BUSINESS ANALYTICS 5TH ASSIGNMENT

16102275 Park Hyun Woo

CONTEXT

1. Get new variable from CompetitionDistance
2. Get new variables from Promo2 and Promo2interval
3. Get new variable from CompetitionOpenSince[Year/Month]
4. Get new variable from historical sales or number of customers
5. Modeling

BEFORE GETTING VARIABLES

- CompetitionDistance, Promo2,PromoInterval and CompetitionOpenSince[Month,Year] are not change value, static value.
- There values are characteristic of each stores

DELETING OUTLIERS

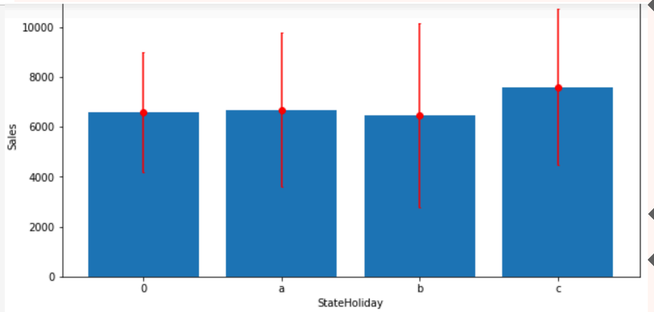
```
train.drop(train[train['Open']==0].index,inplace=True)
```

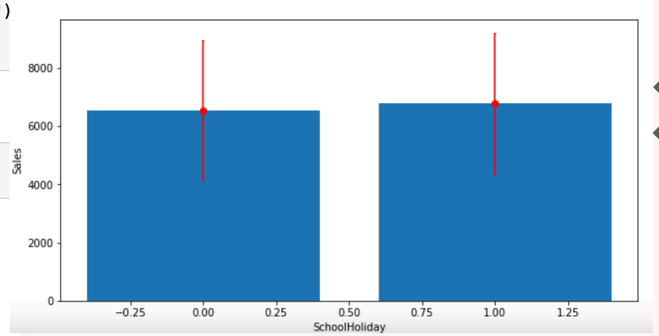
```
train.reset_index(drop=True,inplace=True)

def find_outlier(data):
    Q1 , Q3 = np.percentile(data,[25,75])
    IQR = Q3 - Q1
    Over_outlier = Q3 + 1.5*IQR
    Low_outlier = Q1 - 1.5*IQR
    location = np.where((data>Over_outlier)|(data<Low_outlier))
    result = [location[0],len(location[0])]
    return result

locationOfOutlier  =find_outlier(train['Sales'])[0]
numOfOutlier = find_outlier(train['Sales'])[1]

print("Number of Outlier is " , numOfOutlier)
print("percentage of Outlier of whole data ", (numOfOutlier/len(train))*100,"%")

train.drop(index = locationOfOutlier,inplace = True)
```

```
Number of Outlier is  30769
percentage of Outlier of whole data  3.6439236752598316 %
```

```
train = train.fillna(0)
train['StateHoliday'].value_counts(),train['SchoolHoliday'].value_counts()
```

```
(0    812866
 a       591
 b       108
 c        58
 Name: StateHoliday, dtype: int64,
 0    657162
 1    156461
 Name: SchoolHoliday, dtype: int64)
```





- Firstly I delete data which is open==0 , because this don't have about sales data
- I delete outliers of sales by using IQR method.
- And after deleting outliers, StateHoliday and SchoolHoliday is not used in my model. Those graph shows there are a little sales difference in StateHoliday and SchoolHoliday.

NEW COLUMN AFTER 1WEEK SALES

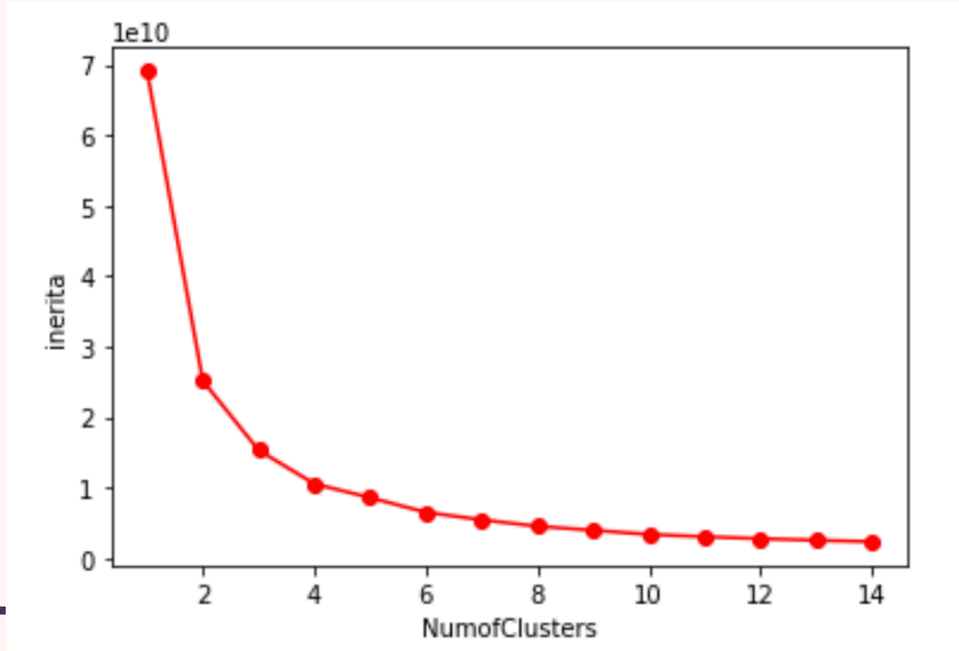
```
train['Date'] = pd.to_datetime(train['Date'])
train['Year'] =train['Date'].dt.year
train['Month']=train['Date'].dt.month

pred_train = train[['Date','Sales','Store']]
pred_train = pred_train.rename(columns = {'Sales':'After1WeekSales'})
pred_train['Date'] = train['Date'] + datetime.timedelta(days=7)
pred_train = train.merge(pred_train,on=['Store','Date'],how='left')
pred_train = pred_train.dropna(subset =['After1WeekSales'])

pred_train
```

