is to find the appropriate values of triplet p, d, q, where p is number of Auto Regressors, d is Integration or difference and q is number of Moving Average by plotting ACF and PACF graphs so we used Auto-ARIMA over ARIMA. In Auto-ARIMA we get to see the triplet (p, d, q) values automatically which more importantly best fits the model.

  Auto-ARIMA library comes under pyramid utility function, hence installed pyramid library and applied the Auto-ARIMA model on the train data. Below is the output of the model and the p, d, q values selected by it (0,0,2)

```
Out[29]: ARIMA(order=(0, 0, 2), scoring_args={}, seasonal_order=(0, 0, 0, 1),
                suppress_warnings=True)
```

- Below is the summary of the model.

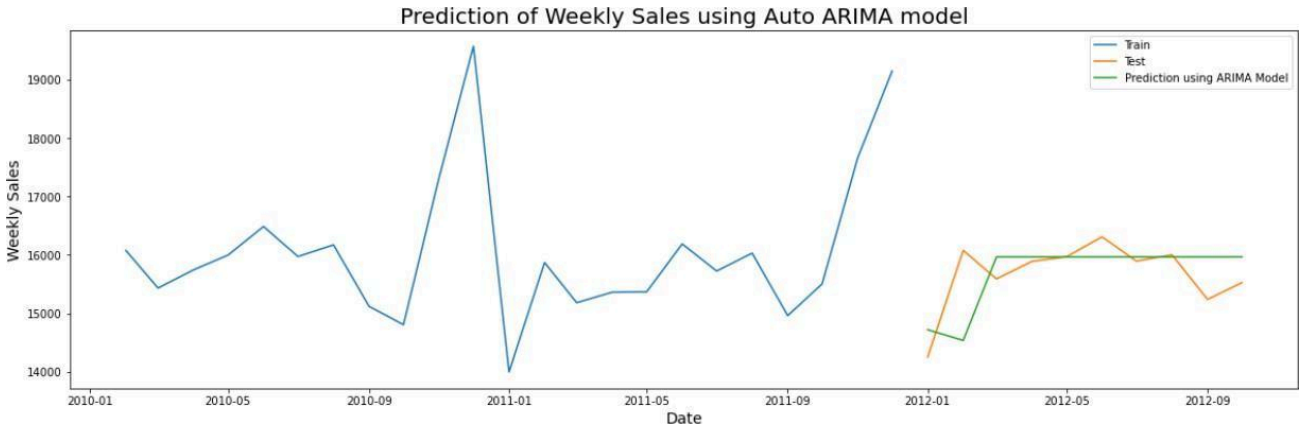
```
In [30]: model_auto_arima.summary()
```

Out[30]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	23
Model:	SARIMAX(0, 0, 2)	**Log Likelihood**	-194.496
Date:	Tue, 17 Nov 2020	**AIC**	396.992
Time:	17:14:58	**BIC**	401.534
Sample:	0	**HQIC**	398.135
	- 23		
Covariance Type:	opg		

- Forecasted the test data values and plotted the visualization of train, test and predicted test data values on a single graph. Below graph shows the prediction of weekly sales by a green line.



- Calculated Root mean squared error (RMSE) on predicted values.

```
# Performance metric for ARIMA model -MSE/RMSE
print('Mean Squared Error (MSE) of ARIMA: ', mean_squared_error(test_data, forecast))
print('Root Mean Squared Error (RMSE) of ARIMA: ', math.sqrt(mean_squared_error(test_data, forecast)))
print('Mean Absolute Deviation (MAD) of ARIMA: ', mean_absolute_error(test_data, forecast))
```

```
Mean Squared Error (MSE) of ARIMA:  358444.2749512625
Root Mean Squared Error (RMSE) of ARIMA:  598.7021587995674
Mean Absolute Deviation (MAD) of ARIMA:  408.0065499192995
```

Holt-Winters Exponential Smoothing:

Holt-Winters is a time-series model and is a way to model three aspects of the time series: a typical value (average), a slope (trend) over time, and a cyclical repeating pattern (seasonality). Holt-Winters uses exponential smoothing to encode lots of values from the past and use them to predict “typical” values for the present and future.

- It is another suite of techniques that also uses historical values. However, a key distinguishing feature is the so-called “exponential smoothing”.

- Exponential smoothing can handle variability within a series by smoothing out white noise. In ARIMA a Moving Average can smooth training data, but it does so by taking an average of past values and by weighting them equally. On the other hand, in Exponential Smoothing, most recent observations are given higher weights than far-away values.

Below steps are required to use Holt-Winters model.

- Below is the summary of the model.

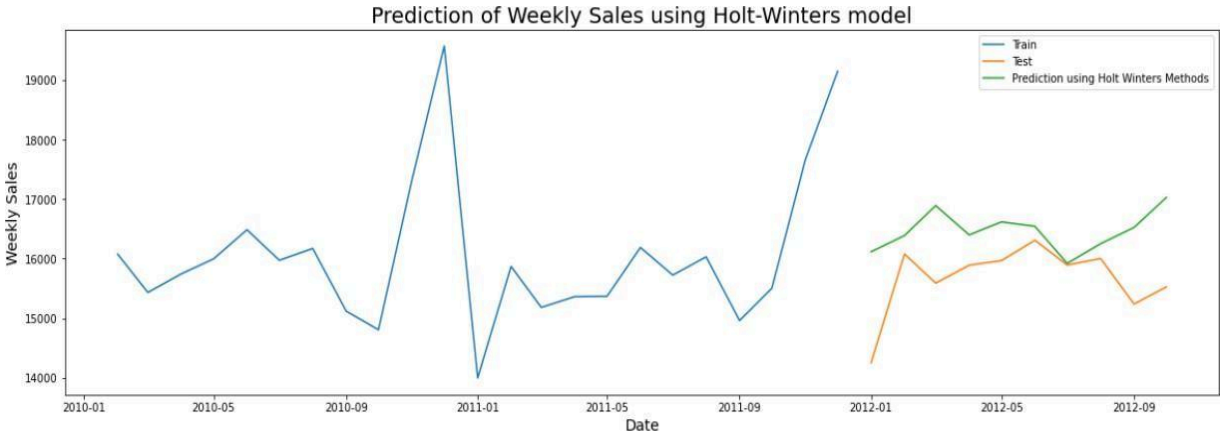
```
In [37]: model_holt_winters.summary()
```

Out[37]:

ExponentialSmoothing Model Results

Dep. Variable:	endog	**No. Observations:**	23
Model:	ExponentialSmoothing	**SSE**	43898740.629
Optimized:	True	**AIC**	354.624
Trend:	Additive	**BIC**	367.114
Seasonal:	Additive	**AICC**	395.068
Seasonal Periods:	7	**Date:**	Tue, 17 Nov 2020
Box-Cox:	False	**Time:**	17:14:58
Box-Cox Coeff.:	None		

- Fit the model using Holt-Winters method on train data and predict the values using test data. In the graph below we visualized the training data, test data and predicted data. Prediction of weekly sales is represented by a green line.



- Calculate Root Mean Squared Error (RMSE) on predicted data

```
# Performance metric for Holt-Winters model -MSE/RMSE
print('Mean Squared Error (MSE) of Holt-Winters: ', mean_squared_error(test_data, pred))
print('Root Mean Squared Error (RMSE) of Holt-Winters: ', math.sqrt(mean_squared_error(test_data, pred)))
print('Mean Absolute Deviation (MAD) of Holt-Winters: ', mean_absolute_error(test_data, pred))
```

```
Mean Squared Error (MSE) of Holt-Winters:  997082.2242852871
Root Mean Squared Error (RMSE) of Holt-Winters:  998.5400464104016
Mean Absolute Deviation (MAD) of Holt-Winters:  793.2124194512116
```

Model Comparison: ARIMA v. Holt Winters:

We compared the results of the two models that we used on the basis of RMSE and AIC.

- **The Akaike information criterion (AIC).** Given a collection of models for the data, AIC estimates the quality of each model, relative to each of the other models.

- **Root Mean Square Error (RMSE)** is the standard deviation of the residuals (prediction errors).

As shown in the figure below, Holt winter seems to be a better model in terms of AIC. While AIC is great at comparing models within the same class (e.g. within ARIMA models), it should not be used to compare two very different model classes (e.g. ARIMA vs HW) so we used RMSE value to compare the two. Since the ARIMA model has a lower RMSE score it is a better model for prediction.

	Model Name	RMSE Score	AIC
1	ARIMA	598.70	396.99
2	HOLT-WINTER	998.54	354.62

Data Pre-Processing for Machine Learning Model

- **Missing Value Imputation:** Data field contains a total of 3 datasets: stores.csv, train.csv and features.csv. Dataset features.csv had missing values for columns MarkDown1, MarkDown2, MarkDown3, MarkDown4, MarkDown5, CPI and Unemployment. The values of MarkDown1-5, CPI and Unemployment were NAN which were imputed with zero. Fillna(0) was used for imputation.

- Python code for count of missing values in features.csv dataset.

```
In [10]: #Count of null values present in respective columns of features.
         features.isnull().sum()

Out[10]: Store              0
         Date               0
         Temperature        0
         Fuel_Price         0
         MarkDown1       4158
         MarkDown2       5269
         MarkDown3       4577
         MarkDown4       4726
         MarkDown5       4140
         CPI              585
         Unemployment     585
         IsHoliday          0
         dtype: int64
```

There are null values present in below columns.

- MarkDown1, MarkDown2, MarkDown3, MarkDown4, MarkDown5, CPI and Unemployment

❖ Python Code for imputing missing values for columns MarkDown1-4

```
In [17]: features =features.fillna(0)

In [18]: features.isnull().sum()

Out[18]: Store          0
         Date           0
         Temperature    0
         Fuel_Price     0
         MarkDown1      0
         MarkDown2      0
         MarkDown3      0
         MarkDown4      0
         MarkDown5      0
         CPI            0
         Unemployment   0
         IsHoliday      0
         dtype: int64
```

❖ **Merging of datasets:** The three datasets stores.csv, train.csv and features.csv were merged using inner join and store as the primary key for analysis.

Merge files

```
]: #Merging the three csv files using inner join.
   data = train.merge(features, on=['Store', 'Date'], how='inner').merge(stores, on=['Store'], how='inner')
   print(data.shape)

   (421570, 17)
```

❖ **Removal of duplicate columns after merging datasets:** The merged dataset had duplicate columns like “IsHoliday_y” and “IsHoliday_x”. Hence, “IsHoliday_y” column was dropped and “IsHoliday_x” was renamed to “IsHoliday”.

❖ Python Code showing the duplicate columns in the merged dataset “data”

```
In [19]: #Merging the three csv files using inner join.
         data = train.merge(features, on=['Store', 'Date'], how='inner').merge(stores, on=['Store'], how='inner')
         print(data.shape)

         (421570, 17)

In [20]: data.isnull().sum()

Out[20]: Store          0
         Dept           0
         Date           0
         Weekly_Sales   0
         IsHoliday_x    0
         Temperature    0
         Fuel_Price     0
         MarkDown1      0
         MarkDown2      0
         MarkDown3      0
         MarkDown4      0
         MarkDown5      0
         CPI            0
         Unemployment   0
         IsHoliday_y    0
         Type           0
         Size           0
         dtype: int64
```

❖ Python code for dropping column “IsHoliday_y” and renaming “IsHoliday_x”

Remove duplicate columns after merging files

```
In [12]: #Removing additional IsHoliday column (IsHoliday_y) and renaming original IsHoliday_x column to IsHoliday.
         data = data.drop(['IsHoliday_y'], axis=1)
         data = data.rename(columns={'IsHoliday_x':'IsHoliday'})
         data.columns

Out[12]: Index(['Store', 'Dept', 'Date', 'Weekly_Sales', 'IsHoliday', 'Temperature',
                'Fuel_Price', 'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4',
                'MarkDown5', 'CPI', 'Unemployment', 'Type', 'Size'],
               dtype='object')
```

Splitting the data

❖ Data was split into train and test sets in the ratio 70:30.

One Hot Encoding

❖ It is a process by which categorical variables are converted to numeric form. The two variables in our dataset were “Type” and “IsHoliday”.

❖ **Need for one hot encoding**: Some machine learning algorithms cannot operate on categorical data directly. They require all input and output variables to be numeric.

❖ **What does one hot encoding do?**

It takes a column which has categorical data, which has been label encoded and then splits the column into multiple columns.

❖ Dataset prior to one hot encoding

```
In [ ]: data.head()
```

Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	Type	Size
24924.50	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	8.106	A	151315
50605.27	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	8.106	A	151315
13740.12	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	8.106	A	151315
39954.04	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	8.106	A	151315
32229.38	False	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	8.106	A	151315

❖ So, after we split the data into train and test in the ratio 70:30, we defined our one hot encoder.

Step1 – After creating the encoder, the first step was to fit the encoder to the training set.

One hot encoding is applied to the "Type" variable after 70:30 split

- Step 1: Fit/train the one hot encoder to X_train set
- Step 2: Transform X_train set using the trained one hot encoder
- Step 3: Transform X_test set using the encoder trained on X_train set

```
#Step: 1
from feature_engine.categorical_encoders import OneHotCategoricalEncoder
ohe_enc = OneHotCategoricalEncoder(
    top_categories=None,
    variables=['Type','IsHoliday'], # we can select which variables to encode
    ) # to return k-1, false to return k


ohe_enc.fit(X_train.fillna('Missing'))
```

❖ **Step2 – In this step, the training set was transformed using the trained encoder.**

```
#Step: 2
X_train = ohe_enc.transform(X_train.fillna('Missing'))

X_train.head()
```

[18]:

	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	Size	Type_B	Type_A	Type_C
se	75.20	3.827	0.0	0.0	0.0	0.0	0.0	135.783742	9.863	93638	1	0	0
se	36.53	2.826	0.0	0.0	0.0	0.0	0.0	131.901968	8.512	152513	0	1	0
se	76.25	2.958	0.0	0.0	0.0	0.0	0.0	136.372289	7.982	204184	0	1	0
se	28.70	3.231	0.0	0.0	0.0	0.0	0.0	137.054243	8.549	103681	1	0	0
se	75.11	2.603	0.0	0.0	0.0	0.0	0.0	214.984655	7.564	207499	0	1	0

❖ **Step3 – The same encoder was used to transform the test set.**

```
In [19]: #Step: 3
X_test = ohe_enc.transform(X_test.fillna('Missing'))

X_test.head()
```

Out[19]:

IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	Size	Type_B	Type_A
False	91.17	3.794	0.00	0.00	0.00	0.00	0.00	129.150774	13.503	112238	1	0
False	76.86	3.786	0.00	0.00	0.00	0.00	0.00	215.154482	7.931	203750	0	1
False	82.50	4.055	548.62	0.00	54.43	57.18	2628.10	130.838161	7.170	39690	0	0
False	91.07	3.684	0.00	0.00	0.00	0.00	0.00	216.107120	6.529	34875	1	0
False	25.56	3.703	7763.37	7366.75	33.48	3995.44	2642.17	137.258310	4.261	155083	0	1

Evaluation Metric:

❖ Walmart has provided Weighted Mean Absolute Error (WMAE) metric, the mathematical function for which is shown below:

$$\text{WMAE} = \frac{1}{\sum w_i} \sum_{i=1}^{n} w_i |y_i - \hat{y}_i|$$

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

❖ Before applying any regression models, we defined the most important metric WMAE (Weighted Mean Absolute Error). We want to predict sales in the holiday week as accurately as possible. By applying a weight of 5 to the sales prediction error during the holiday week, we are applying a 5 times higher penalty to the prediction error during this (holiday) week. This will help us to optimize our machine learning models as accurately as possible especially during the holiday week.

❖ Our aim was to lower WMAE (Weighted Mean Absolute Error) as much as possible along with predicting weekly sales.

- Another challenge was to incorporate this equation in Python. This was done using an anonymous function. In Python, an anonymous function is a function that is defined without a name. Normal functions are defined using def in Python. Anonymous functions are defined using the lambda keyword. Anonymous functions are also called lambda functions.

- Following is the snippet of the code that was used to incorporate this metric.

- Python Code:

```
def wmae_train(test, pred): # WMAE for train
  weights = X_train['IsHoliday'].apply(lambda is_holiday:5 if is_holiday else 1)
  error = np.sum(weights * np.abs(test - pred), axis=0) / np.sum(weights)
  return error

def wmae_test(test, pred): # WMAE for test
  weights = X_test['IsHoliday'].apply(lambda is_holiday:5 if is_holiday else 1)
  error = np.sum(weights * np.abs(test - pred), axis=0) / np.sum(weights)
  return error
```

Machine Learning Models

The dataset comes from the Kaggle platform and consists of data from Walmart Inc. It comprises data from 45 Walmart department stores mainly centered around their sales on a weekly basis. The dataset has 282,452 entries that will be used for training the models. Each entry has attributes as follows: the associated store (recorded as a number), the corresponding department (81 departments, each entered as a number), the date of the starting day in that week, departmental weekly sales, the store size, and a Boolean value specifying if there is a major holiday in the week. The major holidays being one of Thanksgiving, Labor Day, Christmas or Easter. Along with the aforementioned attributes is a parallel set of features for each entry including Consumer Price Index, unemployment rate, temperature, fuel price, and promotional markdowns. Since there is no test-set provided, they are generated from the given training data for cross-validation, and final testing.

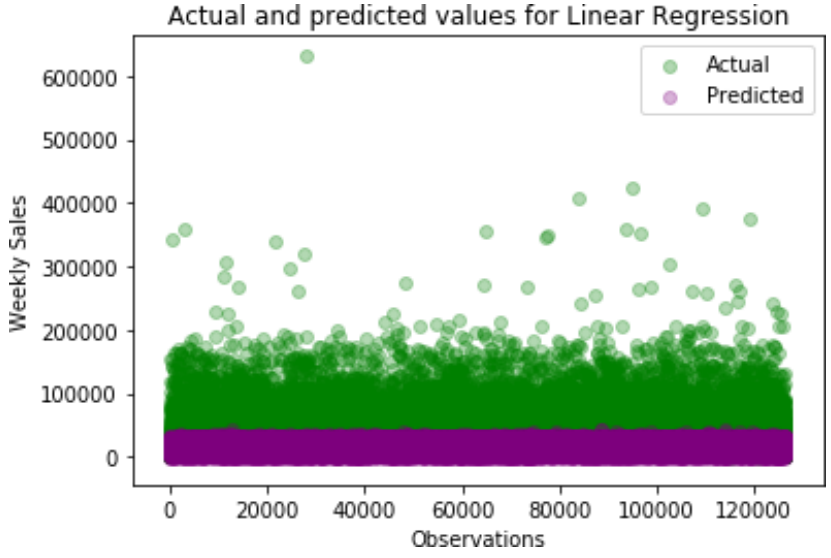
❖ Methods:

Four machine learning (ML) models were constructed for this project using the following algorithms: Linear Regression, Decision Trees, Random Forest, Extreme Gradient Boosting (XG-Boost). All models were implemented in Python 3.7. on the Anaconda distribution using Jupyter Notebooks. The code used for the implementation has been added to the appendix.

A. Linear Regression

Linear regression is a linear approach to modelling the relationship between a scalar response and one or more explanatory variables. The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression. In ML, Linear Regression is a supervised learning algorithm where the predicted output is continuous and has a constant slope. It's used to predict values within a continuous range, (e.g. sales, price) rather than trying to classify them into categories (e.g. cat, dog).

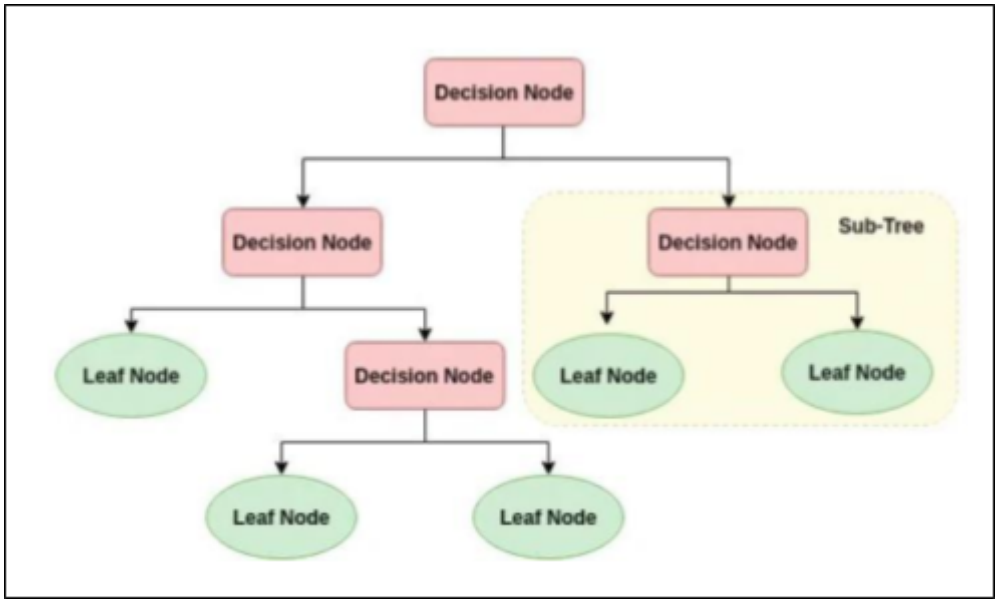
Linear regression models require that all continuous variables should satisfy normality assumptions. In our project we used normalized data to build the regression model. The following figure shows the output of linear regression model as a plot of actual vs predicted weekly sales. The weighted mean average error for this model is 14913.115. This high WMAE value is due to the fact that the model fails to generalize major trends in the dataset, especially sales during holidays. Closer inspection of the plot shows that the linear regression model is not able to predict high weekly sales, which occur mostly during holiday weeks.



