# Regression on Retails Sales Prediction

## By Rahul Inchal

## Problem Description

Rossmann operates over 3,000 drug stores in 7 European countries. Currently, Rossmann store managers are tasked with predicting their daily sales for up to six weeks in advance. Store sales are influenced by many factors, including promotions, competition, school and state holidays, seasonality, and locality. With thousands of individual managers predicting sales based on their unique circumstances, the accuracy of results can be quite varied. You are provided with historical sales data for 1,115 Rossmann stores. The task is to forecast the "Sales" column for the test set. Note that some stores in the dataset were temporarily closed for refurbishment.

## GitHub

https://github.com/rahulinchal/Retail-Sales-Prediction (https://github.com/rahulinchal/Retail-Sales-Prediction)

## Importing packages

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')

# To show all the rows and columns
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

In [2]:
```
rossmann = pd.read_csv("C:/Users\Rahul\Desktop\Data_Science\AlmaBetter\Machine Le
store = pd.read_csv("C:/Users\Rahul\Desktop\Data_Science\AlmaBetter\Machine Learn
```

# Data Description

## Rossmann Dataset

Most of the fields are self-explanatory. The following are descriptions for those that aren't.
**1. Id** - an Id that represents a (Store, Date) duple within the test set
**2. Store** - a unique Id for each store
**3. Sales** - the turnover for any given day (this is what you are predicting)
**4. Customers** - the number of customers on a given day
**5. Open** - an indicator for whether the store was open: 0 = closed, 1 = open
**6. StateHoliday** - indicates a state holiday. Normally all stores, with few exceptions, are closed on state holidays. Note that all schools are closed on public holidays and weekends. a = public holiday, b = Easter holiday, c = Christmas, 0 = None
**7. SchoolHoliday** - indicates if the (Store, Date) was affected by the closure of public schools

## Store Dataset

**1. StoreType** - differentiates between 4 different store models: a, b, c, d
**2. Assortment** - describes an assortment level: a = basic, b = extra, c = extended
**3. CompetitionDistance** - distance in meters to the nearest competitor store
**4. CompetitionOpenSince[Month/Year]** - gives the approximate year and month of the time the nearest competitor was opened
**5. Promo** - indicates whether a store is running a promo on that day
**6. Promo2** - Promo2 is a continuing and consecutive promotion for some stores: 0 = store is not participating, 1 = store is participating
**7. Promo2Since[Year/Week]** - describes the year and calendar week when the store started participating in Promo2
**8. PromoInterval** - describes the consecutive intervals Promo2 is started, naming the months the promotion is started anew. E.g. "Feb,May,Aug,Nov" means each round starts in February, May, August, November of any given year for that store

## Data Wrangling for Rossmann dataset

In [3]:
```
# Getting first 5 rows
rossmann.head(5)
```

Out[3]:

| | Store | DayOfWeek | Date | Sales | Customers | Open | Promo | StateHoliday | SchoolHoliday |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 5 | 2015-07-31 | 5263 | 555 | 1 | 1 | 0 | 1 |
| **1** | 2 | 5 | 2015-07-31 | 6064 | 625 | 1 | 1 | 0 | 1 |
| **2** | 3 | 5 | 2015-07-31 | 8314 | 821 | 1 | 1 | 0 | 1 |
| **3** | 4 | 5 | 2015-07-31 | 13995 | 1498 | 1 | 1 | 0 | 1 |
| **4** | 5 | 5 | 2015-07-31 | 4822 | 559 | 1 | 1 | 0 | 1 |

In [4]:
```
#Getting sales vlaue count
rossmann['Open'].value_counts().iloc[:5]
```

Out[4]:
```
1    844392
0    172817
Name: Open, dtype: int64
```

In [5]:
```
# Getting shape of the rossmann data
rossmann.shape
```

Out[5]:
```
(1017209, 9)
```

### Dropping all the entries from Sales column which has 0 sales. That 0 sales does not give any insights. And, Dropping all the data which has 0 as Open which indicates, that the store was not open. we can drop that as we are looking for sales happend and sales can only happen if the store is open.

In [6]:
```
rossmann = rossmann[rossmann['Sales'] != 0]
rossmann = rossmann[rossmann['Open'] != 0 ]
```

In [7]:
```
rossmann.head()
```

Out[7]:

| | Store | DayOfWeek | Date | Sales | Customers | Open | Promo | StateHoliday | SchoolHoliday |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 5 | 2015-07-31 | 5263 | 555 | 1 | 1 | 0 | 1 |
| **1** | 2 | 5 | 2015-07-31 | 6064 | 625 | 1 | 1 | 0 | 1 |
| **2** | 3 | 5 | 2015-07-31 | 8314 | 821 | 1 | 1 | 0 | 1 |
| **3** | 4 | 5 | 2015-07-31 | 13995 | 1498 | 1 | 1 | 0 | 1 |
| **4** | 5 | 5 | 2015-07-31 | 4822 | 559 | 1 | 1 | 0 | 1 |

In [8]:
```
# Getting the value counts
rossmann['Open'].value_counts()
```

Out[8]:
```
1    844338
Name: Open, dtype: int64
```

## Since open has only one value i.e., 1 we can drop it.

In [9]:
```
#Dropping the open column
rossmann.drop(['Open'], axis = 1, inplace = True)
```

In [10]:
```
#Finding the shape
rossmann.shape
```

Out[10]:
```
(844338, 8)
```

In [11]:
```
# Getting first 5 rows
rossmann.head()
```

Out[11]:

| | Store | DayOfWeek | Date | Sales | Customers | Promo | StateHoliday | SchoolHoliday |
|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 5 | 2015-07-31 | 5263 | 555 | 1 | 0 | 1 |
| **1** | 2 | 5 | 2015-07-31 | 6064 | 625 | 1 | 0 | 1 |
| **2** | 3 | 5 | 2015-07-31 | 8314 | 821 | 1 | 0 | 1 |
| **3** | 4 | 5 | 2015-07-31 | 13995 | 1498 | 1 | 0 | 1 |
| **4** | 5 | 5 | 2015-07-31 | 4822 | 559 | 1 | 0 | 1 |

## Finding the null value_counts

In [12]:
```
rossmann.isnull().sum()
```

Out[12]:
```
Store            0
DayOfWeek        0
Date             0
Sales            0
Customers        0
Promo            0
StateHoliday     0
SchoolHoliday    0
dtype: int64
```

## Finding the duplicated values

In [13]:
```
rossmann.duplicated().sum()
```

Out[13]:
```
0
```

### Since there are no null values and no duplicated values. we can proceed with visualzation

In [14]:
```
# Getting the info of rossmann
rossmann.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 844338 entries, 0 to 1017190
Data columns (total 8 columns):
 #   Column         Non-Null Count   Dtype 
---  ------         --------------   ----- 
 0   Store          844338 non-null  int64 
 1   DayOfWeek      844338 non-null  int64 
 2   Date           844338 non-null  object
 3   Sales          844338 non-null  int64 
 4   Customers      844338 non-null  int64 
 5   Promo          844338 non-null  int64 
 6   StateHoliday   844338 non-null  object
 7   SchoolHoliday  844338 non-null  int64 
dtypes: int64(6), object(2)
memory usage: 58.0+ MB
```

### Changing the format of date to datetime format

In [15]:
```
rossmann['Date'] = pd.to_datetime(rossmann['Date'])
```

### Extracting the year, month, day and, weekday from date column and dropping the date column

In [16]:
```
rossmann['Year'] = rossmann['Date'].dt.year
rossmann['Month'] = rossmann['Date'].dt.month
rossmann['Day'] = rossmann['Date'].dt.day

#Removing weekday bocause we alreaday have one column dayofweek which is the same
#rossmann['Weekday'] = rossmann['Date'].dt.weekday

#renaming the dayofweek column to weekday
rossmann = rossmann.rename(columns={'DayOfWeek': 'Weekday'})

rossmann.drop(['Date'], axis = 1, inplace = True)
```

### Defining Weekday or weekend from Weekday

**If the Weekday is 6 or 7 then it is weekend**

**1 - Monday**

**2 - Tuesday**

**3 - Wednesday**

**4 - Thursday**

**5 - Friday**

**6 - Saturday**

**7- Sunday**

In [17]:
```python
def weekend(val):
    if val == 6:
        return 1
    elif val == 7:
        return 1
    else:
        return 0
```

## Extracting weekday or weekend from weekday column

In [18]:
```python
rossmann['Weekend'] = rossmann['Weekday'].apply(weekend)
```

In [19]:
```python
# Getting random sample
rossmann.sample(10)
```

Out[19]:

| | Store | Weekday | Sales | Customers | Promo | StateHoliday | SchoolHoliday | Year | Month | Da |
|---|---|---|---|---|---|---|---|---|---|---|
| **435760** | 581 | 5 | 6507 | 689 | 1 | 0 | 0 | 2014 | 6 | 6 |
| **994819** | 1025 | 1 | 8828 | 960 | 1 | 0 | 0 | 2013 | 1 | 2 |
| **974844** | 5 | 4 | 4889 | 559 | 1 | 0 | 1 | 2013 | 2 | 7 |
| **783905** | 846 | 1 | 13803 | 1092 | 1 | 0 | 1 | 2013 | 7 | 2 |
| **571057** | 963 | 3 | 12601 | 1130 | 1 | 0 | 0 | 2014 | 2 | 5 |
| **976585** | 631 | 3 | 5009 | 568 | 1 | 0 | 0 | 2013 | 2 | 6 |
| **177123** | 954 | 1 | 4652 | 673 | 0 | 0 | 0 | 2015 | 2 | 2 |
| **8476** | 672 | 5 | 6941 | 1100 | 0 | 0 | 1 | 2015 | 7 | 2 |
| **192589** | 810 | 1 | 4852 | 408 | 0 | 0 | 0 | 2015 | 2 | 9 |
| **15438** | 944 | 6 | 6115 | 1068 | 0 | 0 | 0 | 2015 | 7 | 1 |

In [20]:
```python
rossmann['Weekday'].unique()
```

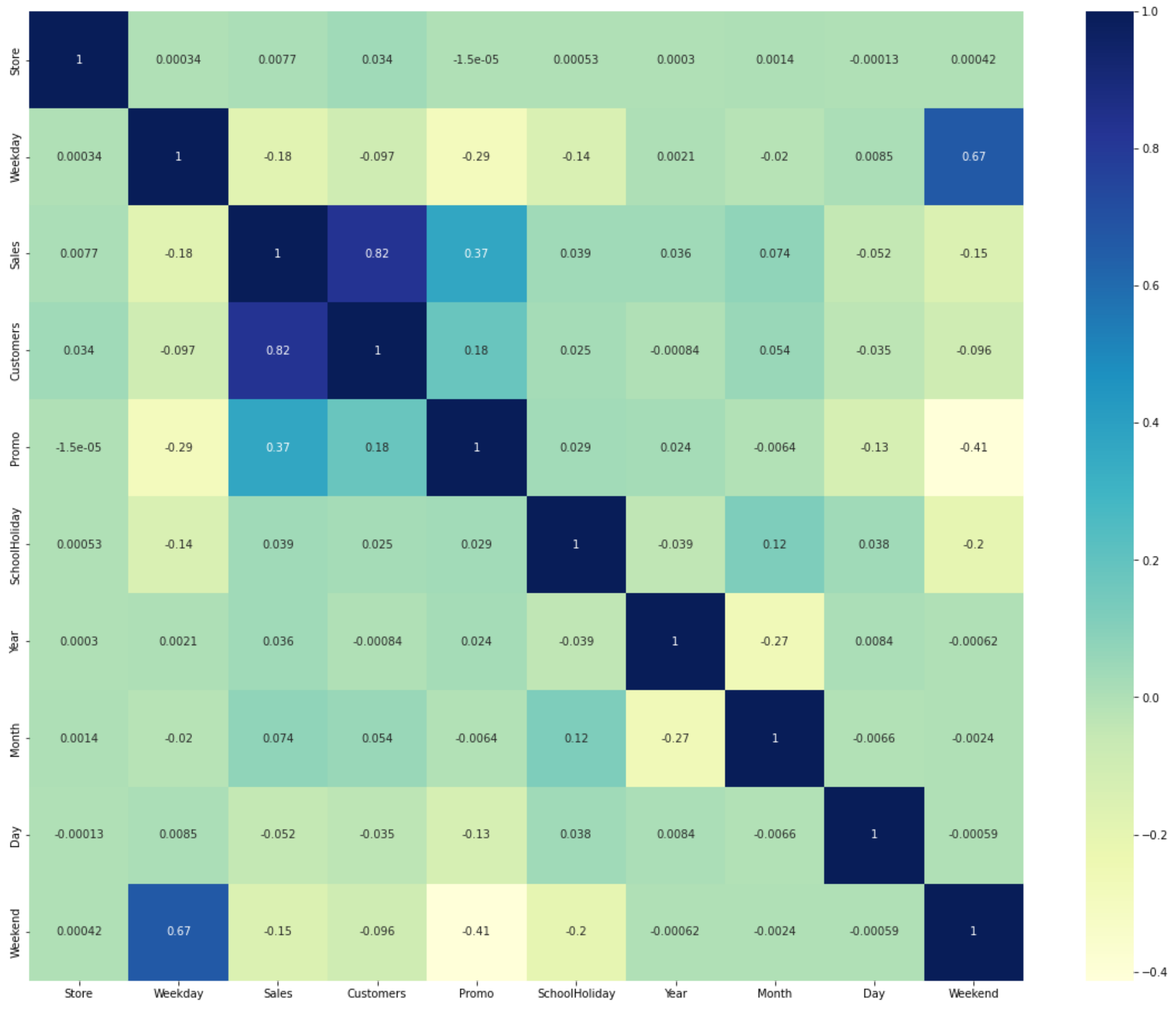
Out[20]:
```
array([5, 4, 3, 2, 1, 7, 6], dtype=int64)
```

```
In [21]:  # Getting info of the data
          rossmann.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 844338 entries, 0 to 1017190
Data columns (total 11 columns):
 #   Column         Non-Null Count   Dtype 
---  ------         --------------   ----- 
 0   Store          844338 non-null  int64 
 1   Weekday        844338 non-null  int64 
 2   Sales          844338 non-null  int64 
 3   Customers      844338 non-null  int64 
 4   Promo          844338 non-null  int64 
 5   StateHoliday   844338 non-null  object
 6   SchoolHoliday  844338 non-null  int64 
 7   Year           844338 non-null  int64 
 8   Month          844338 non-null  int64 
 9   Day            844338 non-null  int64 
 10  Weekend        844338 non-null  int64 
dtypes: int64(10), object(1)
memory usage: 77.3+ MB
```

## Finding the correlation of the rossmann data

```
In [22]:  plt.figure(figsize = (20,16))
          sns.heatmap(rossmann.corr(), annot = True, cmap="YlGnBu")
          plt.show()
```



## Exploring Store data

In [23]:
```
#Getting the first 5 rows
store.head()
```

Out[23]:

| | Store | StoreType | Assortment | CompetitionDistance | CompetitionOpenSinceMonth | CompetitionOpe |
|---|---|---|---|---|---|---|
| **0** | 1 | c | a | 1270.0 | 9.0 | |
| **1** | 2 | a | a | 570.0 | 11.0 | |
| **2** | 3 | a | a | 14130.0 | 12.0 | |
| **3** | 4 | c | c | 620.0 | 9.0 | |
| **4** | 5 | a | a | 29910.0 | 4.0 | |

In [24]:
```
# Getting the sum of null values
store.isnull().sum()
```

Out[24]:
```
Store                          0
StoreType                      0
Assortment                     0
CompetitionDistance            3
CompetitionOpenSinceMonth    354
CompetitionOpenSinceYear     354
Promo2                         0
Promo2SinceWeek              544
Promo2SinceYear              544
PromoInterval                544
dtype: int64
```