- **Because purpose of this analysis is predict after 1week sales.**
- **So I make new column After1WeekSales by each Date + 7 and merge this**

GET NEW VARIABLE FROM COMPETITIONDISTANCE

COMPETITION DISTNACE

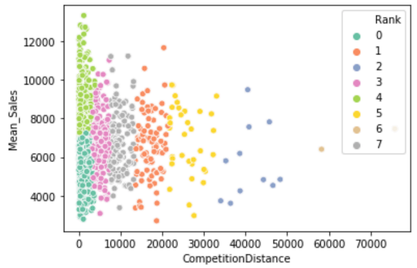
- Because I classify data by using CompetitionDistance and Sales'mean of each store, I use K-means cluster method for classify.
- This inertia graph shows the best num of clusters is over 8
- Mean_Sales column mean each store's mean of sales
- I select num of clusters is 8



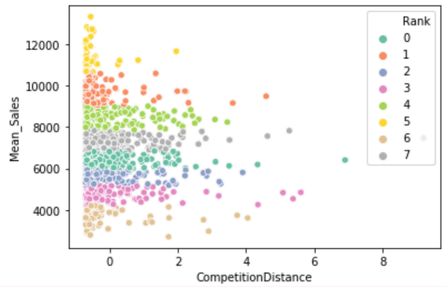
COMPETITION DISTNACE

```
from sklearn.cluster import KMeans
rank_store = new_store[['Store','CompetitionDistance','Mean_Sales']]
kmeans = KMeans(n_clusters=8)
kmeans.fit(rank_store[['CompetitionDistance','Mean_Sales']])

rank_store['Rank'] = kmeans.labels_

sns.scatterplot(x="CompetitionDistance", y="Mean_Sales", hue="Rank",
                data=rank_store[['CompetitionDistance','Mean_Sales','Rank']] ,palette="Set2")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff22a89ebd0>
```




```
std_kmeans = KMeans(n_clusters=8)

std_rank_store = new_store[['Store','CompetitionDistance','Mean_Sales']]
std_rank_store['CompetitionDistance'] = (std_rank_store['CompetitionDistance']-std_rank_store

std_kmeans.fit(std_rank_store[['CompetitionDistance','Mean_Sales']])

std_rank_store['Rank'] = std_kmeans.labels_
sns.scatterplot(x="CompetitionDistance", y="Mean_Sales", hue="Rank",
                data=std_rank_store[['CompetitionDistance','Mean_Sales','Rank']] , palette="S
```

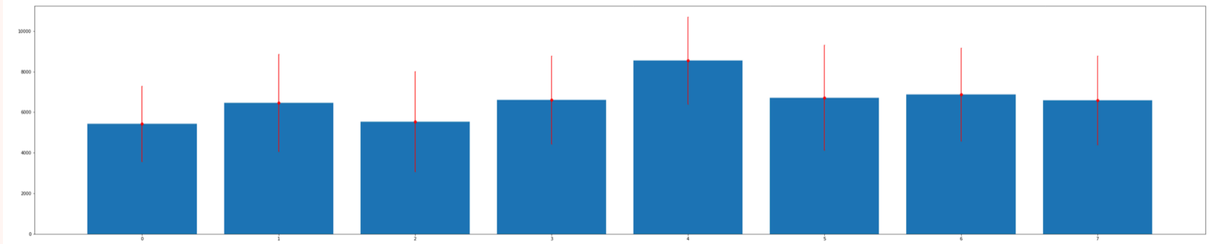


- Left graph is draw by using Mean_Sales and CompetitionDistance. I make rank column for classify CompetitionDistance.
- Right graph is same way with 1st graph, but this data is standardization of CompetitionDistance and Mean_Sales.
- But this not show relation like (CompetitionDistance increase rank increase)

