

✓ **Project Summary -**

There are two datasets, Rossmann store dataset and the store data set. Rossmann store dataset contains 1017209 rows and 9 columns without duplicate data and null values. Store dataset contains 1115 rows and 10 columns without duplicate data and some null values in columns CompetitionDistance, CompetitionOpenSinceMonth,CompetitionOpenSinceYear, Promo2SinceWeek, Promo2SinceYear, PromoInterval. After treating the null values and doing feature engineering we got below results.

In this project, a Random Forest machine learning model has been selected as the final model for predicting sales. Here's a brief summary of the key findings and the project's status:

Model Selection:

The Random Forest machine learning model has been chosen as the final predictive model due to its outstanding performance. The model achieved a high training accuracy of 99.60%, indicating a strong fit to the training data. The model's test accuracy is 97.26%, which suggests that it performs exceptionally well on unseen data, demonstrating its reliability for future predictions.

Root Mean Squared Error (RMSE):

The root mean squared error is impressively low, with a value of 516.18, indicating that the model's predictions are very close to the actual sales values.

Project Objective:

The primary goal of this project is to build a predictive model capable of forecasting sales. By achieving high accuracy and low root mean squared error, the Random Forest model has demonstrated its potential to provide accurate sales predictions, which can be valuable for optimizing inventory, staffing, and other operational decisions for the Rossmann stores in the future.

✓ **Problem Statement**

Rossmann operates a network of more than 3,000 drug stores across seven European countries. At present, store managers at Rossmann are responsible for forecasting their daily sales for a period of up to six weeks in advance. These sales predictions are affected by a multitude of factors, such as promotional activities, competitive factors, school and public holidays, seasonal patterns, and the specific location of each store. Given the diverse set of conditions that each store manager faces, the accuracy of these sales predictions can vary significantly.

Data Description

rossman.csv - historical data including sales

store.csv - supplimental information about the store

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

✓ **Import Libraries**

```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pylab
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import math
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
```

Dataset Loading

```
rossmann_df=pd.read_csv('/content/drive/MyDrive/Datasets/rossmann.csv')
store_df=pd.read_csv('/content/drive/MyDrive/Datasets/store.csv')
```

Dataset First View

rossmann_df

	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

store_df

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2	Promo2SinceWeek	Pi
	0	1	c	a	1270.0	9.0	2008.0	0	NaN
	1	2	a	a	570.0	11.0	2007.0	1	13.0
	2	3	a	a	14130.0	12.0	2006.0	1	14.0
	3	4	c	c	620.0	9.0	2009.0	0	NaN
	4	5	a	a	29910.0	4.0	2015.0	0	NaN

	1110	1111	a	a	1900.0	6.0	2014.0	1	31.0
	1111	1112	c	c	1880.0	4.0	2006.0	0	NaN
	1112	1113	a	c	9260.0	NaN	NaN	0	NaN
	1113	1114	a	c	870.0	NaN	NaN	0	NaN
	1114	1115	d	c	5350.0	NaN	NaN	1	22.0

1115 rows × 10 columns

rossmann_df.describe()


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

store_df.describe()

	Store	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2	Promo2SinceWeek	Promo2SinceYear
count	1115.00000	1112.000000	761.000000	761.000000	1115.000000	571.000000	571.000000
mean	558.00000	5404.901079	7.224704	2008.668857	0.512108	23.595447	2011.763570
std	322.01708	7663.174720	3.212348	6.195983	0.500078	14.141984	1.674930
min	1.00000	20.000000	1.000000	1900.000000	0.000000	1.000000	2009.000000
25%	279.50000	717.500000	4.000000	2006.000000	0.000000	13.000000	2011.000000
50%	558.00000	2325.000000	8.000000	2010.000000	1.000000	22.000000	2012.000000
75%	836.50000	6882.500000	10.000000	2013.000000	1.000000	37.000000	2013.000000
max	1115.00000	75860.000000	12.000000	2015.000000	1.000000	50.000000	2015.000000

Data cleaning

```
#let us see the null rows present the column
store_df.isnull().sum()
```

 Store

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

So, we can fill these null values with many ways such as 0 or mean or mode or median. We decided to fill it with Mean.

```
#Replacing Nan values in CompetitionDistance with mean distance.
store_df['CompetitionDistance'].fillna(store_df['CompetitionDistance'].mean(), inplace = True)
```

As CompetitionOpenSinceMonth, CompetitionOpenSinceYear, Promo2SinceWeek Promo2SinceYear, PromoInterval contains null values and we we do not have much information about them, we will fill it with 0 and PromoInterval with none as it is object type.

```
#code for replacing Nan values with 0 for int datatype columns and none for obj datatype column.
```

```
store_new = store_df.copy()
```

```
#Replacing Nan values with 0 in CompetitionOpenSinceMonth
store_new['CompetitionOpenSinceMonth'] = store_new['CompetitionOpenSinceMonth'].fillna(0)
```


```
#Replacing Nan values with 0 in CompetitionOpenSinceYear
store_new['CompetitionOpenSinceYear'] = store_new['CompetitionOpenSinceYear'].fillna(0)
```

```
#Replacing Nan values with 0 in Promo2SinceWeek
store_new['Promo2SinceWeek'] = store_new['Promo2SinceWeek'].fillna(0)
```

```
#Replacing Nan values with 0 in Promo2SinceYear
store_new['Promo2SinceYear'] = store_new['Promo2SinceYear'].fillna(0)
```


```
#Replacing Nan values with None in PromoInterval
store_new['PromoInterval'] = store_new['PromoInterval'].fillna('None')
```

```
#Let us check the null values after imputation
store_new.isnull().sum()
```

 Store

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

```
#code for changing StateHoliday dtype from object to int.
rossmann_new = rossmann_df.copy()
#Replacing 0 representing no holiday with 0 and a, b, c with 1 representing different holidays
rossmann_new["StateHoliday"] = rossmann_df["StateHoliday"].replace({"0": 0, "a": 1, "b": 1, "c": 1})
rossmann_new['StateHoliday'] = rossmann_new['StateHoliday'].astype(int)
rossmann_new[['StateHoliday']].value_counts()
```


 StateHoliday

0	986159
1	31050
Name:	count, dtype: int64

```
#code for changing Date dtype from object to datetime.
rossmann_new['Date'] = pd.to_datetime(rossmann_new['Date'])
```

```
#code for changing CompetitionOpenSinceYear, CompetitionOpenSinceMonth and Promo2sinceYear dtype from object to int.
store_new['CompetitionOpenSinceYear']= store_new['CompetitionOpenSinceYear'].astype(int)
store_new['CompetitionOpenSinceMonth'] = store_new['CompetitionOpenSinceMonth'].astype(int)
store_new['Promo2SinceYear']= store_new['Promo2SinceYear'].astype(int)
```

```
store_new.dtypes
```

 Store

Store	int64
StoreType	object
Assortment	object
CompetitionDistance	float64
CompetitionOpenSinceMonth	int64
CompetitionOpenSinceYear	int64
Promo2	int64
Promo2SinceWeek	float64
Promo2SinceYear	int64
PromoInterval	object
dtype:	object

Merge the Rossmann_new and Store_new by column 'Store' as it is common in both dataframes

```
rossmann_store = pd.merge(rossmann_new, store_new, on='Store', how='left')
rossmann_store.head(5)
```

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	StoreType	Assortment	CompetitionDistance	Compe
0	1	5	2015-07-31	5263	555	1	1	0	1	c	a	1270.0	
1	2	5	2015-07-31	6064	625	1	1	0	1	a	a	570.0	
2	3	5	2015-07-31	8314	821	1	1	0	1	a	a	14130.0	
3	4	5	2015-07-31	13995	1498	1	1	0	1	c	c	620.0	
4	5	5	2015-07-31	4822	559	1	1	0	1	a	a	29910.0	

```
rossmann_store.tail(5)
```

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	StoreType	Assortment	CompetitionDistance	
1017204	1111	2	2013-01-01	0	0	0	0	1	1	a	a	1900.0	
1017205	1112	2	2013-01-01	0	0	0	0	1	1	c	c	1880.0	
1017206	1113	2	2013-01-01	0	0	0	0	1	1	a	c	9260.0	
1017207	1114	2	2013-01-01	0	0	0	0	1	1	a	c	870.0	
1017208	1115	2	2013-01-01	0	0	0	0	1	1	d	c	5350.0	

```
rossmann_store.shape
```

(1017209, 18)

```
#Gives the month as January=1, December=12 from Date column
rossmann_store['Month'] = pd.DatetimeIndex(rossmann_store['Date']).month
rossmann_store
```

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	StoreType	Assortment	CompetitionDistance	
0	1	5	2015-07-31	5263	555	1	1	0	1	c	a	1270.0	
1	2	5	2015-07-31	6064	625	1	1	0	1	a	a	570.0	
2	3	5	2015-07-31	8314	821	1	1	0	1	a	a	14130.0	
3	4	5	2015-07-31	13995	1498	1	1	0	1	c	c	620.0	
4	5	5	2015-07-31	4822	559	1	1	0	1	a	a	29910.0	
...	
1017204	1111	2	2013-01-01	0	0	0	0	1	1	a	a	1900.0	
1017205	1112	2	2013-01-01	0	0	0	0	1	1	c	c	1880.0	
1017206	1113	2	2013-01-01	0	0	0	0	1	1	a	c	9260.0	
1017207	1114	2	2013-01-01	0	0	0	0	1	1	a	c	870.0	
1017208	1115	2	2013-01-01	0	0	0	0	1	1	d	c	5350.0	

1017209 rows × 19 columns

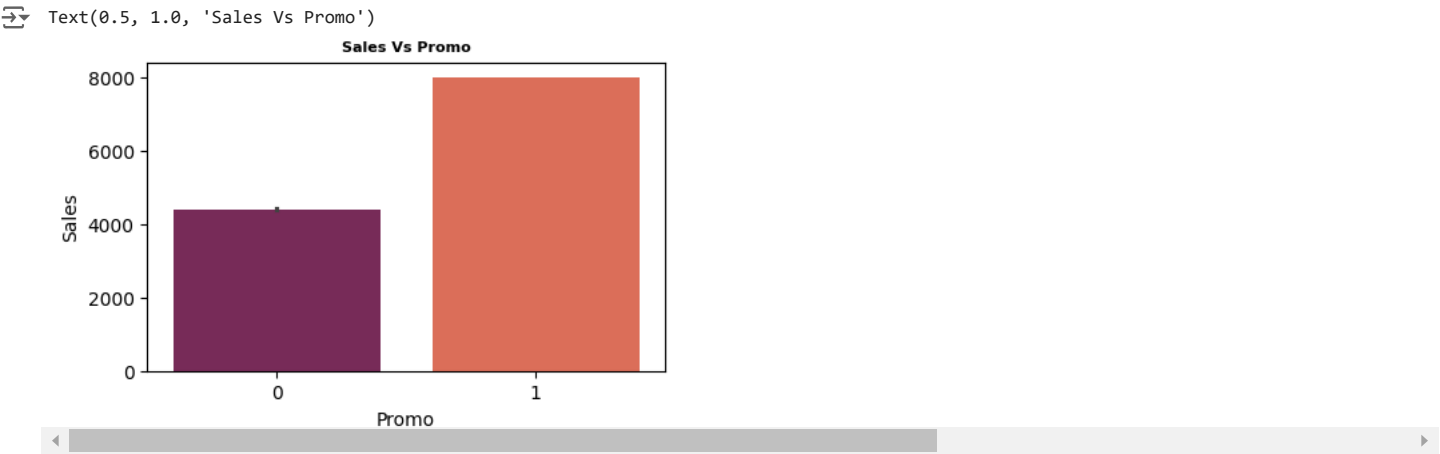
✓

Data Vizualization, Storytelling & Experimenting with charts : Understand the relationships between variables

Chart-1

Does promotion have positive effect on sales?

```
ax=plt.figure(figsize=(5,3))
#Code for poltting bar plot
ax = sns.barplot(data=rossmann_store, x='Promo', y='Sales', palette='rocket')
ax.set_title("Sales Vs Promo",fontsize=8,fontweight='bold')
```

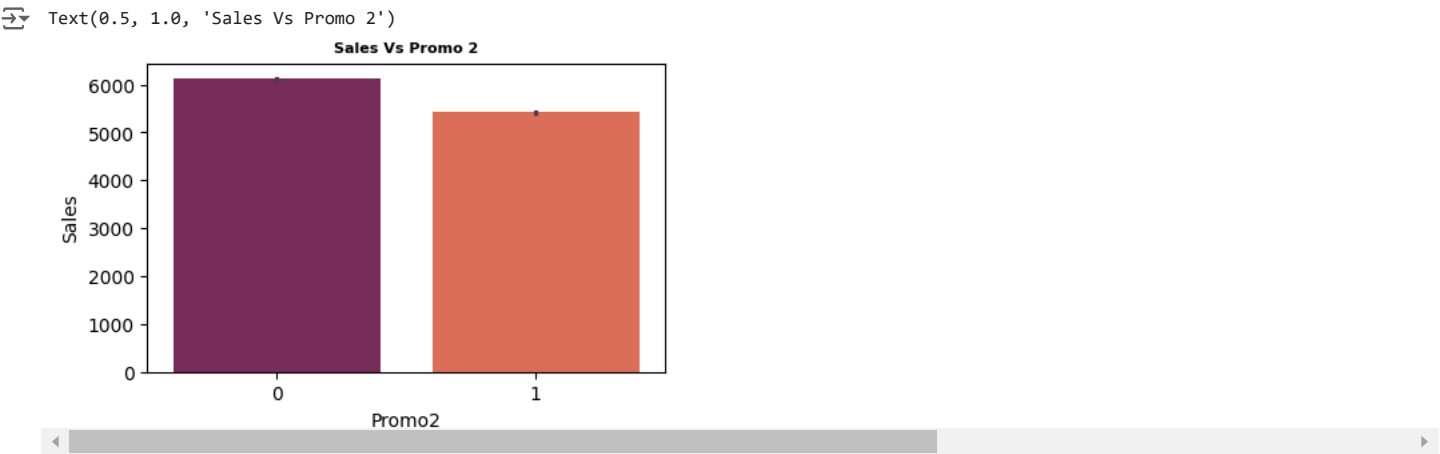


Barplot between promo and Sales shows the effect of promotion on Sales. Here 0 represents the store which didnt opt for promotion and 1 represents for stores who opt for promotion. Those store who did promotions their sales are high as compared to stores who didn't took promotion.

Chart-2

Does promotion 2 have positive effect on sales?

```
ax=plt.figure(figsize=(5,3))
ax = sns.barplot(data=rossmann_store, x='Promo2', y='Sales',palette='rocket')
ax.set_title("Sales Vs Promo 2",fontsize=8,fontweight='bold')
```

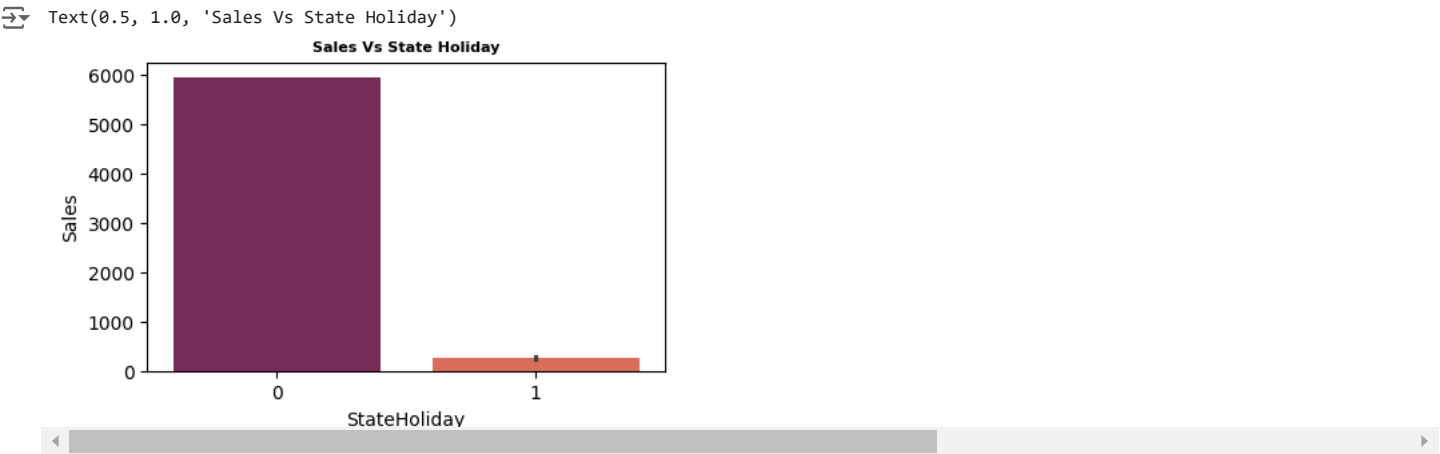


Barplot between promo2 and Sales shows the effect of consecutive promotion on Sales. Here 0 represents the store which didnt opt for promotion and 1 represents for stores who opt for promotion. There is not much effect on sales after promo2

Chart-3

Does state holidays affect on overall sales

```
ax=plt.figure(figsize=(5,3))
ax = sns.barplot(data=rossmann_store, x='StateHoliday', y='Sales',palette='rocket')
ax.set_title("Sales Vs State Holiday",fontsize=8,fontweight='bold')
```

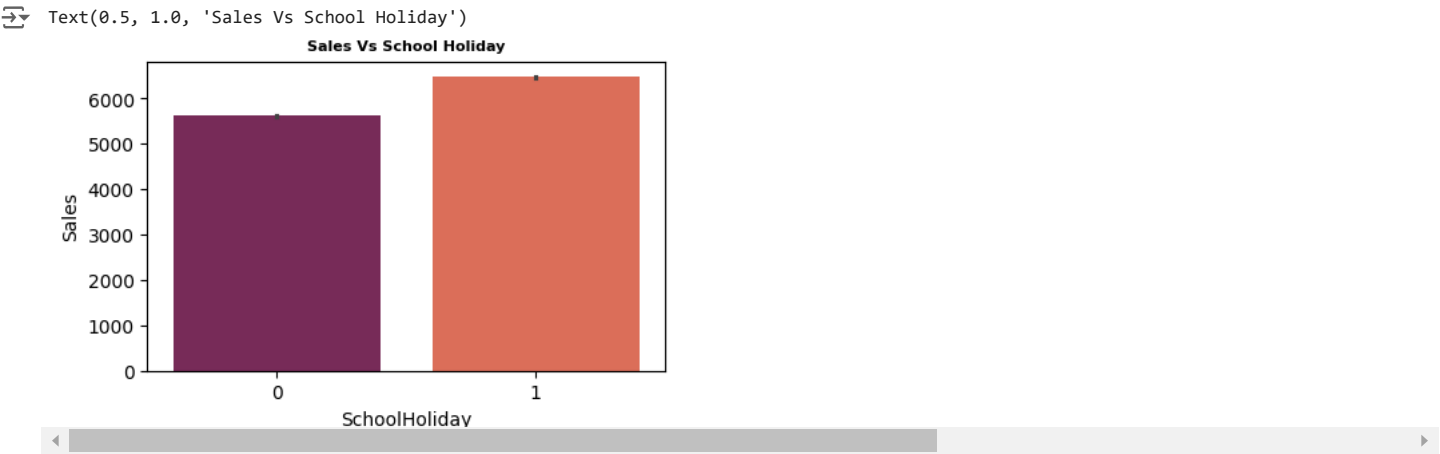


On holidays the sales were quite low as compare to the working days as the most of the stores were closed.

Chart - 4

Does school holiday affects the overall sales

```
ax=plt.figure(figsize=(5,3))
ax = sns.barplot(data=rossmann_store, x='SchoolHoliday', y='Sales',palette='rocket')
ax.set_title("Sales Vs School Holiday",fontsize=8,fontweight='bold')
```



But it is interesting to note that the number of stores opened during School Holidays were more than that were opened during State Holidays. Another important thing to note is that the stores which were opened during School holidays had more sales than normal.

Chart - 5

What was the sales with repective different assortments?

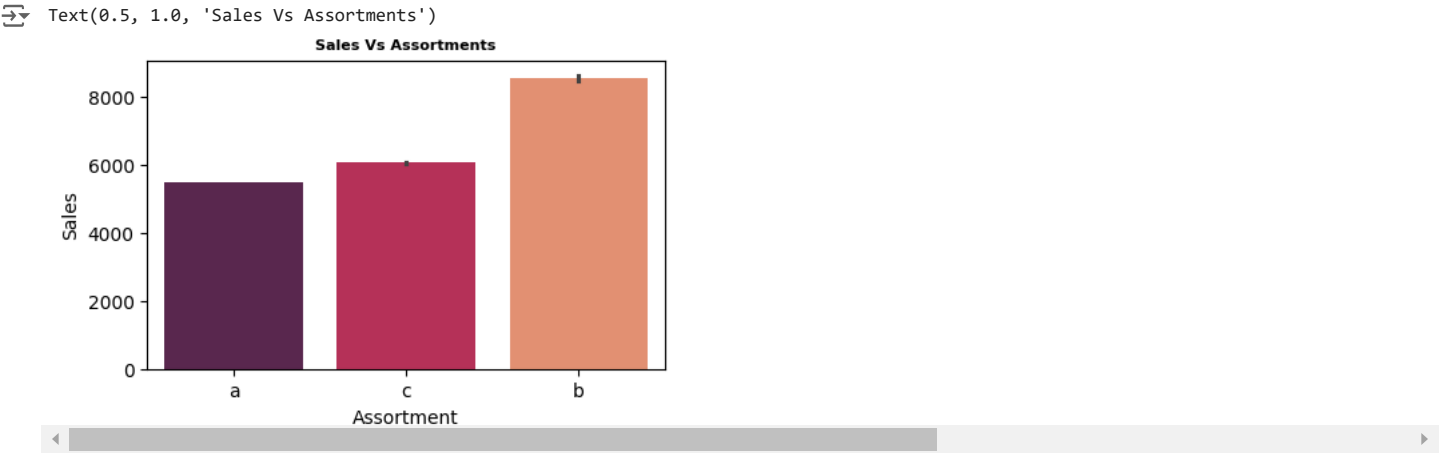
Here a, b and c represents

a means basic things

b means extra things

c means extended things so the highest variety of products.

```
ax=plt.figure(figsize=(5,3))
ax = sns.barplot(data=rossmann_store, x='Assortment', y='Sales',palette='rocket')
ax.set_title("Sales Vs Assortments",fontsize=8,fontweight='bold')
```



Here as we can observe that assortment b had most of the sales followed by the c. Assortment a had the lowest sale may be due to the reason less varities.

Chart-6

How sales varies for different store type?

```
ax=plt.figure(figsize=(5,3))
ax= sns.barplot(data=rossmann_store, x='StoreType', y='Sales',palette='rocket')
ax.set_title("Amount of sale per store type",fontsize=10,fontweight='bold')
```

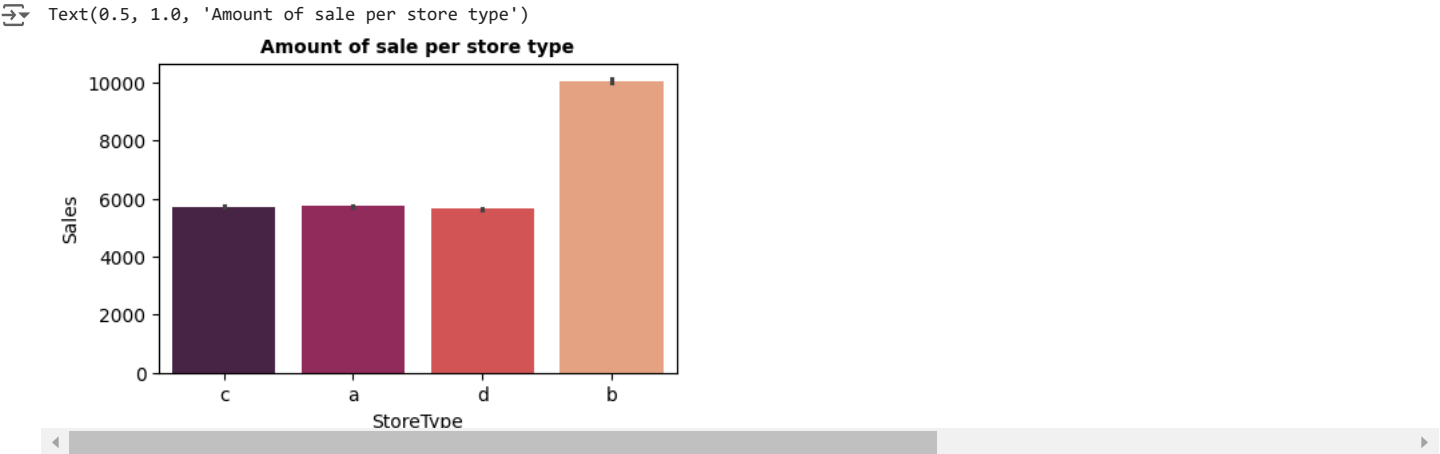

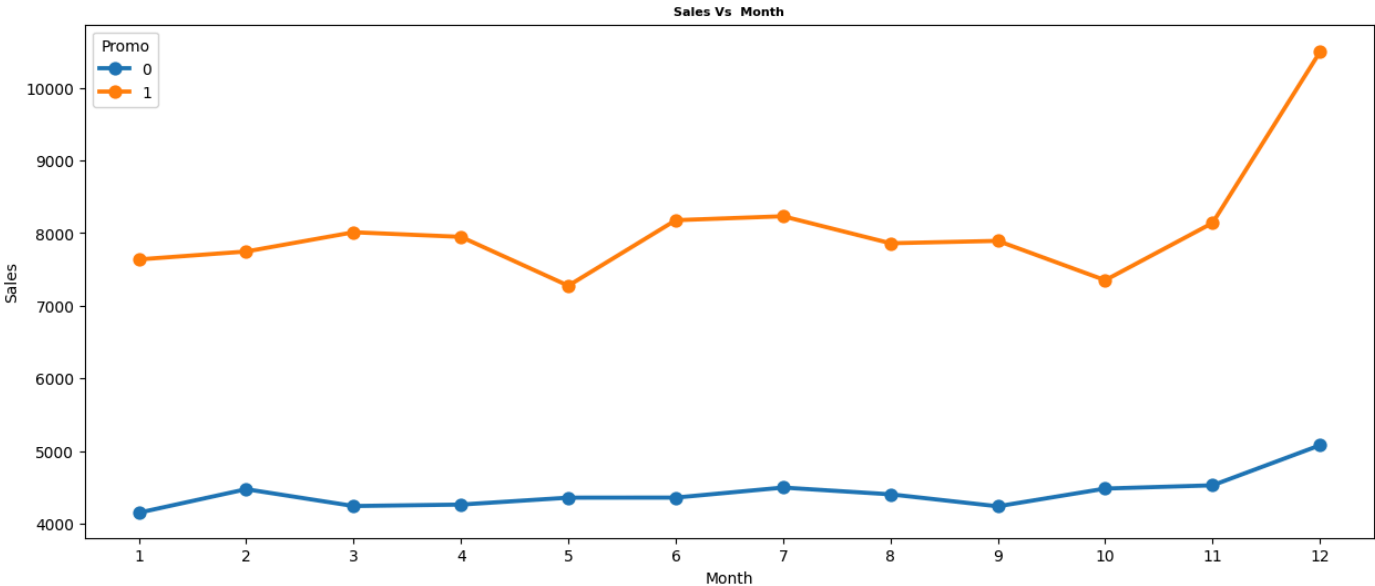


Chart - 7

What was the trend of sales in different months of year with and without promotions?

```
ax=plt.figure(figsize=(15,6))
ax = sns.pointplot(data=rossmann_store, x='Month', y='Sales', hue = 'Promo')
ax.set_title("Sales Vs Month",fontsize=8,fontweight='bold')
```

 Text(0.5, 1.0, 'Sales Vs Month')



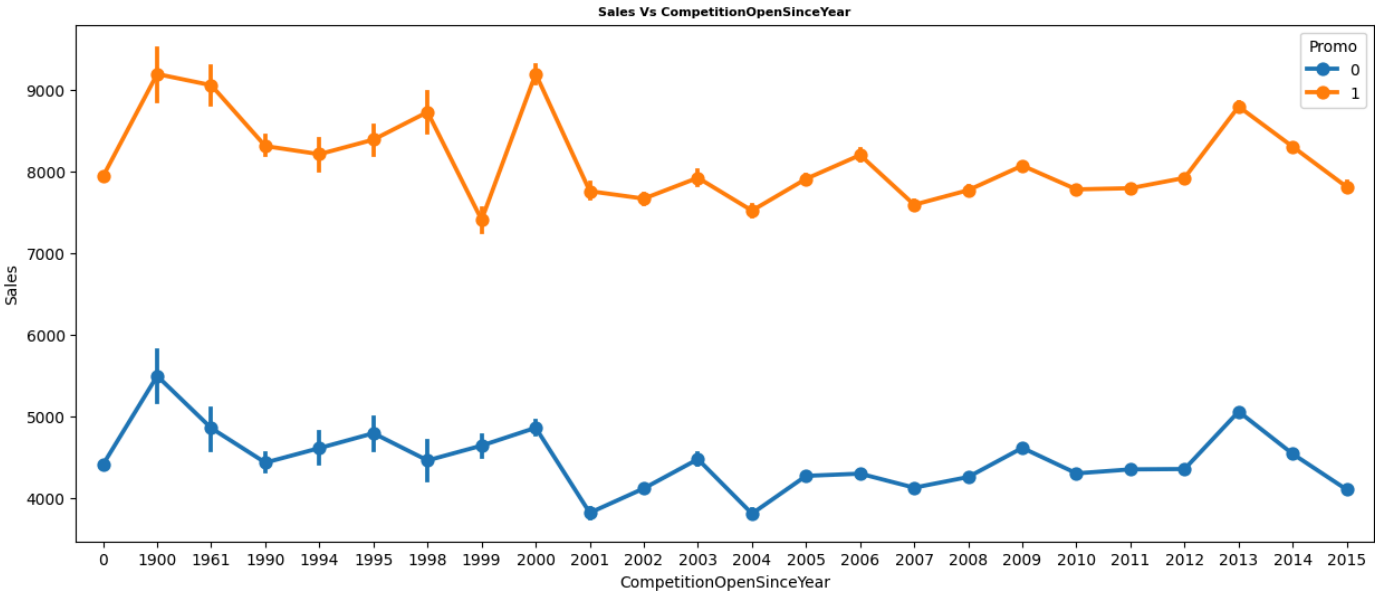
This data shows that sales after month november increases drastically both with promotion and without promotion although the promotion lead to increase the sales by almost 2 fold. This is very clear that in December month due to Christmas Eve and New year celebration everone is buying. So sales of Rossmann store is very high in December.

Chart-8

What was the trend of sales in different years with and without promotions?

```
ax=plt.figure(figsize=(15,6))
ax=sns.pointplot(data=rossmann_store, x='CompetitionOpenSinceYear', y='Sales',hue = 'Promo')
ax.set_title("Sales Vs CompetitionOpenSinceYear",fontsize=8,fontweight='bold')
```

 Text(0.5, 1.0, 'Sales Vs CompetitionOpenSinceYear')



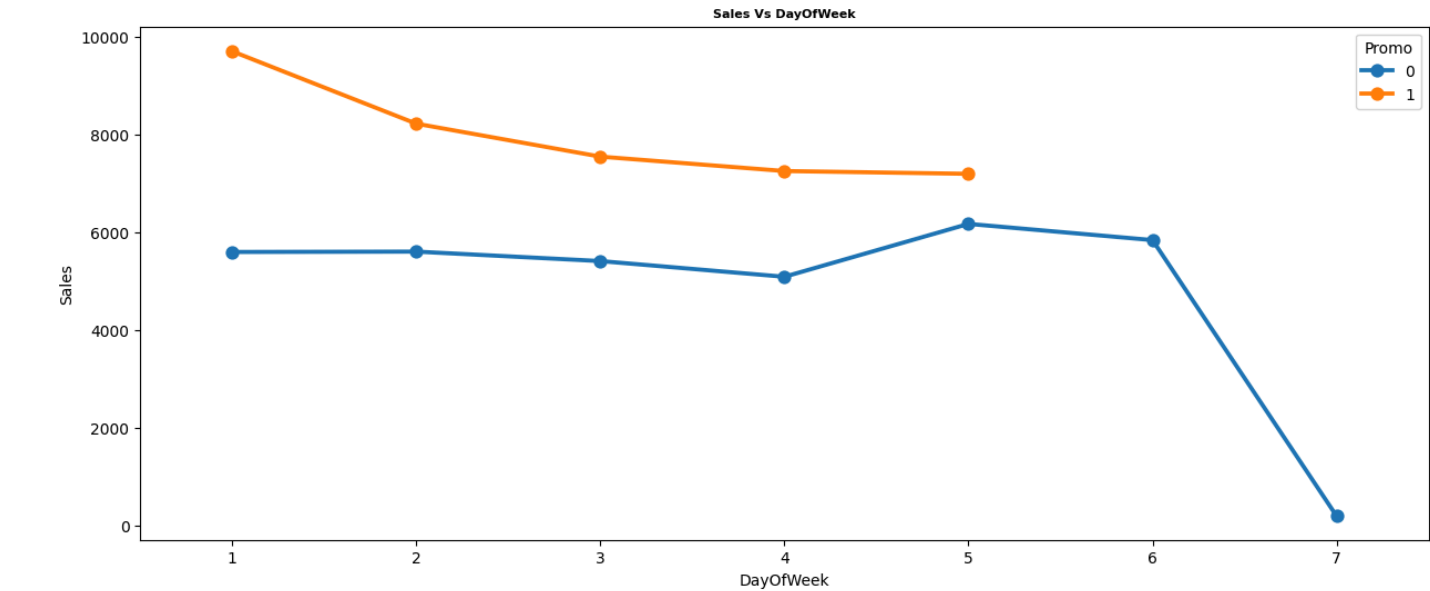
From the Plot we can tell that Sales are high during the year 1900, as there are limited number of stores so there is less competition and sales are high. But as year pass on number of stores increased that means competition also increased and this leads to decline in the sales.

Chart - 9

What was the trend of sales in different days of week with and without promotions?

```
ax=plt.figure(figsize=(15,6))
ax = sns.pointplot(data=rossmann_store, x='DayOfWeek', y='Sales', hue = 'Promo')
ax.set_title("Sales Vs DayOfWeek",fontsize=8,fontweight='bold')
```

Text(0.5, 1.0, 'Sales Vs DayOfWeek')



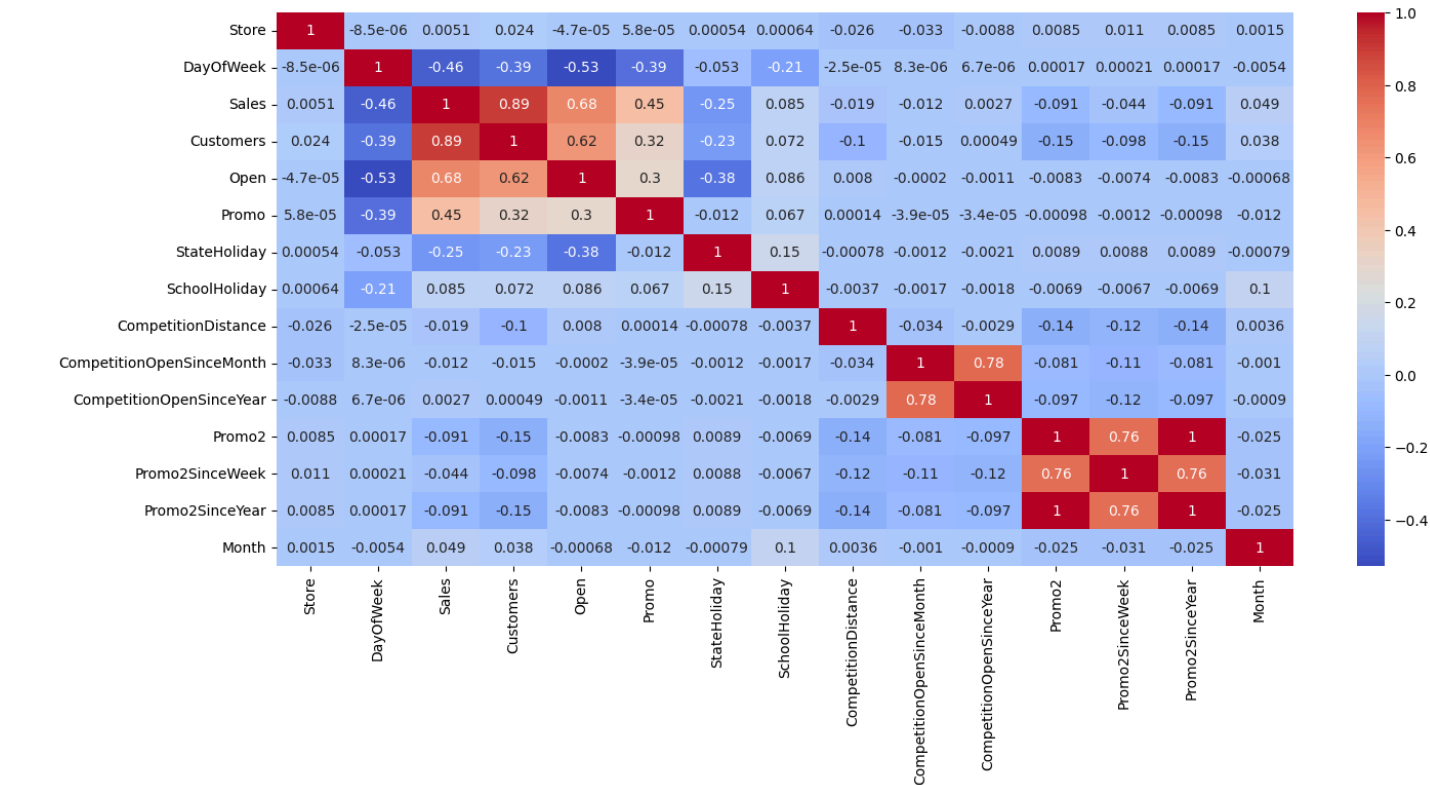
Plot between Sales and Days of week shows that maximum sales is on Monday with promotion it was more than normal and sales gradually decreasing to 6th day of week i.e. on Saturday. It also shows that sales on Sunday is almost near to zero as on sunday maximum stores are closed. Also no promotion were carry out on Saturdays.

Chart-10

Correlation Heatmap

```
plt.figure(figsize=(16,7))
numeric_columns =rossmann_store.select_dtypes(include=['number'])
corr_heat = numeric_columns.corr()
sns.heatmap(corr_heat, annot=True, cmap='coolwarm')
```

<Axes: >

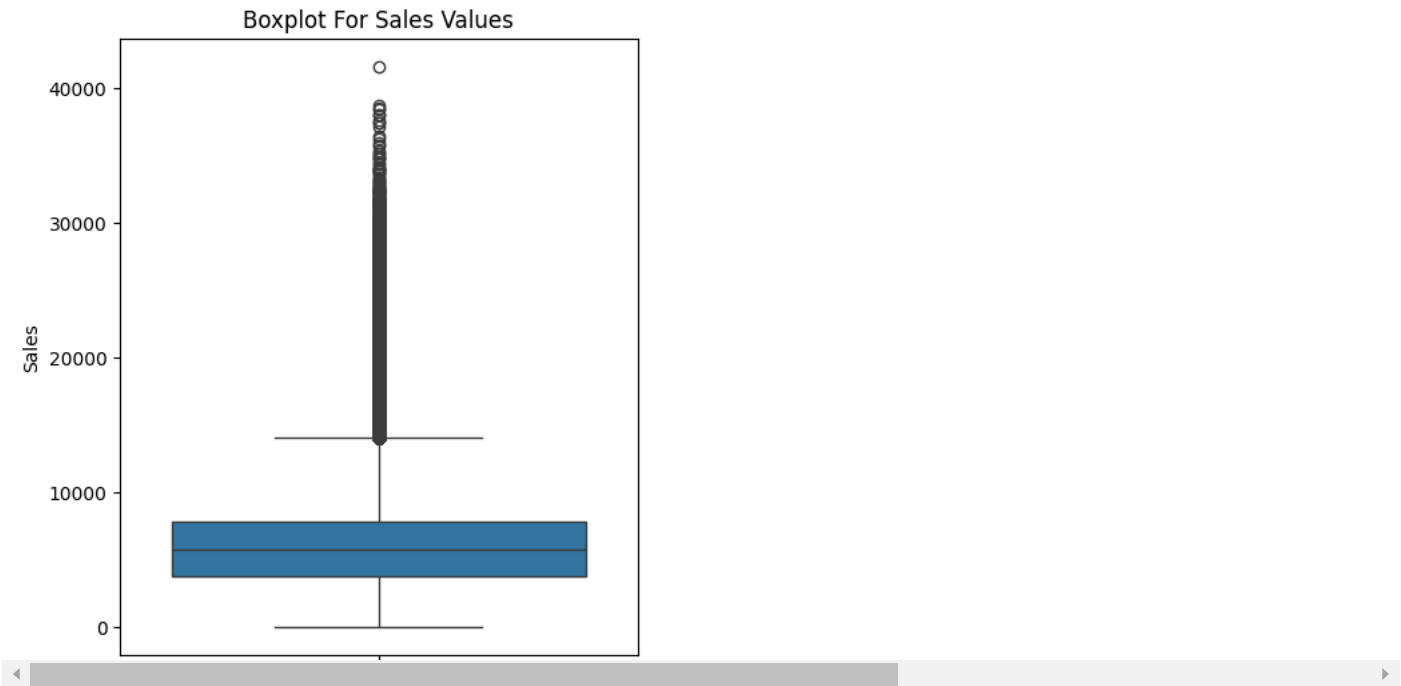


Handling Outliers

```
#checking outliers in sales
plt.figure(figsize=(5, 6))
x = sns.boxplot(rossmann_store['Sales'])
plt.title('Boxplot For Sales Values')
```



```
Text(0.5, 1.0, 'Boxplot For Sales Values')
```



As the high sales which leads to outlier could be the sell on the promotion day so let us not remove them and directly feed to our model.

Feature Manipulation & Selection

1. Feature Manipulation

As we know that closed stores have 0 sales, let us check how many stores are closed

```
# no of observations for closed stores with 0 sales
(rossmann_store[rossmann_store.Open == 0]).shape
```

```
(172817, 19)
```

```
(rossmann_store[rossmann_store.Open != 0]).shape
```

```
(844392, 19)
```

```
#The stores which are closed and with zero sales
rossmann_store[(rossmann_store.Open == 0) & (rossmann_store.Sales == 0)].count()[0]
```

```
172817
```

```
# Let us create new dataset by dropping closed stores with zero sale
rossmann_store_new = rossmann_store.drop(rossmann_store[(rossmann_store.Open == 0) & (rossmann_store.Sales == 0)].index)
```

2. Categorical Encoding

```
# Encode your categorical columns
df_new = pd.get_dummies(rossmann_store_new,columns=['StoreType','Assortment','PromoInterval']).astype(int)
df_new.head(1)
```

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	CompetitionDistance	...	StoreType
0	1	5	1438300800000000000	5263	555	1	1	0	1	1270	...	

1 rows x 27 columns

Data splitting

```
#Defining dependent variable
y = df_new['Sales']
#Defining independent variable
x = df_new.drop(columns=['Promo2SinceYear','Date','Open','Sales'])
```

```
#Splitting the data in train and test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 32 )
```

Data Scaling

```
#Scaling the data using standard scaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

ML MODEL-1

Linear Regression

```
#Fitting the model to training data
lr = LinearRegression()
lr.fit(x_train, y_train)
```

↗

LinearRegression

LinearRegression()

lr.coef_

```
array([-3.78350631e+01, -6.82996572e+01,  2.94853330e+03,  5.69912638e+02,
        7.29365032e+00,  1.15731643e+01,  1.95337545e+02, -1.95906530e+01,
        5.82549130e+01,  4.49300968e+14,  1.32336568e+02,  8.36986670e+01,
       -2.54226739e+15, -6.85421123e+14, -1.73685996e+15, -2.35213631e+15,
        5.51252341e+14,  1.08421952e+14,  5.50566794e+14,  2.32910523e+13,
        3.29110586e+13,  2.13942840e+13,  4.85675708e+14])
```

lr.intercept_

```
6952.858139197485
```

```
#Prediction on test data
y_pred_test = lr.predict(x_test)
y_pred_test
```

```
array([6017.6706392, 5496.2956392, 7473.4206392, ..., 6537.1081392,
       5091.1081392, 3264.1081392])
```

```
#training score
train_score1 = lr.score(x_train, y_train)
train_score1
```

```
0.8246026602810552
```

```
#Testing score
test_score1 = lr.score(x_test, y_test)
test_score1
```

```
0.8263153664926313
```

```
linear_Dataframe=pd.DataFrame(zip(y_test, y_pred_test), columns = ['actual', 'pred'])
linear_Dataframe
```

↗

	actual	pred
0	5800	6017.670639
1	5277	5496.295639
2	7569	7473.420639
3	19374	7632.733139
4	10640	7312.545639
...
168874	3539	3773.108139
168875	8844	7648.608139
168876	6421	6537.108139
168877	4599	5091.108139
168878	3215	3264.108139

168879 rows × 2 columns

Performance measure in regression

```
#Mean Absolute Error or MAE
MAE= mean_absolute_error(y_test,y_pred_test)
#Mean Squared Error or MSE
MSE= mean_squared_error(y_test,y_pred_test)
#Root Mean Squared Error or RMSE
RMSE= mean_squared_error(y_test,y_pred_test,squared=False)
#R2
R2_score= r2_score(y_test, y_pred_test)
#printing test results
print(f'Mean Absolute Error: {MAE}\n')
print(f'Mean Squared Error: {MSE}\n')
print(f'Root Mean Squared Error: {RMSE}\n')
print(f'R^2 score: {R2_score}\n')
```

```
Mean Absolute Error: 943.946039798739
```

Mean Squared Error: 1689268.3310192816

Root Mean Squared Error: 1299.7185583884234

R^2 score: 0.8263153664926313

▼ DECISION TREE

```
#Fitting the decision tree model on training data
decision_tree=DecisionTreeRegressor(max_depth=12)
decision_tree.fit(x_train, y_train)
#Prediction on test data
y_pred_dt = decision_tree.predict(x_test)

#training score
train_score2 = decision_tree.score(x_train,y_train)
train_score2

#test score
test_score2 = decision_tree.score(x_test,y_test)
test_score2

#Dataframe showing both the test data and predicted data
decisiontree_Dataframe = pd.DataFrame(zip(y_test, y_pred_dt), columns = ['actual', 'pred'])
decisiontree_Dataframe
```

	actual	pred
0	5800	6102.756147
1	5277	5219.647828
2	7569	7998.378434
3	19374	8258.163121
4	10640	9634.248175
...
168874	3539	3397.444444
168875	8844	8646.045936
168876	6421	6560.810778
168877	4599	4895.507608
168878	3215	3625.417802

168879 rows x 2 columns

```
#calculate metrics and print the results for test set
#Mean Absolute Error or MAE
MAE= mean_absolute_error(y_test,y_pred_dt)
#Mean Squared Error or MSE
MSE = mean_squared_error(y_test,y_pred_dt)
#Root Mean Squared Error or RMSE
RMSE= mean_squared_error(y_test,y_pred_dt,squared=False)
#R2 score
R2_score= r2_score(y_test, y_pred_dt)
#printing test results
print(f'Mean Absolute Error: {MAE}')
print(f'Mean Squared Error: {MSE}')
print(f'Root Mean Squared Error: {RMSE}')
print(f'R^2 score: {R2_score}')
```

Mean Absolute Error: 667.4864397812279
Mean Squared Error: 874221.5592287281
Root Mean Squared Error: 934.9981600135521
R^2 score: 0.9101156113977081

Hyper parameter tuning

```
#Takes time to run
# params = {
#     'max_depth':[4,8,12,16],
#     'min_samples_split':[2,3,5,7],
#     'min_samples_leaf':[6,8,10]
# }

# grid= GridSearchCV(decision_tree, params, scoring='neg_mean_squared_error', cv=5)

# grid.fit(x_train,y_train)

# grid.best_params_
# {'max_depth': 16, 'min_samples_leaf': 6, 'min_samples_split': 7}

#Fitting model with best parameters after grid search Cv
decision_tree = DecisionTreeRegressor(max_depth=16, min_samples_leaf= 6, min_samples_split= 7)
decision_tree.fit(x_train, y_train)
```

▼	DecisionTreeRegressor
DecisionTreeRegressor(max_depth=16, min_samples_leaf=6, min_samples_split=7)	

```
#Prediction on test data
y_pred_dtCV = decision_tree.predict(x_test)
```

```
train_score3 = decision_tree.score(x_train, y_train)
train_score3
```

0.9571250266542205

```
test_score3 = decision_tree.score(x_test, y_test)
test_score3
```

0.946565317772411

```
#Dataframe showing test data and predicted data
decisiontree_Dataframe = pd.DataFrame(zip(y_test, y_pred_dtCV), columns = ['actual', 'pred'])
decisiontree_Dataframe
```



	actual	pred
0	5800	6347.792553
1	5277	5201.628834
2	7569	8796.144681
3	19374	8228.833333
4	10640	10235.200000
...
168874	3539	3413.272727
168875	8844	9293.434783
168876	6421	6438.317708
168877	4599	4703.886734
168878	3215	3429.132353

168879 rows × 2 columns

```
#calculate metrics and print the results for test set
#Mean Absolute Error or MAE
MAE= mean_absolute_error(y_test,y_pred_dtCV)
#Mean Squared Error or MSE
MSE= mean_squared_error(y_test,y_pred_dtCV)
#Root Mean Squared Error or RMSE
RMSE = mean_squared_error(y_test,y_pred_dtCV,squared=False)
#R2
R2_score= r2_score(y_test, y_pred_dtCV)
#printing test results
print(f'Mean Absolute Error: {MAE}')
print(f'Mean Squared Error: {MSE}')
print(f'Root Mean Squared Error: {RMSE}')
print(f'R^2 score: {R2_score}')
```

Mean Absolute Error: 485.98139216757227
Mean Squared Error: 519709.28367313085
Root Mean Squared Error: 720.9086514067722
R^2 score: 0.946565317772411

ML Model-3

Random Forest

```
#Fitting random forest model on training data
RandomForest = RandomForestRegressor()
RandomForest.fit(x_train, y_train)
```



RandomForestRegressor()

```
# prediction on test data
y_pred_RF = RandomForest.predict(x_test)
```

```
#training score
train_score4 = RandomForest.score(x_train, y_train)
train_score4
```

0.9960915695674782

```
#test score
test_score4 = RandomForest.score(x_test, y_test)
test_score4
```

0.9725282338245743

```
#Dataframe showing test data and predicted data
RandomForest_Dataframe = pd.DataFrame(zip(y_test, y_pred_RF), columns = ['actual', 'pred'])
RandomForest_Dataframe
```




	actual	pred
0	5800	6696.40
1	5277	5088.02
2	7569	8038.46
3	19374	8138.63
4	10640	9932.52
...
168874	3539	3447.11
168875	8844	9399.98
168876	6421	6441.10
168877	4599	4693.97
168878	3215	3498.25

168879 rows × 2 columns



```
#Mean Absolute Error or MAE
MAE= mean_absolute_error(y_test,y_pred_RF)
#Mean Squared Error or MSE
MSE= mean_squared_error(y_test,y_pred_RF)
#Root Mean Squared Error or RMSE
RMSE= mean_squared_error(y_test,y_pred_RF,squared=False)
#R2 score
R2_score= r2_score(y_test, y_pred_RF)
#printing test results
print(f'Mean Absolute Error: {MAE}')
```




Mean Absolute Error: 348.30510133175557
Mean Squared Error: 267192.23030945
Root Mean Squared Error: 516.9064038193471
R^2 score: 0.9725282338245743

ML Model-4


✕ Xg boost

```
import xgboost as xgb
xgboost=xgb.XGBRegressor()
xgboost.fit(x_train,y_train)
```



XGBRegressor

XGBRegressor(base_score=None, booster=None, callbacks=None, colsample_bylevel=None, colsample_bynode=None, colsample_bytree=None, device=None, early_stopping_rounds=None, enable_categorical=False, eval_metric=None, feature_types=None, gamma=None, grow_policy=None, importance_type=None, interaction_constraints=None, learning_rate=None, max_bin=None, max_cat_threshold=None, max_cat_to_onehot=None, max_delta_step=None, max_depth=None, max_leaves=None, min_child_weight=None, missing=nan, monotone_constraints=None, multi_strategy=None, n_estimators=None, n_jobs=None, num_parallel_tree=None, random_state=None, ...)




```
# prediction on test dataset
y_pred_xg=xgboost.predict(x_test)
```

```
#training score
train_score5 = xgboost.score(x_train, y_train)
train_score5
```




0.9669960567179702

```
#test score
test_score5 = xgboost.score(x_test, y_test)
test_score5
```




0.9655258130241613

```
#Dataframe showing test data and predicted data
xgboost_Dataframe = pd.DataFrame(zip(y_test, y_pred_xg), columns = ['actual', 'pred'])
xgboost_Dataframe
```



	actual	pred
0	5800	6617.621094
1	5277	5225.238281
2	7569	8175.179688
3	19374	8220.042969
4	10640	9901.256836

```
#calculate metrics and print the results for test set
#Mean Absolute Error or MAE
MAE = mean_absolute_error(y_test,y_pred_xg)
#Mean Squared Error or MSE
MSE= mean_squared_error(y_test,y_pred_xg)
#Root Mean Squared Error or RMSE
RMSE= mean_squared_error(y_test,y_pred_xg,squared=False)
#R2 score
R2_score= r2_score(y_test, y_pred_xg)
#printing test results
print(f'Mean Absolute Error: {MAE}')
print(f'Mean Squared Error: {MSE}')
print(f'Root Mean Squared Error: {RMSE}')
print(f'R^2 score: {R2_score}')
```



```
Mean Absolute Error: 399.72266104957174
Mean Squared Error: 335298.2421064381
Root Mean Squared Error: 579.0494297609126
R^2 score: 0.9655258130241613
```

✓ **Conclusion**

Promotion Impact:

Stores that opted for promotions (represented by 1) generally had higher sales compared to those that didn't (represented by 0).

Consecutive Promotion (Promo2):

There wasn't a significant effect on sales after Promo2 was introduced, indicating that this type of promotion may not have had a substantial impact.

Holiday Sales:

Sales were notably lower during holidays when many stores were closed. More stores were open during School Holidays compared to State Holidays. Stores open during School holidays had higher sales than on regular days.

Assortment Impact:

Assortment type B had the highest sales, followed by C, while assortment A had the lowest sales, possibly due to a limited variety of products.

Competition:

Sales increased significantly after November, especially with promotions, likely due to holiday shopping. December had exceptionally high sales, likely due to Christmas and New Year celebrations.

Sales Trends Over the Years:

Sales were high in the year 1900 when there were fewer Rossmann stores and less competition. As the number of stores increased over the years, competition grew, leading to a decline in sales.

Sales by Day of the Week:

Monday had the highest sales, with promotions increasing sales further. Sales gradually decreased throughout the week, with the lowest sales on Saturday and almost no sales on Sunday when many stores were closed.