AI Capstone project- prepared and Submitted By Vishal Yerme

Retail

Course-end Project 3 : Retail (by Vishal Yerme)

Description

Problem Statement:

Demand Forecast is one of the key tasks in Supply Chain and Retail Domain in general. It is key in effective operation and optimization of retail supply chain. Effectively solving this problem requires knowledge about a wide range of tricks in Data Sciences and good understanding of ensemble techniques. You are required to predict sales for each Store-Day level for one month. All the features will be provided and actual sales that happened during that month will also be provided for model evaluation.

Project Task: Week 1

Exploratory Data Analysis (EDA) and Linear Regression:

1. **Transform the variables by using data manipulation techniques like, One-Hot Encoding**
2. **Perform an EDA (Exploratory Data Analysis) to see the impact of variables over Sales.**
3. **Apply Linear Regression to predict the forecast and evaluate different accuracy metrices like RMSE (Root Mean Squared Error) and MAE(Mean Absolute Error) and determine which metric makes more sense. Can there be a better accuracy metric?**
   - **Train a single model for all stores, using storeId as a feature.**
   - **Train separate model for each store.**
   - **Which performs better and Why? [In the first case, parameters are shared and not very free but not in second case]**
   - **Try Ensemble of b) and c). What are the findings?**
   - **Use Regularized Regression. It should perform better in an unseen test set. Any insights?**

- **Open-ended modeling to get possible predictions.**

Other Regression Techniques:

1. **When store is closed, sales = 0. Can this insight be used for Data Cleaning? Perform this and retrain the model. Any benefits of this step?**
2. **Use Non-Linear Regressors like Random Forest or other Tree-based Regressors.**
    - **Train a single model for all stores, where storeId can be a feature.**
    - **Train separate models for each store.**

Note: Dimensional Reduction techniques like, PCA and Tree's Hyperparameter Tuning will be required. Cross-validate to find the best parameters. Infer the performance of both the models.

1. **Compare the performance of Linear Model and Non-Linear Model from the previous observations. Which performs better and why?**
2. **Train a Time-series model on the data taking time as the only feature. This will be a store-level training.**
    - **Identify yearly trends and seasonal months**

In [1]:
```
pip install nbconvert
```

```
Requirement already satisfied: nbconvert in c:\users\prasath\anaconda3\lib\site-p
ackages (5.6.1)
Requirement already satisfied: jupyter-core in c:\users\prasath\anaconda3\lib\sit
e-packages (from nbconvert) (4.6.1)
Requirement already satisfied: jinja2>=2.4 in c:\users\prasath\anaconda3\lib\site
-packages (from nbconvert) (2.11.1)
Requirement already satisfied: bleach in c:\users\prasath\anaconda3\lib\site-pack
ages (from nbconvert) (3.1.0)
Requirement already satisfied: testpath in c:\users\prasath\anaconda3\lib\site-pa
ckages (from nbconvert) (0.4.4)
Requirement already satisfied: traitlets>=4.2 in c:\users\prasath\anaconda3\lib\s
ite-packages (from nbconvert) (4.3.3)
Requirement already satisfied: entrypoints>=0.2.2 in c:\users\prasath\anaconda3\l
ib\site-packages (from nbconvert) (0.3)
Requirement already satisfied: pandocfilters>=1.4.1 in c:\users\prasath\anaconda3
\lib\site-packages (from nbconvert) (1.4.2)
Requirement already satisfied: defusedxml in c:\users\prasath\anaconda3\lib\site-
packages (from nbconvert) (0.6.0)
Requirement already satisfied: pygments in c:\users\prasath\anaconda3\lib\site-pa
ckages (from nbconvert) (2.5.2)
Requirement already satisfied: mistune<2,>=0.8.1 in c:\users\prasath\anaconda3\li
b\site-packages (from nbconvert) (0.8.4)
Requirement already satisfied: nbformat>=4.4 in c:\users\prasath\anaconda3\lib\si
te-packages (from nbconvert) (5.0.4)
Requirement already satisfied: pywin32>=1.0; sys_platform == "win32" in c:\users
\prasath\anaconda3\lib\site-packages (from jupyter-core->nbconvert) (227)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\prasath\anaconda3\lib
\site-packages (from jinja2>=2.4->nbconvert) (1.1.1)
Requirement already satisfied: six>=1.9.0 in c:\users\prasath\anaconda3\lib\site-
packages (from bleach->nbconvert) (1.14.0)
Requirement already satisfied: webencodings in c:\users\prasath\anaconda3\lib\sit
e-packages (from bleach->nbconvert) (0.5.1)
Requirement already satisfied: decorator in c:\users\prasath\anaconda3\lib\site-p
ackages (from traitlets>=4.2->nbconvert) (4.4.1)
Requirement already satisfied: ipython-genutils in c:\users\prasath\anaconda3\lib
\site-packages (from traitlets>=4.2->nbconvert) (0.2.0)
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in c:\users\prasath\anacon
da3\lib\site-packages (from nbformat>=4.4->nbconvert) (3.2.0)
Requirement already satisfied: attrs>=17.4.0 in c:\users\prasath\anaconda3\lib\si
te-packages (from jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert) (19.3.0)
Requirement already satisfied: setuptools in c:\users\prasath\anaconda3\lib\site-
packages (from jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert) (45.2.0.post202
00210)
Requirement already satisfied: importlib-metadata; python_version < "3.8" in c:\u
sers\prasath\anaconda3\lib\site-packages (from jsonschema!=2.5.0,>=2.4->nbformat>
=4.4->nbconvert) (1.5.0)
Requirement already satisfied: pyrsistent>=0.14.0 in c:\users\prasath\anaconda3\l
ib\site-packages (from jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert) (0.15.
7)
Requirement already satisfied: zipp>=0.5 in c:\users\prasath\anaconda3\lib\site-p
ackages (from importlib-metadata; python_version < "3.8"->jsonschema!=2.5.0,>=2.4
->nbformat>=4.4->nbconvert) (2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

from sklearn.linear_model import LinearRegression,Ridge,ElasticNet,Lasso
from sklearn.metrics import mean_squared_error,mean_absolute_error,accuracy_score
```

```
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.decomposition import PCA

from statsmodels.tsa.stattools import adfuller
from pylab import rcParams
import statsmodels.api as sm
from statsmodels.tsa.arima_model import ARIMA

from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import GridSearchCV
```

```
Using TensorFlow backend.
```

In [2]:

```
train = pd.read_csv("train_data.csv")
train.head()
```

```
C:\Users\Public\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:304
9: DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or
set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Out[2]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
0	1	2	2015-06-30	5735	568	1	1	0	0
1	2	2	2015-06-30	9863	877	1	1	0	0
2	3	2	2015-06-30	13261	1072	1	1	0	1
3	4	2	2015-06-30	13106	1488	1	1	0	0
4	5	2	2015-06-30	6635	645	1	1	0	0

In [3]:

```
test_val= pd.read_csv("test_data_hidden.csv")
test_val.head()
```

Out[3]:

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

In [4]:
```
test= pd.read_csv("test_data.csv")
test.head()
```

Out[4]:

	Store	DayOfWeek	Date	Open	Promo	StateHoliday	SchoolHoliday
0	1	5	2015-07-31	1	1	0	1
1	2	5	2015-07-31	1	1	0	1
2	3	5	2015-07-31	1	1	0	1
3	4	5	2015-07-31	1	1	0	1
4	5	5	2015-07-31	1	1	0	1

In [5]:
```
train_1 = train.copy()
test_val_1 = test_val.copy()
test_1 = test.copy()
```

In [38]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 982644 entries, 0 to 982643
Data columns (total 12 columns):
Store            982644 non-null int64
DayOfWeek        982644 non-null int64
Date             982644 non-null object
Sales            982644 non-null int64
Customers        982644 non-null int64
Open             982644 non-null int64
Promo            982644 non-null int64
StateHoliday     982644 non-null object
SchoolHoliday    982644 non-null int64
year             982644 non-null int64
month            982644 non-null int64
day              982644 non-null int64
dtypes: int64(10), object(2)
memory usage: 90.0+ MB
```

In [16]:
```
train.isna().sum()
```

Out[16]:
```
Store            0
DayOfWeek        0
Date             0
Sales            0
Customers        0
Open             0
Promo            0
StateHoliday     0
SchoolHoliday    0
dtype: int64
```

In [17]:
```
test.isna().sum()
```

Out[17]:
```
Store            0
DayOfWeek        0
Date             0
Open             0
Promo            0
StateHoliday     0
SchoolHoliday    0
dtype: int64
```

In [14]:
```
train.DayOfWeek.value_counts()
```

Out[14]:
```
2    141204
7    140270
6    140270
5    140270
4    140270
1    140270
3    140090
Name: DayOfWeek, dtype: int64
```

In [10]:
```
test.DayOfWeek.value_counts()
```

Out[10]:
```
5    5575
4    5575
3    5575
7    4460
6    4460
2    4460
1    4460
Name: DayOfWeek, dtype: int64
```

In [11]:
```
train.Open.value_counts()
```

Out[11]:
```
1    814204
0    168440
Name: Open, dtype: int64
```

In [12]:
```
test.Open.value_counts()
```

Out[12]:
```
1    30188
0     4377
Name: Open, dtype: int64
```

In [14]:
```
train.Promo.value_counts()
```

Out[14]:
```
0    609059
1    373585
Name: Promo, dtype: int64
```

In [13]:
```
test.Promo.value_counts()
```

Out[13]:
```
0    20070
1    14495
Name: Promo, dtype: int64
```

In [43]:
```
train.StateHoliday.unique()
```

Out[43]:
```
array(['0', 'a', 'b', 'c', 0], dtype=object)
```

In [16]:
```
test.StateHoliday.value_counts()
```

Out[16]:
```
0    34565
Name: StateHoliday, dtype: int64
```

In [17]:
```
train.SchoolHoliday.value_counts()
```

Out[17]:
```
0    813700
1    168944
Name: SchoolHoliday, dtype: int64
```

In [18]:
```
test.SchoolHoliday.value_counts()
```

Out[18]:
```
0    21788
1    12777
Name: SchoolHoliday, dtype: int64
```

In [ ]:
```
train.Date.unique()
```

In [37]:
```
train['year'].value_counts()
```

Out[37]:
```
2013    406974
2014    373855
2015    201815
Name: year, dtype: int64
```

In [5]:
```
test_val.sort_values(['Store'],inplace=True)
test.sort_values(['Store'],inplace=True)
combi = train.append(test_val , ignore_index=True)
print(combi.shape)
combi =combi.append(test , ignore_index=True)
print(combi.shape)
combi['year']=pd.to_datetime(combi['Date'],format='%Y-%m-%d').dt.year
combi['month']=pd.to_datetime(combi['Date'],format='%Y-%m-%d').dt.month
combi['day']=pd.to_datetime(combi['Date'],format='%Y-%m-%d').dt.day
combi['year'] = combi.year.replace({2013 : 0, 2014 : 1 , 2015 : 2 })
combi['StateHoliday'] = combi.StateHoliday.replace({'0' : 0, 'a' : 1 , 'b' : 2 ,'
combi.head()
```

```
(1017209, 9)
C:\Users\Public\Anaconda3\lib\site-packages\pandas\core\frame.py:6692: FutureWarn
ing: Sorting because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  sort=sort)
(1051774, 9)
```

Out[5]:

	Customers	Date	DayOfWeek	Open	Promo	Sales	SchoolHoliday	StateHoliday	Store
0	568.0	2015-06-30	2	1	1	5735.0	0	0	1
1	877.0	2015-06-30	2	1	1	9863.0	0	0	2
2	1072.0	2015-06-30	2	1	1	13261.0	1	0	3
3	1488.0	2015-06-30	2	1	1	13106.0	0	0	4
4	645.0	2015-06-30	2	1	1	6635.0	0	0	5

In [6]:

```
combi1= pd.get_dummies(combi,columns=['DayOfWeek', 'Open', 'Promo','StateHoliday'
# combi1=pd.read_csv('combi1.csv')
# combi1.drop(['Unnamed: 0'],axis=True,inplace=True)
combi1.head()
```

Out[6]:

	Customers	Date	Sales	DayOfWeek_2	DayOfWeek_3	DayOfWeek_4	DayOfWeek_5	Day(
0	568.0	2015-06-30	5735.0	1	0	0	0	
1	877.0	2015-06-30	9863.0	1	0	0	0	
2	1072.0	2015-06-30	13261.0	1	0	0	0	
3	1488.0	2015-06-30	13106.0	1	0	0	0	
4	645.0	2015-06-30	6635.0	1	0	0	0	

5 rows × 1172 columns

In [6]:

```
combi2= pd.get_dummies(combi,columns=['DayOfWeek', 'Open', 'Promo','StateHoliday'
combi2.head()
```

Out[6]:

	Customers	Date	Sales	Store	DayOfWeek_2	DayOfWeek_3	DayOfWeek_4	DayOfWeek_5
0	568.0	2015-06-30	5735.0	1	1	0	0	(
1	877.0	2015-06-30	9863.0	2	1	0	0	(
2	1072.0	2015-06-30	13261.0	3	1	0	0	(
3	1488.0	2015-06-30	13106.0	4	1	0	0	(
4	645.0	2015-06-30	6635.0	5	1	0	0	(

5 rows × 59 columns

In [8]:
```
train.shape,test_val.shape,test.shape
```

Out[8]:
```
((982644, 9), (34565, 9), (34565, 7))
```

In [8]:
```
train1 = combi1.iloc[:982644].reset_index(drop=True)
test_val1 = combi1.iloc[982644:1017209].reset_index(drop=True)
test1 = combi1.iloc[1017209:].reset_index(drop=True)
train1.shape,test_val1.shape,test1.shape
```

Out[8]:
```
((982644, 1172), (34565, 1172), (34565, 1172))
```

In [7]:
```
train2 = combi2.iloc[:982644].reset_index(drop=True)
test_val2 = combi2.iloc[982644:1017209].reset_index(drop=True)
test2 = combi2.iloc[1017209:].reset_index(drop=True)
train2.shape,test_val2.shape,test2.shape
```

