Rossmann Store Sales

Forecast sales using store, promotion, and competitor data

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. We are provided with historical sales data for 1,115 Rossmann stores. The task is to forecast the "Sales" column.

Data fields

- **Id - an Id that represents a (Store, Date) duple within the test set**
- **Store - a unique Id for each store**
- **Sales - the turnover for any given day (this is what you are predicting)**
- **Customers - the number of customers on a given day**
- **Open - an indicator for whether the store was open: 0 = closed, 1 = open**
- **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**
- **SchoolHoliday - indicates if the (Store, Date) was affected by the closure of public schools**
- **StoreType - differentiates between 4 different store models: a, b, c, d**
- **Assortment - describes an assortment level: a = basic, b = extra, c = extended**
- **CompetitionDistance - distance in meters to the nearest competitor store**
- **CompetitionOpenSince[Month/Year] - gives the approximate year and month of the time the nearest competitor was opened**
- **Promo - indicates whether a store is running a promo on that day**
- **Promo2 - Promo2 is a continuing and consecutive promotion for some stores: 0 = store is not participating, 1 = store is participating**
- **Promo2Since[Year/Week] - describes the year and calendar week when the store started participating in Promo2**
- **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**

In [ ]:

```
the goal of the project is to use the previous store data for training the model and impl
ement the robust machine learning model to predict the store sales for next 6 weeks here
, the target values are continous so we will be using regression models for prediction.
every data science project consists of following steps

1.identify the business statement and think of big picture.
2.Get the data.[Data collection]
3.Exploratory analysis.[statistics of data]
4.Data cleaning[dat wrangling]
5.Select a model and train it.[identify the robust algorithm]
6.Fine-tune your model.[hyper parameter tuning]
7.Present the result.[insights and trends about the data]
8.deploy and provide maintainance
```

In [ ]:

```
#the project follows sequence of steps to derive the insights from the data
1.Understanding the problem statement
2.Data exploration
3.Data Visualization
4.Handling missing values
```

```
5.Handling Outliers
6.Exploring exceptional cases
7.Converting categorical to numeric forms
8.Creating heatmaps
9.Feature selection & its importance
10.Implementation using linear regression
11.Implementation using stochastic gradient descent
12.Implementation using random forest
13.Implementation using decision trees
14.Understanding feature importance

#we will start the project by importing the necessary librarys which are essential for an
alysis of data #here we are importing below librarys and aliasing them to hide the comple
xcity[increases Readbilty]

pds=[data manipulation]
npy=[mathematical operations on arrays]
sea=[data visualization in 3d & attractive visualization]
pt=[data visualization in 2d]
```

In [1]:

```
# importing necessary libraries

import pandas as pds
import numpy as npy
import seaborn as sea
import matplotlib.pyplot as pt
```

What our data looks like

In [2]:

```
#importing and the storing the dataset in S_data by using read_csv function
#here the dataset is broken down into train and test data to feed the model
S_data = pds.read_csv(r"C:\Users\R411996\Desktop\data-science\Rossmann\store.csv")
S_data
```

Out[2]:

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2
0	1	c	a	1270.0	9.0	2008.0	0
1	2	a	a	570.0	11.0	2007.0	1
2	3	a	a	14130.0	12.0	2006.0	1
3	4	c	c	620.0	9.0	2009.0	0
4	5	a	a	29910.0	4.0	2015.0	0
...	...	...	...	...	...	...	...
1110	1111	a	a	1900.0	6.0	2014.0	1
1111	1112	c	c	1880.0	4.0	2006.0	0
1112	1113	a	c	9260.0	NaN	NaN	0
1113	1114	a	c	870.0	NaN	NaN	0
1114	1115	d	c	5350.0	NaN	NaN	1

1115 rows × 10 columns

In [3]:

```
#importing the training data and the storing the dataset in T_data by using read_csv func
tion
T_data = pds.read_csv(r"C:\Users\R411996\Desktop\data-science\Rossmann\train.csv")
T_data
```

```
C:\Users\R411996\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3146: Dtype
Warning: Columns (7) have mixed types.Specify dtype option on import or set low_memory=Fa
lse.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

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
...	...	...	...	...	...	...	...	...	...
1017204	1111	2	2013-01-01	0	0	0	0	a	1
1017205	1112	2	2013-01-01	0	0	0	0	a	1
1017206	1113	2	2013-01-01	0	0	0	0	a	1
1017207	1114	2	2013-01-01	0	0	0	0	a	1
1017208	1115	2	2013-01-01	0	0	0	0	a	1

1017209 rows × 9 columns

In [4]:

```
#merging T_data& S_data on common attribute to get store details and storing in features
variable
features = pds.merge(T_data, S_data, on='Store')
features
```

Out[4]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	StoreType	Assortment	C
0	1	5	2015-07-31	5263	555	1	1	0	1	c	a	
1	1	4	2015-07-30	5020	546	1	1	0	1	c	a	
2	1	3	2015-07-29	4782	523	1	1	0	1	c	a	
3	1	2	2015-07-28	5011	560	1	1	0	1	c	a	
4	1	1	2015-07-27	6102	612	1	1	0	1	c	a	
...	...	...	...	...	...	...	...	...	...	...	...	
1017204	1115	6	2013-01-05	4771	339	1	0	0	1	d	c	
1017205	1115	5	2013-01-04	4540	326	1	0	0	1	d	c	
1017206	1115	4	2013-01-03	4297	300	1	0	0	1	d	c	
1017207	1115	3	2013-01-02	3697	305	1	0	0	1	d	c	
1017208	1115	2	2013-01-01	0	0	0	0	a	1	d	c	

1017209 rows × 18 columns

In [5]:

```
#one can have a count of missing values in dta by using isnull.sum() function
features.shape
```

Out[5]:

```
(1017209, 18)
```

In [6]:

```
#head() function displays the top 5 rows of dataset
features.head()
```

Out[6]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	StoreType	Assortment	Competit
0	1	5	2015-07-31	5263	555	1	1	0	1	c	a	
1	1	4	2015-07-30	5020	546	1	1	0	1	c	a	
2	1	3	2015-07-29	4782	523	1	1	0	1	c	a	
3	1	2	2015-07-28	5011	560	1	1	0	1	c	a	
4	1	1	2015-07-27	6102	612	1	1	0	1	c	a	

In [7]:

```
#we can specify the limit of the number
features.head(10)
```

Out[7]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	StoreType	Assortment	Competit
0	1	5	2015-07-31	5263	555	1	1	0	1	c	a	
1	1	4	2015-07-30	5020	546	1	1	0	1	c	a	
2	1	3	2015-07-29	4782	523	1	1	0	1	c	a	
3	1	2	2015-07-28	5011	560	1	1	0	1	c	a	
4	1	1	2015-07-27	6102	612	1	1	0	1	c	a	
5	1	7	2015-07-26	0	0	0	0	0	0	c	a	
6	1	6	2015-07-25	4364	500	1	0	0	0	c	a	
7	1	5	2015-07-24	3706	459	1	0	0	0	c	a	
8	1	4	2015-07-23	3769	503	1	0	0	0	c	a	
9	1	3	2015-07-22	3464	463	1	0	0	0	c	a	

In [8]:

```
#tail() function displays the bottom 5 rows of dataset
```

```
features.tail()
```

Out[8]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	StoreType	Assortment	Co
1017204	1115	6	2013-01-05	4771	339	1	0	0	1	d	c	
1017205	1115	5	2013-01-04	4540	326	1	0	0	1	d	c	
1017206	1115	4	2013-01-03	4297	300	1	0	0	1	d	c	
1017207	1115	3	2013-01-02	3697	305	1	0	0	1	d	c	
1017208	1115	2	2013-01-01	0	0	0	0	a	1	d	c	

In [9]:

```
#we can specify the limit of the number
features.tail(15)
```

Out[9]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	StoreType	Assortment	Co
1017194	1115	2	2013-01-15	3528	277	1	0	0	0	d	c	
1017195	1115	1	2013-01-14	3158	252	1	0	0	0	d	c	
1017196	1115	7	2013-01-13	0	0	0	0	0	0	d	c	
1017197	1115	6	2013-01-12	4497	350	1	0	0	0	d	c	
1017198	1115	5	2013-01-11	5142	351	1	1	0	1	d	c	
1017199	1115	4	2013-01-10	5007	339	1	1	0	1	d	c	
1017200	1115	3	2013-01-09	4649	324	1	1	0	1	d	c	
1017201	1115	2	2013-01-08	5243	341	1	1	0	1	d	c	
1017202	1115	1	2013-01-07	6905	471	1	1	0	1	d	c	
1017203	1115	7	2013-01-06	0	0	0	0	0	1	d	c	
1017204	1115	6	2013-01-05	4771	339	1	0	0	1	d	c	
1017205	1115	5	2013-01-04	4540	326	1	0	0	1	d	c	
1017206	1115	4	2013-01-03	4297	300	1	0	0	1	d	c	
1017207	1115	3	2013-01-02	3697	305	1	0	0	1	d	c	
1017208	1115	2	2013-01-01	0	0	0	0	a	1	d	c	

In [10]:

```
#the describe function gives the statistics about the data
```

```
#the describe function gives the statistics about the data
features.describe()
```

Out[10]:

	Store	DayOfWeek	Sales	Customers	Open	Promo	SchoolHoliday	CompetitionDist
count	1.017209e+06	1.017209e+06	1.017209e+06	1.017209e+06	1.017209e+06	1.017209e+06	1.017209e+06	1.014567
mean	5.584297e+02	3.998341e+00	5.773819e+03	6.331459e+02	8.301067e-01	3.815145e-01	1.786467e-01	5.430086
std	3.219087e+02	1.997391e+00	3.849926e+03	4.644117e+02	3.755392e-01	4.857586e-01	3.830564e-01	7.715324
min	1.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	2.000000
25%	2.800000e+02	2.000000e+00	3.727000e+03	4.050000e+02	1.000000e+00	0.000000e+00	0.000000e+00	7.100000
50%	5.580000e+02	4.000000e+00	5.744000e+03	6.090000e+02	1.000000e+00	0.000000e+00	0.000000e+00	2.330000
75%	8.380000e+02	6.000000e+00	7.856000e+03	8.370000e+02	1.000000e+00	1.000000e+00	0.000000e+00	6.890000
max	1.115000e+03	7.000000e+00	4.155100e+04	7.388000e+03	1.000000e+00	1.000000e+00	1.000000e+00	7.586000

In [11]:

```
#the info() function enumerates over the column and gives details about the data types
features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1017209 entries, 0 to 1017208
Data columns (total 18 columns):
 #   Column                     Non-Null Count    Dtype
---  ------                     --------------    -----
 0   Store                      1017209 non-null  int64
 1   DayOfWeek                  1017209 non-null  int64
 2   Date                       1017209 non-null  object
 3   Sales                      1017209 non-null  int64
 4   Customers                  1017209 non-null  int64
 5   Open                       1017209 non-null  int64
 6   Promo                      1017209 non-null  int64
 7   StateHoliday               1017209 non-null  object
 8   SchoolHoliday              1017209 non-null  int64
 9   StoreType                  1017209 non-null  object
 10  Assortment                 1017209 non-null  object
 11  CompetitionDistance        1014567 non-null  float64
 12  CompetitionOpenSinceMonth  693861 non-null   float64
 13  CompetitionOpenSinceYear   693861 non-null   float64
 14  Promo2                     1017209 non-null  int64
 15  Promo2SinceWeek            509178 non-null   float64
 16  Promo2SinceYear            509178 non-null   float64
 17  PromoInterval              509178 non-null   object
dtypes: float64(5), int64(8), object(5)
memory usage: 147.5+ MB
```

Exploring Dataset

In [6]:

```
#obtaining the unique values in dataset and removing the irrevalant coloumns which doesnt contribute while predicting the target value
print("Total size of dataset: ", len(features))
print(features.isnull().sum())
```

```
Total size of dataset:  1017209
Store                            0
DayOfWeek                        0
Date                             0
Sales                            0
Customers                        0
Open                             0
Promo                            0
StateHoliday                     0
```

```
SchoolHoliday                     0
StoreType                         0
Assortment                        0
CompetitionDistance            2642
CompetitionOpenSinceMonth    323348
CompetitionOpenSinceYear     323348
Promo2                            0
Promo2SinceWeek              508031
Promo2SinceYear              508031
PromoInterval                508031
dtype: int64
```

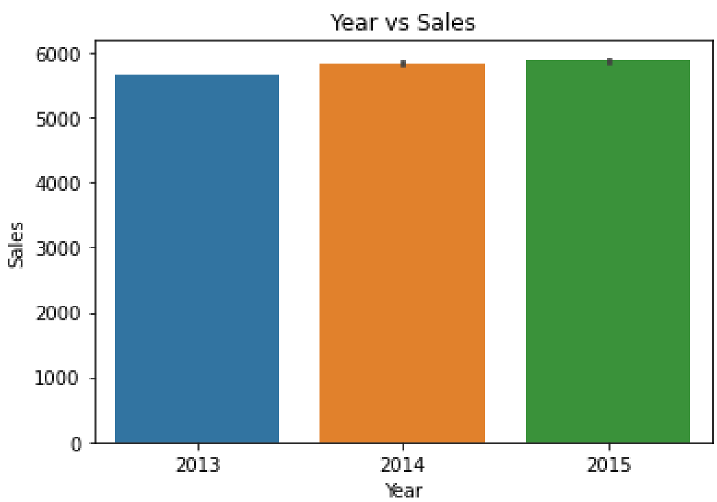
In [7]:

```
#obtaining the unique values in dataset and removing the irrevalant coloumns which doesnt contribute while predicting the target value
columns = list(features.columns)
columns.remove('Date')
columns.remove('CompetitionDistance')
for col in columns:
    print(col,"--------->",features[col].unique())
```

```
Store ---------> [   1    2    3 ... 1113 1114 1115]
DayOfWeek ---------> [5 4 3 2 1 7 6]
Sales ---------> [ 5263  5020  4782 ... 20362 18841 21237]
Customers ---------> [ 555  546  523 ... 3727 4022 4106]
Open ---------> [1 0]
Promo ---------> [1 0]
StateHoliday ---------> ['0' 'a' 'b' 'c' 0]
SchoolHoliday ---------> [1 0]
StoreType ---------> ['c' 'a' 'd' 'b']
Assortment ---------> ['a' 'c' 'b']
CompetitionOpenSinceMonth ---------> [ 9. 11. 12.  4. 10.  8. nan  3.  6.  5.  1.  2.  7.]
CompetitionOpenSinceYear ---------> [2008. 2007. 2006. 2009. 2015. 2013. 2014. 2000. 2011. nan 2010. 2005.
 1999. 2003. 2012. 2004. 2002. 1961. 1995. 2001. 1990. 1994. 1900. 1998.]
Promo2 ---------> [0 1]
Promo2SinceWeek ---------> [nan 13. 14.  1. 45. 40. 26. 22.  5.  6. 10. 31. 37.  9. 39. 27. 18. 35.
 23. 48. 36. 50. 44. 49. 28.]
Promo2SinceYear ---------> [  nan 2010. 2011. 2012. 2009. 2014. 2015. 2013.]
PromoInterval ---------> [nan 'Jan,Apr,Jul,Oct' 'Feb,May,Aug,Nov' 'Mar,Jun,Sept,Dec']
```

In [8]:

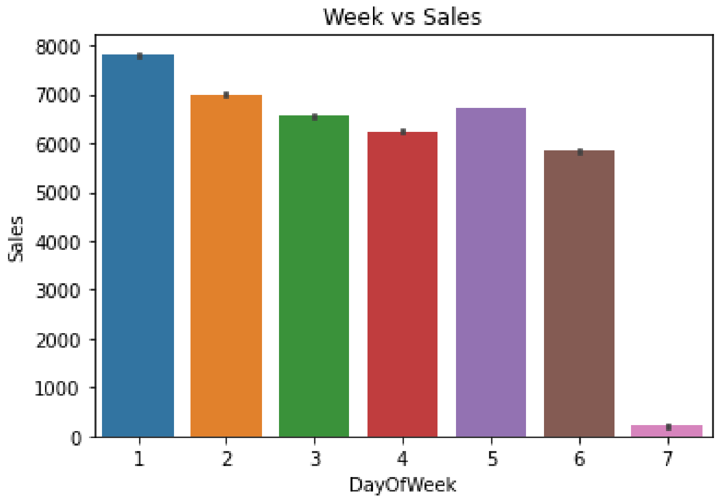
```
#extracting year and month from Date column by defning lamda function by slicing technique and plotting year vs sales by using barplot
features['Year']  = features['Date'].apply(lambda x: int(str(x)[:4]))
features['Month'] = features['Date'].apply(lambda x: int(str(x)[5:7]))


sea.barplot(x='Year', y='Sales', data=features).set(title='Year vs Sales')
pt.show()
```



Sales have been increasing year to year

In [9]:

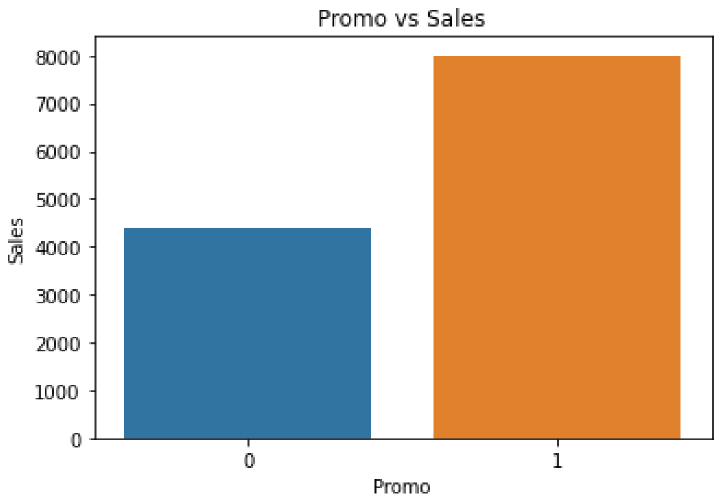
```
#now, we will be plotting each variable with sales variable to determine the the impact o
f each variable and poltting Sales with respect to week[trend analysis
sea.barplot(x='DayOfWeek', y='Sales', data=features).set(title='Week vs Sales')
pt.show()
```



Sales on 1 (Monday) and 5 (Friday) are the highest

In [10]:

```
# poltting Sales with respect to week[trend analysis] and it has high contribution
sea.barplot(x='Promo', y='Sales', data=features).set(title='Promo vs Sales')
pt.show()
```



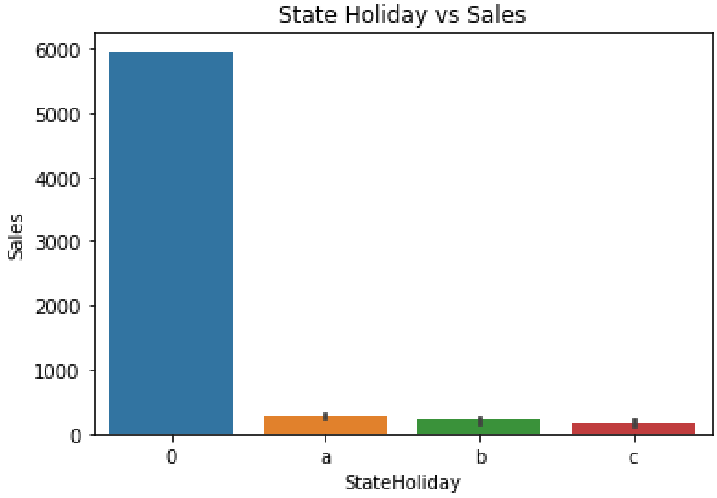
Customers are definately attracted by Promo codes thus sales are higher when there is a Promo code in a Store

In [17]:

```
#here we are changing the data type
#StateHoliday column has values 0 & "0", So, we need to change values with 0 to "0"

features["StateHoliday"].loc[features["StateHoliday"] == 0] = "0"

sea.barplot(x='StateHoliday', y='Sales', data=features).set(title='State Holiday vs Sales
')
pt.show()
```

```
C:\Users\R411996\Anaconda3\lib\site-packages\pandas\core\indexing.py:670: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  iloc._setitem_with_indexer(indexer, value)
```

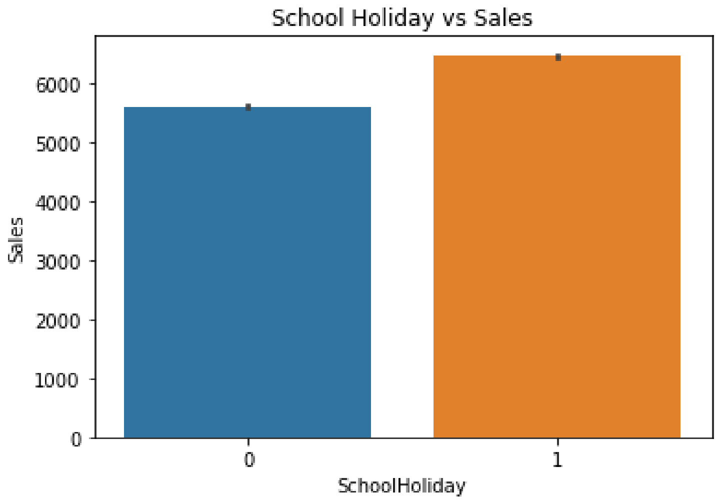


Most stores are closed on State Holidays that's why we can see that there are very less sales in a,b,c where:

- **a = Public Holiday**
- **b = Easter Holiday**
- **c = Chirstmas**
- **0 = No Holiday, Working day**

In [18]:

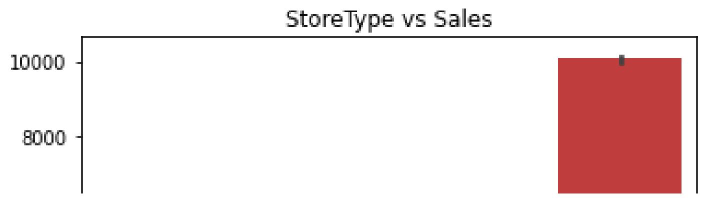
```
#visualization of Sales with respect to School Holiday
sea.barplot(x='SchoolHoliday', y='Sales', data=features).set(title='School Holiday vs Sales')
pt.show()
```

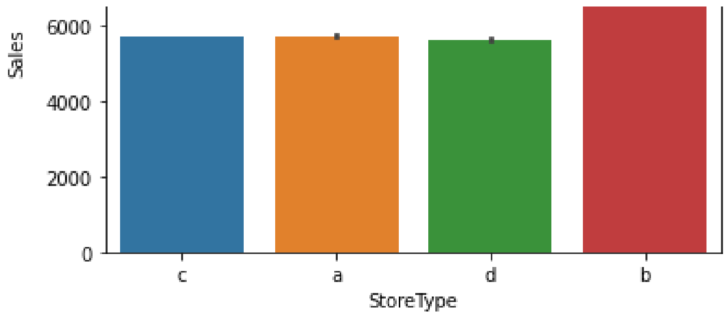


On School Holidays there are more sales!

In [ ]:

```
#visualization of Sales with respect to StoreType
sea.barplot(x='StoreType', y='Sales', data=features).set(title='StoreType vs Sales')
pt.show()
```

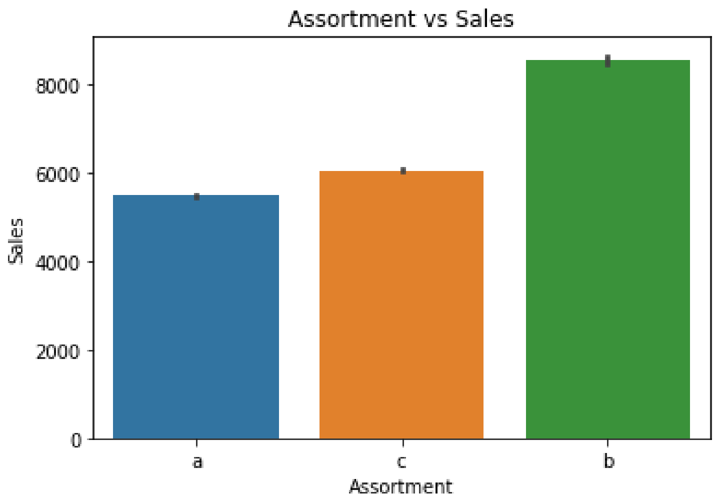




a,b,c,d are store models where b type stores have the highest sales

In [ ]:

```
#visualization of Sales with respect to Assortment
sea.barplot(x='Assortment', y='Sales', data=features).set(title='Assortment vs Sales')
pt.show()
```



Assortment level b have the highest sales

Assortment levels:

- **a = basic**
- **b = extra**
- **c = entended**

Filling Missing Values and Removing Outliers

Few columns have high number of missing values, so we need to fill them with appropriate method for better result

Filling Missing Values

Approach

1: The null values in Column Promo2SinceWeek, Promo2SinceYear, PromoInterval is due to Promo2 is 0 for those stores. So we would fill all the null values in these columns with 0.

2: Since Competition Distance for 3 stores isn't given so we could fill it with mean of the distance given for all other stores

3: CompetitionOpenSinceMonth, CompetitionOpenSinceYear can be filled using the most occuring month and year respectively.

In [ ]:

```
#we need to fill up the missing values by using above methods,to have a better predicting model
```

In [8]:

```
#fucntion to determine the count of missing values
S_data.isnull().sum()
```

Out[8]:

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

In [9]:

```
#fucntion to determine the count of missing values
T_data.isnull().sum()
```

Out[9]:

```
Store            0
DayOfWeek        0
Date             0
Sales            0
Customers        0
Open             0
Promo            0
StateHoliday     0
SchoolHoliday    0
dtype: int64
```

In [ ]:

```
# Filling the missing values of Promo2SinceWeek, Promo2SinceYear, PromoInterval with 0 as promo2 is 0
S_data.updsate(S_data[['Promo2SinceWeek','Promo2SinceYear','PromoInterval']].fillna(0))
```

In [ ]:

```
# Filling CompetitionDistance with mean distance by using mean() function
mean_competition_distance = S_data['CompetitionDistance'].mean()
S_data['CompetitionDistance'].fillna(mean_competition_distance, inpylace=True)
```

In [ ]:

```
# Filling CompetitionOpenSinceMonth, CompetitionOpenSinceYear with most occuring month and year respectively by using mode() function
mode_competition_open_month = S_data['CompetitionOpenSinceMonth'].mode()[0]

mode_competition_open_year = S_data['CompetitionOpenSinceYear'].mode()[0]

S_data['CompetitionOpenSinceMonth'].fillna(mode_competition_open_month,inpylace=True)
S_data['CompetitionOpenSinceYear'].fillna(mode_competition_open_year,inpylace=True)
```

In [ ]:

```
S_data.isnull().sum()
```

Out[ ]:

```
Store                        0
StoreType                    0
Assortment                   0
```

```
Assortment                   0
CompetitionDistance          0
CompetitionOpenSinceMonth    0
CompetitionOpenSinceYear     0
Promo2                       0
Promo2SinceWeek              0
Promo2SinceYear              0
PromoInterval                0
dtype: int64
```

- **All missing values have been filled**

In [ ]:

```python
# combining based on common attribute
features = pds.merge(T_data, S_data, on='Store')
features.head()
```

Out[ ]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	StoreType	Assortment	Competit
0	1	5	2015-07-31	5263	555	1	1	0	1	c	a	
1	1	4	2015-07-30	5020	546	1	1	0	1	c	a	
2	1	3	2015-07-29	4782	523	1	1	0	1	c	a	
3	1	2	2015-07-28	5011	560	1	1	0	1	c	a	
4	1	1	2015-07-27	6102	612	1	1	0	1	c	a	

In [ ]:

```python
#after implenmenting missing values techniques, we find there are no missing values and d
ata is ready to be fed to the model
features.isnull().sum()
```

Out[ ]:

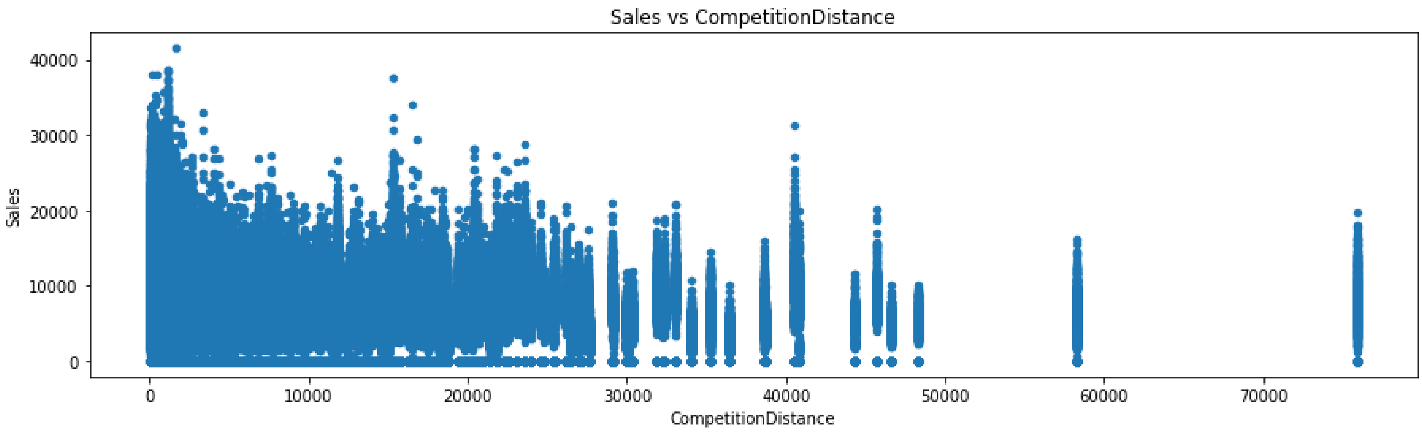
```
Store                        0
DayOfWeek                    0
Date                         0
Sales                        0
Customers                    0
Open                         0
Promo                        0
StateHoliday                 0
SchoolHoliday                0
StoreType                    0
Assortment                   0
CompetitionDistance          0
CompetitionOpenSinceMonth    0
CompetitionOpenSinceYear     0
Promo2                       0
Promo2SinceWeek              0
Promo2SinceYear              0
PromoInterval                0
dtype: int64
```

In [ ]:

```python
features.plot(kind='scatter',x='CompetitionDistance',y='Sales',figsize=(15,4),title="Sal
es vs CompetitionDistance")
```

Out[ ]:

```
<AxesSubplot:title={'center':'Sales vs CompetitionDistance'}, xlabel='CompetitionDistance
', ylabel='Sales'>
```



- **CompetitionDistance is the distance in meters to the nearest competitor store, the more nearer the two stores are the more sales can be seen**

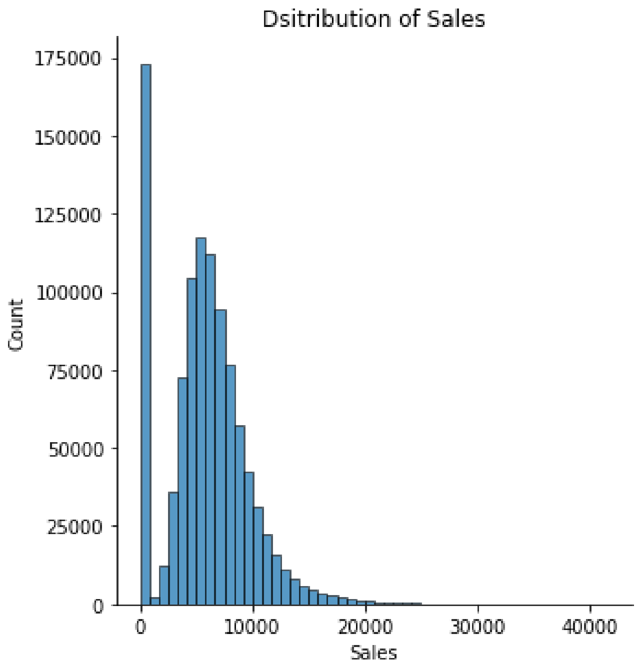
Finding Outliers

In [ ]:

```
# checking distribution of sales
sea.displot(features, x="Sales",bins=50).set(title='Dsitribution of Sales')
```

Out[ ]:

```
<seaborn.axisgrid.FacetGrid at 0x1363b2610>
```



- **As we can see in the distribution plot Sales greater than 30k are very less so they might be the outliers**

Z-Score: If the Z-Score of a datapoint is greater than 3 that can be considered as an Outlier

In [ ]:

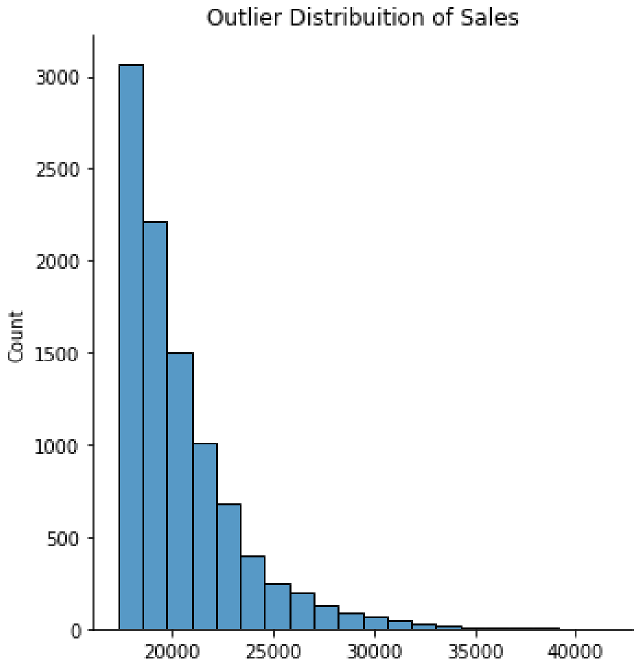
```
mean_of_sales = npy.mean(features['Sales'])
std_of_sales = npy.std(features['Sales'])
print("Mean of Sales: ",mean_of_sales)
print("Standard Deviation of Sales: ",std_of_sales)
```

```
threshold = 3
outlier = []
for i in features['Sales']:
    z = (i-mean_of_sales)/std_of_sales
    if z > threshold:
        outlier.append(i)
print('Total outlier in dataset are: ', len(outlier))
print("Maximum Sales Outlier: ", max(outlier))
print("Minimum Sales Outlier: ", min(outlier))

sea.displot(x=outlier,bins=20).set(title='Outlier Distribuition of Sales')
```

```
Mean of Sales:  5773.818972305593
Standard Deviation of Sales:  3849.924282837463
Total outlier in dataset are:  9731
Maximum Sales Outlier:  41551
Minimum Sales Outlier:  17325
```

Out[ ]:

```
<seaborn.axisgrid.FacetGrid at 0x1364fa4c0>
```



In [ ]:

```
# Looking for outliers
sales_zero = features.loc[features['Sales'] == 0]
sales_greater_than_30 = features.loc[features['Sales'] > 30000]

print("Length of actual dataset:", len(features))
print("Length of data where sales is 0:", len(sales_zero),
      " which is", len(sales_zero)/len(features)*100, "% of the whole data", )

print("Length of data which is greater than 30:", len(sales_greater_than_30),
      "which is ", len(sales_greater_than_30)/len(features)*100, "% of the whole data")
```

```
Length of actual dataset: 1017209
Length of data where sales is 0: 172871  which is 16.994639253093514 % of the whole data
Length of data which is greater than 30: 153 which is  0.015041156733768577 % of the whol
e data
```

- **Droping sales which are greater than 30k as they are very less in the dataset and are probably outliers**

In [ ]:

```
#using inplace to reflect the changes in the original dataset
features.drop(features.loc[features['Sales'] > 30000].index,inplace=True)
features.shape
```

Out[ ]:

```
(1017056, 18)
```

Further EDA - exploring exceptional cases

Looking for a scenerio where the Stores are open and yet there is no sales on that day

In [ ]:

```
# there are no sales when Stores were Open
no_holiday_zero_sales = features.loc[(features['Sales'] == 0) & (features['Open'] == 1)
&
                                                  (features['StateHoliday'] == 0) & (features['
SchoolHoliday'] == 0)]

print("Size of the data where sales were zero even when stores were open: ",len(no_holida
y_zero_sales))
no_holiday_zero_sales.head()
```

```
Size of the data where sales were zero even when stores were open:  12
```

Out[ ]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	StoreType	Assortment	Co
22589	25	4	2014-02-13	0	0	1	0	0	0	c	a	
22590	25	3	2014-02-12	0	0	1	0	0	0	c	a	
25212	28	4	2014-09-04	0	0	1	1	0	0	a	a	
205303	227	4	2014-09-11	0	0	1	0	0	0	a	a	
297110	327	3	2014-03-12	0	0	1	0	0	0	c	c	

- **Removing these data points too as they are an exceptional case**

In [ ]:

```
#using inplace to reflect the changes in the original dataset
features.drop(features.loc[(features['Sales'] == 0) & (features['Open'] == 1) &
                                              (features['StateHoliday'] == 0) &
                                              (features['SchoolHoliday'] == 0)].index,inplace=Tru
e)
features.head()
```

Out[ ]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	StoreType	Assortment	Competit
0	1	5	2015-07-31	5263	555	1	1	0	1	c	a	
1	1	4	2015-07-30	5020	546	1	1	0	1	c	a	
2	1	3	2015-07-29	4782	523	1	1	0	1	c	a	
3	1	2	2015-07-28	5011	560	1	1	0	1	c	a	
4	1	1	2015-07-27	6102	612	1	1	0	1	c	a	

In [ ]:

```
features
```

Converting Categorical Variable to Numeric

In [ ]:

```
# extracting year and month from Date
features['Year']  = features['Date'].apply(lambda x: int(str(x)[:4]))
features['Month'] = features['Date'].apply(lambda x: int(str(x)[5:7]))
features.drop(['Date'],axis=1,inplace=True)
```

In [ ]:

```
features.head()
```

Out[ ]:

	Store	DayOfWeek	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	StoreType	Assortment	CompetitionDist
0	1	5	5263	555	1	1	0	1	c	a	12
1	1	4	5020	546	1	1	0	1	c	a	12
2	1	3	4782	523	1	1	0	1	c	a	12
3	1	2	5011	560	1	1	0	1	c	a	12
4	1	1	6102	612	1	1	0	1	c	a	12

In [ ]:

```
# encoding all categorical varibale to numeric values
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()

features['StoreType']= label_encoder.fit_transform(features['StoreType'])
features['Assortment']= label_encoder.fit_transform(features['Assortment'])

# for promo interval
features["PromoInterval"].loc[features["PromoInterval"] == "Jan,Apr,Jul,Oct"] = 1
features["PromoInterval"].loc[features["PromoInterval"] == "Feb,May,Aug,Nov"] = 2
features["PromoInterval"].loc[features["PromoInterval"] == "Mar,Jun,Sept,Dec"] = 3
'''
new_promo_interval = []
for i in range(len(features)):
    if features['PromoInterval'][i] == 'Jan,Apr,Jul,Oct':
        new_promo_interval.append(1)
    elif features['PromoInterval'][i] == 'Feb,May,Aug,Nov':
        new_promo_interval.append(2)
    elif features['PromoInterval'][i] == 'Mar,Jun,Sept,Dec':
        new_promo_interval.append(3)
    else:
        new_promo_interval.append(0)

features['PromoInterval'] = new_promo_interval
'''

# for State Holiday
features["StateHoliday"].loc[features["StateHoliday"] == "a"] = 1
features["StateHoliday"].loc[features["StateHoliday"] == "b"] = 2
features["StateHoliday"].loc[features["StateHoliday"] == "c"] = 3

'''
state_holiday_list = []
for i in range(len(features)):
    if features['StateHoliday'][i] == 'a':
        state_holiday_list.append(1)
    elif features['StateHoliday'][i] == 'b':
```

```
            state_holiday_list.append(2)
        elif features['StateHoliday'][i] == 'c':
            state_holiday_list.append(3)
        else:
            state_holiday_list.append(0)


features['StateHoliday'] = state_holiday_list
'''
features.head()
```

```
/usr/local/lib/python3.9/site-packages/pandas/core/indexing.py:670: SettingWithCopyWarnin
g:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  iloc._setitem_with_indexer(indexer, value)
```

Out[ ]:

	Store	DayOfWeek	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	StoreType	Assortment	CompetitionDist
0	1	5	5263	555	1	1	0	1	2	0	12
1	1	4	5020	546	1	1	0	1	2	0	12
2	1	3	4782	523	1	1	0	1	2	0	12
3	1	2	5011	560	1	1	0	1	2	0	12
4	1	1	6102	612	1	1	0	1	2	0	12

In [ ]:

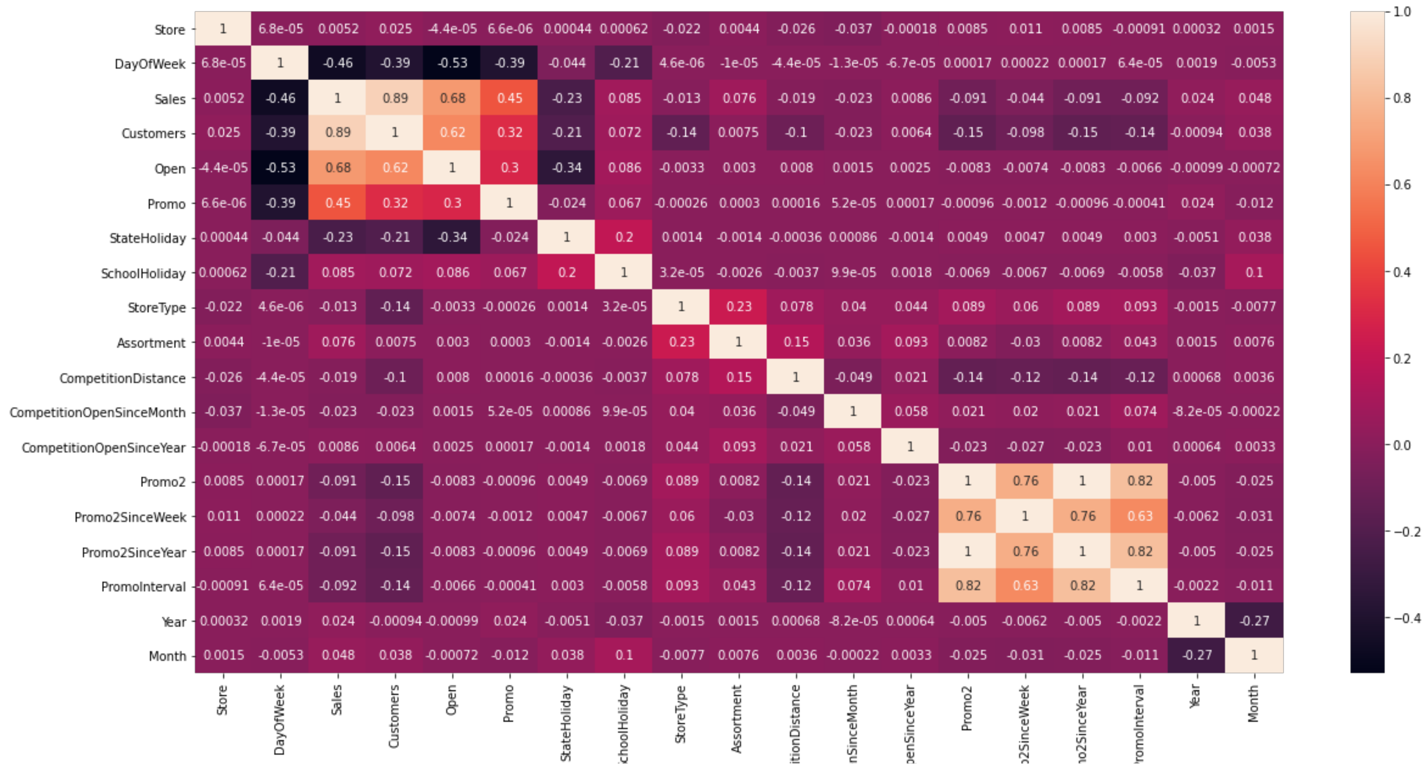
```
#visualizing the correlation between variables by using heatmap
features['StateHoliday'] = pds.to_numeric(features['StateHoliday'])
features['PromoInterval'] = pds.to_numeric(features['PromoInterval'])
```

In [ ]:

```
pt.figure(figsize=(20,10))
sea.heatmap(features.corr(),annot=True)
```

Out[ ]:

<AxesSubplot:>



Correlation map shows

- **Sales is highly correlated with Customers, Open, Promo code**
- **Promo code is highly correlated to Promo2SinceWeek, Promo2SinceYear, PromoInterval**

Implementing Models

In [ ]:

```
features[features['Open']==0]
```

Out[ ]:

	Store	DayOfWeek	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	StoreType	Assortment	Competiti
5	1	7	0	0	0	0	0	0	2	0	
12	1	7	0	0	0	0	0	0	2	0	
19	1	7	0	0	0	0	0	0	2	0	
26	1	7	0	0	0	0	0	0	2	0	
33	1	7	0	0	0	0	0	0	2	0	
...	...	...	...	...	...	...	...	...	...	...	
1017182	1115	7	0	0	0	0	0	0	3	2	
1017189	1115	7	0	0	0	0	0	0	3	2	
1017196	1115	7	0	0	0	0	0	0	3	2	
1017203	1115	7	0	0	0	0	0	1	3	2	
1017208	1115	2	0	0	0	0	1	1	3	2	

172817 rows × 19 columns

In [ ]:

```
#importing the evaluation metrics and librarys
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_absolute_percentage_error
import math
```

In [ ]:

```
#sepearating the traning and testing data
features_subset = features[features['Open'] == 1]
features_subset_closed = features[features['Open'] == 0]
x_train, x_test, y_train, y_test_open = train_test_split(features_subset.drop(['Sales','
Customers', 'Open'],axis=1),
                                                    features_subset['Sales'],test_size=
0.20)
```

In [ ]:

```
epsilon = 1e-10
```

In [ ]:

```
x_train.columns
```

Out[ ]:

```
Index(['Store', 'DayOfWeek', 'Promo', 'StateHoliday', 'SchoolHoliday',
       'StoreType', 'Assortment', 'CompetitionDistance',
       'CompetitionOpenSinceMonth', 'CompetitionOpenSinceYear', 'Promo2',
       'Promo2SinceWeek', 'Promo2SinceYear', 'PromoInterval', 'Year', 'Month'],
      dtype='object')
```

Linear Regression

In [ ]:

```
#next we are spot-checking the different types of regression algorithms and select the al
gorithms which has high prediction accuracy by using evaluation metrics
```

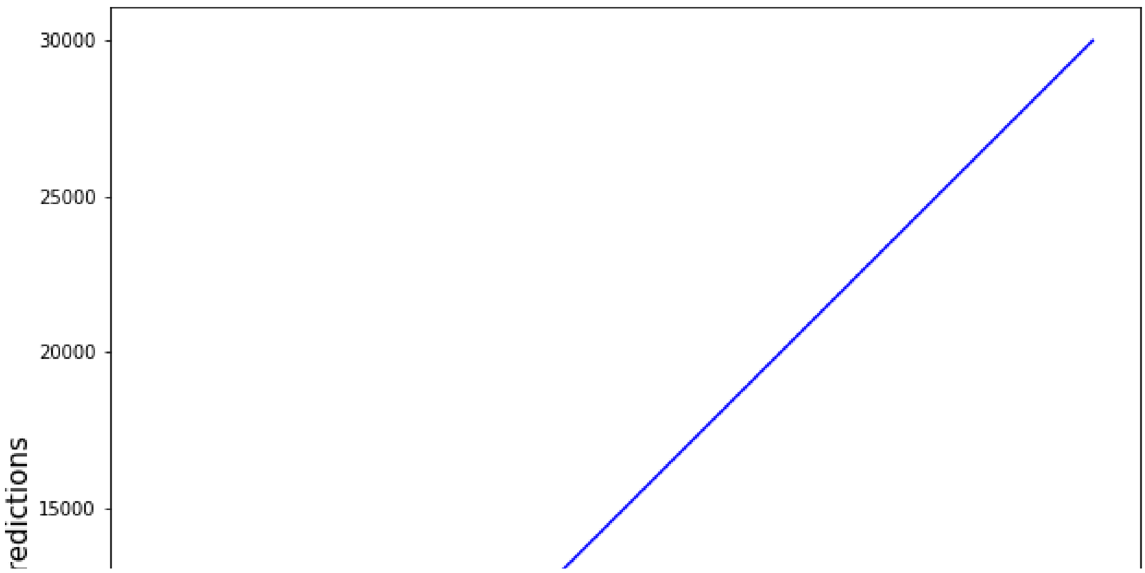
In [ ]:

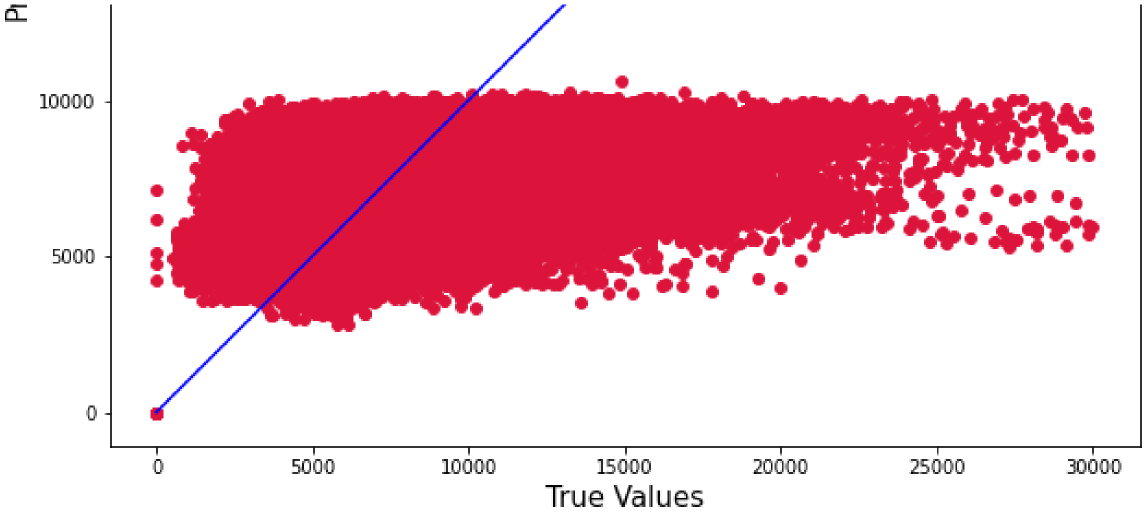
```
from sklearn import linear_model

reg_model = linear_model.LinearRegression() # making regression model
reg_model.fit(x_train, y_train)

prediction_open = reg_model.predict(x_test)

prediction_closed = npy.zeros(features_subset_closed.shape[0])

prediction = npy.append(prediction_open, prediction_closed)

y_test = npy.append(y_test_open, npy.zeros(features_subset_closed.shape[0]))

print("r2_score: ",r2_score(y_test,prediction))
print("Mean absolute error: %.2f" % mean_absolute_error(y_test,prediction))
print("Root mean squared error: ", math.sqrt(mean_squared_error(y_test,prediction)))
```

```
r2_score:  0.7746971591468499
Mean absolute error: 997.04
Root mean squared error:  1940.7729287746029
```

In [ ]:

```
pt.figure(figsize=(10,10))
pt.scatter(y_test,prediction, c='crimson')

p1 = max(max(prediction), max(y_test))
p2 = min(min(prediction), min(y_test))
pt.plot([p1, p2], [p1, p2], 'b-')
pt.xlabel('True Values', fontsize=15)
pt.ylabel('Predictions', fontsize=15)
pt.axis('equal')
pt.show()
```





SGD Regressor

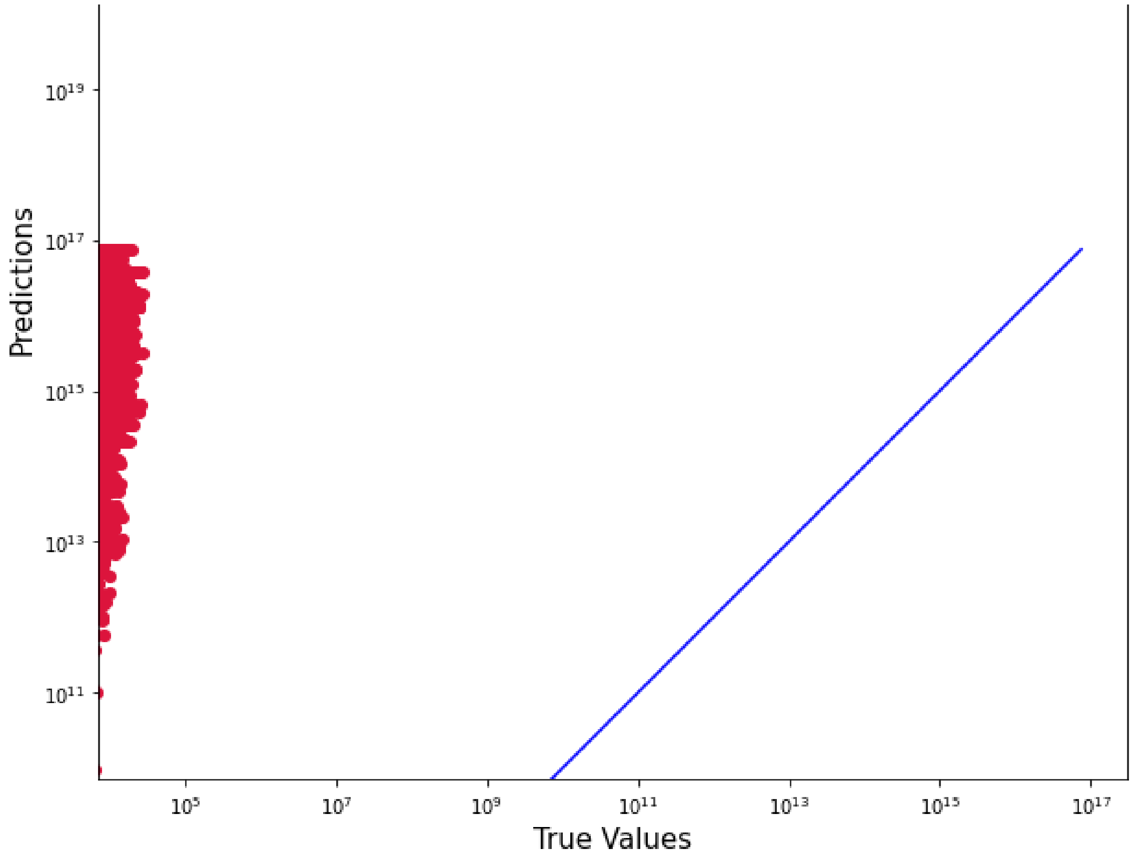
Stochastic Gradient Descent (SGD) is a simple yet efficient optimization algorithm used to find the values of parameters/coefficients of functions that minimize a cost function.

In [ ]:

```
from sklearn.linear_model import SGDRegressor
sgd_regressor_model = SGDRegressor(max_iter=2)  # increasing this value leads to over fit
ting
sgd_regressor_model.fit(x_train,y_train)

prediction_open = sgd_regressor_model.predict(x_test)
prediction_closed = npy.zeros(features_subset_closed.shape[0])

prediction = npy.append(prediction_open, prediction_closed)

y_test = npy.append(y_test_open, npy.zeros(features_subset_closed.shape[0]))

print("r2_score: ",r2_score(y_test,prediction))
print("Mean absolute error: %.2f" % mean_absolute_error(y_test,prediction))
print("Root mean squared error: ", math.sqrt(mean_squared_error(y_test,prediction)))

pt.figure(figsize=(10,10))
pt.scatter(y_test,prediction, c='crimson')
pt.yscale('log')
pt.xscale('log')

p1 = max(max(prediction), max(y_test))
p2 = min(min(prediction), min(y_test))
pt.plot([p1, p2], [p1, p2], 'b-')
pt.xlabel('True Values', fontsize=15)
pt.ylabel('Predictions', fontsize=15)
pt.axis('equal')
pt.show()
```

```
/usr/local/lib/python3.9/site-packages/sklearn/linear_model/_stochastic_gradient.py:1220: ConvergenceWarning: Maximum number of iteration reached before convergence. Consider incr
easing max_iter to improve the fit.
  warnings.warn("Maximum number of iteration reached before "
```

```
r2_score:  -2.0375262233567603e+24
Mean absolute error: 2439886158694418.00
Root mean squared error:  5836372403407445.0
```

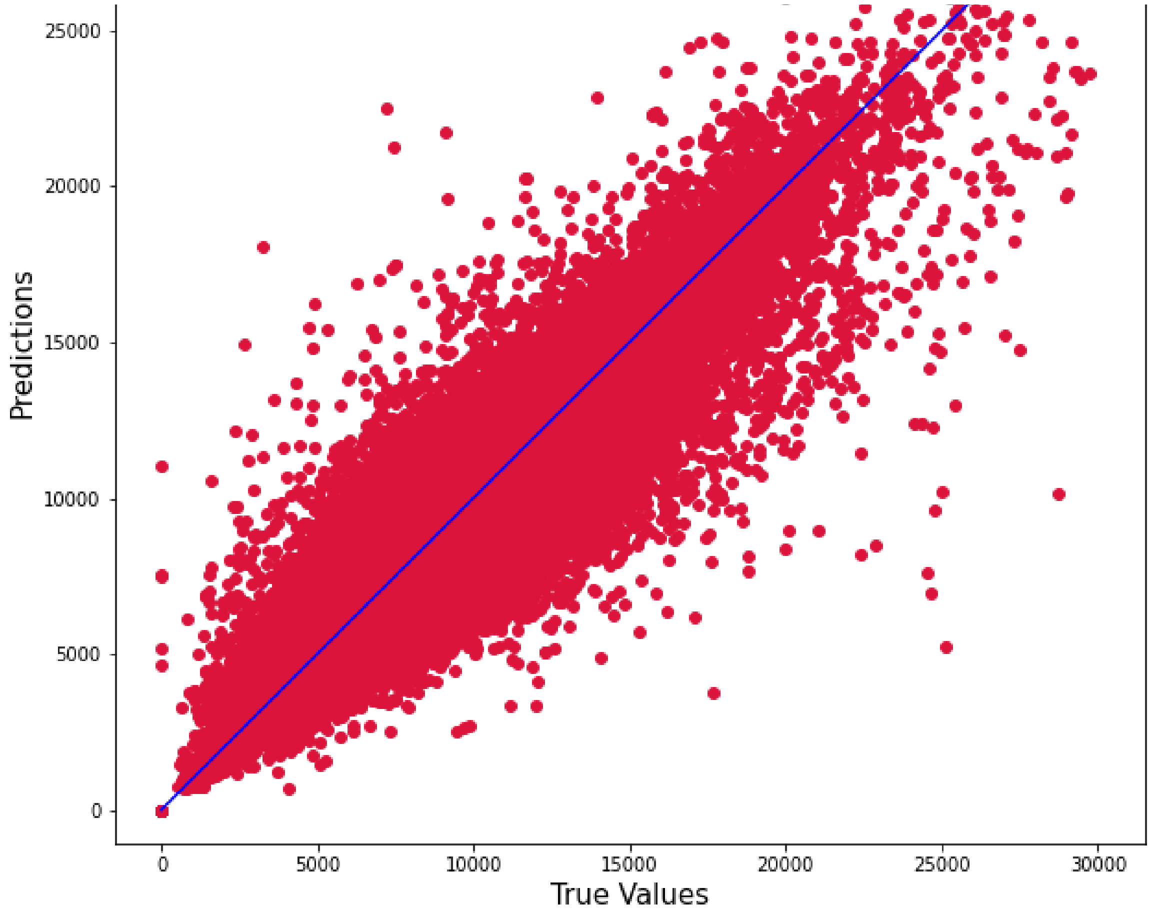


Random Forest Regressor

In [ ]:

```
from sklearn.ensemble import RandomForestRegressor

random_forest_regressor_model = RandomForestRegressor()
random_forest_regressor_model.fit(x_train,y_train)

prediction_open = random_forest_regressor_model.predict(x_test)
prediction_closed = npy.zeros(features_subset_closed.shape[0])

prediction = npy.append(prediction_open, prediction_closed)

y_test = npy.append(y_test_open, npy.zeros(features_subset_closed.shape[0]))

print("r2_score: ",r2_score(y_test,prediction))
print("Mean absolute error: %.2f" % mean_absolute_error(y_test,prediction))
print("Root mean squared error: ", math.sqrt(mean_squared_error(y_test,prediction)))


pt.figure(figsize=(10,10))
pt.scatter(y_test,prediction, c='crimson')

p1 = max(max(prediction), max(y_test))
p2 = min(min(prediction), min(y_test))
pt.plot([p1, p2], [p1, p2], 'b-')
pt.xlabel('True Values', fontsize=15)
pt.ylabel('Predictions', fontsize=15)
pt.axis('equal')
pt.show()
```

```
r2_score:  0.965139516105973
Mean absolute error: 356.90
Root mean squared error:  763.4104445628506
```

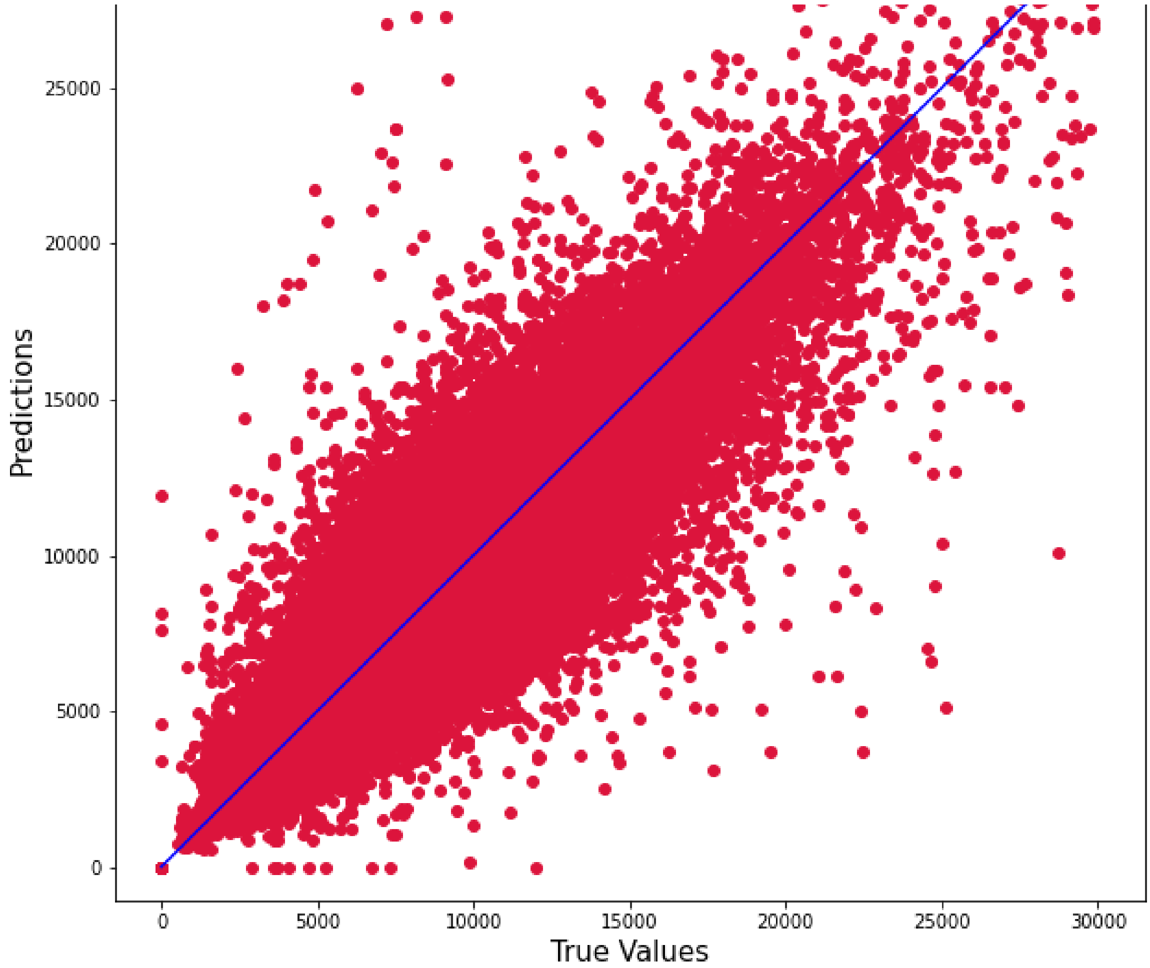


Decision Tree Regressor

In [ ]:

```
from sklearn.tree import DecisionTreeRegressor

decision_tree_regressor_model = DecisionTreeRegressor()
decision_tree_regressor_model.fit(x_train,y_train)

prediction_open = decision_tree_regressor_model.predict(x_test)
prediction_closed = npy.zeros(features_subset_closed.shape[0])

prediction = npy.append(prediction_open, prediction_closed)

y_test = npy.append(y_test_open, npy.zeros(features_subset_closed.shape[0]))

print("r2_score: ",r2_score(y_test,prediction))
print("Mean absolute error: %.2f" % mean_absolute_error(y_test,prediction))
print("Root mean squared error: ", math.sqrt(mean_squared_error(y_test,prediction)))


pt.figure(figsize=(10,10))
pt.scatter(y_test,prediction, c='crimson')

p1 = max(max(prediction), max(y_test))
p2 = min(min(prediction), min(y_test))
pt.plot([p1, p2], [p1, p2], 'b-')
pt.xlabel('True Values', fontsize=15)
pt.ylabel('Predictions', fontsize=15)
pt.axis('equal')
pt.show()
```

```
r2_score:  0.9513412691589237
Mean absolute error: 421.12
Root mean squared error:  901.9278146369298
```



Random Forest Regressor had the lowest error as compared to other stores that means it is better at predicting sales than other models so we have selected that as our model

Understanding the important features

In [ ]:

```
# determining the  weights of all the features used in the data
feature_importance = random_forest_regressor_model.feature_importances_
feature_importance
```

Out[ ]:

```
array([0.17975614, 0.08158972, 0.13989285, 0.00119411, 0.01263475,
       0.03416831, 0.03026641, 0.20963526, 0.07001563, 0.07440272,
       0.00335478, 0.02902703, 0.03388058, 0.01311974, 0.02398712,
       0.06307485])
```

In [ ]:

```
# features used
columns = list(x_train.columns)
columns
```

Out[ ]:

```
['Store',
 'DayOfWeek',
 'Promo',
 'StateHoliday',
 'SchoolHoliday',
 'StoreType',
 'Assortment',
 'CompetitionDistance',
 'CompetitionOpenSinceMonth',
 'CompetitionOpenSinceYear',
 'Promo2',
 'Promo2SinceWeek'
```

```
 'Promo2SinceWeek',
 'Promo2SinceYear',
 'PromoInterval',
 'Year',
 'Month']
```

In [ ]:

```
#rounding the feature important  values upto 5 digits
feature_importance_value = []
for i in range(len(feature_importance)):
    feature_importance_value.append(round(feature_importance[i],5))
feature_importance_value
```

Out[ ]:

```
[0.17976,
 0.08159,
 0.13989,
 0.00119,
 0.01263,
 0.03417,
 0.03027,
 0.20964,
 0.07002,
 0.0744,
 0.00335,
 0.02903,
 0.03388,
 0.01312,
 0.02399,
 0.06307]
```

In [ ]:

```
# creating dataframe of columns and their respective feature important values
feature_importance_df = pds.DataFrame({"Features":columns,
                                       "Values":feature_importance_value })
feature_importance_df
```

Out[ ]:

	Features	Values
0	Store	0.17976
1	DayOfWeek	0.08159
2	Promo	0.13989
3	StateHoliday	0.00119
4	SchoolHoliday	0.01263
5	StoreType	0.03417
6	Assortment	0.03027
7	CompetitionDistance	0.20964
8	CompetitionOpenSinceMonth	0.07002
9	CompetitionOpenSinceYear	0.07440
10	Promo2	0.00335
11	Promo2SinceWeek	0.02903
12	Promo2SinceYear	0.03388
13	PromoInterval	0.01312
14	Year	0.02399
15	Month	0.06307

In [ ]:

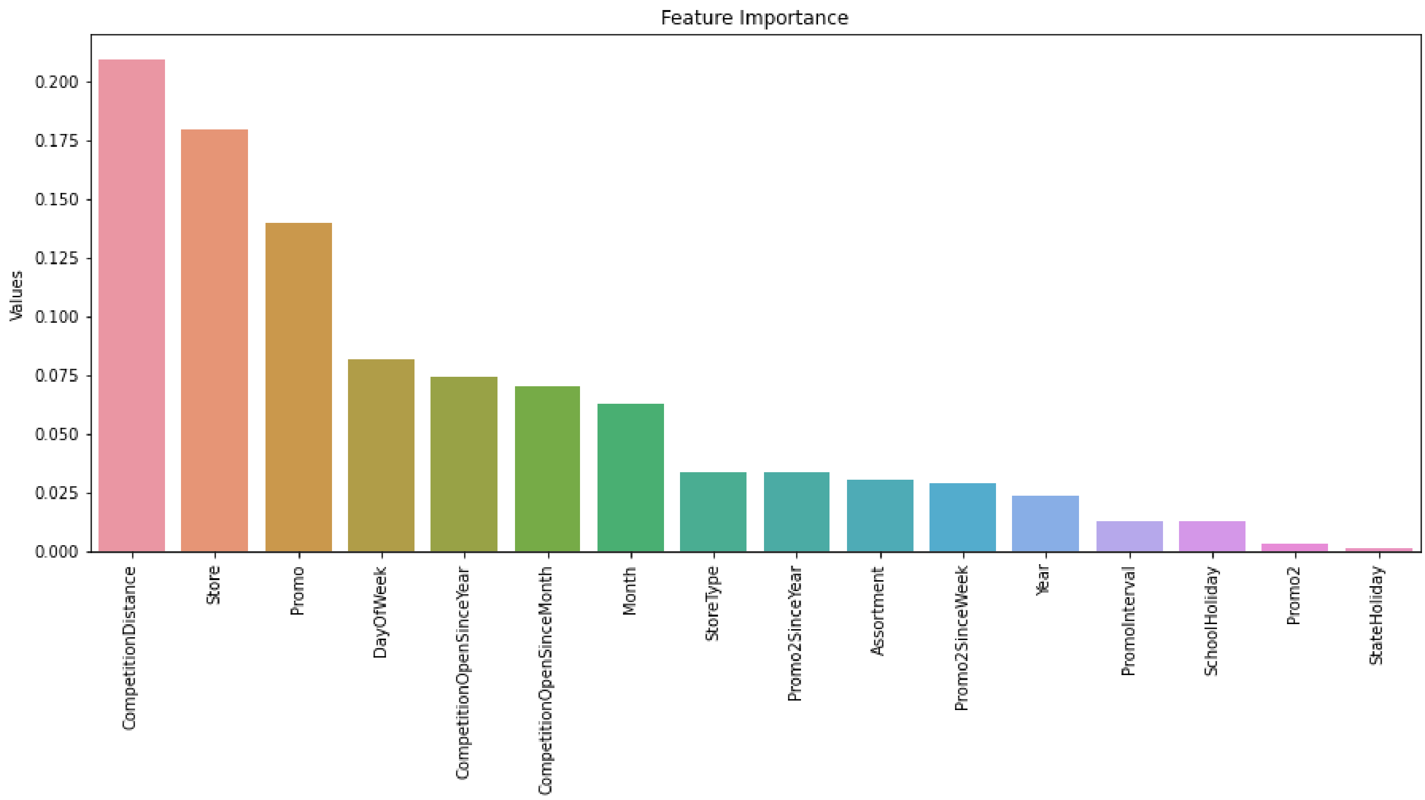
```
#sorting variables with respective contribution while determining the target variable
feature_importance_df.sort_values(by=["Values"], inplace=True, ascending=False)
feature_importance_df
```

Out[ ]:

	Features	Values
7	CompetitionDistance	0.20964
0	Store	0.17976
2	Promo	0.13989
1	DayOfWeek	0.08159
9	CompetitionOpenSinceYear	0.07440
8	CompetitionOpenSinceMonth	0.07002
15	Month	0.06307
5	StoreType	0.03417
12	Promo2SinceYear	0.03388
6	Assortment	0.03027
11	Promo2SinceWeek	0.02903
14	Year	0.02399
13	PromoInterval	0.01312
4	SchoolHoliday	0.01263
10	Promo2	0.00335
3	StateHoliday	0.00119

In [ ]:

```
#visualization of  Feature Importance[higher values have huge contribution while determin
ing the target variable
pt.figure(figsize=(15,6))

sea.barplot(x=feature_importance_df['Features'], y=feature_importance_df['Values'],
            data = feature_importance_df ).set(title='Feature Importance')

pt.xticks(rotation=90)
pt.show()
```



In [ ]:

```
#inights from the data
1.competion distance,type of store and promo can be used to boost the sales by implement
ing appropriate measures
2.feature engineering helps to make better predictions by removing low feature importanc
e variables as they dont have much contribution
3.Store Type affects the sales
4.Promo code can help increase in the competition and lead to more sales
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: