DATA71011 Report

- Rossmann Sales Analysis

Data Science MSc
Understanding Data & Their Environment

Student Id: 11063737

Introduction

Rossmann data has been shared with historical sales data to build a forecasting model to predict sales for each store up to six weeks in advance. Sales at different stores are influenced by different factors such as competition, promotions, holidays, seasonality & location. For the reliability of the forecasting model, the above factors & missing data should be taken into account.

Data Description

Three csv files have been provided: “train.csv”, “test.csv” & “store.csv”.
“store.csv” - 1115 observations(stores) and its features like promotion, competition & type.
“train.csv” - 1017209 observations(daily sales of different stores) from 2013 -2015.
“test.csv” - 1115 observations(stores) with the sales column omitted(to be predicted by the forecasting model)
The store column is common in both “train.csv” & “store.csv” and can be used to join the two datasets.

```
RangeIndex: 1115 entries, 0 to 1114
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Store                      1115 non-null   int64
 1   StoreType                  1115 non-null   object
 2   Assortment                 1115 non-null   object
 3   CompetitionDistance        1112 non-null   float64
 4   CompetitionOpenSinceMonth  761 non-null    float64
 5   CompetitionOpenSinceYear   761 non-null    float64
 6   Promo2                     1115 non-null   int64
 7   Promo2SinceWeek            571 non-null    float64
 8   Promo2SinceYear            571 non-null    float64
 9   PromoInterval              571 non-null    object
```

Fig 1. store.csv columns with their datatypes and non-null count (columns with null values will be solved in the next step of data preparation for better model accuracy)
Dtypes - int64, object & float64
`df_store = pd.read_csv('store. csv')`

Fig 2. train.csv columns & their data types (Every column has 1017209 values)
Dtypes - int64 & object

`df_train = pd.read_csv('train. csv', low_memory=False)`

```
RangeIndex: 1017209 entries, 0 to 1017208
Data columns (total 9 columns):
 #   Column         Non-Null Count    Dtype
---  ------         --------------    -----
 0   Store          1017209 non-null  int64
 1   DayOfWeek      1017209 non-null  int64
 2   Date           1017209 non-null  object
 3   Sales          1017209 non-null  int64
 4   Customers      1017209 non-null  int64
 5   Open           1017209 non-null  int64
 6   Promo          1017209 non-null  int64
 7   StateHoliday   1017209 non-null  object
 8   SchoolHoliday  1017209 non-null  int64
```

```
2   Date           41088 non-null  object
3   Sales          0 non-null      float64
4   Customers      0 non-null      float64
```

Fig 3. test.csv with no values for the Sales & Customers column.

Data Pre-processing

Considering the null count of the training dataset(Fig 2), no missing data imputation is required. However, on diving deep we find that there were days when stores had zero customers and sales. Over two years, 172817 is no. of times different stores were closed on given days. The breakdown of holidays was as follows:

- School Holiday: 3067
- Bank Holiday: 30140 (Easter/Christmas)
- Shops closed without disclosed reason: 139610 (No pattern is observed for these closings)

Since there is no explanation for 139610 closed observations. It is better to consider only open stores for our model. (to prevent false guidance). Hence, only open stores are analyzed as closed ones yield a profit of zero. The updated training set now has 844392 observations.

Next, the distributions of Customers and Sales were analyzed for any anomalies/outliers.

1) Sales:

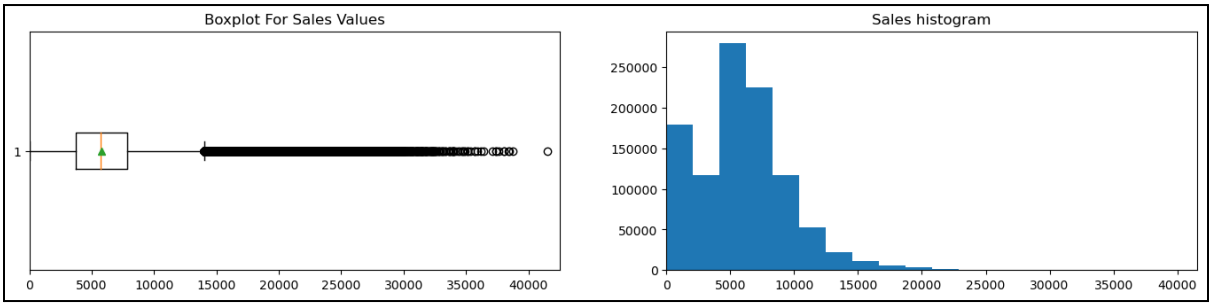

Fig 4. BoxPlot and Histogram representing outliers and distribution of Sales column (in train dataset)

```
Sales: {"Mean":np.mean(df_train.Sales),"Median":np.median(df_train.Sales)}
{'Mean': 6955.959133664481, 'Median': 6369.0}
```

From Fig 4. we can consider Sales>=14,000 as outliers which amount to 3.21 percent times more than average sales. Increased Sales can be due to many factors like popularity, no competition, etc.

2) Customers:

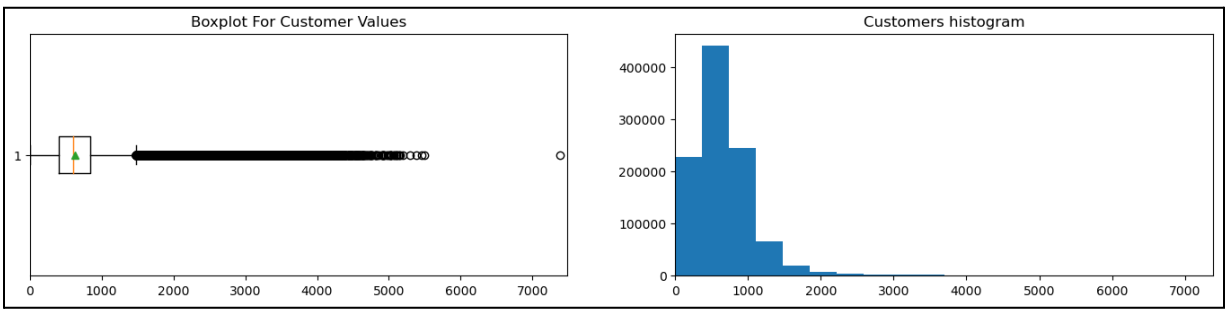

Fig 5. Boxplot & Histogram representing outliers and distribution of Customers column (in train dataset)

Customers:
`{"Mean":np.mean(df_train.Customers),"Median":np.median(df_train.Customers)}`
`{'Mean': 762.777166253325, 'Median': 676.0}`
Similarly, 4.37 percent (times) Stores had more Customers than usual. Reasons for this anomaly could be heavy promotions going on in the respected stores.

The outliers in both distributions push it to left showing right skewness.

`stats.pearsonr(df_train.Customers, df_train.Sales)[0] = 0.82355172020002143`
Sales & Customers represent 0.82 Pearson's correlation factor proving a direct relationship between Customers & Sales.

Missing Data Imputation

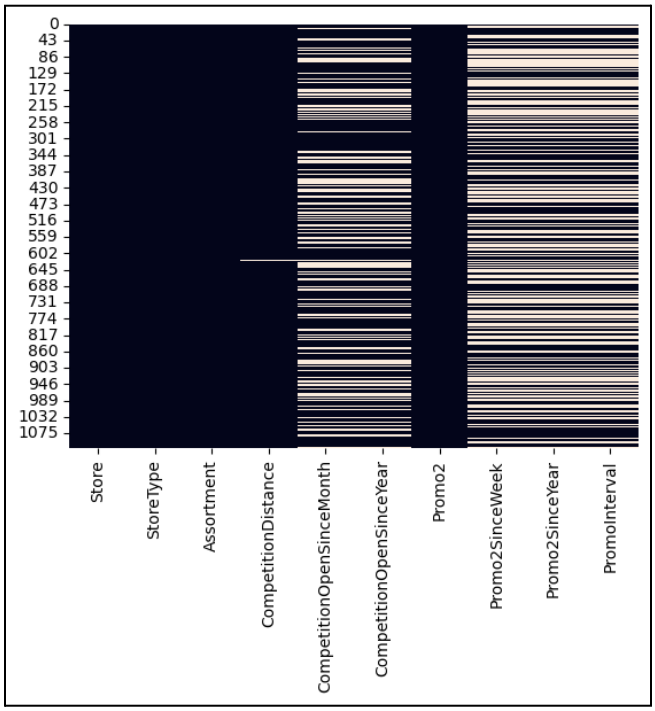

Fig 6. Heatmap representing Null values (through white portion)
Therefore, columns that need missing data imputation are -
ColumnName (Fill Rate %)

1) CompetitionDistance (99.73%)
2) CompetitionOpenSinceMonth (68.25%)
3) CompetitionOpenSinceYear (68.25%)
4) Promo2SinceWeek (51.21%)
5) Promo2SinceYear (51.21%)
6) PromoInterval (51.21%)

1) **CompetitionDistance**
df['CompetitionDistance'].skew() = 2.9285340174784116
Data is positive/right skewed & median imputation can be applied (since the mean is biased by those outliers) (From Fig 7)
`{'Mean': 5404.9010791366909, 'Median': 2325.0, 'Standard Dev': 7659.7282732764415}`

2) **CompetitionOpenSinceMonth**
Since data is fairly symmetrical, both KDE and skewness explain it. So, we replace null values with mean. (From Fig 8)
{'Mean': 7.2247043363994745}

3) **CompetitionOpenSinceYear**
It doesn't follow the rules of symmetry or skewness, so there is no accurate way to fill in missing data. Hence, zeroes can be placed in the place of null values.

```
df_store.CompetitionOpenSinceYear.fillna(0,inplace=True)
```

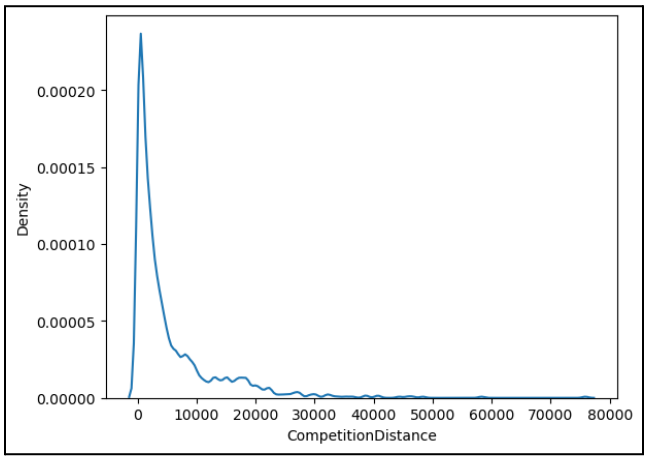

Fig 7. KDE plot for CompetitionDistance

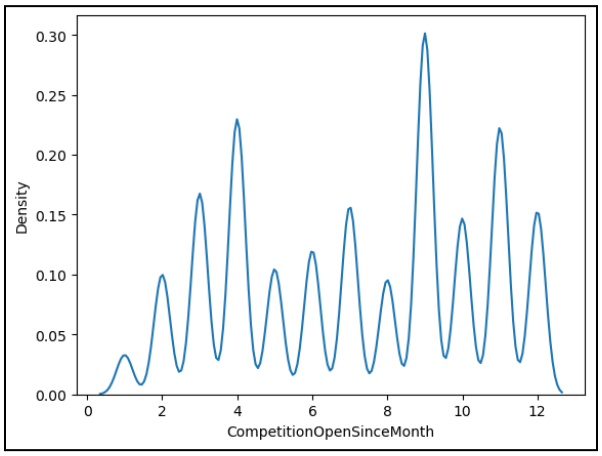

Fig 8. KDE plot for CompetitionOpenSinceMonth

4) **Promo2SinceWeek, Promo2SinceYear, PromoInterval**
The above three columns depend on **Promo2** and the missing values are for Promo2=0 which represents no promotion activities for those stores. As they are linked to Promo2 and have no promotions meaning those fields will be zero as well.
So, we replace the missing values with zero.

```
df_store.Promo2SinceWeek.fillna(0,inplace=True)
df_store.Promo2SinceYear.fillna(0,inplace=True)
df_store.PromoInterval.fillna(0,inplace=True)
```

Missing data in the store dataset is replaced with desired values.

```
df_store.count(0)/df_store.shape[0] * 100
```

Store	100.0
StoreType	100.0
Assortment	100.0
CompetitionDistance	100.0
CompetitionOpenSinceYear	100.0
CompetitionOpenSinceMonth	100.0
Promo2SinceWeek	100.0
Promo2SinceYear	100.0
PromoInterval	100.0
Promo2	100.0

Exploratory Data Analysis

Now, the store.csv & train.csv is combined through inner join with Store as the common column in both datasets. This helps us get all possible events.

```
df = pd.merge(df_store,df_train,how = "inner",on = "Store")
```

Multivariate Analysis -

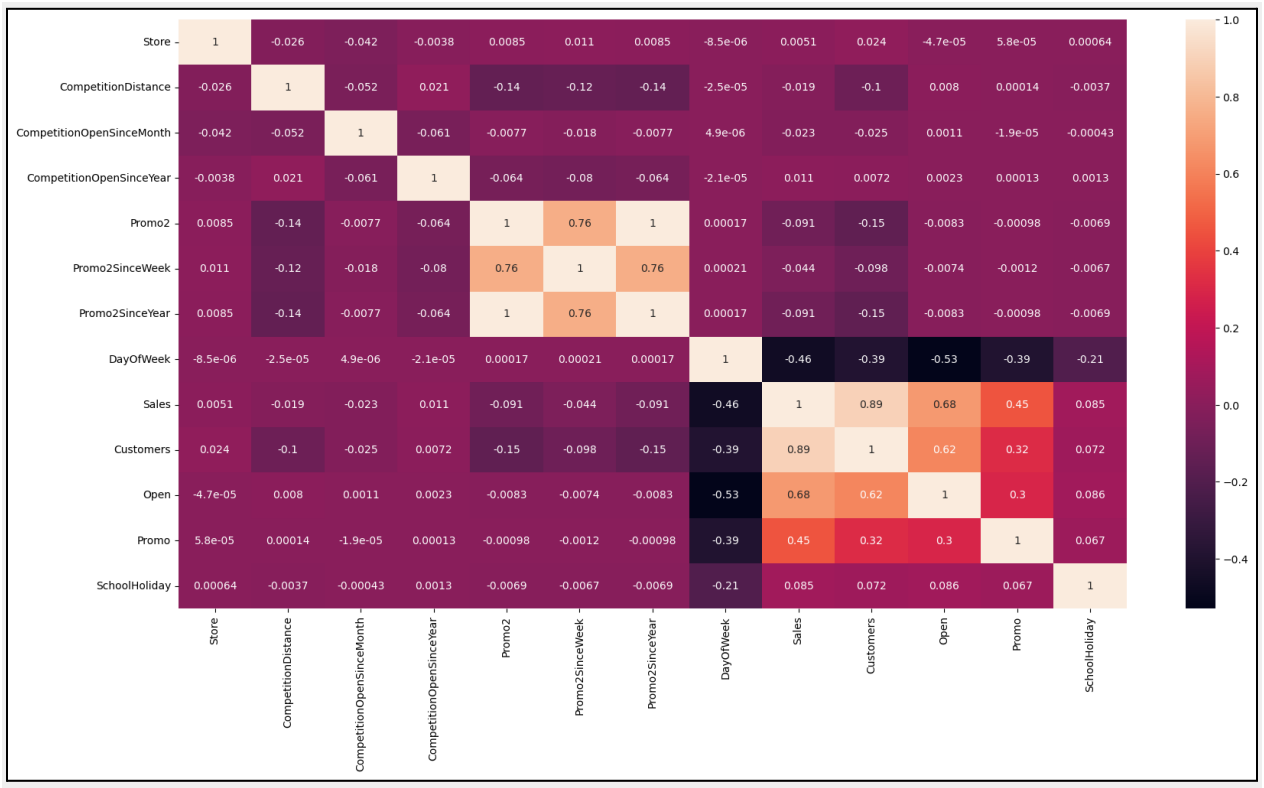

Fig 9. Correlation graph of different attributes

From the above graph, we can see a positive correlation between Sales & Customers in a store. Similarly, promotion and customers again show a positive correlation. A negative correlation of an attribute with Sales represents a drop in Sales.
Example - More competitors - fewer sales, or more promotion -more sales.

SalesPerCustomer (Fig 10) **-**
To get a better understanding & to normalize the data, we create a column 'SalesPerCustomer'. Through, this store which makes the customer more/less can be found.

```
df_train_store['SalesPerCustomer']=df_train_store['Sales']/df_train_store['Customers']
```

Some interpretations from Fig 10. are as follows:

- Among all four different types of Stores, A has the highest no. of Customers, Sales & Branches.
- However, Store D has the best average spending per customer. The reason includes higher competition distance making it a monopoly in its neighborhood.

- Store B with only 17 stores has the highest average sales & customers.

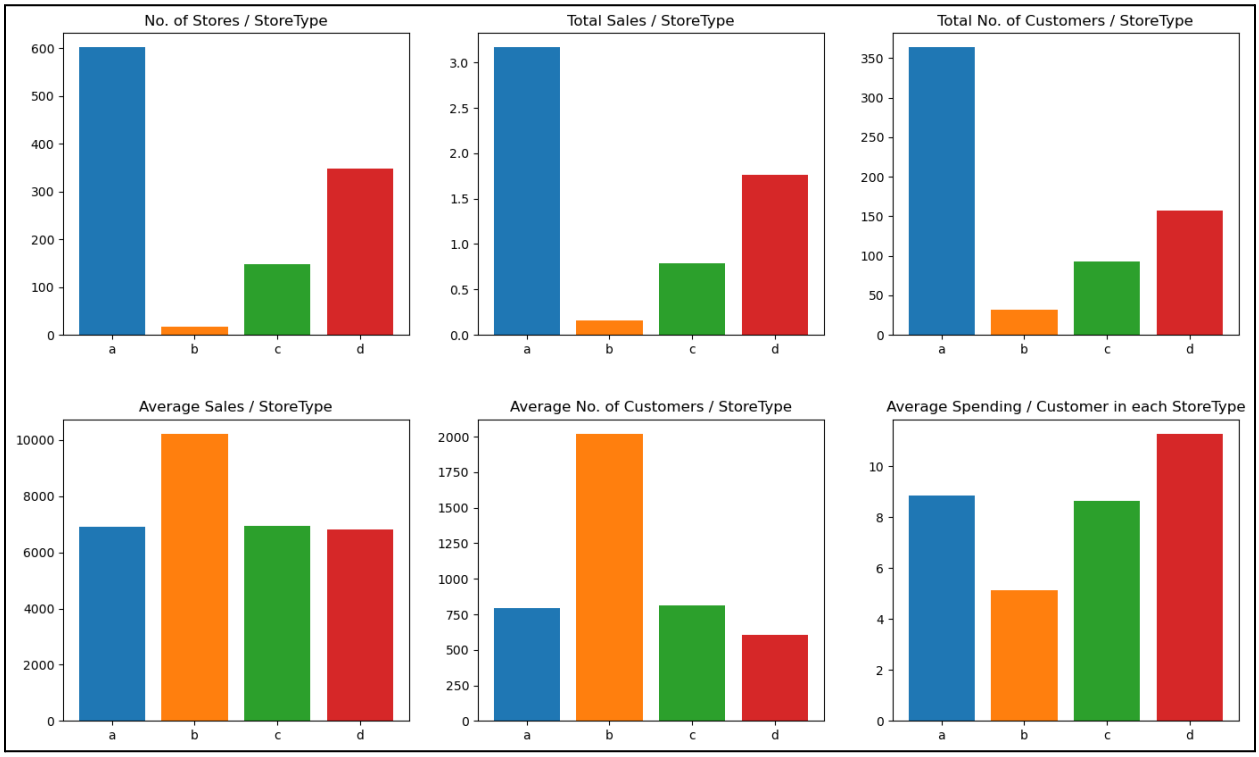

Fig 10. Different Store types with different averages/totals are represented through bar graphs.

Assortment (Fig 11) -

- A & C are the most common assortments type.
- Store D has the highest average spending per customer because it has mostly C assortment type, increasing the average bucket value.
- Store B is the only one with all assortment types, driving high traffic in this store.

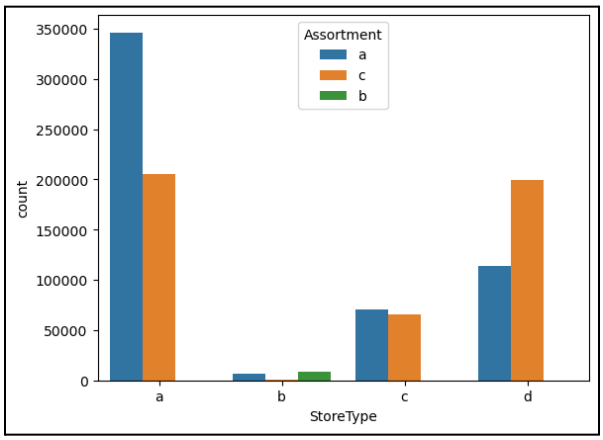

Fig 11. Assortment type available at different stores

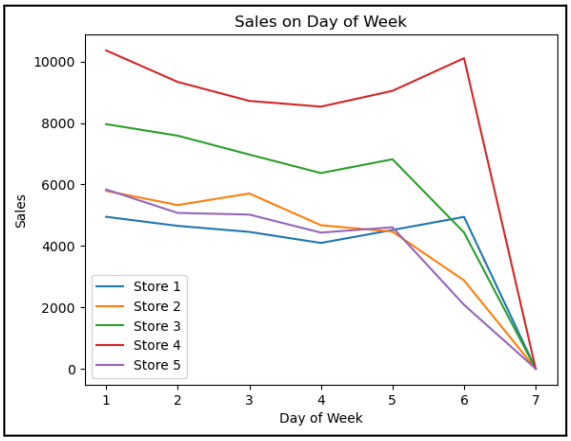

Fig 12. Sales on different days in a week

Promotion (Fig 13)
Promo = 0 & Promo = 1 show drastic change implying the role of promotion is crucial in sales of a store. Easter and Christmas months show a spike in sales when compared to not-a-holiday months. The SalesPerCustomer graph to promotion shows low sales initially and promotion shows an increase in buying of customers. Promotions are highest on Mondays with high sales. Promotions are kept closed on the weekends.

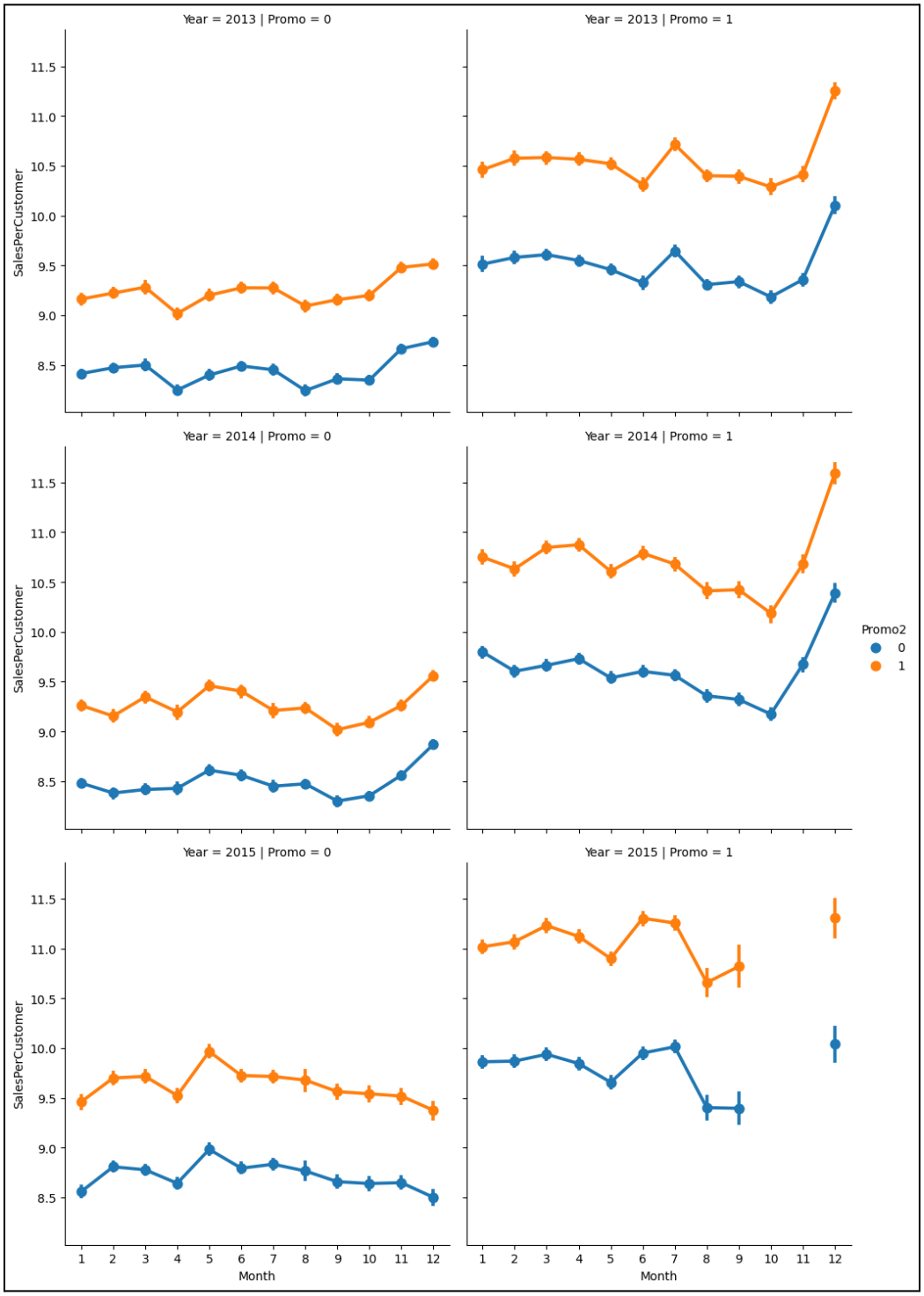

Fig 13. SalesPerCustomer to Year (Month-wise) graph with promotion as hue.

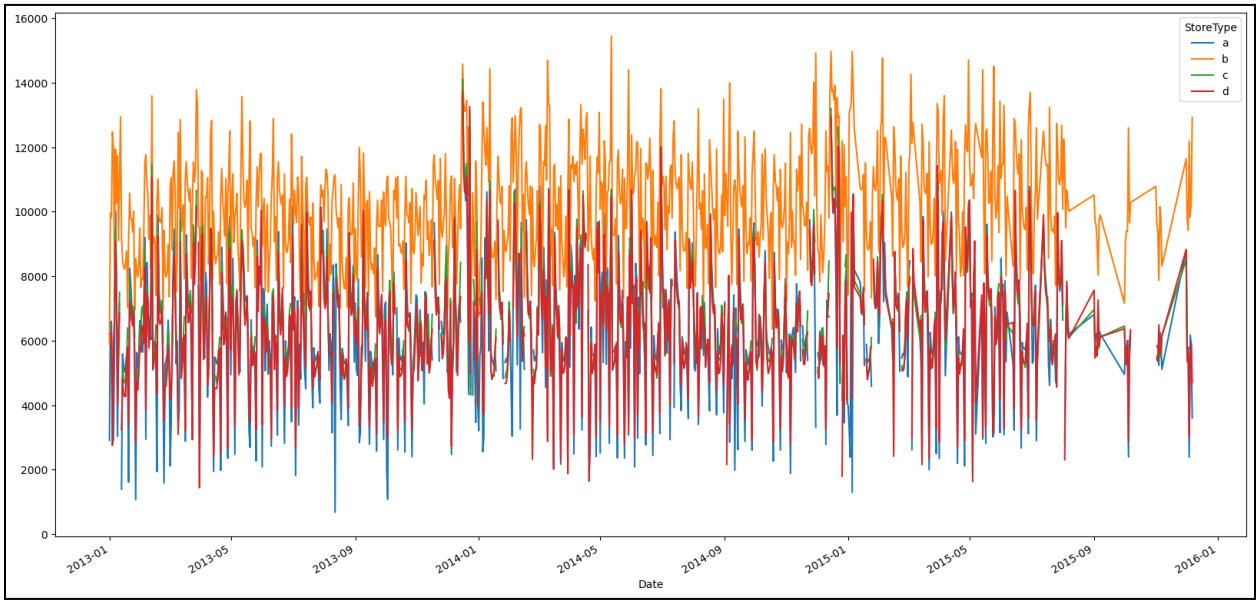

Fig 14. Line graph representing different Store types and their average sales.

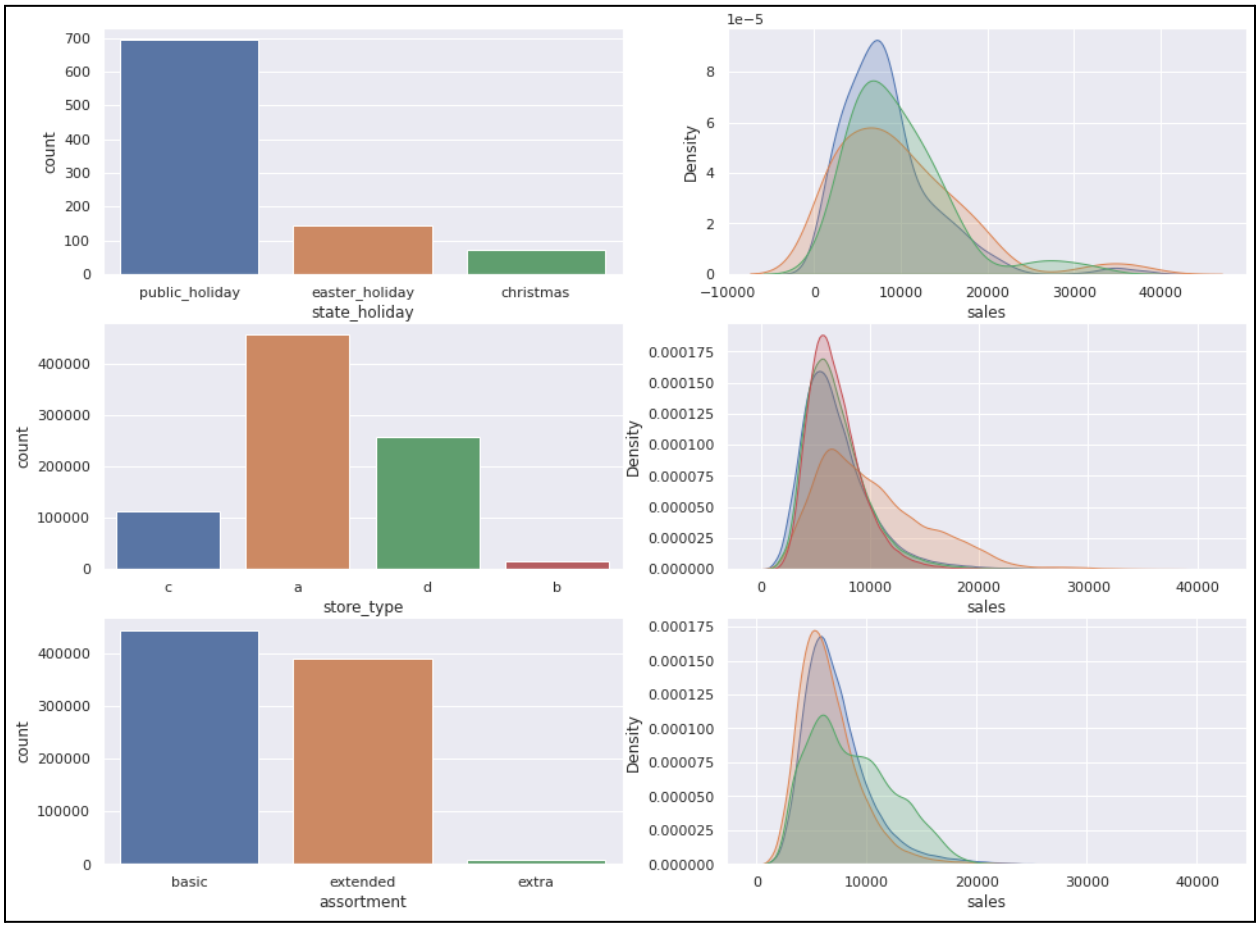

Fig 15. Amount of Sales with factors (their category) affecting it.

Model Selection & Fitting

Validation Metrics
We use RMSPE to compute our validation error.

Model,
The dataset is divided into an 80-20 ratio before building any model. Initially, every model was built using a part of the training data and another for validation. The model with the best validation score was run against the test dataset.

Before model fitting, we drop correlated columns like Customers, SalesPerCustomer & PromoInterval.
`xgboost_df = xgboost_df.drop(['Customers', 'SalesPerCustomer', 'PromoInterval'], axis=1)`
Also, we combine columns like `CompetitionOpenSinceMonth`, and `CompetitionOpenSinceYear` to CompetitionOpen and drop the original ones. Similarly, `PromoOpen` is a combination of `Promo2SinceWeek` & `Promo2SinceYear`.

```
xgboost_df['CompetitionOpen'] = 12 * (xgboost_df.Year - xgboost_df.CompetitionOpenSinceYear) + (xgboost_df.Month - xgboost_df.CompetitionOpenSinceMonth)
```

```
xgboost_df['PromoOpen'] = 12 * (xgboost_df.Year - xgboost_df.Promo2SinceYear) + (xgboost_df.WeekOfYear - xgboost_df.Promo2SinceWeek) / 4.0
```

Also, categorical columns like StateHoliday, Assortments & StoreType are converted to numerical columns.

```
category = {0:0, "0": 0, "a": 1, "b": 1, "c": 1}
xgboost_df["StateHoliday_Cat"] = xgboost_df["StateHoliday"].map(category)
xgboost_df["StoreType_Cat"] = xgboost_df["StoreType"].map(category)
xgboost_df["Assortment_Cat"] = xgboost_df["Assortment"].map(category)
xgboost_df = xgboost_df.drop(["StateHoliday", "StoreType", "Assortment"], axis = 1) # dropping not required columns
```

The forecasting model chosen for the future Sales Analysis is **XGBoost (Gradient Boosting).**

After splitting the data on the basis of Sales, the default optimum parameter for the XGBoost model is applied. (eta: 0.3 is default value & max_depth:10 is for more complexity)

```
params = {'max_depth':10, "booster": "gbtree", 'eta':0.3, 'objective':'reg:linear','enable_categorical':True}
```

Training the model

```
xgboost = xgb.train(params, dtrain, 100,
evals=watchlist,early_stopping_rounds= 100, verbose_eval=True)
```

Output : `[99]train-rmse:723.56967  eval-rmse:829.79503`
`Root Mean Squared Error for XGBoost: 829.7950310498247`

Feature Importance (Fig 16)
Similar to Random Forests & Decision Trees, XGBoost also provides us with Feature importance score for each/different columns.

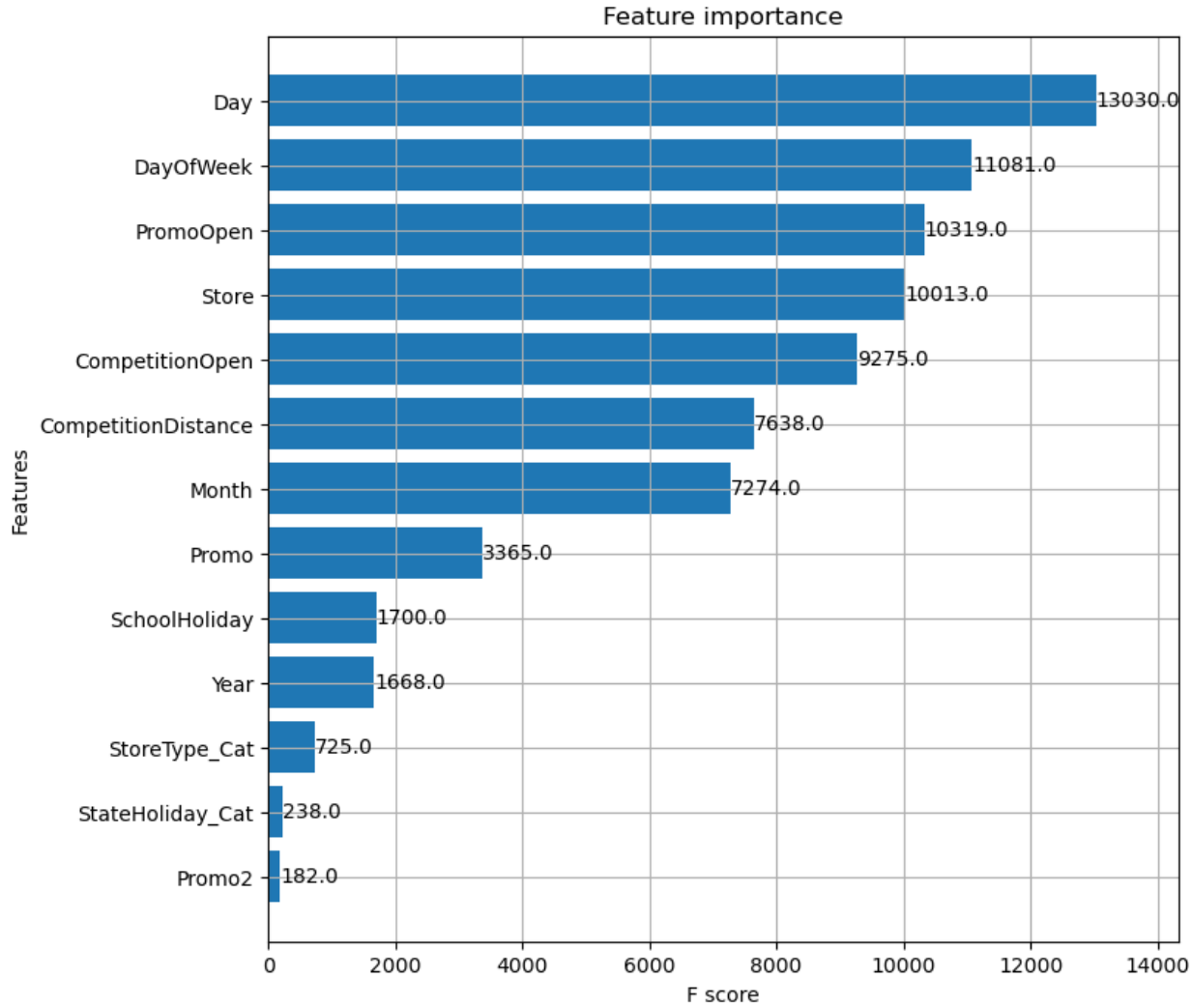

Fig 16. Feature Importance graph for XGBoost Model

Results & Conclusion

Initial decision included choosing Random Forest as the suitable forecasting model because of its advantages like fast run times, and ability to deal with missing data. However, its disadvantages like hard to interpret because it consists of more than 50 decision trees. RF is biased for categorical variables with a different number of levels while calculating feature importance.
XGBoostt sums up the prediction of multiple trees. The boosting is an additive process with a trained model addition with every prediction. XGBoost and RF both generate decision trees. GB builds it over residuals whereas RF uses bootstrapping. Own loss functions like RMPSE can be established in GB for optimization but not in RF.

Some observations after running XGBoost predicted Sales for the Test dataset are as follows -

1) Sales are not affected by the opening/closing dates of a Store. (Example - A store closed on Monday should not expect increased sales on Tuesday)
2) Sales are affected by DayOfMonth. (Ex - Sales are higher on weekends/paydays)
3) SalesPerCustomer or AverageSalesPerCustomer increased the efficiency of the model. As the performance of a Store depends on Profits, Profits depend on Sales and Sales depend on Customers. However, considering the majority of Customers just browsing Stores decrease the accuracy of our model.
4) Hence, SalesPerCustomer can be considered as the best measure/indicator to measure the performance of a store.

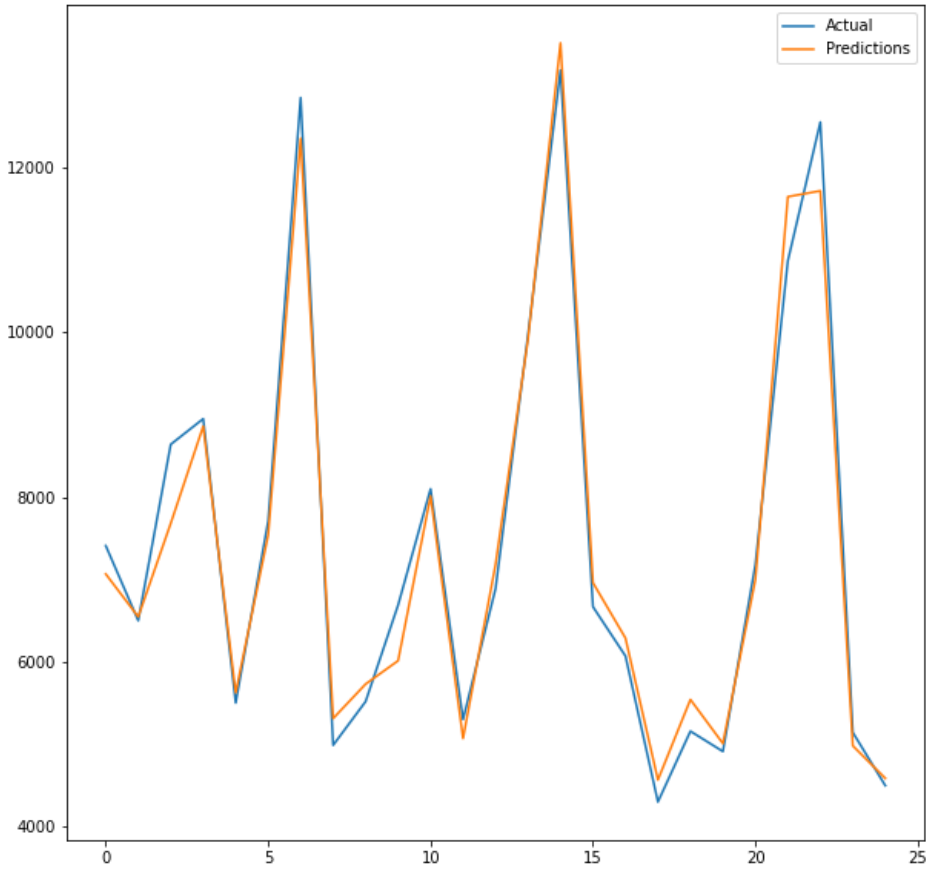

Fig 17 shows the predicted future sales

Conclusion
The Rossmann dataset is more focused on data cleaning and feature selection than model selection. Analyzing the data was the major step.
`Best Indicator: SalesPerCustomer`
`Best Model: XGBoost > Random Forest`

Reasons for choosing Gradient Boosting model -

1) XGBoost predictions are Stable with low variance.
2) Trends & Seasonal fluctuation (like holidays) can be identified by the model.
3) It can handle time variables.

Assumptions

- DATA71011 module is considered to be more focused on Data Preparation (which includes data description, pre-processing, cleaning, and missing data) rather than model building. Hence, the report is structured in a way where the majority portion represents Data Preparation and the rest describes the Forecasting Model and its reliability.
- Considering, a data science firm report rather than code graphs/images are added considering the tech understanding of a wider audience.
- Considering the word limit all attributes and their imputation/cleaning have not been explained in detail. Similarly, the model choice reasons are also within the word limit.

References

J. Fan, Q. Yao, Nonlinear Time Series. Nonparametric and Parametric Methods, Springer Series in Statistics, Springer, New York (2003).

Chen, T. & Guestrin, C. (2016) „XGBoost: A scalable tree boosting system", in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16. New York, USA: ACM Press, p. 785–794

Ghosh, R. & Purkayastha, P. (2017) „Forecasting profitability in equity trades using random forest, support vector machine, and xgboost", in 10th International Conference on Recent Trades in Engineering Science and Management, p. 473–486.

Gurnani, M. et al. (2017) „Forecasting of sales by using the fusion of machine learning techniques", in 2017 International Conference on Data Management, Analytics and Innovation (ICDMAI). IEEE, p. 93–101

Gumus, M. & Kiran, M. S. (2017) „Crude oil price forecasting using XGBoost", in 2017 International Conference on Computer Science and Engineering (UBMK). IEEE, p. 1100–1103.