COMPETITIONDISTANCE

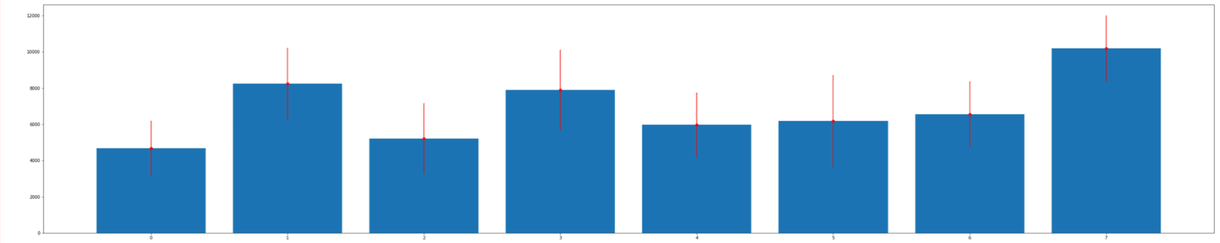
```
A = rank_train.groupby(['Rank'])['After1WeekSales'].mean()
B = rank_train.groupby(['Rank'])['After1WeekSales'].std()
plt.figure(figsize=(50,10))
plt.bar(range(len(A)),A)
plt.errorbar(range(len(A)),A,yerr=B, fmt='o', c='r',ecolor='r',capthick=1,capsize=1)
```

```
<ErrorbarContainer object of 3 artists>
```



```
A = std_rank_train.groupby(['Rank'])['After1WeekSales'].mean()
B = std_rank_train.groupby(['Rank'])['After1WeekSales'].std()
plt.figure(figsize=(50,10))
plt.bar(range(len(A)),A)
plt.errorbar(range(len(A)),A,yerr=B, fmt='o', c='r',ecolor='r',capthick=1,capsize=1)
```

```
<ErrorbarContainer object of 3 artists>
```



- So I make mean graph by using our target value ‘After1WeekSales’
- This graph don’t show also Rank increase and mean increase.
- So I don’t use Rank Column . I don’t make new column from CompetitionDistance

GET NEW VARIABLES FROM PROMO2 AND PROMO2INTERVAL

PROMO2

```
promo_store
```

	Store	Start1stPromo	Start2ndPromo	Start3rdPromo	Start4thPromo	Promo2StartDate
0	2	1	4	7	10	2010-04-02
1	3	1	4	7	10	2011-04-09
2	11	1	4	7	10	2012-01-08
3	12	1	4	7	10	2010-04-02
4	13	2	5	8	11	2009-11-12
...	...	...	...	...	...	...
565	1106	1	4	7	10	2013-08-06
566	1107	1	4	7	10	2010-04-02
567	1109	1	4	7	10	2012-06-03
568	1111	1	4	7	10	2013-08-06
569	1115	3	6	9	12	2012-06-03

570 rows × 6 columns

- I make promo_store data frame , new columns are there
- Promo2StartDate is made for differentiating before Promo2 sales and after promo2 sales by each Store.
- Promo interval is regular -> 1,4,7,10 month set 2,5,8,11 month set and 3,6,9,12 month set

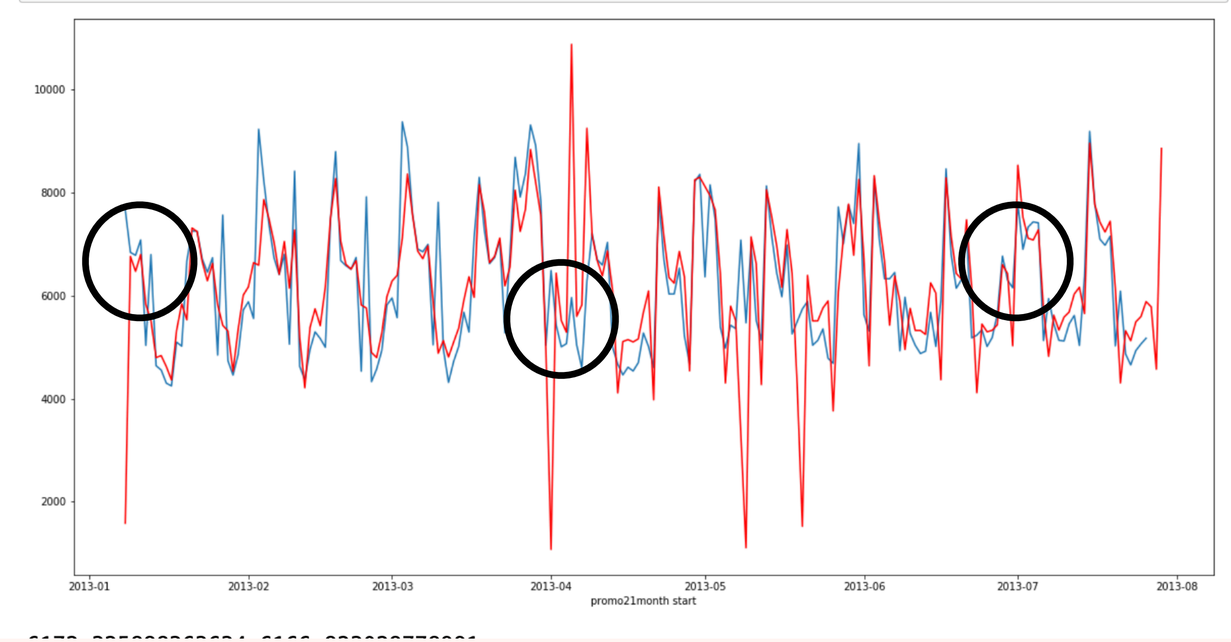
PROMO2

```
#for testing sales data which have each samples classfied by PromoInterval
for i in range(1,4):
    test1_sales = after_sales[after_sales['Start1stPromo']==i]
    test2_sales = before_sales[before_sales['Start1stPromo']==i]
    fig = plt.figure(figsize=(20,10))
    A = test1_sales.groupby(['Date'])['Sales'].mean().head(200)
    B = test2_sales.groupby(['Date'])['Sales'].mean().head(200)
    plt.plot(A.index,A)
    plt.plot(B.index,B,color = 'r')
    plt.xlabel(i)
    plt.show()
    print(A.mean(),B.mean())
```