Out[7]:
```
((982644, 59), (34565, 59), (34565, 59))
```

In [27]:
```
train.corr()['Sales']
```

Out[27]:
```
Store               0.005338
Sales               1.000000
Customers           0.895700
month               0.048435
day                -0.014450
DayOfWeek_2         0.132176
DayOfWeek_3         0.081984
DayOfWeek_4         0.048159
DayOfWeek_5         0.099717
DayOfWeek_6         0.010149
DayOfWeek_7        -0.587966
Open_1              0.679248
Promo_1             0.451383
StateHoliday_1     -0.205744
StateHoliday_2     -0.119044
StateHoliday_3     -0.093835
SchoolHoliday_1     0.076141
year_1              0.014717
year_2              0.009503
Name: Sales, dtype: float64
```

In [19]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
x=train._get_numeric_data()
vif=pd.DataFrame()
vif["VIF"]=[variance_inflation_factor(x.values,i) for i in range(x.shape[1])]
vif["features"]=x.columns
vif
```

Out[19]:

	VIF	features
0	3.945474	Store
1	22.878068	Sales
2	15.470602	Customers
3	4.541866	month
4	4.257005	day
5	2.011796	DayOfWeek_2
6	2.025271	DayOfWeek_3
7	2.046647	DayOfWeek_4
8	1.995472	DayOfWeek_5
9	2.271216	DayOfWeek_6
10	2.532029	DayOfWeek_7
11	21.389022	Open_1
12	2.605402	Promo_1
13	1.300803	StateHoliday_1
14	1.136446	StateHoliday_2
15	1.164511	StateHoliday_3
16	1.376474	SchoolHoliday_1
17	1.910265	year_1
18	1.680706	year_2

Linear Regression with STORE as feature

In [26]:
```
Y_train = train1['Sales']
Y_val = test_val1['Sales']
```

In [22]:
```
X_train = train1.drop(['Sales','Date','Customers'],axis=1).values
X_val = test_val1.drop(['Sales','Date','Customers'],axis=1).values
lr_1 = LinearRegression()
lr_1.fit(X_train,Y_train)
Y_pred1 = lr_1.predict(X_val)
print('MSE',np.sqrt(mean_squared_error(Y_pred1,Y_val)))
print('MAE',mean_absolute_error(Y_pred1,Y_val))
print('train model score',lr_1.score(X_train,Y_train))
print('test model score',lr_1.score(X_val,Y_val))
```

```
MSE 1428.9181706827264
MAE 1051.5557239043858
train model score 0.8365645595889427
test model score 0.8430035399815864
```

Linear Regression without STORE as feature

In [13]:
```
X_train1 = train2.drop(['Sales','Date','Customers'],axis=1).values
X_val1 = test_val2.drop(['Sales','Date','Customers'],axis=1).values
lr_2 = LinearRegression()
lr_2.fit(X_train1,Y_train)
Y_pred2 = lr_2.predict(X_val1)
print('MSE',np.sqrt(mean_squared_error(Y_pred2,Y_val)))
print('MAE',mean_absolute_error(Y_pred2,Y_val))
print('train model score',lr_2.score(X_train1,Y_train))
print('test model score',lr_2.score(X_val1,Y_val))
```

```
MSE 2520.0716734481657
MAE 1731.7047379515243
train model score 0.564626733803036
test model score 0.5116840301951024
```

Linear Regression - Separate model for each STORE

In [19]:
```
Y_pred3=np.zeros(test_val.shape[0])

train_store = train2.groupby(['Store'])
test_store = test_val2.groupby(['Store'])

for i in range(1,1116):
    a = train_store.get_group(i)
    b = test_store.get_group(i)
    X_train = a.drop(['Sales','Date','Store','Customers'],axis=1).values
    X_val = b.drop(['Sales','Date','Store','Customers'],axis=1).values
    Y_train = a['Sales']
    #Y_val = b['Sales']
    lr = LinearRegression()
    lr.fit(X_train,Y_train)
    pred = lr.predict(X_val)
    i=0
    for j in b.index:
        Y_pred3[j]=pred[i]
        i+=1
print('MSE',np.sqrt(mean_squared_error(Y_pred3,Y_val)))
print('MAE',mean_absolute_error(Y_pred3,Y_val))
```

```
MSE 2886004774448802.5
MAE 65858982569725.73
```

So from the above 3 models we can conclude that the model perform better with 'Store' as feature. Also the average of all the separate model based on Store Id is the worst model.

Average Ensemble Model of first and second model

In [23]:
```
final_pred=(Y_pred1+Y_pred2)/2
print('MSE',np.sqrt(mean_squared_error(final_pred,Y_val)))
print('MAE',mean_absolute_error(final_pred,Y_val))
```

```
MSE 1786.572723527162
MAE 1295.5365594353661
```

Weighted Average Ensemble Model of first and second model

In [24]:
```
final_pred=Y_pred1*0.7+Y_pred2*0.3
print('MSE',np.sqrt(mean_squared_error(final_pred,Y_val)))
print('MAE',mean_absolute_error(final_pred,Y_val))
```

```
MSE 1578.2148177872348
MAE 1163.7769636251703
```

Regularization of 1st Model

In [32]:
```
X_train = train1.drop(['Sales','Date','Customers'],axis=1).values
X_val = test_val1.drop(['Sales','Date','Customers'],axis=1).values
rr =Ridge(alpha=10)
rr.fit(X_train,Y_train)
Y_pred1 = rr.predict(X_val)
print('MSE',np.sqrt(mean_squared_error(Y_pred1,Y_val)))
print('MAE',mean_absolute_error(Y_pred1,Y_val))
print('train model score',rr.score(X_train,Y_train))
print('test model score',rr.score(X_val,Y_val))
```

```
MSE 1431.5196149136414
MAE 1053.9640706528064
train model score 0.8363551306623415
test model score 0.8424313738215597
```

Regualrization technique is not enhancing the performance.

Project Task: Week 2

Implementing Neural Networks:

- **Train a LSTM on the same set of features and compare the result with traditional time-series model.**
- **Comment on the behavior of all the models you have built so far**
- **Cluster stores using sales and customer visits as features. Find out how many clusters or groups are possible. Also visualize the results.**
- **Is it possible to have separate prediction models for each cluster? Compare results with the previous models.**

In [5]:
```
train=train[train.Open==1]
shape1=train.shape[0]
print(train.shape[0])
combi = train.append(test_val , ignore_index=True,sort=False)
shape2=combi.shape[0]
print(combi.shape)
combi =combi.append(test , ignore_index=True,sort=False)
print(combi.shape)
combi['year']=pd.to_datetime(combi['Date'],format='%Y-%m-%d').dt.year
combi['month']=pd.to_datetime(combi['Date'],format='%Y-%m-%d').dt.month
combi['day']=pd.to_datetime(combi['Date'],format='%Y-%m-%d').dt.day
combi['year'] = combi.year.replace({2013 : 0, 2014 : 1 , 2015 : 2 })
combi['StateHoliday'] = combi.StateHoliday.replace({'0' : 0, 'a' : 1 , 'b' : 2 ,'
#with Store Id as features
combi1= pd.get_dummies(combi,columns=['DayOfWeek', 'Promo','StateHoliday', 'Schoo
#without Store Id as features
combi2= pd.get_dummies(combi,columns=['DayOfWeek', 'Promo','StateHoliday', 'Schoo
```

