AI Capstone Project
Retail

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.

Evaluation:

Root mean squared error (RMSE) is the square root of the mean of the square of all of the error. The use of RMSE is very common, and it is considered an excellent general-purpose error metric for numerical predictions.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

RSME is usually preferred as it can deal with zero and extreme (outliers) observed values, while on the other hand, MAPE is challenged when observed values comprise zeros and extreme values. We therefore tend to get a lower MAPE vs RSME

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.

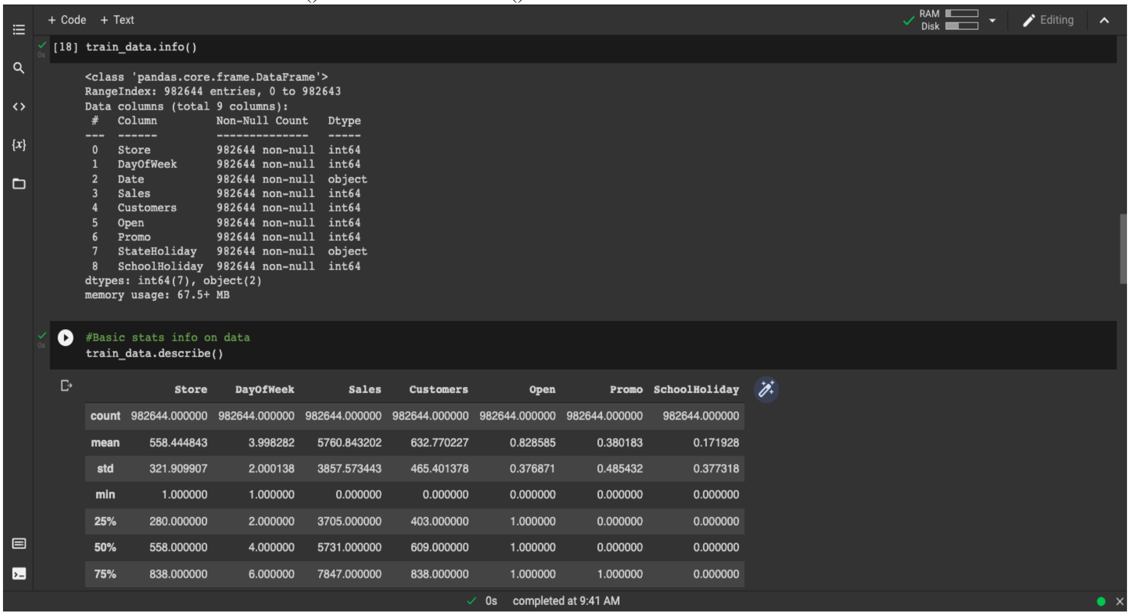
```
[ ] original_data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/AI Capstone/retail_datasets/train_data.csv')
    test_data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/AI Capstone/retail_datasets/test_data.csv')

    /usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2718: DtypeWarning: Columns (7) have mixed types.Specify dtype option on import or se
      interactivity=interactivity, compiler=compiler, result=result)
```

```
[ ] #Sample Data
    original_data.head()
```

	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

Exploratory Data Analysis

- Basic EDA – df.info() and df.describe()



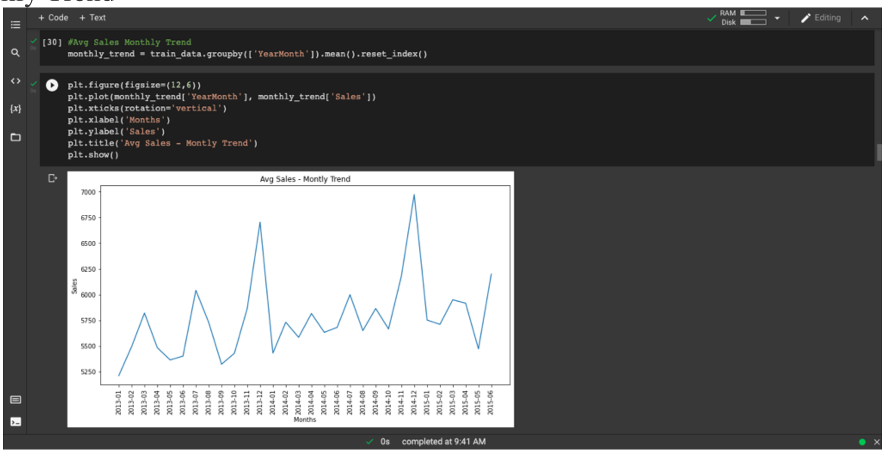
```
[18] train_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 982644 entries, 0 to 982643
Data columns (total 9 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Store          982644 non-null  int64
 1   DayOfWeek      982644 non-null  int64
 2   Date           982644 non-null  object
 3   Sales          982644 non-null  int64
 4   Customers      982644 non-null  int64
 5   Open           982644 non-null  int64
 6   Promo          982644 non-null  int64
 7   StateHoliday   982644 non-null  object
 8   SchoolHoliday  982644 non-null  int64
dtypes: int64(7), object(2)
memory usage: 67.5+ MB
```

```
#Basic stats info on data
train_data.describe()
```

	Store	DayOfWeek	Sales	Customers	Open	Promo	SchoolHoliday
count	982644.000000	982644.000000	982644.000000	982644.000000	982644.000000	982644.000000	982644.000000
mean	558.444843	3.998282	5760.843202	632.770227	0.828585	0.380183	0.171928
std	321.909907	2.000138	3857.573443	465.401378	0.376871	0.485432	0.377318
min	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	280.000000	2.000000	3705.000000	403.000000	1.000000	0.000000	0.000000
50%	558.000000	4.000000	5731.000000	609.000000	1.000000	0.000000	0.000000
75%	838.000000	6.000000	7847.000000	838.000000	1.000000	1.000000	0.000000

0s completed at 9:41 AM

- Monthly Trend

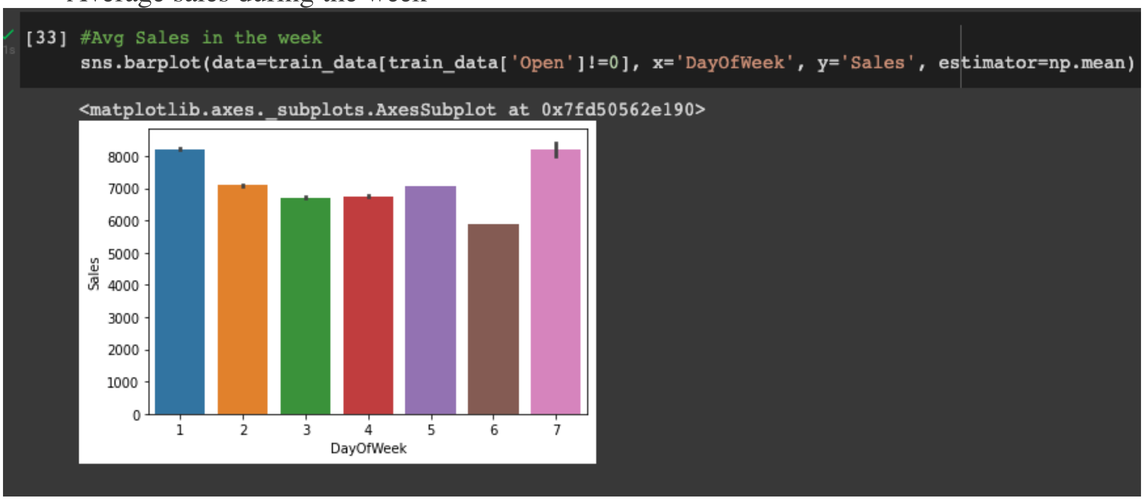


```
[30] #Avg Sales Monthly Trend
monthly_trend = train_data.groupby(['YearMonth']).mean().reset_index()
```

```
plt.figure(figsize=(12,6))
plt.plot(monthly_trend['YearMonth'], monthly_trend['Sales'])
plt.xticks(rotation='vertical')
plt.xlabel('Months')
plt.ylabel('Sales')
plt.title('Avg Sales - Montly Trend')
plt.show()
```

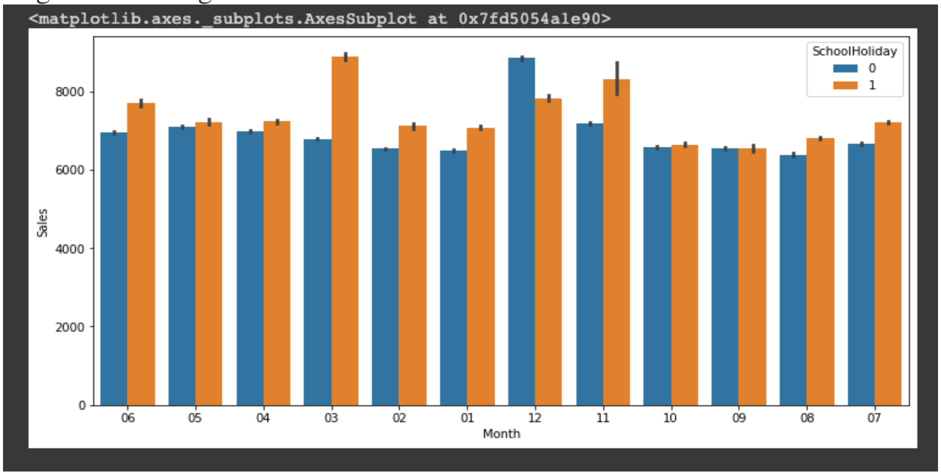

0s completed at 9:41 AM

- Average sales during the week

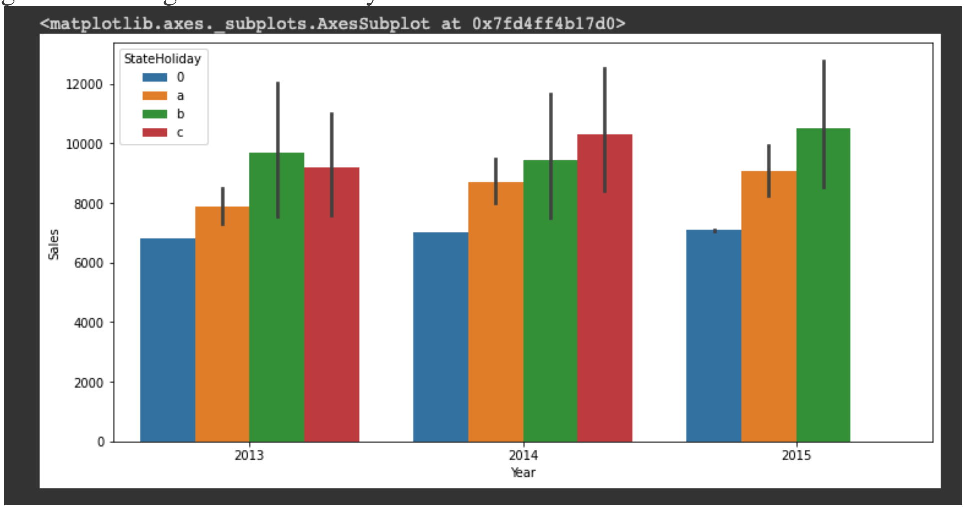


```
[33] #Avg Sales in the week
sns.barplot(data=train_data[train_data['Open']!=0], x='DayOfWeek', y='Sales', estimator=np.mean)

<matplotlib.axes._subplots.AxesSubplot at 0x7fd50562e190>
```

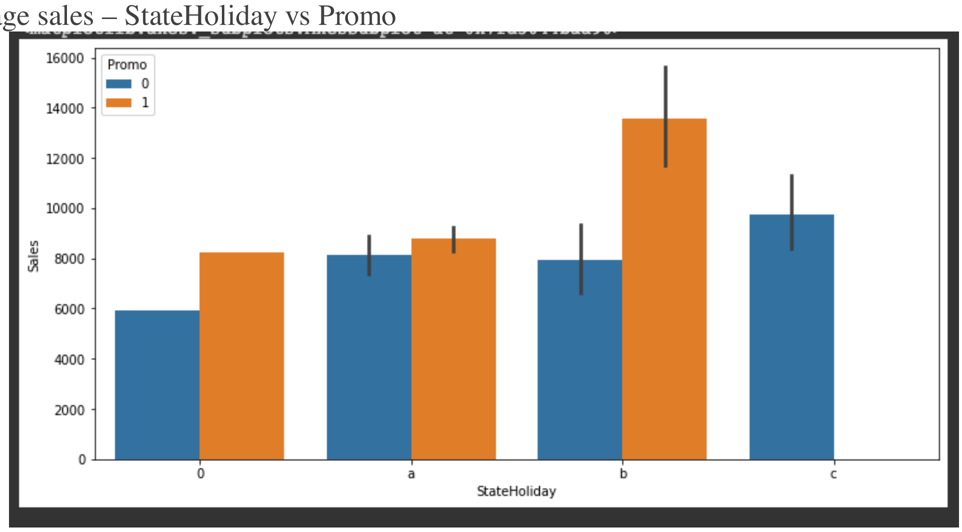
- Average sales during the months



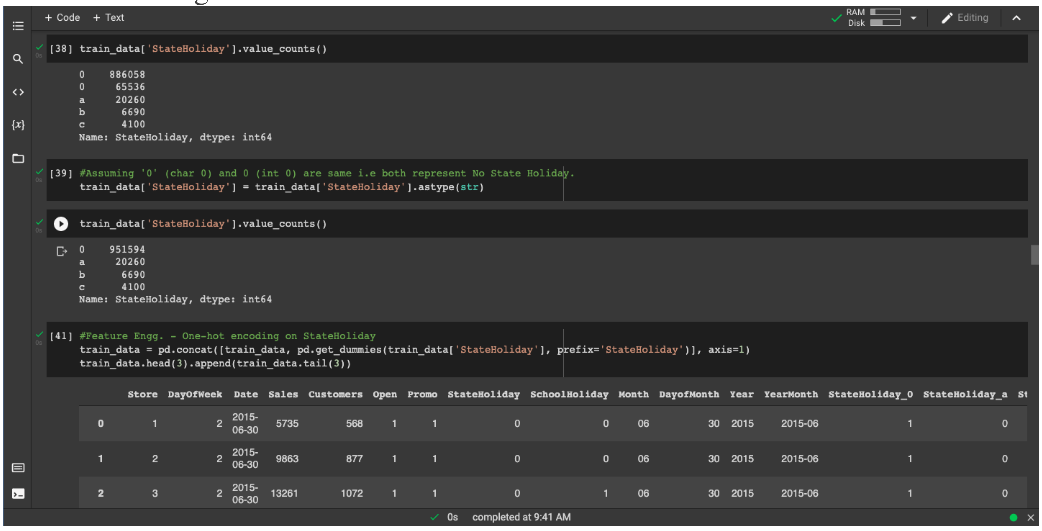
- Average sales during the sales holiday



- Average sales – StateHoliday vs Promo



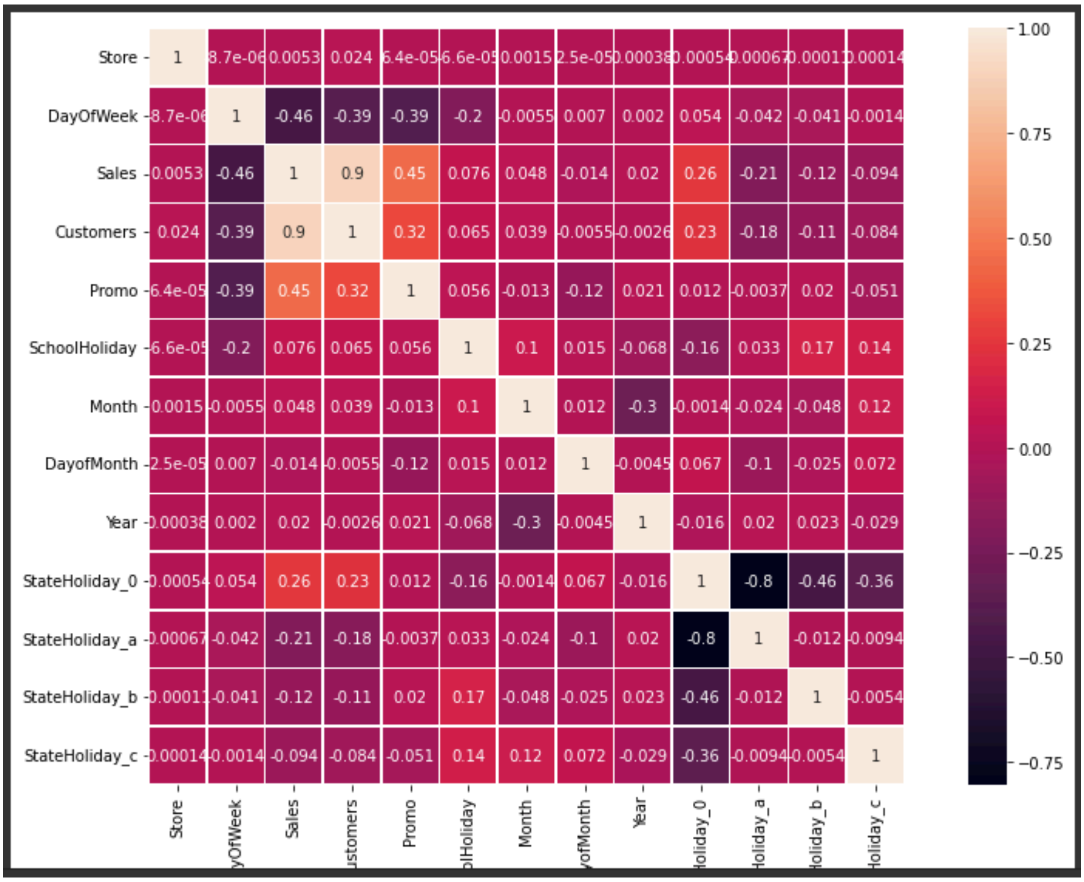
- One-Hot encoding



```
[38] train_data['StateHoliday'].value_counts()

0    886058
0     65536
a     20260
b      6690
c      4100
Name: StateHoliday, dtype: int64

[39] #Assuming '0' (char 0) and 0 (int 0) are same i.e both represent No State Holiday.
     train_data['StateHoliday'] = train_data['StateHoliday'].astype(str)

     train_data['StateHoliday'].value_counts()

0    951594
a     20260
b      6690
c      4100
Name: StateHoliday, dtype: int64

[41] #Feature Engg. - One-hot encoding on StateHoliday
     train_data = pd.concat([train_data, pd.get_dummies(train_data['StateHoliday'], prefix='StateHoliday')], axis=1)
     train_data.head(3).append(train_data.tail(3))
```

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	Month	DayofMonth	Year	YearMonth	StateHoliday_0	StateHoliday_a
0	1	2	2015-06-30	5735	568	1	1	0	0	06	30	2015	2015-06	1	0
1	2	2	2015-06-30	9863	877	1	1	0	0	06	30	2015	2015-06	1	0
2	3	2	2015-06-30	13261	1072	1	1	0	1	06	30	2015	2015-06	1	0

- Correlation



- Features used for Training

Index(['Store', 'DayOfWeek', 'Sales', 'Customers', 'Open', 'Promo', 'SchoolHoliday', 'Month', 'DayofMonth', 'Year','StateHoliday_0', 'StateHoliday_a', 'StateHoliday_b', 'StateHoliday_c'],
dtype='object')

Model Building/Training

Data is split into train test with 20% as testing data.

1. LinearRegression

- single model for all stores, using storeId as a feature.

```
[63] lr_single = LinearRegression()

     lr_single.fit(x_train, y_train)

     LinearRegression()

[64] #prediction using LinearRegression model
     y_pred = lr_single.predict(x_test)
```

Model Evaluation
LR - MAE : 982.6866249739529
LR - RMSE : 1468.2049179921212

Models results after considering sales of only open stores, and log standardization of target variable sales
LR - RMSE : 0.25095160243016823

- separate model for each store.

```
lr_sep = LinearRegression()
lr_sep.fit(xy_train.drop('Sales', axis=1)[xy_train['Store'] == i],
xy_train[xy_train['Store'] == i]['Sales'])
```

Model Evaluation
Avg. of RMSE of each store models : 452.20473461235525
RMSE of predication made by all the models : 486.01163821783393

2. Regularized Regression

```
lasso_single = Lasso(alpha=0.01)
lasso_single.fit(x_train, y_train)
```

Model Evaluation
Lasso RMSE : 1468.204792771783

From here on models are trained considering sales of only open stores, and log standardization of target variable sales

3. Tree-based Regressor: XGRegressor

- Basic

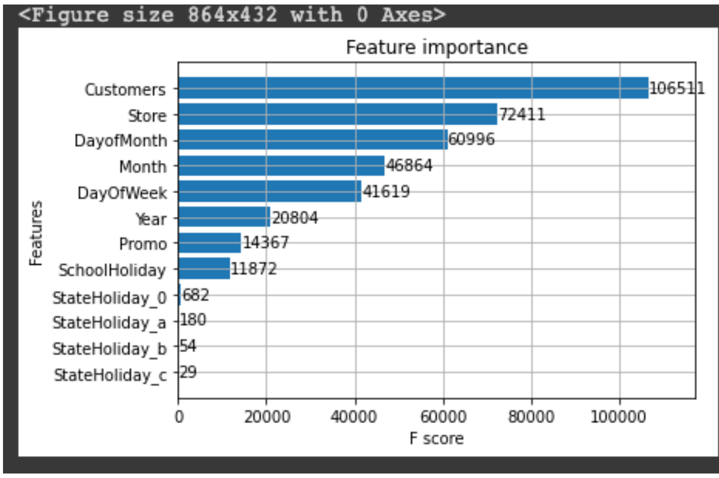
```
xgbReg = XGBRegressor(n_estimators=500,
                      max_depth=10,
                      subsample=0.8,
                      colsample_bytree=0.85,
                      learning_rate=0.1,
                      seed=99,
                      tree_method='hist'
                      )
```

Model result

```
RMSE : 0.1307980346904776
```

Feature Importance

```
<Figure size 864x432 with 0 Axes>
```



- Applying **GridSearchCV** and **KFold** for Tree's Hyperparameter Tuning.

```
xgbRegGridCV = XGBRegressor(silent=True,
                            seed=99, tree_method='hist')
#parameters of XGBRegressor for GridSearchCV
param_dict = {
  'n_estimators': [300, 600],
  'eta':  [0.01,0.03],
  'max_depth': [6,12],
  'subsample': [0.3,0.7],
  'colsample_bytree': [0.5,0.9],
  'min_child_weight':[8,12]
}
kf = KFold(n_splits=5, shuffle=False)
grid_search = GridSearchCV(xgbRegGridCV, param_dict, cv=kf,
scoring='neg_root_mean_squared_error', verbose=0)
```

After fitting data to GridSearchCV which took nearlt 9 hours to fit, the best parameters fo und where

```
{'colsample_bytree': 0.9,
 'eta': 0.01,
 'max_depth': 12,
 'min_child_weight': 12,
 'n_estimators': 300,
 'subsample': 0.7}
```

And the RMSE of best estimator or model with above parameters was

```
RMSE : 0.130982466420859564
```

For later XGRegressor model training above parameters were used.

- Model Training for each store

```
xgbReg_sep.fit(xy_train.drop('Sales',axis=1)[xy_train['Store']= =i],
xy_train[xy_train['Store'] == i]['Sales'])
```

- **PCA + XGRegressor**

```
pca = PCA(n_components=5, random_state=99)
dr_x_train = pca.fit_transform(x_train)
dr_x_test = pca.transform(x_test)

xgbReg_pca.fit(dr_x_train, y_train)
```

4. ANN

Model Architecture

```
ann_model = Sequential([
  Dense(hidden_units1, activation='relu'),
  Dense(hidden_units2, activation='relu'),
  Dropout(0.3),
  Dense(hidden_units3, activation='relu'),
  Dropout(0.4),
  Dense(1, activation='linear'),
])


ann_model.compile(
    loss='mse',
    optimizer='adam',
)
```

Trained for 50 epochs and batch size as 64, RMSE 385.24363997415014

- Used **KerasRegressor**, **GridSeachCV** and **KFold** for hyperparameter tuning.

Model Architecture used for Hyperparameter tuning.

```
hidden_units1 = 60
hidden_units2 = 25
model = Sequential([
  Dense(hidden_units1, activation='relu', kernel_initializer='normal'),
  Dropout(0.3),
  Dense(hidden_units2, activation='relu', kernel_initializer='normal'),
  Dense(1),
])

model.compile(loss='mean_squared_error', optimizer=optimizer)
```

GridSearchCV parameter's

```
batch_size = [30, 90, 180]
epochs = [10, 50]
optimizer = ['SGD', 'RMSprop', 'Adam']
```

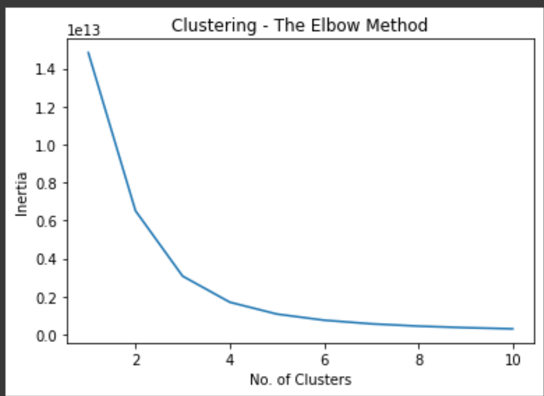
Best Estimator RMSE : RMSE : 0.19344919754254103
Best Estimatore Parameters :
{'batch_size': 180, 'epochs': 50, 'optimizer': 'SGD'}

5. Clustering stores using sales and customers visits as features.

- Model used for Clustering : **KMeans**
- Method Used for find the optimal number of cluster is **Elbow Method**.

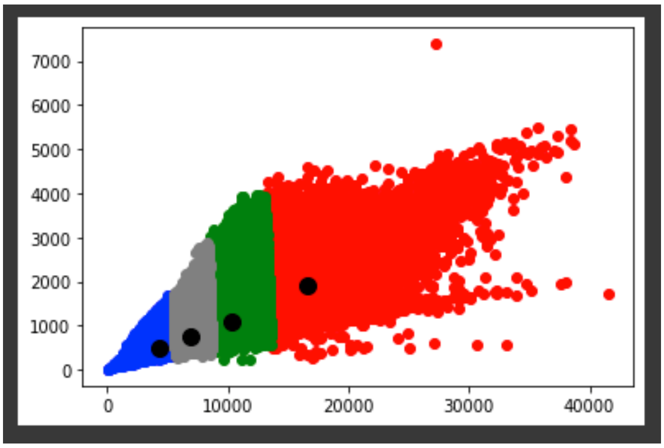
```
clusters_inertia = []
for i in range(1, 11):
  kmeans = KMeans(n_clusters=i, init='k-means++', random_state=99)
  kmeans.fit(clustering_data[['Sales', 'Customers']])
  clusters_inertia.append(kmeans.inertia_)
```

```
[ ] plt.plot(range(1,11), clusters_inertia)
    plt.title('Clustering - The Elbow Method')
    plt.xlabel('No. of Clusters')
    plt.ylabel('Inertia')
    plt.show()
```



- Optimal number of cluster found using the elbow method is 4.

- Visualization of the clusters.



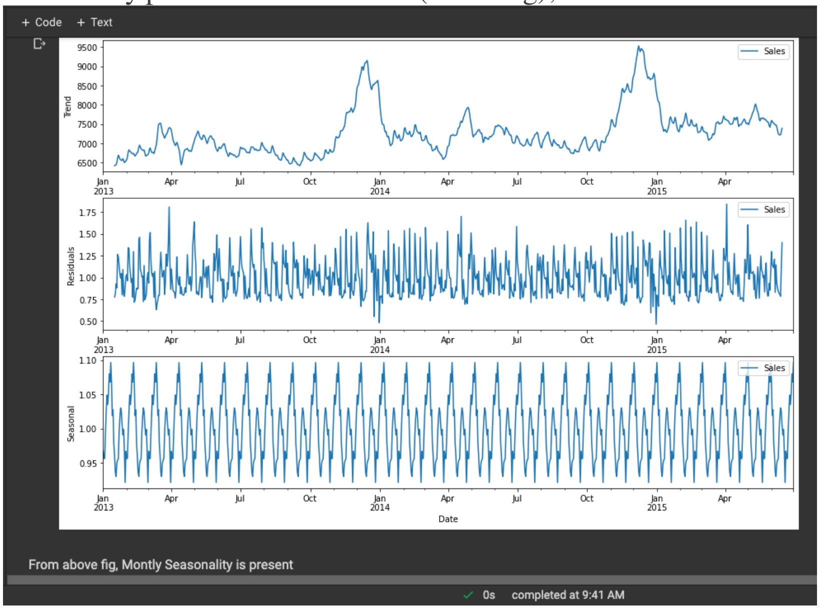
- RMSE score's of XGRegressor model for each of the above clusters,

XGBReg for Store_Cluster : 0 RMSE : 0.12777343309630645
XGBReg for Store_Cluster : 1 RMSE : 0.1573671486118741
XGBReg for Store_Cluster : 2 RMSE : 0.1053192664003051
XGBReg for Store_Cluster : 3 RMSE : 0.1078885463513178

6. Time Series Forecasting

- Seasonal ARIMA model

As there is seasonality present in the sales data(below fig), Seasonal ARIMA model is used.



Auto ARIMA model is used to find the optimal order values p, d, q and Seasonal values P, D, Q, m.

```
auto_arima(time_series_data.groupby('Date').mean()['Sales'],
                seasonal=True, m=7,
                max_p=4, max_d=4,max_q=4,
                max_P=4, max_D=4,max_Q=4,
                trace=True,
                error_action='ignore',
                suppress_warnings=True,
                stepwise=True).summary()
```

Optimal order values were (2,1,0) and seasonal values were (4, 0, 0, 7) with lowest AIC score as 15540.

SARIMAX Results

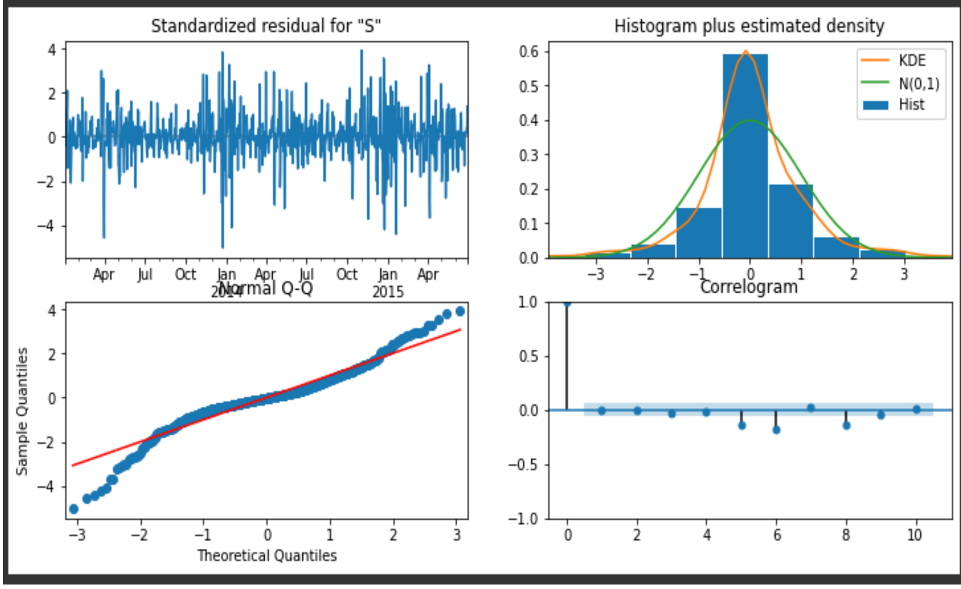
Dep. Variable:	Sales	**No. Observations:**	911
Model:	SARIMAX(2, 1, 0)x(4, 0, 0, 7)	**Log Likelihood**	-7763.229
Date:	Sun, 16 Jan 2022	**AIC**	15540.459
Time:	08:08:09	**BIC**	15574.153
Sample:	01-01-2013	**HQIC**	15553.323
	- 06-30-2015		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.3545	0.021	-16.622	0.000	-0.396	-0.313
ar.L2	-0.1622	0.027	-6.047	0.000	-0.215	-0.110
ar.S.L7	0.0807	0.025	3.176	0.001	0.031	0.130
ar.S.L14	0.5078	0.024	21.192	0.000	0.461	0.555
ar.S.L21	0.0810	0.026	3.078	0.002	0.029	0.133
ar.S.L28	0.1436	0.026	5.543	0.000	0.093	0.194
sigma2	1.485e+06	4.62e+04	32.125	0.000	1.39e+06	1.58e+06

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	509.75
Prob(Q):	0.93	**Prob(JB):**	0.00
Heteroskedasticity (H):	2.05	**Skew:**	-0.30

SARIMAX model diagnostics

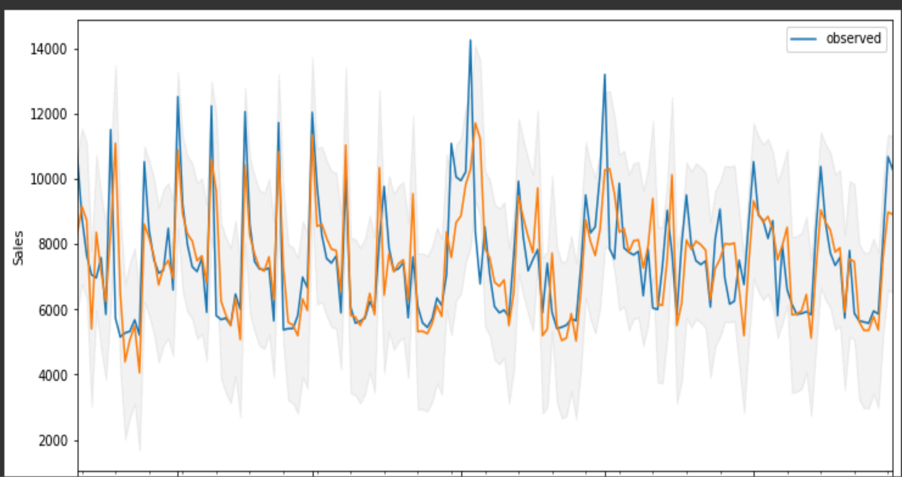
```
arima_result.plot_diagnostics(figsize=(12,6))
plt.show()
```



Model Validation

+ Code + Text

```
#Validating arima forecast
pred = arima_result.get_prediction(start = pd.to_datetime("2015-01-11"), dynamic = False)
pred_conf_int = pred.conf_int()

ax = ts_sales_data["2015-01-11":].plot(label = "observed", figsize=(12, 6), legend=True)
pred.predicted_mean.plot(ax = ax, alpha = 1)
ax.fill_between(pred_conf_int.index,
                pred_conf_int.iloc[:, 0],
                pred_conf_int.iloc[:, 1],
                color = "k", alpha = 0.05)

ax.set_xlabel("Date")
ax.set_ylabel("Sales")
plt.legend
plt.show()
```



✓ 0s completed at 9:41 AM

RMSE score: 1303.443722159693

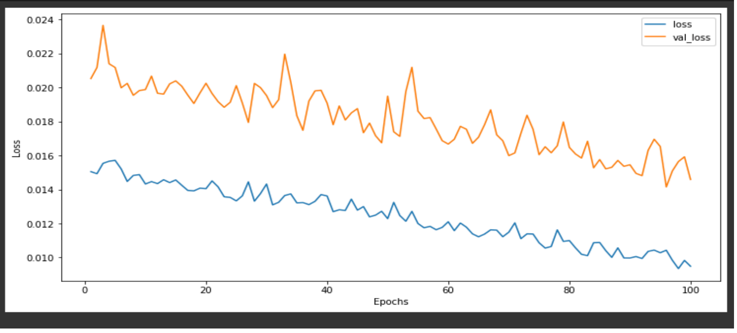
- **LSTM Forecasting**

  - Data was normalized with MinMaxScaler()
  - From Sales data, time series of 7 days dataset was created to feed the LSTM.
  - Model Summary

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param#
=================================================================
lstm_8  (LSTM)               (None,  7,  50)           10400
lstm_9 (LSTM)                (None, 7, 50)             20200
lstm_10 (LSTM)               (None, 50)                20200
dense_4 (Dense)              (None, 1)                 51
=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
```

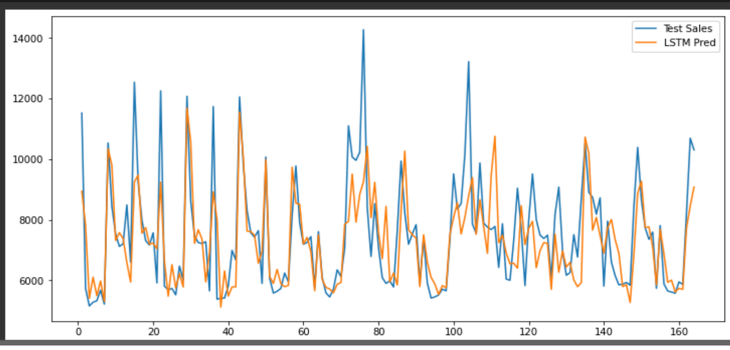
  - Above Sequential model trained for 100 epoch and 64 batch size.
  - Model loss visualization

```
#Model loss visualization
plt.figure(figsize=(12,6))
plt.plot(range(1,epochs+1), history.history['loss'], label='loss')
plt.plot(range(1,epochs+1), history.history['val_loss'], label='val_loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



  - Trained model forecasting visualization

```
#Test results visualization
plt.figure(figsize=(12,6))
plt.plot(range(1,len(y_test)+1), test_values[window:], label='Test Sales')
plt.plot(range(1,len(test_pred)+1), test_pred, label='LSTM Pred')
plt.legend()
plt.show()
```



  - RMSE score :

    Train RMSE : 7321.2716013796935

    Test RMSE : 7457.362736351862

Conclusion

RMSE scores of all the trained models

Simple LinearRegression - RMSE score: 1468.2049179921212
Simple LinearRegression for each store –
  Combined prediction RMSE: 486.01163821783393
  Avg. RMSE of all the models: 452.20473461235525
Regularized Regression - RMSE score: 1468.204792771783

==
Results after Considering features only for the open stores + Log Standardization of Sales
----
LinearRegression -RMSE score: 0.25095160243016823
XGBRegressor -RMSE score: 0.1307980346904776
GridSearchCV for XGBRegressor, best_estimator - RMSE score: 0.13098246642085956
XGBRegressor(best param found in GridSearchCV) –
  Combined prediction of all store models RMSE: 0.1291709892962826
  Avg. RMSE's of all the models: 0.12458709861495086
PCA + XGBRegressor(with best param found in GridSearchCV) –
  RMSE score: 0.15655728194229135

From above results, XGBRegressor (with best param found in GridSearchCV) trained for each store has better RMSE than other model.

==
Sales Time Series Forecasting

Seasonal ARIMA Forecasting RMSE: 1303.443722159693
LSTM Forecasting
  - Train data RMSE score: 7321.2716013796935
  - Test data RMSE score: 7457.362736351862

Seasonal ARIMA model has better RMSE than LSTM model.