- **For checking promo2 interval start month make more sales. I make graphs.**

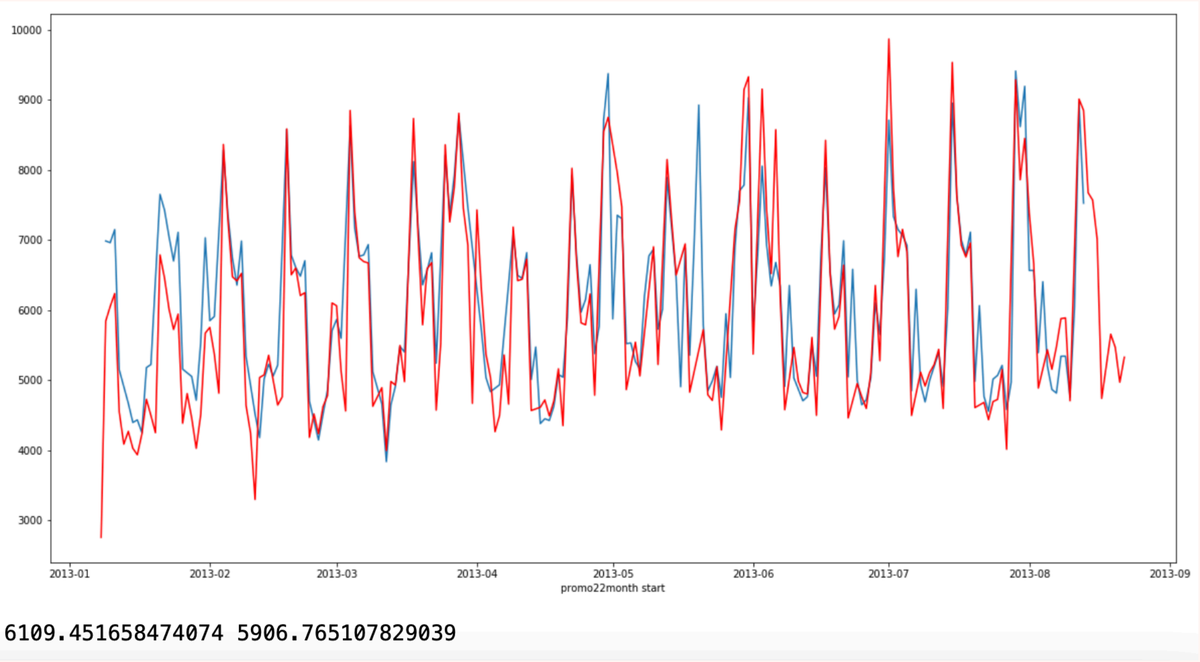
PROMO2

```
#for testing sales data which have each samples classfied by PromoInterval
for i in range(1,4):
    test1_sales = after_sales[after_sales['Start1stPromo']==i]
    test2_sales = before_sales[before_sales['Start1stPromo']==i]
    fig = plt.figure(figsize=(20,10))
    A = test1_sales.groupby(['Date'])['Sales'].mean().head(200)
    B = test2_sales.groupby(['Date'])['Sales'].mean().head(200)
    plt.plot(A.index,A)
    plt.plot(B.index,B,color = 'r')
    plt.xlabel("promo2"+str(i)+"month start")
    plt.show()
    print(A.mean(),B.mean())
```