Actual vs Predicted weekly sales – Linear Regression

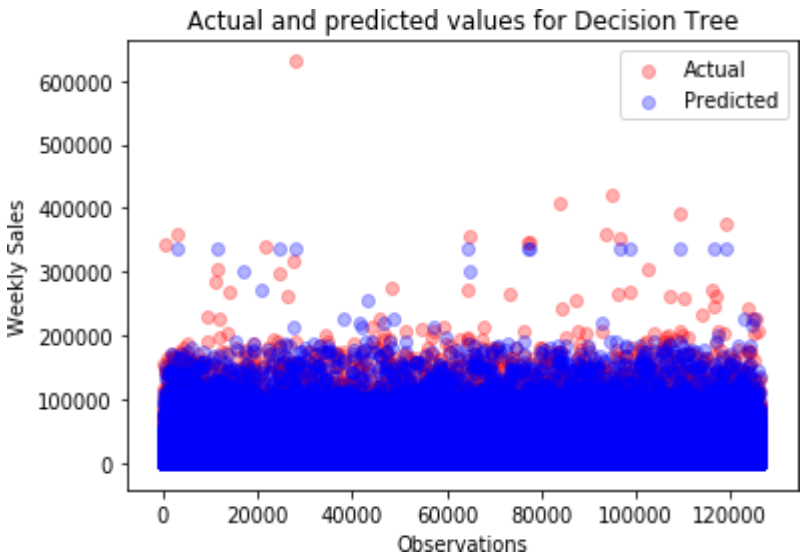
B. Decision Tree

A decision tree is one of most frequently and widely used supervised machine learning algorithms that can perform both regression and classification tasks. The tree architecture is described in the following figure**.**



Decision tree algorithm logic

For each attribute in the dataset, the decision tree algorithm forms a node, where the most important attribute is placed at the root node. For evaluation we start at the root node and work our way down the tree by following the corresponding node that meets our condition or "decision". This process continues until a leaf node is reached, which contains the prediction or the outcome of the decision tree. Tree-based algorithms have a disadvantage of sometimes overfitting the training set and performing poorly on test sets. Overfitting increases when the tree gets deeper. So, it is important to limit the max depth of the decision tree.

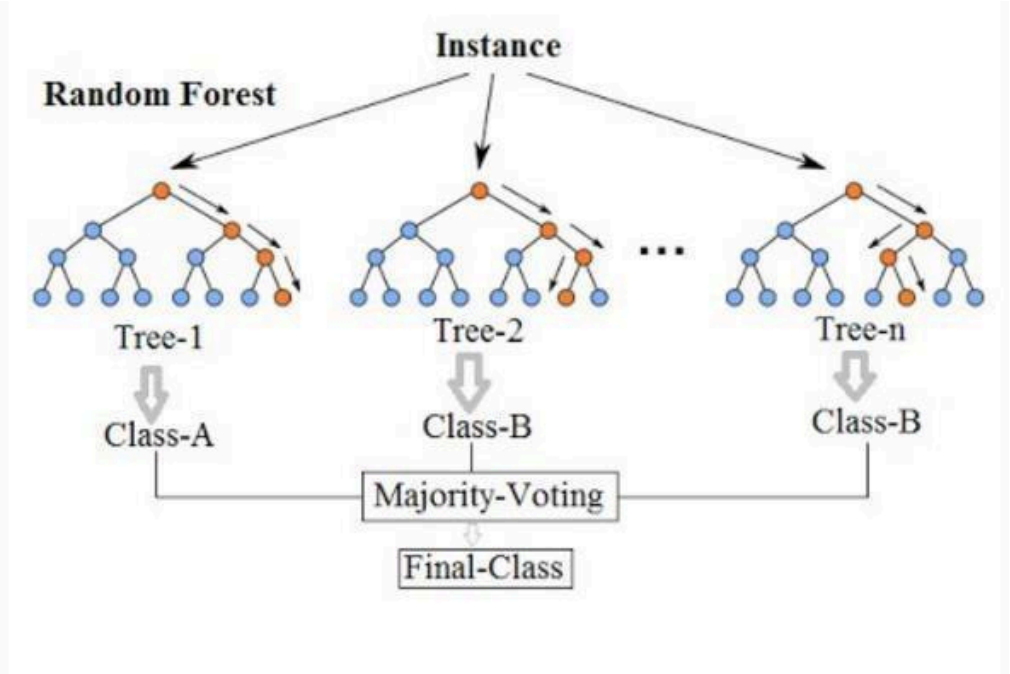


Actual vs Predicted weekly sales – Decision Tree; WMAE = 2525.476

For this project we built a decision tree model using optimized hyperparameters as discussed previously. Our decision tree model improved the prediction accuracy significantly and was able to predict high weekly sales during holiday weeks better than the linear regression model. Above figure shows the actual vs prediction plot for the decision tree model. The WMAE was also reduced significantly compared to the linear regression model.

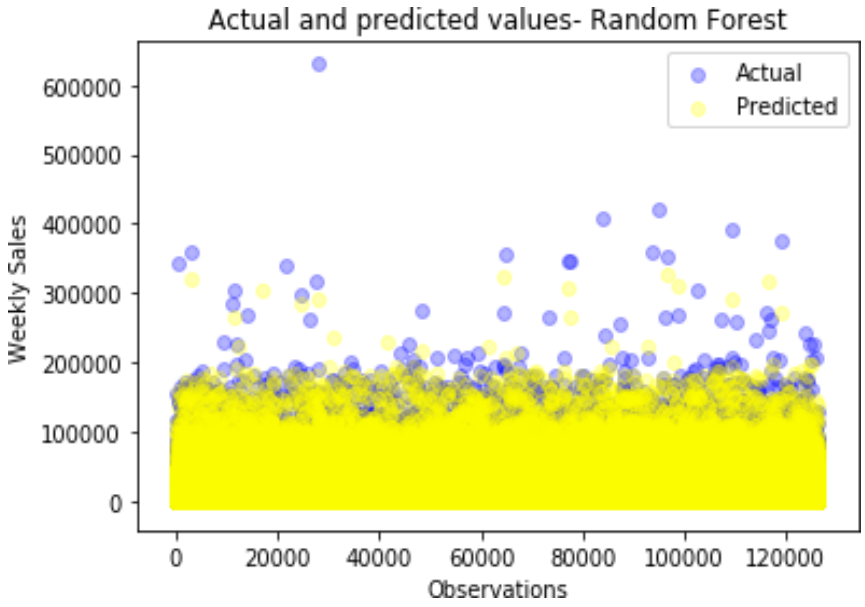
C. Random Forest Regression

Random forest regression is an ensemble learning method for regression that operates by constructing a multitude of decision trees at training time and outputting mean/average prediction of the individual trees. Random forest corrects for trees' habit of overfitting to their training set by adding some noise to each tree during training. This helps it to capture more general trends in the training dataset rather than noise. The Random Forest architecture is described by the figure on the next page**.** As more trees are grown, the Random Forest algorithm adds more randomness to the model. It searches for the best feature amidst a random subset of features in place of searching for the most relevant feature while splitting a node. This results in a more accurate model as it leads to a much greater diversity. Thus, in Random Forest, only a random subset of the features is considered by the algorithm for diverging a node. Trees can be made more random by using random thresholds for each feature instead of searching for the best thresholds (like a normal decision tree does).



Simplified random forest algorithm architecture.

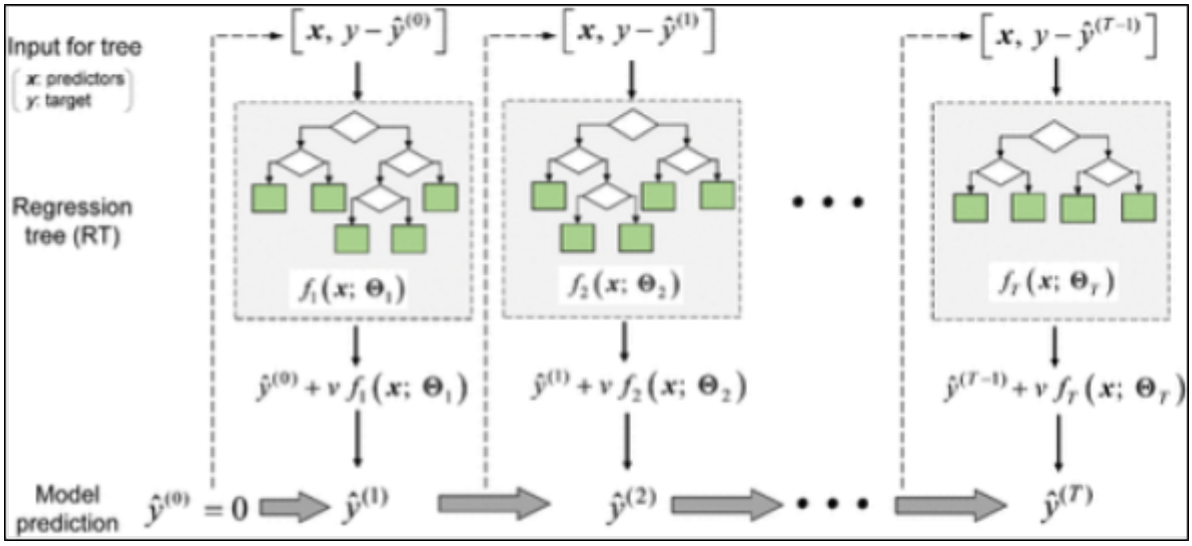
The following figure shows a plot of actual vs predicted values of the weekly sales with the hyperparameters set at the optimized values. Similar to the decision tree algorithm, the random forest algorithm is also able to predict holiday weekly sales with higher accuracy. The prediction error, calculated as WMAE, is further lowered when compared to decision trees.



Actual vs Predicted weekly sales – Random Forest; WMAE = 2231.912

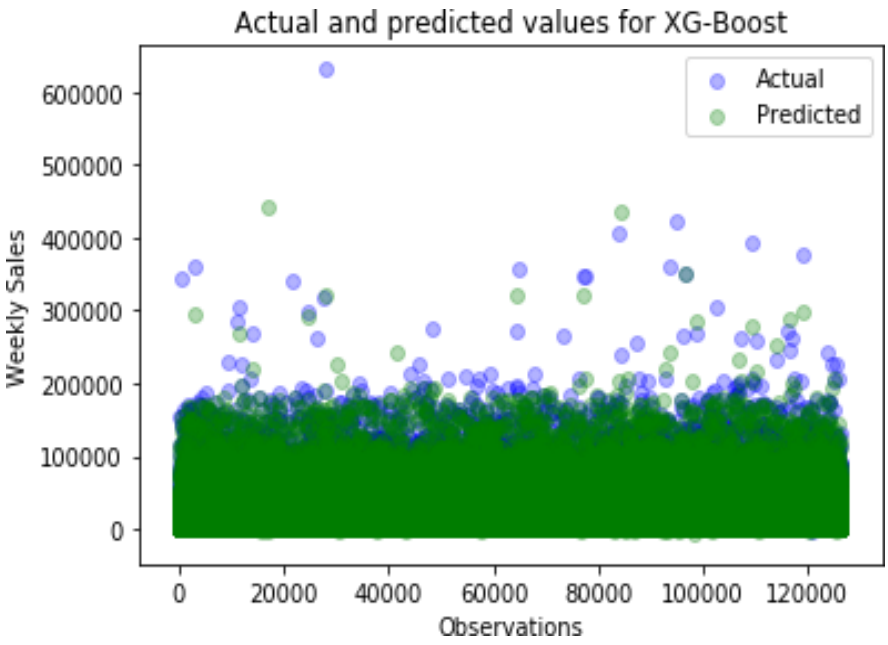
D. Extreme Gradient Boosting (XG-Boost) Regression

XG-Boost regression is a decision-tree-based ensemble Supervised Machine Learning algorithm that uses a gradient boosting framework. XG-Boost in general performs better than random forest algorithms. A dominant factor of XG-Boost is regularization, which is used to avoid overfitting the model on training sets. Another important factor is cross-validation which is inbuilt. Finally, XG-Boost is designed in such a way that it can handle missing values. It finds out the trends in the missing values and apprehends them. Figure shows behind the scene working of boosting algorithms. Behind the scenes, XG-Boost carries out a gradient boosting decision tree algorithm. Boosting is nothing but ensemble techniques where previous model errors are resolved in the new models. These models are added straight until no other improvement is seen.



XG-Boost algorithm depiction.

The following figure shows a plot of actual vs predicted values of weekly sales as an output of XG- Boost algorithm.



Actual vs Predicted weekly sales – XG-Boost; WMAE = 2146.34

Model Comparison

The following table shows a comparison of all of our machine learning models based on the WMAE score. Based on that XG-Boost model performs the best as it has the lowest WMAE score.

Model Name	WMAE Score (70:30)
Linear Regression	14913.12
XGBoost Regression	2146.34
Decision Tree Regression	2525.48
Random Forest Regression	2131.91

Future Scope of The Project

This project was a proof of concept study based on weekly sales data of Walmart obtained from 2010-12. Our model development approach shows that the models we built are able to accurately predict the weekly sales, especially during holiday weeks. We have used WMAE (Weighted Mean Absolute Error) as our evaluation metric for machine learning models and RMSE

(Root Mean Squared Error) for time series models. For future studies, we can use the same evaluation metric to compare machine learning and time series models on recent data to predict weekly sales and to further gain insights on the most efficient technique to be used for prediction.

APPENDIX

The next page contains the python code for our Walmart Sales Prediction project.

Import Libraries

In [1]:
```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Import datasets

In [2]:
```
#Loading the data from csv files.
features = pd.read_csv(r"C:\Users\gayat\Desktop\AI\walmart-recruiting-store-s
stores = pd.read_csv(r"C:\Users\gayat\Desktop\AI\walmart-recruiting-store-sal
train = pd.read_csv(r"C:\Users\gayat\Desktop\AI\walmart-recruiting-store-sale
test = pd.read_csv(r"C:\Users\gayat\Desktop\AI\walmart-recruiting-store-sales
```

stores.csv

In [3]:
```
print(f"Shape of stores.csv: {stores.shape}")
stores.head()
```

```
Shape of stores.csv: (45, 3)
```

Out[3]:

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

In [4]:
```
#Check for null values in stores.
stores.isnull().any()
```

Out[4]:
```
Store    False
Type     False
Size
False dtype:
bool
```

In [5]:
```
#Check for null values in train
train.isnull().any()
```

Out[5]:
```
Store           False
Dept            False
Date            False
Weekly_Sales    False
IsHoliday       False
dtype: bool
```

In [6]:
```
#Check for null values in features.
features.isnull().any()
```

Out[6]:
```
Store           False
Date            False
Temperature     False
Fuel_Price      False
MarkDown1        True
MarkDown2        True
MarkDown3        True
MarkDown4        True
MarkDown5        True
CPI              True
Unemployment     True
IsHoliday       False
dtype: bool
```

In [7]:
```
#Count of null values present in respective columns of features.
features.isnull().sum()
```

Out[7]:
```
Store              0
Date               0
Temperature        0
Fuel_Price         0
MarkDown1       4158
MarkDown2       5269
MarkDown3       4577
MarkDown4       4726
MarkDown5       4140
CPI              585
Unemployment     585
IsHoliday          0
dtype: int64
```

There are null values present in below columns.

- MarkDown1, MarkDown2, MarkDown3, MarkDown4, MarkDown5, CPI and Unemployment

Merge files

In [8]:
```
#Merging the three csv files using inner join.
data = train.merge(features, on=['Store', 'Date'], how='inner').merge(stores,
print(data.shape)
```

```
(421570, 17)
```

In [9]:
```
#Original shape of train, features and stores data.
print('train: ', train.shape)
print('features: ', features.shape)
print('stores ', stores.shape)
```

```
train:  (421570, 5)
features:  (8190, 12)
stores  (45, 3)
```

Remove duplicate columns after merging files

In [10]:
```
#Removing additional IsHoliday column
(IsHoliday_y) and renaming original IsH
data = data.drop(['IsHoliday_y'], axis=1)
data =
data.rename(columns={'IsH
oliday_x':'IsHoliday'})
data.columns
```

Out[10]:
```
Index(['Store', 'Dept', 'Date', 'Weekly_Sales', 'IsHoliday', 'Temperature',
       'Fuel_Price', 'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4',
       'MarkDown5', 'CPI', 'Unemployment', 'Type', 'Size'],
      dtype='object')
```

In [11]:
```
data.shape
```

Out[11]: (421570, 16)

Missing data imputation

In [12]:
```
# Removing rows with null values in all columns
data.dropna(axis=0, how="all", inplace=True)
```

In [13]:
```
data.shape
```

Out[13]: (421570, 16)

In [14]:
```
# Removing all rows with null values in all rows
data.dropna(axis=1, how="all", inplace=True)
```

In [15]:
```
# Fill missing values with 0
data=data.fillna(0)
```

In [16]:
```
data.isna().sum()
```

Out[16]:
```
Store           0
Dept            0
Date            0
Weekly_Sales    0
IsHoliday       0
Temperature     0
Fuel_Price      0
MarkDown1       0
MarkDown2       0
MarkDown3       0
MarkDown4       0
MarkDown5       0
CPI             0
Unemployment    0
Type            0
Size            0
dtype: int64
```

In [17]:
```
print('Final Data Shape: ', data.shape)
```

```
Final Data Shape:  (421570, 16)
```

Exploratory Data Analysis

In [18]:
```
data.describe()
```

Out[18]:

	Store	Dept	Weekly_Sales	Temperature	Fuel_Price	MarkDo
count	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	421570.00
mean	22.200546	44.260317	15981.258123	60.090059	3.361027	2590.07
std	12.785297	30.492054	22711.183519	18.447931	0.458515	6052.38
min	1.000000	1.000000	-4988.940000	-2.060000	2.472000	0.00
25%	11.000000	18.000000	2079.650000	46.680000	2.933000	0.00
50%	22.000000	37.000000	7612.030000	62.090000	3.452000	0.00
75%	33.000000	74.000000	20205.852500	74.280000	3.738000	2809.05
max	45.000000	99.000000	693099.360000	100.140000	4.468000	88646.76

By seeing the statistics of the dataframe we come to know that there are some rows for which Weekly sales have negative values. Since sales values can't be negative, we skipped those rows having negative weekly sales.

In [19]:
```
data = data[data['Weekly_Sales'] >= 0]
```

In [20]:
```
data.describe()
```

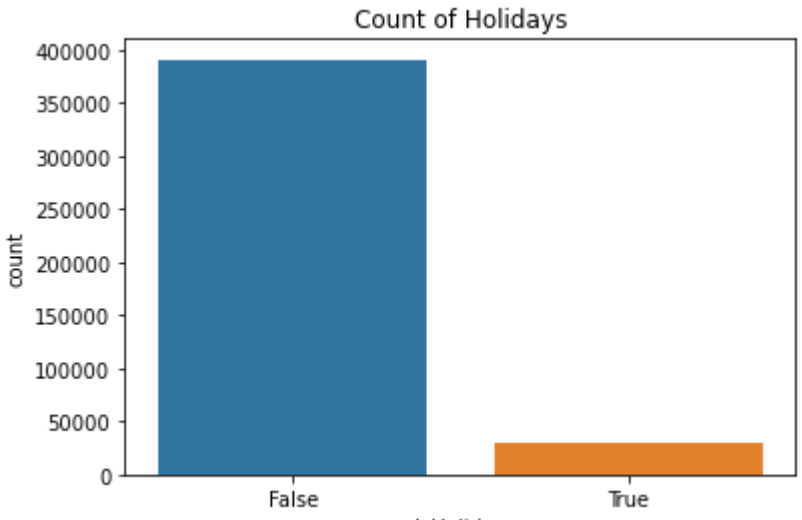
Out[20]:

	Store	Dept	Weekly_Sales	Temperature	Fuel_Price	MarkDo
count	420285.000000	420285.000000	420285.000000	420285.000000	420285.000000	420285.00
mean	22.195477	44.242771	16030.329773	60.090474	3.360888	2590.18
std	12.787213	30.507197	22728.500149	18.448260	0.458523	6053.22
min	1.000000	1.000000	0.000000	-2.060000	2.472000	0.00
25%	11.000000	18.000000	2117.560000	46.680000	2.933000	0.00
50%	22.000000	37.000000	7659.090000	62.090000	3.452000	0.00
75%	33.000000	74.000000	20268.380000	74.280000	3.738000	2801.50
max	45.000000	99.000000	693099.360000	100.140000	4.468000	88646.76

In [21]:
```
data['IsHoliday'].value_counts()
```

Out[21]:
```
False    390722
True      29563
Name: IsHoliday, dtype: int64
```

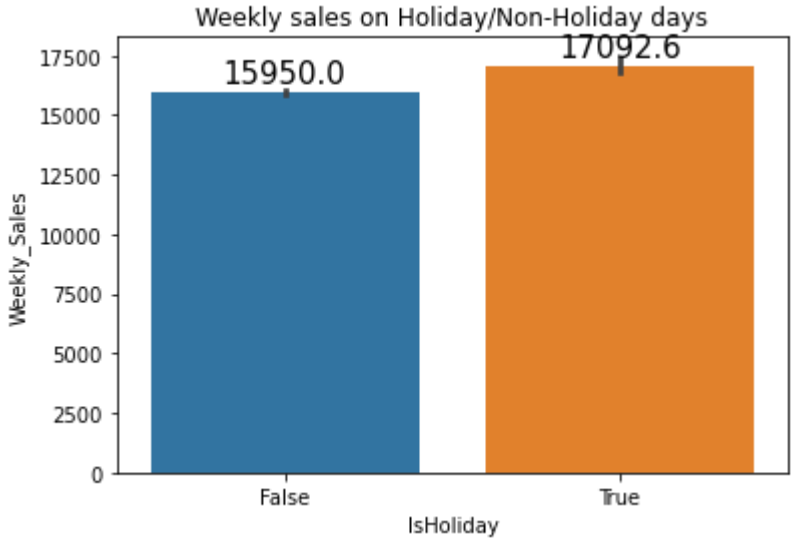
In [22]:
```
plt.title('Count of Holidays')
sns.countplot(data['IsHoliday'])
plt.show()
```



Weekly Sales on Holidays Vs Non-Holiday Days

In [23]:
```
def annotate_horizontal(splot):
    for p in splot.patches:
        splot.annotate(format(p.get_height(), '.1f'),
(p.get_x() + p.get_width() / 2., p.get_height()), ha = 'center', va =
                    'center',
size=15,

xytext = (0, 9),
textcoords = 'offset points')

def annotate_vertical(splot):
    for p in splot.patches:
splot.annotate(format(p.get_height(), '.1f'),
(p.get_x() + p.get_width() / 2., p.get_height()), ha = 'center', va =
                    'center',
size=13,
                     rotation=90 ,
                    xytext = (0, 4),
textcoords = 'offset points')
```

In [24]:
```
plt.title("Weekly sales on Holiday/Non-Holiday days")
splot=sns.barplot(x='IsHoliday', y='Weekly_Sales', data=data)
annotate_horizontal(splot)
```



In [25]:
```
print("Average weekly_sales on a Holiday
weekend is : {:.2f}". format(np.mean
print("Average weekly_sales on a non Holiday
weekend is : {:.2f}". format(np.
```

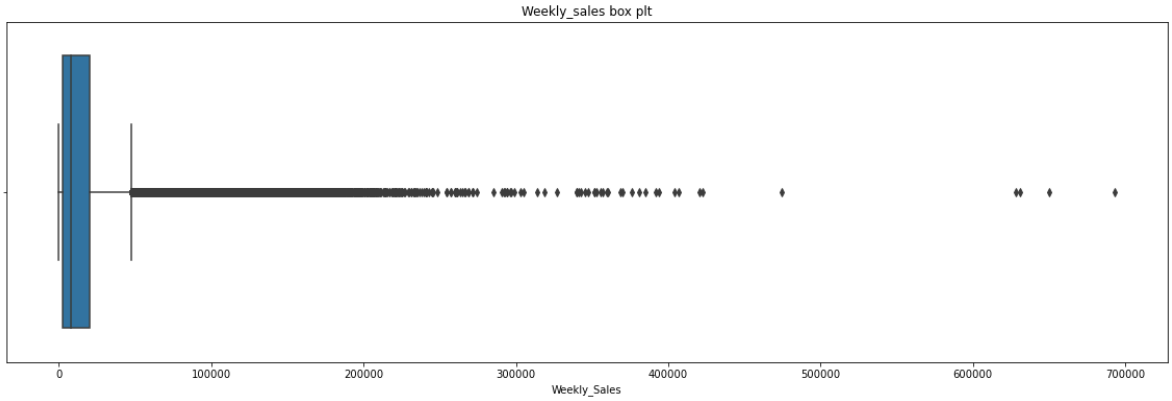
```
Average weekly_sales on a Holiday weekend is : 17092.57
Average weekly_sales on a non Holiday weekend is : 15949.96
```

Observation:

1. Sales in Holiday weekes is ~8% higher than non Holiday weeks.

In [26]:
```
plt.figure(figsize=(20,6))
plt.title("Weekly_sales box plt")
sns.boxplot(data['Weekly_Sales'])
```

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x2a1852bbac0>



Observation:

1. Most of the sales are in the range of 10k-40k. The highest being >600k which might be during the special days and Holidays.

In [27]:
```
data.loc[data['Weekly_Sales']>600000]
```

Out[27]:

	Store	Dept	Date	Weekly_S
90645	10	11-26		
94393	10			
333594	35	72	2010-	693099.36 True
337053	35	72	2011-	630999.19 True
		72	2010-	627962.93 True
		72	2011-	649770.18 True
		11-25		
		11-26		
		11-25		

Observation:

1. These are the 4 days data where the sales are higher than 600k. And these corresponds to holiday in different stores in the dept 72.

In [28]:
```
for i in range(90,100):
print(np.percentile(data['Weekly_Sales'],i))
```

