AI Capstone project - Retail

1. Linear Regression - Full Dataset :

```
0.8565126461130257
0.8551438975627743
```

```
[144] metrics(y_test_full,y_pred_full)
```

```
MAE:979.0617407406729
MSE:2116533.271937682
RMSE:1454.8310114709825
[979.0617407406729, 2116533.271937682, 1454.8310114709825]
```

2. Linear Regression - Separate Dataset :

```
Model for Store :  1115
(183,)
0.9848135932198266
0.9819253567905813
183
MAE:269.08183341826367
MSE:133606.9860285345
RMSE:365.52289398686713
328.65618910952594 231008.16004389813 448.58913107748157 0.9768410147317578 0.9749422983811393
```

Last line represents mean values of MAE,MSE, RMSE, train score and test score.

3. Linear Regression - Metamodel using Model stacking :

```
print("Meta-Model Metrics:")
print("Mean Absolute Error (MAE):", meta_mae)
print("Mean Squared Error (MSE):", meta_mse)
print("Root Mean Squared Error (RMSE):", meta_rmse)
```

```
Meta-Model Metrics:
Mean Absolute Error (MAE): 7.131235921585449e-12
Mean Squared Error (MSE): 9.740274064239342e-23
Root Mean Squared Error (RMSE): 9.869282681248593e-12
```

4. Lasso Regression - Full Dataset :

```
] #lasso regression
  from sklearn.linear_model import Lasso
  LR=Lasso(alpha=0.1)
  LR.fit(X_train_full,y_train_full)
  train_l1score =LR.score(X_train_full,y_train_full)
  test_l1score=LR.score(X_test_full,y_test_full)
  y_pred_LR=LR.predict(X_test_full)
  metrics(y_test_full,y_pred_LR)
  print(train_l1score,test_l1score)
```

```
MAE:979.1529201889432
MSE:2116584.5525155156
RMSE:1454.8486356028643
0.8551433537902865 0.8565091696194044
```

5. Lasso Regression - Separate Dataset :

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_coordinate_descent.py:631: ConvergenceW
  model = cd_fast.enet_coordinate_descent(
Model for Store :  1115
0.9924062208228094
0.9909625385059582
(183, 2)
MAE:134.4916282835989
MSE:66804.5270702032
RMSE:182.76286147656367
164.30235761903862 115522.37052595196 224.30500818421527 0.9884220736453596 0.987469938750994
```

Last line represents mean values of MAE,MSE, RMSE, train score and test score.

6. Linear Regression - Sales prediction on test data :

```
3      2015     7    31  0  0  0
4      2015     7    31  0  0  0
...     ...    ...   ... .. .. ..
34560  2015     7     1  0  0  0
34561  2015     7     1  0  0  0
34562  2015     7     1  0  0  0
34563  2015     7     1  0  0  0
34564  2015     7     1  0  0  0

[34565 rows x 13 columns]

Predictions on Test Data:
[ 7995.88686783  8649.09583909 10478.35125554 ...  9340.30098287
 37509.26329099  6325.21492438]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but LinearRegression was fitted without feature names
  warnings.warn(
```

7. Linear Regression - Sales prediction on trained data for sales =0 is deleted :

```
lin_reg_upd=LinearRegression()
lin_reg_upd.fit(X_train_full,y_train_full)
y_pred_upd=lin_reg_upd.predict(X_test_full)
print(lin_reg_upd.score(X_test_full,y_test_full))
print(lin_reg_upd.score(X_train_full,y_train_full))
```

```
0.7320004965398414
0.7339966120622974
```

```
metrics(y_pred_upd,y_test_full)
```

```
MAE:1150.1274937937037
MSE:2590415.8521776185
RMSE:1609.476887742604
[1150.1274937937037, 2590415.8521776185, 1609.476887742604]
```

8. XGBoost - Sales prediction on full train data :

```
y_pred_xgb=xgb_reg.predict(X_test_full)
xgb=metrics(y_test_full,y_pred_xgb)
print(xgb)
print(xgb_reg.score(X_train_full,y_train_full))
print(xgb_reg.score(X_test_full,y_test_full))
```

```
MAE:1306.6983958028263
MSE:3136808.0399010032
RMSE:1771.1036220111469
[1306.6983958028263, 3136808.0399010032, 1771.1036220111469]
0.675275524569257
0.6754717986934016
```

9. XGBoost - Sales prediction on separate datasets :

```
Model for Store :  1115
0.4991983694944764
0.4710770261115416
MAE:158.02484373383174
MSE:87301.22819012517
RMSE:208.92735483627467
159.52063039395338 111150.41373913678 220.98081382253 0.4990469295749233 0.46741600830235064
```

Last line represents mean values of MAE,MSE, RMSE, train score and test score.

10. XGBoost - Sales prediction after hyperparameter tuning using RandomsearchCV :

```
y_pred_xgb_hp=xgb_reg_hp.predict(X_test_full)
xgb_hp=metrics(y_test_full,y_pred_xgb_hp)
print(xgb_hp)
print(xgb_reg_hp.score(X_train_full,y_train_full))
print(xgb_reg_hp.score(X_test_full,y_test_full))
```

```
MAE:1421.5741630817993
MSE:3790979.752508196
RMSE:1947.0438496624045
[1421.5741630817993, 3790979.752508196, 1947.0438496624045]
0.6077542652073118
0.6077924359343182
```

11. PCA - Sales prediction - Linear regression on Full dataset:

```
lin_reg_pca= LinearRegression()
lin_reg_pca.fit(X_train_pca,y_train_pca)
y_pred_pca=lin_reg_pca.predict(X_test_pca)
metrics(y_pred_pca,y_test_pca)
print(lin_reg_pca.score(X_train_pca,y_train_pca))
print(lin_reg_pca.score(X_test_pca,y_test_pca))
```

```
MAE:607.0299073507095
MSE:1441110.0644264568
RMSE:848.8551904497541
0.9010212376336282
0.9023019989592326
```

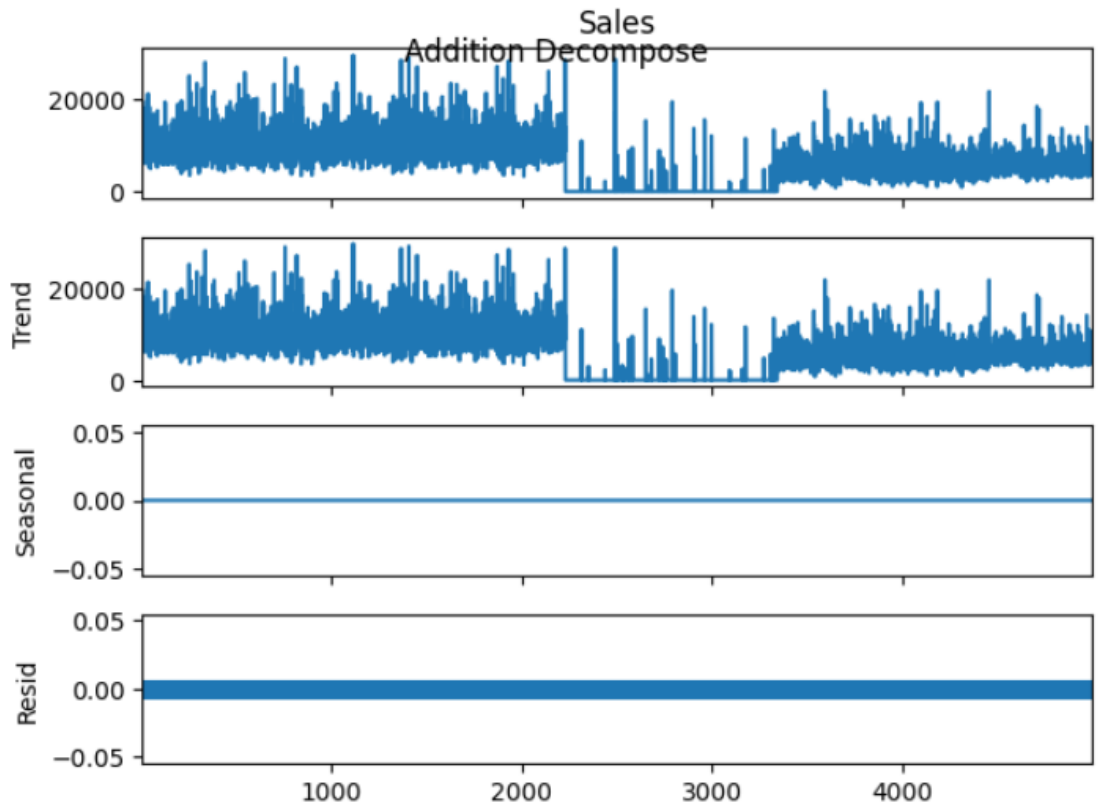
Based on the results, model performs well on linear regression as the independent variables are more or less linearly correlated with dependent variable. With PCA ,and regularisation ,results are even better on full dataset .
On separate datasets, results seem to be fine with mean of all the metrics for each store using linear regression and regularisation techniques.

Time Series :

1. **Multiplicative decomposition :**

```
#mul_result.plot().suptitle('\n Multiplication Decompose',fontsize=12)
add_result.plot().suptitle('\n Addition Decompose',fontsize=12)
```

```
Text(0.5, 0.98, '\n Addition Decompose')
<Figure size 5000x5000 with 0 Axes>
```



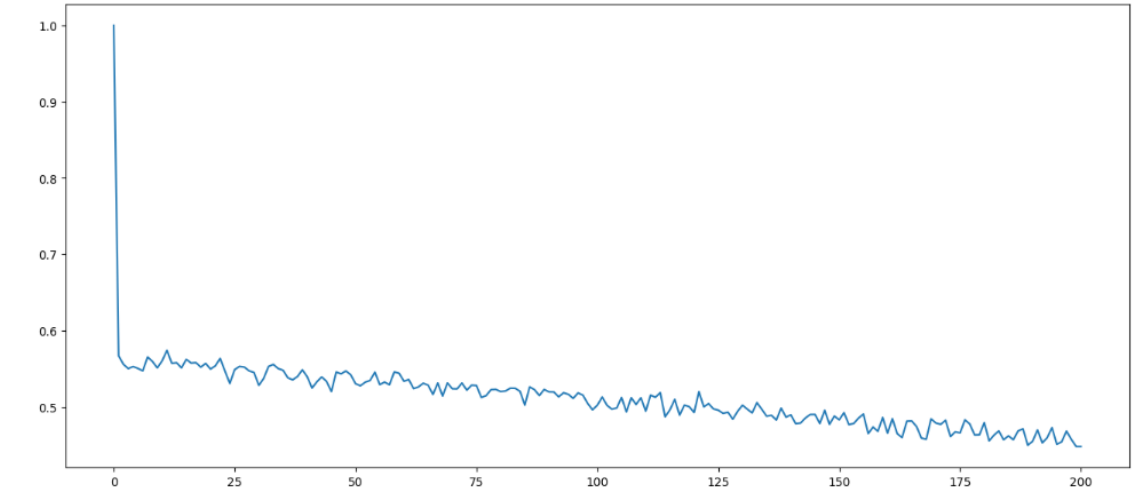
2. **ADFuller test :** shows data is non stationary.

```
#stationarity check
adfuller_result=adfuller(data1[1:5000].values,autolag='AIC')
print(f'ADfuller statistic : {adfuller_result[0]}')
print(f'ADfuller p value : {adfuller_result[1]}')
print(adfuller_result)
```

```
ADfuller statistic : -2.0588586050716655
ADfuller p value : 0.2613761411515416
(-2.0588586050716655, 0.2613761411515416, 32, 4966, {'1%': -3.4316674956516784, '5%': -2.8621221897344484, '10%': -2.567079900452355}, 93733.81799826455)
```

3. **ACF curve :**

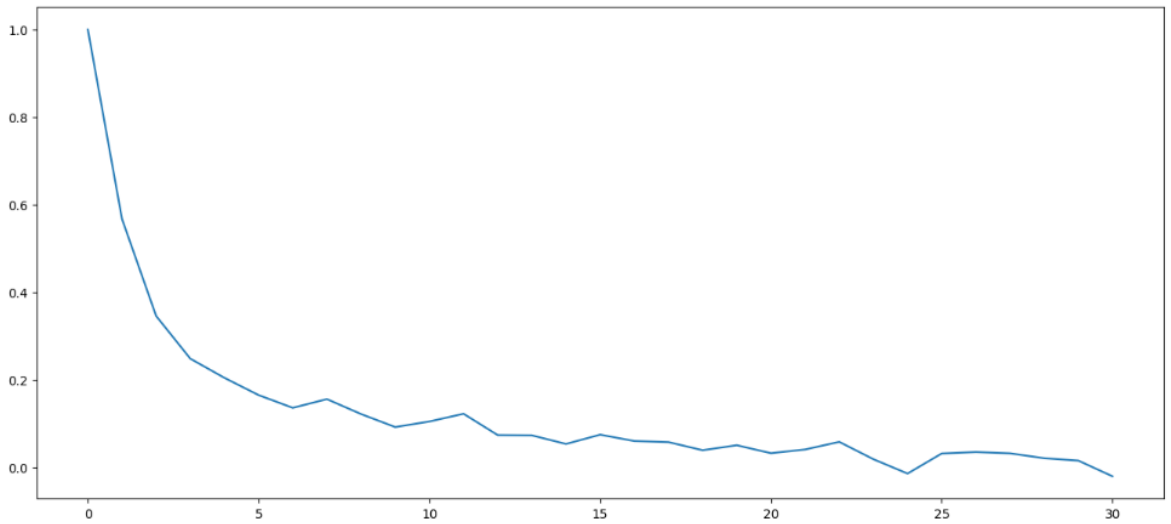
```
[<matplotlib.lines.Line2D at 0x7fa1ae4ff0a0>]
```



4. PACF :

```
#plot pacf
plt.figure(figsize=(16,7))
plt.plot(pacf_lag)
```

```
[<matplotlib.lines.Line2D at 0x7fa1ae42d6f0>]
```

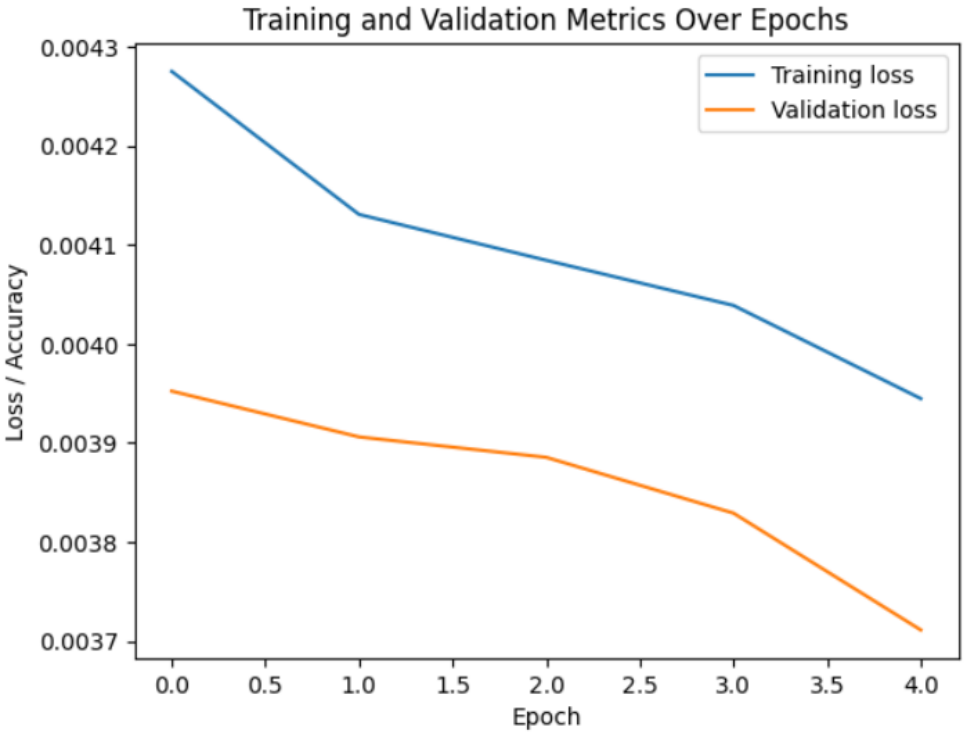


5. ARIMA model :

```
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to converge.
  warnings.warn("Maximum Likelihood optimization failed to "
1000     9886.791657
1001     9664.481748
1002    10635.645284
1003    10558.152291
1004     9986.111133
1005    10327.552270
1006    10565.998227
1007     9585.080773
1008    10592.397894
1009    10335.024532
1010    10001.736187
1011    10680.431626
1012    10309.830711
1013    10056.010776
1014    10386.256976
1015    10205.308541
1016     9976.044256
1017    10500.538586
1018    10323.753804
1019    10183.615656
1020    10565.446707
Name: predicted_mean, dtype: float64
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/representation.py:374: FutureWarning: Unknown keyword arguments: dict_keys(
  warnings.warn(msg, FutureWarning)
```

Based on first 1000 sales data values , next 20 values are predicted using ARIMA model

6. Time series using LSTM :




```
#Validation loss
#y_pred=scaler.inverse_transform(train_predictions)
validation_loss = model.evaluate(X_test, y_test)
```

```
6142/6142 [==============================] - 45s 7ms/step - loss: 0.0037
```

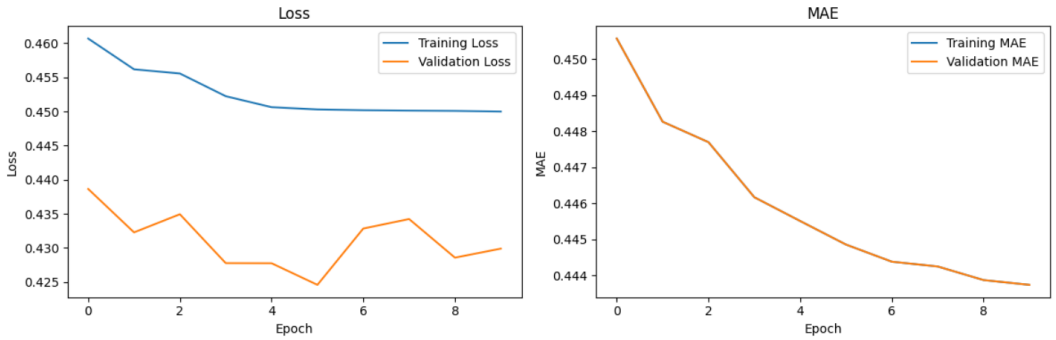
```
print("Test Loss:", validation_loss)
```

```
Test Loss: 0.003721779678016901
```

```
#generalization
test_predictions = model.predict(testX)
y_pred_test=scaler_y.inverse_transform(test_predictions)
y_pred_test
```

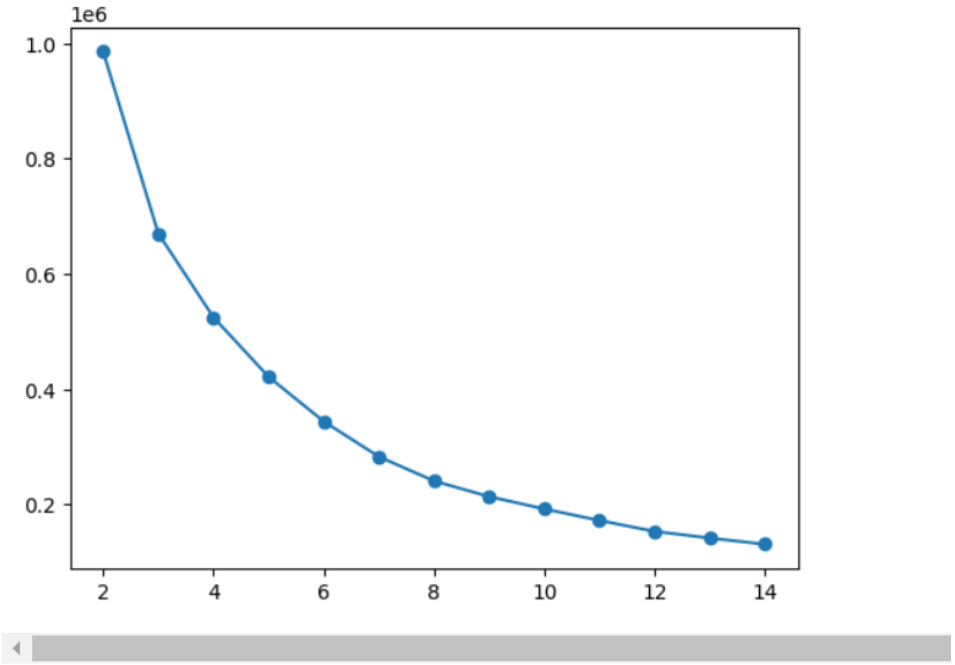
```
1080/1080 [==============================] - 8s 8ms/step
array([[ 451.40778],
       [ 966.6415 ],
       [ 969.2233 ],
       ...,
       [5589.321  ],
       [5392.8174 ],
       [3169.016  ]], dtype=float32)
```

7. Using ANN :

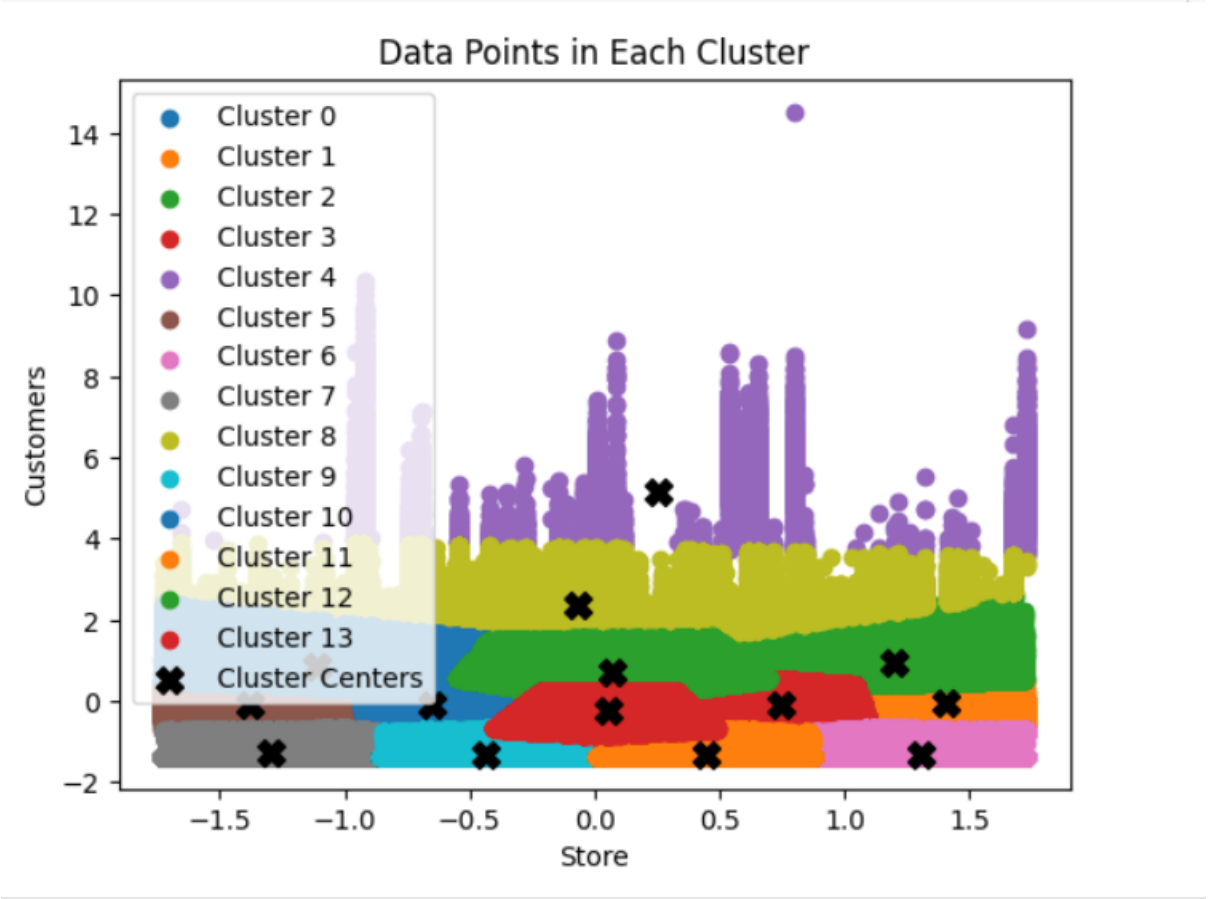


8. Using K means clustering to group Sales data :

```
[<matplotlib.lines.Line2D at 0x7e88662a61d0>]
```



Finding optimal cluster number = 14


Data Points in Each Cluster
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 7
Cluster 8
Cluster 9
Cluster 10
Cluster 11
Cluster 12
Cluster 13
Cluster Centers
Customers
Store
14
12
10
8
6
4
2
0
−2
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5