AI Capstone Project report by Vishal Yerme

Project title : Retail

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

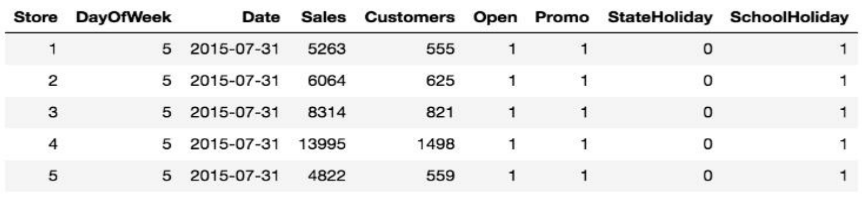
Description

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.

Dataset Snapshot:

Training Data Description: Historic sales at Store-Day level for about two years for a retail giant, for more than 1000 stores. Also, other sale influencers like, whether on a particular day the store was fully open or closed for renovation, holiday and special event details, are also provided.



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

Steps Performed:

Project Task: Week 1

1. Exploratory Data Analysis (EDA) and Linear Regression:
2. Transform the variables by using data manipulation techniques like, One-Hot Encoding
3. Perform an EDA (Exploratory Data Analysis) to see the impact of variables over Sales.
4. 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?
5. Train a single model for all stores, using storeId as a feature.
6. Train separate model for each store.
7. Which performs better and Why? [In the first case, parameters are shared and not very free but not in second case]

8. Try Ensemble of b) and c). What are the findings?

9. Use Regularized Regression. It should perform better in an unseen test set. Any insights?

Open-ended modeling to get possible predictions.

Linear Regression Results with STORE as feature :

MSE	***1428.9181706826785***
MAE	***1051.5557239043492***
train model score	***0.8365645595889428***
test model score	***0.8430035399815969***

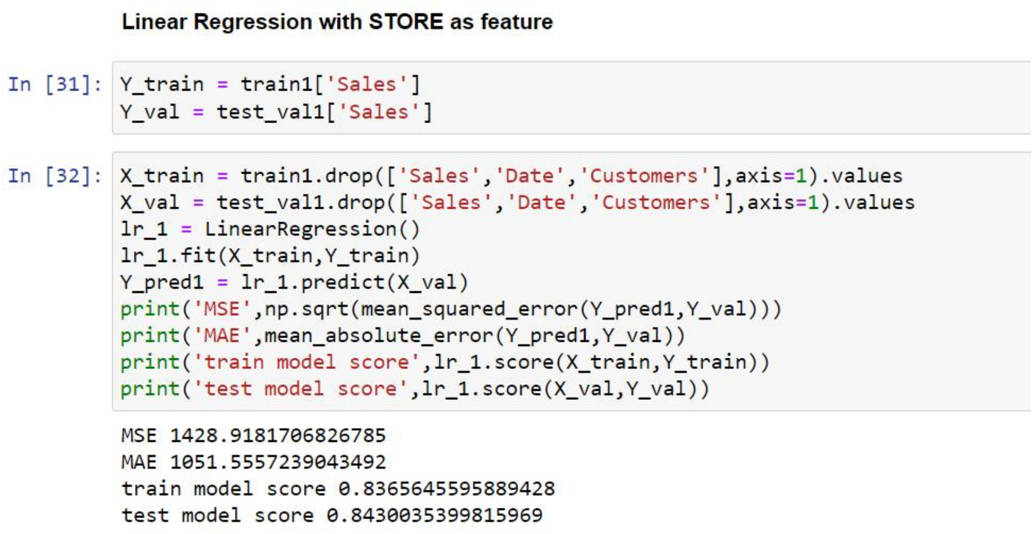
Linear Regression Results without STORE as feature :

MSE	***2520.0716734481693***
MAE	***1731.704737951633***
train model score	***0.5646267338030362***
test model score	***0.5116840301951011***

Linear Regression - Separate model for each STORE

MSE	***617791598632999.0***
MAE	***14098079978199.238***

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.

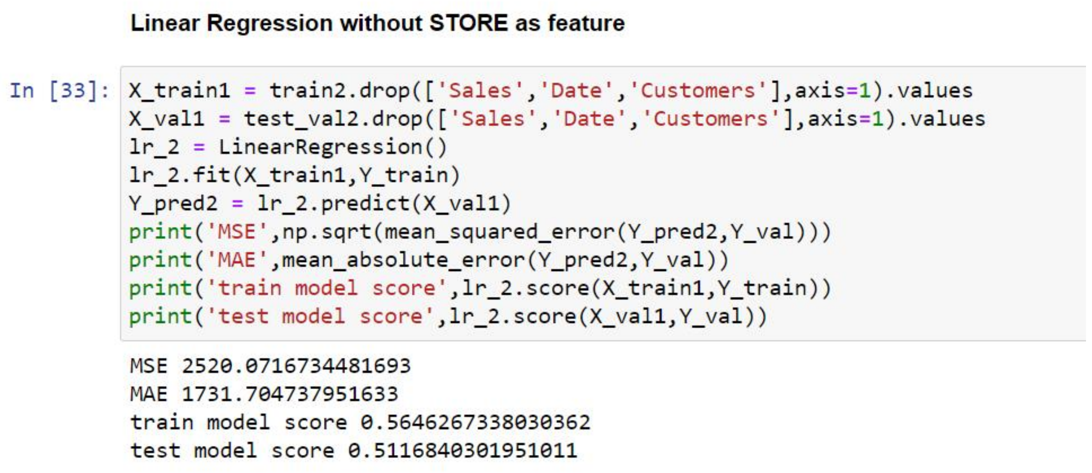


Linear Regression with STORE as feature

```
In [31]: Y_train = train1['Sales']
         Y_val = test_val1['Sales']
```

```
In [32]: 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 1428.9181706826785
MAE 1051.5557239043492
train model score 0.8365645595889428
test model score 0.8430035399815969
```


Linear Regression without STORE as feature

```
In [33]: 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 2520.0716734481693
MAE 1731.704737951633
train model score 0.5646267338030362
test model score 0.5116840301951011
```

Other Regression Techniques:

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?

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.

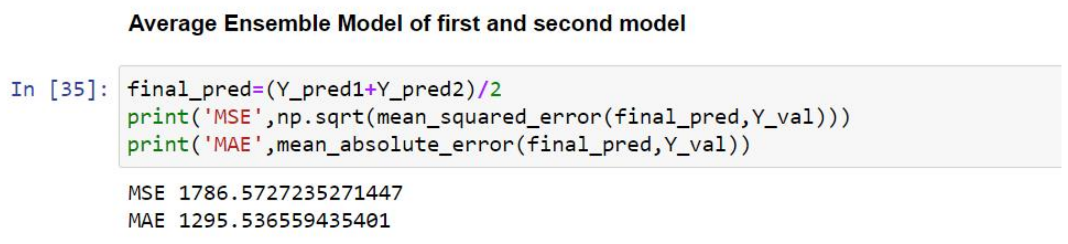
Train separate models for each store.

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.

Compare the performance of Linear Model and Non-Linear Model from the previous observations. Which performs better and why?

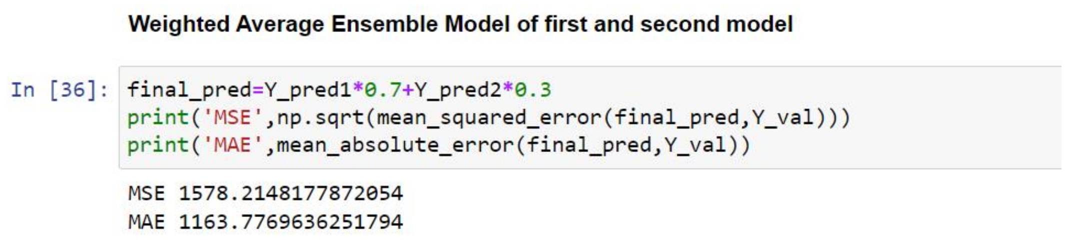
Train a Time-series model on the data taking time as the only feature. This will be a store-level training.

Identify yearly trends and seasonal months


Average Ensemble Model of first and second model

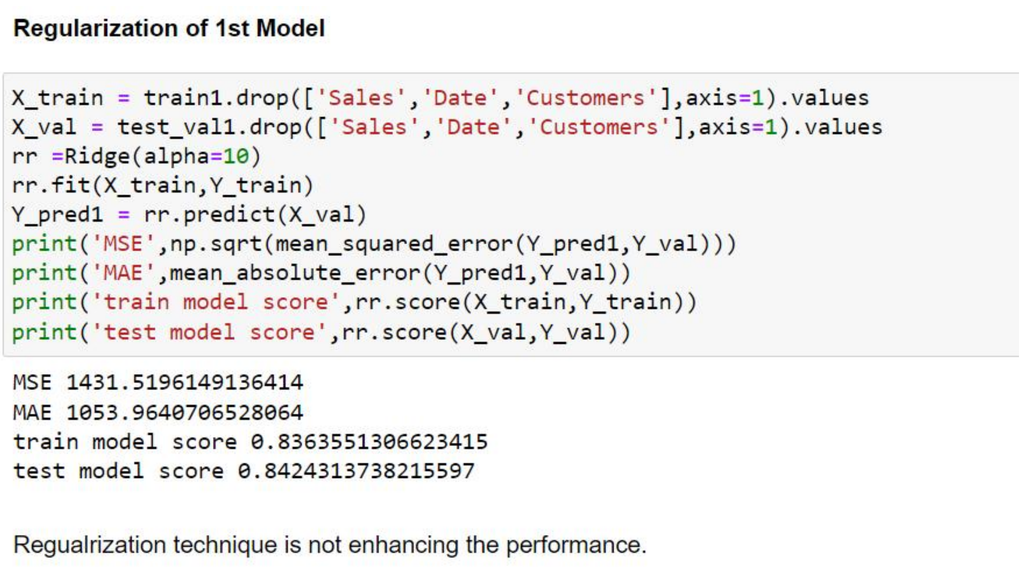
```
In [35]: final_pred=(Y_pred1+Y_pred2)/2
         print('MSE',np.sqrt(mean_squared_error(final_pred,Y_val)))
         print('MAE',mean_absolute_error(final_pred,Y_val))
```

```
MSE 1786.5727235271447
MAE 1295.536559435401
```


Weighted Average Ensemble Model of first and second model

```
In [36]: 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 1578.2148177872054
MAE 1163.7769636251794
```



Regularization of 1st Model

```
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:

10. Train a LSTM on the same set of features and compare the result with traditional time-series model.

11. Comment on the behavior of all the models you have built so far

12. Cluster stores using sales and customer visits as features. Find out how many clusters or groups are possible. Also visualize the results.

13. Is it possible to have separate prediction models for each cluster? Compare results with the previous models.

Applying ANN:

14. Use ANN (Artificial Neural Network) to predict Store Sales.

15. Fine-tune number of layers,

16. Number of Neurons in each layers.

17. Experiment in batch-size.

18. Experiment with number of epochs. Carefully observe the loss and accuracy? What are the observations?

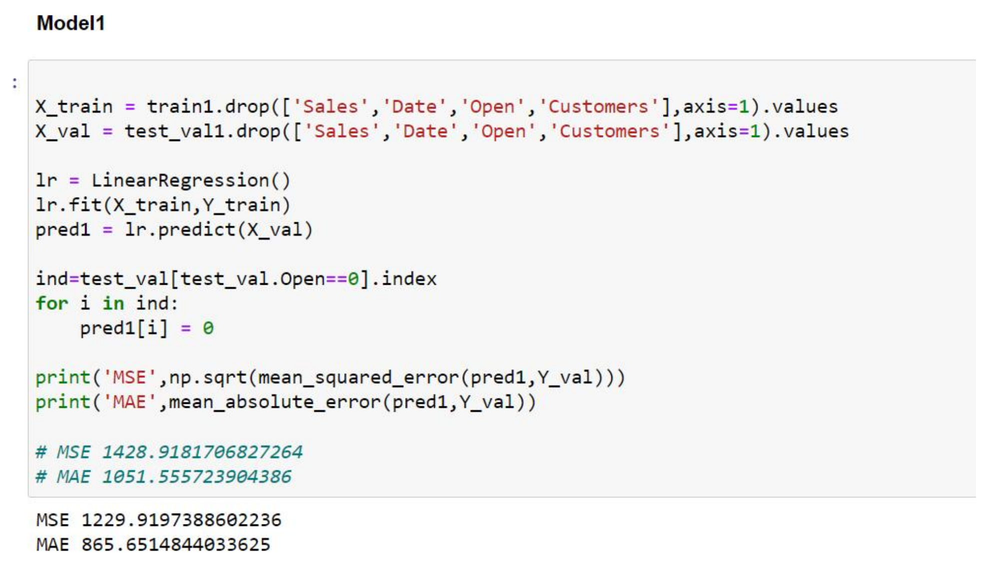
19. Play with different Learning Rate variants of Gradient Descent like Adam, SGD, RMS-prop.

20. Which activation performs best for this use case and why?

21. Check how it performed in the dataset, calculate RMSE.

22. 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.

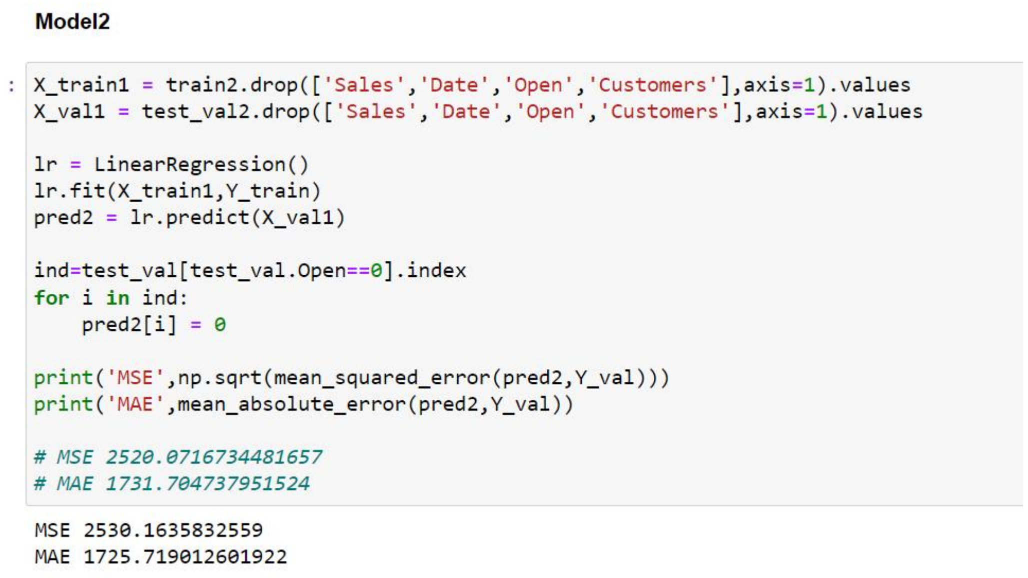
23. Find the best setting of neural net that minimizes the loss and can predict the sales best. Use techniques like Grid search, cross-validation and Random search.



Model1

```
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
```

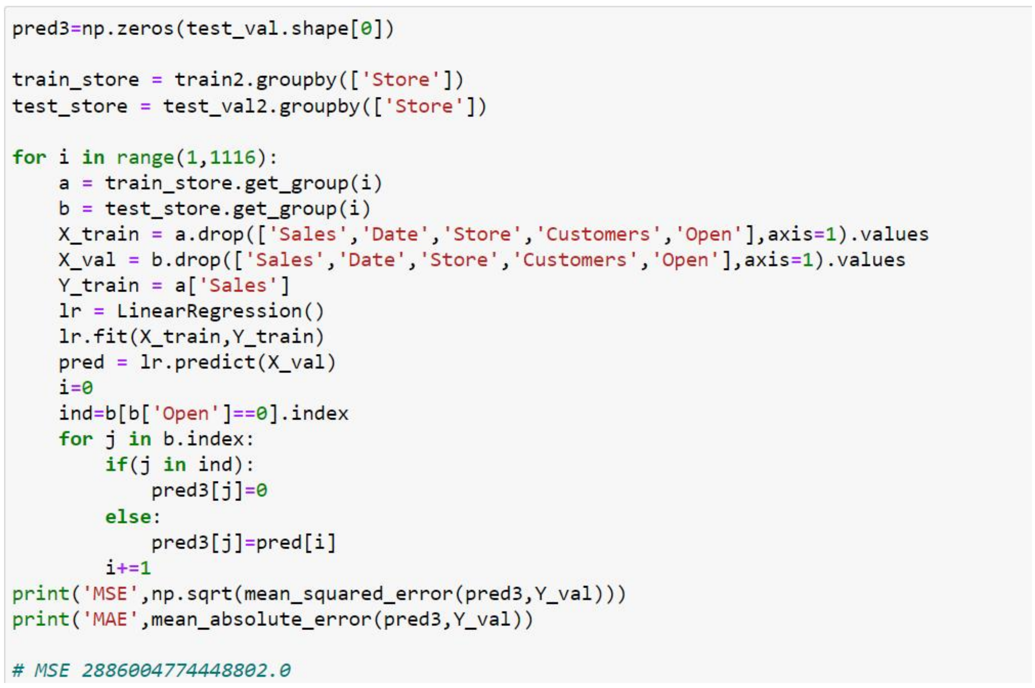


Model2

```
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
```

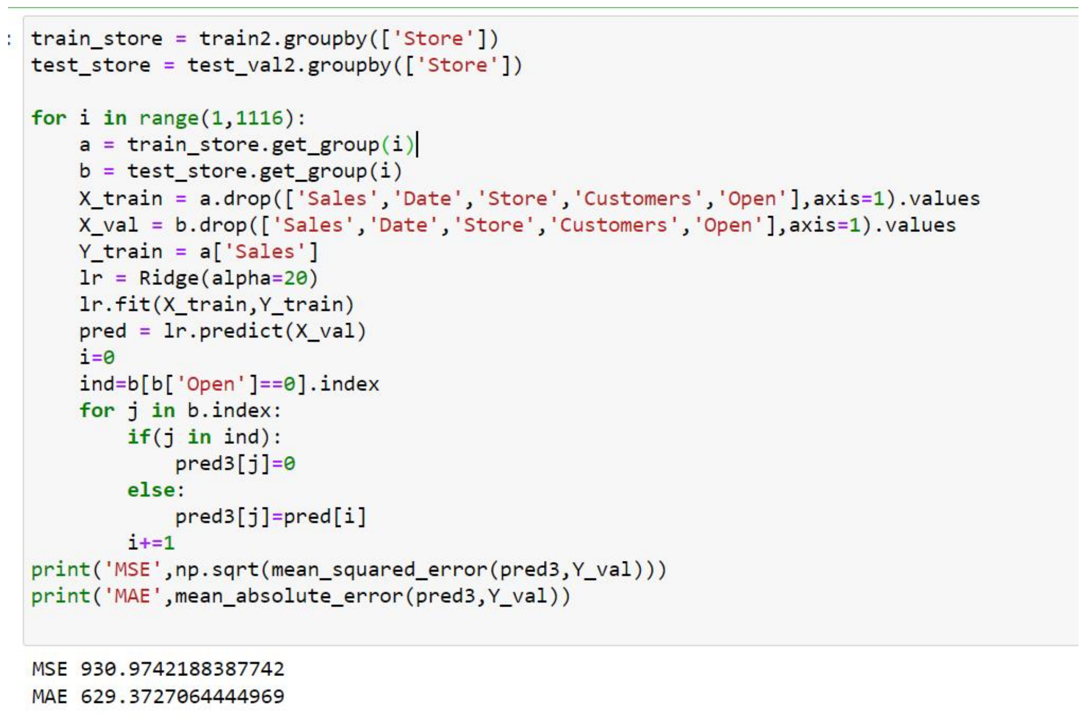
```
MSE 2530.1635832559
MAE 1725.719012601922
```

Model3



```
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
```

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.



```
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 regularization: MSE 930.9742188387742 MAE 629.3727064444969

- **Random Forest Regression with store as a feature:**

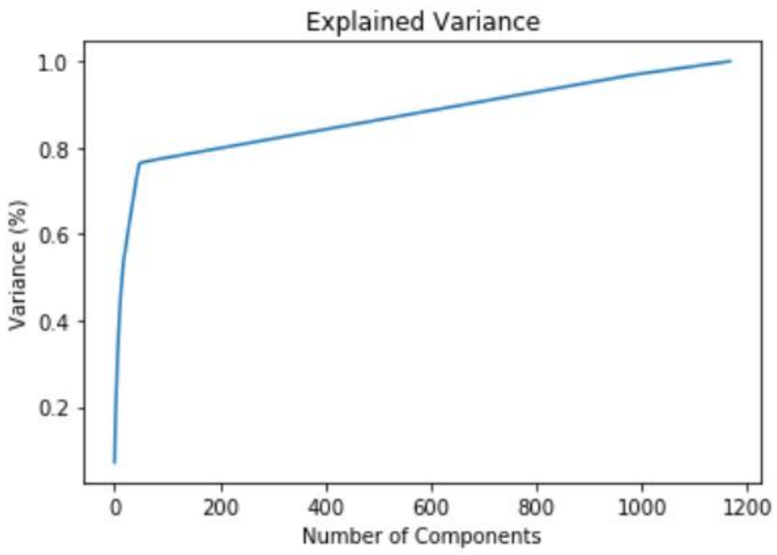
  MSE 2571.8525994831966
  MAE 1786.634280806513

- **Random Forest Regression without store as a feature:**

  MSE 2544.663201550362
  MAE 1728.0781382597204

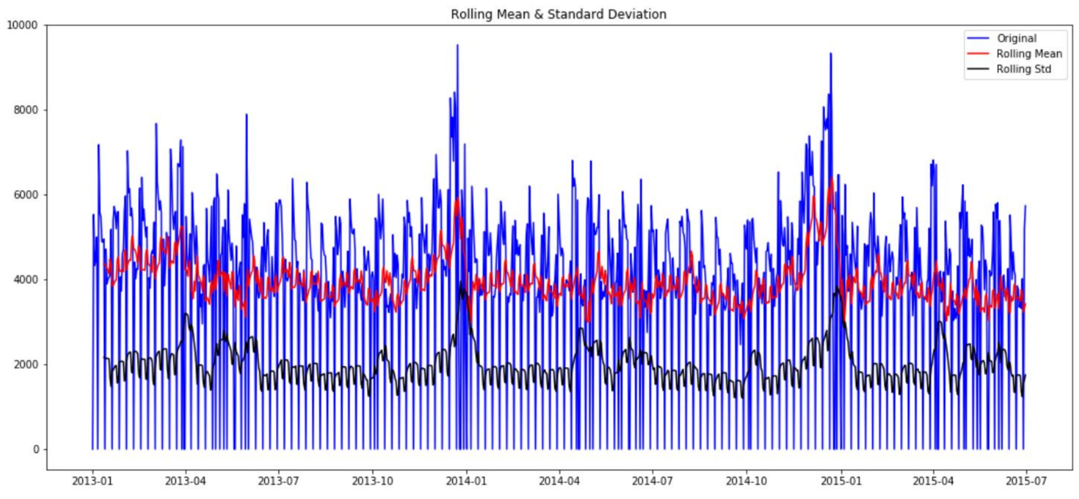
- **Separate model for each Store:**

  MSE 1077.7202738114058
  MAE 728.2337472832369



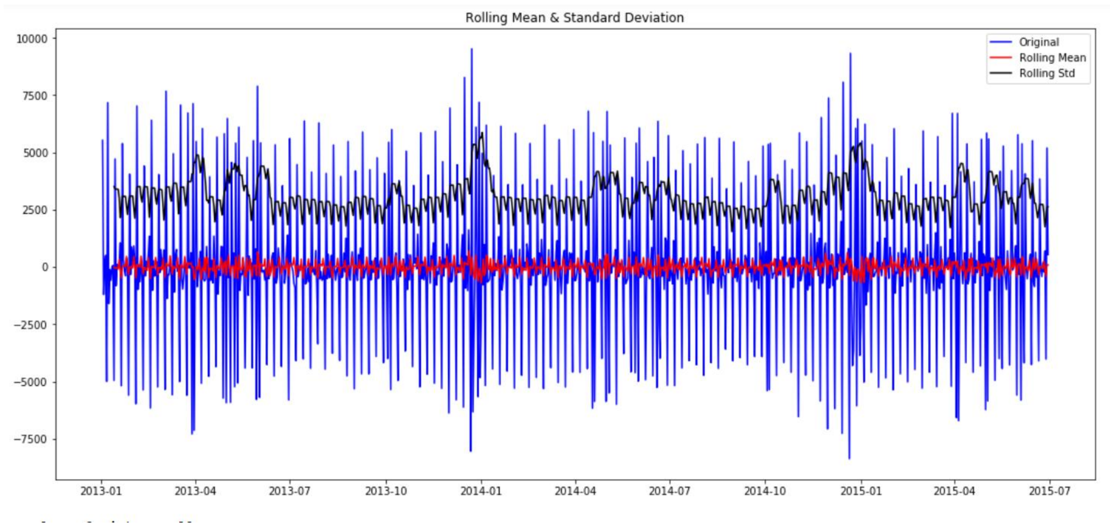
XGB: -with store MSE 1116.6123278288517
MAE 742.5063903587868

Without store MSE 1138.3182388080122
MAE 764.0298774444434



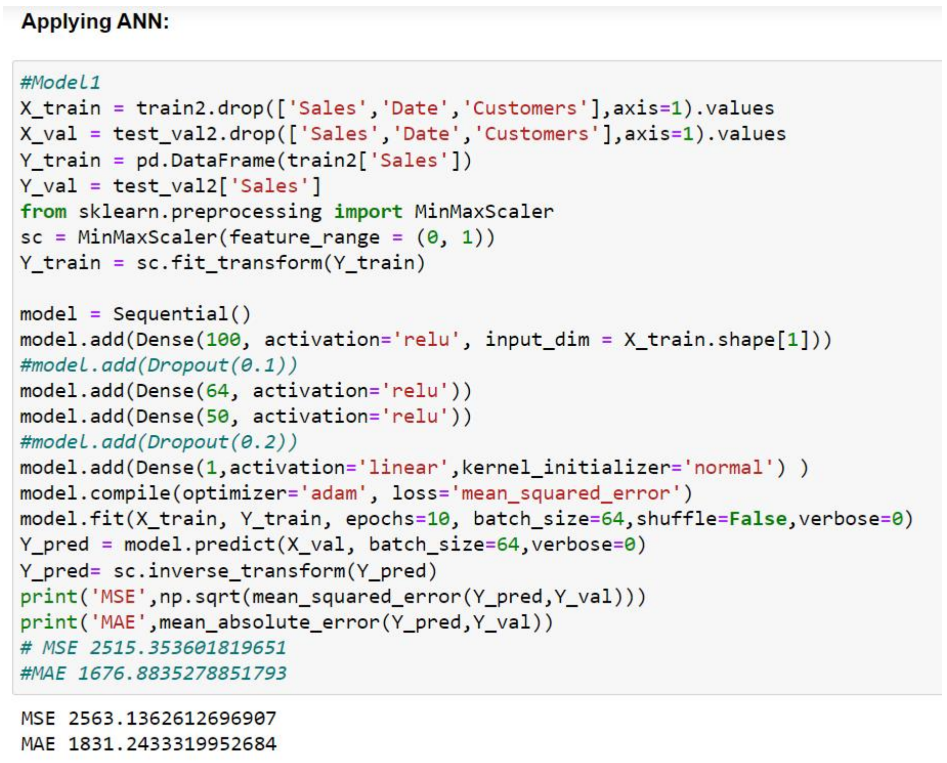
```
Results of Dickey-Fuller Test:
Test Statistic                  -4.236942
p-value                          0.000570
#Lags Used                      21.000000
Number of Observations Used    889.000000
Critical Value (1%)             -3.437727
Critical Value (5%)             -2.864797
Critical Value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

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 techniques. We can use log, differentiating etc..



```
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
```

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



Applying ANN:

```
#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
```