```
42920.59000000003
45702.964
48823.9676
52379.9132
56547.303999999946
61268.571999999986
67427.3972
74962.70959999999
85455.56520000004
106565.43159999997
```

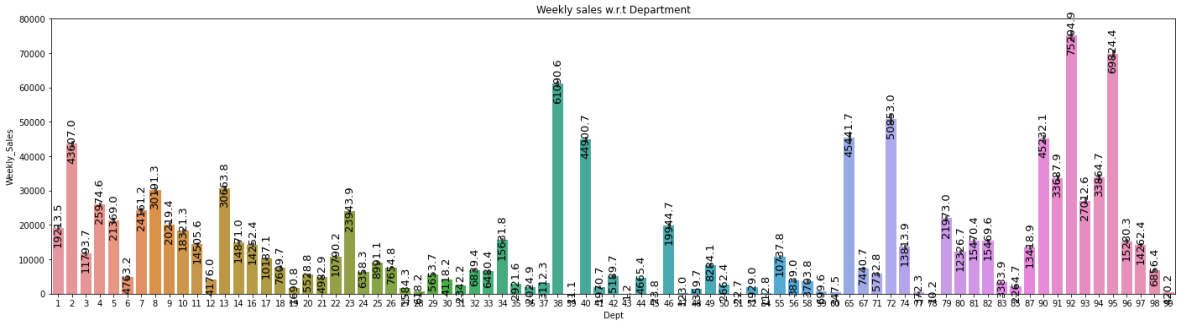
In [29]:
```
for i in np.arange(99,100,0.1):
print(np.percentile(data['Weekly_Sales'],i))
```

```
106565.43159999997
109770.5484400001
113162.88727999979
117500.68107999982
123404.36103999987
130491.7499999996
138614.76951999686
147641.33859999495
157771.01423999044
174912.91659999263
```

99% of the sales are within the range of 170k. And very few days has crossed the range of 200k being the special holidays.

Weekly Sales w.r.t Department

In [30]:
```
plt.figure(figsize=(24,6))
plt.title("Weekly sales w.r.t Department")
splot=sns.barplot(x='Dept', y='Weekly_Sales', data=data)
annotate_vertical(splot)
```



In [31]:
```
data['Weekly_Sales'].loc[data['Dept']==43].sum()
```

Out[31]: 14.32

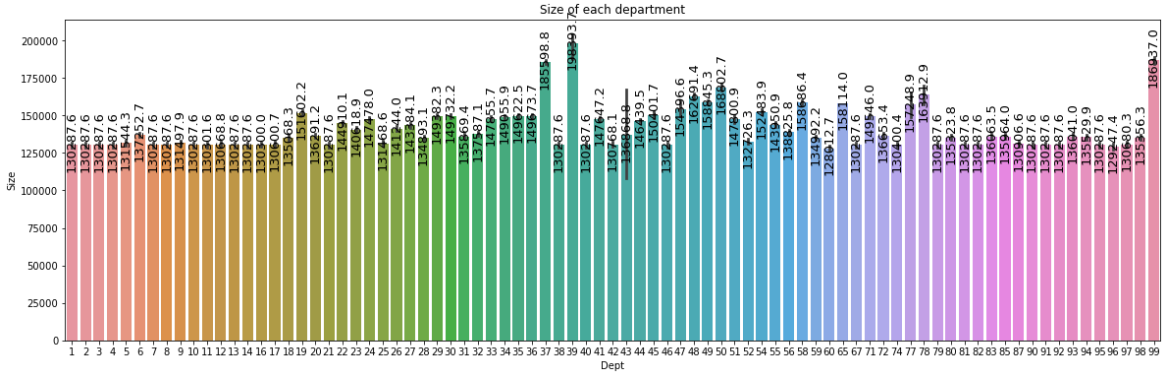
In [32]:
```
data['Weekly_Sales'].loc[data['Dept']==92].sum()
```

Out[32]: 483943341.87

Observation:

1. Dept 92 has the highest recorded weekly sales with 483 Million $ followed by 95 and 38 departments.
2. Few of the departments has very low weekly sales records like 43, 47 etc.

In [33]:
```
plt.figure(figsize=(20,6))
plt.title("Size of each department")
splot=sns.barplot(x='Dept', y='Size', data=data)
annotate_vertical(splot)
```

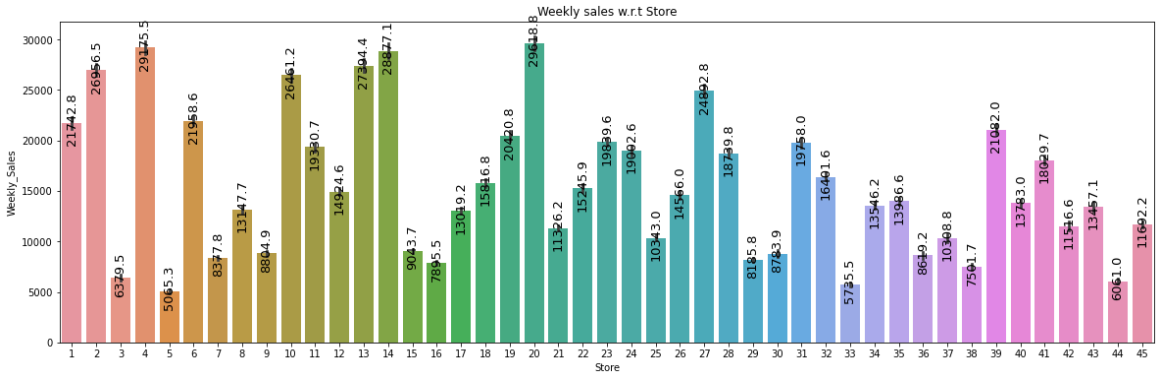


Observation:

1. Department size is almost equal for all, though the dept 92,95,38 has highest sales.
2. This may be because of the cost of items in these depts.
3. We can conclude that dept size is not proportional to the weekly sales.

Location of Stores and Weekly Sales

In [34]:
```
plt.figure(figsize=(20,6))
plt.title("Weekly sales w.r.t Store")
splot=sns.barplot(x='Store', y='Weekly_Sales', data=data)
annotate_vertical(splot)
```

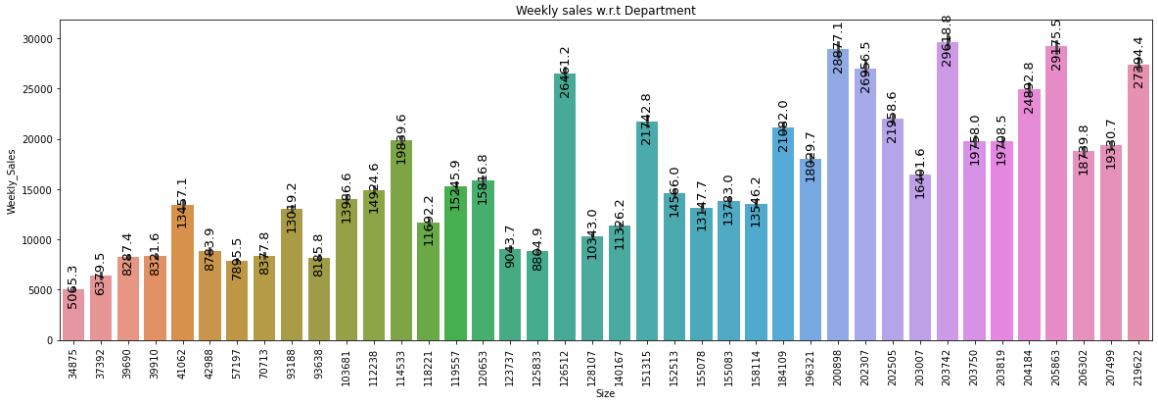


Observation:

1. In particular store 20,14,13,4,2 has the highest weekly sales and this may be due to the location of the stores.

Size of Stores and Weekly Sales

In [35]:
```
plt.figure(figsize=(20,6))
plt.title("Weekly sales w.r.t Department")
plt.xticks(rotation='vertical')
splot=sns.barplot(x='Size', y='Weekly_Sales', data=data)
annotate_vertical(splot)
```



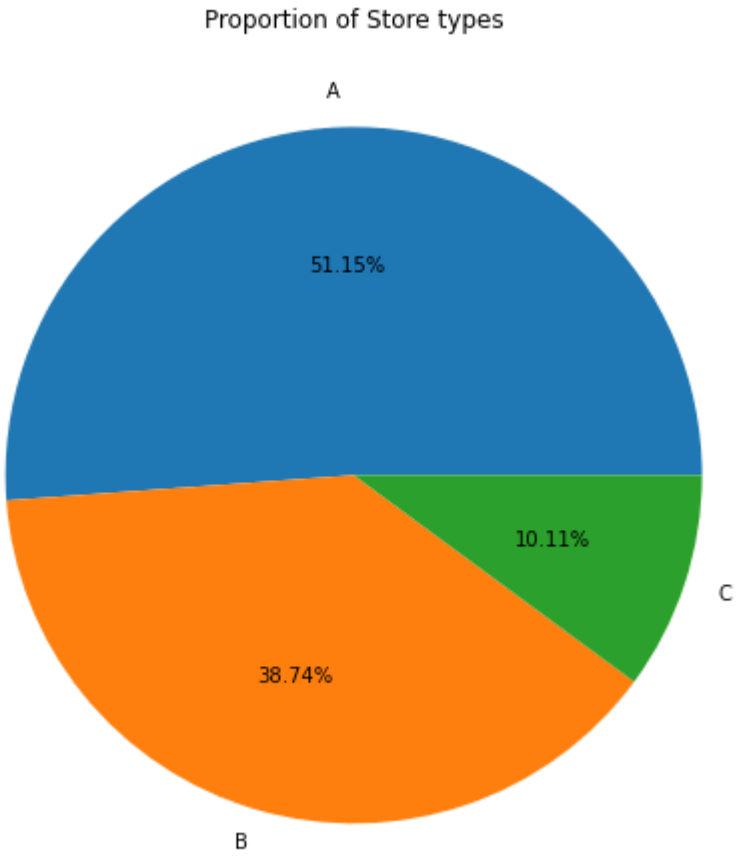
Observation:

1. Weekly sales are higher for the stores which are larger in size to some extent but cannot be concluded exactly as certain stores with larger size has also recorded a low weekly sale.

In [36]:
```
data['Type'].unique()
```

Out[36]: array(['A', 'B', 'C'], dtype=object)

In [37]:
```
sums = data['Type'].value_counts()
plt.figure(figsize=(16,8))
#ax.axis('equal')
plt.title("Proportion of Store types")
plt.pie(sums, labels=sums.index,autopct='%1.2f%%');
plt.show()
```

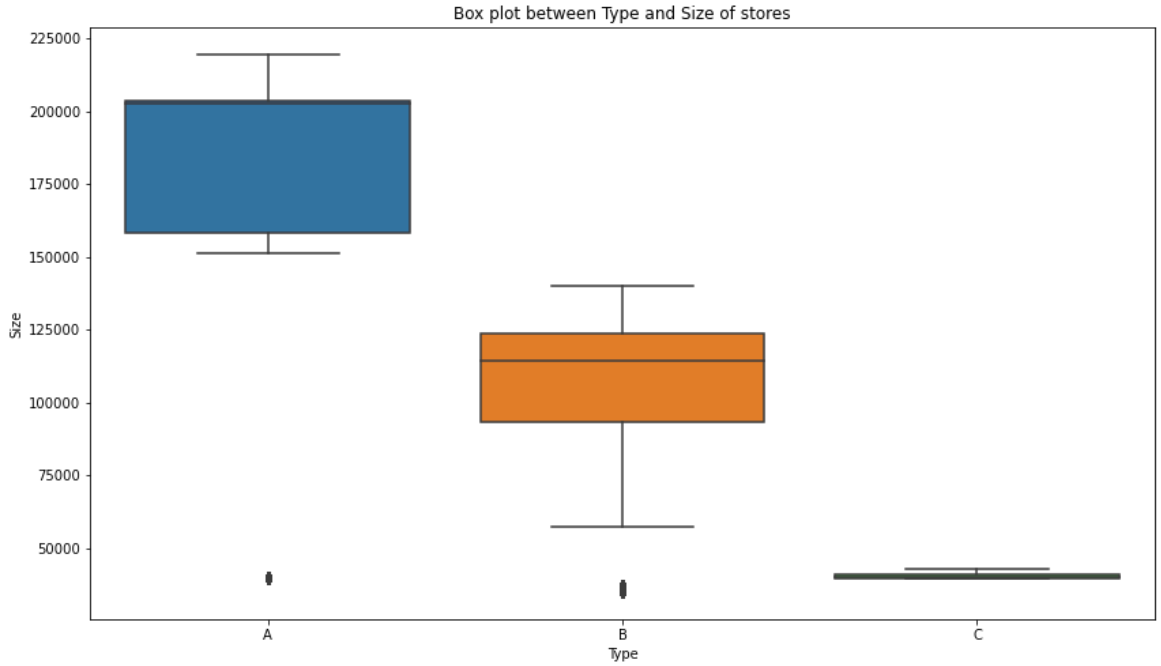


Observation:

1. Stores are paritioned into 3 types. Type A, B, C.

2. Type A stores are larger in proportionate than Type B and C occupying 51%
3. Type B has the second largerst proportionate occupying 38% of the data.

In [38]:
```
plt.figure(figsize=(14,8))
plt.title("Box plot between Type and Size of stores")
sns.boxplot(x='Type', y='Size',data=data)
plt.show()
```

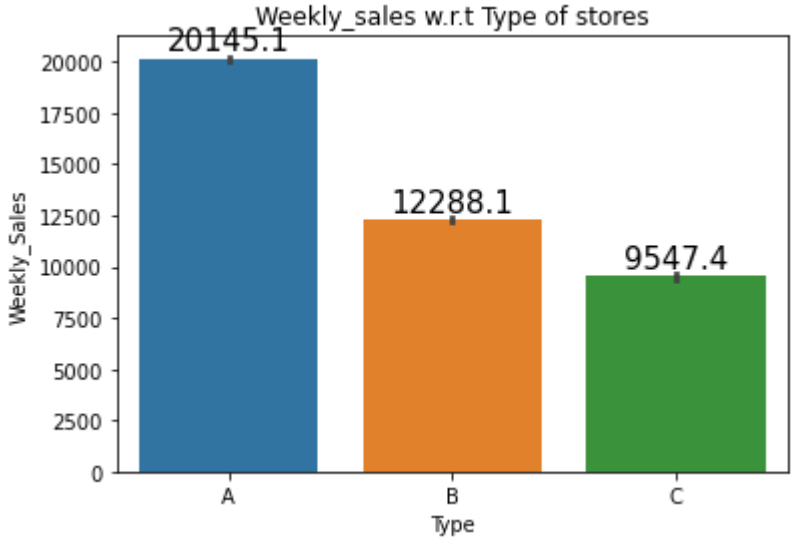


Observations:

Type A store is the largest store and C being the smallest. There is considerable separation among the store types, hence store type is best predictor for store size.

Weekly Sales w.t.t Type of Stores

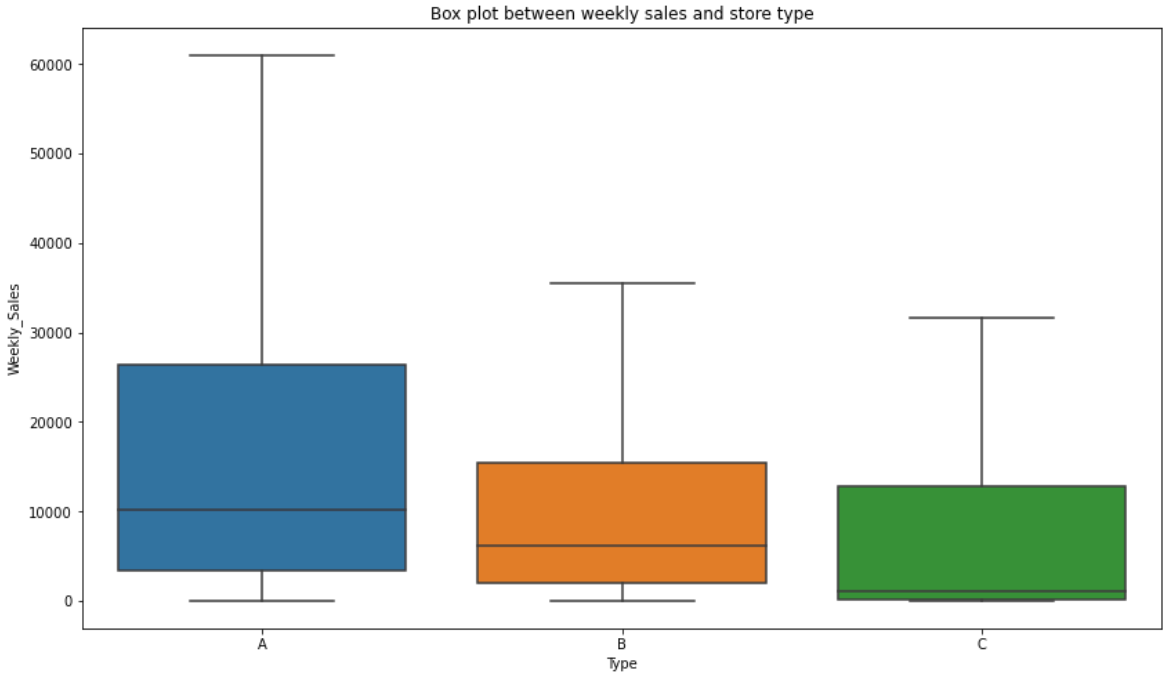
In [39]:
```
plt.title("Weekly_sales w.r.t Type of stores")
splot=sns.barplot(x='Type', y='Weekly_Sales', data=data)
annotate_horizontal(splot)
```



Observation:

1. Weekly sales are directly proportion to the count of store types.
2. Type A stores are higher and the weekly sales are also higher in these stores.
3. Type C stores are lesser in count and the weeklys sales in these stores are also not high.
4. Size of Store types A, B, C is not overlapping. We can distinguish the stores with basic if-else.
5. All the store size of > 150k can be classified as Store A and if size > 62 k and < 140k can be classified as Store B and rest as Store type C.

In [40]:
```
plt.figure(figsize=(14,8))
plt.title("Box plot between weekly sales and store type")
sns.boxplot(x='Type', y='Weekly_Sales',data=data,showfliers=False)
plt.show()
```

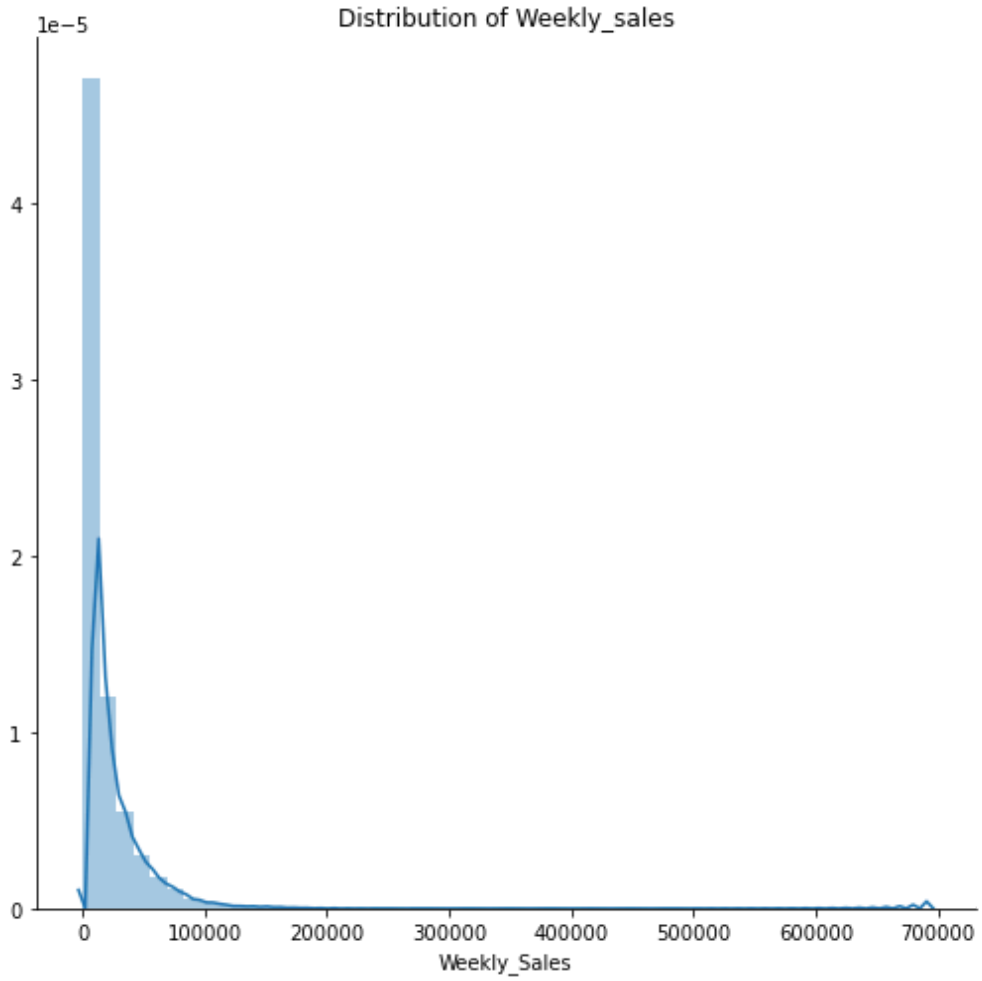


Observation:

1. Store type A has median greater than any other store types.

Distribution of Weekly Sales

In [41]:
```
sns.FacetGrid(data, height=7).map(sns.distplot, "Weekly_Sales").add_legend();
plt.title("Distribution of Weekly_sales")
plt.show()
```



Observation:

1. 85% of the sales happened at around 10k. And is more likely representing the powerlaw distribution, as most of 80% of the sales are in the first 20% of the distribution.

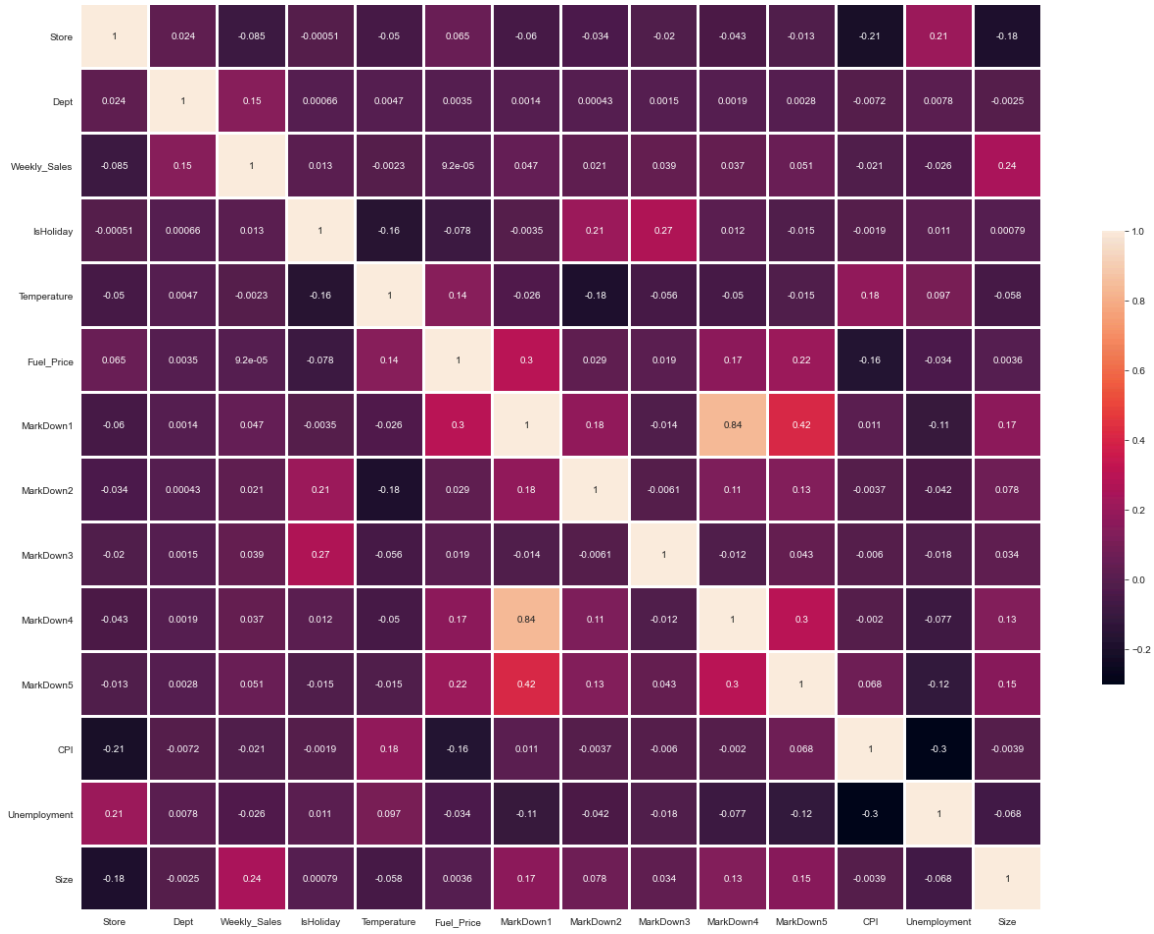
Correlation Matrix

In [42]:
```
#Checking for correlation in the data
corr = data.corr()
corr
```

Out[42]:

	Store	Dept	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkD
Store	1.000000	0.024236	-0.085099	-0.000511	-0.050226	0.065296	-0.0
Dept	0.024236	1.000000	0.148704	0.000662	0.004718	0.003543	0.0
Weekly_Sales	-0.085099	0.148704	1.000000	0.012856	-0.002333	0.000092	0.0
IsHoliday	-0.000511	0.000662	0.012856	1.000000	-0.155779	-0.078140	-0.0
Temperature	-0.050226	0.004718	-0.002333	-0.155779	1.000000	0.143718	-0.0
Fuel_Price	0.065296	0.003543	0.000092	-0.078140	0.143718	1.000000	0.2
MarkDown1	-0.059948	0.001444	0.047260	-0.003522	-0.026426	0.297084	1.0
MarkDown2	-0.033727	0.000428	0.020961	0.207314	-0.179669	0.029288	0.1
MarkDown3	-0.020299	0.001514	0.038530	0.266718	-0.056010	0.018645	-0.0
MarkDown4	-0.042792	0.001872	0.037481	0.011621	-0.050310	0.166654	0.8
MarkDown5	-0.012595	0.002754	0.050612	-0.015189	-0.014836	0.215597	0.4
CPI	-0.211240	-0.007172	-0.021153	-0.001946	0.182191	-0.164207	0.0
Unemployment	0.208753	0.007785	-0.025831	0.010540	0.096792	-0.033897	-0.1
Size	-0.182784	-0.002471	0.244089	0.000786	-0.058414	0.003620	0.1

In [43]:
```
sns.set_style('white')
corr = data.corr()
f,ax = plt.subplots(figsize=(22,17))
fig = sns.heatmap(corr, annot=True,linewidths= 2, cbar_kws={"shrink": .5})
```



Observation:

1. Though there is less correlation it is obvious that the unemployment index increase leads to the less shopping, and this goes with the Fuel_price as well.

2. Increase and decrease in temperature affects sales. Lower temperature and high temperature can decrease weekly sales.

In [44]:
```
# FEATURE ENGINEERING for the date column

#Extracting the info from the Date column as the date directly can't be
passe

data['Date']=pd.to_datetime(data.Date) #Converting the string type to pandas
#Extracting year,month, day, day of the year, week of the year, is month star
data['Year']=pd.DatetimeIndex(data['Date']).year
data['Month']=pd.DatetimeIndex(data['Date']).month
data['Day']=pd.DatetimeIndex(data['Date']).day
data['Day_of_year']=pd.DatetimeIndex(data['Date']).dayofyear
data['Week_of_year']=pd.DatetimeIndex(data['Date']).weekofyear

data['Month_start']=pd.DatetimeIndex(data['Date']).is_month_start
data['Month_end']=pd.DatetimeIndex(data['Date']).is_month_end
```

In [45]:
```
# #Extracting the info from the Date column
as the date directly can't be pas

#     test['Date']=pd.to_datetime(test.Date)
#Converting the string type to pand #
#Extracting year,month, day, day of the
year, week of the year, is month st #
test['Year']=pd.DatetimeIndex(test['Date']).
year
#
test['Month']=pd.Dat
etimeIndex(test['Dat
e']).month #
test['Day']=pd.Datet
imeIndex(test['Date'
]).day
#
test['Day_of_year']=pd.DatetimeIndex(test['Da
te']).dayofyear
#
test['Week_of_year']=pd.DatetimeIndex(test['D
ate']).weekofyear

#
test['Month_start']=pd.DatetimeInde
x(test['Date']).is_month_start #
test['Month_end']=pd.DatetimeIndex(
test['Date']).is_month_end
```

In [46]:
```
data
```

Out[46]:

	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	M
	24924.50	False	42.31	2.572	0.00	
	50605.27	False	42.31	2.572	0.00	
	13740.12	False	42.31	2.572	0.00	
	39954.04	False	42.31	2.572	0.00	
	32229.38	False	42.31	2.572	0.00	
	...		..	..	...	
	2487.80	...	.	.	4018.91	
			58.85	3.882		
		False				
	5203.31	False	58.85	3.882	4018.91	
	56017.47	False	58.85	3.882	4018.91	
	6817.48	False	58.85	3.882	4018.91	
	1076.80	False	58.85	3.882	4018.91	

	Store	Dept	Date
0	1	1	2010-02-05
1	1	2	2010-02-05
2	1	3	2010-02-05
3	1	4	2010-02-05
4	1	5	2010-02-05
...	...	...	...
421565	45	93	2012-10-26
421566	45	94	2012-10-26
421567	45	95	2012-10-26
421568	45	97	2012-10-26
421569	45	98	2012-10-26

420285 rows × 23 columns

In [47]:
```
#test
```

In [48]:
```
import math
def add_week_of_month(df):
df['week_in_month'] = pd.to_numeric(df.Day/7)
df['week_in_month'] = df['week_in_month'].apply(lambda x: math.ceil(x))
    return df
```

In [49]:
```
#add_week_of_month(test)
```

In [50]:
```
add_week_of_month(data)
```

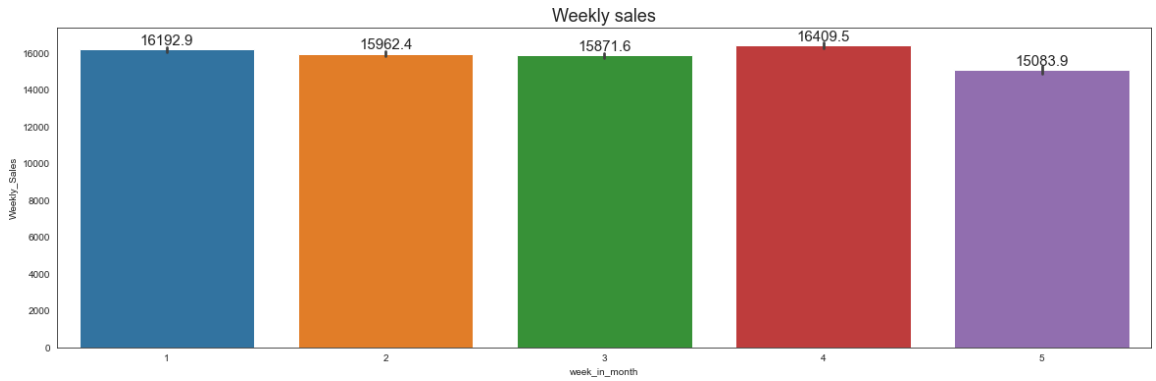
Out[50]:

	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	M
	24924.50	False	42.31	2.572	0.00	
	50605.27	False	42.31	2.572	0.00	
	13740.12	False	42.31	2.572	0.00	
	39954.04	False	42.31	2.572	0.00	
	32229.38	False	42.31	2.572	0.00	
	...		..	..	...	
	2487.80	...	.	.	4018.91	
			58.85	3.882		
		False				
	5203.31	False	58.85	3.882	4018.91	
	56017.47	False	58.85	3.882	4018.91	
	6817.48	False	58.85	3.882	4018.91	
	1076.80	False	58.85	3.882	4018.91	

	Store	Dept	Date
0	1	1	2010-02-05
1	1	2	2010-02-05
2	1	3	2010-02-05
3	1	4	2010-02-05
4	1	5	2010-02-05
...	...	...	...
421565	45	93	2012-10-26
421566	45	94	2012-10-26
421567	45	95	2012-10-26
421568	45	97	2012-10-26
421569	45	98	2012-10-26

420285 rows × 24 columns

In [51]:
```
plt.figure(figsize=(20,6))
plt.title("Weekly sales",fontsize=18)
splot=sns.barplot(x='week_in_month', y='Weekly_Sales', data=data)
annotate_horizontal(splot)
```



In [52]:

```
week_1_4=np.mean(data['Weekly_Sales'][(data['

week_in_month']==1 ) | (data['w

week_2_3_5=np.mean(data['Weekly_Sales'][(data

['week_in_month']==2 ) | (data[

if week_1_4 > week_2_3_5:
    print("There is an inrease in sales of
    {:.2f}% in week 1,4 than other wee
else:
    print("There is an decrease in sales of
    {:.2f}% in week 1,4 than other we
```

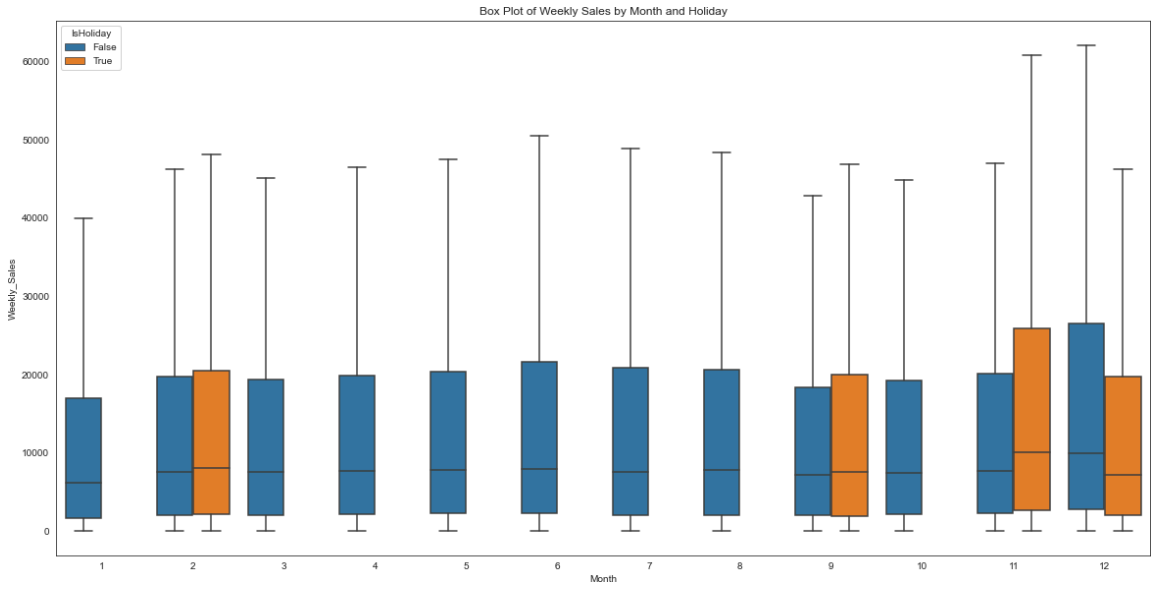
```
There is an inrease in sales of 2.41% in week 1,4 than other weeks
```

Observation:

Weekly sales on average is High in 4th week but not significantly high. That is end of the month the weekly sales are higher than usual sales from the beginning of the month.

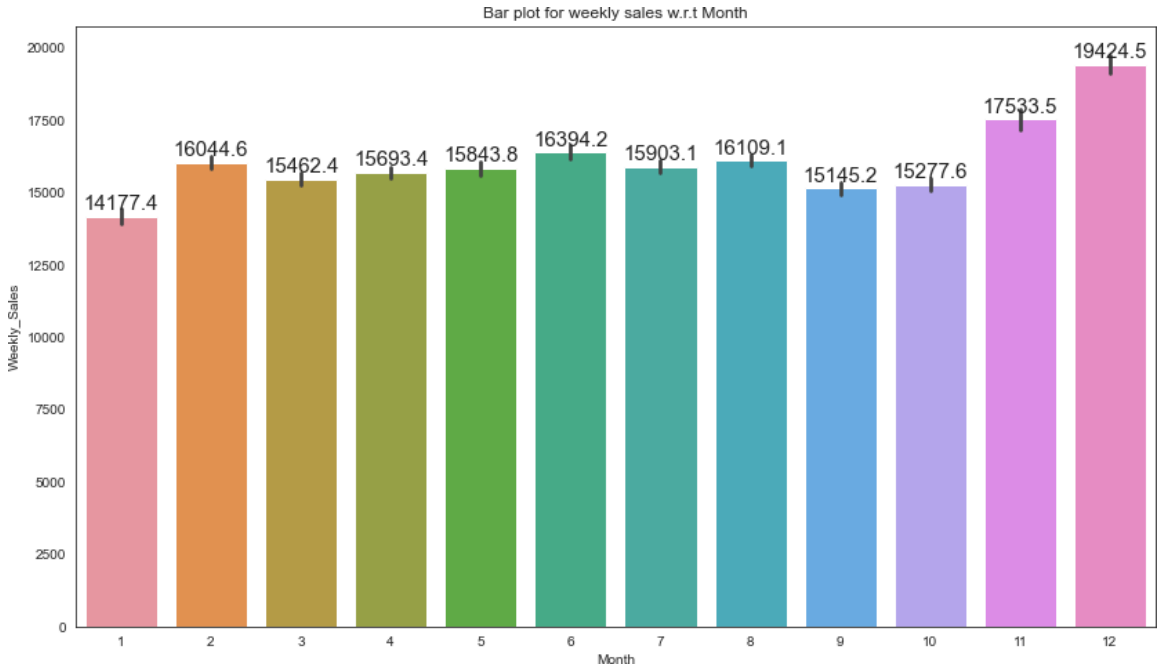
In [53]:

```
data_14 = pd.concat([data['Month'],
data['Weekly_Sales'], data['IsHoliday']],
plt.figure(figsize=(20,10))
plt.title('Box Plot of Weekly Sales by Month
and Holiday')
fig = sns.boxplot(x='Month',
y='Weekly_Sales', data=data_14,
showfliers=False
```



In [54]:
```
plt.figure(figsize=(14,8))
splot=sns.barplot(data['Month'], data['Weekly_Sales'])
annotate_horizontal(splot)
plt.title("Bar plot for weekly sales w.r.t Month")
```

Out[54]: Text(0.5, 1.0, 'Bar plot for weekly sales w.r.t Month')




```
monthly_1_10=np.mean(data['Weekly_Sales'][(data['Month']!=12) | (data['Month
monthly_11_12=np.mean(data['Weekly_Sales'][(data['Month']==12) | (data['Mont

print("There is an inrease in sales of {:.2f}% in month 11,12 than other mon
```

In [55]:

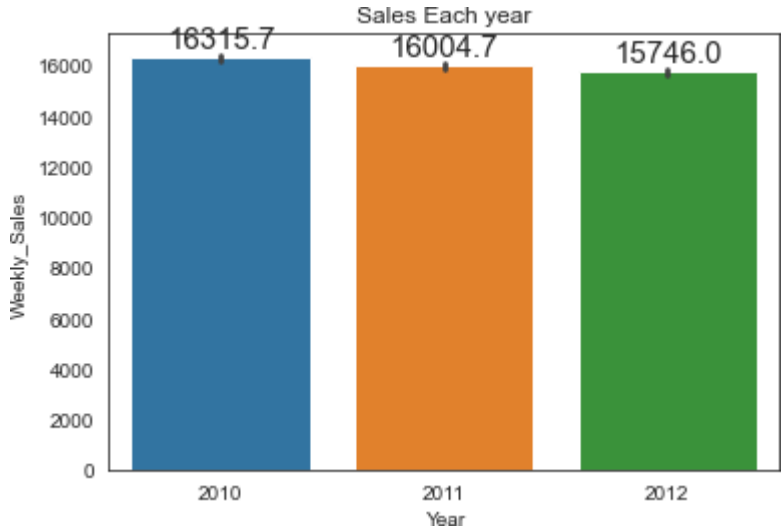
There is an inrease in sales of 15.96% in month 11,12 than other months

Observation:

1. Dec, Nov months has slight higher sales. This may be because of the holiday season (like Thanksgiving, Christmas).

Sales Each Year

In [56]:
```
plt.title("Sales Each year")
splot=sns.barplot(data['Year'], data['Weekly_Sales'])
annotate_horizontal(splot)
```

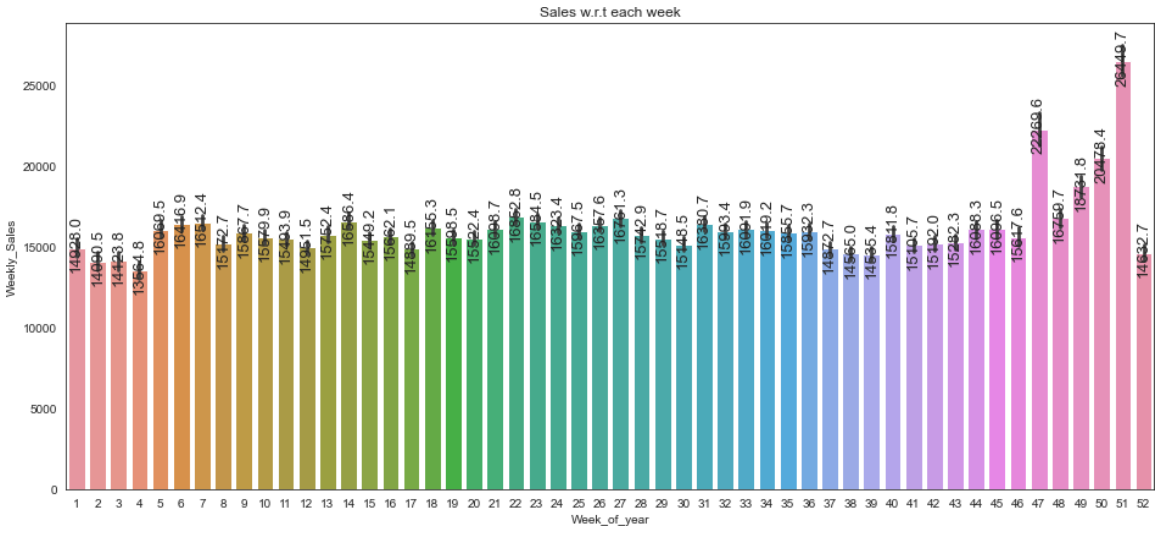


Observation:

1. There is no difference in the average of sales throughout the given 3 years of data. Sales remain almost similar.

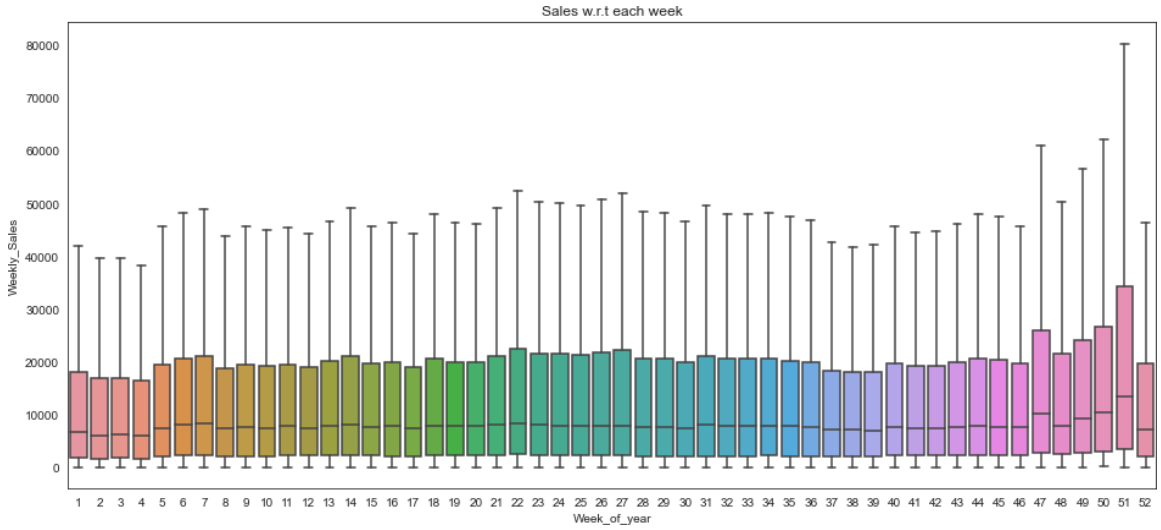
Sales w.r.t each week

In [57]:
```
plt.figure(figsize=(16,7))
plt.title("Sales w.r.t each week")
splot=sns.barplot(x='Week_of_year', y='Weekly_Sales',data=data)
annotate_vertical(splot)
```



In [58]:
```
plt.figure(figsize=(16,7))
plt.title("Sales w.r.t each week")
sns.boxplot(x='Week_of_year', y='Weekly_Sales',data=data,showfliers=False)
```

Out[58]: <matplotlib.axes._subplots.AxesSubplot at 0x2a186154df0>



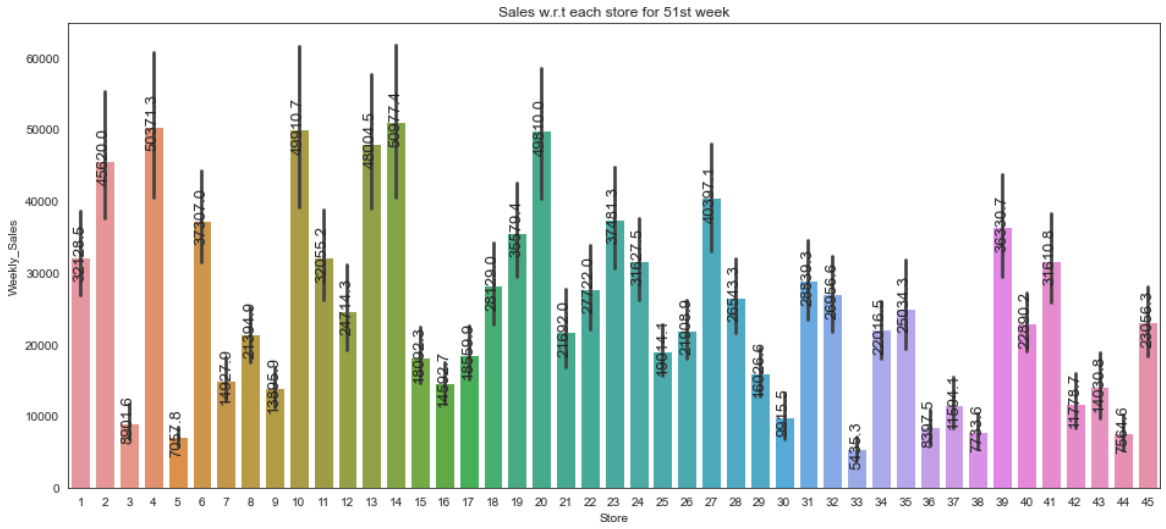
Observation:

Sales in the 51st,47th week is significantly higher, and this is because of the holidays (like Christmas days, Thanksgiving days respectively).

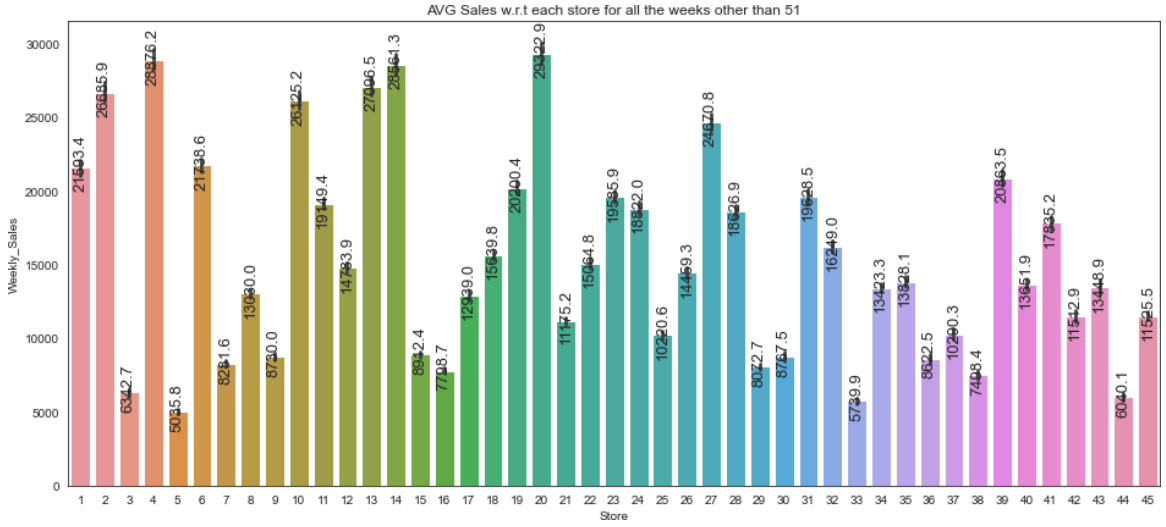
In [59]:
```
data.columns
```

Out[59]:
```
Index(['Store', 'Dept', 'Date', 'Weekly_Sales', 'IsHoliday', 'Temperature',
       'Fuel_Price', 'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4',
       'MarkDown5', 'CPI', 'Unemployment', 'Type', 'Size', 'Year', 'Month',
       'Day', 'Day_of_year', 'Week_of_year', 'Month_start', 'Month_end',
       'week_in_month'],
      dtype='object')
```

In [60]:
```
plt.figure(figsize=(16,7))
plt.title("Sales w.r.t each store for 51st week")
splot=sns.barplot(x=data['Store'],y=data['Weekly_Sales'][data['Week_of_year']
annotate_vertical(splot)
```



In [61]:
```
plt.figure(figsize=(16,7))
plt.title("AVG Sales w.r.t each store for all the weeks other than 51")
splot=sns.barplot(x=data['Store'],y=data['Weekly_Sales'][data['Week_of_year']
annotate_vertical(splot)
```



In [62]:
```
print("Avg sales of all the stores in 51st
week is :",(np.mean(data['Weekly_S print("Avg
weekly sales of all the stores except 51st
week is :", (np.mean(da
```

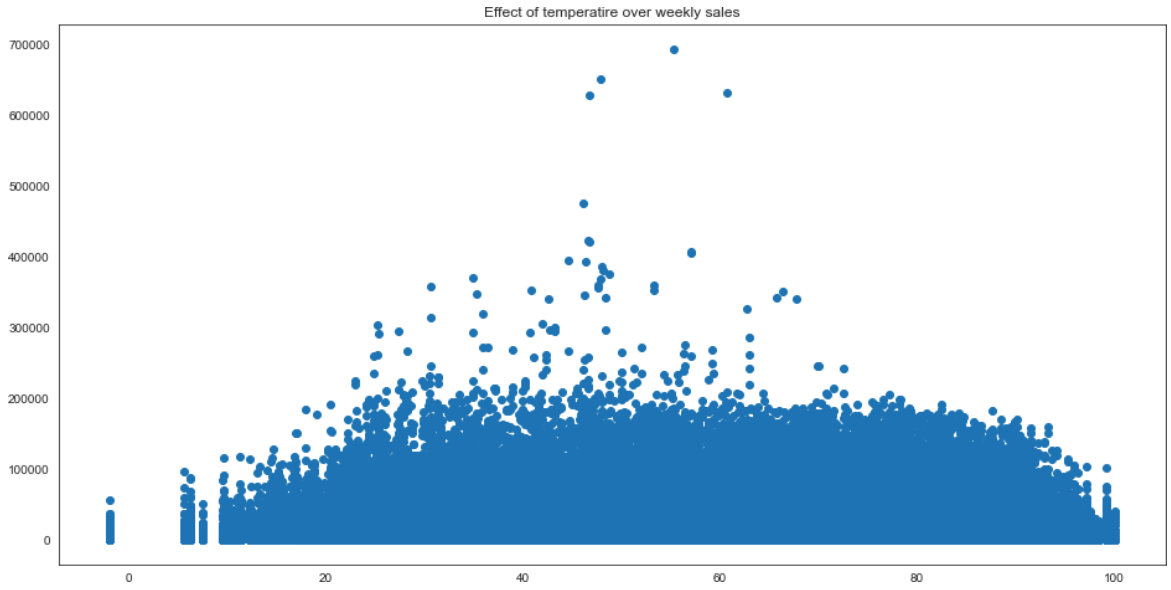
```
Avg sales of all the stores in 51st week is : 26449.733674426392
Avg weekly sales of all the stores except 51st week is : 15880.167672103773
```

Observation :

1. Weekly sales in all the stores in the week of 51 hiked up due to the christmas season.
2. Avg sales in all the stores in the remaining weeks being 158k , the 51st week recorded 263k i.e: An increase of ~60% can be seen.

Temperature & Weekly Sales

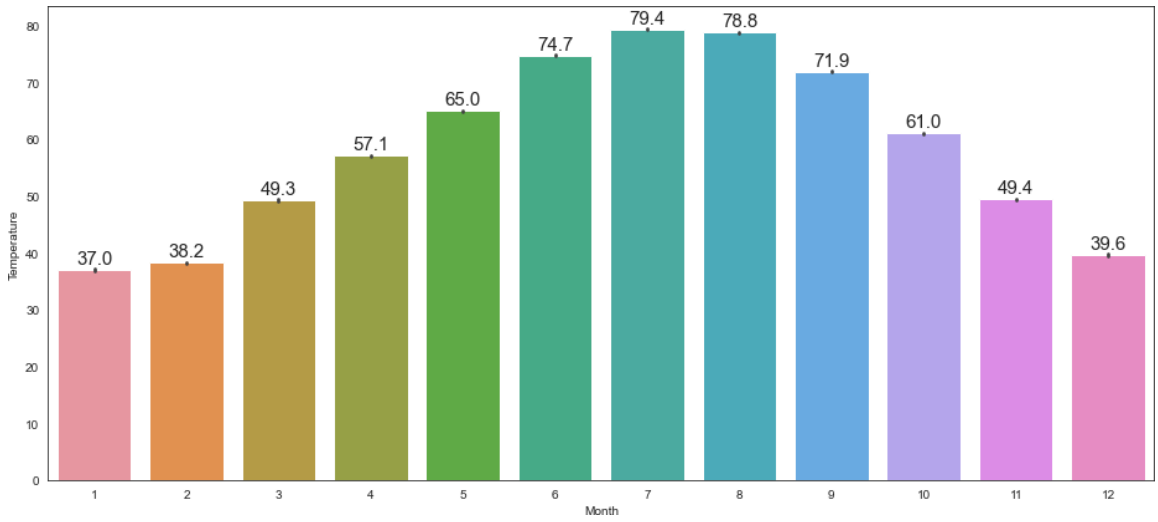
In [63]:
```
plt.figure(figsize=(16,8))
plt.scatter(data['Temperature'], data['Weekly_Sales'])
plt.title("Effect of temperatire over weekly sales")
plt.show()
```



Observation:

1. Average temperature ranges in 40-65 and the sales are higher in the mean temperatures.
2. Sales is affected in lower temperatures and > 90 degrees temp.

In [64]:
```
plt.figure(figsize=(16,7))

splot=sns.barplot(x='Month', y='Temperature',data=data)
annotate_horizontal(splot)
```



In [65]:

```
for i in data['Month'].unique():


    if (np.mean(data['Weekly_Sales'][data['Mon
        increase=(np.mean(data['Weekly_Sales']
        d

        print("There is an weekly_sales increa
    elif(np.mean(data['Weekly_Sales'][data['Mo
        decrease=(np.mean(data['Weekly_Sales']
        t  print("There  is  an  weekly_sales  d
        {},{}
```

```
There is an weekly_sales decrease of 3.77% in betw
There is an weekly_sales increase of 1.49% in betw
There is an weekly_sales increase of 0.96% in betw
There is an weekly_sales increase of 3.47% in betw
There is an weekly_sales decrease of 3.09% in betw
There is an weekly_sales increase of 1.30% in betw
There is an weekly_sales decrease of 6.36% in betw
There is an weekly_sales increase of 0.87% in
between 9,10 months
There is an weekly_sales increase
of 14.77% in between 10,11 months
There is an weekly_sales increase
of 10.79% in between 11,12 months
There is an weekly_sales increase
of 13.17% in between 1,2 months
```

In [66]:

```
data[['Month','IsHoliday']][data['IsHoliday']
==True].groupby(by='Month').coun
```

Out[66]:

Month	IsHoliday
2	8874
9	8833
11	5946
12	5910

Observation:

1. The highest decrease of sales is in between 8th to 9th month that is because of the the temperature fall.
2. As the temperature increases the sales are increasing in most cases, but inversely the sales are increasing in the months of 11,12 th month inspite of the temperature decreasing because of the Holidays.

In [67]:

```
store_sales_temp=[]

for i in range(1,46):

    if (np.mean(data['Weekly_Sales'][(data['Mo
        per_increase=(np.mean(data['Weekly_Sal
        (data  print("There  is  an  increase  o
        store {}".

        store_sales_temp.append(per_increase*1


    else:
        per_decrease=(np.mean(data['Weekly_Sal
        (data print("There is a decrease of {:
        store {}".f

        store_sales_temp.append((per_decrease-
```

```
There is a decrease of 4.26% in the 7th month in s
There is a decrease of 3.92% in the 7th month in s
There is a decrease of 5.72% in the 7th month in s
There is a decrease of 4.39% in the 7th month in s
There is a decrease of 3.54% in the 7th month in s
There is an increase of
4.87% in the 7th month in
store 6 There is an
increase of 19.29% in the
7th month in store 7
There is a decrease of 6.29% in the 7th month in s
There is a decrease of 6.64% in the 7th month in s
There is a decrease of 5.36% in the 7th month in s
There is a decrease of 4.85% in the 7th month in s
There is a decrease of 6.99% in the 7th month in s
There is a decrease of 0.51% in the 7th month in s
There is a decrease of 4.50% in the 7th month in s
There is an increase of 1.56%
in the 7th month in store 15
There is an increase of 16.19%
in the 7th month in store 16
There is an increase of 2.97%
in the 7th month in store 17
There is a decrease of 2.47% in the 7th month in s
There is a decrease of 1.19% in the 7th month in s
There is a decrease of 2.69% in the 7th month in s
There is a decrease of 3.15% in the 7th month in s
There is a decrease of 1.22% in the 7th month in s
There is an increase of
0.91% in the 7th month in
store 23 There is an
increase of 7.39% in the
7th month in store 24
There is a decrease of
0.52% in the 7th month in
store 25 There is an
increase of 7.09% in the
7th month in store 26
There is an increase of
5.00% in the 7th month in
store 27
There is a decrease of 4.83% in the 7th month in s
There is a decrease of 2.22% in the 7th month in s
There is a decrease of 2.53% in the 7th month in s
There is a decrease of 3.76% in the 7th month in s
There is a decrease of 1.15% in the 7th month in s
There is an increase of
0.77% in the 7th month in
store 33 There is a
decrease of 5.64% in the
7th month in store 34
There is an increase of
5.46% in the 7th month in
store 35 There is an
increase of 2.29% in the
7th month in store 36
There is a decrease of
```

```
3.51% in the 7th month in
store 37 There is a
decrease of 2.97% in the
7th month in store 38
```

In [68]:
```
max_store_sales_temp_index=store_sales_temp.index(
min_store_sales_temp_index=store_sales_temp.index(
print("Average weekly sales in 7th month is  hiked
th print("Average weekly sales in 7th month is   d
the
```

```
There is a decrease of
1.93% in the 7th month in
store 39 There is an
increase of 5.85% in the
7th month in store 40
There is an increase of
1.68% in the 7th month in
store 41 There is a
decrease of 3.12% in the
7th month in store 42
There is a decrease of
3.10% in the 7th month in
store 43 There is an
increase of 0.85% in the
7th month in store 44
There is a decrease of
3.88% in the 7th month in
store 45
```