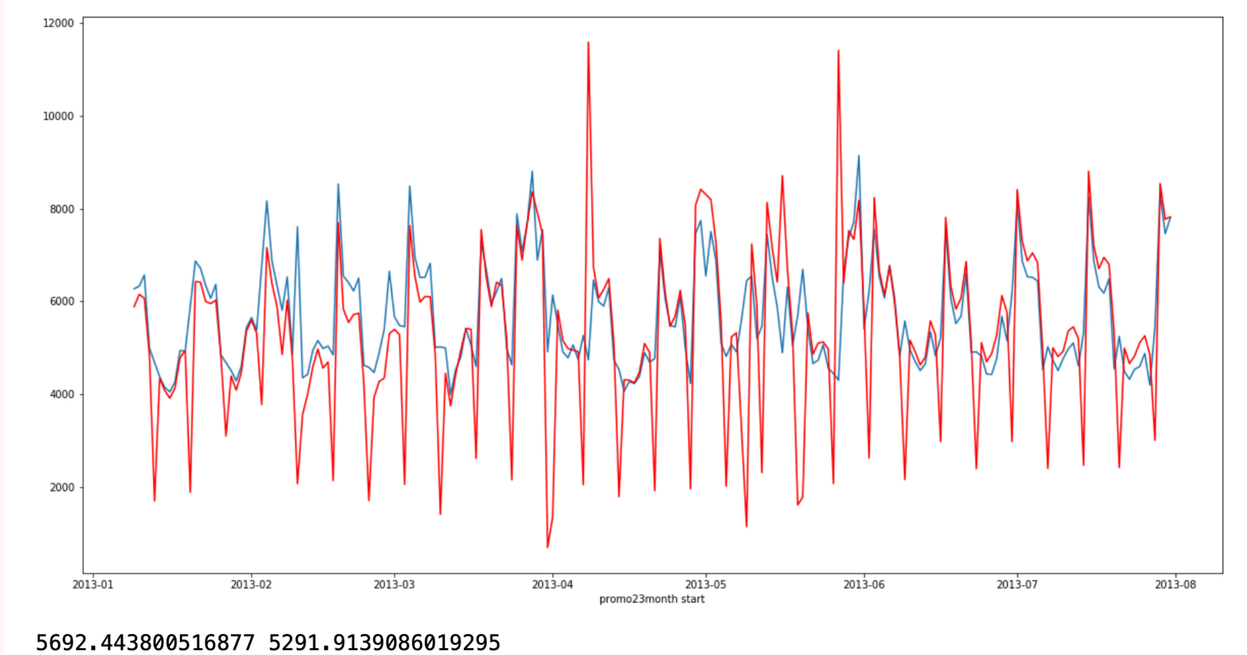


- This data shows month 1 month 4 month 7 month 10 start promo2.
- Blue line graph is Promo2 going on data, red line is promo2 not going on data. But this graph shows no different at each promo2 start month.

PROMO2



6109.451658474074 5906.765107829039



5692.443800516877 5291.9139086019295

- Left graph is promo2 interval 2,5,8,11 month. And right graph is promo2 interval 3,6,9,12 month. That is same no difference.
- But below number is mean of after promo2 start and before promo2 start sales.

- That shows difference of data

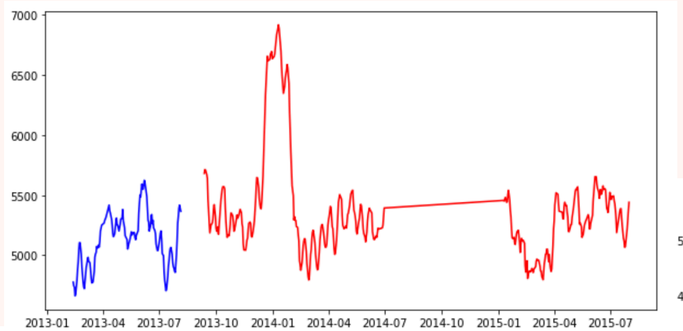
PROMO2

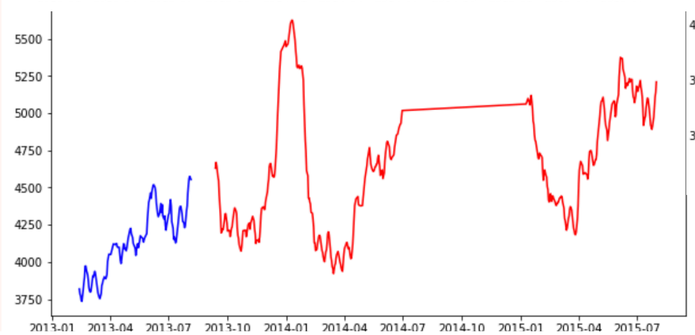
```
#for compare each store that have promo2 starttime in data's date period(2013.1.1~2015.7)
import random

starttime = datetime.datetime(2013,1,1)
testing_promo2 = sales[sales['Promo2StartDate']>starttime][['Store','Date','Sales','Promo2Sta
testing_promo2 = testing_promo2.sort_values(['Store','Date'])
testing_promo2.index = testing_promo2['Date']

store_num = testing_promo2['Store'].unique()
```

