Approach: Jobathon (Sep-2021)

Pavan Tiwari
Tiwaripav9427@gmail.com

9910552971
LinkedIn : https://www.linkedin.com/in/pavan-tiwari-80a7b31a5/

Data Analysis and Processing:

- **Null or duplicated data:** There was **no null data** or **any type of duplicated data**: hence no need for imputing, modification at this step.

- **Dealing With Categorical data:** The majority of features were categorical, only a few features were numerical including order and sales. I used the **Label Encoders** to convert my categorical features into numerical values. I used label encoding as i needed distinct **numerical replacement to categorical data**, also label encoding proves to be a good encoder for better results as it doesn't add any unnecessary columns.

- **Removal of feature not Present in Test dataset:** The number of **#orders feature had a very high correlation with the target variable**, but **it was not present in test data**, I created a separate model to first predict #orders for test data, then use it to predict Sales, but **my result proved better when the order was left out altogether**.

- **Detail Analysis of Data with Pandas Profile Report and Visualization:** I surveyed my data with a pandas profiling report of test and train dataset, to check each variable influence on the result, type of values, etc.
  Some interesting insights:

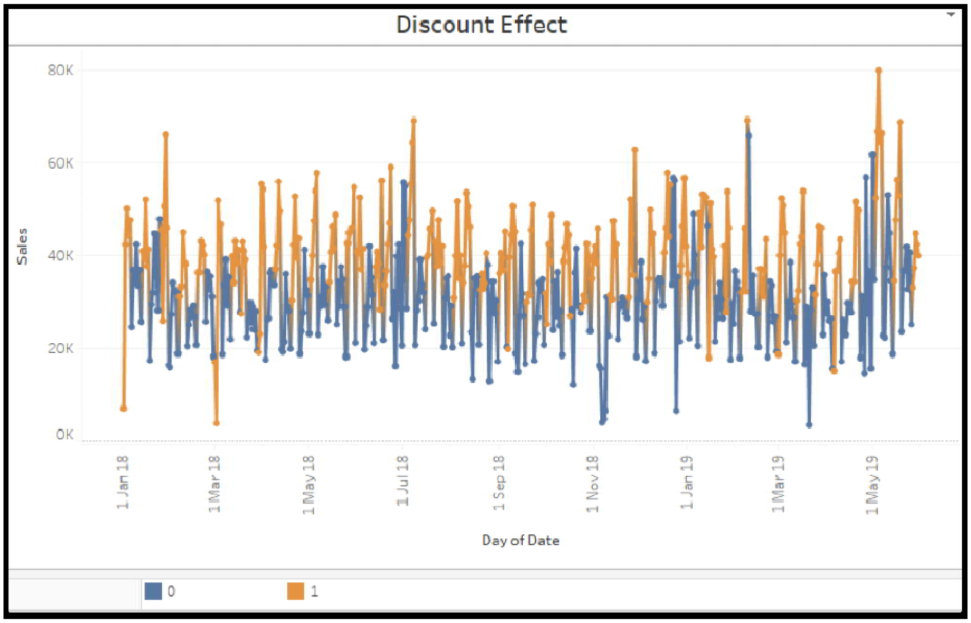
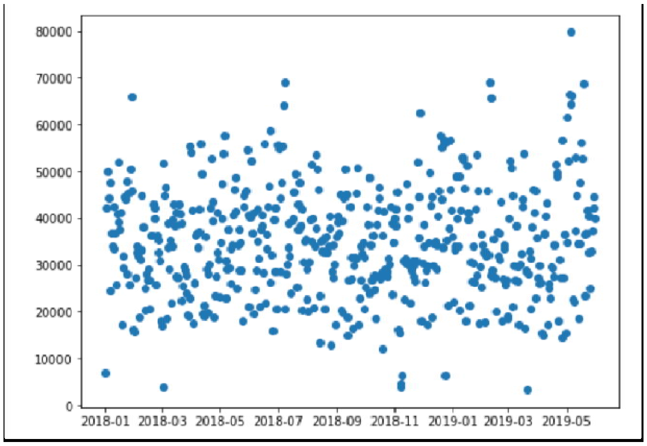
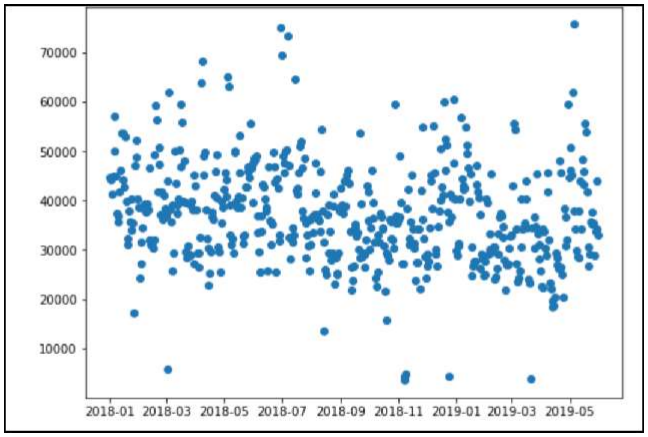


Figure: Effect of Discount on Sales

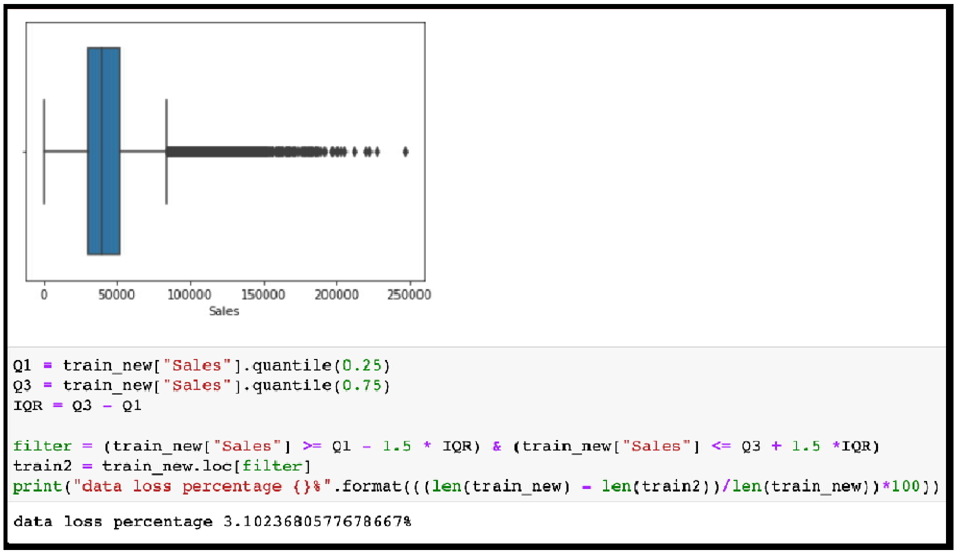


Sales Vs Time for Store_id == 1



Sales Vs Time for Store_id == 100

- **Dealing with Outliers:** The only effective numerical data was Sales, calculating outliers for Sales, I found only 3% data was in Outliers, As it was minimal as compared to my data size, I removed the Outliers and performance improved!



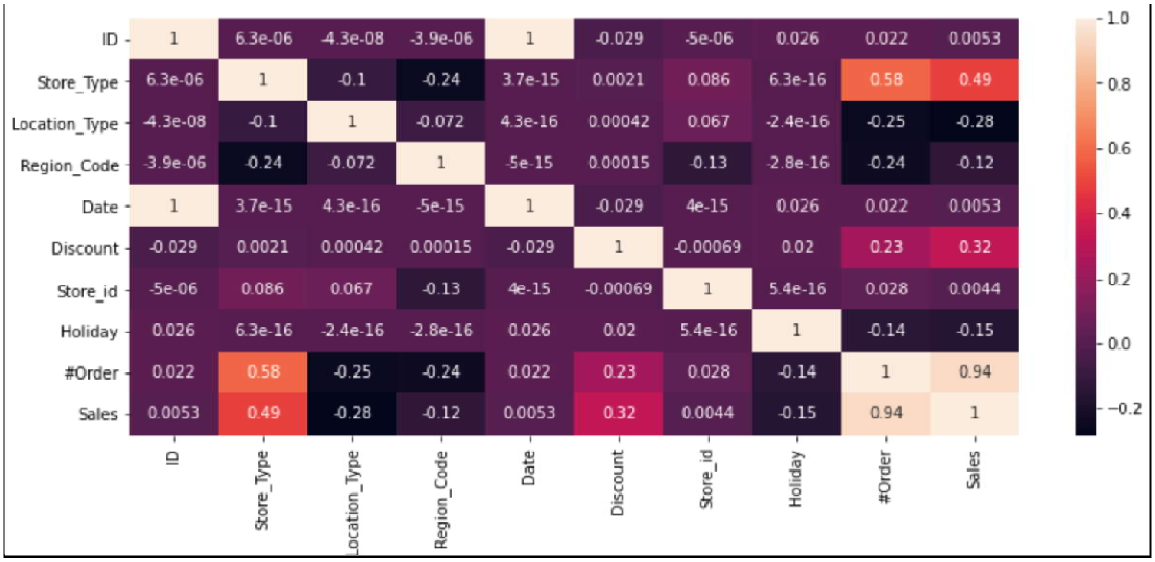
```
Q1 = train_new["Sales"].quantile(0.25)
Q3 = train_new["Sales"].quantile(0.75)
IQR = Q3 - Q1

filter = (train_new["Sales"] >= Q1 - 1.5 * IQR) & (train_new["Sales"] <= Q3 + 1.5 *IQR)
train2 = train_new.loc[filter]
print("data loss percentage {}%".format(((len(train_new) - len(train2))/len(train_new))*100))
```

```
data loss percentage 3.1023680577678667%
```

Figure: Outliers in Dataset

Feature Selection:

- **Correlations: 2 Types of Correlation Analysed**
  - **Correlation Among All Variables**

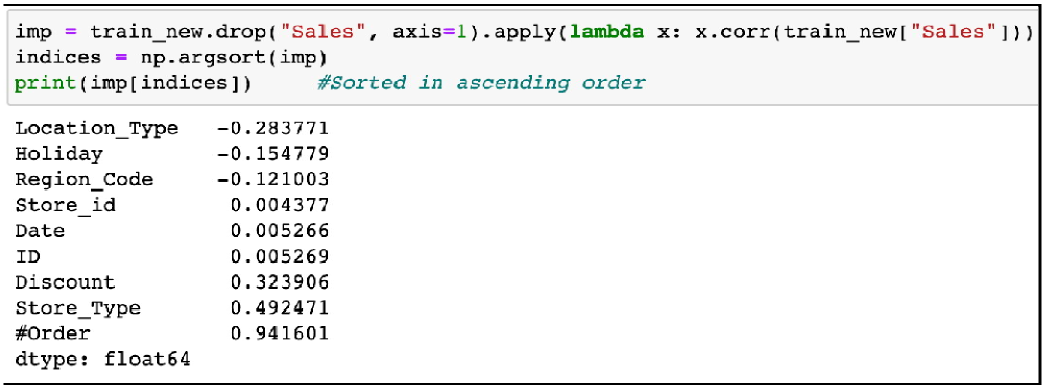


	ID	Store_Type	Location_Type	Region_Code	Date	Discount	Store_id	Holiday	#Order	Sales
ID	1	6.3e-06	-4.3e-08	-3.9e-06	1	-0.029	-5e-06	0.026	0.022	0.0053
Store_Type	6.3e-06	1	-0.1	-0.24	3.7e-15	0.0021	0.086	6.3e-16	0.58	0.49
Location_Type	-4.3e-08	-0.1	1	-0.072	4.3e-16	0.00042	0.067	-2.4e-16	-0.25	-0.28
Region_Code	-3.9e-06	-0.24	-0.072	1	-5e-15	0.00015	-0.13	-2.8e-16	-0.24	-0.12
Date	1	3.7e-15	4.3e-16	-5e-15	1	-0.029	4e-15	0.026	0.022	0.0053
Discount	-0.029	0.0021	0.00042	0.00015	-0.029	1	-0.00069	0.02	0.23	0.32
Store_id	-5e-06	0.086	0.067	-0.13	4e-15	-0.00069	1	5.4e-16	0.028	0.0044
Holiday	0.026	6.3e-16	-2.4e-16	-2.8e-16	0.026	0.02	5.4e-16	1	-0.14	-0.15
#Order	0.022	0.58	-0.25	-0.24	0.022	0.23	0.028	-0.14	1	0.94
Sales	0.0053	0.49	-0.28	-0.12	0.0053	0.32	0.0044	-0.15	0.94	1

Figure - Correlation Matrix

Except for Sales and Order, no other features had that high correlation with each other, and the Order was already not considered in training

  - **Correlation of independent Variables with Target variables:**



```
imp = train_new.drop("Sales", axis=1).apply(lambda x: x.corr(train_new["Sales"]))
indices = np.argsort(imp)
print(imp[indices])      #Sorted in ascending order
```

```
Location_Type    -0.283771
Holiday          -0.154779
Region_Code      -0.121003
Store_id          0.004377
Date              0.005266
ID                0.005269
Discount          0.323906
Store_Type        0.492471
#Order            0.941601
dtype: float64
```

Figure: Correlation of independent Variables with Target variables

As can be seen from calculations, Date, ID, Store_id were the features that had the least impact on the Target variable: sales.

Dealing with less correlational variables/ feature Selection:

1) ID: Removed for model training and prediction.
2) Date: I tried using the Date column in a different way with the help of the ARIMA model, the accuracy of train data remains the same, however, my accuracy for test data greatly decreased: MSLE: 426, hence I didn't use the date as a feature.
3) **Store_id:**
    a) I first train the model with no Store_id, my MSLE for test data showed around 244.
    b) **Using mean sales for a store:** I then grouped the train data by Store_id and calculated mean sales for a store when other parameters (holiday, discount, etc) were the same, the model performed exceptionally for my training data but not that well on test data (overfitting, as by grouping: train data reduced to minimal 1500 rows!)
    c) **Using max Sales of a single store:** calculated max sales for a store when other parameters (holiday, discount, etc) were the same. The reason I thought max would do good was **MSLE penalizes underestimates more than overestimates,** the model performed ok.
    d) **Best Performer:** the model performed best when encoded Store_id was passed directly to the model training.

Model Used:

I tried more than 10 algorithms, the model performed the same for all good algorithms, My best results were in the most simple approach, the params when kept at default performed best in my case (Still don't know how, but open to wonders)!

Best accuracy achieved with **RandomForest Regressor:**

MSLE*1000 = 224 on test data
MSLE*1000 = 100 on train data

Features Used to Predict Sales: **'Store_Type', 'Location_Type', 'Region_Code', 'Discount', 'Store_id', 'Holiday'**

Conclusion: Final Thoughts From This Hackathon
After exhaustive attempts of 2 days and giving my best, I received the best MSLE*1000 Error of 225 (rank 78!). I tried multiple approaches, used the data-centric approach by making changes to data multiple times (as many ways as my brain could think of) apart from changing models.

I tried multiple models (approx 10), also implemented an algorithm I never used before ARIMA (in this hectic pressures!), much to my disappointment the model didn't perform that well as compared to ordinary ones.

I also tried to group the data for same-store and make prediction store-wise instead of generic, which also didn't result in good results test results though performed exceptionally on my train dataset(overfitting because of fewer train data when clubbed).

My best results were in the most simple approach, **(Simplicity at its best!)** all the good models gave me almost equal results, with all features except Date, the params when kept at default performed best in my case(Still don't know how, but open to wonders)!

As this hackathon nears the closure, my mind is still restless with what can be the approach (eagerly waiting for top coder code file!!)

Still, I feel satisfied as I gave my 100% at every hour of this weekend challenge and learned a lot!

Peace :)