```
print(train.shape,test_val.shape,test.shape)

train1 = combi1.iloc[:shape1].reset_index(drop=True)
test_val1 = combi1.iloc[shape1:shape2].reset_index(drop=True)
test1 = combi1.iloc[shape2:].reset_index(drop=True)
print(train1.shape,test_val1.shape,test1.shape)

train2 = combi2.iloc[:shape1].reset_index(drop=True)
test_val2 = combi2.iloc[shape1:shape2].reset_index(drop=True)
test2 = combi2.iloc[shape2:].reset_index(drop=True)
print(train2.shape,test_val2.shape,test2.shape)
```

```
814204
(848769, 9)
(883334, 9)
(814204, 9) (34565, 9) (34565, 7)
(814204, 1172) (34565, 1172) (34565, 1172)
(814204, 59) (34565, 59) (34565, 59)
```

In [6]:
```
Y_train = train1['Sales']
Y_val = test_val1['Sales']
```

Model1

In [41]:
```
X_train = train1.drop(['Sales','Date','Open','Customers'],axis=1).values
X_val = test_val1.drop(['Sales','Date','Open','Customers'],axis=1).values

lr = LinearRegression()
lr.fit(X_train,Y_train)
pred1 = lr.predict(X_val)

ind=test_val[test_val.Open==0].index
for i in ind:
    pred1[i] = 0

print('MSE',np.sqrt(mean_squared_error(pred1,Y_val)))
print('MAE',mean_absolute_error(pred1,Y_val))

# MSE 1428.9181706827264
# MAE 1051.555723904386
```

```
MSE 1229.9197388602236
MAE 865.6514844033625
```

Model2

In [47]:
```
X_train1 = train2.drop(['Sales','Date','Open','Customers'],axis=1).values
X_val1 = test_val2.drop(['Sales','Date','Open','Customers'],axis=1).values

lr = LinearRegression()
lr.fit(X_train1,Y_train)
pred2 = lr.predict(X_val1)

ind=test_val[test_val.Open==0].index
for i in ind:
    pred2[i] = 0

print('MSE',np.sqrt(mean_squared_error(pred2,Y_val)))
print('MAE',mean_absolute_error(pred2,Y_val))

# MSE 2520.0716734481657
# MAE 1731.704737951524
```

```
MSE 2530.1635832559
MAE 1725.719012601922
```

Model3

In [48]:
```
pred3=np.zeros(test_val.shape[0])

train_store = train2.groupby(['Store'])
test_store = test_val2.groupby(['Store'])

for i in range(1,1116):
    a = train_store.get_group(i)
    b = test_store.get_group(i)
    X_train = a.drop(['Sales','Date','Store','Customers','Open'],axis=1).values
    X_val = b.drop(['Sales','Date','Store','Customers','Open'],axis=1).values
    Y_train = a['Sales']
    lr = LinearRegression()
    lr.fit(X_train,Y_train)
    pred = lr.predict(X_val)
    i=0
    ind=b[b['Open']==0].index
    for j in b.index:
        if(j in ind):
            pred3[j]=0
        else:
            pred3[j]=pred[i]
        i+=1
print('MSE',np.sqrt(mean_squared_error(pred3,Y_val)))
print('MAE',mean_absolute_error(pred3,Y_val))

# MSE 2886004774448802.0
# MAE 65858982569725.75
```

```
MSE 1014.9293535430203
MAE 670.5513943441184
```

From the above model,we can see the performance has increased due to data cleaning except in 2nd model which remains almost same. In this case third model has outperformed which was earlier worst model.

In [49]:
```
train_store = train2.groupby(['Store'])
test_store = test_val2.groupby(['Store'])

for i in range(1,1116):
    a = train_store.get_group(i)
    b = test_store.get_group(i)
    X_train = a.drop(['Sales','Date','Store','Customers','Open'],axis=1).values
    X_val = b.drop(['Sales','Date','Store','Customers','Open'],axis=1).values
    Y_train = a['Sales']
    lr = Ridge(alpha=20)
    lr.fit(X_train,Y_train)
    pred = lr.predict(X_val)
    i=0
    ind=b[b['Open']==0].index
    for j in b.index:
        if(j in ind):
            pred3[j]=0
        else:
            pred3[j]=pred[i]
        i+=1
print('MSE',np.sqrt(mean_squared_error(pred3,Y_val)))
print('MAE',mean_absolute_error(pred3,Y_val))
```

```
MSE 930.9742188387742
MAE 629.3727064444969
```

Only 3rd model's performance is increasing with regularization

model3: MSE 1014.9293535430203 MAE 670.5513943441184

after reegularization: MSE 930.9742188387742 MAE 629.3727064444969

Random Forest Regression

In [50]:
```
#With Store as Feature
X_train = train1.drop(['Sales','Date','Open','Customers'],axis=1).values
X_val = test_val1.drop(['Sales','Date','Open','Customers'],axis=1).values

clf = RandomForestRegressor(n_estimators=500,max_features='sqrt',max_depth=6,rand
clf.fit(X_train,Y_train)
pred1 = clf.predict(X_val)

ind=test_val[test_val.Open==0].index
for i in ind:
    pred1[i] = 0

print('MSE',np.sqrt(mean_squared_error(pred1,Y_val)))
print('MAE',mean_absolute_error(pred1,Y_val))
```

```
MSE 2571.8525994831966
MAE 1786.634280806513
```

In [7]:
```
#Without Store as Feature
X_train = train2.drop(['Sales','Date','Open','Customers'],axis=1).values
X_val = test_val2.drop(['Sales','Date','Open','Customers'],axis=1).values
clf = RandomForestRegressor(n_estimators=500,max_features='sqrt',max_depth=6,rand
clf.fit(X_train,Y_train)
pred1 = clf.predict(X_val)

ind=test_val[test_val.Open==0].index
for i in ind:
    pred1[i] = 0

print('MSE',np.sqrt(mean_squared_error(pred1,Y_val)))
print('MAE',mean_absolute_error(pred1,Y_val))
```

```
MSE 2544.663201550362
MAE 1728.0781382597204
```

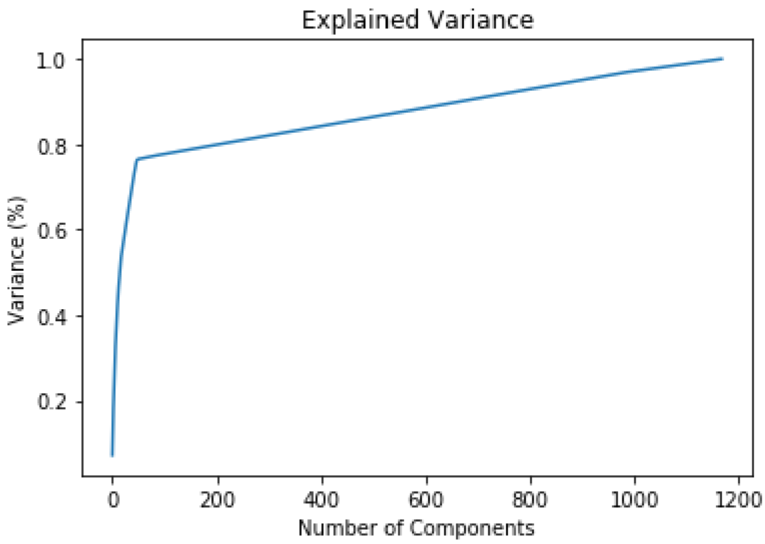
In [8]:
```
#Separate model for each Store
pred3=np.zeros(test_val.shape[0])

train_store = train2.groupby(['Store'])
test_store = test_val2.groupby(['Store'])

for i in range(1,1116):
    a = train_store.get_group(i)
    b = test_store.get_group(i)
    X_train = a.drop(['Sales','Date','Store','Customers','Open'],axis=1).values
    X_val = b.drop(['Sales','Date','Store','Customers','Open'],axis=1).values
    Y_train = a['Sales']
    clf = RandomForestRegressor(n_estimators=500,max_features='sqrt',max_depth=6,
    clf.fit(X_train,Y_train)
    pred = clf.predict(X_val)
    i=0
    ind=b[b['Open']==0].index
```

