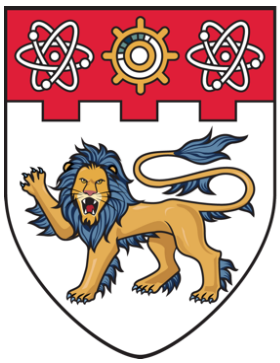
CZ 4041 - Machine Learning



NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Rossmann Store Sales Challenge

Private Score: 0.11880 (Leaderboard: ~414th out of 3,303 Submissions)
Public Score: 0.10640 (Leaderboard: ~1,055th out of 3,303 Submissions)

Group 5
Nikhil Venkatesh (U1423078G)
Suyash Lakhotia (U1423096J)
Prasanth Karthikeyan (U1421934F)
Rainy Sokhonn (U1423091B)

Table of Contents

Problem Description

Background Information

Rossmann operates a chain of over 3,000 drug stores in 7 European countries. Currently, Rossmann store managers are tasked with predicting their daily sales for up to six weeks in advance. Store sales are influenced by many factors, including promotions, competition, school and state holidays, seasonality, and locality. With thousands of individual managers predicting sales based on their unique circumstances, the accuracy of results can be quite varied.

Problem Statement

Predict 6 weeks of daily sales for 1,115 drug stores operated by Rossmann located across Germany to enable store managers to create effective staff schedules that increase productivity and motivation.

Kaggle Evaluation

Evaluation is based on the Root Mean Squared Percentage Error (RMSPE) metric, which should be minimized:

$$\text{RMSPE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{y_i}\right)^2}$$

where y_i denotes the sales of a single store on a single day and $\hat{y}_i$ denotes the corresponding prediction. Any day and store with 0 sales is ignored.

Understanding the Dataset

There are two training datasets provided to us (train.csv & store.csv) and one test dataset (test.csv) whose sales we have to predict.

train.csv

This is the main training dataset which contains data about the sales figures for a store on a particular date.

Data Fields:

1. *Store*: a unique numerical store identifier (1 - 1,115)
2. *DayOfWeek*: the day of week (1 - 7)
3. *Date*: the date, ranging from 2013-01-01 to 2015-07-31
4. *Sales*: the turnover of the specified store on the specified date
5. *Customers*: the number of customers of the specified store on the specified date
6. *Open*: indicates whether the store was open (0 = Closed, 1 = Open)
7. *Promo*: indicates whether the store was running a promotion (0 = No, 1 = Yes)
8. *StateHoliday*: indicates if it was a state holiday (a = Public Holiday, b = Easter holiday, c = Christmas, 0 = None)
9. *SchoolHoliday*: indicates if it was a school holiday (0 = No, 1 = Yes)

No. of Records: 1,017,209

store.csv

This dataset contains supplementary information about each of the 1,115 stores and helps identify unique features which may (or may not) affect sales.

Data Fields:

1. *Store*: a unique numerical store identifier (1 - 1,115)
2. *StoreType*: differentiates between the 4 different types of stores (a, b, c, d)
3. *Assortment*: describes the assortment of goods carried by the store (a = Basic, b = Extra, c = Extended)
4. *CompetitionDistance*: the distance (in metres) to the nearest competitor's store
5. *CompetitionOpenSinceMonth*: the month in which the competition opened
6. *CompetitionOpenSinceYear*: the year in which the competition opened
7. *Promo2*: indicates if a store is participating in a continuing and consecutive promotion (0 = No, 1 = Yes)

8. *Promo2SinceWeek*: the week of the year in which the store began participating in *Promo2* (from 1 - 52, presumably, but some weeks are unrepresented in the data)
9. *Promo2SinceYear*: the year in which the store began participating in *Promo2* (from 2009 - 2015)
10. *PromoInterval*: describes the consecutive intervals in which *Promo2* is activated, giving the months the promotion is renewed (either "Jan,Apr,Jul,Oct", "Feb,May,Aug,Nov" or "Mar,Jun,Sept,Dec")

No. of Records: 1,115

test.csv

This dataset is to be used for testing and evaluating the model.

Data Fields: Same as train.csv, with the exclusion of *Customers* & *Sales* (*Sales* to be predicted by the model) and an additional field *Id* which represents a (*Store*, *Date*) tuple that is used to label predictions for submission to Kaggle.

No. of Records: 41,088

Preprocessing the Data

Replacing Missing Values & Fixing Corrupted Data

- **Replacing NaN values for *Open*:** Missing values in the *Open* data field were replaced with '1' if the *DayOfWeek* was not Sunday.
- **Replacing NaN values for *CompetitionDistance*:** Since no record exists where *CompetitionDistance* is NaN and *CompetitionOpenSince[X]* is not NaN, the *Competition-Distance* was set to 0 if it was NaN.
- **Replacing NaN values for *CompetitionSince[X]*:** If *CompetitionDistance* is not 0 but the *CompetitionSince[X]* columns are missing, the earliest possible values were inserted i.e. 1900 and 01 for *CompetitionSinceYear* & *CompetitionSinceMonth* respectively.
- **Converting all 0 (number) values in *StateHoliday* to "0" (string) values:** Data in the *StateHoliday* is a mix of numerical and string values. Hence, for the sake of consistency, all values were converted to string.

One-Hot Encoding

Using the *pandas.get_dummies()* function available in the Pandas library for Python, we performed one-hot encoding on the categorical features present in our dataset, including *DayOfWeek* & *StateHoliday* in train.csv & test.csv and *StoreType* & *Assortment* in store.csv.

Additional Features

- ***DayOfMonth, Year, Month, YearMonth & WeekOfYear*:** These data fields were extracted from the original *Date* column to provide a better understanding of date-related trends.
- ***StateHolidayBinary*:** Apart from one-hot encoding *StateHoliday*, another column was created that simply indicated whether it was a state holiday using 0 or 1.
- ***AvgCustStore, AvgCustStoreMonth, AvgCustStoreYear*:** Stores the average no. of customers per store (overall), per store per month and per store per year respectively.
- ***IsPromoMonth*:** Indicates if the record is in the month of a running *Promo2*.
- ***CompetitionOpenYearMonthInteger*:** Stores the YYYYMM version of *Competition-OpenSince[X]*.
- ***AvgSales, AvgCustomers, AvgSalesPerCustomer*:** Stores the average sales per day, average no. of customers per day and average sales per customer per day respectively for each store.

Analyzing the Dataset

Basic Analysis

Min, Max, Mean & S.D.

Field Name	Min	Max	Mean	Standard Deviation
Store	1	1,115	N/A	N/A
Customers	0	7,388	633.1459	464.4117
Sales	0	41,551	5,773.819	3,849.926

Frequency Distribution

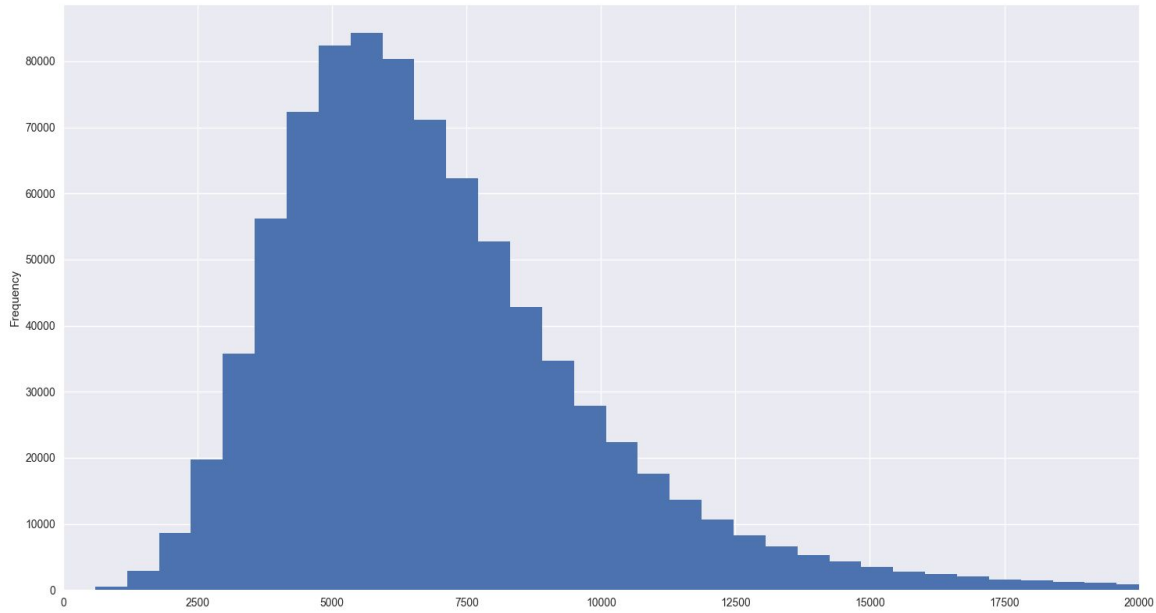


Fig. 1: Frequency of *Sales* Values for Open Stores

By looking at the chart above, we can see that the *Sales* values forms a Poisson-like distribution, reaching its peak of frequency between 5,000 and 7,500. At the same time, it is uncommon for *Sales* to fall below 2,500 (on open days) or go beyond 12,500. The average value for *Sales* for open stores is 6,955.514.

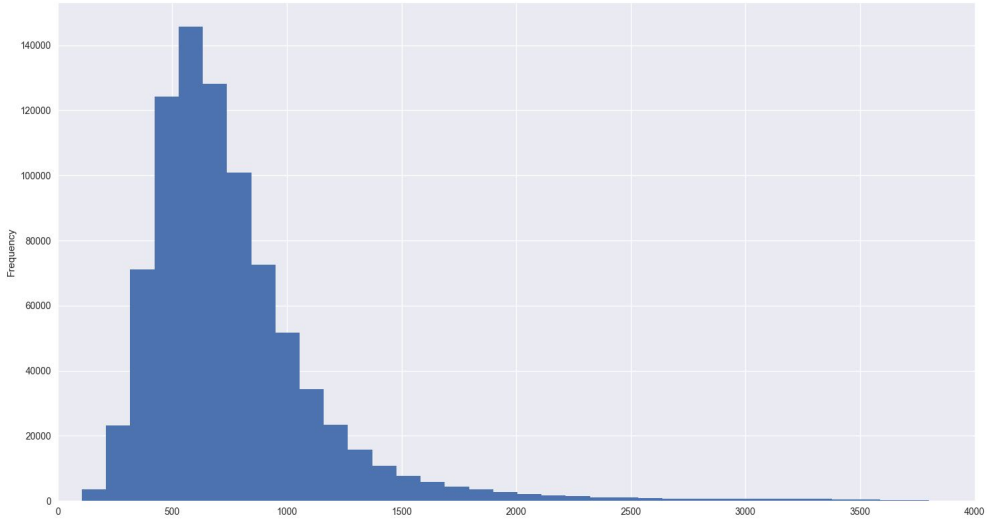


Fig. 2: Frequency of *Customers* Values for Open Stores

Similarly, the distribution of *Customers* values follows a Poisson-like distribution, indicating that the number of customers commonly ranges between 500 and 1,000, while a value higher than 1,000 has relatively low frequency. The average value for *Customers* for open stores is 762.728.

Possible Outliers

When studying the data distribution, several records were highlighted as possible outliers in the data. Firstly, there are 52 records in the training data where the store is indicated to be open but *Sales* and *Customers* are both 0. This could mean that the store was mistakenly indicated as being open. Because of this, we decided to exclude all records where the store was open but *Sales* was 0 from our training data to avoid such erratic records.

Additionally, there were several records that seemed to have unusually high *Sales* values.

Sales Value	No. Of Records
> 45,000	1
> 35,000	18
> 30,000	153
> 25,000	758

This suggests that *Sales* values greater than 35,000 can be safely ignored to avoid overfitting as those records are extremely rare and only occur in a handful of stores but could affect the model's predictions for the other stores.

Trend Analysis for Date-Related Fields

Weekly Trends

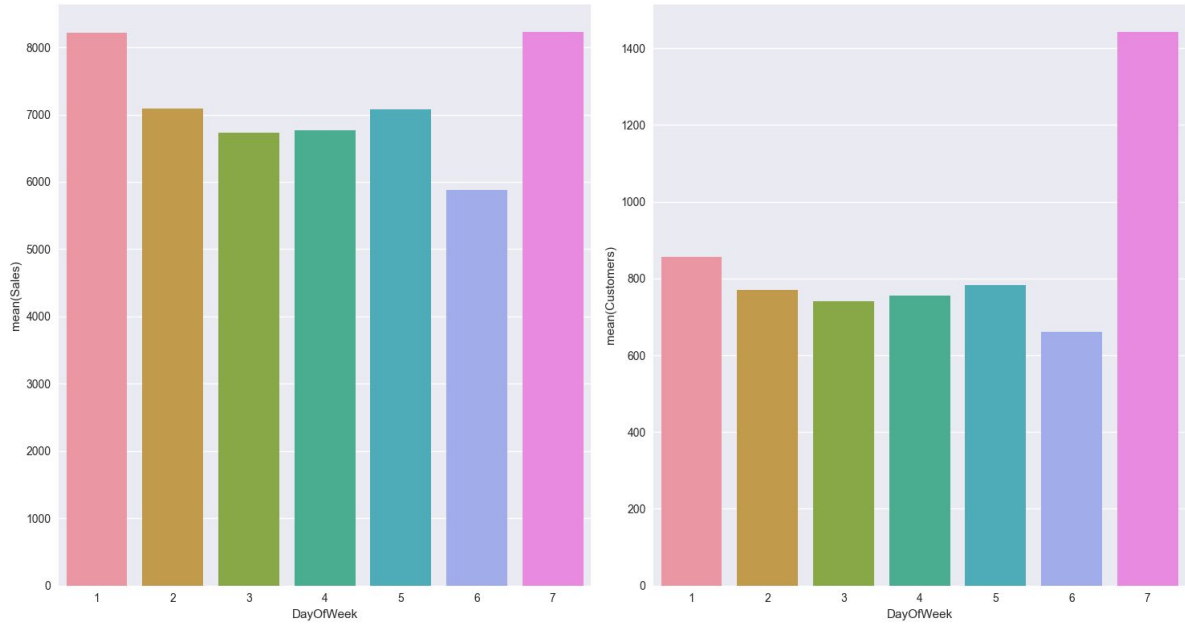


Fig. 3: Average Sales & Average Customers for Open Stores by Day of Week

As can be seen in the chart above, there are three peaks in the average sales and average no. of customers in the week. The first two are on Mondays & Fridays, which is probably due to these days being the start and end of the work week. The highest peak, however, is on Sundays, which is probably due to the fact that most other stores are closed. An interesting observation here is that there is a larger difference between the average no. of customers and average sales on Sundays as compared to other days of the week. While the average sales is approximately the same on Sundays & Mondays, there is a drastic difference in the no. of customers, which hints at a large number of window shoppers on Sundays.

Annual Trends

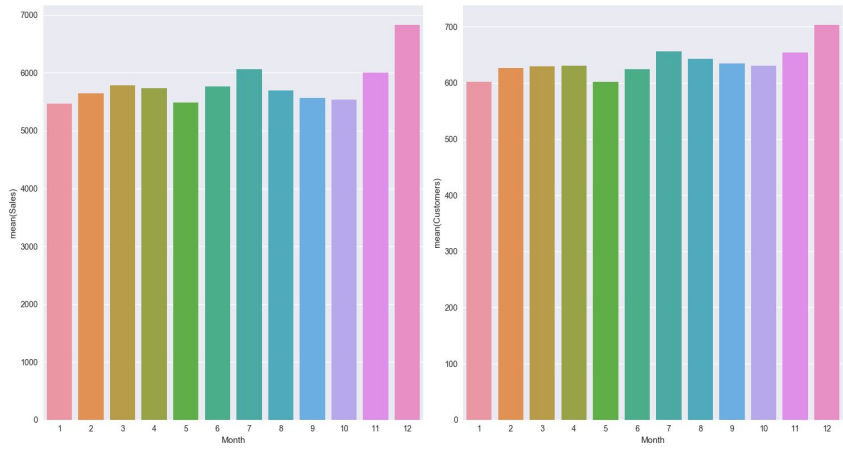


Fig. 4: Average Sales & Average Customers by Month

When plotting the average sales & no. of customers by month, we see an annual trend similar to most retail stores. There is an uptick in the months of June - July owing to summer holidays and another bigger uptick in December, which is likely due to the holiday season.

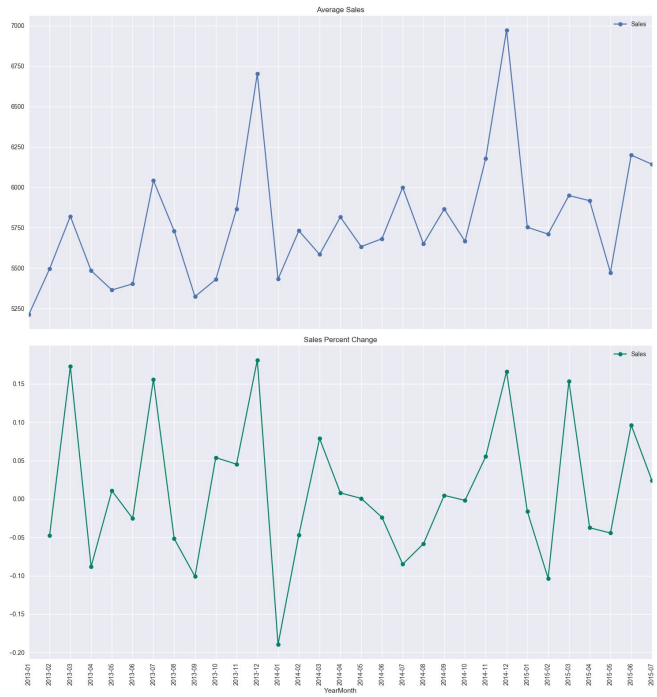


Fig. 5: Average Sales & Percentage Change by Year-Month

The annual trends are further corroborated when plotting the average sales data per month for the entirety of the training dataset (January 2013 to July 2015). We see the same annual trends being followed in the 2.5 years, with increasing peak values every year hinting at improved store performance from 2013 to 2014 to 2015.

Effect of Promotions & Holidays

Promo

The figure below shows the average customers and average sales (across all stores) on days with and without promotions. The effect of having a promotion on sales and customers is clearly evident, with the promotion having a strong positive effect on both.

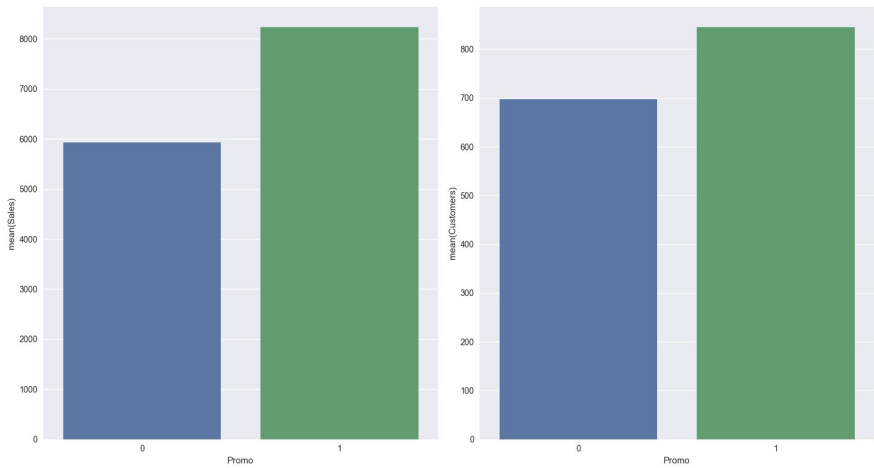


Fig. 6: Average Sales & Average Customers for Open Days with/without Promo

State Holidays

Since most stores are closed on public holidays and closed stores can be ignored when making predictions, the average sales figures and average number of customers on state holidays is plotted, but only for open stores in the figure below. It is evident that when a store is open on state holidays, it attracts more customers and accrues more sales.

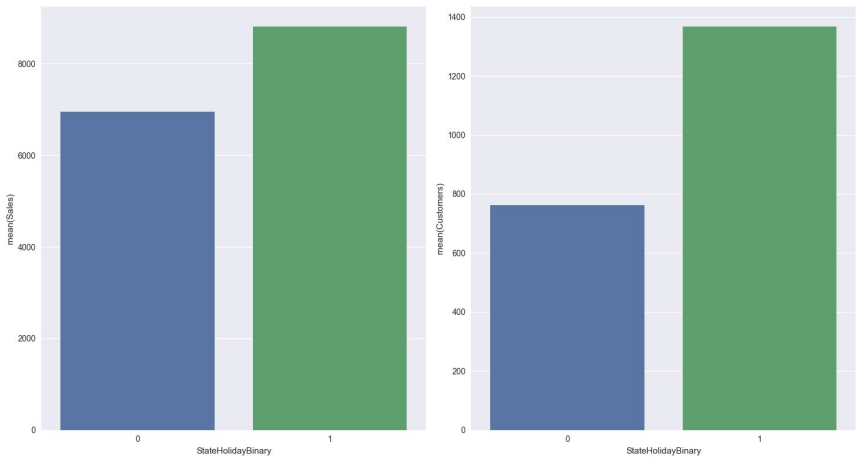


Fig. 7: Average Sales & Average Customers for Open Stores on State Holidays

When the plot is further broken down to the individual holidays in the figure below, we see that Christmas and Easter have the best average sales, with public holidays second and non-holidays faring the worst for average sales.

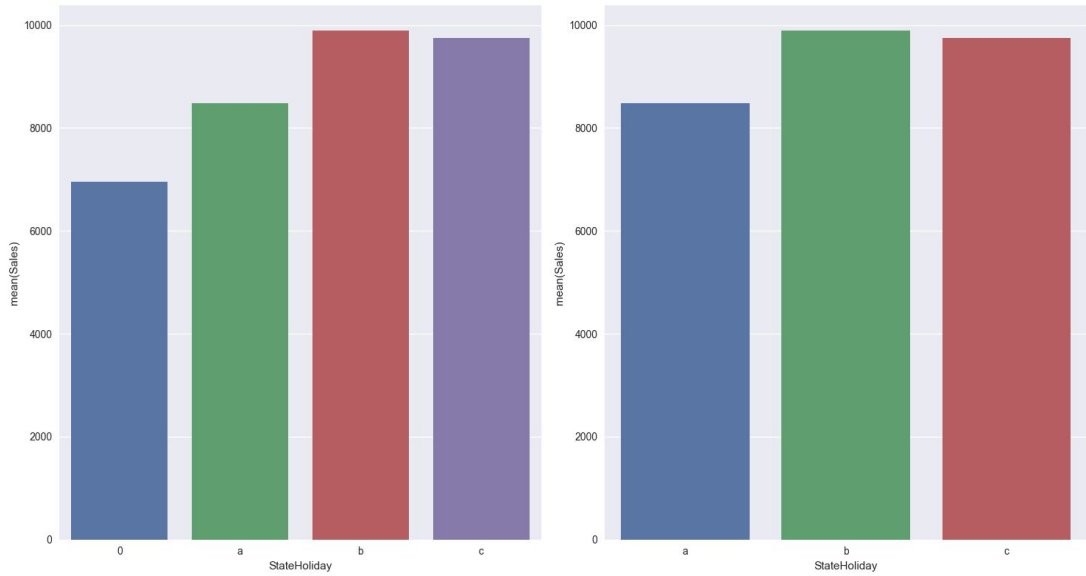


Fig. 8: Average Sales for Open Stores on Normal Days & State Holidays

School Holidays

When we look at the sales averages and average number of customers in stores during school holidays, we see a small increase (~5%), which hints at a likely correlation.

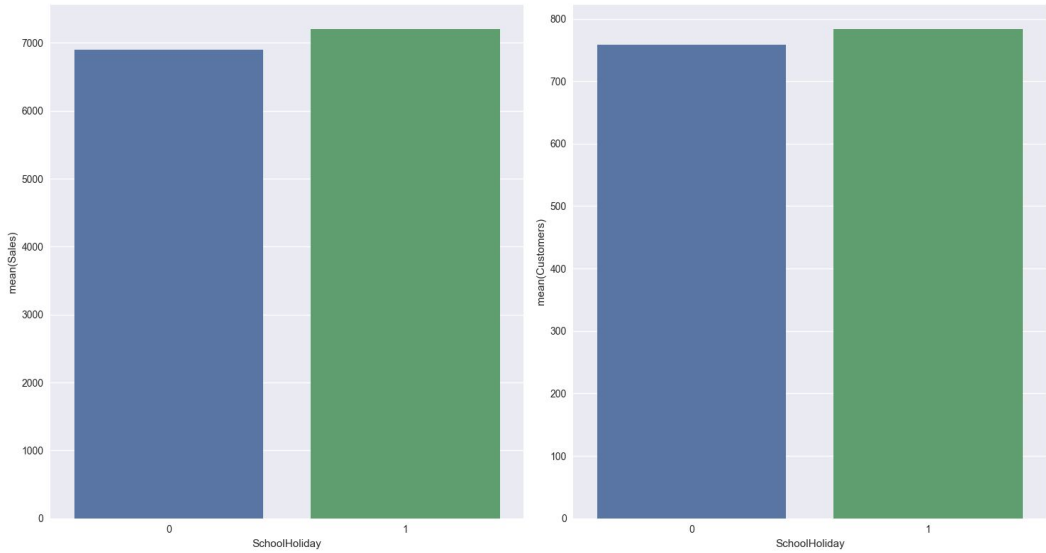


Fig. 9: Average Sales & Average Customers for Open Stores with/without School Holidays

Effect of Competition

When we study the correlation between the competition distance and sales/customers, we see that higher average sales figures are achieved when *CompetitionDistance* equals 0, or in other words, there is no nearby competition.

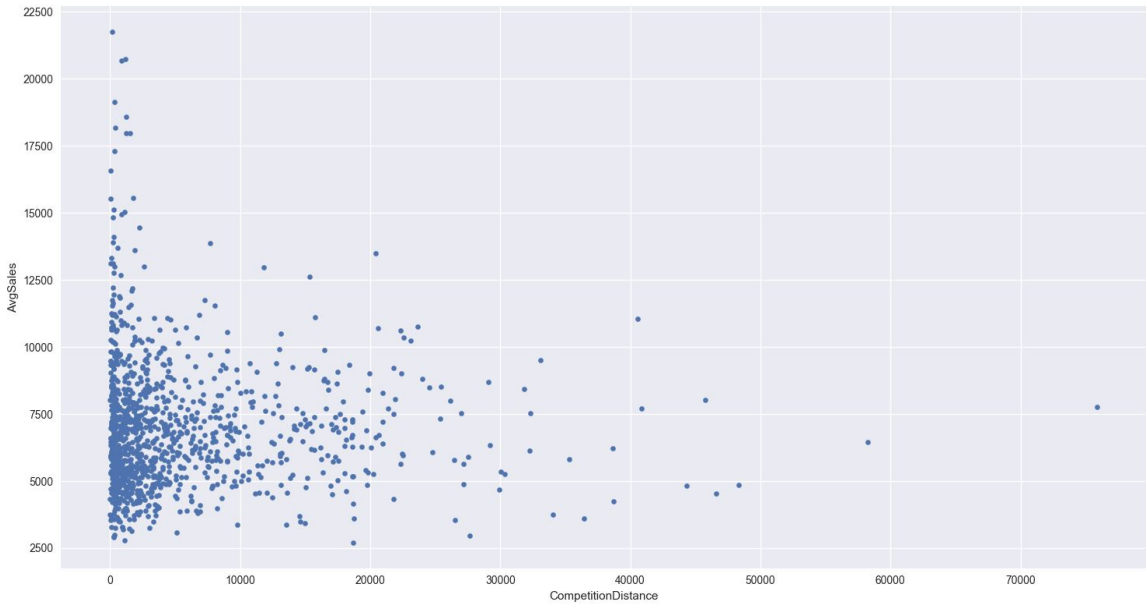


Fig. 10: Competition Distance vs. Average Sales for each Store

We see a similar relationship with the *CompetitionDistance* and average no. of customers plot below.

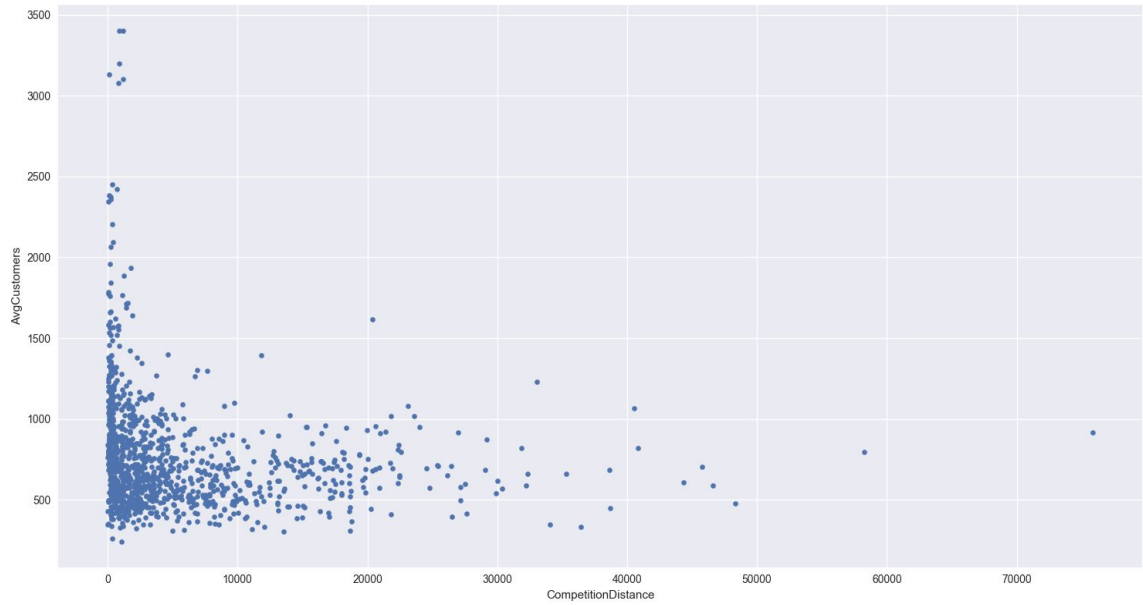


Fig. 11: *CompetitionDistance* vs. Average No. of Customers for each Store

In order to verify the effect of competition on store sales, a plot was generated selecting one particular store (Store 6) and studying its sales before and after competition opened nearby. The results are given in the plot below. The effect is immediately evident, with sales plunging right as the competition opens.

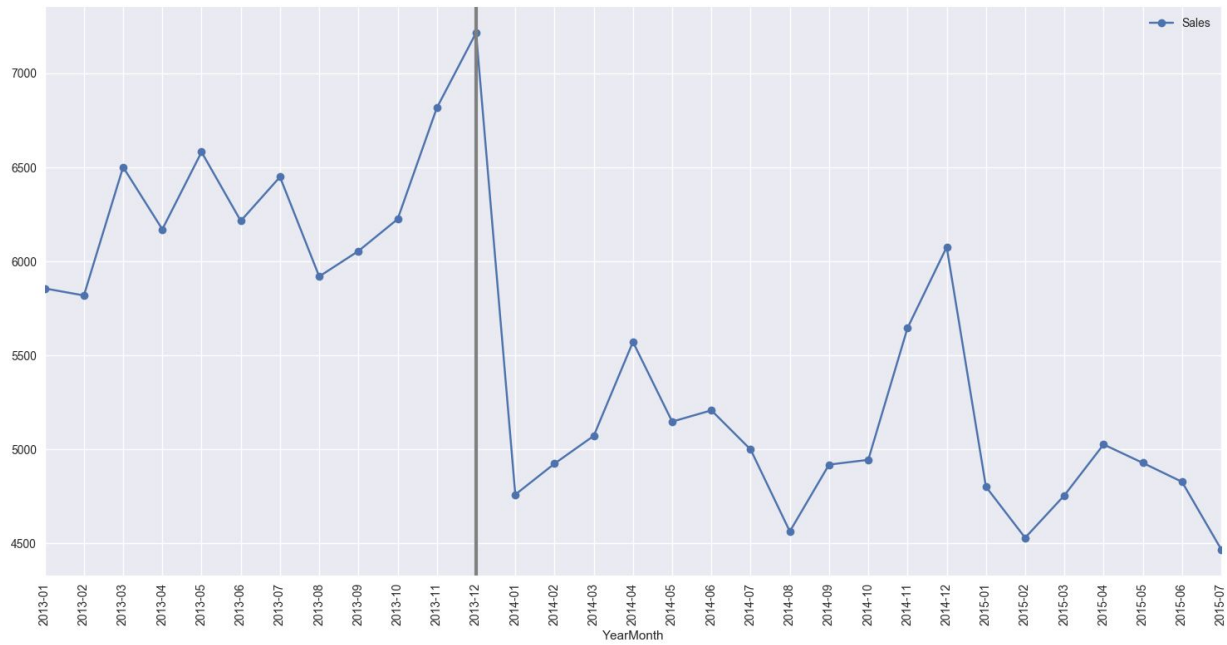


Fig. 12: Effect of Competition on Store 6 by Year-Month

Effect of Store Type & Assortment Type

Store Type

A plot of the average sales and average customers of each store type shows that there is a strong correlation between the store type and average sales.

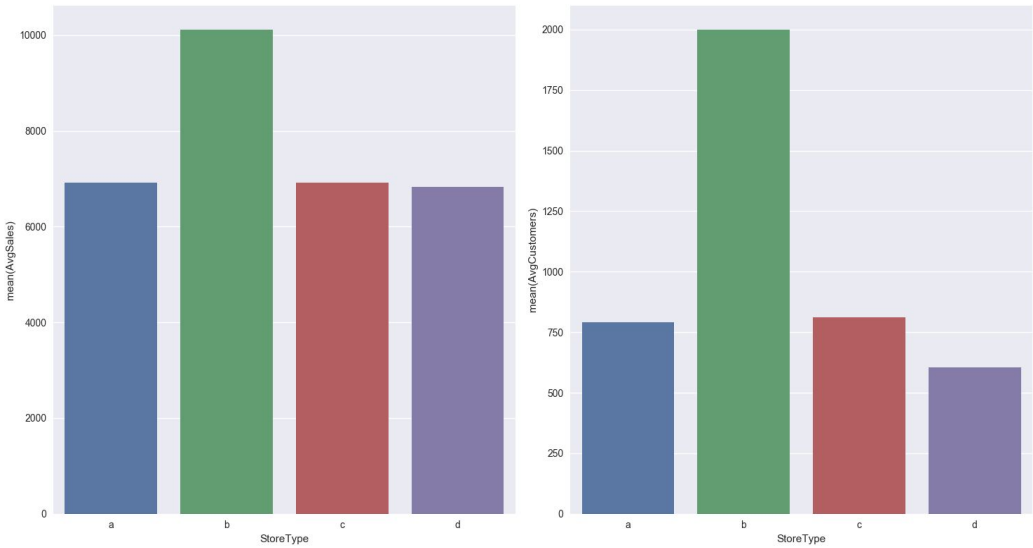


Fig. 13: Average Sales & Customers on Open Days for each Store Type

Assortment Type

A similar correlation is observed in the average sales and average customers for stores with different assortment types.

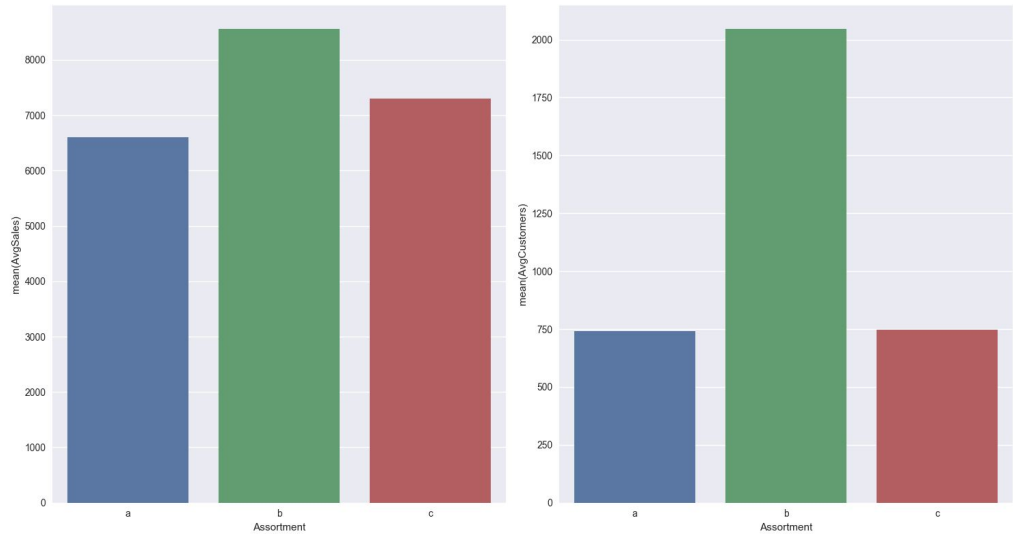


Fig. 14: Average Sales & Customers on Open Days for each Assortment Type

Correlation Matrix of Features

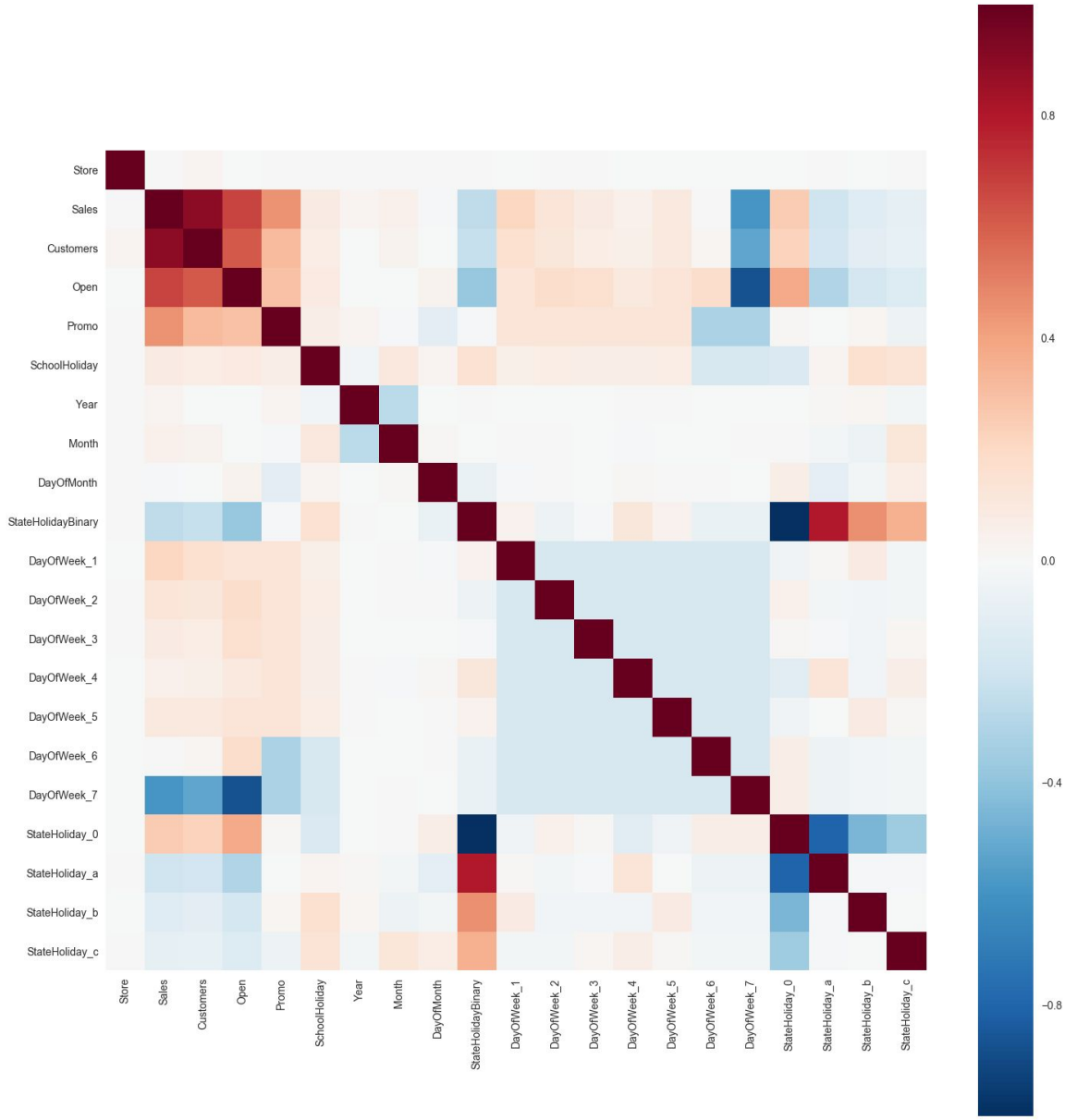


Fig. 15: Correlation Matrix of Features from train.csv

The correlation matrix above shows the correlation between the different features in the training data, with categorical features being one-hot encoded. As can be seen, the sales is strongly affected by any running promotions, the day of the week and holidays along with the obvious correlation between the sales and the no. of customers and whether or not the store is open.

Models

Summary

The models we built were coded in Python using the publicly available scikit-learn and xgboost libraries. Through the course of this project, we created several models using different sets of features and different assumptions. After our initial attempts, we noticed that creating an accurate model for this problem would require a significant amount of effort in feature engineering and data analysis as opposed to the actual implementation of the model.

The first thing we did was set a benchmark for our predictive models using a simple geometric mean of *Sales* values. Our first attempt at a machine learning model was a simple linear regression model. We built several different linear regression models using different sets of features and assumptions. Next, we attempted an ensemble learning model with extreme gradient boosting (XGBoost). We observed that the ensemble learning model performed better than all our other previous models. Our final model combined the results of two versions of XGBoost models using a custom static combiner.

We implemented a validation procedure using the last six weeks of the training data as the test dataset and the remaining data for training. This process allowed us to predict the "future" sales using "historical" data. Also, by assuming the last six weeks of training data for testing, it was possible to simulate a scenario close to the test conducted on Kaggle. We determined the RMSPE score for a few models using this validation procedure.

A list of some of the most significant models we implemented can be found in Appendix A.

Simple Geometric Mean Model

This model was used as a benchmark. It simply calculates the geometric mean value for every [*Store*, *DayOfWeek*, *Promo*] group and assigns that value as the prediction for every [*Store*, *DayOfWeek*, *Promo*] group in the test data.

Feature Selection

Store, *DayOfWeek* & *Promo*

Assumptions

1. The only factors that significantly affect the sales in a particular store are *DayOfWeek* & *Promo* as they have the strongest correlation in the correlation matrix (Fig. 15).

2. The geometric mean value is used instead of the arithmetic mean to normalise the different ranges in the *Sales* values.

Results

RMSPE Score on Kaggle

- Private Score: 0.15996 (Leaderboard: ~2,425th out of 3,303 Submissions)
- Public Score: 0.15390 (Leaderboard: ~2,518th out of 3,303 Submissions)

Linear Regression Model

We selected a simple linear regression model as the first model type to implement as it is the most straightforward algorithm. We attempted the linear regression model using two different approaches. In the first approach, we treated each store as an independent regression problem and trained a model for each store to predict its *Sales* value. In the second approach, we treated the entire dataset as a single regression problem. However, the latter performed quite poorly in comparison to the independent regression problem approach.

Linear Regression Version 2 (linearregression-independent2.py)

Feature Selection

Promo, *DayOfWeek* (one-hot encoded)

Assumptions

- The *Year-Month* feature has no effect on the accuracy as the test data is only for 2015-08 & 2015-09.
- The store's opening/closing dates does not affect the store's performance. For example, a store that was closed yesterday will not get more sales today because of that.
- The competition of each store will affect it consistently, hence, it does not matter.
- Each store's sales value is independent of the other stores and can be treated as independent regression problems.
- State and school holidays don't matter.

Results

RMSPE Score on Kaggle

- Private Score: 0.16405 (Leaderboard: ~2,483rd out of 3,303 Submissions)
- Public Score: 0.14499 (Leaderboard: ~2,413rd out of 3,303 Submissions)

Observations and Inferences

Although this was a machine learning model, it underperformed compared to the Simple Geometric Mean in the private leaderboard. However, the public leaderboard score was slightly better. The poor performance is probably due to the minimal set of features used.

Linear Regression Model Version 4 (linearregression-independent4.py)

Feature Selection

Promo, *SchoolHoliday*, *Year*, *Month*, *DayOfWeek* (one-hot encoded), *StateHoliday* (one-hot encoded), *AvgCustStore* (Average Customers for Store #), *AvgCustStoreMonth* ((Average Customers for Store # by Month)

Assumptions

- The store's opening/closing dates does not affect the store's performance. For example, a store that was closed yesterday will not get more sales today because of that.
- The competition of each store will affect it consistently, hence, it does not matter.
- Each store's sales value is independent of the other stores and can be treated as independent regression problems.

Results

RMSPE Score on Kaggle

- Without Logarithmic Standardization of *Sales*
    - Private Score: 0.16209 (Leaderboard: ~2,456th out of 3,303 Submissions)
    - Public Score: 0.14587 (Leaderboard: ~2,419th out of 3,303 Submissions)
- With Logarithmic Standardization of *Sales*
    - Private Score: 0.15522 (Leaderboard: ~2,368th out of 3,303 Submissions)
    - Public Score: 0.13742 (Leaderboard: ~2,178th out of 3,303 Submissions)

Observations and Inferences

In this model, we made use of features such as *AvgCustStore* & *AvgCustStoreMonth* which improved the score compared to the previous model, however, the RMSPE score was still fairly poor. An analysis we made at this point revealed that rather than training the model with *Sales*, it is more beneficial to train the model against log(*Sales*). Since the *Sales* distribution is a Poisson distribution, log(*Sales*) standardizes the data and improves the performance of the model significantly.

The linear regression models, although machine learning algorithms, were underperforming as compared to the geometric mean. Upon doing further research, we observed that in several cases an ensemble learning model works very well for a regression problem.

XGBRegressor Model

At first, we sought to use scikit-learn's ElasticNet that combines two linear regression models. However, since the linear regression model underperformed, we wanted to try a tree based model. We found that the extreme gradient boosting (XGBoost) model is one of the more common boosting algorithms used for machine learning. XGBoost is a boosting ensemble algorithm commonly used for supervised learning problems. XGBoost makes use of decision trees for classification and regression problems. When the problem is a regression problem, the decision tree is called a regression tree. The main component of XGBoost is its tree ensembles which sum up the predictions of multiple trees. The boosting procedure in XGBoost is an additive process where a trained model is added to the prediction at each step. The optimization problem is solved to select which tree is added at each step. In our implementation, we used the mean square error as the loss function for optimization.

Our first attempt at the XGBRegressor model was using the independent regression problem strategy described previously. However, this model underperformed similar to the linear regression model. Next, we tried to train the model against the whole training dataset. This model performed much better than our previous models. We also performed local validation for this model. In the validation process, we selected the last 6 weeks of the training data as test data and computed the RMSPE.

XGBRegressor Version 2 (xgboostregressor-log.py)

Feature Selection

Store, *DayOfWeek*, *Year*, *Month*, *DayOfMonth*, *Open*, *Promo*, *StateHoliday*, *SchoolHoliday*, *StoreType*, *Assortment*, *CompetitionDistance*, *Promo2*

Assumptions

- DayOfMonth has an effect on sales, for example, the sales is higher on paydays.
- The store's opening/closing dates does not affect the store's performance. For example, a store that was closed yesterday will not get more sales today because of that.

Results

RMSPE Score on Kaggle

- Private Score: 0.12728 (Leaderboard: ~1,432nd out of 3,303 Submissions)
- Public Score: 0.10754 (Leaderboard: ~1,141st out of 3,303 Submissions)
- Local Score: 0.12857

Observations and Inferences

The XGBoost based model performed much better compared to all the linear regression models. We observed a significant decrease in the RMSPE scores. We inferred that either the tree model for the given problem performs much better or the ensemble learning approach is causing the model to perform better. In order to validate the first inference, we tried an ensemble of Random Forests Regression models which performed poorly, thus we concluded that approaching this problem using a tree model will lead to the best results

XGBRegressor Version 5 (xgboostregressor-log5.py)

Feature Selection

Store, *DayOfMonth*, *Week*, *Month*, *Year*, *DayOfYear*, *DayOfWeek*, *Open*, *Promo*, *SchoolHoliday*, *StateHoliday*, *StoreType*, *Assortment*, *CompetitionDistance*, *Competition-OpenYearMonthInteger*, *AvgSales*, *AvgCustomers*, *AvgSalesPerCustomer*

Assumptions

- DayOfMonth has an effect on sales, for example, the sales is higher on paydays.
- The store's opening/closing dates does not affect the store's performance. For example, a store that was closed yesterday will not get more sales today because of that.
- *Sales* has a correlation with Average Sales per Customer.
- *Sales* is affected by *CompetitionOpenSince[X]*. Thus, a new feature merging the year and month of the opening date is created, *CompetitionOpenYearMonthInteger*.

Results

RMSPE Score on Kaggle

- Private Score: 0.12305 (Leaderboard: ~1,219th out of 3,303 Submissions)
- Public Score: 0.11276 (Leaderboard: ~1,362nd out of 3,303 Submissions)
- Local Score: 0.11976

Observations and Inferences

This particular XGBoost model used a set of features that improved the performance of the model only on the private leaderboard. However, on the public leaderboard we see a slight decline in performance. Since the private leaderboard tests against 70% of the test data, we can safely assume that this model performed better than XGBRegressor Version 2.

Static Combiner Model (xgboostensemble.py)

After building several models with the XGBoost library, we obtained a very good RMSPE score but it had plateaued. At this point, we turned to the ensemble learning approach again to further improve the RMSPE score. This time, we attempted a custom built static combiner that combines the predictions of two XGBoost models using a weighted average (i.e. *y_pred = y_pred1 * w1 + y_pred2 * w2*).

Our initial attempt was to use weights of 0.5 and 0.5 and combine the results from XGBRegressor Version 2 and XGBRegressor Version 5. Local validation using this ratio gave us a bad RMSPE score compared to the individual models. We wrote a script to attempt different possible weights and determine the weights that gave the minimum RMSPE score. An initial run gave us a set of weights to use. We observed at this point, that the weighted averaged predictions were poor because of the lack of precision in the weights. To obtain more precision, we modified the equation above to incorporate a correction factor i.e. *y _pred = (y_pred1 * w + y_pred2 * (1-w)) * correction_factor*.

The above equation was computed for *w* values ranging from 0.0 to 1.0 in steps of 0.05 and *correction_factor* from 0.9 to 1.0 in steps of 0.005. The best RMSPE score was obtained for *w* = 0.75 and *correction_factor* = 0.99.

Once we obtained the correct *w* and *correction_factor* values using the local RMSPE score for the last six weeks of the training data, we tweaked the values manually to get the best possible score on Kaggle as converging the values for *w* & *correction_score* using the training data for several iterations would be too computationally expensive and would take days on our personal laptops (explained further in Observations and Inferences).

Results

RMSPE Score on Kaggle

- With manual correction to weights:
  - Private Score: 0.11880 (Leaderboard: ~414th out of 3,303 Submissions)
  - Public Score: 0.10640 (Leaderboard: ~1,055th out of 3,303 Submissions)
- With weights obtained using local RMSPE:
  - Private Score: 0.12057 (Leaderboard: ~922nd out of 3,303 Submissions)
  - Public Score: 0.10934 (Leaderboard: ~1,229th out of 3,303 Submissions)

Observations and Inferences

Although this model gave the best result on Kaggle, the approach used to determine the weights may not be correct. Since we only used the last six weeks of training data for

validation, the weights determined using the script are overfitted for just the last six weeks. In practice, it might be better to use a backward sliding window of 6 weeks from the last day of the training set to create different test sets for validation. This will provide more scope for the true weights to converge and prevent overfitting. However, this process involves re-training the two XGBoost models that are being combined for every 6 week test period and then converging the values of *w* and *correction_factor* over several iterations, which would take too long on the computation power of our personal laptops.

Kaggle Hacks

Fitting the Model for Test Data

There are two very interesting observations about the test data. The first is the fact that there are 259 stores missing from the test data, which suggests that excluding these stores from training as well would increase the accuracy of the predictions. Additionally, since the test data is only for a period of six weeks, the training data can be tweaked to create an extremely specific model for this duration. However, we chose to ignore both of these things in an attempt to create a more general model.

Using External Datasets

We found from the discussion forum of the Kaggle competition that an easy way to get a very high leaderboard ranking is to use external datasets such as Google's Daily Search Trends for the keyword "Rossmann", weather data per state and state data. By using just these datasets against sales, it is possible to get a leaderboard rank of 5 with a simple ensemble model. However, we chose to ignore this method as it goes against the motivation of building a generalized model in the constraints of the competition.

Manual Adjustments of Weights in the Static Combiner Model

One method to improve our own score on Kaggle was to adjust the weights by submitting multiple times on Kaggle itself and using the RMSPE values to converge the weights. However, this process goes against the purpose of the model as it essentially treats the test data as a training dataset for adjusting the weights. However, by tweaking the weights enough, it was possible for us to achieve a best score of 0.11381 on the private leaderboard and 0.10462 on the public leaderboard. However, since this was a "hack", we have not submitted this as a model.

Conclusions

The Rossmann Store Sales problem is a very interesting data science problem to solve. We observed that the problem is more focused on feature engineering and feature selection than on model selection. We spent around 70% of our time analyzing the data for trends in order to make feature selection easier. We attempted several models using different features and assumptions. We concluded that ensemble learning improves the prediction accuracy considerably. We learnt the benefits of boosted trees and their applications in machine learning. However, complicated ensemble learning approaches may not be practically applicable in many scenarios as it tends to overfit the training data.

In the future, we want to implement the backward sliding window of six weeks to converge the values of the weights & *correction_factor* in our ensemble of boosted trees as opposed to the current single validation set. Since the data is based on time, simply using random subsampling will not be the best approach as future data needs to be predicted from historical data and not just a random selection of training instances. We also look to improve the existing models, experimenting with different sets of features.

References

1. CZ 4041 Machine Learning, Nanyang Technological University
2. Kaggle Competition - https://www.kaggle.com/c/rossmann-store-sales
3. Data Visualization - https://seaborn.pydata.org/
4. Data Visualization - https://matplotlib.org/
5. Linear Regression - https://explorable.com/linear-regression-analysis
6. scikit-learn Linear Models - http://scikit-learn.org/stable/modules/linear_model.html
7. scikit-learn Linear Regression - http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
8. scikit-learn Cross Validation - http://scikit-learn.org/stable/modules/cross_validation.html
9. scikit-learn Evaluation - http://scikit-learn.org/stable/modules/model_evaluation.html
10. Boosted Trees - https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf
11. XGBoost Model - http://xgboost.readthedocs.io/en/latest/model.html

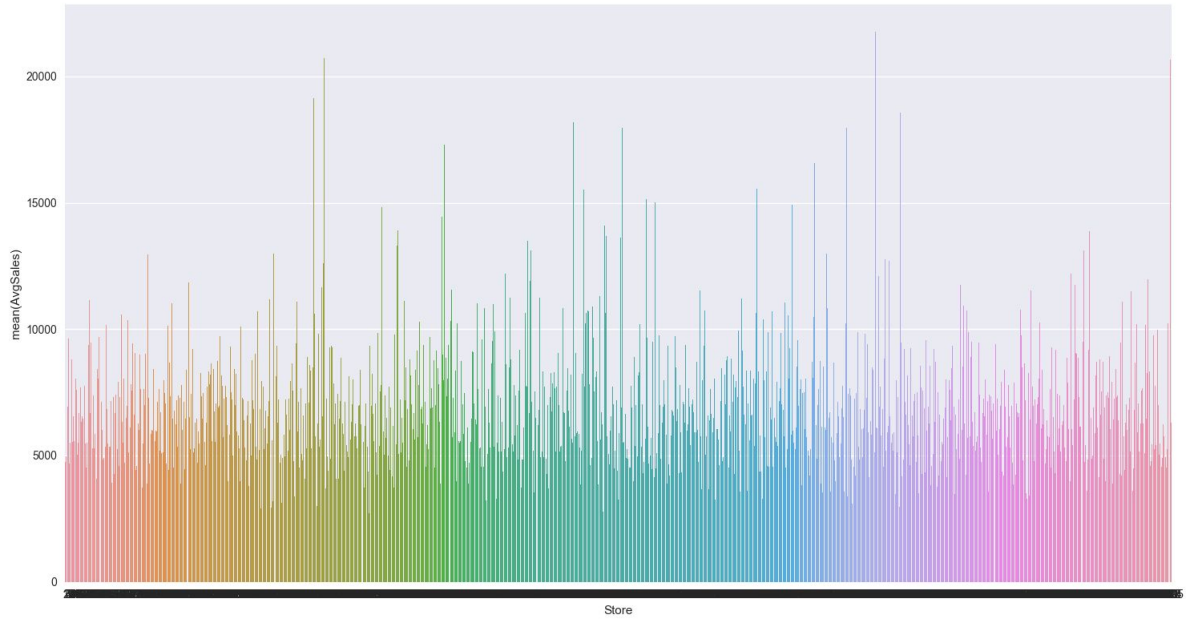
Appendices

Appendix A - List of Models

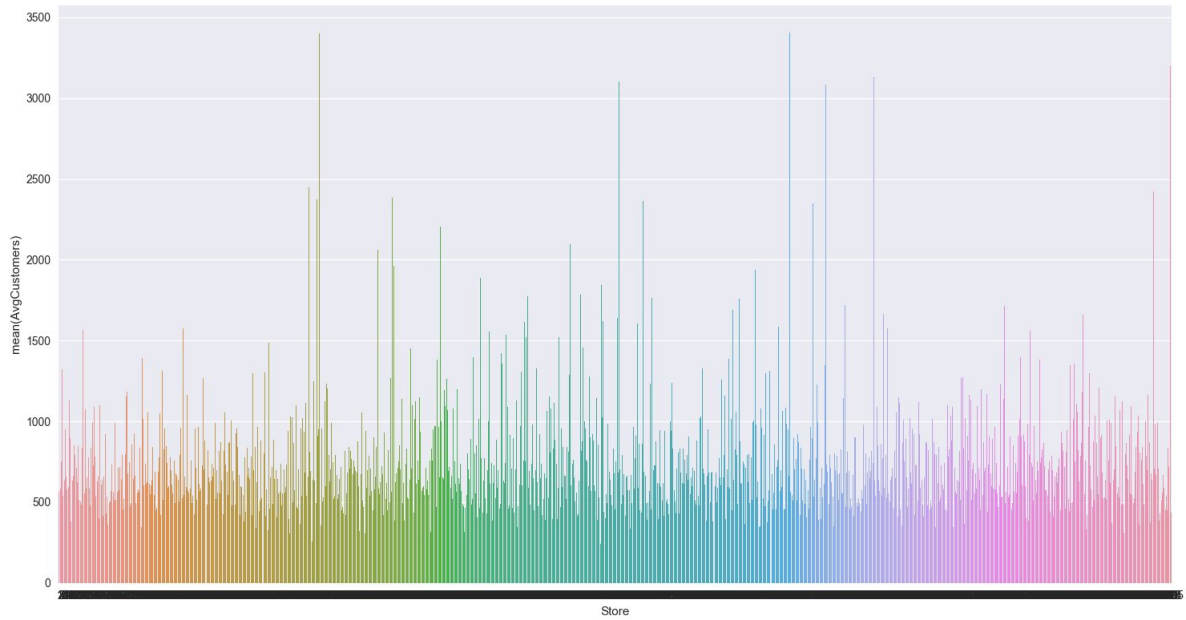
S/N	Name	Private Score	Public Score	Features
1	Simple Geometric Mean	0.15996	0.1539	Store, DayOfWeek, Promo
2	Simple Median	0.14598	0.14001	Store, DayOfWeek, Promo
3	Linear Regression - Independent 2	0.16405	0.14499	Promo, DayOfWeek (one-hot encoded)
4	Linear Regression - Independent 4	0.16209	0.14587	Promo, SchoolHoliday, Year, Month, DayOfWeek (one-hot encoded), StateHoliday (one-hot encoded), AvgCustStore, AvgCustStoreMonth
5	Linear Regression - Independent - log	0.15522	0.13742	Promo, SchoolHoliday, Year, Month, DayOfWeek (one-hot encoded), StateHoliday (one-hot encoded), AvgCustStore, AvgCustStoreMonth
6	XGBoost Regressor - log	0.12728	0.10754	Store, DayOfWeek, Year, Month, DayOfMonth, Open, Promo, StateHoliday, SchoolHoliday, StoreType, Assortment, CompetitionDistance, Promo2
7	XGBoost Regressor - log 5	0.12305	0.11276	Store, DayOfMonth, Week, Month, Year, DayOfYear, DayOfWeek, Open, Promo, SchoolHoliday, StateHoliday, StoreType, Assortment, CompetitionDistance, CompetitionOpenYearMonthInteger, AvgSales, AvgCustomers, AvgSalesPerCustomer
8	XGBoost Ensemble	0.1188	0.10640	xgboostregressor-log5 Predictions, xgboostregressor-log Predictions

Appendix B - Additional Charts

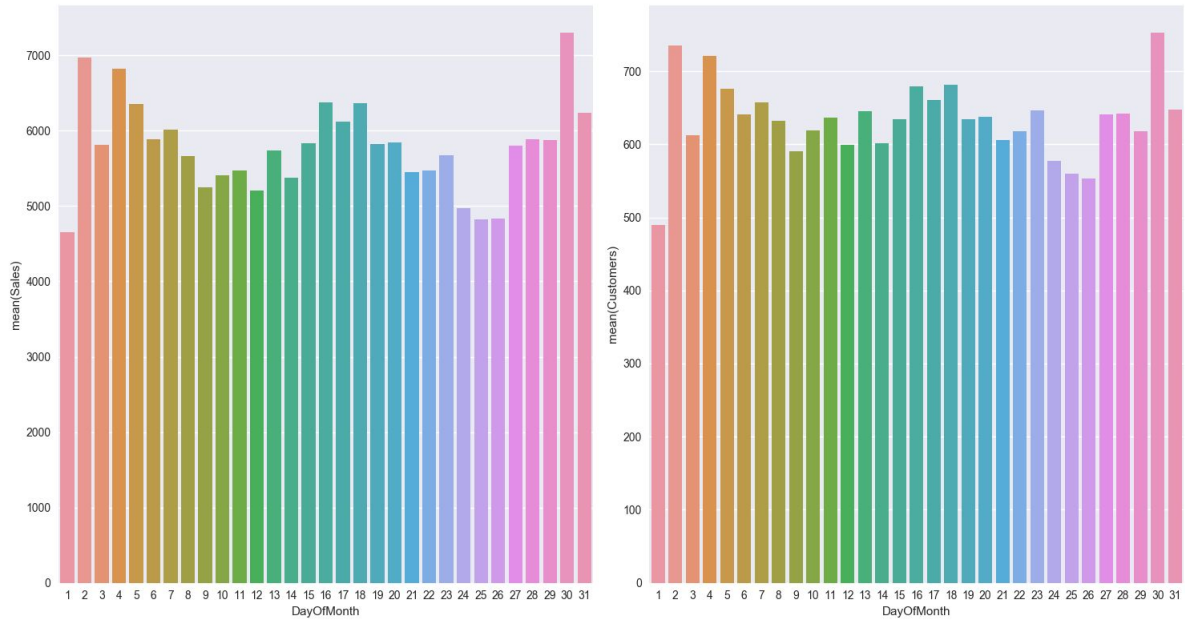
Average Sales for each Store:



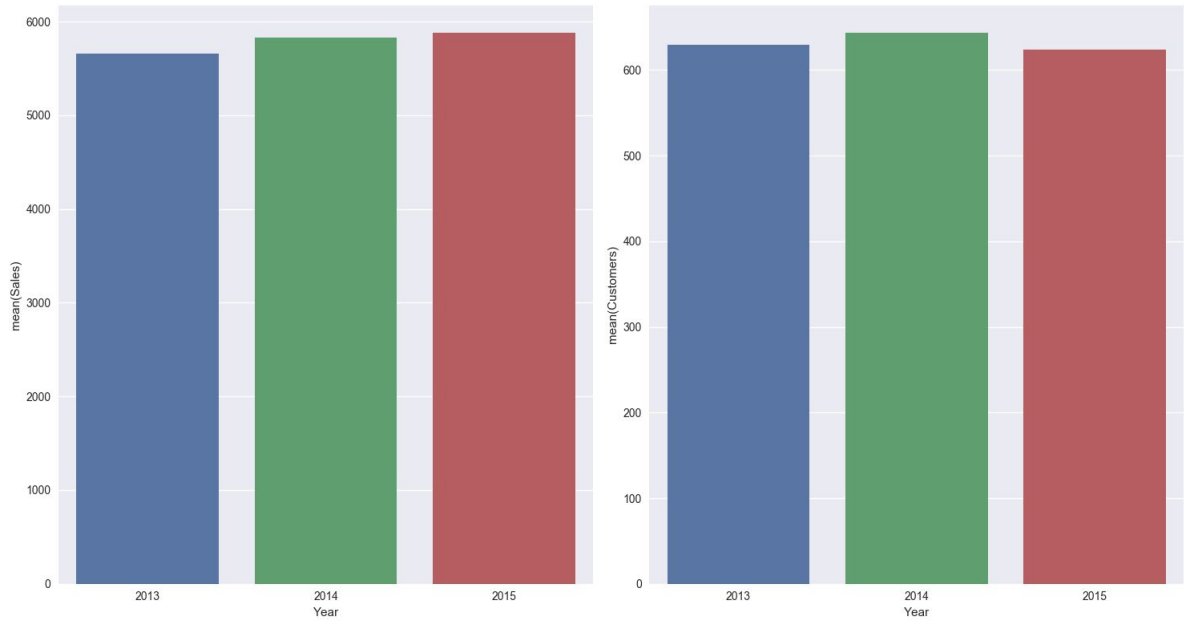
Average Customers for each Store:



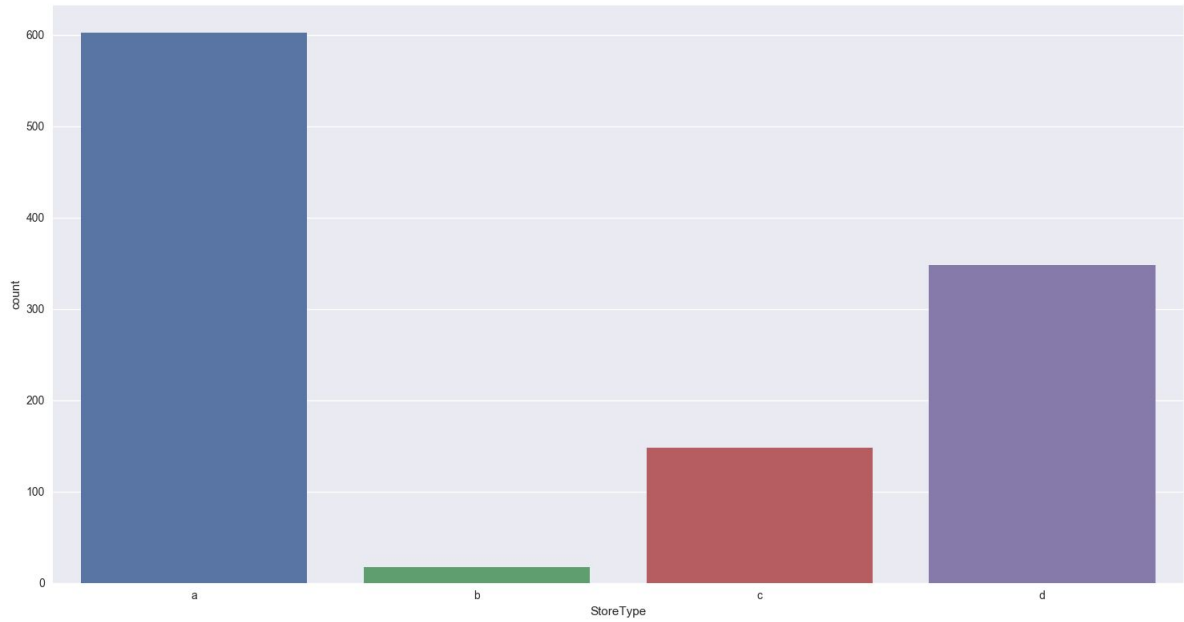
Average Sales & Average Customers by Day of Month:



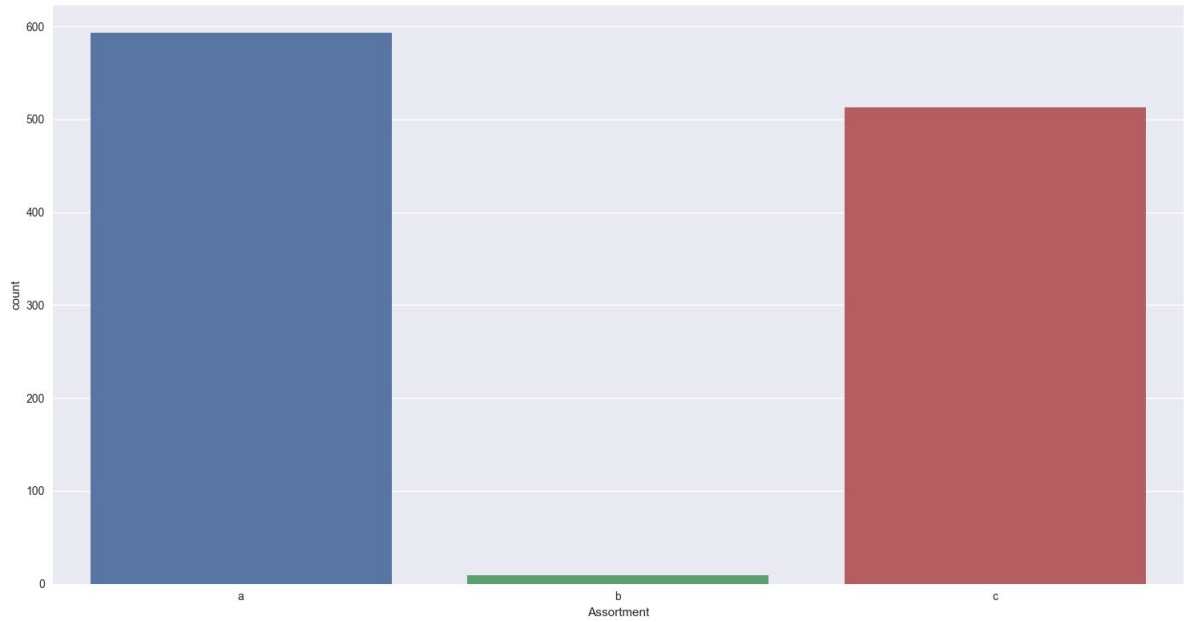
Average Sales & Average Customers by Year:



No. of Stores (by Store Type):



No. of Stores (by Assortment):



No. of Stores Closed/Open (by Day of Week):