ARCADA UNIVERSITY OF APPLIED SCIENCE

Analytical Service Development Course

Forecasting Rossmann Store Sales Prediction

Mahalete Haile
Haritha Nayani
Venu Voleti

Date: 2.3.2018

INTRODUCTION

The project is based on Kaggle competition : Rossmann Store Sales. Rossmann is a drug store that operates in over 7 European countries with over 3000 drug stores. The task is to predict 6 weeks daily sales for 1,115 stores located across Germany . Sales of the store are highly influenced by many factors such as promotions, competition,school and State holidays, locality. The data is directly taken from Kaggle, and also the data at first glance looks organized and structured our biggest challenge was cleaning the data to bring it to a workable format.

Project task

Predicting 6 weeks daily sales of 1,115 individual stores located in Germany.

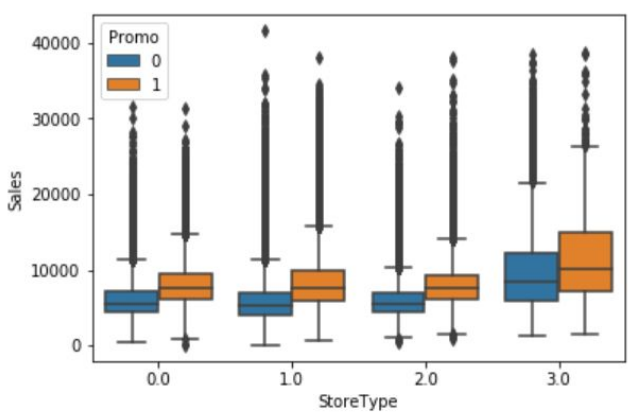
The Robust prediction model should be able to boost the sales of the company. Store managers must be able to manage the store efficiently by better staff management and increase the efficiency of employees. Predicting the sales and customers.

PRELIMINARY DATA VISUALIZATION

One of the situation we tried to see, is the relationship of various stores in comparison to the average sales during promotion period. It is unilateral visible below that whenever there is promotion all store types sales is higher that when there is no promotion.

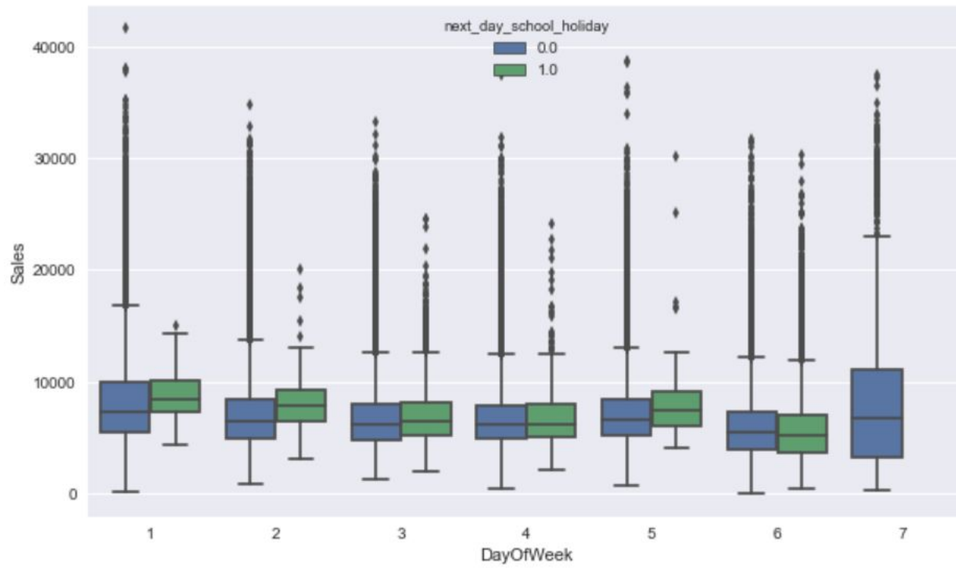
Compare the sales distribution for different store type and analyze promo effect

```
In [128]: sns.boxplot( x = 'StoreType', y = 'Sales', hue = 'Promo', data = train_model_copy  )
Out[128]: <matplotlib.axes._subplots.AxesSubplot at 0x226a7ce9080>
```

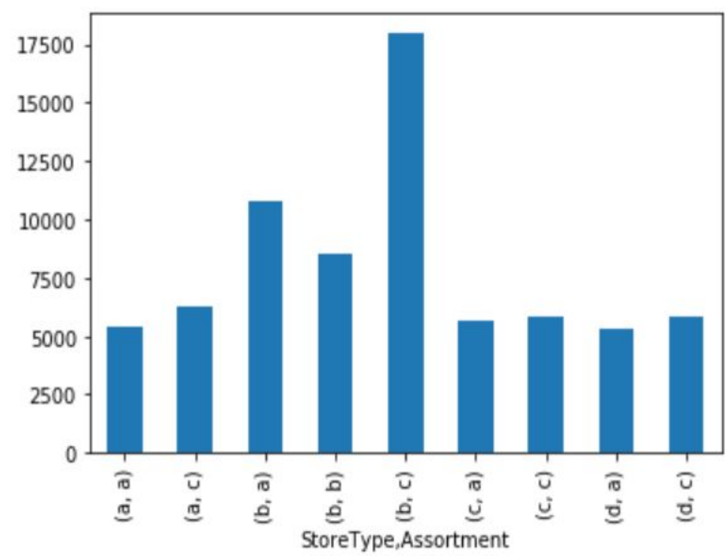


We also tried to see if sales is affected when there is holiday the next day like (school holiday). Picture below. There is no impact on sales on thursdays and sales are lower on saturdays. The remaining days sales are higher.

Out[132]: <matplotlib.axes._subplots.AxesSubplot at 0x226d2980dd8>



We also tried to see the various store type assortments in relation to the average sales.



DATA ANALYSIS & PREPARATION

The first look at the data gives us an insight in to the project. We are provided with historical data of 1115 drug stores in Germany. The files contain train, test and store data in csv format. Pandas were used to fetch the data, Numpy and Scipy to manipulate the data , while Matplotlib and Seaborn are used for plotting.

I) Data Description

Data Set	Variables	No of Variables	No of Observations
Train	Store,Day of week,Date,Sales,Customer, Open,Promo,State holiday,School holiday	9	9154881
Test	Id,Store,Day of week,Date,Open,Promo,State holiday,School holiday	8	328704
Store	Store,Storetype,Assortment,Competition distance, Competition open since month,Promo2,Promo2 since week,Promo2 since year, Promo interval	10	11150

Train dataset

In analyzing the Training dataset using the *head () and tail ()* functions, we notice that on the tail part of the dataset sales = 0. Which is a problem as Sales is the target column and thus it should have value.

```
In [6]: datatrain.head()
Out[6]:
```

	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 [7]:
```
datatrain.tail()
```
Out[7]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
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

In continuing to work on the dataset the next step is to create new columns of Year and Months to make analysis of seasonal effects on sales.

In [8]:
```
#Create new columns Year and Month to be used in the analysis of seasonal effects on sales.
datatrain['Year'] = pd.DatetimeIndex(datatrain['Date']).year
datatrain['Month'] = pd.DatetimeIndex(datatrain['Date']).month
```

In [9]:
```

datatrain['Date'] = pd.to_datetime(datatrain['Date'], format='%Y-%m-%d')
datatest['Date'] = pd.to_datetime(datatest['Date'], format='%Y-%m-%d')
```

In [11]:
```
#Checking the NaN values
datatrain.isnull().any()
```
Out[11]:
```
Store            False
DayOfWeek        False
Date             False
Year             False
Month            False
Customers        False
Open             False
Promo            False
StateHoliday     False
SchoolHoliday    False
Sales            False
dtype: bool
```

The above code returns whether there are any null values in the train data set. There are no null values in the data set. While this is a good outcome we have to further analyse the data types of the columns. StateHoliday is in the Object format which needs to be converted to integer value so that the values are in the similar, otherwise it throughs error .

```
In [12]: # Checking the data types
         datatrain.dtypes
```

```
Out[12]: Store                     int64
         DayOfWeek                 int64
         Date             datetime64[ns]
         Year                      int64
         Month                     int64
         Customers                 int64
         Open                      int64
         Promo                     int64
         StateHoliday             object
         SchoolHoliday             int64
         Sales                     int64
         dtype: object
```

```
In [13]: # Unique values of StateHoliday
         datatrain['StateHoliday'].unique()

Out[13]: array(['0', 'a', 'b', 'c', 0], dtype=object)
```

```
In [14]: #convert data to numeric data
         datatrain.loc[datatrain['StateHoliday'] == '0', 'StateHoliday'] = 0
         datatrain.loc[datatrain['StateHoliday'] == 'a', 'StateHoliday'] = 1
         datatrain.loc[datatrain['StateHoliday'] == 'b', 'StateHoliday'] = 1
         datatrain.loc[datatrain['StateHoliday'] == 'c', 'StateHoliday'] = 1
         datatrain['StateHoliday'] = datatrain['StateHoliday'].astype(int, copy=False)
```

```
In [15]: datatrain.StateHoliday.unique()

Out[15]: array([0, 1], dtype=int64)
```

The above code shows that there are 4 different values for the state holiday a = public holiday, b = Easter holiday, c = Christmas, 0 = None; which returns an object which is converted into binary value. The data description shows 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. Since all 4 categories show the same thing (that it is a state holiday on that day therefore returning value=1) and returning value =0 when there is no state holiday.

In [17]:
```
# Check the data types
datatrain.dtypes
```

Out[17]:
```
Store                     int64
DayOfWeek                 int64
Date             datetime64[ns]
Year                      int64
Month                     int64
Customers                 int64
Open                      int64
Promo                     int64
StateHoliday              int32
SchoolHoliday             int64
Sales                     int64
dtype: object
```

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

Out[18]:

	Store	DayOfWeek	Year	Month	Customers	Open	Promo	StateHoliday	SchoolHoliday	Sales
count	1.017209e+06	1.017209e+06	1.017209e+06	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	2.013832e+03	5.846762e+00	6.331459e+02	8.301067e-01	3.815145e-01	3.052470e-02	1.786467e-01	5.773819e+03
std	3.219087e+02	1.997391e+00	7.773960e-01	3.326097e+00	4.644117e+02	3.755392e-01	4.857586e-01	1.720261e-01	3.830564e-01	3.849926e+03
min	1.000000e+00	1.000000e+00	2.013000e+03	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	2.800000e+02	2.000000e+00	2.013000e+03	3.000000e+00	4.050000e+02	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	3.727000e+03
50%	5.580000e+02	4.000000e+00	2.014000e+03	6.000000e+00	6.090000e+02	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	5.744000e+03
75%	8.380000e+02	6.000000e+00	2.014000e+03	8.000000e+00	8.370000e+02	1.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	7.856000e+03
max	1.115000e+03	7.000000e+00	2.015000e+03	1.200000e+01	7.388000e+03	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	4.155100e+04

The train dataset is cleaned for null values and ready for modelling

Test dataset

The first difference we notice between the two datasets is that the Test dataset not contain both the Customers and Sales columns. The Sales column is the target column therefore is not available.
In order to make further analysis on the dataset the test data passes through the same process as in the train dataset.

In [19]:
```
#Change Year and Month column
datatest['Year'] = pd.DatetimeIndex(datatest['Date']).year
datatest['Month'] = pd.DatetimeIndex(datatest['Date']).month
```

Adding year and month column to test data set as we did in train data set.

In [20]:
```
datatest.head()
```
Out[20]:

	Id	Store	DayOfWeek	Date	Open	Promo	StateHoliday	SchoolHoliday	Year	Month
0	1	1	4	2015-09-17	1.0	1	0	0	2015	9
1	2	3	4	2015-09-17	1.0	1	0	0	2015	9
2	3	7	4	2015-09-17	1.0	1	0	0	2015	9
3	4	8	4	2015-09-17	1.0	1	0	0	2015	9
4	5	9	4	2015-09-17	1.0	1	0	0	2015	9

In [21]:
```
datatest.tail()
```
Out[21]:

	Id	Store	DayOfWeek	Date	Open	Promo	StateHoliday	SchoolHoliday	Year	Month
41083	41084	1111	6	2015-08-01	1.0	0	0	0	2015	8
41084	41085	1112	6	2015-08-01	1.0	0	0	0	2015	8
41085	41086	1113	6	2015-08-01	1.0	0	0	0	2015	8
41086	41087	1114	6	2015-08-01	1.0	0	0	0	2015	8
41087	41088	1115	6	2015-08-01	1.0	0	0	1	2015	8

The test data set is analysed to see if there are any stores that are non-functional (not open). The result shows that there are 5984 stores which are not open.

In [22]:
```
# To check how many closed stores are there
sum(datatest['Open'] == 0)
```
Out[22]:
```
5984
```

Furthermore, the test data is checked for NaN values. And there is a NaN value

In [24]:
```
#To check the NaN values in dataset
datatest.isnull().any()
```
Out[24]:
```
Store            False
DayOfWeek        False
Date             False
Year             False
Month            False
Open              True
Promo            False
StateHoliday     False
SchoolHoliday    False
dtype: bool
```

```
In [25]: #To check the  missing values in Open column.
         print(datatest.loc[np.isnan(datatest['Open'])])
```

```
       Store  DayOfWeek       Date  Year  Month  Open  Promo StateHoliday  \
479      622          4 2015-09-17  2015      9   NaN      1            0
1335     622          3 2015-09-16  2015      9   NaN      1            0
2191     622          2 2015-09-15  2015      9   NaN      1            0
3047     622          1 2015-09-14  2015      9   NaN      1            0
4759     622          6 2015-09-12  2015      9   NaN      0            0
5615     622          5 2015-09-11  2015      9   NaN      0            0
6471     622          4 2015-09-10  2015      9   NaN      0            0
7327     622          3 2015-09-09  2015      9   NaN      0            0
8183     622          2 2015-09-08  2015      9   NaN      0            0
9039     622          1 2015-09-07  2015      9   NaN      0            0
10751    622          6 2015-09-05  2015      9   NaN      0            0

       SchoolHoliday
479                0
1335               0
2191               0
3047               0
4759               0
5615               0
6471               0
7327               0
8183               0
9039               0
10751              0
```

Further investigating we see that the store with number 622 is the only store that has 11 NaN values in the test data set in spite of not being a state holiday or school holiday nor a Sunday. We adjusted the value NaN to 1, as we figures this will not impact the result of the outcome as a whole.

```
In [26]: #converting missing values of Open column in to 1(Because all DayofWeek 1-6 )
         datatest.loc[np.isnan(datatest['Open']), 'Open'] = 1
```

```
In [27]: #Rechecking for NaN values
         datatest.isnull().any()
```

```
Out[27]: Store            False
         DayOfWeek        False
         Date             False
         Year             False
         Month            False
         Open             False
         Promo            False
         StateHoliday     False
         SchoolHoliday    False
         dtype: bool
```

The data set free from NaN values after the code cleaning.
Continuing in the similar pattern, the State Holiday in the test data is changed from obj to int.

```
In [28]: #Checking for data types
         datatest.dtypes
Out[28]: Store                     int64
         DayOfWeek                 int64
         Date             datetime64[ns]
         Year                      int64
         Month                     int64
         Open                    float64
         Promo                     int64
         StateHoliday             object
         SchoolHoliday             int64
         dtype: object
```

