CZ 4032 – Data Analytics & Mining

Forecasting Sales for Chain Stores

GROUP 7

Suyash Lakhotia (U1423096J) [20%]

Nikhil Venkatesh (U1423078G) [20%]

Virat Krishan Chopra (U1422252H) [20%]

Priyanshu Singh Kumar (U1422744C) [20%]

Shantanu Kamath (U1422577F) [20%]

Table of Contents

1. Abstract

This report documents our approaches in tackling the problem of predicting sales for chain stores based on historical data. In particular, we present our solutions to the Rossmann Store Sales problem from Kaggle. We used several data analysis techniques to understand the given datasets and chose several machine learning algorithms to predict the sales. We used the Knowledge Discovery in Databases (KDD) approach to find trends in the dataset and implemented commonly-used machine learning techniques to build several models in order to make the sales predictions. Full-resolution plots can be found in the submitted .zip file, which also contains all of our source code written in Python 3.

2. Problem Description

2.1 Motivation & Problem Inspiration

Predicting sales performance effectively and efficiently is a significant problem that many retail outlets face today. In order to make operations predictable while simultaneously providing customers with the best possible experience, retailers often forecast sales on a weekly/monthly basis based on a variety of metrics such as historical sales, promotions, competition presence, seasonality, geographic presence of stores etc. [1]

With the proliferation of modern technology, especially in the field of data mining and machine learning, retail stores are recording large amounts of data that can be leveraged upon to extract and analyse insights and identify trends for predicting sales. We can achieve this using new age data mining and machine learning techniques that bring out important details often overlooked by human evaluators.

2.2 Problem Definition & Scope

Since the generalized problem of *predicting sales for chain stores* is vast, we have decided to tackle a specific instance of the problem — *predicting 6 weeks of daily sales for 1,115 drug stores operated by Rossmann Stores across Germany*. The approaches used in this project can be applied to similar retail chains across various industries and thereby help tackle the bigger sales prediction problem that we have identified.

2.3 Related Work in Existing Literature

A large number of studies using advanced data mining techniques exist around the problem of predicting retail sales numbers. With respect to the particular instance of Rossmann Stores, the winner of the Kaggle competition [2] identified the importance of time-series data and found temporal dependencies within the dataset. The approach used to preprocess the data involved the calculation of averages over various time windows, post which these averages were split using important features like day of week, month, year and relative time indicators that incorporate seasonality. The learning algorithm used to forecast the sales was extreme gradient boosting.

Another important study [3] used autoregressive correlation graphs that identified a 7-day seasonality on the time-series data and hence determined the most important features to be d7, d14, d21 etc. corresponding to the sales 7 days ago, 14 days ago and 21 days ago respectively. The model used to forecast sales was based on decision trees and linear regression and learned using the AdaBoost algorithm.

3. Approach

3.1 Fundamental Methodology

3.1.1 Data Selection

The dataset for this problem was obtained from Kaggle and contains two sets of training data (`train.csv` and `store.csv`) and one set of test data (`test.csv`). `train.csv` contains data about the sales figures for a particular store on a particular date while `store.csv` contains additional factual information about each store. `test.csv` holds the data-points for which the sales need to be predicted, identified by a combination of the store number and date.

3.1.2 Data Cleaning and Preprocessing

Once the datasets were identified and understood, we proceeded with data cleaning and preprocessing, which involved removing outliers from the data, making sure the data was stored in a consistent format and handling missing data values.

3.1.3 Data Transformation

During this step, the data was transformed into forms relevant for data mining using one-hot encoding for the categorical features such as `DayOfWeek`, `StateHoliday` etc. and creating additional features by combining or splitting existing features in the dataset.

3.1.4 Data Mining

This step involved using intelligent machine learning models like linear regression and boosted decision trees to predict the store sales on selected subsets of features (identified using conclusions from our analysis and experimentation).

3.1.5 Experimentation & Evaluation

The final step in the knowledge discovery process involved establishing comparison schemes for the various feature sets & models identified by measuring their performance on the test set. Evaluation within this project is based on the Kaggle private score, which is the root mean squared percentage error (RMSPE, given below) on ~70% of the test dataset.

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

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.

3.2 Approach

We identified that the sales data for a particular date is highly correlated with the corresponding factual and temporal data. Thereby, the sales for the stores can be predicted using the daily sales numbers for the preceding time interval or using factual data surrounding those numbers such as day of the week, state holiday, promotions etc. Existing literature uses time-series data analysis, which relies on rolling prediction methodology, i.e., it uses predictions for one time window as the truth values for predicting the future sales of the next time window. This could easily result in a snowball effect as the error may accumulate as we continue predicting into the future. We correct for this by using a factual approach that does not depend on the model's own predictions for the time window but rather on discrete characteristics of the dataset.

3.3 Algorithms

3.3.1 Benchmark Models

We employed two simple models to obtain the benchmark for result comparisons. The first model, Simple Geometric Mean, simply calculates the geometric mean value for every `(Store, DayOfWeek, Promo)` combination and assigns that value as the prediction for the same combination in the test dataset. The second benchmark model, Simple Median, calculates the median value instead of the geometric mean for the same combination of features. These models provide us with the baseline metric to evaluate the results of the more complex models we implemented.

3.3.2 Continuous Time-Series Model

Since the provided data is in the form of a time series, we utilized recurrent neural networks (RNNs) to build a time series model. RNNs have an internal state that incorporates contextual information by including the past historical data for a predefined time window [5]. That is, the connections in a RNN form a directed cycle such that the decision at the current time step is influenced by the decision at the previous time step and the current input data. This property of RNNs make them ideal for largely unsegmented temporally connected data. RNNs can be trained using the Backpropagation Through Time (BPTT) algorithm which is an extension of the backpropagation algorithm. We employed a particular type of RNN known as Long Short Term Memory (LSTM) for our analysis.

3.3.3 Discrete Attribute-Based Models

3.3.3.1 Linear Regression

Since the output class of the problem is in a continuous domain, we decided to tackle it as a regression. Our first attempt was to use a linear regression model. In a linear regression model, the output variable is predicted as a linear combination of weighted input variables.

3.3.3.2 Random Forest Regression

A random forest is an ensemble model that works by using multiple classifying decision trees on various subsets of the dataset and then classifies by outputting the mean of the results from all decision trees. This helps in reducing overfitting and enhances accuracy.

3.3.3.3 Ridge Regression

Within datasets that have inherent multicollinearity, ridge regression is often employed to analyse the data. Through the addition of a specific degree of bias, standard errors within estimations are significantly reduced thereby giving highly reliable predictions.

3.3.3.4 XGBRegressor Model

XGBoost is a highly efficient and flexible extreme gradient boosting library that implements parallel decision tree boosting to reduce computation time and effectively allocate memory resources. The predictions from multiple trees are summed up within the tree ensembles. After each predictive iteration, a trained model is added to optimize the learning algorithm.

4. Implementation

4.1 Understanding the Dataset

4.1.1 train.csv

This file contains the training dataset which describes the sales figures for a store on a particular date. The data fields are described below:

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)

Number of Records: 1,017,209

4.1.2 store.csv

This file contains describes additional information about each of the store. The data fields are described below:

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")

Number of Records: 1,115

4.1.3 test.csv

The dataset provided within this file is used for testing and evaluating the implemented learning models. The data fields are the 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.

4.2 Preprocessing the Dataset

4.2.1 Data Cleaning

This involved replacing missing values and correcting for misrepresented data using the following strategies:

1. Ensuring consistent data formats for attributes: For example, the `StateHoliday` attribute contains both numerical and string representations of numbers. In order to make it consistent, all the values were changed to the string representation.
2. Substituting NaN values: NaN values were replaced for the following attributes:
    a. `CompetitionOpenSince[X]`: If `CompetitionOpenSince[X]` is missing but `CompetitonDistance` is not equal to 0, then ‘1900’ was inserted for `CompetitionOpenSinceYear` and ‘01’ for `CompetitionOpenSinceMonth`.
    b. `CompetitionDistance`: If `CompetitionOpenSince[X]` is `NaN` (not a number) `CompetitionDistance` is set to 0.
    c. `Open`: If the `DayofWeek` attribute was anything except Sunday, then the missing `Open` data field will be replaced with ‘1’ (i.e. open).

4.2.2 One-Hot Encoding

As machine learning models tend to look for patterns within data, we performed one-hot encoding on categorical features such as `DayOfWeek, StateHoliday, StoreType and Assortment` as they don’t have an inherent order or numerical significance.

4.2.3 Other Features

- `DayOfMonth, Year, Month, YearMonth & WeekOfYear`: The date field was broken down into these fields to provide a richer insight into date-related sales trends.
- `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.
- `AvgCustStore, AvgCustStoreMonth, AvgCustStoreYear:` Stores the average no. of customers per store (overall), per store for a particular month and per store for a particular year respectively.

- `StateHolidayBinary:` In addition to categorizing `StateHoliday` by type, an additional column was added to simply indicate whether it was a state holiday or not.
- `IsPromoMonth:` Indicates if the record is in the month of a running store promotion.
- `CompetitionOpenYearMonthInteger:` Stores the “YYYYMM” version of `CompetitionOpenSinceYear` and `CompetitionOpenSinceMonth`.

5. Feature Analysis and Selection

5.1 Frequency Distributions

We can infer a Poisson-like distribution for the sales & customers values on open days from Figure 1 & Figure 2. The sales data has a peak frequency between 5,000 and 7,500 and rarely falls below 2,500 or breaks through the 12,500 mark. The average sales value obtained is 6,955.51. Similarly, the number of customers commonly falls between 500 and 1,000. Values above 1,000 are comparatively unlikely while the average number of customers for open stores is 762.73.

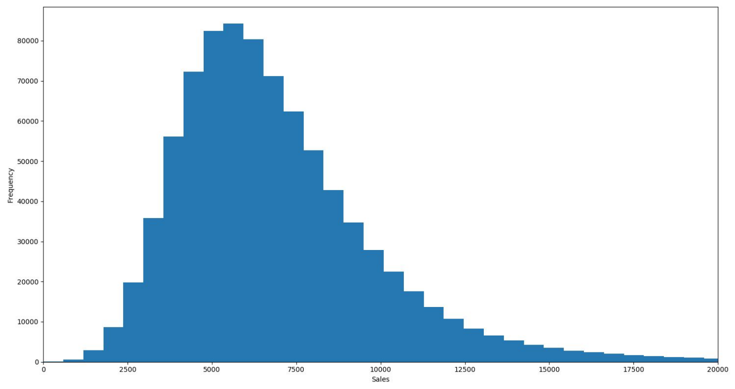

Figure 1: Frequency of Sales Values for Open Stores

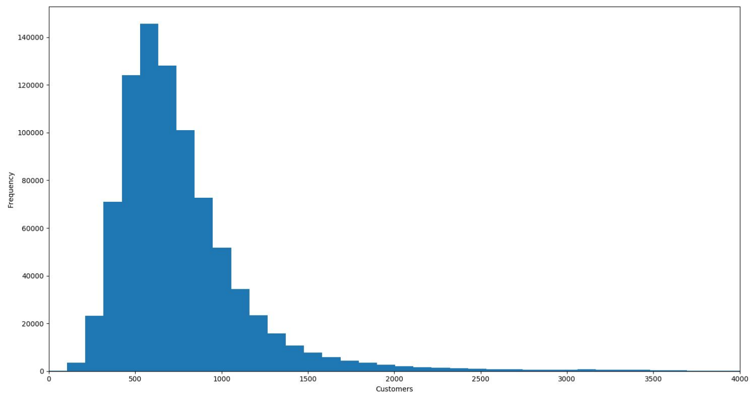

Figure 2: Frequency of Number of Customers for Open Stores

5.2 Trend Analysis for Date-Related Fields

5.2.1 Weekly Trends

According to Figure 3 below, we identified three peaks within the weekly sales and customer data. The two peaks on Mondays and Fridays possibly stem from them being the start and end of the work week respectively whereas the biggest peak on Sunday is possibly due to other competitors being closed. A useful insight comes from the disproportionality between the average number of customers and average sales on Sundays, which implies that a large amount of footfall comes from window shoppers on Sundays.

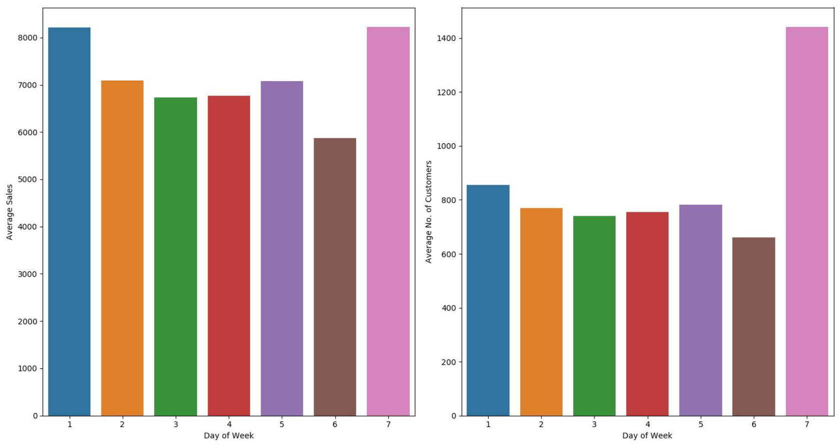

Figure 3: Average Sales & No. of Customers per Day of Week for Open Stores

5.2.2 Annual Trends

From Figure 4 below, we can observe a clear sales boost in the months of June and July due to the summer holidays and another increase in December due to festivities.

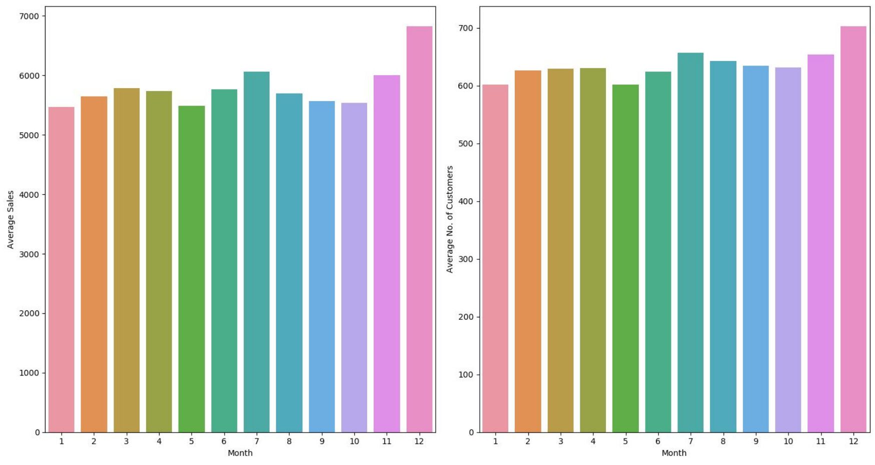

Figure 4: Average Sales & No. of Customers per Month

Figure 5 below represents the average sales data for each month throughout the domain of the training dataset (Jan 2013 - July 2015). We can draw the conclusion that there is a significant year-on-year growth from 2013 to 2015 with similar monthly trends every year.

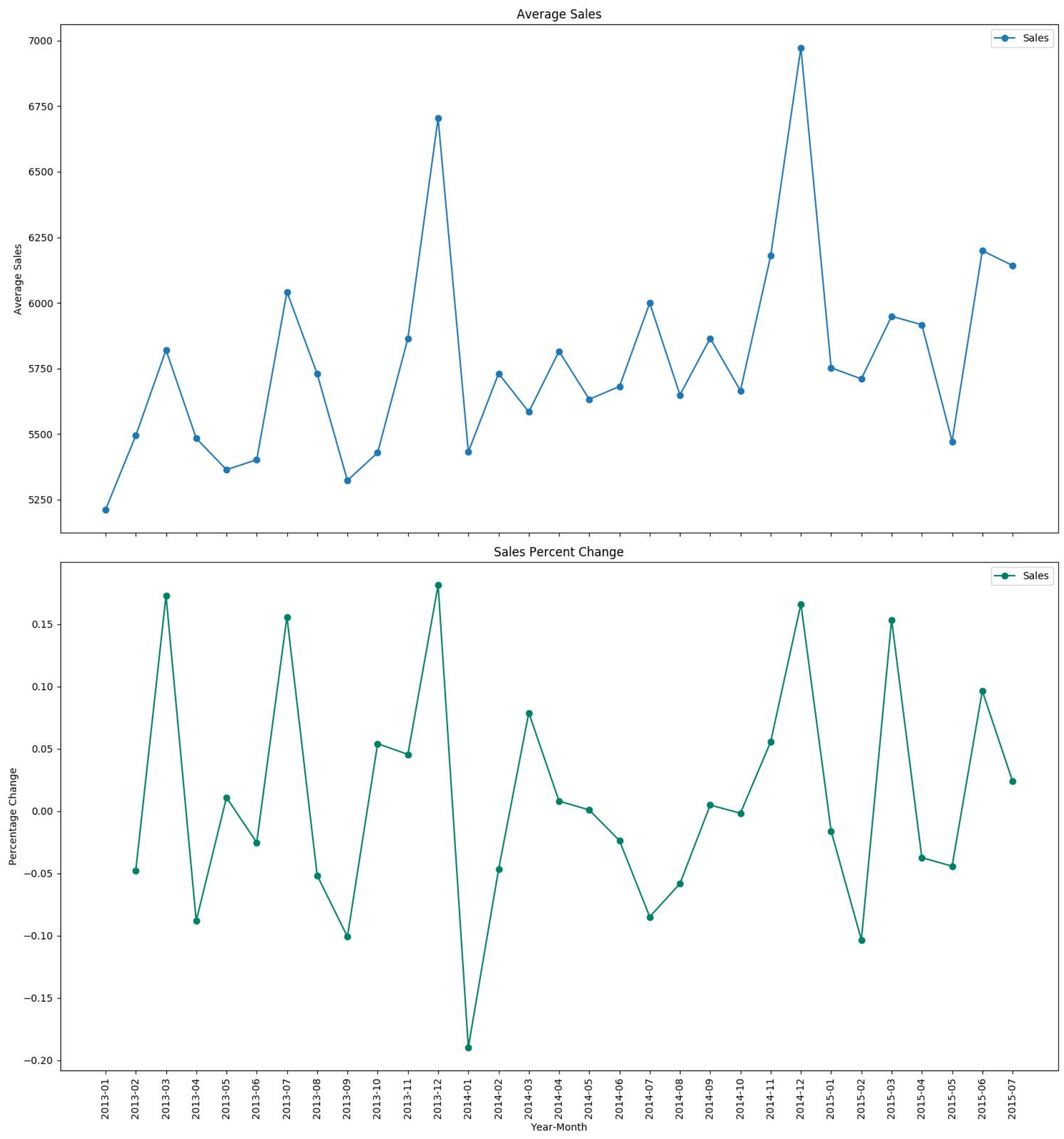

Figure 5: Sales & Percentage Change per Year-Month

5.3 Effect of Promos and Holidays

5.3.1 Promos

Figure 6 below shows the average number of customers and average sales numbers across the dataset on days with (orange) and without (blue) promotions. It can be observed that a promotion strongly correlates with both high average sales and high customer footfall.

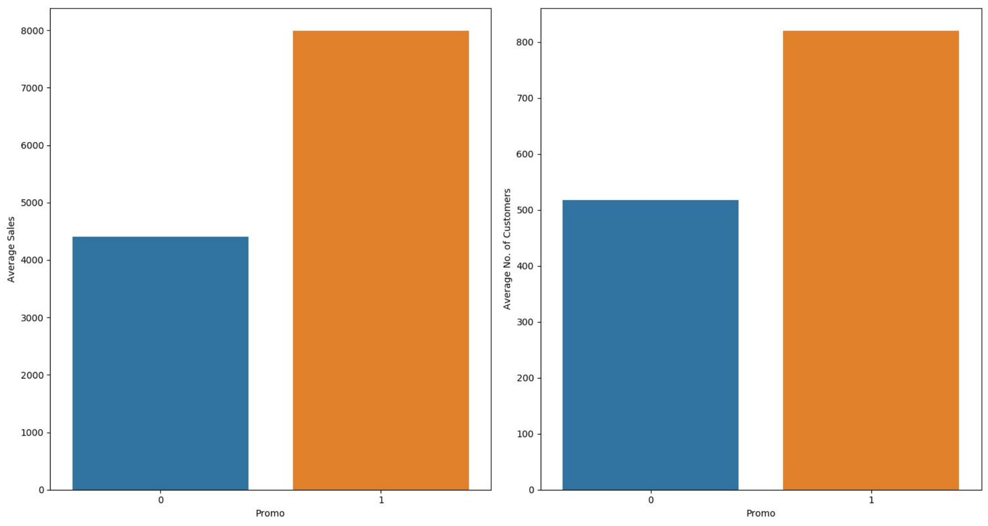

Figure 6: Average Sales & No. of Customers with & without Promo

5.3.2 State Holidays

From Figure 7 below, it can inferred that for stores open on state holidays (orange), there is a relatively high customer footfall and average sales as opposed to normal days (blue).

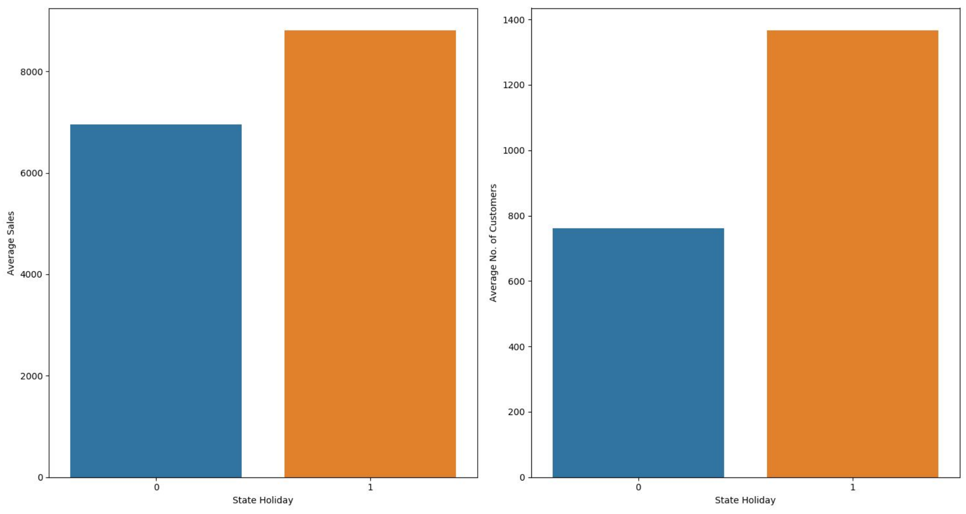

Figure 7: Average Sales & No. of Customers during State Holidays for Open Stores

5.3.3 School Holidays

According to Figure 8 below, there is a small increase in the sales numbers when there is a school holiday (orange) as opposed to normal days (blue).

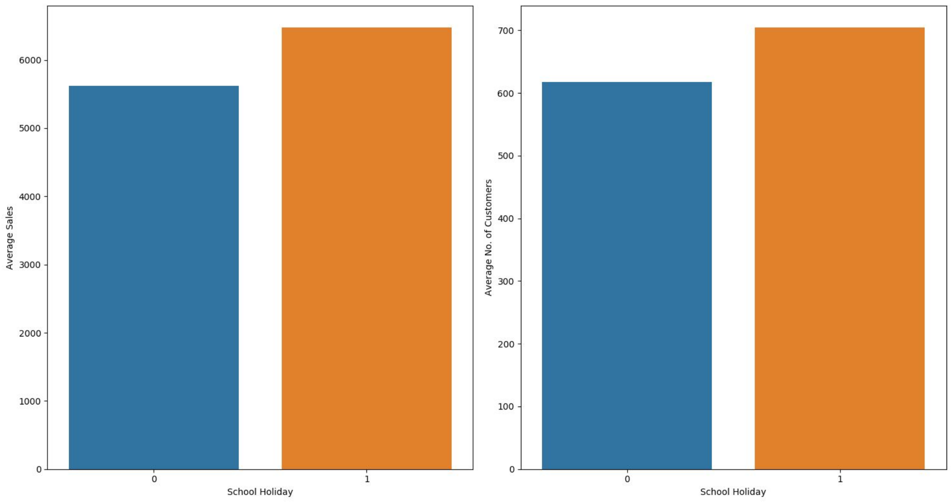

Figure 8: Average Sales & No. of Customers on School Holidays

5.3 Effect of Competition

From the scatter plots in Figure 9, we observe higher sales figures when competition is nearby, which suggests there exists a positive correlation between sales figures and competition. We also gather similar insights about the number of customers.

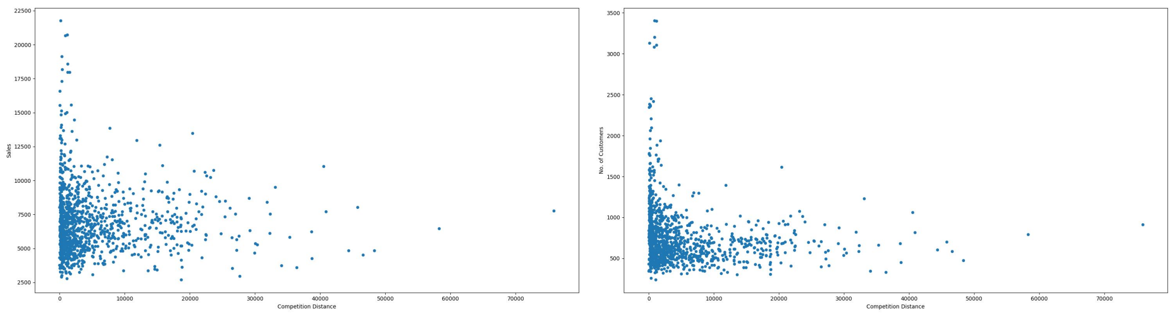

Figure 9: Sales [Left] & No. of Customers [Right] for Diff. Competition Distances

5.4 Effect of Store Type and Assortment Type

5.4.1 Store Type

We can infer from Figure 10 that average sales and customer footfall is significantly more in Store Type B as compared to other store types.

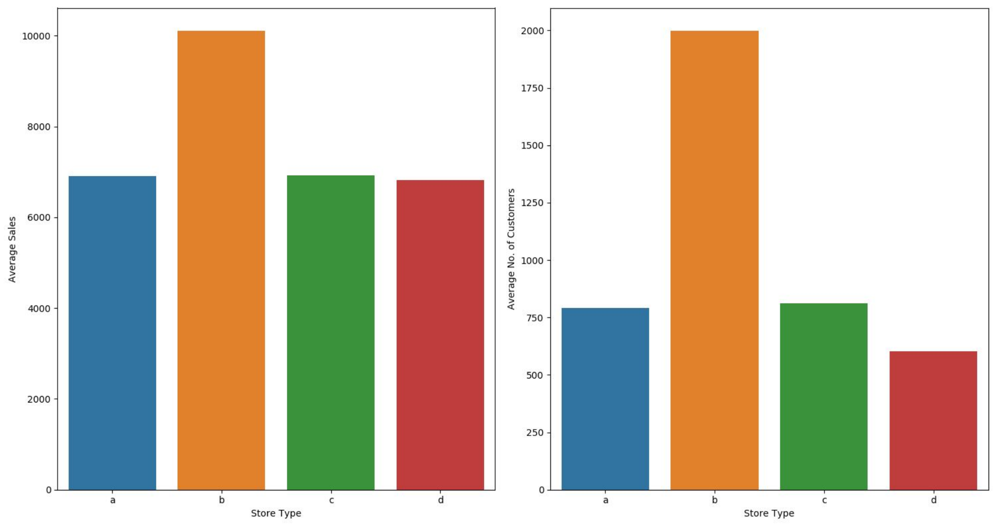

Figure 10: Average Sales & No. of Customers for Diff. Store Types

5.4.2 Assortment Type

From Figure 11, we observe that Assortment Type B (orange) results in significantly higher sales and customer footfall. However, the average spending per customer seems to be higher for Assortment Type A (blue) & Assortment Type C (green).

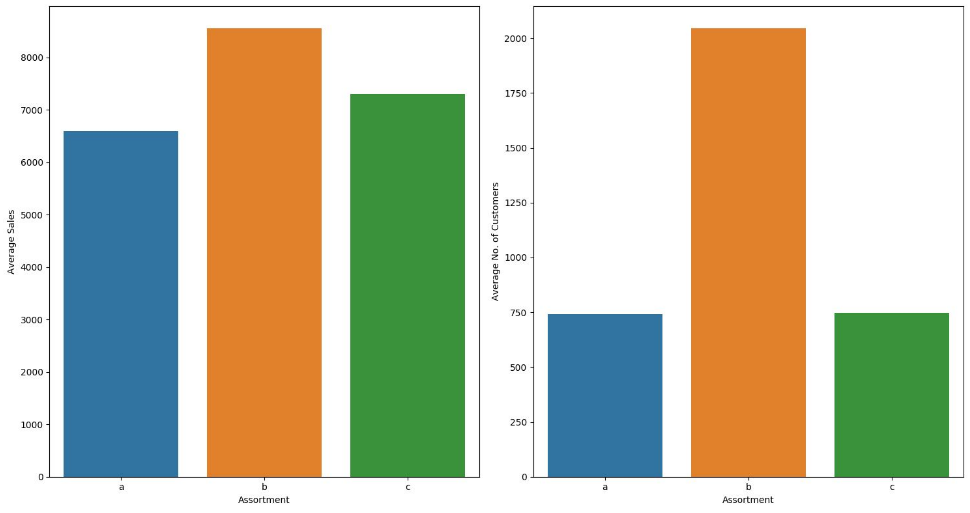

Figure 11: Average Sales & No. of Customers for Diff. Store Assortments

5.5 Correlation Matrix for Features

From the correlation matrix below, we observe that there is a strong positive correlation between the sales and the day of the week, holidays & promotions. There is also an extremely strong correlation between the sales and no. of customers but this is ignored as we do not know the no. of customers for a particular day in advance.

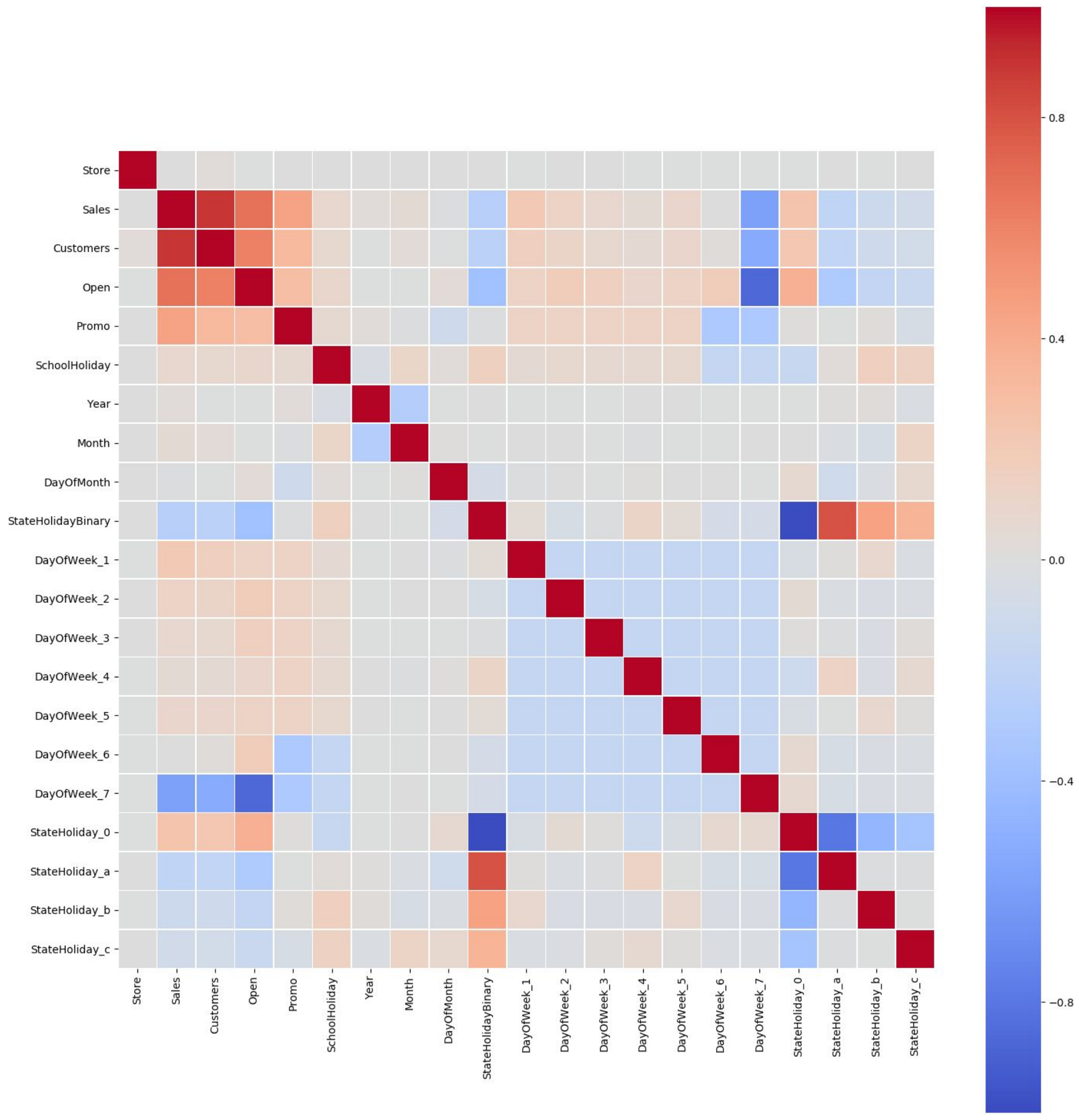

Figure 12: Correlation Matrix of Features in `train.csv`

6. Models and Experiments

6.1 Simple Geometric Mean Model

We use this as the benchmark model which calculates the geometric mean value for every `(Store, DayOfWeek, Promo)` combination.

6.1.1 Results

Kaggle Private Score on Kaggle: **0.15996**

6.2 Linear Regression

The linear regression in the project was implemented using two different approaches. In the first approach (6.2.1), the entire dataset was tackled as a single regression problem and in the second approach (6.2.2), each store was treated as an isolated regression problem. The latter performed significantly better as it broke down the overall problem for every store.

6.2.1 Linear Regression Version 1 (`linear_regression_single.py`)

6.2.1.1 Feature Selection

`Store, Promo, SchoolHoliday, Year, Month, DayOfWeek (one-hot encoded), StateHoliday (one-hot encoded), CompetitionDistance, StoreType (one-hot encoded), Assortment (one-hot encoded), AvgCustStore, AvgCustStoreMonth, AvgCustStoreYear`

6.2.1.2 Results

Kaggle Private Score: **0.26837**

6.2.2 Linear Regression Version 2 (`linear_regression_3.py`)

6.2.2.1 Feature Selection

`Promo, SchoolHoliday, Year, Month, DayOfWeek (one-hot encoded), StateHoliday (one-hot encoded), AvgCustStore, AvgCustStoreMonth.`

6.2.2.2 Results

Kaggle Private Score: **0.16209**

6.2.3 Linear Regression Version 3 (`linear_regression_log.py`)

This linear regression model tackled the prediction problem using the isolated independent store approach but using `log(Sales)` values instead. This provides numerical stability and smoothens the training function.

6.2.3.1 Feature Selection

`Promo, SchoolHoliday, Year, Month, DayOfWeek (one-hot encoded), StateHoliday (one-hot encoded), AvgCustStore, AvgCustStoreMonth.`

6.2.3.2 Results

Kaggle Private Score: **0.15522**

6.2.4 Observations & Inferences

The two linear regression experiments (6.2.1 and 6.2.2) undertaken above, underperformed as compared to the benchmark model. One possible reason behind this lacklustre performance could be attributed to not using log-normalized values for `Sales` and hence as seen above, using standardized `log(Sales)` values (6.2.3) improves the performance considerably and brings it below the benchmark standard.

6.3 XGBRegressor Model

After observing the performance of the linear regression model, we decided to try out a decision tree based model. After preliminary research, we settled on using an extreme gradient boosting model, which is a boosting ensemble algorithm that works well with supervised classification and regression problems. We used both, the single model and isolated model approaches described in 6.2. The best results using the log-normalized version of the dataset are detailed below. The mean squared error was used as the loss function for optimization, as this is close to the actual evaluation metric used in Kaggle and this report.

6.3.1 XGBRegressor Version 1 (`xgboost_regression_log.py`)

6.3.1.1 Feature Selection

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

6.3.1.2 Results

Kaggle Private Score: **0.12727**

6.3.2 XGBRegressor Version 3 (`xgboost_regression_log3.py`)

6.3.2.1 Feature Selection

`Store, DayOfMonth, Week, Month, Year, DayOfYear, DayOfWeek, Open, Promo, SchoolHoliday, StateHoliday, StoreType, Assortment, CompetitionDistance, CompetitionOpenYearMonthInteger, AvgSales, AvgCustomers, AvgSalesPer Customer`

6.3.2.2 Results

Kaggle Private Score: **0.12305**

6.3.3 Observations & Inferences

As observed from the above results, XGBoost based models gave a much higher performance as compared to the linear regression models as well as the benchmark models. This could be due to the fact that ensemble learning methods tend to outperform simple linear regressions in similar problems. The second experiment tried above (6.3.2) used an extended set of features that improved the model’s performance considerably.

6.4 Static Combiner Model

After assessing the performance of several XGBoost regressor models, we noticed that the RMSPE score had plateaued. At this point, we turned to the ensemble learning approach to further improve the RMSPE score. This time, we attempted to build a static combiner that would simply consider two models and apply a weighted average to get the final predictions (i.e. *`y_pred = y_pred_1 * w1 + y_pred_2 * w2`*).

In order to determine the correct weights, we started out with random weights and adjusted them manually until we obtained a better RMSPE score. We combined the weighted predictions from XGBRegressor Version 1 and XGBRegressor Version 3.

6.4.1 Results

Kaggle Private Score: **0.11880**

6.4.2 Observations & Inferences

Upon combining the predictions of several models together in an ensemble manner, we are able to attain a much higher score. Although this model performs better in Kaggle, it may not necessarily do so in a real scenario as the weights are manually tuned for the test dataset.

6.5 Other Models Implemented

S No.	Model Implemented	Kaggle Private Score
1	Simple Geometric Median Model	0.14598
2	Ridge Regression (log-normalized data)	0.15482
2	Random Forest Ensemble (log-normalized data)	0.14502
3	LSTM (non-normalized data)	0.16041

Table 1: Alternative Models Implemented

7. Post-Project Analysis

The KDD approach taken within this project helps us in solving the bigger problem identified at the start of the project, i.e., forecasting sales for large chain stores such as Walmart, Walgreens etc. The types of trends identified through our data analysis, such as the correlation of sales with the day of the week, state holidays, promotions, competition along with the learning models implemented to forecast sales can be generalized to the bigger problem. We believe the accuracy of the forecasting lies in store-specific and task-specific fine-tuning of the models implemented as well as identifying and selecting the right features within the particular chain store’s dataset. Furthermore, we have identified that external datasets referring to past weather conditions, Google search trends etc. can also be incorporated in the model to get a higher accuracy but we chose against including them as it undercuts our ability to build a truly generalized model.

Future improvements for our project include fine-tuning the hyperparameters for the time-series data analysis done using the LSTM model and learning the weights for combining the learning models in our XGBoost ensemble approach rather than manually tuning the weights.

8. Conclusions

The specific instance of the Rossmann Stores provides us with a good starting point to tackle the sales forecasting problem we identified. Within such problems, the focus is primarily on feature selection and engineering post which various models can be applied to those features. Therefore, our approach mainly revolved around analyzing the data and identifying interesting trends and patterns, which helped us select the right features to train our models on. As a result of the different experiments, we concluded that ensemble learning provides the best accuracy and boosted decision trees, in particular, work extremely well for sales forecasting problems.

8. References

[1] D. (www.dw.com), "Schlecker drugstores to close for good | Business | DW | 04.06.2012", *DW.COM*, 2017. [Online]. Available: http://www.dw.com/en/schlecker-drugstores-to-close-for-good/a-15996229.

[2] "Rossmann Store Sales, Winner's Interview: 1st place, Gert Jacobusse", *No Free Hunch*, 2017. [Online]. Available: http://blog.kaggle.com/2015/12/21/rossmann-store-sales-winners-interview-1st-place-gert/.

[3] "The case of Rossmann by zonakostic", *Zonakostic.github.io*, 2017. [Online]. Available: http://zonakostic.github.io/CS_109_EUROPE/.

[4] "KDD Process/Overview", *Www2.cs.uregina.ca*, 2017. [Online]. Available: http://www2.cs.uregina.ca/~dbd/cs831/notes/kdd/1_kdd.html.

[5] Y. Bengio, P. Simard and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult", *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157-166, 1994.

[6] J. Brownlee, "A Gentle Introduction to Long Short-Term Memory Networks by the Experts - Machine Learning Mastery", *Machine Learning Mastery*, 2017. [Online]. Available: https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts.

9. Appendix A – Datasets

All datasets for this project can be downloaded from the Kaggle competition for Rossmann Store Sales available at this link: https://www.kaggle.com/c/rossmann-store-sales/data. Alternatively, the extracted datasets for this project are available in the submitted .zip file under the `data/` directory. Detailed descriptions of the dataset attributes can be found in Section 4.1 of this report.

10. Appendix B – Source Code Examples

10.1 Feature Analysis

An important analysis we performed was drawing conclusions about the annual trends. A snippet of code to plot Average Sales & No. of Customers by Month is shown below.

```
# Generate plots for Avg. Sales & Customers (by Month)
fig, (axis1, axis2) = plt.subplots(1, 2, figsize=(15, 8))
ax1 = sns.barplot(x="Month", y="Sales", data=training_df, ax=axis1,
ci=None)
ax2 = sns.barplot(x="Month", y="Customers", data=training_df, ax=axis2,
ci=None)
ax1.set(xlabel="Month", ylabel="Average Sales")
ax2.set(xlabel="Month", ylabel="Average No. of Customers")
fig.tight_layout()
fig.savefig("plots/Avg. Sales & Customers (by Month).png", dpi=fig.dpi)
fig.clf()
plt.close(fig)
```

Another analysis to study the effect of competition:

```
# Generate plot for CompetitionDistance vs. Avg. Sales
fig, (axis1) = plt.subplots(1, 1, figsize=(15, 8))
ax = store_df.plot(kind="scatter", x="CompetitionDistance",
y="AvgSales", ax=axis1)
ax.set(xlabel="Competition Distance", ylabel="Sales")
fig.tight_layout()
fig.savefig("plots/Competition Distance vs. Sales.png", dpi=fig.dpi)
fig.clf()
plt.close(fig)
```

10.2 Evaluation

Our evaluation metric was RMSPE. Our implementation of RMSPE is shown below:

```python
def rmspe(y_true, y_pred):
    """
    RMSPE = sqrt(1/n * sum(((y_true - y_pred)/y_true) ** 2))
    """
    # multiplying_factor = 1/y_true when y_true != 0, else
    # multiplying_factor = 0
    multiplying_factor = np.zeros(y_true.shape, dtype=float)
    indices = y_true != 0
    multiplying_factor[indices] = 1.0 / (y_true[indices])
    diff = y_true - y_pred
    diff_percentage = diff * multiplying_factor
    diff_percentage_squared = diff_percentage ** 2
    rmspe = np.sqrt(np.mean(diff_percentage_squared))
    return rmspe
```

10.3 Models

Our most successful model was the boosted tree ensemble model with extreme gradient boosted trees. An implementation of an XGBoost model is shown below:

```python
X_train = training_df[features]
X_test = test_df[features]
y_train = training_df["Sales"]

################ TRAINING ###############
print("Training...")
regressor = XGBRegressor(n_estimators=3000, nthread=-1, max_depth=12,
                         learning_rate=0.02, silent=True, subsample=0.9,
                         colsample_bytree=0.7)
regressor.fit(np.array(X_train), y_train)
with open("models/xgboost_regression_log3.pkl", "wb") as f:
    pickle.dump(regressor, f)
########### TRAINING COMPLETED ##########

print("Making predictions...")
xgbPredict = regressor.predict(np.array(X_test))
result = pd.DataFrame({"Id": test_df["Id"], "Sales":
np.expm1(xgbPredict)})
result.to_csv("predictions/xgboost_regression_log3.csv", index=False)
```

11. Appendix C – Implementation Guidelines

11.1 Setup

Download the .zip file containing all source code and unzip the file. All code for this project is written in Python 3. After installing Python 3, make sure it is setup to run on a terminal. Open a terminal and enter ```$ python --version``` to verify that the correct version is installed.

The list of dependencies can be found in `requirements.txt`. To set up your development environment, navigate to the unzipped folder and run the following on a terminal:

```
$ pip install -r requirements.txt
```

The `data/` directory contains all data files downloaded from Kaggle. The `plots/` directory contains all plots generated and all output files are recorded in the `predictions/` directory.

11.2 Running

11.2.1 Data Analysis

To generate all plots we used for data analysis, run the following:

```
$ python DataAnalysis/generate_plots.py
```

11.2.2 Models

For each type of model, the codes can be found in the respective directories. The private leaderboard RMSPE score of each model is reported in the doc string of the python file. Description of the model and assumptions made can also be found in the python file. To run the code, run the following from the root of the repository:

```
$ python ModelType/model-name.py
```

e.g. ```$ python BenchmarkModels/simple_geometric_mean.py```

The code will automatically generate the output predictions in `predictions/`. The generated `.csv` file can be uploaded to Kaggle to get the private & public RMSPE scores.