## Cleaning the store dataset with appropriate mean, median and mode

In [25]:
```
# Replacing the NAN values with median
store['CompetitionDistance'].fillna(store['CompetitionDistance'].median(), inpla

# Replacing NAN values with 0 in CompetitionOpenSinceMonth
store['CompetitionOpenSinceMonth'] = store['CompetitionOpenSinceMonth'].fillna(0

# Replacing NAN values with 0 in CompetitionOpenSinceYear
store['CompetitionOpenSinceYear'] = store['CompetitionOpenSinceYear'].fillna(0)

# Replacing NAN values with 0 in Promo2SinceWeek
store['Promo2SinceWeek'] = store['Promo2SinceWeek'].fillna(0)

# Replacing NAN values with 0 in Promo2SinceYear
store['Promo2SinceYear'] = store['Promo2SinceYear'].fillna(0)

# Replacing NAN values with 0 in PromoInterval
store['PromoInterval'].fillna(store['PromoInterval'].mode().values[0], inplace =
```

## Now checking NAN values

In [26]:
```
store.isnull().sum()
```

Out[26]:
```
Store                        0
StoreType                    0
Assortment                   0
CompetitionDistance          0
CompetitionOpenSinceMonth    0
CompetitionOpenSinceYear     0
Promo2                       0
Promo2SinceWeek              0
Promo2SinceYear              0
PromoInterval                0
dtype: int64
```

In [27]:
```
store.duplicated().sum()
```

Out[27]:
```
0
```

## The data is free from null values and duplicated values

In [28]:
```
# Getting the info
store.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1115 entries, 0 to 1114
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Store                      1115 non-null   int64  
 1   StoreType                  1115 non-null   object 
 2   Assortment                 1115 non-null   object 
 3   CompetitionDistance        1115 non-null   float64
 4   CompetitionOpenSinceMonth  1115 non-null   float64
 5   CompetitionOpenSinceYear   1115 non-null   float64
 6   Promo2                     1115 non-null   int64  
 7   Promo2SinceWeek            1115 non-null   float64
 8   Promo2SinceYear            1115 non-null   float64
 9   PromoInterval              1115 non-null   object 
dtypes: float64(5), int64(2), object(3)
memory usage: 87.2+ KB
```

In [29]:
```
store.head()
```

Out[29]:

| | Store | StoreType | Assortment | CompetitionDistance | CompetitionOpenSinceMonth | CompetitionOpe |
|---|---|---|---|---|---|---|
| **0** | 1 | c | a | 1270.0 | 9.0 | |
| **1** | 2 | a | a | 570.0 | 11.0 | |
| **2** | 3 | a | a | 14130.0 | 12.0 | |
| **3** | 4 | c | c | 620.0 | 9.0 | |
| **4** | 5 | a | a | 29910.0 | 4.0 | |

## Drawing the correlation

In [ ]:

```
```

## Changing the datatype of all the other columns which should be in int datatype

In [30]:

```
# Changing the datatype of CompetitionDistance from float to int
store['CompetitionDistance'] = store['CompetitionDistance'].astype(int)
```

In [31]:

```
# Changing the datatype of CompetitionOpenSinceMonth from float to int
store['CompetitionOpenSinceMonth'] = store['CompetitionOpenSinceMonth'].astype(i
```

In [32]:

```
# Changing the datatype of Promo2SinceWeek from float to int
store['Promo2SinceWeek'] = store['Promo2SinceWeek'].astype(int)
```

In [33]:

```
# Changing the datatype of CompetitionOpenSinceYear from float to int
store['CompetitionOpenSinceYear'] = store['CompetitionOpenSinceYear'].astype(int
```

In [34]:

```
# Changing the datatype of Promo2SinceYear from float to int
store['Promo2SinceYear'] = store['Promo2SinceYear'].astype(int)
```

In [35]:

```
store.head()
```

Out[35]:

| | Store | StoreType | Assortment | CompetitionDistance | CompetitionOpenSinceMonth | CompetitionOpe |
|---|---|---|---|---|---|---|
| **0** | 1 | c | a | 1270 | 9 | |
| **1** | 2 | a | a | 570 | 11 | |
| **2** | 3 | a | a | 14130 | 12 | |
| **3** | 4 | c | c | 620 | 9 | |
| **4** | 5 | a | a | 29910 | 4 | |

In [36]:

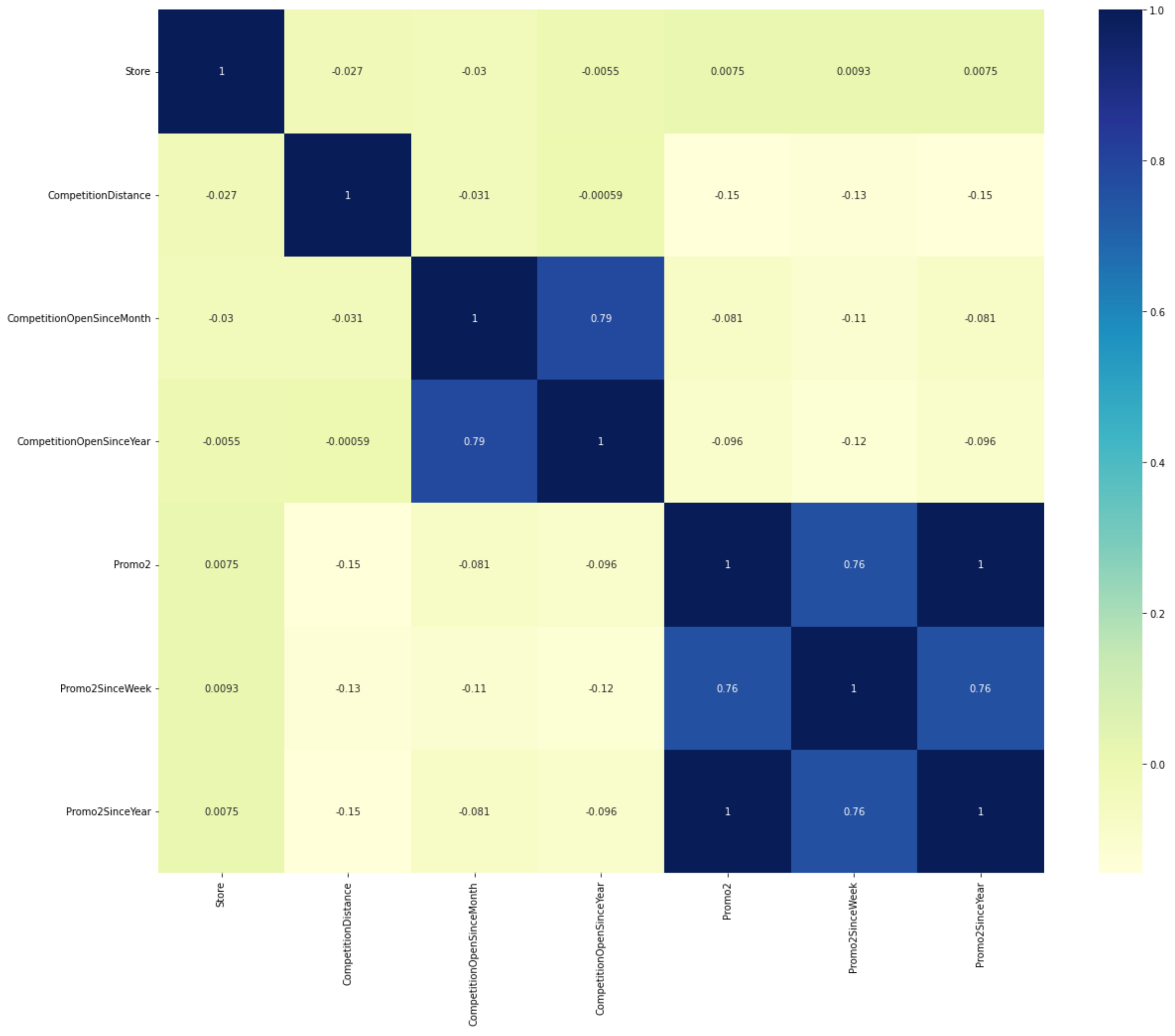
```
store.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1115 entries, 0 to 1114
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype 
---  ------                     --------------  ----- 
 0   Store                      1115 non-null   int64 
 1   StoreType                  1115 non-null   object
 2   Assortment                 1115 non-null   object
 3   CompetitionDistance        1115 non-null   int32 
 4   CompetitionOpenSinceMonth  1115 non-null   int32 
 5   CompetitionOpenSinceYear   1115 non-null   int32 
 6   Promo2                     1115 non-null   int64 
 7   Promo2SinceWeek            1115 non-null   int32 
 8   Promo2SinceYear            1115 non-null   int32 
 9   PromoInterval              1115 non-null   object
dtypes: int32(5), int64(2), object(3)
memory usage: 65.5+ KB
```

## Drawing Correlation

In [37]:

```
plt.figure(figsize = (20,16))
sns.heatmap(store.corr(), annot = True, cmap="YlGnBu")
plt.show()
```



## Merging both the dataset and storing it in df

In [43]:
```
df = pd.merge(rossmann, store, on='Store',how='left')
df.head()
```

Out[43]:

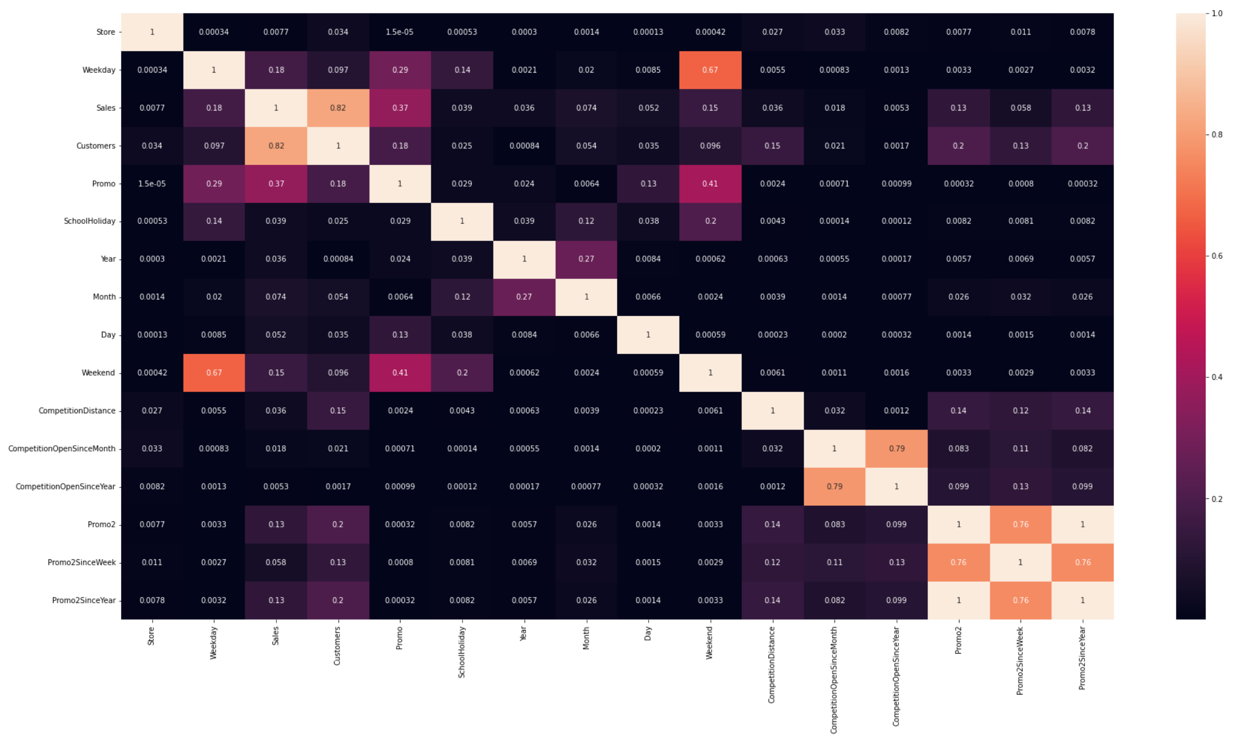
| | Store | Weekday | Sales | Customers | Promo | StateHoliday | SchoolHoliday | Year | Month | Day | We |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 5 | 5263 | 555 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **1** | 2 | 5 | 6064 | 625 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **2** | 3 | 5 | 8314 | 821 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **3** | 4 | 5 | 13995 | 1498 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **4** | 5 | 5 | 4822 | 559 | 1 | 0 | 1 | 2015 | 7 | 31 | |

In [44]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 844338 entries, 0 to 844337
Data columns (total 20 columns):
 #   Column                     Non-Null Count   Dtype 
---  ------                     --------------   ----- 
 0   Store                      844338 non-null  int64 
 1   Weekday                    844338 non-null  int64 
 2   Sales                      844338 non-null  int64 
 3   Customers                  844338 non-null  int64 
 4   Promo                      844338 non-null  int64 
 5   StateHoliday               844338 non-null  object
 6   SchoolHoliday              844338 non-null  int64 
 7   Year                       844338 non-null  int64 
 8   Month                      844338 non-null  int64 
 9   Day                        844338 non-null  int64 
 10  Weekend                    844338 non-null  int64 
 11  StoreType                  844338 non-null  object
 12  Assortment                 844338 non-null  object
 13  CompetitionDistance        844338 non-null  int32 
 14  CompetitionOpenSinceMonth  844338 non-null  int32 
 15  CompetitionOpenSinceYear   844338 non-null  int32 
 16  Promo2                     844338 non-null  int64 
 17  Promo2SinceWeek            844338 non-null  int32 
 18  Promo2SinceYear            844338 non-null  int32 
 19  PromoInterval              844338 non-null  object
dtypes: int32(5), int64(11), object(4)
memory usage: 119.2+ MB
```

In [45]:
```
plt.figure(figsize = (30,15))
sns.heatmap(abs(df.corr()), annot = True)
```

Out[45]:
```
<AxesSubplot:>
```



## Data Visualization on df dataset

### 1. Year on year growth

In [46]:
```
# Getting first 5 rows
df.head()
```

Out[46]:

| | Store | Weekday | Sales | Customers | Promo | StateHoliday | SchoolHoliday | Year | Month | Day | We |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 5 | 5263 | 555 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **1** | 2 | 5 | 6064 | 625 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **2** | 3 | 5 | 8314 | 821 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **3** | 4 | 5 | 13995 | 1498 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **4** | 5 | 5 | 4822 | 559 | 1 | 0 | 1 | 2015 | 7 | 31 | |

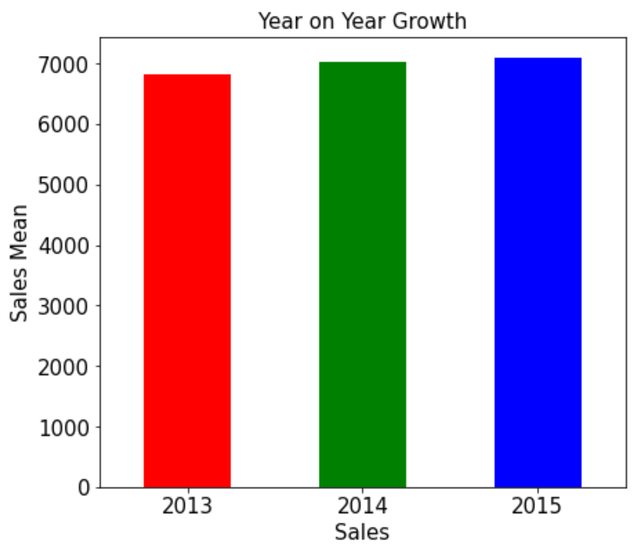
In [47]:
```
# Grouping the year with sales for visualization
year_on_sale = df.groupby(['Year'])['Sales'].mean()
year_on_sale
```

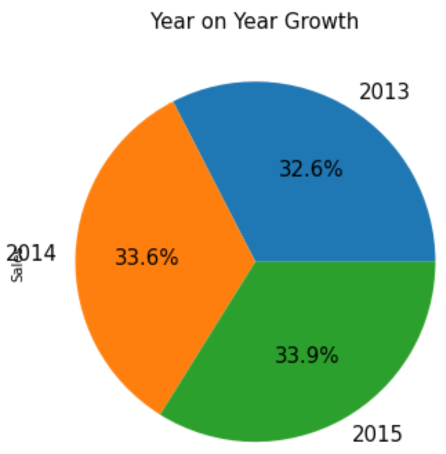
Out[47]:
```
Year
2013    6814.775168
2014    7026.128505
2015    7088.235123
Name: Sales, dtype: float64
```

In [48]:
```
# Visualizong using bar graph
plt.figure(figsize = (15,6))
plt.subplot(1,2,1)
year_on_sale.plot(kind = 'bar', color = ['r', 'g', 'b'], fontsize = 15)
plt.xticks(rotation = 360)
plt.title("Year on Year Growth", fontsize = 15)
plt.xlabel("Sales", fontsize = 15)
plt.ylabel("Sales Mean",  fontsize = 15)