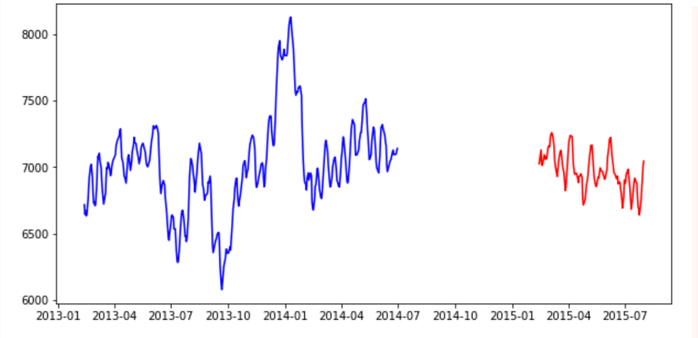
```
select_random = random.sample(store_num.tolist(),5)

for i in select_random:
    test = testing_promo2[testing_promo2['Store']==i]
    before_test = test[test['Date']<test['Promo2StartDate'].values[0]]
    after_test = test[test['Date']>test['Promo2StartDate'].values[0]]
    plt.figure(figsize=(10,5))
    plt.plot(before_test.index,before_test['Sales'].rolling(30).mean(),color ='b')
    plt.plot(after_test.index,after_test['Sales'].rolling(30).mean(),color ='r')
```

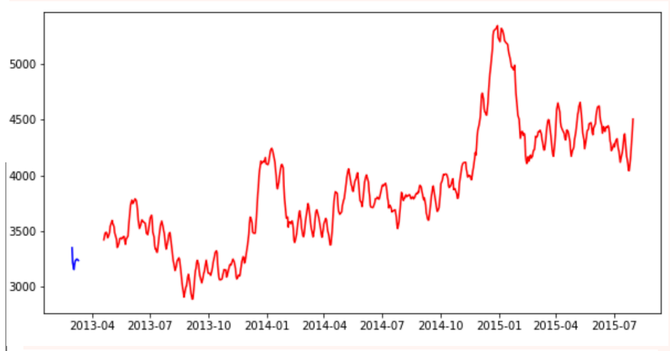
- So I testing of promo2 with sales data.
- Before starting promo2 data's line graph and after starting promo2 data's line graph are draw by using moving average sales data in each store's sales.
- I choose store radomly. Because I cannot show 512 (store num promo2 is going on)graph in python file.

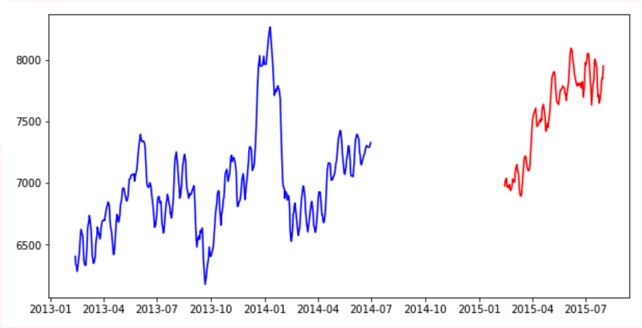
PROMO2











- That 5 graph is randomly selected store's sales moving-average graph.
- Blue line graph is before promo2 start date sales graph
- Red line graph is after promo2 start date sales graph
- Many times execute that code, there are difference between red-line and blue-line
- So I decide promo2 column in whole dataset(train dataset). Promo2 ==1 mean promotion is going on each data's date, and Promo2 == 0 mean promotion is not going on each data's date

PROMO2

```
sales['Promo2'] = 0

sales.loc[(sales['Promo2StartDate'].notnull()) & (sales['Date']>sales['Promo2StartDate']),'Pr
sales = sales.sort_values(['Store','Date'])
sales
```

After1WeekSales	Start1stPromo	Start2ndPromo	Start3rdPromo	Start4thPromo	Promo2StartDate	AfterPromo2Sales	Promo2
5530.0	NaN	NaN	NaN	NaN	NaT	0	0
4327.0	NaN	NaN	NaN	NaN	NaT	0	0
4486.0	NaN	NaN	NaN	NaN	NaT	0	0
4997.0	NaN	NaN	NaN	NaN	NaT	0	0
7176.0	NaN	NaN	NaN	NaN	NaT	0	0
...	...	...	...	...	...	...	...
6083.0	3	6	9	12	2012-06-03	10712	1
5074.0	3	6	9	12	2012-06-03	8093	1
5342.0	3	6	9	12	2012-06-03	7661	1
6150.0	3	6	9	12	2012-06-03	8405	1
5816.0	3	6	9	12	2012-06-03	8680	1

- This is result of making promo2.
- Difference with Store dataset’s promo2 is store data set’s promo2 is store promo2 is going on any day or not . But this data’s promo2 shows real promo2 going on that data’s date

GET NEW VARIABLE FROM COPMPETITIONOPENSINCE[YEAR/MONTH]

COMPETITIONOPEN

```
#I make CompetitionPresent value 1 or 0
#because competitionDistance only 3 null value this mean 3 store don't have competition
#below process is made for competition store made in between year 2013 to 2015
#after CompetitionOpenDate , Competition Present value is 1
len(store[store['CompetitionOpenSinceYear'] ==2013]),len(
    store[store['CompetitionOpenSinceYear'] ==2014]),len(
    store[store['CompetitionOpenSinceYear'] ==2015])
```

```
(83, 70, 38)
```

- There are some store in data set's interval(2013-01 ~2015-07) open CompetitionStore open.Above 191(83+70+38 store) is that store And don't have any competition Store.
- So I make column CompetitionOpen column. Value ==1 -> Competition is present date. And value ==0 Competition is not present.

COMPETITIONOPEN

```
sales
```

	Store	Date	Sales	Year	Month	After1WeekSales	AfterPromo2Sales	Promo2	CompetitionOpen
0	1	2013-01-09	5471.0	2013.0	1.0	5530.0	0.0	0.0	1
1	1	2013-01-10	4892.0	2013.0	1.0	4327.0	0.0	0.0	1
2	1	2013-01-11	4881.0	2013.0	1.0	4486.0	0.0	0.0	1
3	1	2013-01-12	4952.0	2013.0	1.0	4997.0	0.0	0.0	1
4	1	2013-01-14	4717.0	2013.0	1.0	7176.0	0.0	0.0	1
...	...	...	...	...	...	...	...	...	...
764195	1115	2015-07-28	8093.0	2015.0	7.0	5074.0	8093.0	1.0	1
764196	1115	2015-07-29	7661.0	2015.0	7.0	5342.0	7661.0	1.0	1
764197	1115	2015-07-30	8405.0	2015.0	7.0	6150.0	8405.0	1.0	1
764198	1115	2015-07-31	8680.0	2015.0	7.0	5816.0	8680.0	1.0	1
764199	262	NaT	NaN	NaN	NaN	NaN	NaN	NaN	1

764200 rows × 9 columns

- **This is the data with CompetitionOpen column**

GET NEW VARIABLE FROM HISTORICAL SALES OR NUMBER OF CUSTOMERS

MOVING AVERAGE CUSTOMERS AND SALES

```
#making predict_customers by use moving average of customer number data
moving_average = train[['Date','Store','Customers','Sales']]
moving_average = moving_average.sort_values(['Store','Date'])
moving_average.index = moving_average['Date']
```

```
new_variables = moving_average.groupby('Store')['Customers'].rolling(window='7D').mean().to_f
new_variables['Customers2W'] = moving_average.groupby('Store')['Customers'].rolling(window='1

new_variables['Sales1W'] = moving_average.groupby('Store')['Sales'].rolling(window='7D').mean
new_variables['Sales2W'] = moving_average.groupby('Store')['Sales'].rolling(window='14D').mea

new_variables=new_variables.reset_index()

new_variables['Date']=new_variables['Date']+pd.to_timedelta('7D')

new_variables.head()
```

	Store	Date	Customers1W	Customers2W	Sales1W	Sales2W
0	1	2013-01-09	668.000000	668.000000	5530.0	5530.0
1	1	2013-01-10	623.000000	623.000000	4928.5	4928.5
2	1	2013-01-11	621.666667	621.666667	4781.0	4781.0
3	1	2013-01-12	625.000000	625.000000	4835.0	4835.0
4	1	2013-01-14	657.000000	657.000000	5303.2	5303.2

- **Depending on class's Programming, I make customers and sales moving average. Period is 1week and 2week**

DATASET

```
sales
```

	Store	Date	After1WeekSales	AfterPromo2Sales	Promo2	CompetitionOpen	Customers1W	Customers2W	Sale
0	1	2013-01-15	5580.0	0.0	0.0	1	656.500000	656.500000	5349.33
1	1	2013-01-16	5471.0	0.0	0.0	1	649.500000	652.142857	5339.50
2	1	2013-01-17	4892.0	0.0	0.0	1	655.666667	647.500000	5433.66
3	1	2013-01-18	4881.0	0.0	0.0	1	651.166667	641.333333	5499.50
4	1	2013-01-19	4952.0	0.0	0.0	1	653.000000	641.800000	5492.00
...	...	...	...	...	...	...	...	...	
758875	1115	2015-07-27	6083.0	10712.0	1.0	1	476.000000	464.583333	6902.00
758876	1115	2015-07-28	5074.0	8093.0	1.0	1	453.333333	463.750000	6487.33
758877	1115	2015-07-29	5342.0	7661.0	1.0	1	442.000000	461.166667	6371.16
758878	1115	2015-07-30	6150.0	8405.0	1.0	1	438.666667	462.750000	6297.83
758879	1115	2015-07-31	5816.0	8680.0	1.0	1	421.666667	463.833333	5954.83

758880 rows × 14 columns

- **The final sales dataset is this**

MODELING

MODELING

```
catvar = ['StoreType','Assortment','DayOfWeek']
for c in catvar:
    temp =pd.get_dummies(sales[c],prefix=c,drop_first=True)
    sales=pd.concat((sales,temp),axis=1)
sales.drop(columns = catvar+['Store'], inplace =True)
```

- Before making modeling , there are 5 categorical variables in sales datasets
- Promo2 and promo value is only 1 and 0.
- So I make 3 variables's dummy variables

MODELING

```
linear = LinearRegression()
ridge = Ridge()
lasso = Lasso()

train_set = sales[sales['Date']<=pd.to_datetime('20141231')]
val_set = sales[sales['Date']>pd.to_datetime('20141231')]
train_set.drop(columns=['Date'],inplace=True)
val_set.drop(columns=['Date'],inplace=True)
```

```
trainX = train_set.drop(columns = ['After1WeekSales'])
trainy = train_set['After1WeekSales']

valX = val_set.drop(columns = ['After1WeekSales'])
valy = val_set['After1WeekSales']
```

```
linear.fit(trainX,trainy)
linear.score(valX,valy)
```

```
0.7074781879005773
```

- Train set and validation set distinguised by criteria is 20141231
- For comparing each linear model, I make linear, ridge and lasso
- And linear score is like that

MODELING

```
#for scaling problem present in Lasso And Ridge model, we should standardization data for Reg
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

std_trainX = scaler.fit(trainX).transform(trainX)
std_trainy = (trainy - trainy.mean())/trainy.std()


std_valX = scaler.fit(valX).transform(valX)
std_valy = (valy - valy.mean())/valy.std()
```

```
alphas = np.logspace(-3,3,30)

result = pd.DataFrame(index=alphas,columns=['Ridge','Lasso'])

for alpha in alphas:
    ridge.alpha = alpha
    lasso.alpha = alpha
    ridge.fit(std_trainX,std_trainy)
    result.loc[alpha,'Ridge'] =ridge.score(std_valX,std_valy)
    lasso.fit(std_trainX,std_trainy)
    result.loc[alpha,'Lasso'] =lasso.score(std_valX,std_valy)
```

```
result
#this is when alpha is 0.001
```

	Ridge	Lasso
0.001000	0.697186	0.697202
0.001610	0.697186	0.697123
0.002593	0.697186	0.696901
0.004175	0.697186	0.696302
0.006723	0.697186	0.694739
0.010826	0.697186	0.692153
0.017433	0.697186	0.686688
0.028072	0.697186	0.67751
0.045204	0.697186	0.665633
0.072790	0.697186	0.651154
0.117210	0.697186	0.617302
0.188739	0.697186	0.55682
0.303920	0.697186	0.498424
0.489390	0.697186	0.350615
0.788046	0.697187	0.0
1.268961	0.697187	0.0
2.043360	0.697187	0.0
3.290345	0.697187	0.0
5.298317	0.697188	0.0

- For applying modeling in ridge and lasso, I should standardization because of scaling problem. So I standardization train set and validation set.
- And for finding optimized alpha value of ridge and lasso, I run that code.
- Result is like that. When I have alpha = 0.001 , the score of each model is highest

MODELING

```
from sklearn.model_selection import KFold
kfold = KFold(n_splits=5,shuffle=True)
linear_score = []
ridge_score = []
lasso_score = []
trainX.reset_index(drop=True,inplace=True)
trainy = trainy.reset_index(drop=True)
std_trainy = std_trainy.reset_index(drop=True)


for train_idx, test_idx in kfold.split(trainX):
    train_X, test_X = trainX.iloc[train_idx],trainX.iloc[test_idx]
    std_train_X,std_test_X = std_trainX[train_idx],std_trainX[test_idx]

    train_y, test_y = trainy.iloc[train_idx],trainy.iloc[test_idx]
    std_train_y,std_test_y = std_trainy[train_idx],std_trainy[test_idx]

    linear.fit(train_X,train_y)
    linear_score.append(linear.score(test_X,test_y))

    ridge.fit(std_train_X,std_train_y)
    ridge_score.append(ridge.score(std_test_X,std_test_y))

    lasso.fit(std_train_X,std_train_y)
    lasso_score.append(lasso.score(std_test_X,std_test_y))
```

```
: print(np.mean(linear_score))
  print(np.mean(ridge_score))
  print(np.mean(lasso_score))

  0.7154138754081092
  0.7154138754111754
  0.715352745533575
```

- I use 5-fold validation on training set to find best model.
- The mean of each model's score is like that. That is a little difference in score
- Ridge model is a bit higher. So I select Ridge model

THANK YOU