```
Average weekly sales in 7th month is
hiked up by max by 19.29% in the Stor e :
7
Average weekly sales in 7th month is
decreased max by -6.99% in the Store
: 12
```

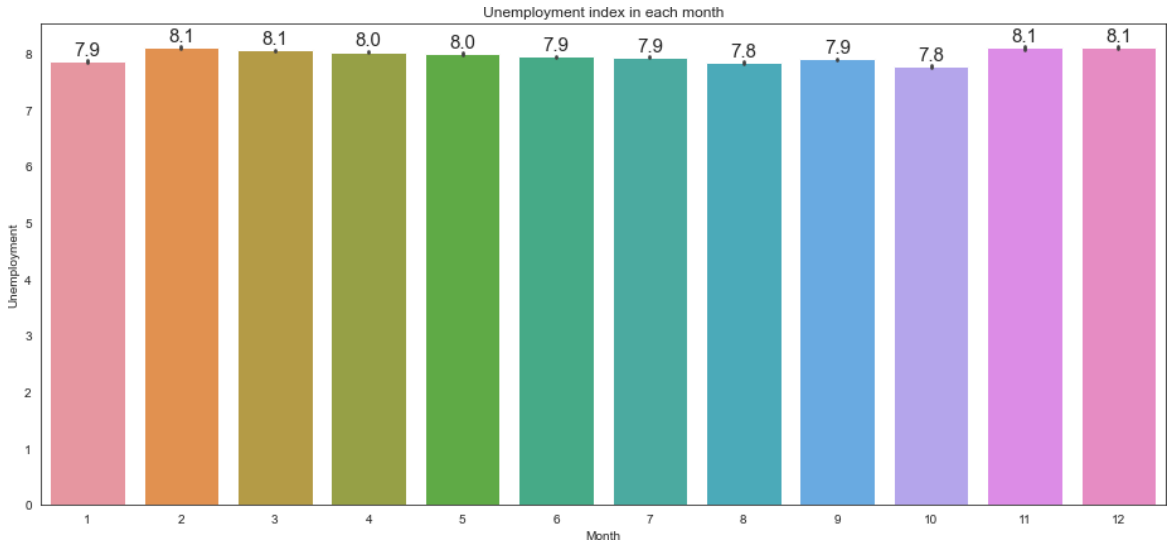
Observation:

1. In the store number 7 we can see that there is an increase of 40% sales on an average compared to all other months being the highest temperature month.

Unemployment and Weekly Sales

In [69]:
```
plt.figure(figsize=(16,8))
plt.scatter(data['Unemployment'], data['Weekly_Sales'])
plt.title("Effect of Unemployment over weekly sales")
plt.show()
```

In [70]:
```
plt.figure(figsize=(16,7))
plt.title("Unemployment index in each month")
splot=sns.barplot(x='Month', y='Unemployment',data=data)
annotate_horizontal(splot)
```



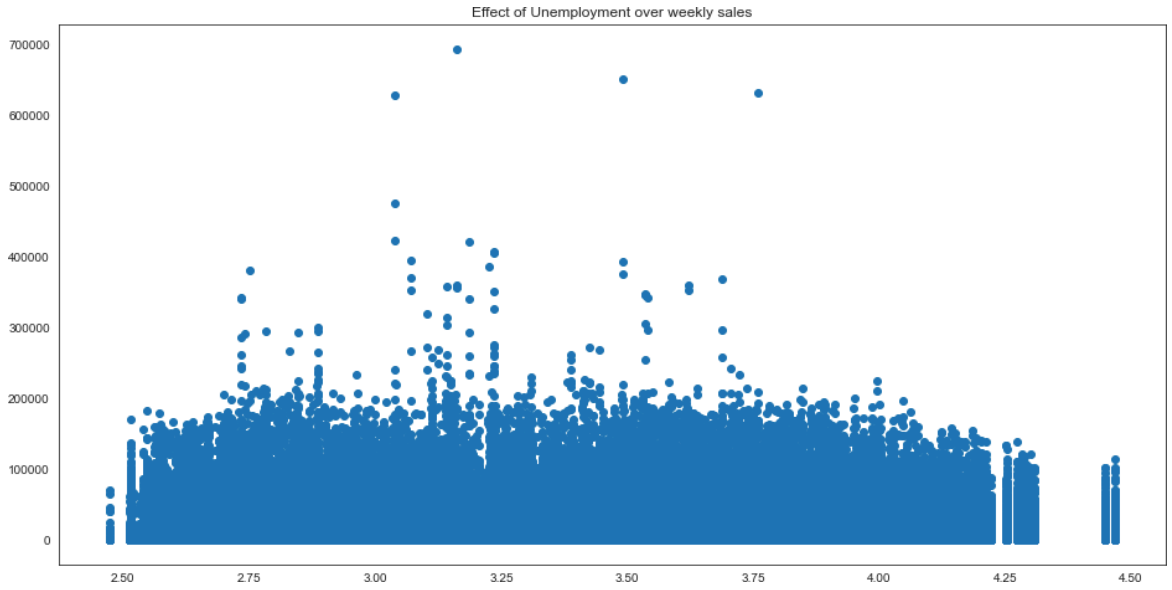
In [71]:
```
weekly_sales_least_unemployment=np.mean(data[
'Weekly_Sales'][data['Unemployme
weekly_sales_highest_unemployment=np.mean(dat
a['Weekly_Sales'][data['Unemploy


if weekly_sales_highest_unemployment >
weekly_sales_least_unemployment:
    print("There is an inrease in sales of
    {:.2f}% inspite of unemployment ri
else:
    print("There is an decrease in sales of
    {:.2f}% inspite of unemployment d
```

```
There is an inrease in sales of 2.23% inspite of unemployment rise from 7.8
to 8.1
```

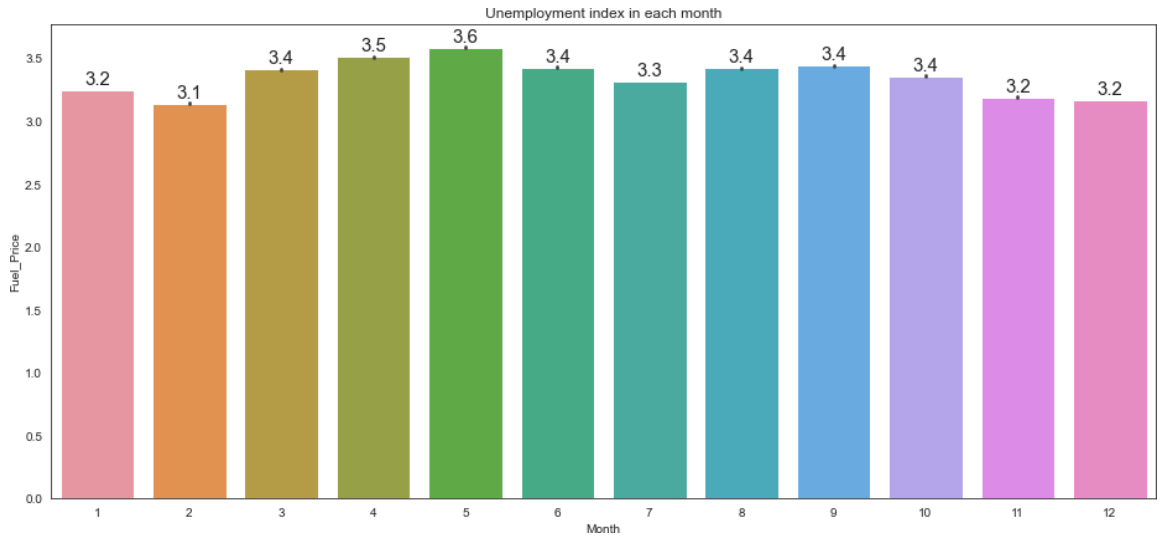
Observation:

1. Unemployment index is mostly in range of 5-9.
2. Average sales of 200k is happening all the time without the factor of unemployment index.
3. Unemployment is almost similar in all the months
4. Hence unemployment is not much affecting the weekly_sales.
5. Weekly Sales are not affected by unemployment, though there is rise in unemployment there is no significatn decrease and instead there is an increase in sales of 2.22% irrespective to the unemployment index.

In [72]:
```
plt.figure(figsize=(16,8))
plt.scatter(data['Fuel_Price'], data['Weekly_Sales'])
plt.title("Effect of Unemployment over weekly sales")
plt.show()
```



In [73]:
```
plt.figure(figsize=(16,7))
plt.title("Unemployment index in each month")
splot=sns.barplot(x='Month', y='Fuel_Price',data=data)
annotate_horizontal(splot)
```



Fuel Price & Weekly Sales

In [74]:
```
print("Avg sales of all the stores when the Fuel price is >3.25 :",(np.mean(d print("Avg sales of all the stores when the Fuel price is <3.25 :", (np.mean(
```

```
Avg sales of all the stores when the Fuel price is >3.25 : 15942.432961935054
Avg sales of all the stores when the Fuel price is <3.25 : 16158.532075484944
```

In [75]:
```
for i in data['Month'].unique():


    if (np.mean(data['Weekly_Sales'][data['Mon
        increase=(np.mean(data['Weekly_Sales']
        d

        print("There is an weekly_sales increa
    elif(np.mean(data['Weekly_Sales'][data['Mo
        decrease=(np.mean(data['Weekly_Sales']
        t  print("There  is  an  weekly_sales  d
        {},{}
```

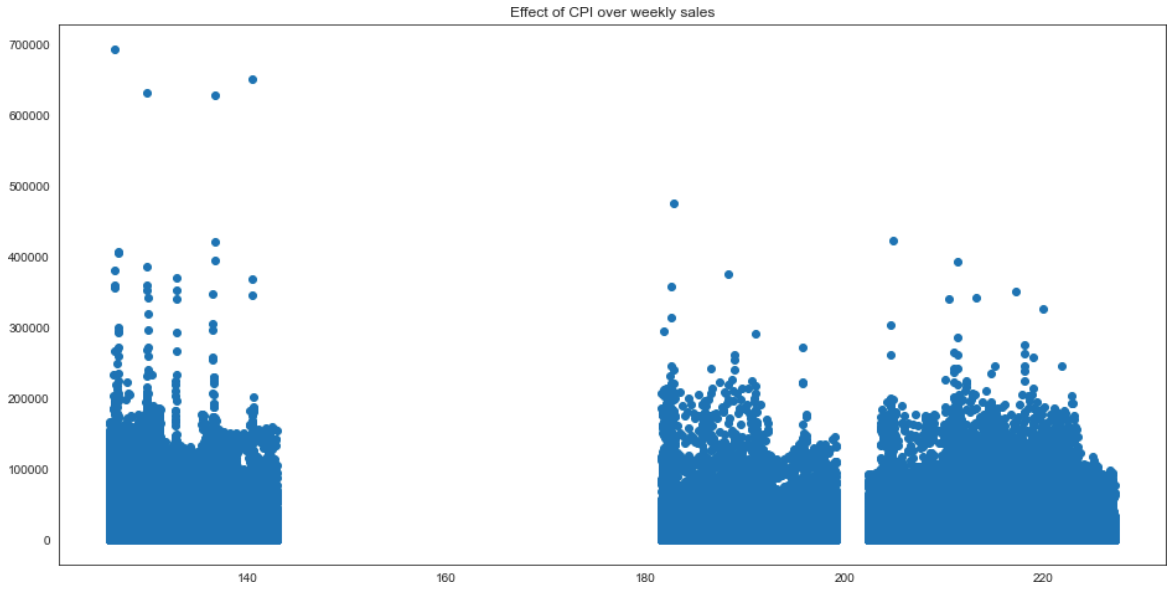
```
There is an weekly_sales decrease of 3.77% in betw
There is an weekly_sales increase of 1.49% in betw
There is an weekly_sales increase of 0.96% in betw
There is an weekly_sales increase of 3.47% in betw
There is an weekly_sales decrease of 3.09% in betw
There is an weekly_sales increase of 1.30% in betw
There is an weekly_sales decrease of 6.36% in betw
There is an weekly_sales increase of 0.87% in
between 9,10 months
There is an weekly_sales increase
of 14.77% in between 10,11 months
There is an weekly_sales increase
of 10.79% in between 11,12 months
There is an weekly_sales increase
of 13.17% in between 1,2 months
```

Observation:

1. Fuel price ranges from 2.6-4.25 and the weekly sales is constant and less than 200k
2. Average sales is high when the Fuel_price is less than 3.25
3. During the month of May the fuel_price is high due to which the sales have gone low.
4. There is a decrease in sales of 3.84% between the 2nd and 3rd month, this may be beacause of the increase in Fuel price and the increase in temperature.

CPI and Weekly Sales

In [76]:
```
plt.figure(figsize=(16,8))
plt.scatter(data['CPI'], data['Weekly_Sales'])
plt.title("Effect of CPI over weekly sales")
plt.show()
```



Observation:

1. There is no continous data for the CPI index. There is no observed data with the CPI from 150-180.
2. Weekly sales are slight higher as the CPI is in range of 210-225.

Sales during the week and month

```
month_start_sales=np.mean(data['Weekly_Sales'][data['Month_start']==True])
month_end_sales= np.mean(data['Weekly_Sales'][data['Month_end']==True])


print("There is an icrease of {:.2f}% sales in month_starting compared to mo
```

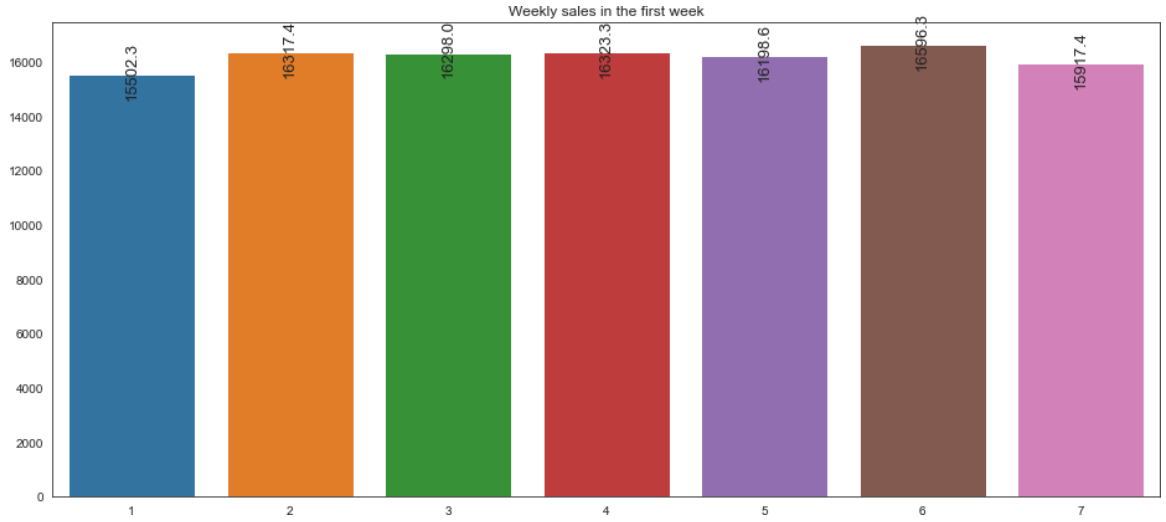
In [77]:

```
There is an icrease of 4.76% sales in month_starting compared to month endi
ng
```

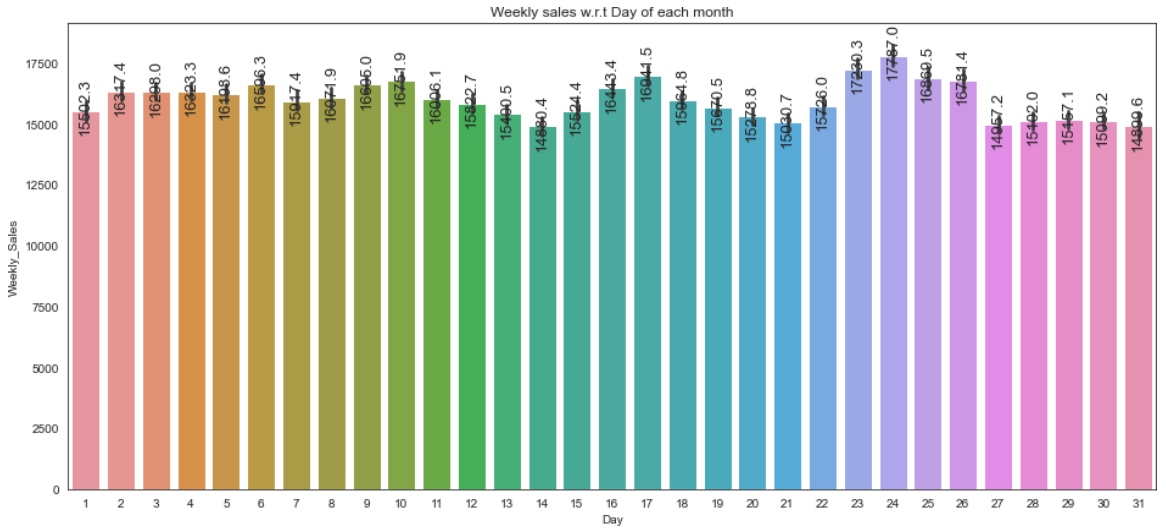
Observation:

1. There is 4.79% increase in sales in the month starting and this may be because of the credit of the salary in the month start.

In [78]:
```
sales=[]
 for i in [1,2,3,4,5,6,7]:
sales.append(np.mean(data['Weekly_Sales'][data['Day']==i]))


plt.figure(figsize=(16,7))
plt.title("Weekly sales in the first week")
splot=sns.barplot(x=[1,2,3,4,5,6,7], y=sales,data=data)
 annotate_vertical(splot)
```



In [79]:
```
plt.figure(figsize=(16,7))
plt.title("Weekly sales w.r.t Day of each month")
splot=sns.barplot(x='Day', y='Weekly_Sales',data=data)
 annotate_vertical(splot)
```



1. Sales are high in the week start of the month end. And again high in the first week of the month.

In [80]:
```
train = data.sort_values(by='Date', ascending=True) # Sorting the data in inc
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) # Tr
```

In [81]:
```
train.describe()
```

Out[81]:

	Store	Dept	Weekly_Sales	Temperature	Fuel_Price	MarkDo
count	420285.000000	420285.000000	420285.000000	420285.000000	420285.000000	420285.00
mean	22.195477	44.242771	16030.329773	60.090474	3.360888	2590.18
std	12.787213	30.507197	22728.500149	18.448260	0.458523	6053.22
min	1.000000	1.000000	0.000000	-2.060000	2.472000	0.00
25%	11.000000	18.000000	2117.560000	46.680000	2.933000	0.00
50%	22.000000	37.000000	7659.090000	62.090000	3.452000	0.00
75%	33.000000	74.000000	20268.380000	74.280000	3.738000	2801.50
max	45.000000	99.000000	693099.360000	100.140000	4.468000	88646.76

Check data type

In [82]:
```
#X_train['Type'].value_counts()
X_train.dtypes
```

Out[82]:
```
Store                    int64
Dept                     int64
Date            datetime64[ns]
IsHoliday                 bool
Temperature            float64
Fuel_Price             float64
MarkDown1              float64
MarkDown2              float64
MarkDown3              float64
MarkDown4              float64
MarkDown5              float64
CPI                    float64
Unemployment           float64
```

```
Type                 object
Size                  int64
Year                  int64
Month                 int64
Day                   int64
Day_of_year           int64
Week_of_year          int64
Month_start            bool
Month_end              bool
week_in_month
int64 dtype: object
```

ARIMA

```
# It is necessary to have the date columns present in the dataset to be avai
# Hence converting the string formatted Date into datetime format.
data.Date = pd.to_datetime(data.Date,format='%Y-%m-%d')
data.index = data.Date
data = data.drop('Date', axis=1)
```

In [83]:

In [84]:
```
data.describe()
```

Out[84]:

	Store	Dept	Weekly_Sales	Temperature	Fuel_Price	MarkDo
count	420285.000000	420285.000000	420285.000000	420285.000000	420285.000000	420285.00
mean	22.195477	44.242771	16030.329773	60.090474	3.360888	2590.18
std	12.787213	30.507197	22728.500149	18.448260	0.458523	6053.22
min	1.000000	1.000000	0.000000	-2.060000	2.472000	0.00
25%	11.000000	18.000000	2117.560000	46.680000	2.933000	0.00
50%	22.000000	37.000000	7659.090000	62.090000	3.452000	0.00
75%	33.000000	74.000000	20268.380000	74.280000	3.738000	2801.50
max	45.000000	99.000000	693099.360000	100.140000	4.468000	88646.76

In [85]:
```
data = data.resample('MS').mean() #
Resmapling the time series data with mont
```

In [86]:
```
data.head()
```

22.025485	44.211836	16115.148511	0.249682	35.584066	2.693198	0.0
22.074351	44.070778	15476.271493	0.000000	46.695720	2.786447	0.0
22.088466	44.150672	15784.325587	0.000000	56.407791	2.867529	0.0
22.150366	44.298667	16061.212479	0.000000	64.883664	2.917184	0.0
22.216235	44.230524	16548.757204	0.000000	74.595623	2.787994	0.0

Out[86]:

	Store	Dept	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	M
Date								
2010-02-01								
2010-03-01								
2010-04-01								
2010-05-01								
2010-06-01								

5 rows × 22 columns

In [87]:
```
# Train-Test splitting of time series data
train_data = data[:int(0.7*(len(data)))]
test_data = data[int(0.7*(len(data))):]

print('Train:', train_data.shape)
print('Test:', test_data.shape)
```

```
Train: (23, 22)
Test: (10, 22)
```

In [88]:
```
print('Train data:\n')
print(train_data.tail())
print('='*50,'\n')
print('Test data:\n')
print(test_data.head())
```

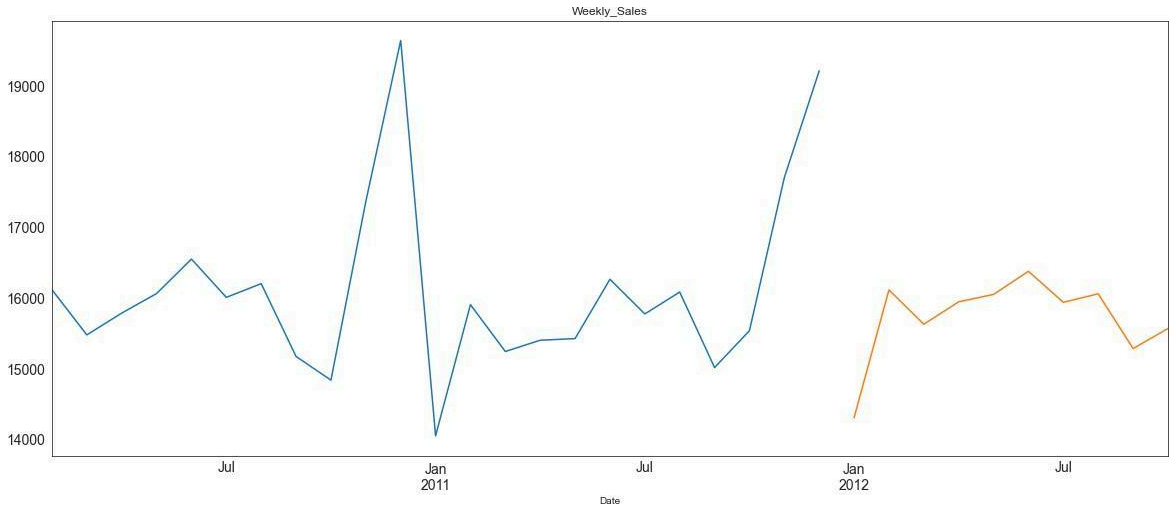
```
Train data:

                Store       Dept  Weekly_Sales  IsHoliday  Temperature  \
Date
2011-08-01  22.242006  44.356613  16082.682055   0.000000    79.810086
2011-09-01  22.207410  44.091774  15013.965477   0.200612    72.491448
2011-10-01  22.147677  44.216090  15536.033513   0.000000    60.285056
2011-11-01  22.164828  44.374295  17700.949518   0.254022    49.549289
2011-12-01  22.178926  44.644548  19211.934999   0.199066    40.162278

            Fuel_Price    MarkDown1    MarkDown2     MarkDown3    MarkDown4
\
Date
2011-08-01    3.699652     0.000000     0.000000      0.000000     0.000000
2011-09-01    3.648625     0.000000     0.000000      0.000000     0.000000
2011-10-01    3.524900     0.000000     0.000000      0.000000     0.000000
2011-11-01    3.484023  4568.775420  2619.660126  14553.279071  1099.829074
2011-12-01    3.332347  3526.042467  8363.970291   1189.398194  1273.053472

            ...  Unemployment           Size    Year  Month        Day  \
Date        ...
2011-08-01  ...      8.052652  136949.137802  2011.0    8.0  15.533129
2011-09-01  ...      8.047771  136920.280761  2011.0    9.0  15.995241
2011-10-01  ...      7.763311  136511.822313  2011.0   10.0  17.505341
2011-11-01  ...      7.764858  136616.262360  2011.0   11.0  14.559252
2011-12-01  ...      7.769400  136280.271090  2011.0   12.0  16.003735

            Day_of_year  Week_of_year  Month_start   Month_en  week_in_mont
h                                                          d
Date
2011-08-01   227.533129     32.504733          0.0   0.000000       2.50473
3
2011-09-01   258.995241     36.999320          0.0   0.199728       2.99932
0
2011-10-01   290.505341     41.500763          0.0   0.000000       2.50076
3
2011-11-01   318.559252     45.508465          0.0   0.000000       2.50846
5
2011-12-01   350.003735     50.000534          0.0   0.000000       3.00053
4

[5 rows x 22 columns]
==================================================

Test data:

                Store       Dept  Weekly_Sales  IsHoliday  Temperature  \
Date
2012-01-01  22.186330  44.217667  14304.774064   0.000000    40.193324
2012-02-01  22.199413  44.516023  16112.853323   0.251174    41.859007
2012-03-01  22.239420  44.230915  15626.987004   0.000000    52.002728
```

```
2012-04-01  22.212592  44.292852  15944.188683   0.000000    59.310268
2012-05-01  22.235504  44.491923  16049.064164   0.000000    66.679698

            Fuel_Price     MarkDown1    MarkDown2   MarkDown3     MarkDown4
\
Date
2012-01-01    3.369467   3106.852436  9375.565479   84.308777    580.552694
2012-02-01    3.533896  17638.770146  5458.023120   61.991177  12365.539905
2012-03-01    3.813175   8566.833642   512.630812    9.364497   3759.472833
2012-04-01    3.974822   5480.360158  1959.732838   25.618202   1186.180023
2012-05-01    3.850028   8679.569274    17.359293  156.480443   2119.002806

            ...  Unemployment           Size    Year  Month        Day  \
Date        ...
2012-01-01  ...      7.482099  136357.435081  2012.0    1.0  16.481918
2012-02-01  ...      7.484182  136534.094211  2012.0    2.0  13.487668
2012-03-01  ...      7.484947  136203.840904  2012.0    3.0  15.965035
2012-04-01  ...      7.401690  136315.606549  2012.0    4.0  16.469575
2012-05-01  ...      7.400577  136560.111631  2012.0    5.0  14.483931

            Day_of_year  Week_of_year  Month_start   Month_en  week_in_mont
h                                                          d
Date
2012-01-01    16.481918      2.497417          0.0        0.0       2.49741
7
2012-02-01    44.487668      6.498238          0.0        0.0       2.49823
8
2012-03-01    75.965035     10.995005          0.0        0.0       2.99500
5
2012-04-01   107.469575     15.495654          0.0        0.0       2.49565
4
2012-05-01   135.483931     19.497704          0.0        0.0       2.49770
4

[5 rows x 22 columns]
```

In [89]:

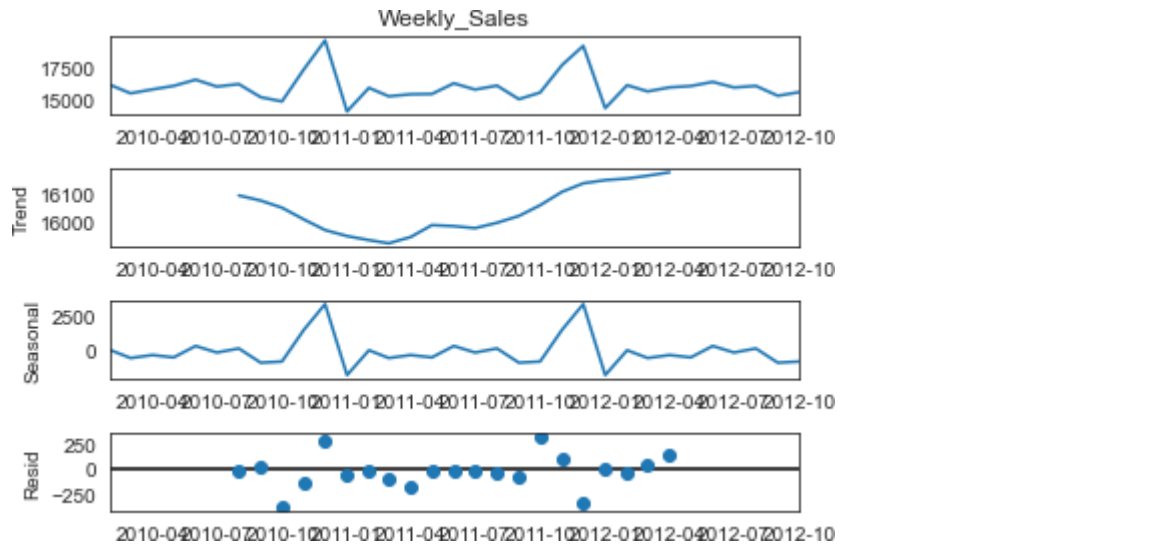
```
# ARIMA takes univariate data.
train_data = train_data['Weekly_Sales']
test_data = test_data['Weekly_Sales']
```

In [90]:
```
# Plot of Weekly_Sales with respect to years in train and test.
train_data.plot(figsize=(20,8), title= 'Weekly_Sales', fontsize=14)
test_data.plot(figsize=(20,8), title= 'Weekly_Sales', fontsize=14)
plt.show()
```




```
# Decomposition of time series data. It is necessary to see whether the trend
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(data['Weekly_Sales'], model='additive')
result.plot()
plt.show()
```

In [91]:



Observation: The trend of weekly sales is such that it first decreases then elevates. Given walmart dataset is seasonal dataset as it is repeating the same pattern in the month of Nov-Dec.

Dickey-Fuller Test for Stationarity: Officially, this is called the ‘augmented Dickey-Fuller test’, but most folks just say ‘Dickey-Fuller’ when talking about it. This is a test that tests the null hypothesis that a unit root is present in time series data. To make things a bit more clear, this test is checking for stationarity or non-stationary data. The test is trying to reject the null hypothesis that a unit root exists and the data is non-stationary. If the null hypothesis is rejected, then the alternate can be considered valid (e.g., the data is stationary). When you run the test, you’ll get an ADF value and a p-value. The ADF number should be a negative number and the p-value should be beneath a certain threshold value (e.g., 1% or 5%, etc) for a confidence level. Here, we'll use 5% (or 95% confidence level), so if the p-value is greater than 0.05 then we say we fail to reject the null hypothesis because the data has a unit root and is non-stationary. If the p-value is less than or equal to 0.05, we can say we reject the null hypothesis because the data does not have a unit root and is stationary.

In [92]:
```
# A check of sationarity of data using Dicky-Fuller test.
from statsmodels.tsa.stattools import adfuller
result = adfuller(data['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: -4.173916935101503
p-value: 0.0007291844915317363
Critical Values:
        1%: -3.769732625845229
        5%: -3.005425537190083
        10%: -2.6425009917355373
```

In [93]:
```
pip install pmdarima
```

```
Requirement already satisfied: pmdarima in
c:\users\gayat\anaconda3\lib\sit e-packages (1.7.1)
Requirement already satisfied: urllib3 in c:\users\gayat\anaconda3\lib\site
-packages (from pmdarima) (1.25.9)
Requirement already satisfied: Cython<0.29.18,>=0.29 in
c:\users\gayat\anac onda3\lib\site-packages (from pmdarima) (0.29.17)
Requirement already satisfied: setuptools<50.0.0 in
c:\users\gayat\anaconda 3\lib\site-packages (from pmdarima)
(49.2.0.post20200714)
Requirement already satisfied: numpy>=1.17.3 in
c:\users\gayat\anaconda3\li b\site-packages (from pmdarima) (1.18.5)
Requirement already satisfied: scipy>=1.3.2 in c:\users\gayat\anaconda3\lib
\site-packages (from pmdarima) (1.5.0)
Requirement already satisfied: joblib>=0.11 in c:\users\gayat\anaconda3\lib
\site-packages (from pmdarima) (0.16.0)
Requirement already satisfied: scikit-learn>=0.22 in
c:\users\gayat\anacond a3\lib\site-packages (from pmdarima) (0.23.1)
Requirement already satisfied: pandas>=0.19 in c:\users\gayat\anaconda3\lib
\site-packages (from pmdarima) (1.0.5)
Requirement already satisfied: statsmodels<0.12,>=0.11 in
c:\users\gayat\an aconda3\lib\site-packages (from pmdarima) (0.11.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\users\gayat\anaco nda3\lib\site-packages (from
scikit-learn>=0.22->pmdarima) (2.1.0)
Requirement already satisfied: pytz>=2017.2 in c:\users\gayat\anaconda3\lib
\site-packages (from pandas>=0.19->pmdarima) (2020.1)
Requirement already satisfied: python-dateutil>=2.6.1 in
c:\users\gayat\ana conda3\lib\site-packages (from pandas>=0.19->pmdarima)
(2.8.1)
Requirement already satisfied: patsy>=0.5 in
c:\users\gayat\anaconda3\lib\s ite-packages (from
statsmodels<0.12,>=0.11->pmdarima) (0.5.1)
Requirement already satisfied: six>=1.5 in
c:\users\gayat\anaconda3\lib\sit e-packages (from
python-dateutil>=2.6.1->pandas>=0.19->pmdarima) (1.15.0)
Note: you may need to restart the kernel to use updated packages.
```

In [94]:
```
# auto-arima library comes under pyramid
utility function, hence installing p
import pmdarima
#from pmdarima import auto_arima
from pmdarima import ARIMA, auto_arima
```

In [95]:
```
# Applying auto_arima model on train data.
#model_auto_arima = auto_arima(train_data, trace=True, error_action='ignore',
model_auto_arima = auto_arima(train_data, trace=True,start_p=0, start_q=0, st
                  max_p=10, max_q=10, max_P=10, max_Q=10, seasonal=True,
                  stepwise=False, suppress_warnings=True, D=1, max_D=10,
                  error_action='ignore',approximation = False)
model_auto_arima.fit(train_data)
```

```
 ARIMA(0,0,0)(0,0,0)[1] intercept  : AIC=398.029, Time=0.01 sec
 ARIMA(0,0,1)(0,0,0)[1] intercept  : AIC=399.604, Time=0.05 sec
 ARIMA(0,0,2)(0,0,0)[1] intercept  : AIC=397.039, Time=0.13 sec
 ARIMA(0,0,3)(0,0,0)[1] intercept  : AIC=397.780, Time=0.16 sec
 ARIMA(0,0,4)(0,0,0)[1] intercept  : AIC=399.228, Time=0.17 sec
 ARIMA(0,0,5)(0,0,0)[1] intercept  : AIC=425.783, Time=0.20 sec
 ARIMA(1,0,0)(0,0,0)[1] intercept  : AIC=399.847, Time=0.01 sec
 ARIMA(1,0,1)(0,0,0)[1] intercept  : AIC=401.739, Time=0.02 sec
 ARIMA(1,0,2)(0,0,0)[1] intercept  : AIC=404.410, Time=0.04 sec
 ARIMA(1,0,3)(0,0,0)[1] intercept  : AIC=400.484, Time=0.09 sec
 ARIMA(1,0,4)(0,0,0)[1] intercept  : AIC=399.763, Time=0.27 sec
 ARIMA(2,0,0)(0,0,0)[1] intercept  : AIC=399.812, Time=0.06 sec
 ARIMA(2,0,1)(0,0,0)[1] intercept  : AIC=403.489, Time=0.16 sec
 ARIMA(2,0,2)(0,0,0)[1] intercept  : AIC=405.580, Time=0.09 sec
 ARIMA(2,0,3)(0,0,0)[1] intercept  : AIC=402.550, Time=0.26 sec
 ARIMA(3,0,0)(0,0,0)[1] intercept  : AIC=401.000, Time=0.11 sec
 ARIMA(3,0,1)(0,0,0)[1] intercept  : AIC=403.157, Time=0.15 sec
 ARIMA(3,0,2)(0,0,0)[1] intercept  : AIC=405.443, Time=0.31 sec
 ARIMA(4,0,0)(0,0,0)[1] intercept  : AIC=402.920, Time=0.12 sec
 ARIMA(4,0,1)(0,0,0)[1] intercept  : AIC=405.119, Time=0.29 sec
 ARIMA(5,0,0)(0,0,0)[1] intercept   : AIC=404.381, Time=0.12 sec
Total fit time: 2.841 seconds
```

Out[95]:
```
ARIMA(order=(0, 0, 2), scoring_args={}, seasonal_order=(0, 0, 0, 1),
      suppress_warnings=True)
```

In [96]:
```
model_auto_arima.summary()
```

Out[96]:

SARIMAX Results

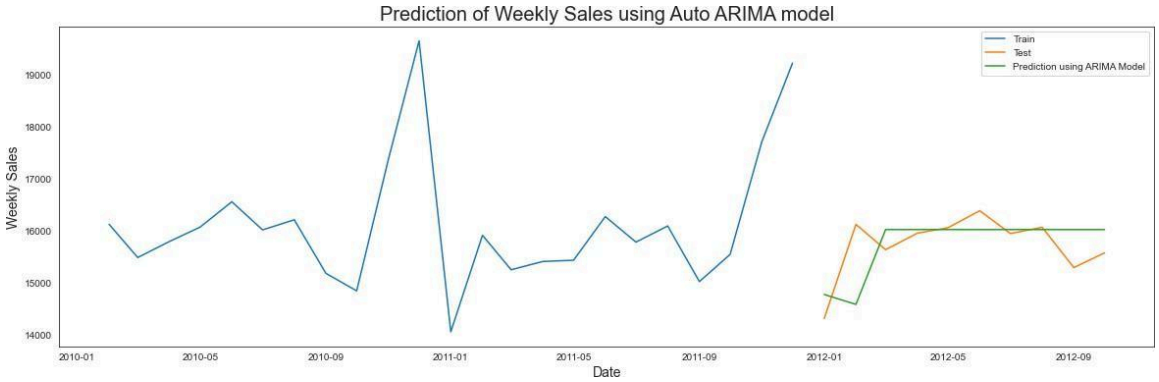
Dep. Variable:	y	**No. Observations:**	23
Model:	SARIMAX(0, 0, 2)	**Log Likelihood**	-194.520
Date:	Wed, 25 Nov 2020	**AIC**	397.039
Time:	17:54:08	**BIC**	401.581
Sample:	0	**HQIC**	398.182
	- 23		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	1.601e+04	54.085	296.086	0.000	1.59e+04	1.61e+04
ma.L1	-0.4352	0.548	-0.794	0.427	-1.509	0.638
ma.L2	-0.5623	0.235	-2.391	0.017	-1.023	-0.101
sigma2	1.157e+06	0.000	5.69e+09	0.000	1.16e+06	1.16e+06

Ljung-Box (Q):	18.78	**Jarque-Bera (JB):**	11.13
Prob(Q):	0.66	**Prob(JB):**	0.00
Heteroskedasticity (H):	12.38	**Skew:**	1.42
Prob(H) (two-sided):	0.00	**Kurtosis:**	4.87

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 7.34e+26. Standard errors may be unstable.

In [97]:
```
# Predicting the test values using predict function.
forecast = model_auto_arima.predict(n_periods=len(test_data))
forecast = pd.DataFrame(forecast,index = test_data.index,columns=['Prediction
plt.figure(figsize=(20,6))
plt.title('Prediction of Weekly Sales using Auto ARIMA model', fontsize=20)
plt.plot(train_data, label='Train')
plt.plot(test_data, label='Test')
plt.plot(forecast, label='Prediction using ARIMA Model')
plt.legend(loc='best')
plt.xlabel('Date', fontsize=14)
plt.ylabel('Weekly Sales', fontsize=14)
plt.show()
```



In [98]:
```
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

In [99]:
```
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

In [100]:
```
# Performance metric for ARIMA model -MSE/RMSE
print('Mean Squared Error (MSE) of ARIMA: ', mean_squared_error(test_data, fo
print('Root Mean Squared Error (RMSE) of ARIMA: ', math.sqrt(mean_squared_err
print('Mean Absolute Deviation (MAD) of ARIMA: ', mean_absolute_error(test_da
```

```
Mean Squared Error (MSE) of ARIMA:  360486.42795007967
Root Mean Squared Error (RMSE) of ARIMA:  600.4052197891684
Mean Absolute Deviation (MAD) of ARIMA:  415.1849590228274
```

2. Holt Winters Model

Holt-Winters is a time-series model and is a way to model three aspects of the time series: a typical value (average), a slope (trend) over time, and a cyclical repeating pattern (seasonality). Holt-Winters uses exponential smoothing to encode lots of values from the past and use them to predict “typical” values for the present and future. Below steps are required to use Holt-Winters model.

Load the data.

Visualize the available univariate data in timely fashion.

Fit the model using Holt-Winters method on train data.

Predict the values using of test data.

Visualize the training data, test data and predicted data on single graph.

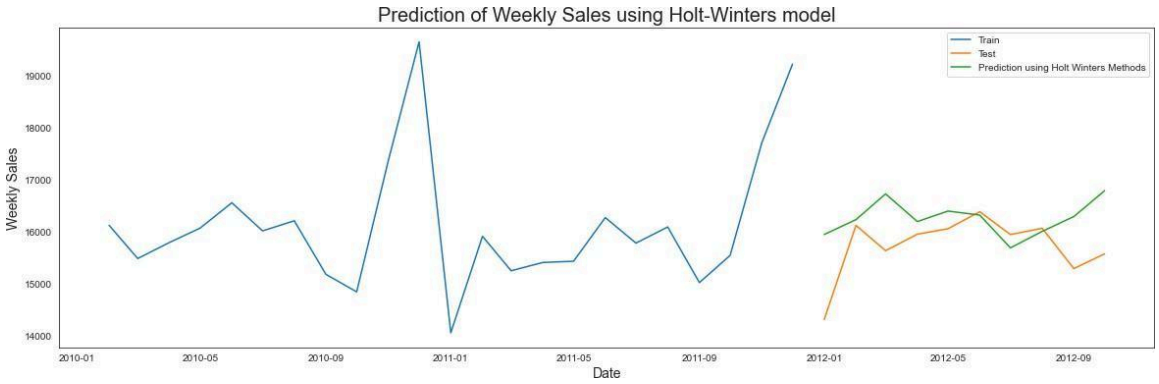
Calculate Root Mean Squared Error (RMSE) on predicted data..

In [101]:
```
print('Train data shape: ', train_data.shape)
print('Test data shape: ', test_data.shape)
print('='*30)
print('Train data:\n')
print(train_data.tail(),'\n')
print('Test data:\n')
print(test_data.head())
```

```
Train data shape:
(23,) Test data shape:
                  (10,)
==============================
Train data:

Date
2011-08-01    16082.682055
2011-09-01    15013.965477
2011-10-01    15536.033513
2011-11-01    17700.949518
2011-12-01    19211.934999
Freq: MS, Name: Weekly_Sales, dtype: float64

Test data:

Date
2012-01-01    14304.774064
2012-02-01    16112.853323
2012-03-01    15626.987004
2012-04-01    15944.188683
2012-05-01    16049.064164
Freq: MS, Name: Weekly_Sales, dtype: float64
```

In [102]:

```
# Fitting the Holt-Winters method for Weekly Sales.
from statsmodels.tsa.api import ExponentialSmoothing
model_holt_winters = ExponentialSmoothing(train_data, seasonal_periods=7, tre
pred = model_holt_winters.forecast(len(test_data))# Predict the test data

#Visualize train, test and predicted data.
plt.figure(figsize=(20,6))
plt.title('Prediction of Weekly Sales using Holt-Winters model', fontsize=20)
plt.plot(train_data, label='Train')
plt.plot(test_data, label='Test')
plt.plot(pred, label='Prediction using Holt Winters Methods')
plt.legend(loc='best')
plt.xlabel('Date', fontsize=14)
plt.ylabel('Weekly Sales', fontsize=14)
plt.show()
```



In [103]:
```
model_holt_winters.summary()
```

Out[103]:

ExponentialSmoothing Model Results

Dep. Variable:	endog	**No. Observations:**	23
Model:	ExponentialSmoothing	**SSE**	44778377.928
Optimized:	True	**AIC**	355.080
Trend:	Additive	**BIC**	367.570
Seasonal:	Additive	**AICC**	395.525
Seasonal Periods:	7	**Date:**	Wed, 25 Nov 2020
Box-Cox:	False	**Time:**	17:54:09
Box-Cox Coeff.:	None		

	coeff	code	optimized
smoothing_level	0.0526313	alpha	True
smoothing_slope	0.0526366	beta	True
smoothing_seasonal	9.9999e-05	gamma	True
initial_level	16096.920	l.0	True
initial_slope	0.5576630	b.0	True
initial_seasons.0	18.220141	s.0	True
initial_seasons.1	-620.62193	s.1	True
initial_seasons.2	-312.57738	s.2	True
initial_seasons.3	-35.676749	s.3	True
initial_seasons.4	451.79746	s.4	True
initial_seasons.5	-89.873708	s.5	True
initial_seasons.6	104.19114	s.6	True

In [104]:
```
# Performance metric for Holt-Winters model -MSE/RMSE
print('Mean Squared Error (MSE) of Holt-Winters: ', mean_squared_error(test_d
print('Root Mean Squared Error (RMSE) of Holt-Winters: ', math.sqrt(mean_squa
print('Mean Absolute Deviation (MAD) of Holt-Winters: ', mean_absolute_error(
```

```
Mean Squared Error (MSE) of Holt-Winters:  658969.440361476
Root Mean Squared Error (RMSE) of Holt-Winters:  811.7693270637146
Mean Absolute Deviation (MAD) of Holt-Winters:  601.0603637923869
```

Model Comparison

In [105]:
```
models = pd.DataFrame({
'Model Name': ['ARIMA','HOLT-WINTER'],

'RMSE Score': ['598.70','998.54'],
'AIC': ['396.99','354.62'],

    })

Index = pd.Series([1, 2])
models.set_index(Index, inplace=True)
models
```

Out[105]:

	Model Name	RMSE Score	AIC
1	ARIMA	598.70	396.99
2	HOLT-WINTER	998.54	354.62

Machine Learning Models

In [106]:
```
pip install feature-engine
```

```
Requirement already satisfied: feature-engine in
c:\users\gayat\anaconda3\l ib\site-packages (0.6.1)
Requirement already satisfied: statsmodels>=0.11.1 in
c:\users\gayat\anacon da3\lib\site-packages (from feature-engine) (0.11.1)
Requirement already satisfied: numpy>=1.18.2 in
c:\users\gayat\anaconda3\li b\site-packages (from feature-engine) (1.18.5)
Requirement already satisfied: scikit-learn>=0.22.2 in
c:\users\gayat\anaco nda3\lib\site-packages (from feature-engine) (0.23.1)
Requirement already satisfied: pandas>=1.0.3 in
c:\users\gayat\anaconda3\li b\site-packages (from feature-engine) (1.0.5)
Requirement already satisfied: scipy>=1.4.1 in c:\users\gayat\anaconda3\lib
\site-packages (from feature-engine) (1.5.0)
Requirement already satisfied: patsy>=0.5 in
c:\users\gayat\anaconda3\lib\s ite-packages (from
statsmodels>=0.11.1->feature-engine) (0.5.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\users\gayat\anaco nda3\lib\site-packages (from
scikit-learn>=0.22.2->feature-engine) (2.1.0) Requirement already
satisfied: joblib>=0.11 in c:\users\gayat\anaconda3\lib
\site-packages (from scikit-learn>=0.22.2->feature-engine) (0.16.0)
Requirement already satisfied: python-dateutil>=2.6.1 in
c:\users\gayat\ana conda3\lib\site-packages (from
pandas>=1.0.3->feature-engine) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in c:\users\gayat\anaconda3\lib
\site-packages (from pandas>=1.0.3->feature-engine) (2020.1)
Requirement already satisfied: six in c:\users\gayat\anaconda3\lib\site-pac
kages (from patsy>=0.5->statsmodels>=0.11.1->feature-engine) (1.15.0)
Note: you may need to restart the kernel to use updated packages.
```

One hot encoding is applied to the "Type" variable after 80:20 split

- Step 1: Fit/train the one hot encoder to X_train set
- Step 2: Transform X_train set using the trained one hot encoder

- Step 3: Transform X_test set using the encoder trained on X_train set

In [107]:
```
#Step: 1
from feature_engine.categorical_encoders import OneHotCategoricalEncoder
ohe_enc = OneHotCategoricalEncoder(
    top_categories=None,
    variables=['Type'], # we can select which variables to encode
    ) # to return k-1, false to return k


ohe_enc.fit(X_train.fillna('Missing'))
```

Out[107]: OneHotCategoricalEncoder(variables=['Type'])

In [108]:
```
#Step: 2
X_train = ohe_enc.transform(X_train.fillna('Missing'))

X_train.head()
```

	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2	Mar
	False	82.58	3.355	0.00	0.00	
	False	67.91	3.845	12063.62	6.78	
	False	83.86	3.374	29127.20	42.27	
	False	50.81	2.843	0.00	0.00	
	False	83.75	2.623	0.00	0.00	

Out[108]:

	Store	Dept	Date
25943	3	90	201 09-:
56822	6	36	2012 03-:
38872	4	22	2012 08-[
364175	39	79	2010 12-
50463	6	37	2010

	1- 0
	- 0
	- 3
	- 0
	-

5 rows × 25 columns

In [109]:
```
#Step: 3
X_test = ohe_enc.transform(X_test.fillna('Missing'))

X_test.head()
```

Out[109]:

	Store	Dept	Da	IsHoliday	Temperature	Fuel_Price	MarkDown1	Mark
265327	27	83	2012-05-	False	65.15	4.029	13340.68	
419912	45	24	2012-05-	False	61.24	3.889	12611.18	
29049	3	49	2012-09-	True	84.99	3.730	2266.83	
381198	41	16	2010-02-	False	21.84	2.586	0.00	
52464	6	67	201	False	44.98	3.010	0.00	

e

-8

-1

-7

-6

1-01-28

5 rows × 25 columns

In [110]:
```
# Remove Date column as it does not allow the models to fit on the data.
X_train = X_train.drop(['Date'], axis=1)
X_test = X_test.drop(['Date'], axis=1)
```

In [ ]:

XGBoost Regressor

```
from xgboost import XGBRegressor
model_xgb = XGBRegressor(n_estimators=20, max_depth=15, objective='reg:squar
y_predXGB = model_xgb.predict(X_test) # Predict test data.
```

In [111]:

In [112]:

```
def wmae_train(test, pred): # WMAE for train
  weights = X_train['IsHoliday'].apply(lambda is_holiday:5 if is_holiday else
error = np.sum(weights * np.abs(test - pred), axis=0) / np.sum(weights)
  return error

def wmae_test(test, pred): # WMAE for test
  weights = X_test['IsHoliday'].apply(lambda is_holiday:5 if is_holiday else
error = np.sum(weights * np.abs(test - pred), axis=0) / np.sum(weights)
  return error
```

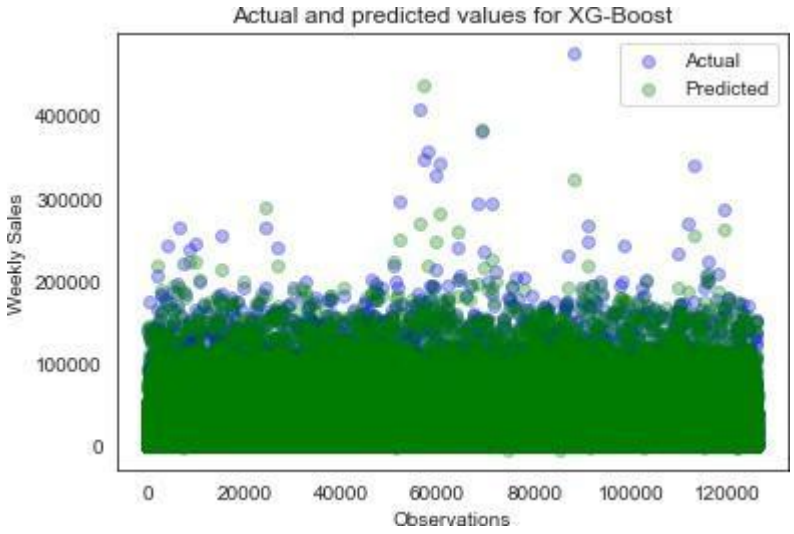
In [113]:

```
print('Weighted Mean Absolute Error (WMAE) for
```

```
Weighted Mean Absolute Error (WMAE) for
XGBoost Regression: 1468.9990830391 932
```

In [114]:

```
import matplotlib.pyplot as plt
_, ax = plt.subplots()

ax.scatter(x = range(0, y_test.size), y=y_test, c = 'blue', label = 'Actual',
ax.scatter(x = range(0, y_predXGB.size), y=y_predXGB, c = 'green', label = 'P

plt.title('Actual and predicted values for XG-Boost')
plt.xlabel('Observations')
plt.ylabel('Weekly Sales')
plt.legend()
plt.show()

ax.scatter(x = range(0, y_predXGB.size), y=y_predXGB, c = 'green', label = 'P
```



Out[114]: <matplotlib.collections.PathCollection at 0x2a186e0daf0>

Random Forest Regressor

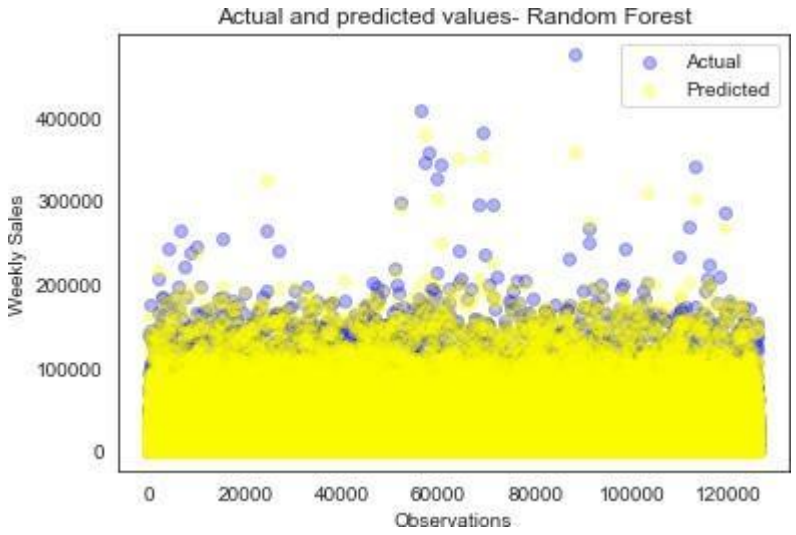
In [115]:

```
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor

model_rf = RandomForestRegressor(max_depth= 35, n_estimators=80).fit(X_train,
y_pred = model_rf.predict(X_test) # Predict the test data.
print('Weighted Mean Absolute Error (WMAE) for Random Forest Regression:', wm
```

```
Weighted Mean Absolute Error (WMAE) for Random Forest Regression: 1516.4991
061599371
```

In [116]:

```
import matplotlib.pyplot as plt
_, ax = plt.subplots()

ax.scatter(x = range(0, y_test.size), y=y_test, c = 'blue', label = 'Actual',
ax.scatter(x = range(0, y_pred.size), y=y_pred, c = 'yellow', label = 'Predic

plt.title('Actual and predicted values- Random Forest')
plt.xlabel('Observations')
plt.ylabel('Weekly Sales')
plt.legend()
plt.show()

ax.scatter(x = range(0, y_pred.size), y=y_pred, c = 'green', label = 'Predict
```



Out[116]: <matplotlib.collections.PathCollection at 0x2a19fc7ff40>

Decision Tree Regression

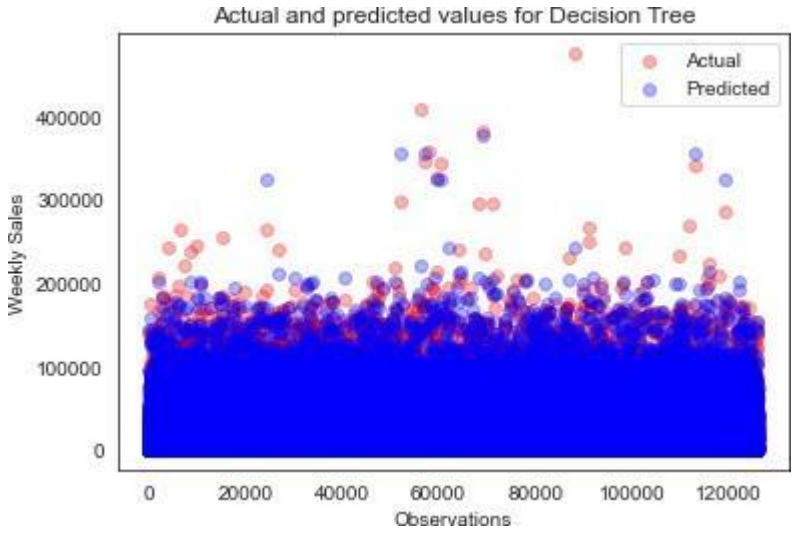
In [117]:

```
from sklearn.tree import DecisionTreeRegressor
"""Calculate Prediction and WMAE score."""

model_dt = DecisionTreeRegressor(max_depth=25, min_samples_leaf=5).fit(X_trai
y_predDT = model_dt.predict(X_test) # Predict the test data.
print('Weighted Mean Absolute Error (WMAE) for Decision Tree Regression:', wm
```

```
Weighted Mean Absolute Error (WMAE) for Decision Tree Regression: 1913.5849
502375784
```

In [118]:

```
import matplotlib.pyplot as plt
_, ax = plt.subplots()

ax.scatter(x = range(0, y_test.size), y=y_test, c = 'red', label = 'Actual',
ax.scatter(x = range(0, y_predDT.size), y=y_predDT, c = 'blue', label = 'Pred

plt.title('Actual and predicted values for Decision Tree')
plt.xlabel('Observations')
plt.ylabel('Weekly Sales')
plt.legend()
plt.show()

#ax.scatter(x = range(0, y_predDT.size), y=y_predDT, c = 'green', label = 'Pr
```



Linear Regression

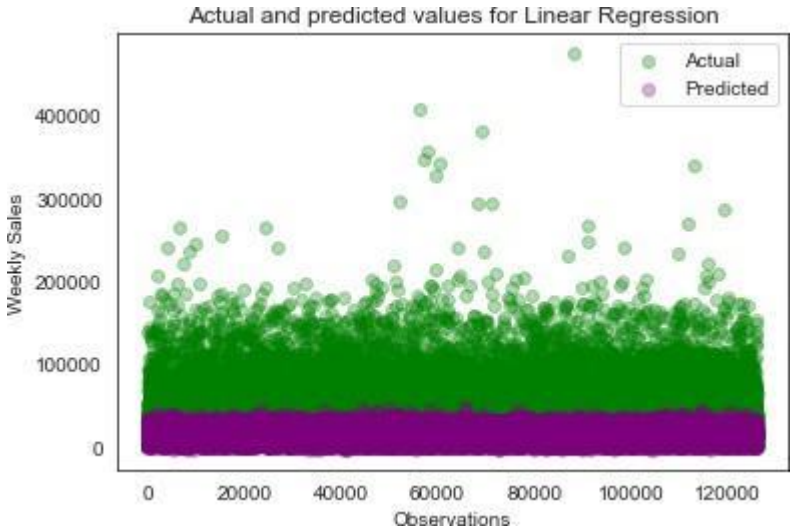
In [119]:

```
from sklearn.linear_model import LinearRegression
"""Calculate Prediction and WMAE score."""

model_linear_reg = LinearRegression(fit_intercept=True,normalize=True).fit(X_
y_predlr = model_linear_reg.predict(X_test) # Predict test data.
print('Weighted Mean Absolute Error (WMAE) for Linear Regression:', wmae_test
```

```
Weighted Mean Absolute Error (WMAE) for Linear Regression: 14756.0147360431
27
```

In [120]:

```
import matplotlib.pyplot as plt
_, ax = plt.subplots()

ax.scatter(x = range(0, y_test.size), y=y_test, c = 'green', label = 'Actual'
ax.scatter(x = range(0, y_predlr.size), y=y_predlr, c = 'purple', label = 'Pr

plt.title('Actual and predicted values for Linear Regression')
plt.xlabel('Observations')
plt.ylabel('Weekly Sales')
plt.legend()
plt.show()

ax.scatter(x = range(0, y_predlr.size), y=y_predlr, c = 'green', label = 'Pre
```



Out[120]: <matplotlib.collections.PathCollection at 0x2a19fc5a2e0>

Results

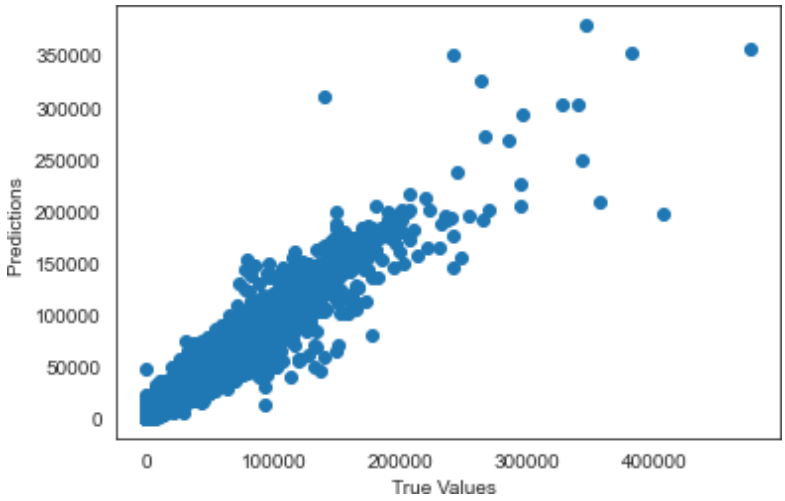
In [121]:
```
models = pd.DataFrame({
'Model Name': ['Linear Regression','XGBoost Regession','Decision Tree Reg

'WMAE Score': ['14913.115', '2146.341', '2525.476', '2231.912']
    })

Index = pd.Series([1, 2, 3, 4])
models.set_index(Index, inplace=True)
models
```

Out[121]:

	Model Name	WMAE Score
1	Linear Regression	14913.115
2	XGBoost Regession	2146.341
3	Decision Tree Regression	2525.476
4	Random Forest Regression	2231.912

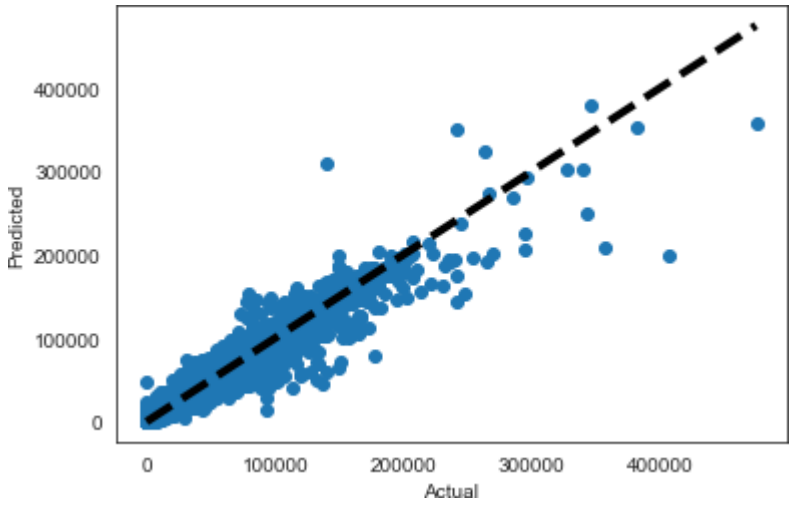
In [122]:
```
plt.scatter(y_test, y_pred)
plt.xlabel('True Values ')
plt.ylabel('Predictions ')
# plt.axis('equal')
# plt.axis('square')
# plt.xlim([0, plt.xlim()])
# plt.ylim([0, plt.ylim()])
plt.plot()
```

Out[122]: []



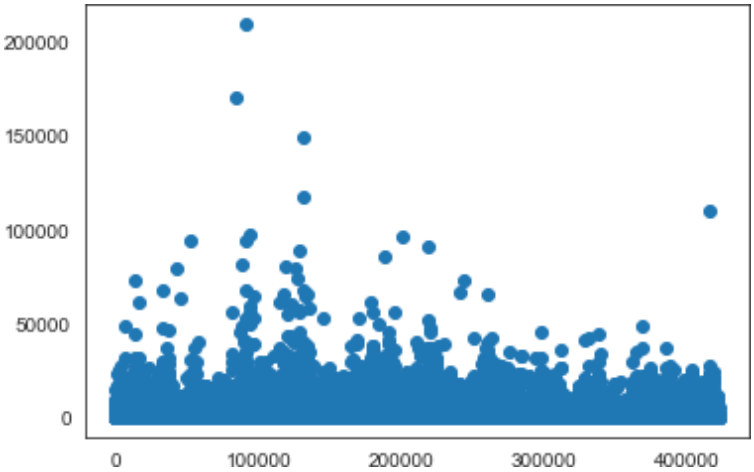
In [123]:

```
fig, ax = plt.subplots()
ax.scatter(y_test, y_pred)
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw
 ax.set_xlabel('Actual')
ax.set_ylabel('Predicted')
 plt.show()
```



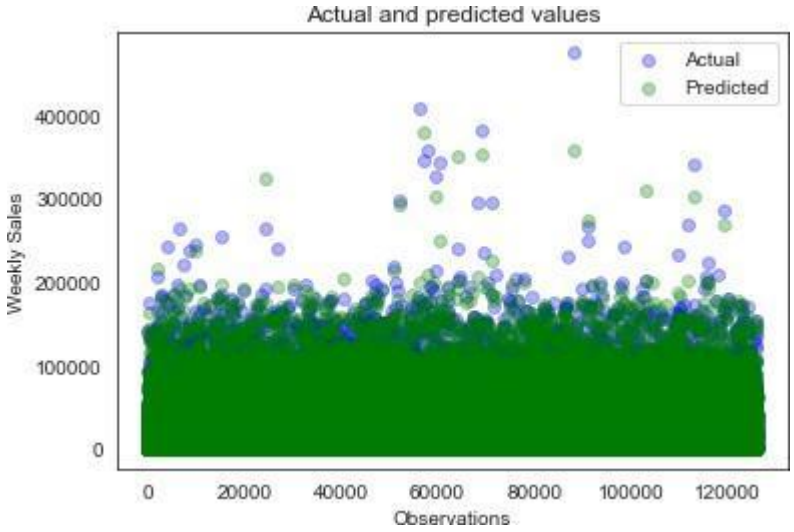
In [124]:

```
g=plt.plot(np.abs(y_test - y_pred),marker='o',linestyle='')
```



In [125]:

```
import matplotlib.pyplot as plt
_, ax = plt.subplots()

ax.scatter(x = range(0, y_test.size), y=y_test, c = 'blue', label = 'Actual',
ax.scatter(x = range(0, y_pred.size), y=y_pred, c = 'green', label = 'Predict

plt.title('Actual and predicted values')
plt.xlabel('Observations')
plt.ylabel('Weekly Sales')
plt.legend()
plt.show()

ax.scatter(x = range(0, y_pred.size), y=y_pred, c = 'green', label = 'Predict
```



Out[125]: <matplotlib.collections.PathCollection at 0x2a1f6fa6b20>

In [126]:

```
import matplotlib.pyplot as plt
_, ax = plt.subplots()

#ax.scatter(x = range(0, y_test.size), y=y_test, c = 'blue', label = 'Actual'
ax.scatter(x = range(0, y_pred.size), y=y_pred, c = 'green', label = 'Predict

plt.title('Actual and predicted values')
plt.xlabel('Observations')
plt.ylabel('Weekly Sales')
plt.legend()
plt.show()
```