# Visualizing with pie plot
plt.subplot(1,2,2)
year_on_sale.plot(kind = 'pie', fontsize = 15, autopct = '%1.1f%%')
plt.title("Year on Year Growth", fontsize = 15)
```

Out[48]:
```
Text(0.5, 1.0, 'Year on Year Growth')
```





## Observation

1. The year on year sales is increasing which is a good point for the company.
2. the YOY growth is positive yet it is very minimal.

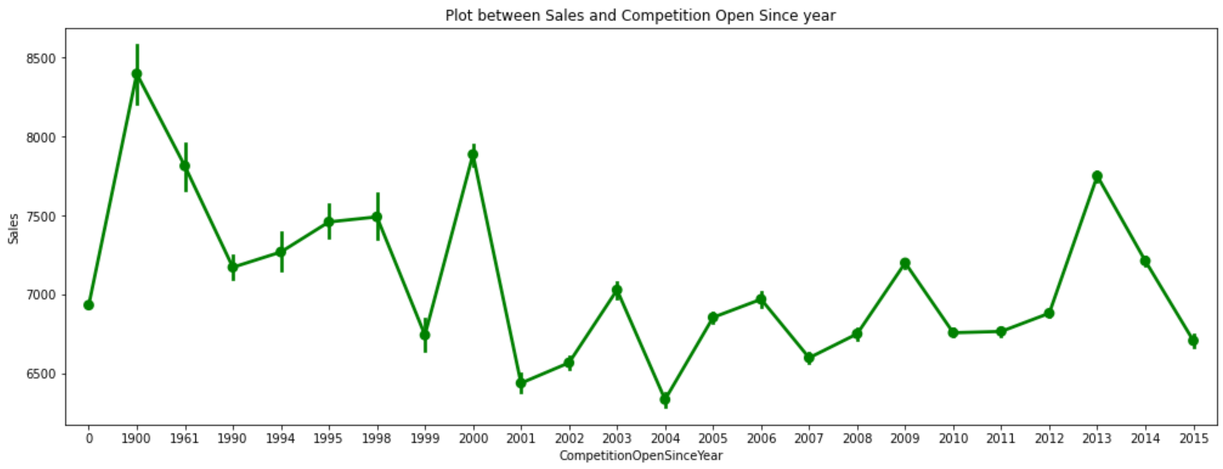
## 2. Competition Open Since Year

In [49]:
```
df.head()
```

Out[49]:

| | Store | Weekday | Sales | Customers | Promo | StateHoliday | SchoolHoliday | Year | Month | Day | We |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 5 | 5263 | 555 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **1** | 2 | 5 | 6064 | 625 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **2** | 3 | 5 | 8314 | 821 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **3** | 4 | 5 | 13995 | 1498 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **4** | 5 | 5 | 4822 | 559 | 1 | 0 | 1 | 2015 | 7 | 31 | |

In [50]:
```
# Visulaizing using point plot
plt.figure(figsize=(17,6))
sns.pointplot(data = df, x= df['CompetitionOpenSinceYear'], y= df['Sales'], colo
plt.title('Plot between Sales and Competition Open Since year')
plt.show()
```



## Observation

1. From the Plot we can tell that Sales are high during the year 1900, as there are very few store were operated of Rossmann so there is less competition and sales are high.
2. As year pass on number of stores increased that means competition also increased and this leads to decline in the sales.

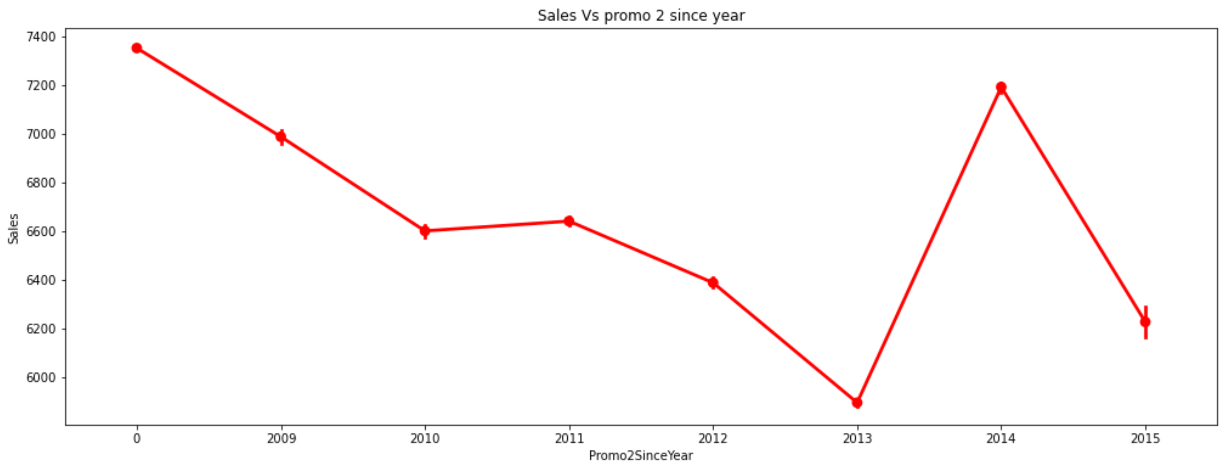
## 3. Promo 2 Since Year

In [51]:
```
df.head()
```

Out[51]:

| | Store | Weekday | Sales | Customers | Promo | StateHoliday | SchoolHoliday | Year | Month | Day | We |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 5 | 5263 | 555 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **1** | 2 | 5 | 6064 | 625 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **2** | 3 | 5 | 8314 | 821 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **3** | 4 | 5 | 13995 | 1498 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **4** | 5 | 5 | 4822 | 559 | 1 | 0 | 1 | 2015 | 7 | 31 | |

In [52]:
```
# Visualizing using point plot
plt.figure(figsize=(17,6))
sns.pointplot(data = df, x = df['Promo2SinceYear'], y = df['Sales'], color = 'r'
plt.title("Sales Vs promo 2 since year")
plt.show()
```



## Observation

1. Since year shows that effect of sales of stores which continue their promotion. this data is available from year 2009 to 2015.
2. Promo2 has very good effect on sales but in year 2013 sales be minimum and also in year 2012 and 2015 sales are very low.

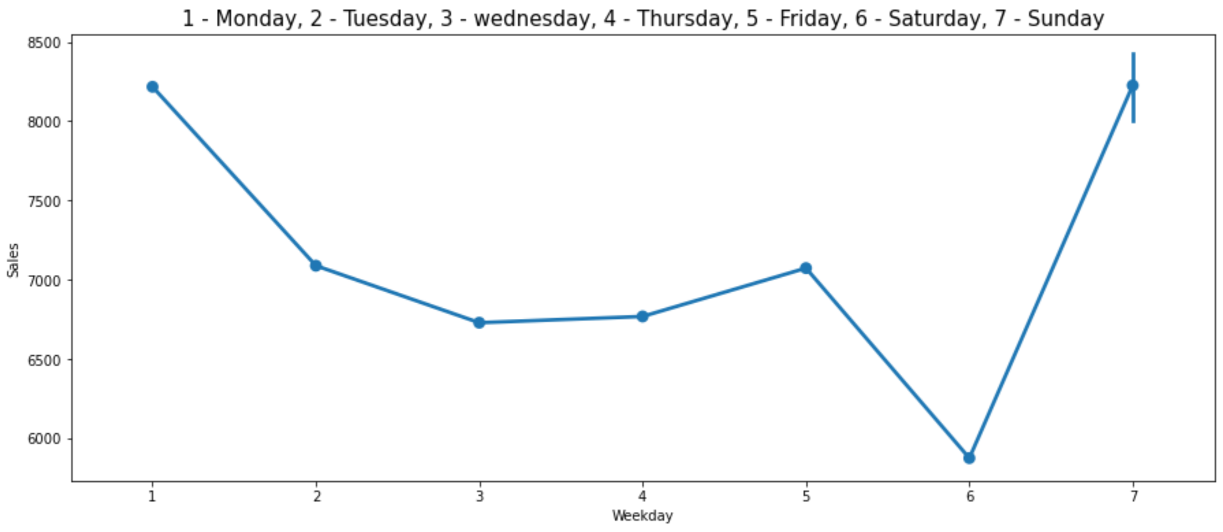
## 4. Weekday

In [53]:
```
df.head()
```

Out[53]:

| | Store | Weekday | Sales | Customers | Promo | StateHoliday | SchoolHoliday | Year | Month | Day | We |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 5 | 5263 | 555 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **1** | 2 | 5 | 6064 | 625 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **2** | 3 | 5 | 8314 | 821 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **3** | 4 | 5 | 13995 | 1498 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **4** | 5 | 5 | 4822 | 559 | 1 | 0 | 1 | 2015 | 7 | 31 | |

In [54]:
```
# Visualizing using pointplot
plt.figure(figsize=(15,6))
sns.pointplot(data = df, x = df['Weekday'], y= df['Sales'])
plt.title("1 - Monday, 2 - Tuesday, 3 - wednesday, 4 - Thursday, 5 - Friday, 6 -
plt.show()
```



## Observation

1. From monday till Saturday, the sale drop
2. On sunday the sale drastically goes up because its sunday.

## 5. Monthly sales

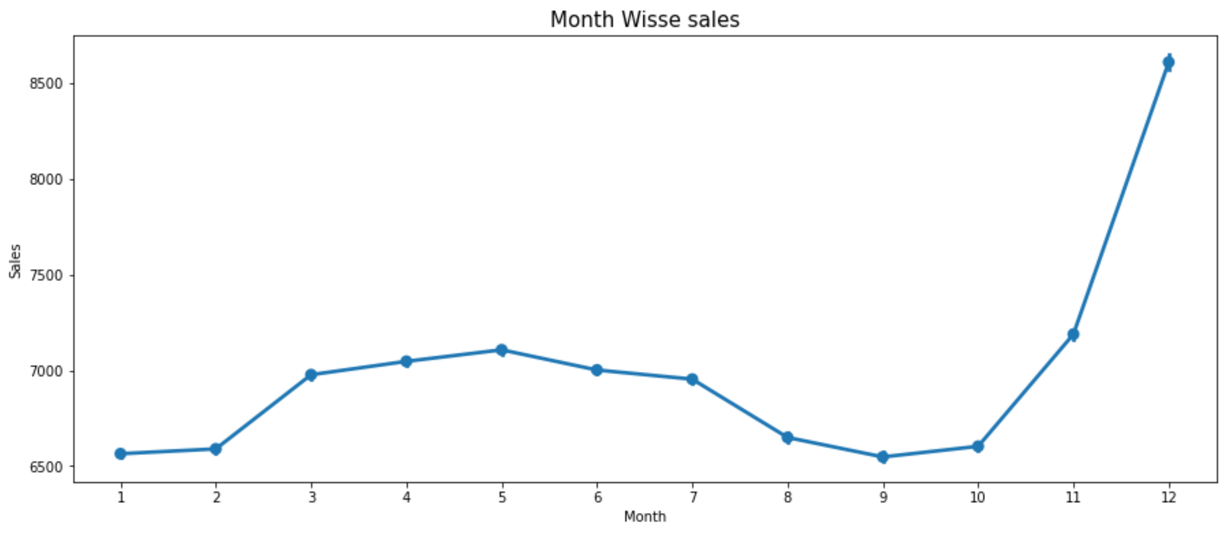
In [55]:
```
df.head()
```

Out[55]:

| | Store | Weekday | Sales | Customers | Promo | StateHoliday | SchoolHoliday | Year | Month | Day | We |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 5 | 5263 | 555 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **1** | 2 | 5 | 6064 | 625 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **2** | 3 | 5 | 8314 | 821 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **3** | 4 | 5 | 13995 | 1498 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **4** | 5 | 5 | 4822 | 559 | 1 | 0 | 1 | 2015 | 7 | 31 | |

In [56]:

```python
# Visualizing using pointplot
plt.figure(figsize = (15,6))
sns.pointplot(data = df, x = df['Month'], y = df['Sales'])
plt.title("Month Wisse sales", fontsize = 15)
plt.show()
```



## Observatoin

1. As we can see from the pointplot, the sale has a minimal of growth from january till may
2. after may it has a little dip in the salestill octber.
3. after october till november it has good growth in sale.
4. In december, the growth is unexceptionally good. This may be because of the holiday season including christmas and new years eve.

## 8. Analyzing December month sales

In [57]:

```python
# Getting the First 5 rows
df.head()
```

Out[57]:

| | Store | Weekday | Sales | Customers | Promo | StateHoliday | SchoolHoliday | Year | Month | Day | We |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 5 | 5263 | 555 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **1** | 2 | 5 | 6064 | 625 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **2** | 3 | 5 | 8314 | 821 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **3** | 4 | 5 | 13995 | 1498 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **4** | 5 | 5 | 4822 | 559 | 1 | 0 | 1 | 2015 | 7 | 31 | |

## Extracting only december month sales and storing it in december

In [58]:
```
december = df[df['Month'] == 12]
december.head()
```

Out[58]:

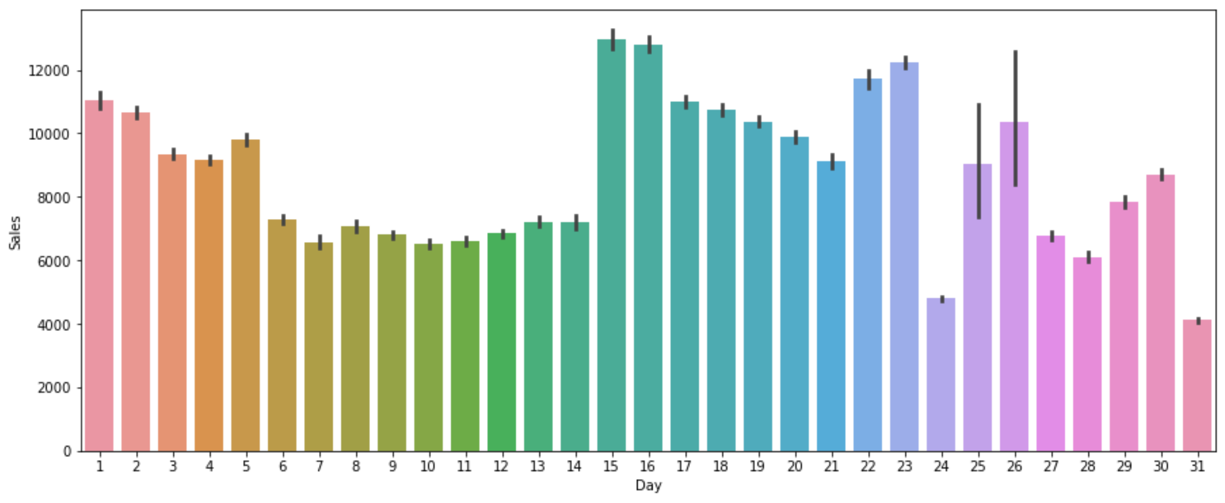
| | Store | Weekday | Sales | Customers | Promo | StateHoliday | SchoolHoliday | Year | Month | Da |
|---|---|---|---|---|---|---|---|---|---|---|
| **196029** | 1 | 3 | 2605 | 327 | 0 | 0 | 1 | 2014 | 12 | 3 |
| **196030** | 2 | 3 | 2269 | 252 | 0 | 0 | 1 | 2014 | 12 | 3 |
| **196031** | 3 | 3 | 3804 | 408 | 0 | 0 | 1 | 2014 | 12 | 3 |
| **196032** | 4 | 3 | 10152 | 1311 | 0 | 0 | 1 | 2014 | 12 | 3 |
| **196033** | 5 | 3 | 1830 | 217 | 0 | 0 | 1 | 2014 | 12 | 3 |

In [59]:
```
# Getting the value count
december.groupby(['Day'])['Sales'].mean()
```

Out[59]:
```
Day
1      11026.264706
2      10642.079024
3       9352.908780
4       9150.200976
5       9804.197561
6       7273.598049
7       6567.294430
8       7083.043022
9       6798.685366
10      6520.997561
11      6595.880000
12      6839.711707
13      7215.386829
14      7212.126437
15     12969.971639
16     12802.247805
17     10993.843415
18     10730.676098
19     10370.127805
20      9898.003415
21      9128.383538
22     11704.693666
23     12225.453659
24      4802.694634
25      9029.424242
26     10364.078947
27      6770.175610
28      6102.494728
29      7831.228125
30      8711.160000
31      4112.417073
Name: Sales, dtype: float64
```

In [60]:
```
plt.figure(figsize = (15,6))
sns.barplot(data = december, x = december['Day'], y = december["Sales"])
```

Out[60]:
```
<AxesSubplot:xlabel='Day', ylabel='Sales'>
```



## Observation

1. It looks like there is different kind of sales happening over there.
2. 15th and 16th has the highest of sales.
3. Third week looks like it is busy, sales are much higher than other weeks

## 9. Weekend sales

In [61]:
```
df.sample(10)
```

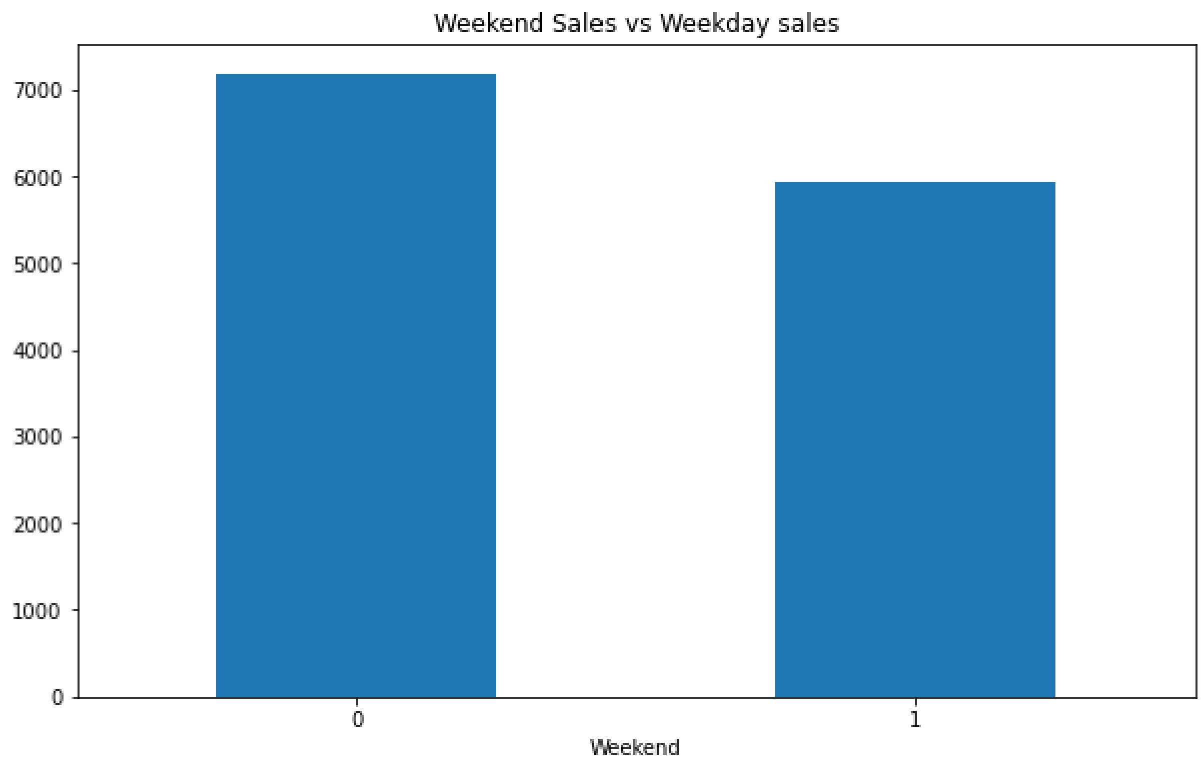
Out[61]:

| | Store | Weekday | Sales | Customers | Promo | StateHoliday | SchoolHoliday | Year | Month | Day |
|---|---|---|---|---|---|---|---|---|---|---|
| **841711** | 724 | 5 | 5621 | 665 | 0 | 0 | 1 | 2013 | 1 | 4 |
| **322409** | 393 | 3 | 3912 | 444 | 0 | 0 | 1 | 2014 | 7 | 23 |
| **256414** | 1095 | 3 | 4386 | 645 | 0 | 0 | 1 | 2014 | 10 | 15 |
| **149704** | 284 | 4 | 5432 | 458 | 1 | 0 | 1 | 2015 | 2 | 19 |
| **62025** | 38 | 3 | 5068 | 473 | 0 | 0 | 0 | 2015 | 5 | 27 |
| **737196** | 820 | 5 | 11952 | 1097 | 1 | 0 | 0 | 2013 | 4 | 26 |
| **796533** | 1054 | 4 | 5794 | 602 | 1 | 0 | 0 | 2013 | 2 | 21 |
| **476404** | 944 | 1 | 10311 | 1349 | 1 | 0 | 0 | 2014 | 2 | 3 |
| **118842** | 703 | 2 | 2937 | 314 | 0 | 0 | 0 | 2015 | 3 | 24 |
| **27242** | 398 | 5 | 6321 | 646 | 1 | 0 | 0 | 2015 | 7 | 3 |

In [62]:
```
df.groupby(['Weekend'])['Sales'].mean()
```

Out[62]:
```
Weekend
0    7172.903208
1    5932.264337
Name: Sales, dtype: float64
```

In [63]:
```
# Visualizing using barplot
plt.figure(figsize = (10,6))
df.groupby(['Weekend'])['Sales'].mean().plot(kind = 'bar')
plt.xticks(rotation = 360)
plt.title("Weekend Sales vs Weekday sales")
plt.show()
```



### Observation

1. As per graph we can see that the sales happend in weekdays are more than weekend.
2. Still weekday consists of 5 days i.e., from monday to friday, and weekend consists of only 2 days, i.e., Saturday and sunday. But still the sales are almost comparable.
3. Which shows that weekend sales are higher.

## 10. Day wise sales

In [64]:
```
df.head()
```

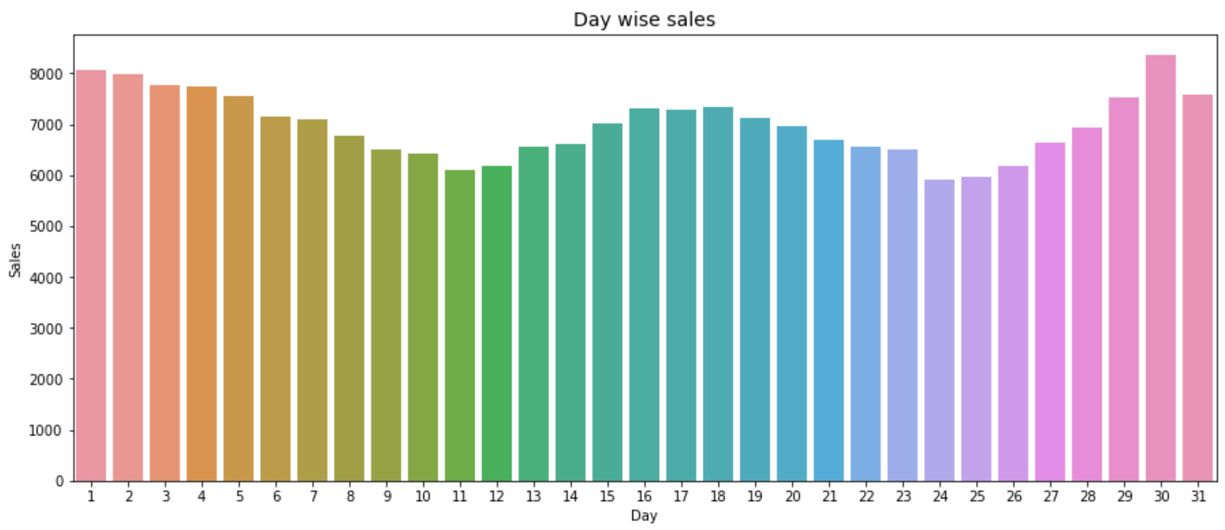
Out[64]:

| | Store | Weekday | Sales | Customers | Promo | StateHoliday | SchoolHoliday | Year | Month | Day | We |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 5 | 5263 | 555 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **1** | 2 | 5 | 6064 | 625 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **2** | 3 | 5 | 8314 | 821 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **3** | 4 | 5 | 13995 | 1498 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **4** | 5 | 5 | 4822 | 559 | 1 | 0 | 1 | 2015 | 7 | 31 | |

In [65]:
```
# Grouping the days from different month with mean of sales
day_wise_sales = df.groupby(['Day'])['Sales'].mean()
day_wise_sales
```

Out[65]:
```
Day
1     8054.505835
2     7987.998803
3     7765.916826
4     7746.632622
5     7556.054806
6     7149.914351
7     7101.614663
8     6785.606424
9     6499.517013
10    6429.867986
11    6088.286098
12    6186.692977
13    6570.339941
14    6606.648700
15    7018.797807
16    7314.330149
17    7284.416418
18    7340.772490
19    7115.279322
20    6955.004553
21    6693.696159
22    6544.923929
23    6498.481514
24    5916.886849
25    5968.280641
26    6190.007567
27    6636.996208
28    6943.514789
29    7514.074032
30    8355.098655
31    7577.710796
Name: Sales, dtype: float64
```

In [66]:
```
plt.figure(figsize = (15,6))
sns.barplot(data = df, x = day_wise_sales.keys(), y = day_wise_sales)
plt.title("Day wise sales", Fontsize = 14)
plt.show()
```



## Observation

1. There is no particular trend as par say

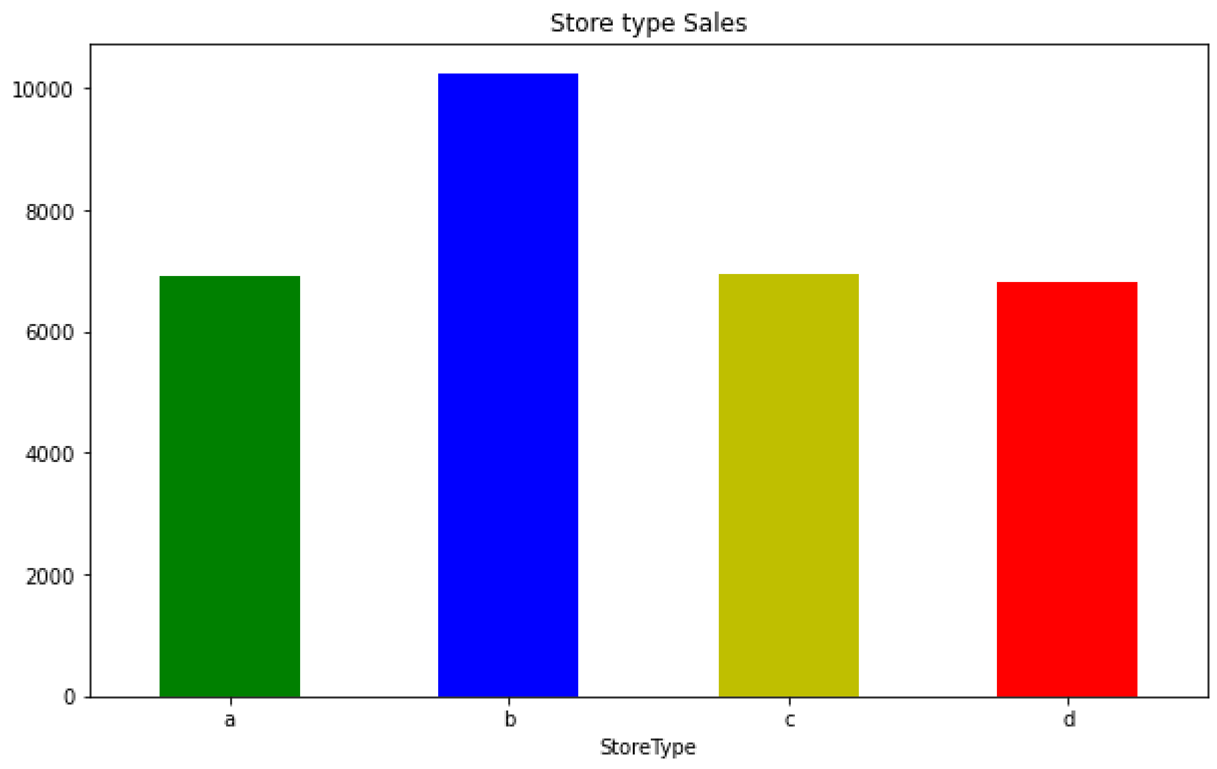
2. But, we can see that at the starting of the month and at the end of the month the sales are pretty high.
3. Sales are high at the starting of the month can be related as the salary of most of the employee is credited at the end of the month.
4. The sales are high at the end of the month, could have been highly inspired by the december month sales as the december month has the highest sales of all the months. As we have taken the mean of the day sales from every month.

## Store Type Sales

In [67]:
```
store_type = df.groupby(['StoreType'])['Sales'].mean()
store_type
```

Out[67]:
```
StoreType
a     6925.697986
b    10233.380141
c     6933.126425
d     6822.300064
Name: Sales, dtype: float64
```

In [68]:
```
plt.figure(figsize = (10,6))
store_type.plot(kind = 'bar', color = ['g','b', 'y', 'r'])
plt.xticks(rotation = 360)
plt.title("Store type Sales")
plt.show()
```



## Observation

1. It is very clear from the graph that store type B has the highest number of sales
2. other than store B, all other store ahs almost equal sales