```
    for j in b.index:
        if(j in ind):
            pred3[j]=0
        else:
            pred3[j]=pred[i]
        i+=1
print('MSE',np.sqrt(mean_squared_error(pred3,Y_val)))
print('MAE',mean_absolute_error(pred3,Y_val))
```

```
MSE 1077.7202738114058
MAE 728.2337472832369
```

PCA

In [13]:
```
X_train = train1.drop(['Sales','Date','Open','Customers'],axis=1).values
X_val = test_val1.drop(['Sales','Date','Open','Customers'],axis=1).values
pca = PCA().fit(X_train)
#Plotting the Cumulative Summation of the Explained Variance
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Variance (%)') #for each component
plt.title('Explained Variance')
plt.show()
# Cumulative Variance explains
# var1=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
# print(var1.shape)
# print(var1)
```



In [14]:
```
X_train = train1.drop(['Sales','Date','Open','Customers'],axis=1).values
X_val = test_val1.drop(['Sales','Date','Open','Customers'],axis=1).values
Y_train = train1['Sales']
Y_val = test_val1['Sales']

pca = PCA(n_components=50)
X_train = pca.fit_transform(X_train)
X_val= pca.transform(X_val)

clf = RandomForestRegressor(n_estimators=500,max_features='sqrt',max_depth=6,rand
clf.fit(X_train,Y_train)
pred1 = clf.predict(X_val)

ind=test_val[test_val.Open==0].index
for i in ind:
    pred1[i] = 0
```

```
print('MSE',np.sqrt(mean_squared_error(pred1,Y_val)))
print('MAE',mean_absolute_error(pred1,Y_val))
```

```
MSE 2516.7922443348452
MAE 1710.1717909599372
```

XGBRegressor

In [15]:
```
#With Store as Feature
X_train = train1.drop(['Sales','Date','Open','Customers'],axis=1).values
X_val = test_val1.drop(['Sales','Date','Open','Customers'],axis=1).values

clf = XGBRegressor(n_estimators=500, learning_rate=0.5,max_depth=6,random_state=0
clf.fit(X_train,Y_train)
pred1 = clf.predict(X_val)

ind=test_val[test_val.Open==0].index
for i in ind:
    pred1[i] = 0

print('MSE',np.sqrt(mean_squared_error(pred1,Y_val)))
print('MAE',mean_absolute_error(pred1,Y_val))
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning: Se
ries.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:588: FutureWarning: Se
ries.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
MSE 1116.6123278288517
MAE 742.5063903587868
```

In [44]:
```
#Without Store as Feature
X_train = train2.drop(['Sales','Date','Open','Customers'],axis=1).values
X_val = test_val2.drop(['Sales','Date','Open','Customers'],axis=1).values
clf = XGBRegressor(n_estimators=500, learning_rate=0.5,max_depth=6,random_state=0
clf.fit(X_train,Y_train)
pred1 = clf.predict(X_val)

ind=test_val[test_val.Open==0].index
for i in ind:
    pred1[i] = 0

print('MSE',np.sqrt(mean_squared_error(pred1,Y_val)))
print('MAE',mean_absolute_error(pred1,Y_val))
```

```
MSE 1138.3182388080122
MAE 764.0298774444434
```

In [45]:
```
#Separate model for each Store
pred3=np.zeros(test_val.shape[0])

train_store = train2.groupby(['Store'])
test_store = test_val2.groupby(['Store'])

for i in range(1,1116):
    a = train_store.get_group(i)
    b = test_store.get_group(i)
    X_train = a.drop(['Sales','Date','Store','Customers','Open'],axis=1).values
    X_val = b.drop(['Sales','Date','Store','Customers','Open'],axis=1).values
    Y_train = a['Sales']
    clf = XGBRegressor(n_estimators=500, learning_rate=0.5,max_depth=6,random_sta
    clf.fit(X_train,Y_train)
    pred = clf.predict(X_val)
```

```
    i=0
    ind=b[b['Open']==0].index
    for j in b.index:
        if(j in ind):
            pred3[j]=0
        else:
            pred3[j]=pred[i]
        i+=1
print('MSE',np.sqrt(mean_squared_error(pred3,Y_val)))
print('MAE',mean_absolute_error(pred3,Y_val))
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning: Se
ries.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:588: FutureWarning: Se
ries.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
MSE 1163.4746405811502
MAE 754.1279379541305
```

In [11]:
```
X_train.shape,Y_train.shape
```

Out[11]:
```
((814204, 50), (754,))
```

In [12]:
```
X_train = train1.drop(['Sales','Date','Customers'],axis=1).values
X_val = test_val1.drop(['Sales','Date','Customers'],axis=1).values
Y_train = train1['Sales']
Y_val = test_val1['Sales']
pca = PCA(n_components=50)
X_train = pca.fit_transform(X_train)
X_val= pca.transform(X_val)

clf = XGBRegressor(n_estimators=500, learning_rate=0.1,max_depth=6,random_state=0
                   booster='gbtree')
clf.fit(X_train,Y_train)
pred1 = clf.predict(X_val)

ind=test_val[test_val.Open==0].index
for i in ind:
    pred1[i] = 0

print('MSE',np.sqrt(mean_squared_error(pred1,Y_val)))
print('MAE',mean_absolute_error(pred1,Y_val))
```

```
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning: Se
ries.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/opt/anaconda3/lib/python3.7/site-packages/xgboost/core.py:588: FutureWarning: Se
ries.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
MSE 3750.853932992984
MAE 2391.859036063734
```

Time-series model

In [34]:
```
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
Train = pd.read_csv("train_data.csv", parse_dates=['Date'], index_col='Date',date
Test_val = pd.read_csv("test_data_hidden.csv", parse_dates=['Date'], index_col='D
Train=Train[['Store','Sales','Open','DayOfWeek']]
Test_val=Test_val[['Store','Sales','Open','DayOfWeek']]
print ('\n Parsed Data:')
Train.sort_values(['Date'],axis=0,inplace=True)
```