We have one school holiday in this dataset:

```
In [29]: #Unique values of StateHoliday
         datatest['StateHoliday'].unique()
Out[29]: array(['0', 'a'], dtype=object)
```

```
In [30]: #convert data to numeric data
         datatest.loc[datatest['StateHoliday'] == '0', 'StateHoliday'] = 0
         datatest.loc[datatest['StateHoliday'] == 'a', 'StateHoliday'] = 1
         datatest['StateHoliday'] = datatest['StateHoliday'].astype(int, copy=False)
```

```
In [31]: datatest['StateHoliday'].unique()
Out[31]: array([0, 1], dtype=int64)
```

```
In [33]: datatest.dtypes
Out[33]: Store                     int64
         DayOfWeek                 int64
         Date             datetime64[ns]
         Year                      int64
         Month                     int64
         Open                    float64
         Promo                     int64
         StateHoliday              int32
         SchoolHoliday             int64
         dtype: object
```

```
In [34]: datatest.describe()
Out[34]:
```

	Store	DayOfWeek	Year	Month	Open	Promo	StateHoliday	SchoolHoliday
count	41088.000000	41088.000000	41088.0	41088.000000	41088.000000	41088.000000	41088.000000	41088.000000
mean	555.899533	3.979167	2015.0	8.354167	0.854361	0.395833	0.004381	0.443487
std	320.274496	2.015481	0.0	0.478266	0.352748	0.489035	0.066044	0.496802
min	1.000000	1.000000	2015.0	8.000000	0.000000	0.000000	0.000000	0.000000
25%	279.750000	2.000000	2015.0	8.000000	1.000000	0.000000	0.000000	0.000000
50%	553.500000	4.000000	2015.0	8.000000	1.000000	0.000000	0.000000	0.000000
75%	832.250000	6.000000	2015.0	9.000000	1.000000	1.000000	0.000000	1.000000
max	1115.000000	7.000000	2015.0	9.000000	1.000000	1.000000	1.000000	1.000000

The test data set is also free from NaN values and is all set for modelling.

Store dataset

```
In [35]: datastore.head()
```

Out[35]:

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2	Promo2SinceWeek	Promo2SinceYear	F
0	1	c	a	1270.0	9.0	2008.0	0	NaN	NaN	
1	2	a	a	570.0	11.0	2007.0	1	13.0	2010.0	J
2	3	a	a	14130.0	12.0	2006.0	1	14.0	2011.0	J
3	4	c	c	620.0	9.0	2009.0	0	NaN	NaN	
4	5	a	a	29910.0	4.0	2015.0	0	NaN	NaN	

```
In [36]: datastore.tail()
```

Out[36]:

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2	Promo2SinceWeek	Promo2SinceYear
1110	1111	a	a	1900.0	6.0	2014.0	1	31.0	2013.0
1111	1112	c	c	1880.0	4.0	2006.0	0	NaN	NaN
1112	1113	a	c	9260.0	NaN	NaN	0	NaN	NaN
1113	1114	a	c	870.0	NaN	NaN	0	NaN	NaN
1114	1115	d	c	5350.0	NaN	NaN	1	22.0	2012.0

We can clearly see lot of NaN values.

```
In [37]: #To check for NaN values
         datastore.isnull().sum()

Out[37]: Store                          0
         StoreType                      0
         Assortment                     0
         CompetitionDistance            3
         CompetitionOpenSinceMonth    354
         CompetitionOpenSinceYear     354
         Promo2                         0
         Promo2SinceWeek              544
         Promo2SinceYear              544
         PromoInterval                544
         dtype: int64
```

CompetitionDistance, CompetitionOpenSinceMonth, CompetitionOpenSinceYear, Promo25inceWeek, Promo25inceYear,PromoInterval have all different since of missing values. By checking the unique values in those columns we used Scikit-learn build in command Imputer forcompliting missing values

```
In [38]: def convert_to_int(df, colname, start_value=0):
             while df[colname].dtype == object:
                 myval = start_value # factor starts at "start_value".
                 for sval in df[colname].unique():
                     df.loc[df[colname] == sval, colname] = myval
                     myval += 1
                 df[colname] = df[colname].astype(int, copy=False)
             print('levels :', df[colname].unique(), '; data type :', df[colname].dtype)
```

```
In [39]: datastore['StoreType'].unique()

Out[39]: array(['c', 'a', 'd', 'b'], dtype=object)
```

```
In [40]: convert_to_int(datastore, 'StoreType')
         convert_to_int(datastore, 'Assortment')
         #datastore.dtypes

         levels : [0 1 2 3] ; data type : int32
         levels : [0 1 2] ; data type : int32
```

```
In [41]: datastore['PromoInterval'].unique()

Out[41]: array([nan, 'Jan,Apr,Jul,Oct', 'Feb,May,Aug,Nov', 'Mar,Jun,Sept,Dec'],
               dtype=object)

In [42]: datastore.loc[datastore['Promo2'] == 0, ['Promo2SinceWeek', 'Promo2SinceYear', 'PromoInterval']] = 0

In [43]: datastore.loc[datastore['Promo2'] != 0, 'Promo2SinceWeek'] = datastore['Promo2SinceWeek'].max() - datastore.loc[datastore['Promo2

In [44]: datastore.loc[datastore['Promo2'] != 0, 'Promo2SinceYear'] = datastore['Promo2SinceYear'].max() - datastore.loc[datastore['Promo2

In [45]: convert_to_int(datastore, 'PromoInterval', start_value=0)

         levels : [0 1 2 3] ; data type : int32
```

```
In [46]: #datastore.isnull().any()
         datastore.isnull().sum()

Out[46]: Store                          0
         StoreType                      0
         Assortment                     0
         CompetitionDistance            3
         CompetitionOpenSinceMonth    354
         CompetitionOpenSinceYear     354
         Promo2                         0
         Promo2SinceWeek                0
         Promo2SinceYear                0
         PromoInterval                  0
         dtype: int64
```

In [47]:
```
from sklearn.preprocessing import Imputer
imputer = Imputer().fit(datastore)
store_imputed = imputer.transform(datastore)
```

In [48]:
```
store = pd.DataFrame(store_imputed, columns=datastore.columns.values)
```

In [49]:
```
store.isnull().any()
```

Out[49]:
```
Store                        False
StoreType                    False
Assortment                   False
CompetitionDistance          False
CompetitionOpenSinceMonth    False
CompetitionOpenSinceYear     False
Promo2                       False
Promo2SinceWeek              False
Promo2SinceYear              False
PromoInterval                False
dtype: bool
```

After checking whether columns are similar in both train and store data we merge train and store datasets before modeling the data.

In [50]:
```
#To check the columns are similar in both train and store datasets
len(store['Store']) - sum(store['Store'].isin(datatrain['Store']))
```

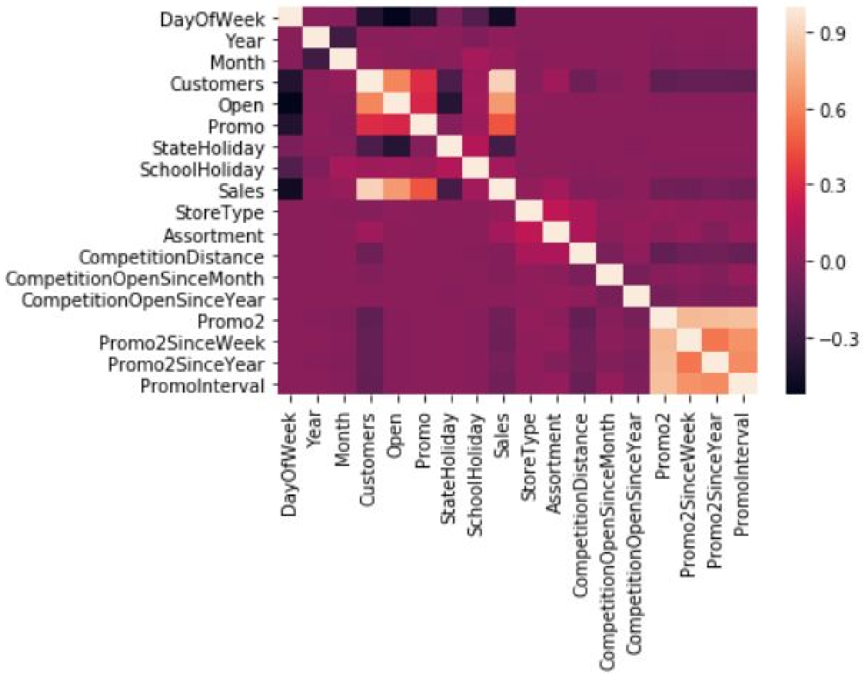
Out[50]:
```
0
```

In [51]:
```
#Merge train and store datasets
train_store = pd.merge(datatrain, store, how = 'left', on='Store')
```

In [53]:
```
coriMat = pd.DataFrame(train_store.loc[:, ['DayOfWeek','Date', 'Year','Month','Customers','Open','Promo','StateHoliday','SchoolHo
print(coriMat)
```

```
                           DayOfWeek      Year     Month  Customers      Open  \
DayOfWeek                   1.000000  0.001937 -0.005362  -0.386445 -0.528963
Year                        0.001937  1.000000 -0.269382  -0.001212 -0.001009
Month                      -0.005362 -0.269382  1.000000   0.038179 -0.000681
Customers                  -0.386445 -0.001212  0.038179   1.000000  0.616768
Open                       -0.528963 -0.001009 -0.000681   0.616768  1.000000
Promo                      -0.392925  0.024300 -0.011747   0.316169  0.295042
StateHoliday               -0.052889  0.006074 -0.000794  -0.226608 -0.378378
SchoolHoliday              -0.205388 -0.036535  0.103282   0.071568  0.086171
Sales                      -0.462125  0.023519  0.048768   0.894711  0.678472
StoreType                   0.000061 -0.001792 -0.009107  -0.011882  0.017250
Assortment                 -0.000052  0.001492  0.007586   0.078964  0.012970
CompetitionDistance        -0.000025  0.000702  0.003574  -0.102777  0.007981
CompetitionOpenSinceMonth   0.000005 -0.000100 -0.000515  -0.025098  0.001144
CompetitionOpenSinceYear   -0.000021  0.000636  0.003232   0.007242  0.002288
Promo2                      0.000168 -0.004982 -0.025323  -0.150159 -0.008309
Promo2SinceWeek             0.000061 -0.001831 -0.009305  -0.134759 -0.005624
Promo2SinceYear             0.000116 -0.003439 -0.017481  -0.131701 -0.007413
PromoInterval               0.000074 -0.002213 -0.011245  -0.135765 -0.006659
```

```
In [54]: sns.heatmap(data=coriMat)
plt.show()
```



The correlation shows that there is best correlation between Customers, Open and Promo.

PREDICTION MODELS

In model selection the first model we tried is the linear regression **. The prediction was done with feature and without feature selection.**

The prediction scores differ slightly by dropping in accuracy when selecting the features the accuracy score is 0.54 while without feature selection the accuracy gives better results of 0.56.

Feature selection

In [67]:
```
train_feature = train_model
test_feature = test_model
```

In [68]:
```
train_feature = train_feature.drop(['Year','Month','StoreType','Assortment','CompetitionOpenSinceMonth','CompetitionOpenSinceYear
```

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

Out[69]:

	Store	DayOfWeek	Open	Promo	StateHoliday	SchoolHoliday	CompetitionDistance	Sales
0	1	5	1	1	0	1	1270.0	5263
1	2	5	1	1	0	1	570.0	6064
2	3	5	1	1	0	1	14130.0	8314
3	4	5	1	1	0	1	620.0	13995
4	5	5	1	1	0	1	29910.0	4822

In [70]:
```
test_feature = test_feature.drop(['Year','Month','StoreType','Assortment','CompetitionOpenSinceMonth','CompetitionOpenSinceYear',
```

In [72]:
```
from sklearn.cross_validation import train_test_split
Xf = train_feature.drop('Sales', axis=1)
yf = train_feature['Sales']
Xf_train, Xf_test, yf_train, yf_test = train_test_split(Xf, yf, random_state=42)
```

```
C:\Users\abhin\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module was deprecated in ve
rsion 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that
the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [73]:
```
from sklearn.linear_model import LinearRegression
from sklearn import cross_validation as cv
```

In [74]:
```
lr = LinearRegression()
lr.fit(Xf_train, yf_train)
```

Out[74]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [75]:
```
from sklearn.model_selection import cross_val_score
#print(lr.score(X_test, y_test))
#print(" train set accuracy: {:.2f}".format(lr.score(X_train, y_train)))
print(" test set accuracy: {:.2f}".format(lr.score(Xf_test, yf_test)))

scores = cross_val_score(lr, Xf_test, yf_test, cv=5)
scores
```

```
 test set accuracy: 0.54
```

Out[75]:
```
array([0.53679674, 0.53540661, 0.53746898, 0.53172823, 0.53566987])
```

Without feature selection

```
In [78]: from sklearn.cross_validation import train_test_split
         X = train_model.drop('Sales', axis=1)
         y = train_model['Sales']
         X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

In [79]: lr = LinearRegression()
         lr.fit(X_train, y_train)

Out[79]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

In [80]: from sklearn.model_selection import cross_val_score
         print(" test set accuracy for sales: {:.2f}".format(lr.score(X_test, y_test)))

         scores = cross_val_score(lr, X_test, y_test, cv=5)
         scores

          test set accuracy for sales: 0.56

Out[80]: array([0.56025766, 0.55880188, 0.55987175, 0.55491396, 0.55873551])
```

The second model we selected is **Random forest** which proved to be the most accurate model for our prediction. Random Forest Tree tries to construct a multitude of decision trees and uses random amount of data for training. With this randomized data, it is hard for random forest tree to overfit.

The accuracy score we got is in fact optimal in selecting the features the score is 0.91, while without the feature selection is 0.93.

```
In [76]: from sklearn.ensemble import RandomForestRegressor
         rf = RandomForestRegressor(n_jobs=-1, random_state=42)
         rf.fit(Xf_train, yf_train)

Out[76]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                    max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=-1,
                    oob_score=False, random_state=42, verbose=0, warm_start=False)

In [77]: print(" test set accuracy: {:.2f}".format(rf.score(Xf_test, yf_test)))
         scores = cross_val_score(rf, Xf_test, yf_test, cv=5)
         scores

          test set accuracy: 0.91

Out[77]: array([0.90185776, 0.89795661, 0.90240245, 0.8992019 , 0.90218422])
```

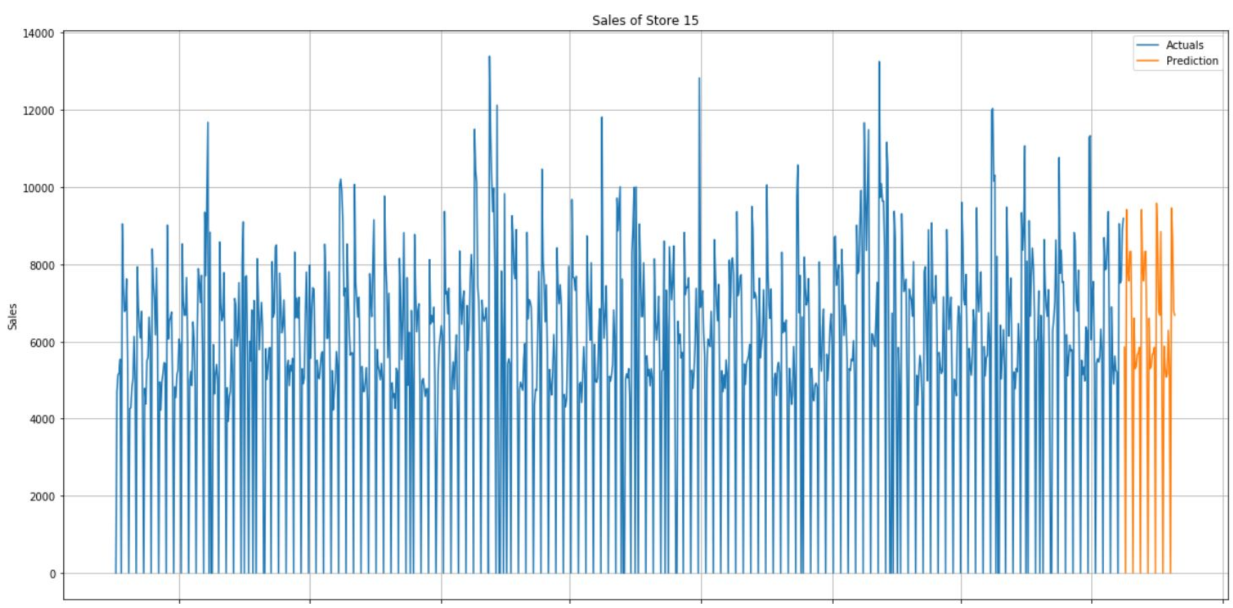
Without Feature selection

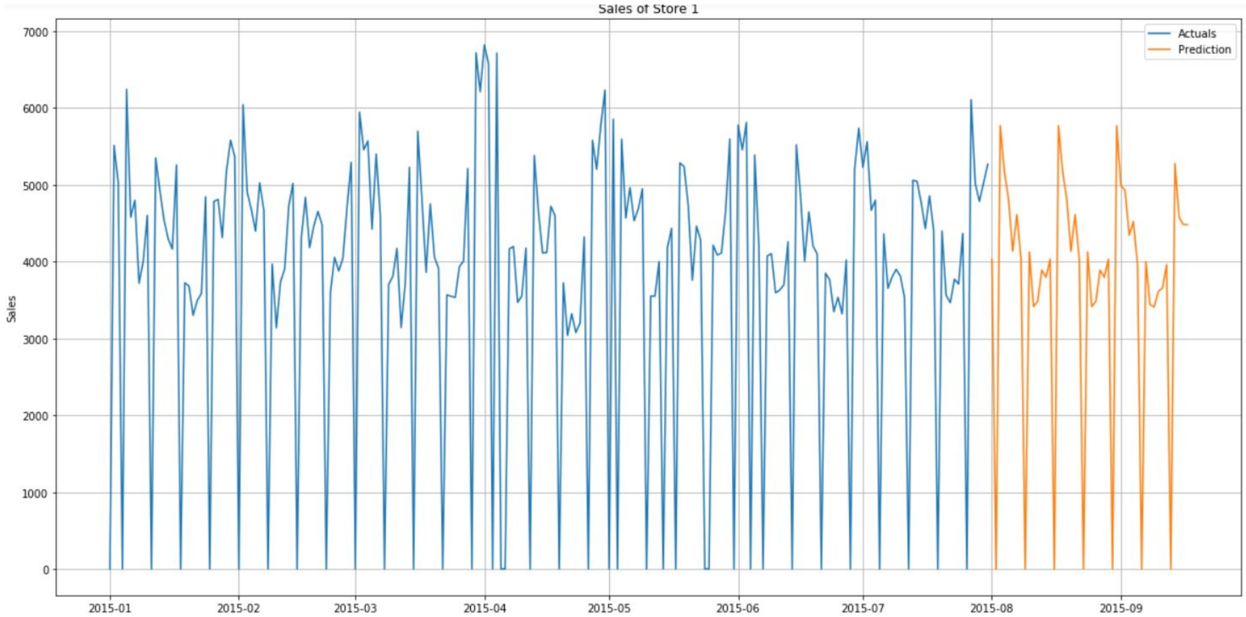
```
In [88]: from sklearn.ensemble import RandomForestRegressor
         rf = RandomForestRegressor(n_jobs=-1, random_state=42)
         rf.fit(X_train, y_train)

Out[88]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                    max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=-1,
                    oob_score=False, random_state=42, verbose=0, warm_start=False)

In [89]: print(" test set accuracy: {:.2f}".format(rf.score(X_test, y_test)))
         scores = cross_val_score(rf, X_test, y_test, cv=5)
         scores

          test set accuracy: 0.93

Out[89]: array([0.91835825, 0.91568859, 0.91892682, 0.91811208, 0.92078823])
```

We also tried to see the the mean absolute error(MAE), accuracy score in both linear regression and random forest. MAE- is in fact one of a number of ways of comparing forecasts with their eventual outcomes. In linear regression Mean absolute error is: 1771.71 and in random forest the Mean absolute error is: 616.93 .

RESULT ANALYSIS

The model successfully predicts 6 weeks daily sales of each store . As a sample we took store 1 and 15 .





We can clearly see from the above picture that every two weeks whenever there is promotion, the sales spikes upwards. This information would help managers improve employees work schedules much better .

After the sales prediction to further improve the decision making process of managers we tried predict the number of customers at a particular store. We started predicting the sales and we realized we can also predict the number customers this is the add on version of the project after our presentation

By using random forest the accuracy score is :0.96

Customer prediction

```
In [114]: Xc = train_model_cust.drop('Customers', axis=1)
          yc = train_model_cust['Customers']
          Xc_train, Xc_test, yc_train, yc_test = train_test_split(Xc, yc, random_state=42)

In [115]: from sklearn.ensemble import RandomForestRegressor
          rf = RandomForestRegressor(n_jobs=-1, random_state=42)
          rf.fit(Xc_train, yc_train)

Out[115]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                     max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=-1,
                     oob_score=False, random_state=42, verbose=0, warm_start=False)

In [116]: print(" test set accuracy: {:.2f}".format(rf.score(Xc_test, yc_test)))
          scores_Customers = cross_val_score(rf, Xc_test, yc_test, cv=5)
          scores_Customers

           test set accuracy: 0.96

Out[116]: array([0.95524214, 0.95377177, 0.95535059, 0.95664713, 0.95705829])
```

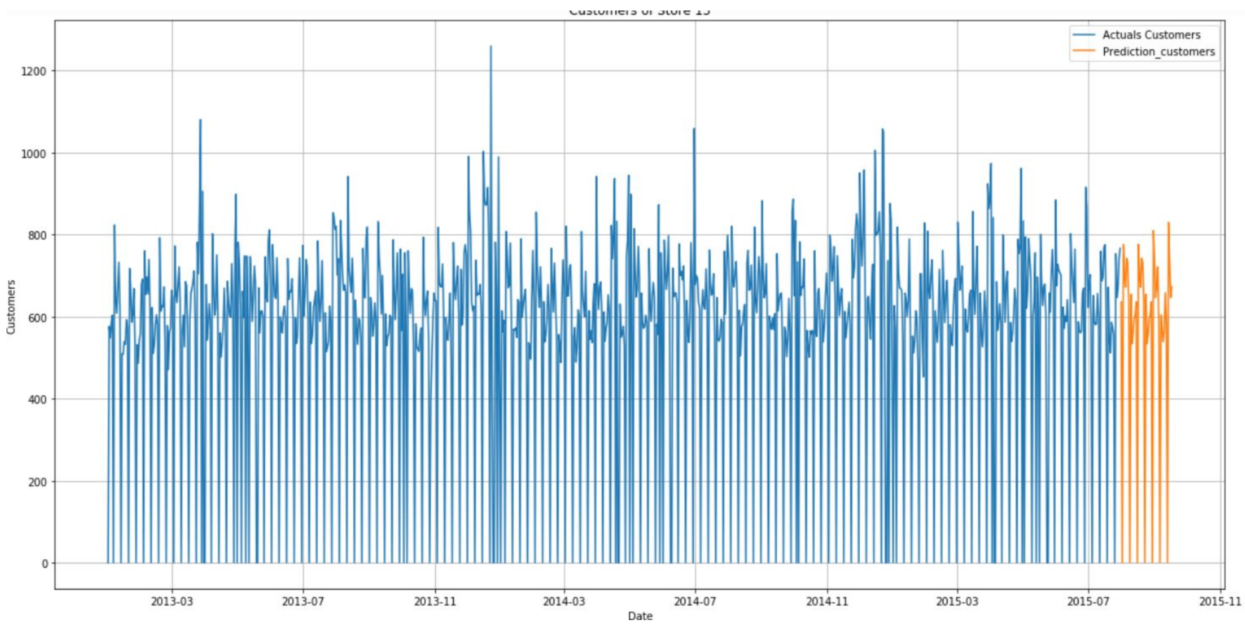
In [117]:
```
y_pred_Customers = rf.predict(Xc_test)
```

In [118]:
```

test_model_cust['Customers'] = rf.predict(test_model_cust)
```

In [119]:
```
test_model_cust['Date'] = test_store['Date']
train_model_cust['Date'] = train_store['Date']
```

In [121]:
```
storetrain_cust_15 = train_model_cust[train_model_cust['Store'] == 15]
storetest_cust_15 = test_model_cust[test_model_cust['Store'] == 15]
```

In [122]:
```
plt.figure(figsize=(20,10))
plt.plot(storetrain_cust_15['Date'], storetrain_cust_15['Customers'],label="Actuals Customers")
plt.plot(storetest_cust_15['Date'], storetest_cust_15['Customers'],label="Prediction_customers")
plt.title("Customers of Store 15")
plt.ylabel("Customers")
plt.xlabel("Date")
plt.grid(True)
plt.legend()
plt.show()
```



CONCLUSION

- Effective prediction for 6 weeks daily sales prediction for each store.
- Effective prediction of number of customers for 6 weeks.
- By seeing daily customers and sales managers can schedule employees for better supply chain management.
- Data preparation was the major obstacle of this project, and it was a optimal learning curve for us.