```
In [69]:  df.head()
```

Out[69]:

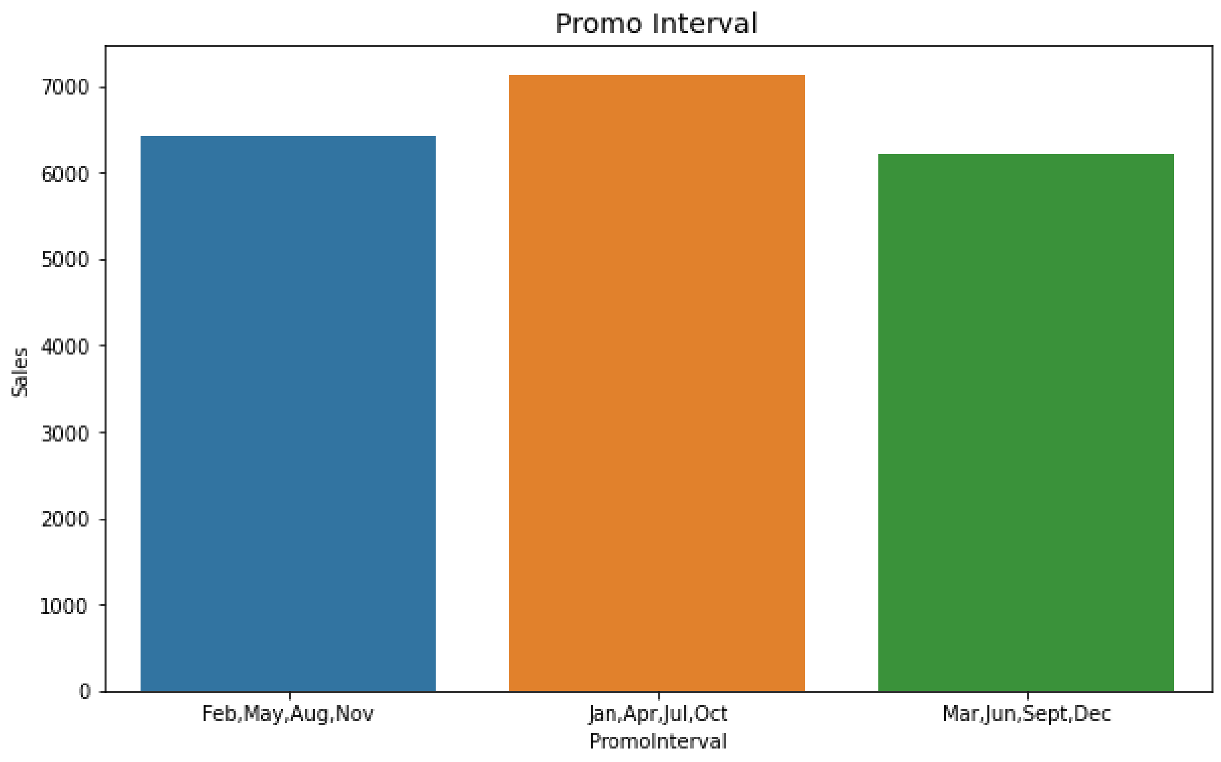
| | Store | Weekday | Sales | Customers | Promo | StateHoliday | SchoolHoliday | Year | Month | Day | We |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 5 | 5263 | 555 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **1** | 2 | 5 | 6064 | 625 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **2** | 3 | 5 | 8314 | 821 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **3** | 4 | 5 | 13995 | 1498 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **4** | 5 | 5 | 4822 | 559 | 1 | 0 | 1 | 2015 | 7 | 31 | |

```
In [70]:  df.groupby(['PromoInterval'])['Sales'].mean()
```

```
Out[70]: PromoInterval
         Feb,May,Aug,Nov      6427.367069
         Jan,Apr,Jul,Oct      7123.437381
         Mar,Jun,Sept,Dec     6215.888185
         Name: Sales, dtype: float64
```

```
In [71]:  # Visualizing using Barplot
          plt.figure(figsize = (10,6))
          sns.barplot(data = df, x = df.groupby(['PromoInterval'])['Sales'].mean().keys(),
                      y =df.groupby(['PromoInterval'])['Sales'].mean())
          plt.title("Promo Interval", fontsize = 14)
```

```
Out[71]: Text(0.5, 1.0, 'Promo Interval')
```



## Observation

1. It is very clear from the Graph that promo on JAN, APR, JUL and OCT has high sales
2. Other months has almost equal sales

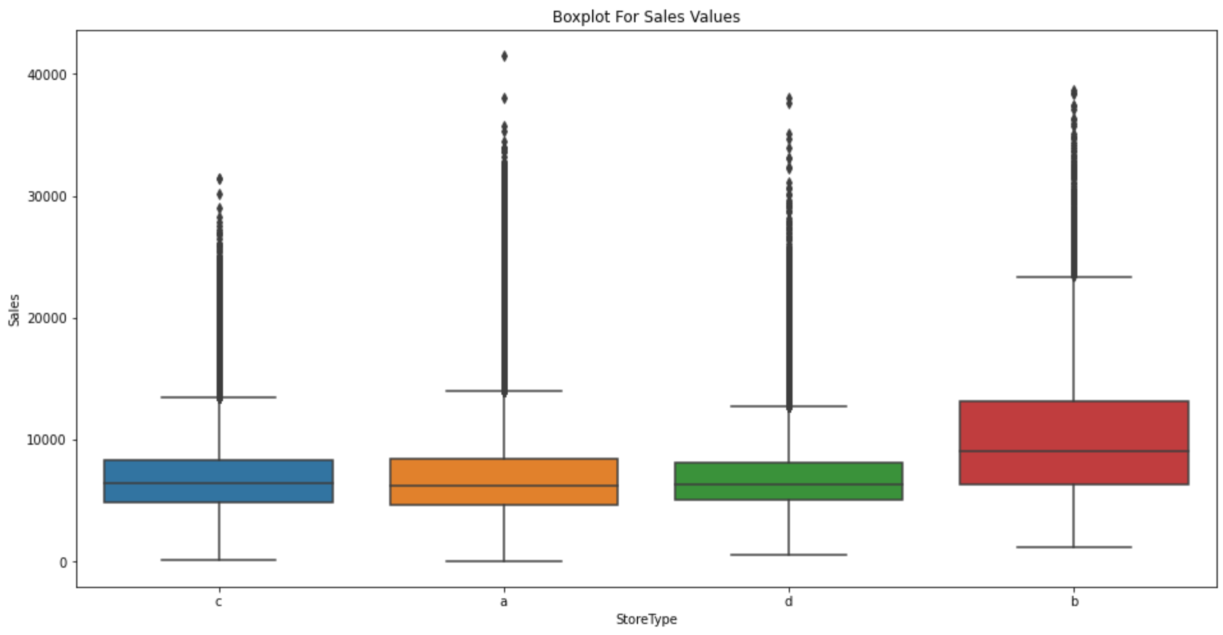
## Boxplots

In [72]:
```
df.head()
```

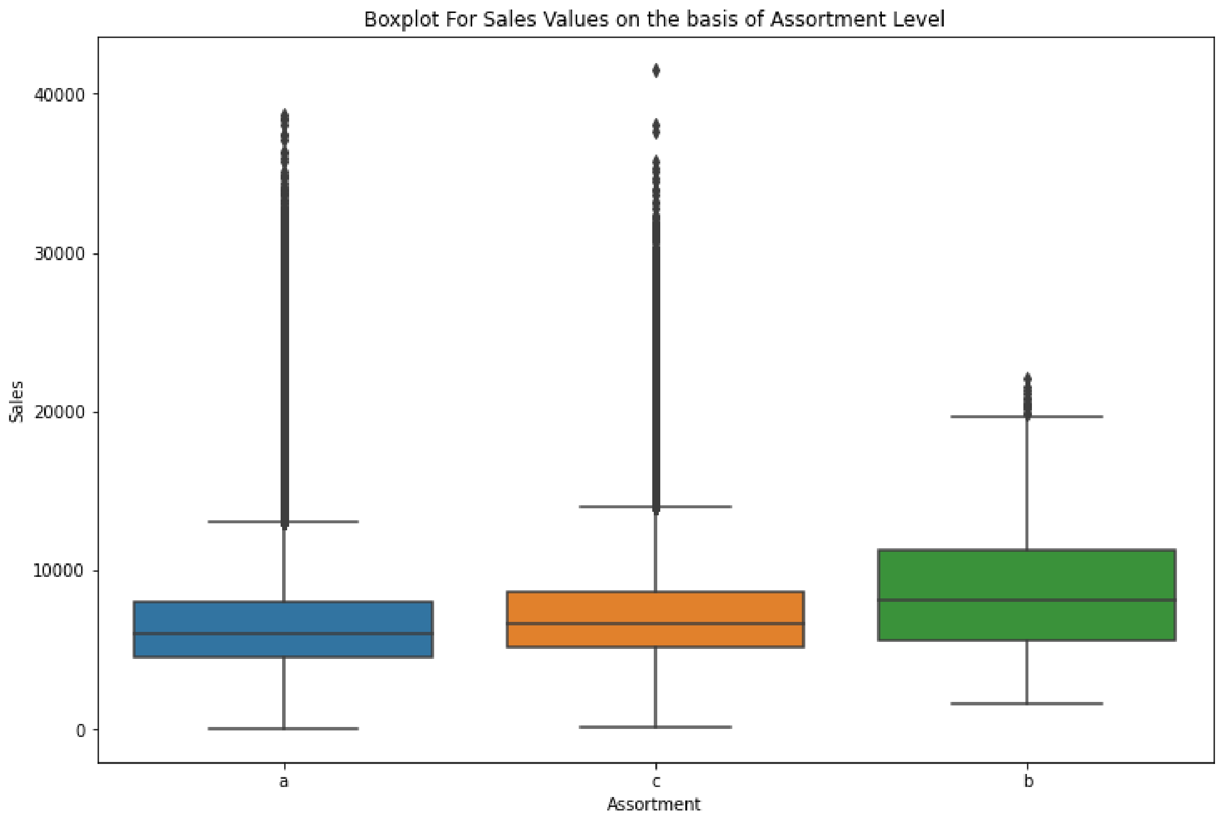
Out[72]:

| | Store | Weekday | Sales | Customers | Promo | StateHoliday | SchoolHoliday | Year | Month | Day | We |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 5 | 5263 | 555 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **1** | 2 | 5 | 6064 | 625 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **2** | 3 | 5 | 8314 | 821 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **3** | 4 | 5 | 13995 | 1498 | 1 | 0 | 1 | 2015 | 7 | 31 | |
| **4** | 5 | 5 | 4822 | 559 | 1 | 0 | 1 | 2015 | 7 | 31 | |

In [73]:
```
# Visualizing the boxplot for stroretype with sales
plt.figure(figsize = (16,8))
sns.boxplot(data = df, x = df['StoreType'], y = df['Sales'])
plt.title('Boxplot For Sales Values')
plt.show()
```



In [74]:
```
# Visualizing the boxplot for assortment
plt.figure(figsize=(12, 8))
sns.boxplot(x="Assortment", y="Sales", data=df)
plt.title('Boxplot For Sales Values on the basis of Assortment Level')
plt.show()
```



### Observation

There are a lot of outliers but we cannot remove them as they belong to sales and there can be price of any articles which may cost much higher than normam range.

## Observation from Exploratory Data Analysis

1) From plot sales and competition Open Since Month shows sales go increasing from November and highest in month December.
2) From plot Sales and day of week, Sales highest on Monday and start declining from Tuesday to Saturday and on Sunday Sales almost near to Zero.
3) Plot between Promotion and Sales shows that promotion helps in increasing Sales.
4) Type of Store plays an important role in opening pattern of stores.
5) All Type ‘b’ stores never closed except for refurbishment or other reason.
6) All Type ‘b’ stores have comparatively higher sales and it mostly constant with peaks appears on weekends.
7) We can observe that most of the stores remain closed during State Holidays. But it is interesting to note that the number of stores opened during School Holidays were more than that were opened during State Holidays.

## Data visulaization is done and we head towards model building

## Multicollinearity

Multicollinearity occurs when two or more independent variables(also known as predictor) are highly correlated with one another in a regression model.

This means that an independent variable can be predicted from another independent variable in a regression model. For Example, height, and weight, student consumption and father income, age and experience, mileage and price of a car, etc.

Let us take a simple example from our everyday life to explain this. Assume that we want to fit a regression model using the independent features such as pocket money and father income, to find the dependent variable, Student consumption here we cannot find an exact or individual effect of all the independent variables on the dependent variable or response since here both independent variables are highly correlated means as father income increases pocket money also increases and if father income decreases pocket money also decreases.

This is the multicollinearity problem!

## The problem with having multicollinearity

Since in a regression model our research objective is to find out how each predictor is impacting the target variable individually which is also an assumption of a method namely Ordinary Least Squares through which we can find the parameters of a regression model. So finally to fulfill our research objective for a regression model we have to fix the problem of multicollinearity which is finally important for our prediction also.

Let say we have the following linear equation

Y=a0+a1*X1+a2*X2

Here X1 and X2 are the independent variables. The mathematical significance of a1 is that if we shift our X1 variable by 1 unit then our Y shifts by a1 units keeping X2 and other things constant. Similarly, for a2 we have if we shift X2 by one unit means Y also shifts by one unit keeping X1 and other factors constant.

But for a situation where multicollinearity exists our independent variables are highly correlated, so if we change X1 then X2 also changes and we would not be able to see their Individual effect on Y which is our research objective for a regression model.

**This makes the effects of X1 on Y difficult to differentiate from the effects of X2 on Y.**

## Detecting Multicollinearity using VIF

VIF determines the strength of the correlation between the independent variables. It is predicted by taking a variable and regressing it against every other variable. " or VIF score of an independent variable represents how well the variable is explained by other independent variables. R^2 value is determined to find out how well an independent variable is described by the other independent variables. A high value of R^2 means that the variable is highly correlated with the other variables. This is captured by the VIF which is denoted below:

`VIF=1/(1-R^2)`

So, the closer the R^2 value to 1, the higher the value of VIF and the higher the multicollinearity with the particular independent variable.

1. VIF starts at 1(when R^2=0, VIF=1 – minimum value for VIF) and has no upper limit.
2. VIF = 1, no correlation between the independent variable and the other variables.

3. VIF exceeding 5 or 10 indicates high multicollinearity between this independent variable and the others.
4. Some researchers assume VIF > 5 as a serious issue for our model while some researchers assume VIF>10 as serious, it varies from person to person.

In [75]:

```python
# Defining a function calc_vif to calculate the VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor
def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[

    return(vif)
```

In [76]:

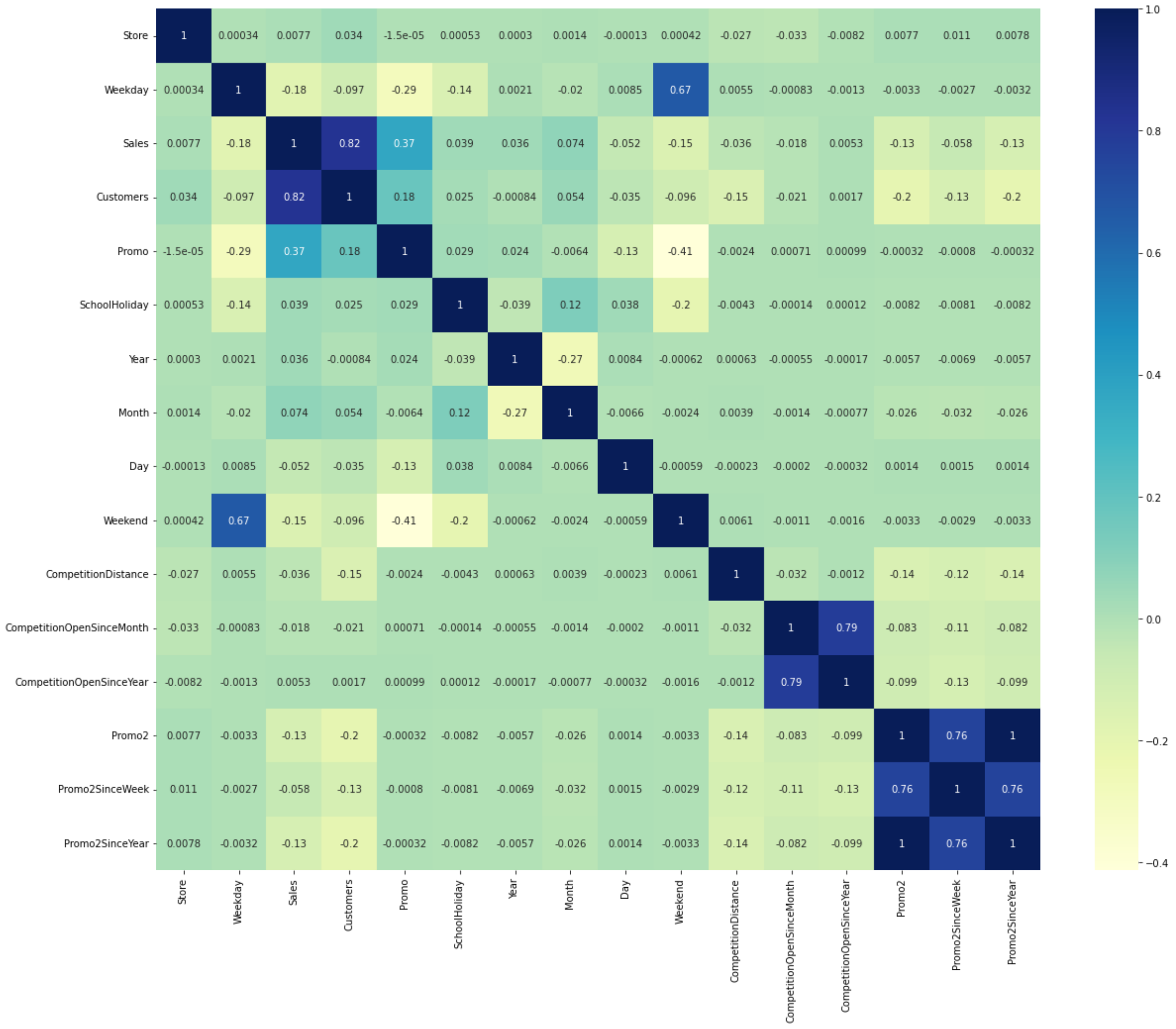
```python
# Calculating the VIF
calc_vif(df[[i for i in df.describe().columns if i not in ['Sales']]])
```

Out[76]:

| | variables | VIF |
|---|---|---|
| **0** | Store | 4.030699e+00 |
| **1** | Weekday | 9.381469e+00 |
| **2** | Customers | 5.217864e+00 |
| **3** | Promo | 2.288766e+00 |
| **4** | SchoolHoliday | 1.319086e+00 |
| **5** | Year | 2.828420e+01 |
| **6** | Month | 4.170063e+00 |
| **7** | Day | 4.417196e+00 |
| **8** | Weekend | 2.496299e+00 |
| **9** | CompetitionDistance | 1.584958e+00 |
| **10** | CompetitionOpenSinceMonth | 6.108136e+00 |
| **11** | CompetitionOpenSinceYear | 8.256236e+00 |
| **12** | Promo2 | 1.573450e+06 |
| **13** | Promo2SinceWeek | 3.988833e+00 |
| **14** | Promo2SinceYear | 1.572397e+06 |

In [77]:

```python
plt.figure(figsize = (20,16))
sns.heatmap(df.corr(), annot = True, cmap="YlGnBu")
plt.show()
```



## Multicollinearity of Year and Promo2SinceYear and Weekend is pretty high hence we decide to drop it.

In [78]:
```
# Calculating the VIF
calc_vif(df[[i for i in df.describe().columns if i not in ['Sales','Year','Promo2
```

Out[78]:

| | variables | VIF |
|---|---|---|
| **0** | Store | 3.612927 |
| **1** | Weekday | 4.306986 |
| **2** | Customers | 4.268165 |
| **3** | Promo | 1.910253 |
| **4** | SchoolHoliday | 1.272439 |
| **5** | Month | 3.747233 |
| **6** | Day | 3.744487 |
| **7** | CompetitionDistance | 1.480676 |
| **8** | CompetitionOpenSinceMonth | 6.091867 |
| **9** | CompetitionOpenSinceYear | 8.046782 |
| **10** | Promo2 | 4.674431 |
| **11** | Promo2SinceWeek | 3.749959 |

## Since all the column Variance Inflation Factor is less than 10, we are good to go.

In [79]:
```
# Dropping all the VIF columns more than 10
df = df.drop(['Year','Promo2SinceYear', 'Weekend'], axis = 1)
```

## One hot encoding for df

In [80]:
```
df = pd.get_dummies(df, columns = ['StoreType','Assortment','PromoInterval', 'Sta
df.sample(10)
```

Out[80]:

| | Store | Weekday | Sales | Customers | Promo | SchoolHoliday | Month | Day | CompetitionDistan |
|---|---|---|---|---|---|---|---|---|---|
| **390475** | 1108 | 3 | 6651 | 687 | 1 | 0 | 5 | 7 | 5 |
| **664782** | 408 | 1 | 6392 | 581 | 1 | 0 | 7 | 15 | 15 |
| **773090** | 1014 | 1 | 17945 | 1938 | 1 | 0 | 3 | 18 | 2 |
| **627713** | 86 | 4 | 3264 | 514 | 0 | 1 | 8 | 22 | 4 |
| **348413** | 951 | 1 | 6154 | 655 | 0 | 0 | 6 | 23 | 7 |
| **406485** | 237 | 3 | 5394 | 633 | 1 | 1 | 4 | 16 | 14 |
| **80025** | 44 | 3 | 5838 | 691 | 1 | 0 | 5 | 6 | 5 |
| **319451** | 198 | 6 | 638 | 83 | 0 | 0 | 7 | 26 | 2 |
| **243029** | 741 | 5 | 8132 | 992 | 0 | 1 | 10 | 31 | 119 |
| **592025** | 162 | 6 | 7330 | 573 | 0 | 0 | 9 | 28 | 53 |

In [81]:
```
# Getting the info of the new final dataset before the model is built.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 844338 entries, 0 to 844337
Data columns (total 28 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   Store                           844338 non-null  int64
 1   Weekday                         844338 non-null  int64
 2   Sales                           844338 non-null  int64
 3   Customers                       844338 non-null  int64
 4   Promo                           844338 non-null  int64
 5   SchoolHoliday                   844338 non-null  int64
 6   Month                           844338 non-null  int64
 7   Day                             844338 non-null  int64
 8   CompetitionDistance             844338 non-null  int32
 9   CompetitionOpenSinceMonth       844338 non-null  int32
 10  CompetitionOpenSinceYear        844338 non-null  int32
 11  Promo2                          844338 non-null  int64
 12  Promo2SinceWeek                 844338 non-null  int32
 13  StoreType_a                     844338 non-null  uint8
 14  StoreType_b                     844338 non-null  uint8
 15  StoreType_c                     844338 non-null  uint8
 16  StoreType_d                     844338 non-null  uint8
 17  Assortment_a                    844338 non-null  uint8
 18  Assortment_b                    844338 non-null  uint8
 19  Assortment_c                    844338 non-null  uint8
 20  PromoInterval_Feb,May,Aug,Nov   844338 non-null  uint8
 21  PromoInterval_Jan,Apr,Jul,Oct   844338 non-null  uint8
 22  PromoInterval_Mar,Jun,Sept,Dec  844338 non-null  uint8
 23  StateHoliday_0                  844338 non-null  uint8
 24  StateHoliday_0                  844338 non-null  uint8
 25  StateHoliday_a                  844338 non-null  uint8
 26  StateHoliday_b                  844338 non-null  uint8
 27  StateHoliday_c                  844338 non-null  uint8
dtypes: int32(4), int64(9), uint8(15)
memory usage: 121.6 MB
```

## Defining X and Y

In [82]:
```
# Create the data of independent variables
x = df.drop(['Sales'], axis = 1).values # independent variable, Predictor

# Create the data of dependent variable
y = df['Sales'].values # dependent variable, Y = mx + c, Y = b0+ b1x1 + b2x2 + ..
```

## Splitting the data into training and testing

In [83]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.3, random_
```

In [84]:
```
# Printing the shape of the train and test dataset
print(x_train.shape)
print(x_test.shape)
```

```
(591036, 27)
(253302, 27)
```

```
In [85]:  # Getting the coefficient before scaling
           x_train

Out[85]: array([[ 569,    4,  743, ...,    0,    0,    0],
                [1045,    6,  622, ...,    0,    0,    0],
                [ 112,    3,  694, ...,    0,    0,    0],
                ...,
                [ 927,    3,  679, ...,    0,    0,    0],
                [ 533,    1, 1124, ...,    0,    0,    0],
                [ 586,    3, 2463, ...,    0,    0,    0]], dtype=int64)
```

## Feature Scaling

#### What is Feature Scaling?

Feature scaling is a method used to normalize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data preprocessing step. Just to give you an example — if you have multiple independent variables like age, salary, and height; With their range as (18–100 Years), (25,000–75,000 Euros), and (1–2 Meters) respectively, feature scaling would help them all to be in the same range, for example- centered around 0 or in the range (0,1) depending on the scaling technique.

In order to visualize the above, let us take an example of the independent variables of alcohol and Malic Acid content in the wine dataset from the “Wine Dataset” that is deposited on the UCI machine learning repository. Below you can see the impact of the two most common scaling techniques (Normalization and Standardization) on the dataset.

### Normalization

Also known as min-max scaling or min-max normalization, it is the simplest method and consists of rescaling the range of features to scale the range in [0, 1]. The general formula for normalization is given as:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Here, max(x) and min(x) are the maximum and the minimum values of the feature respectively.

### Standardization

Feature standardization makes the values of each feature in the data have zero mean and unit variance. The general method of calculation is to determine the distribution mean and standard deviation for each feature and calculate the new data point by the following formula:

$$x' = \frac{x - \bar{x}}{\sigma}$$

Here, σ is the standard deviation of the feature vector, and x̄ is the average of the feature vector.

In [86]:
```
# Importing MinMaxScaler from sklearn
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

In [87]:
```
# Getting the coeficient after scaling
x_train
```

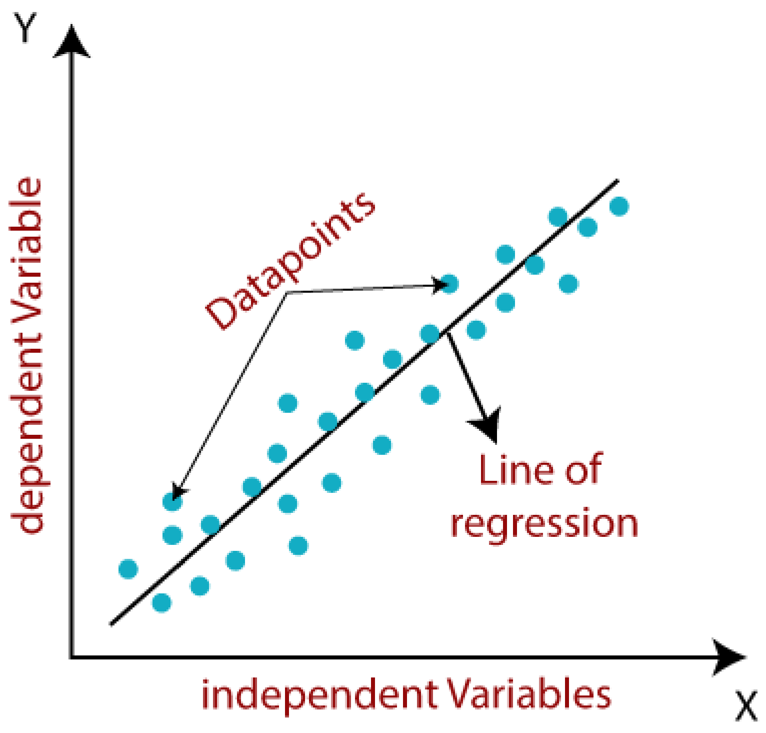
Out[87]:
```
array([[0.50987433, 0.5       , 0.1339774 , ..., 0.        , 0.        ,
        0.        ],
       [0.93716338, 0.83333333, 0.11192125, ..., 0.        , 0.        ,
        0.        ],
       [0.09964093, 0.33333333, 0.12504557, ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.83123878, 0.33333333, 0.12231134, ..., 0.        , 0.        ,
        0.        ],
       [0.47755835, 0.        , 0.2034269 , ..., 0.        , 0.        ,
        0.        ],
       [0.52513465, 0.33333333, 0.44750273, ..., 0.        , 0.        ,
        0.        ]])
```

## Linear Regression

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



Mathematically, we can represent a linear regression as:

```
In [88]:  # Importing the LinearRegression model from SCIKIT Learn
          from sklearn.linear_model import LinearRegression
          lr = LinearRegression()
          lr.fit(x_train, y_train)

Out[88]:  LinearRegression()
```

### Predicting the results

```
In [89]:  #Prediction the value
          y_pred = lr.predict(x_test)
          y_pred

Out[89]:  array([5258.75, 9842.  , 8773.  , ..., 5515.75, 4014.  , 6306.75])
```

## Checking for overfitting and underfitting via score

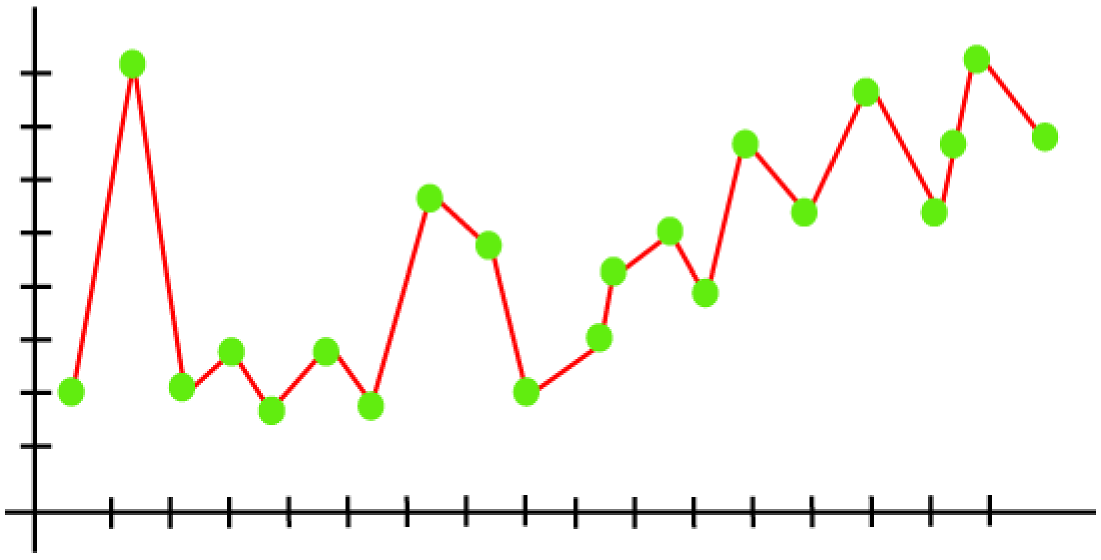
### Overfitting

Overfitting occurs when our machine learning model tries to cover all the data points or more than the required data points present in the given dataset. Because of this, the model starts caching noise and inaccurate values present in the dataset, and all these factors reduce the efficiency and accuracy of the model. The overfitted model has low bias and high variance

The chances of occurrence of overfitting increase as much we provide training to our model. It means the more we train our model, the more chances of occurring the overfitted model.

Overfitting is the main problem that occurs in supervised learning.

Example: The concept of the overfitting can be understood by the below graph of the linear regression output:



## How to avoid the Overfitting in Model

Both overfitting and underfitting cause the degraded performance of the machine learning model. But the main cause is overfitting, so there are some ways by which we can reduce the occurrence of overfitting in our model.

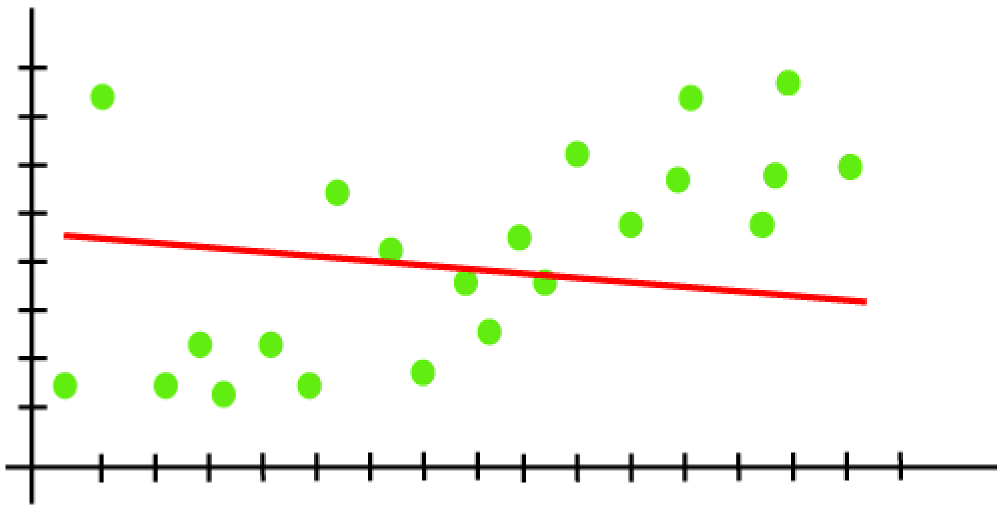
**Cross-Validation**
**Training with more data**
**Removing features**
**Early stopping the training**
**Regularization**
**Ensembling**

## Underfitting

Underfitting occurs when our machine learning model is not able to capture the underlying trend of the data. To avoid the overfitting in the model, the fed of training data can be stopped at an early stage, due to which the model may not learn enough from the training data. As a result, it may fail to find the best fit of the dominant trend in the data.

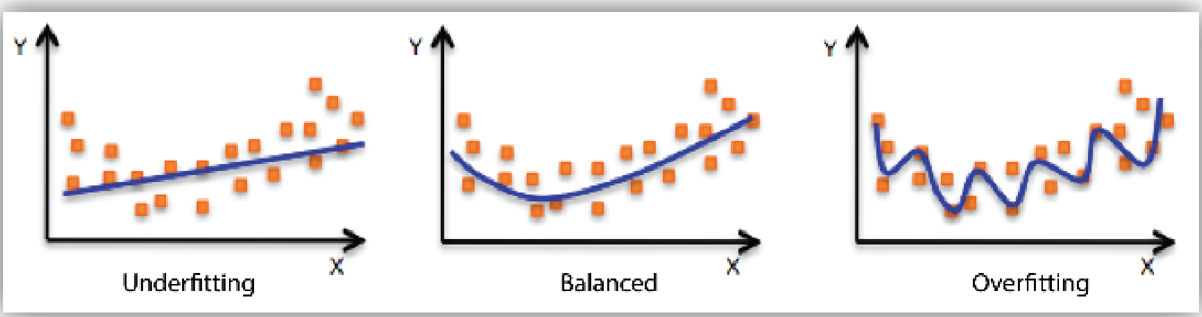
In the case of underfitting, the model is not able to learn enough from the training data, and hence it reduces the accuracy and produces unreliable predictions. An underfitted model has high bias and low variance. Example: We can understand the underfitting using below output of the linear regression model:

Example: We can understand the underfitting using below output of the linear regression model:



## How to avoid underfitting:

**By increasing the training time of the model.**
**By increasing the number of features.**



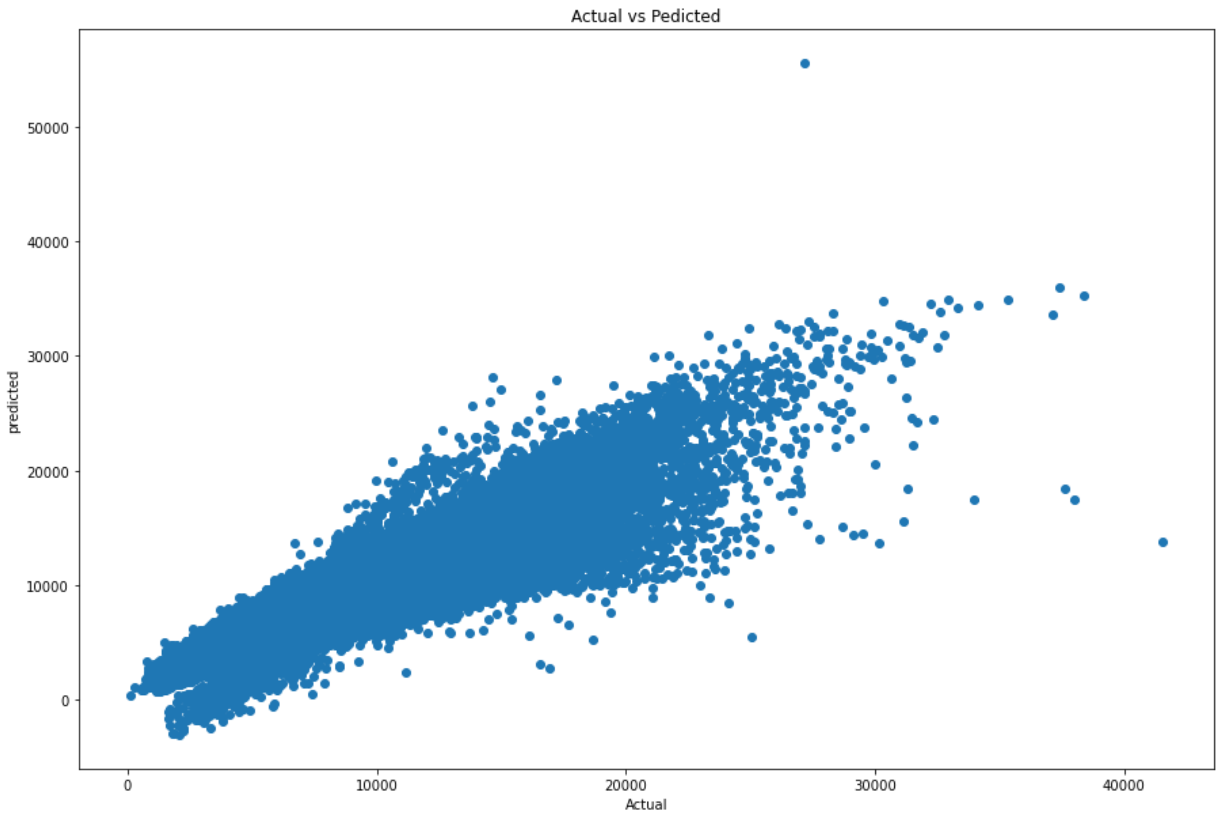
In [90]:
```
# Checking the score
(lr.score(x_train, y_train))*100, (lr.score(x_test, y_test))*100
```

Out[90]:
```
(82.5531565762266, 82.41720232184113)
```

## The model is good as it shows no overfitting and underfitting

## Visualizing through scatter plot

In [91]:
```
# Visualizing the actual and predicted values
plt.figure(figsize = (15,10))
plt.scatter(y_test, y_pred)
plt.title('Actual vs Pedicted')
plt.xlabel('Actual')
plt.ylabel('predicted')
plt.show()
```



```
In [92]:  # Getting the difference between actial and predicted value
          predicted_value = pd.DataFrame({'Actual Value': y_test, 'Predicted Value': y_pre
          predicted_value.sample(10)
```

Out[92]:

| | Actual Value | Predicted Value | Difference |
|---|---|---|---|
| **233749** | 7611 | 8414.25 | -803.25 |
| **112303** | 4677 | 4536.50 | 140.50 |
| **1789** | 7097 | 7901.75 | -804.75 |
| **170790** | 7507 | 8175.25 | -668.25 |
| **59214** | 5887 | 5707.50 | 179.50 |
| **49248** | 5743 | 6108.50 | -365.50 |
| **38290** | 6842 | 5929.50 | 912.50 |
| **85869** | 3437 | 4270.00 | -833.00 |
| **74536** | 12240 | 10237.25 | 2002.75 |
| **141905** | 7027 | 6780.50 | 246.50 |

## Evaluation Metrics

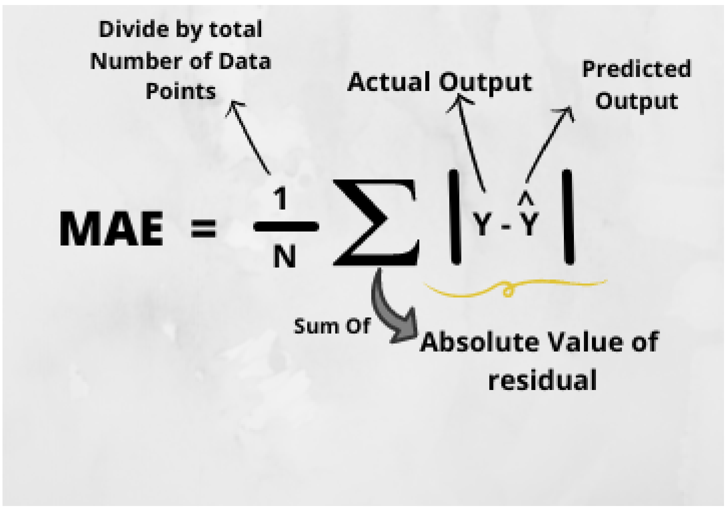
### 1) Mean Absolute Error(MAE)

MAE is a very simple metric which calculates the absolute difference between actual and predicted values.

To better understand, let’s take an example you have input data and output data and use Linear Regression, which draws a best-fit line.

Now you have to find the MAE of your model which is basically a mistake made by the model known as an error. Now find the difference between the actual value and predicted value that is an absolute error but we have to find the mean absolute of the complete dataset.

so, sum all the errors and divide them by a total number of observations And this is MAE. And we aim to get a minimum MAE because this is a loss.


Divide by total Number of Data Points
Actual Output
Predicted Output
$MAE = \frac{1}{N}\sum|Y - \hat{Y}|$
Sum Of
Absolute Value of residual

#### Advantages of MAE

- The MAE you get is in the same unit as the output variable.
- It is most Robust to outliers.

#### Disadvantages of MAE

- The graph of MAE is not differentiable so we have to apply various optimizers like Gradient descent which can be differentiable. from sklearn.metrics import mean_absolute_error print("MAE",mean_absolute_error(y_test,y_pred))

Now to overcome the disadvantage of MAE next metric came as MSE.

## 2) Mean Squared Error(MSE)

MSE is a most used and very simple metric with a little bit of change in mean absolute error. Mean squared error states that finding the squared difference between actual and predicted value.

So, above we are finding the absolute difference and here we are finding the squared difference.

What actually the MSE represents? It represents the squared distance between actual and predicted values. we perform squared to avoid the cancellation of negative terms and it is the benefit of MSE.

$$MSE = \frac{1}{n} \Sigma \underbrace{\left( y - \widehat{y} \right)^2}_{\text{The square of the difference between actual and predicted}}$$

## 3) Root Mean Squared Error(RMSE)

As RMSE is clear by the name itself, that it is a simple square root of mean squared error.

$$\textbf{RMSE} = \sqrt{\textbf{MSE}}$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)}$$

**Advantages of RMSE**

- The output value you get is in the same unit as the required output variable which makes interpretation of loss easy.

**Disadvantages of RMSE**

- It is not that robust to outliers as compared to MAE. for performing RMSE we have to NumPy NumPy square root function over MSE.

print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred)))

Most of the time people use RMSE as an evaluation metric and mostly when you are working with deep learning techniques the most preferred metric is RMSE.

## 4) R Squared (R2)

R2 score is a metric that tells the performance of your model, not the loss in an absolute sense that how many wells did your model perform.

In contrast, MAE and MSE depend on the context as we have seen whereas the R2 score is independent of context.

So, with help of R squared we have a baseline model to compare a model which none of the other metrics provides. The same we have in classification problems which we call a threshold which is fixed at 0.5. So basically R2 squared calculates how must regression line is better than a mean line.

Hence, R2 squared is also known as Coefficient of Determination or sometimes also known as Goodness of fit.

$$\textbf{R2 Squared} = 1 - \frac{SSr}{SSm}$$

**SSr = Squared sum error of regression line**

**SSm = Squared sum error of mean line**

Now, how will you interpret the R2 score? suppose If the R2 score is zero then the above regression line by mean line is equal means 1 so 1-1 is zero. So, in this case, both lines are overlapping means model performance is worst, It is not capable to take advantage of the output column.

Now the second case is when the R2 score is 1, it means when the division term is zero and it will happen when the regression line does not make any mistake, it is perfect. In the real world, it is not possible.

So we can conclude that as our regression line moves towards perfection, R2 score move towards one. And the model performance improves.

The normal case is when the R2 score is between zero and one like 0.8 which means your model is capable to explain 80 per cent of the variance of data.

```
from sklearn.metrics import r2_score
r2 = r2_score(y_test,y_pred)
print(r2)
```

## 5) Adjusted R Squared

The disadvantage of the R2 score is while adding new features in data the R2 score starts increasing or remains constant but it never decreases because It assumes that while adding more data variance of data increases.

But the problem is when we add an irrelevant feature in the dataset then at that time R2 sometimes starts increasing which is incorrect.

Hence, To control this situation Adjusted R Squared came into existence.

$$R_a^2 = 1 - \left[ \left( \frac{n-1}{n-k-1} \right) \times (1 - R^2) \right]$$

where:

| | |
|---|---|
| $n$ | = number of observations |
| $k$ | = number of independent variables |
| $R_a^2$ | = adjusted $R^2$ |

Now as K increases by adding some features so the denominator will decrease, n-1 will remain constant. R2 score will remain constant or will increase slightly so the complete answer will increase and when we subtract this from one then the resultant score will decrease. so this is the case when we add an irrelevant feature in the dataset.

And if we add a relevant feature then the R2 score will increase and 1-R2 will decrease heavily and the denominator will also decrease so the complete term decreases, and on subtracting from one the score increases.

n=40
k=2
adj_r2_score = 1 - ((1-r2)*(n-1)/(n-k-1))

In [93]:
```python
# Importing Evaluation metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# Mean Squared Error
MSE = mean_squared_error(y_test, y_pred)
print("MSE :",MSE)

# Root mean Square error
RMSE = np.sqrt(MSE)
print("RMSE :", RMSE)

# Adjusted R squared
r2 = r2_score(y_test, y_pred)
print("R2 Linear regression :" ,r2*100)
```

```
MSE : 1692554.4321526282
RMSE : 1300.9821029332525
R2 Linear regression : 82.41720232184113
```

## Regularization

### For Example Y = 0.9 + 1.1X1 + 5X2 + 40X3

### Ridge Regressor

Ridge regression is a model tuning method that is used to analyse any data that suffers from multicollinearity. This method performs L2 regularization. When the issue of multicollinearity occurs, least-squares are unbiased, and variances are large, this results in predicted values being far away from the actual values.

In [94]:
```
# Importing the Packages
from sklearn.linear_model import Ridge
```

In [95]:
```
# Training the model
ridgeregressor = Ridge(alpha = 1)
ridgeregressor.fit(x_train, y_train)
```

Out[95]:
```
Ridge(alpha=1)
```

In [96]:
```
# Predicting the model
ridge_y_pred = ridgeregressor.predict(x_test)
ridge_y_pred
```

Out[96]:
```
array([5254.90582885, 9862.95254569, 8808.61974856, ..., 5498.00881366,
       3997.38965642, 6392.34137461])
```

## Checking for over fitting and underfitting

In [97]:
```
(ridgeregressor.score(x_train, y_train))*100, (ridgeregressor.score(x_test, y_te
```

Out[97]:
```
(82.56528783604806, 82.43054263131096)
```

In [98]:
```
# Coefficient Difference
ridge_coef = pd.DataFrame({'lr coefficient':lr.coef_ , 'Ridge coefficient': ridg
ridge_coef.head()
```

Out[98]:

| | lr coefficient | Ridge coefficient | Difference |
|---|---|---|---|
| **0** | -134.440134 | -134.283482 | -0.156651 |
| **1** | -267.119160 | -242.664790 | -24.454371 |
| **2** | 40340.703166 | 40388.385322 | -47.682156 |
| **3** | 1092.118896 | 1148.859168 | -56.740272 |
| **4** | 24.138984 | 37.466390 | -13.327406 |

## Calculating the errors

In [99]:
```
MSE_ridge = mean_squared_error(y_test, ridge_y_pred)
print("MSE :",MSE_ridge)

RMSE_ridge = np.sqrt(MSE)
print("RMSE :", RMSE_ridge)

r2_ridge = r2_score(y_test, ridge_y_pred)
print("R2 Ridge :" ,(r2_ridge)*100)
```

```
MSE : 1691270.267918203
RMSE : 1300.9821029332525
R2 Ridge : 82.43054263131096
```

### Lasso regressor

Lasso regression algorithm is defined as a regularization algorithm that assists in the elimination of irrelevant parameters, thus helping in the concentration of selection and regularizes the models. Lasso models can be evaluated using various metrics such as RMSE and R-Square.

$$\text{Lasso Regressor} = \text{Loss} + \alpha \, || \, w \, ||$$

In [100]:
```
# Impoting the lasso regression
from sklearn.linear_model import Lasso
```

In [101]:
```
# Training the model
lassoregressor = Lasso(alpha = 0.01)
lassoregressor.fit(x_train, y_train)
```

Out[101]:
```
Lasso(alpha=0.01)
```

In [102]:
```
# Predicting the model
lasso_y_pred = lassoregressor.predict(x_test)
lasso_y_pred
```

Out[102]:
```
array([5254.51995513, 9864.11795004, 8809.29792504, ..., 5497.64242635,
       3997.0463246 , 6391.2276783 ])
```

### Checking for overfitting and underfitting

In [103]:
```
(lassoregressor.score(x_train, y_train))*100, (lassoregressor.score(x_test, y_te
```

Out[103]:
```
(82.5652832210641, 82.43035623848732)
```

### Calculating the errors

In [104]:
```
MSE_lasso = mean_squared_error(y_test, lasso_y_pred)
print("MSE :",MSE_lasso)

RMSE_lasso = np.sqrt(MSE)
print("RMSE :", RMSE_lasso)

r2_lasso = r2_score(y_test, lasso_y_pred)
print("R2 Lasso :" ,(r2_lasso)*100)
```

```
MSE : 1691288.210455308
RMSE : 1300.9821029332525
R2 Lasso : 82.43035623848732
```

In [105]:
```
# Coeficient difference
lasso_coef = pd.DataFrame({'lr coefficient':lr.coef_ , 'Lasso coefficient': lasso
lasso_coef.head()
```

Out[105]:

| | lr coefficient | Lasso coefficient | Difference |
|---|---|---|---|
| **0** | -134.440134 | -134.237015 | -0.203119 |
| **1** | -267.119160 | -242.343414 | -24.775747 |
| **2** | 40340.703166 | 40404.105209 | -63.402043 |
| **3** | 1092.118896 | 1148.418202 | -56.299306 |
| **4** | 24.138984 | 37.495317 | -13.356332 |

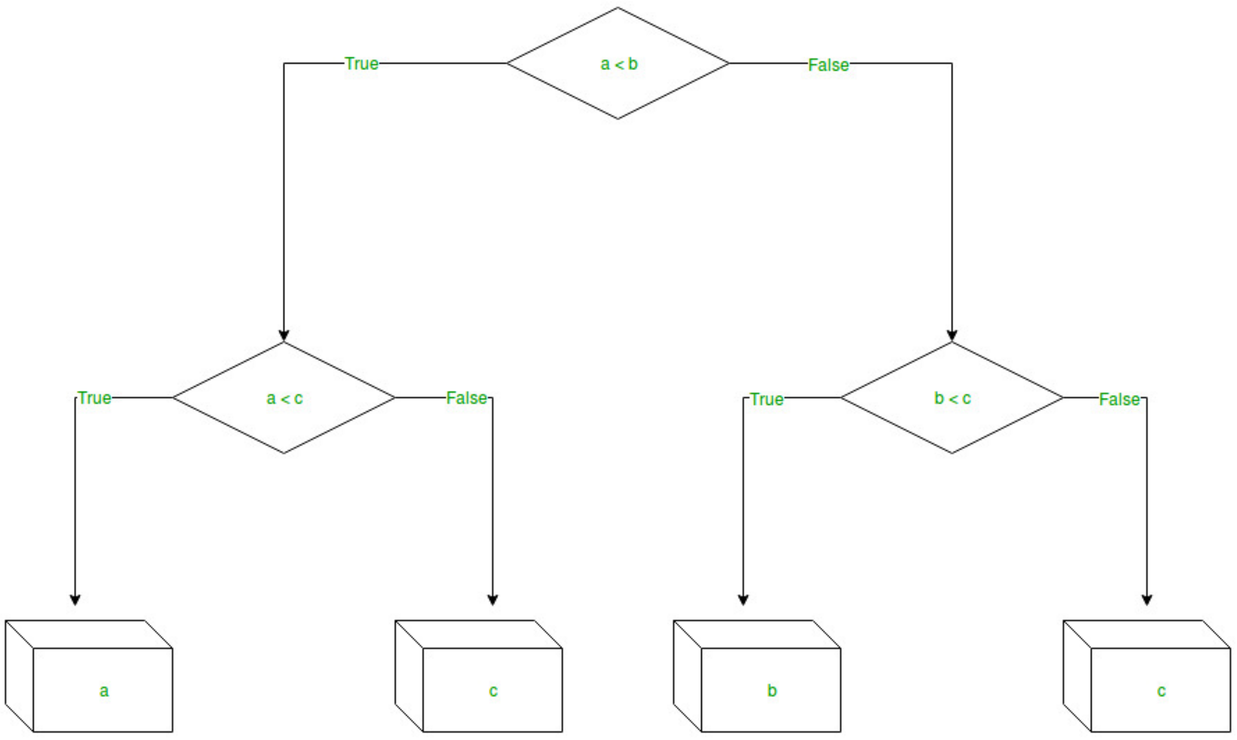
## Decision Tree

Decision Tree is a decision-making tool that uses a flowchart-like tree structure or is a model of decisions and all of their possible results, including outcomes, input costs, and utility. Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.

The branches/edges represent the result of the node and the nodes have either:

1. Conditions [Decision Nodes]
2. Result [End Nodes]

The branches/edges represent the truth/falsity of the statement and take makes a decision based on that in the example below which shows a decision tree that evaluates the smallest of three numbers:



In [106]:
```
# Importing the packages
from sklearn.tree import DecisionTreeRegressor
decision_tree = DecisionTreeRegressor(max_depth=14)

# Fitting the train model
decision_tree.fit(x_train, y_train)

# Predicting from the model
DT_y_pred = decision_tree.predict(x_test)
DT_y_train = decision_tree.predict(x_train)

# Finding the error
MSE = mean_squared_error(y_test,DT_y_pred)
print("MSE:", MSE)

# Root mean squared error
RMSE = np.sqrt(MSE)
print("RMSE :" , RMSE)

# Adjusted r2
r2 = r2_score(y_test, DT_y_pred)
print("R2 for Decision Tree Regressor : ", r2*100)
```

```
MSE: 682301.7584781337
RMSE : 826.0155921519482
R2 for Decision Tree Regressor :  92.91203074661813
```

## Checking for overfitting and underfitting

In [107]:
```
(decision_tree.score(x_train, y_train))*100, (decision_tree.score(x_test, y_test
```

Out[107]:
```
(93.93079843280286, 92.91203074661813)
```

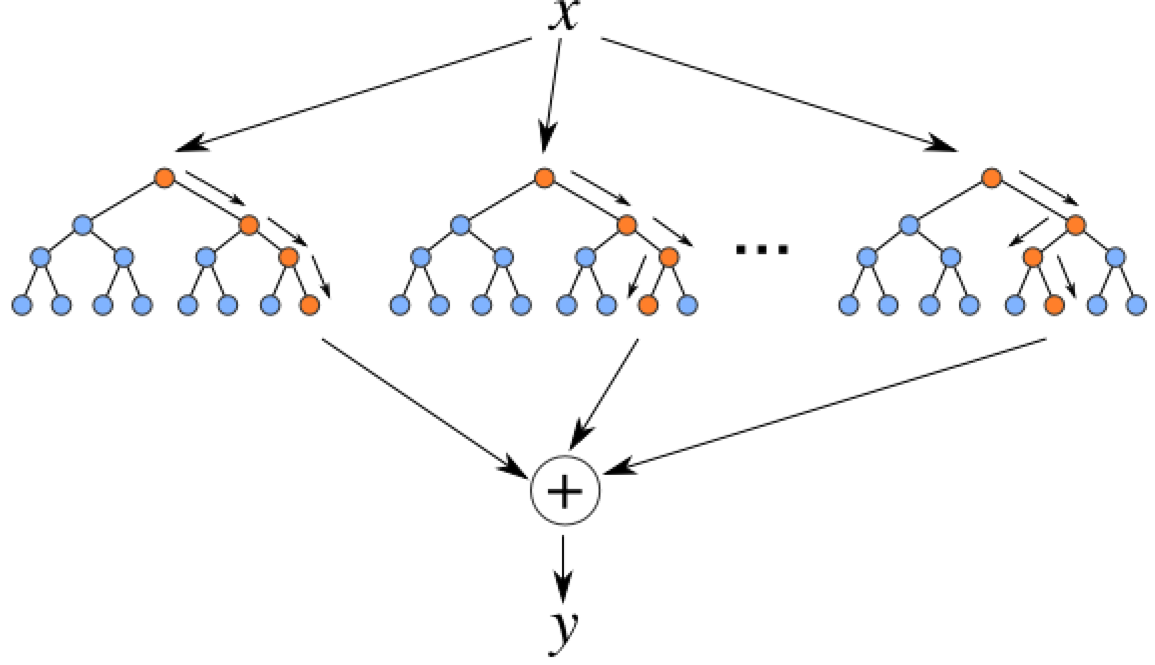
In [108]:
```
# Calculating the difference between actual value and predicted value
predicted_value = pd.DataFrame({'Actual Value': y_test, 'Predicted Value': DT_y_
predicted_value.sample(10)
```

Out[108]:

| | Actual Value | Predicted Value | Difference |
|---|---|---|---|
| **82531** | 9583 | 9475.653543 | 107.346457 |
| **177667** | 4632 | 4659.686857 | -27.686857 |
| **60324** | 10855 | 13215.857143 | -2360.857143 |
| **248609** | 5164 | 5581.570747 | -417.570747 |
| **16471** | 4117 | 4110.531250 | 6.468750 |
| **79784** | 4807 | 5459.872910 | -652.872910 |
| **187339** | 10193 | 8477.730263 | 1715.269737 |
| **59309** | 4661 | 5876.286316 | -1215.286316 |
| **87749** | 6379 | 6500.781955 | -121.781955 |
| **160498** | 3554 | 3921.436620 | -367.436620 |

## Random Forest

Random Forest Regression is a supervised learning algorithm that uses ensemble learning method for regression. Ensemble learning method is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model.



```
In [109]:  # Importing the model
           from sklearn.ensemble import RandomForestRegressor

           # Setting the hyperparameter values
           random_forest = RandomForestRegressor(n_estimators = 20, max_depth=8)
           random_forest.fit(x_train,y_train)
           rf_y_pred = random_forest.predict(x_test)

           # Mean squared error
           MSE = mean_squared_error(y_test, rf_y_pred)
           print("MSE :", MSE)

           # Root mean squared error
           RMSE = np.sqrt(MSE)
           print("RMSE :", RMSE)

           # Adjusted R2
           r2 = r2_score(y_test, rf_y_pred)
           print("R2 for Random Forest :", r2*100)
```

```
MSE : 1220219.2118172387
RMSE : 1104.6353297886315
R2 for Random Forest : 87.32397191084826
```

## Checking for over fitting and underfitting

```
In [110]:  (random_forest.score(x_train, y_train))*100, (random_forest.score(x_test, y_test

Out[110]:  (87.44172065829335, 87.32397191084826)
```

```
In [111]: # Calculating the difference between actual and predicted value
          predicted_value = pd.DataFrame({'Actual Value': y_test, 'Predicted Value': rf_y_
          predicted_value.sample(10)
```

Out[111]:

| | Actual Value | Predicted Value | Difference |
|---|---|---|---|
| **118603** | 16695 | 16262.561842 | 432.438158 |
| **221468** | 4062 | 4309.703418 | -247.703418 |
| **7223** | 3940 | 2740.331372 | 1199.668628 |
| **143038** | 5646 | 6331.379947 | -685.379947 |
| **92365** | 5191 | 5683.186231 | -492.186231 |
| **235713** | 8647 | 8615.019086 | 31.980914 |
| **75871** | 6596 | 6562.804229 | 33.195771 |
| **13293** | 5127 | 4581.400479 | 545.599521 |
| **60864** | 3617 | 3274.498480 | 342.501520 |
| **74928** | 13292 | 11715.903891 | 1576.096109 |

## Printing all the scores

```
In [112]: print("Linear Regression : ", (lr.score(x_train, y_train))*100,",", (lr.score(x_
          print("Ridge Regressor : ", (ridgeregressor.score(x_train, y_train))*100,",", (r
          print("Lasso Regressor : ", (lassoregressor.score(x_train, y_train))*100,",", (la
          print("Decision Tree Regressor : ", (decision_tree.score(x_train, y_train))*100,'
          print("Random Forest Regressor : ", (random_forest.score(x_train, y_train))*100,'
```

```
Linear Regression :  82.5531565762266 , 82.41720232184113
Ridge Regressor :  82.56528783604806 , 82.43054263131096
Lasso Regressor :  82.5652832210641 , 82.43035623848732
Decision Tree Regressor :  93.93079843280286 , 92.91203074661813
Random Forest Regressor :  87.44172065829335 , 87.32397191084826
```

## Printing all the scores using dataframe

```
In [113]: overall_scores = pd.DataFrame({'Linear Regression': ((lr.score(x_train, y_train)
                                         'Ridge Regressor': ((ridgeregressor.score(x_train
                                        'Lasso Regressor': ((lassoregressor.score(x_train,
                                        'Decision Tree Regressor': ((decision_tree.score(x
                                        'Random Forest Regressor': ((random_forest.score(x
          overall_scores.T.rename(columns = {0:'Training Score', 1 : 'Test Score'})
```

Out[113]:

| | Training Score | Test Score |
|---|---|---|
| **Linear Regression** | 82.553157 | 82.417202 |
| **Ridge Regressor** | 82.565288 | 82.430543 |
| **Lasso Regressor** | 82.565283 | 82.430356 |
| **Decision Tree Regressor** | 93.930798 | 92.912031 |
| **Random Forest Regressor** | 87.441721 | 87.323972 |

## Observation

This dataset is a live dataset of Rossmann Stores. On analysing this problem we observe that rossmann problem is a regression problem and our primarily goal is to predict the sales figures of Rossmann problem. In this Notebook we work on following topics Analysing the dataset by using Exploratory Data Analysis using exponential moving averages analyse trends and seasonality in Rossmann dataset Analyse Regression using following prediction analysis.

After Performing different Analysis, we got the following results,

A) Linear Regression Analysis = 82.553157, 82.417202
B) Elastic Regression

- Ridge Regression = 82.565288, 82.430543
- Lasso REgression = 82.565283, 82.430356

C) Dession tree Regression = 93.930798, 92.915127 **BEST**
D) Random Forest Regressor = 87.404635, 87.307340