```
Test_val.sort_values(['Date'],axis=0,inplace=True)
print (Train.head())
```

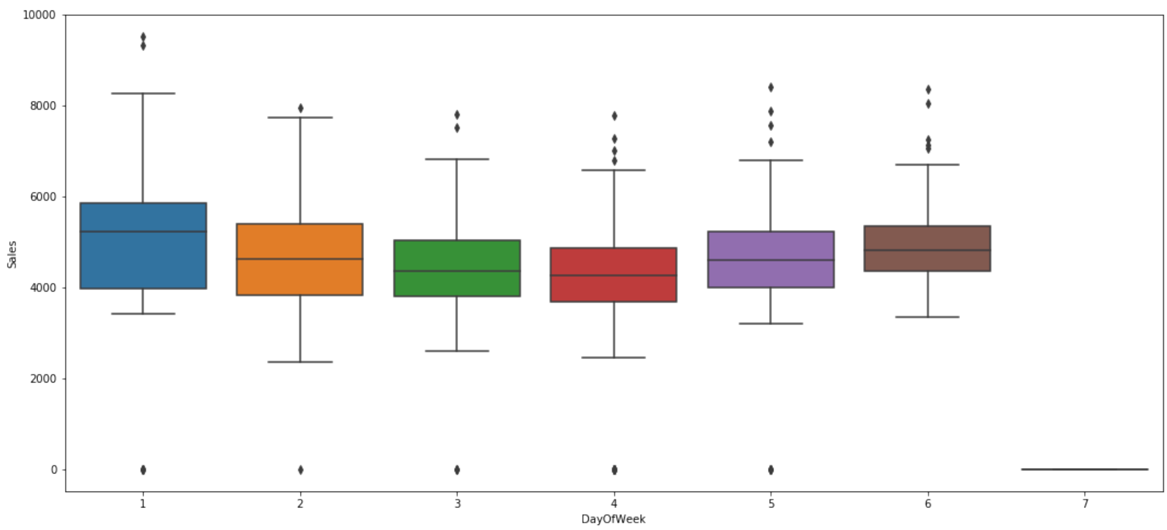
```
/opt/anaconda3/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3049:
DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or set
low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
 Parsed Data:
             Store  Sales  Open  DayOfWeek
Date
2013-01-01    1115      0     0          2
2013-01-01     379      0     0          2
2013-01-01     378      0     0          2
2013-01-01     377      0     0          2
2013-01-01     376      0     0          2
```

Store 1

In [4]:
```
store1=Train[Train.Store==1]
test_store1=Test_val[Test_val.Store==1]
```

In [72]:
```
sns.boxplot(x="DayOfWeek", y="Sales", data=store1)
```

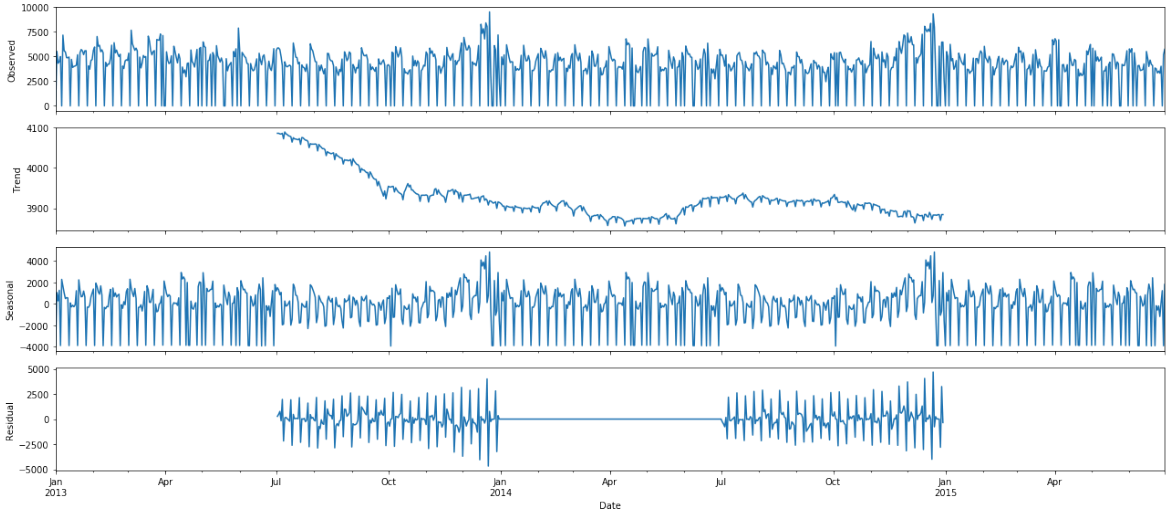
Out[72]:
```
&lt;matplotlib.axes._subplots.AxesSubplot at 0x3d0baab898&gt;
```



Monday=1, Sunday=7.

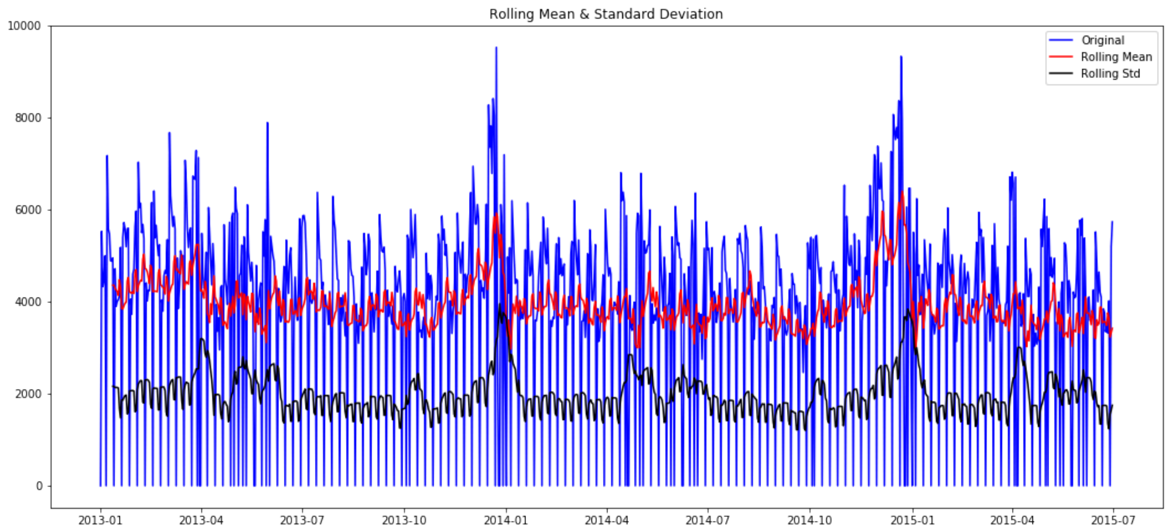
Here we can find on Sunday stores are closed. Monday has little larger sales, Thurdays has little smaller. There's a few outliers on all days(except Sunday) but it is less on Weekdays(1,3)

In [8]:
```
rcParams['figure.figsize'] = 18, 8
decomposition = sm.tsa.seasonal_decompose(store1['Sales'], model='additive',freq=
fig = decomposition.plot()
plt.show()
```



In [9]:
```
def test_stationarity(timeseries):

    #Determing rolling statistics
    rolmean = timeseries.rolling(12).mean()
    rolstd = timeseries.rolling(12).std()

    #Plot rolling statistics:
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')

    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    #Perform Dickey-Fuller test:
    print('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Us
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
```

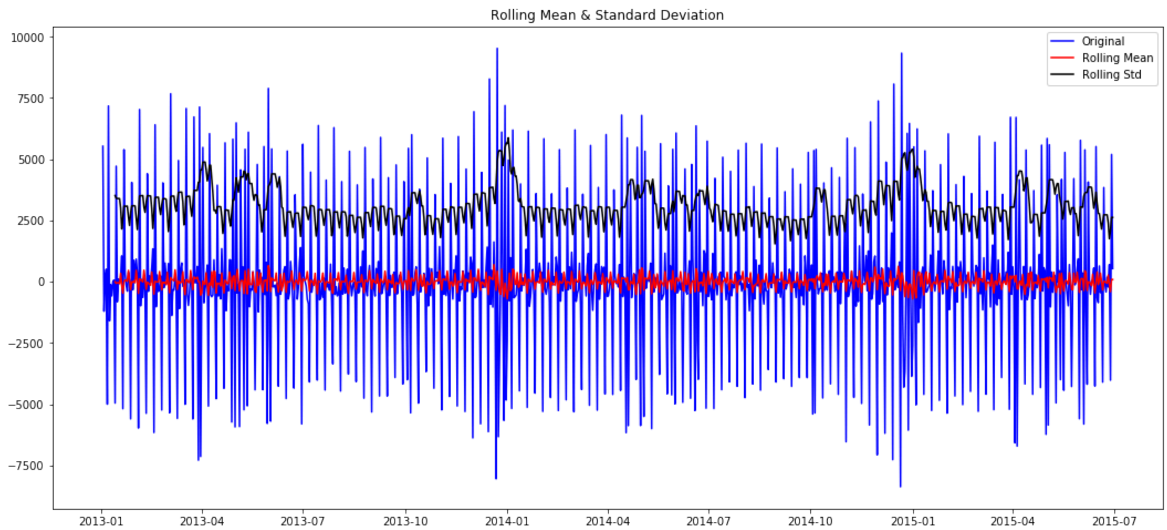
In [10]:
```
test_stationarity(store1['Sales'])
```



```
Results of Dickey-Fuller Test:
Test Statistic                  -4.236942
p-value                          0.000570
#Lags Used                      21.000000
Number of Observations Used    889.000000
Critical Value (1%)             -3.437727
Critical Value (5%)             -2.864797
Critical Value (10%)            -2.568504
dtype: float64
```

The smaller p-value, the more likely it's stationary. Here our p-value is 0.000415. It's actually good, but as we just visually found a little downward trend, we want to be more strict, i.e. if the p value further decreases, this series would be more likely to be stationary. To get a stationary data, there's many techiniques. We can use log, differencing etc..

In [11]:
```
first_diff = store1['Sales'] - store1['Sales'].shift(1)
first_diff = first_diff.dropna(inplace = False)
test_stationarity(first_diff)
```




```
Results of Dickey-Fuller Test:
Test Statistic                -1.134395e+01
p-value                        1.038132e-20
#Lags Used                     2.000000e+01
Number of Observations Used    8.890000e+02
Critical Value (1%)           -3.437727e+00
Critical Value (5%)           -2.864797e+00
Critical Value (10%)          -2.568504e+00
dtype: float64
```

After differencing, the p-value is extremely small. Thus this series is very likely to be stationary.

In [80]:
```
#AR model
ar_mod = ARIMA(store1.Sales, (9,1,0),freq='D')
res=ar_mod.fit(disp=False)
Y_pred = res.forecast(steps=31)[0]
print('MSE',np.sqrt(mean_squared_error(Y_pred,test_store1.Sales)))
print('MAE',mean_absolute_error(Y_pred,test_store1.Sales))
```

```
C:\Users\Public\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:19
1: FutureWarning: Creating a DatetimeIndex by passing range endpoints is deprecat
ed.  Use `pandas.date_range` instead.
  start=index[0], end=index[-1], freq=freq)
```

```
MSE 1133.8562710249823
MAE 895.9855008699199
```

In [84]:
```
#MA model
ma_mod = ARIMA(store1.Sales, (0,1,1),freq='D')
res=ma_mod.fit(disp=False)
Y_pred = res.forecast(steps=31)[0]
print('MSE',np.sqrt(mean_squared_error(Y_pred,test_store1.Sales)))
print('MAE',mean_absolute_error(Y_pred,test_store1.Sales))
```

```
MSE 1642.0868150322526
MAE 1182.9753111799089
```

In [90]:
```
#ARIMA model
arima_mod = ARIMA(store1.Sales, (9,1,9),freq='D')
res=arima_mod.fit(disp=False)
Y_pred = res.forecast(steps=31)[0]
print('MSE',np.sqrt(mean_squared_error(Y_pred,test_store1.Sales)))
print('MAE',mean_absolute_error(Y_pred,test_store1.Sales))
store1['pred']=Y_pred
```

```
MSE 633.5916329917548
MAE 465.4295796025833
```

```
C:\Users\Public\Anaconda3\lib\site-packages\statsmodels\base\model.py:488: Hessia
nInversionWarning: Inverting hessian failed, no bse or cov_params available
  'available', HessianInversionWarning)
C:\Users\Public\Anaconda3\lib\site-packages\statsmodels\base\model.py:508: Conver
genceWarning: Maximum Likelihood optimization failed to converge. Check mle_retva
ls
  "Check mle_retvals", ConvergenceWarning)
```

Project Task: Week 2

Implementing Neural Networks:

1. **Train a LSTM on the same set of features and compare the result with traditional time-series model.**
2. **Comment on the behavior of all the models you have built so far**
3. **Cluster stores using sales and customer visits as features. Find out how many clusters or groups are possible. Also visualize the results.**
4. **Is it possible to have separate prediction models for each cluster? Compare results with the previous models.**

LSTM for store1

In [8]:
```
train_store1 = store1.iloc[:, 1:2].values
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
train_store1 = sc.fit_transform(train_store1)

X_train = []
Y_train = []
for i in range(30, 911):
    X_train.append(train_store1[i-30:i, 0])
    Y_train.append(train_store1[i, 0])
X_train, Y_train = np.array(X_train), np.array(Y_train)
```

```
# Reshaping
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
```

```
C:\Users\Public\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595: Data
ConversionWarning: Data with input dtype int64 was converted to float64 by MinMax
Scaler.
  warnings.warn(msg, DataConversionWarning)
```

In [28]:
```
regressor = Sequential()
regressor.add(LSTM(units = 30, return_sequences = True, input_shape = (X_train.sh
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(LSTM(units = 70, return_sequences = True))
regressor.add(LSTM(units = 50))
regressor.add(Dense(units = 1))
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
regressor.fit(X_train, Y_train, epochs = 100, batch_size = 64, shuffle=False)
```

```
Epoch 1/100
881/881 [==============================] - 23s 26ms/step - loss: 0.0923
Epoch 2/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0485
Epoch 3/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0454
Epoch 4/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 5/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0450
Epoch 6/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0450
Epoch 7/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0450
Epoch 8/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0450
Epoch 9/100
881/881 [==============================] - 3s 3ms/step - loss: 0.0451
Epoch 10/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0451
Epoch 11/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0451
Epoch 12/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0450
Epoch 13/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0450
Epoch 14/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0450
Epoch 15/100
881/881 [==============================] - 4s 5ms/step - loss: 0.0450
Epoch 16/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0450
Epoch 17/100
881/881 [==============================] - 4s 4ms/step - loss: 0.0450
Epoch 18/100
881/881 [==============================] - 4s 4ms/step - loss: 0.0450
Epoch 19/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0450
Epoch 20/100
881/881 [==============================] - 4s 4ms/step - loss: 0.0450
Epoch 21/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 22/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 23/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 24/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 25/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 26/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 27/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 28/100
881/881 [==============================] - 3s 3ms/step - loss: 0.0449
Epoch 29/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 30/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 31/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 32/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
```

```
Epoch 33/100
881/881 [==============================] - 3s 3ms/step - loss: 0.0449
Epoch 34/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 35/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 36/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 37/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 38/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 39/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 40/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 41/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 42/100
881/881 [==============================] - 3s 3ms/step - loss: 0.0449
Epoch 43/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 44/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 45/100
881/881 [==============================] - 3s 3ms/step - loss: 0.0449
Epoch 46/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 47/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 48/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 49/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 50/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 51/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 52/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 53/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 54/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0449
Epoch 55/100
881/881 [==============================] - 3s 3ms/step - loss: 0.0448
Epoch 56/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0448
Epoch 57/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0448
Epoch 58/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0448
Epoch 59/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0448
Epoch 60/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0447
Epoch 61/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0447
Epoch 62/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0447
Epoch 63/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0447
Epoch 64/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0447
```

```
Epoch 65/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0447
Epoch 66/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0447
Epoch 67/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0447
Epoch 68/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0446
Epoch 69/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0446
Epoch 70/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0446
Epoch 71/100
881/881 [==============================] - 4s 4ms/step - loss: 0.0446
Epoch 72/100
881/881 [==============================] - 4s 4ms/step - loss: 0.0445
Epoch 73/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0445
Epoch 74/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0445
Epoch 75/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0446
Epoch 76/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0442
Epoch 77/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0440
Epoch 78/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0443
Epoch 79/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0446
Epoch 80/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0444
Epoch 81/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0440
Epoch 82/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0435
Epoch 83/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0434
Epoch 84/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0437
Epoch 85/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0434
Epoch 86/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0431
Epoch 87/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0430
Epoch 88/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0430
Epoch 89/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0429
Epoch 90/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0428
Epoch 91/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0427
Epoch 92/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0426
Epoch 93/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0427
Epoch 94/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0428
Epoch 95/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0431
Epoch 96/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0425
```

```
Epoch 97/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0421
Epoch 98/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0432
Epoch 99/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0439
Epoch 100/100
881/881 [==============================] - 3s 4ms/step - loss: 0.0434
```

Out[28]:

```
<keras.callbacks.History at 0xf1e92cb198>
```

In [29]:

```python
total_data = pd.concat((store1['Sales'], test_store1['Sales']), axis = 0)
inputs = total_data[len(total_data) - len(test_store1) - 30:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
X_test = []
for i in range(30, 61):
    X_test.append(inputs[i-30:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
pred = regressor.predict(X_test)
pred= sc.inverse_transform(pred)
print(np.sqrt(mean_squared_error(pred,test_store1.Sales)))
print(mean_absolute_error(pred,test_store1.Sales))
```

```
1544.3219893558846
1043.4500456779233
```

In [ ]:

```python
# Visualising the results
plt.plot(test_store1.Sales, color = 'red', label = 'Actual Sales')
plt.plot(pred, color = 'blue', label = 'Predicted Sales')
plt.title('Sales Prediction')
plt.xlabel('Time')
plt.ylabel('Sale')
plt.legend()
plt.show()
```

Applying ANN:

1. **Use ANN (Artificial Neural Network) to predict Store Sales.**
    - **Fine-tune number of layers,**
    - **Number of Neurons in each layers.**
    - **Experiment in batch-size.**
    - **Experiment with number of epochs. Carefully observe the loss and accuracy? What are the observations?**
    - **Play with different Learning Rate variants of Gradient Descent like Adam, SGD, RMS-prop.**
    - **Which activation performs best for this use case and why?**
    - **Check how it performed in the dataset, calculate RMSE.**
2. **Use Dropout for ANN and find the optimum number of clusters (clusters formed considering the features: sales and customer visits). Compare model performance with traditional ML based prediction models.**

3. Find the best setting of neural net that minimizes the loss and can predict the sales best. Use techniques like Grid search, cross-validation and Random search.

In [15]:
```
#Model1
X_train = train2.drop(['Sales','Date','Customers'],axis=1).values
X_val = test_val2.drop(['Sales','Date','Customers'],axis=1).values
Y_train = pd.DataFrame(train2['Sales'])
Y_val = test_val2['Sales']
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
Y_train = sc.fit_transform(Y_train)

model = Sequential()
model.add(Dense(100, activation='relu', input_dim = X_train.shape[1]))
#model.add(Dropout(0.1))
model.add(Dense(64, activation='relu'))
model.add(Dense(50, activation='relu'))
#model.add(Dropout(0.2))
model.add(Dense(1,activation='linear',kernel_initializer='normal') )
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, Y_train, epochs=10, batch_size=64,shuffle=False,verbose=0)
Y_pred = model.predict(X_val, batch_size=64,verbose=0)
Y_pred= sc.inverse_transform(Y_pred)
print('MSE',np.sqrt(mean_squared_error(Y_pred,Y_val)))
print('MAE',mean_absolute_error(Y_pred,Y_val))
# MSE 2515.353601819651
#MAE 1676.8835278851793
```

```
MSE 2563.1362612696907
MAE 1831.2433319952684
```

In [10]:
```
#model2
X_train = train1.drop(['Sales','Date','Customers'],axis=1).values
X_val = test_val1.drop(['Sales','Date','Customers'],axis=1).values
Y_train = pd.DataFrame(train1['Sales'])
Y_val = test_val1['Sales']
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
Y_train = sc.fit_transform(Y_train)

model = Sequential()
model.add(Dense(100, activation='relu', input_dim = X_train.shape[1]))
#model.add(Dropout(0.1))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1,activation='linear') )
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, Y_train, epochs=10, batch_size=64,shuffle=False,verbose=0)
Y_pred = model.predict(X_val, batch_size=64,verbose=0)
Y_pred= sc.inverse_transform(Y_pred)


print('MSE',np.sqrt(mean_squared_error(Y_pred,Y_val)))
print('MAE',mean_absolute_error(Y_pred,Y_val))
```

```
WARNING:tensorflow:From /opt/anaconda3/lib/python3.7/site-packages/tensorflow/pyt
hon/framework/op_def_library.py:263: colocate_with (from tensorflow.python.framew
ork.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /opt/anaconda3/lib/python3.7/site-packages/keras/backend/
tensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) w
ith keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_
prob`.
WARNING:tensorflow:From /opt/anaconda3/lib/python3.7/site-packages/tensorflow/pyt
hon/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is depre
cated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
MSE 1690.6897455191363
MAE 1170.5848143327298
```

In [ ]: