CE/CZ4041: Machine Learning

Corporación Favorita Grocery Sales Forecasting

Project Report

Group 13:
Arkar Min-U1721052K
Deon Tan Jun Wei - U1621715F
Goh Wan Chin - U1522065L
Neo Kian How - U1722942K
Ranjit Binu Rani - U1522970J

School of Computer Science and Engineering

Roles & Responsibilities

Arkar Min
Data Preprocessing
Time-Series Forecasting using LightGBM
Time-Series Forecasting using Neural Network
Ensemble Forecasting using weighted average
Report & Presentation

Neo Kian How
Data Analysis
Report & Presentation

Ranjit Binu Rani
Research in ML Techniques
Report & Presentation

Deon Tan Jun Wai
Research in ML Techniques
Report & Presentation

Goh Wan Chin
Research in ML Techniques
Report & Presentation

TABLE OF CONTENTS

Problem Statement

Corporación Favorita Grocery Sale Forecasting is the time-series forecasting competition. The organizer provided 8 files namely *train.csv, test.csv, sample_submission.csv, stores.csv, items.csv, oil.csv, transactions.csv, holidays_events.csv*. Only *train.csv, stores.csv, items.csv, oil.csv, transactions.csv* and *holidays_event.csv* can be used for forecasting. Based on the test.csv, we found out that we were challenged to build a model that can more accurately predict the sale of items sold in different branches of Corporación Favorita for next 16 days.

Submissions for the competition were evaluated using the Normalized Weighted Root Mean

$$NWRMSLE = \sqrt{\frac{\sum_{i=1}^{n} w_i(\ln(\hat{y}_i + 1) - \ln(y_i + 1))^2}{\sum_{i=1}^{n} w_i}}$$

where for row i, $\hat{y}_i$ is the predicted unit_sales of an item and y_i is the actual unit_sales; n is the total number of rows in the test set. The weights, w_i, can be found in the items.csv file. Perishable items are given a weight of 1.25 where all other items are given a weight of 1. We should be careful not to make any negative predictions since the metric uses ln(y+1).

Methodology

Throughout this project, we used Jupyter Notebook as the main application to run all the data processing Python codes. Jupyter Notebook allows us to integrate different plugins that were required for this project. Also, Google Colab was the secondary online platform used to run the tests for Neural Networks as it provides free GPU processing and 12GB RAM.

Datasets were analysed and split before selecting and extracting features from them. These features were combined in the training data to get a more accurate prediction. The codes implemented to describe the different models were run and submitted to the Kaggle competition to get ranking and scores. The parameters of the models were fine tuned to possibly decrease error rate and get a better score by predicting more accurately.

Data Analysis

Before diving into machine learning techniques, we explored the data first to know more about the nature of data. Time-series data always involves trends and seasonality. We expected to find those in the data provided by Kaggle.

Firstly, we implemented source codes to check if the fields in any of the files contain missing or null value. To perform this check, a package named ‘missingno’ was used to check for null or missing value and a package named ‘matplotlib’ was used to plot graphs to provide better visualization.

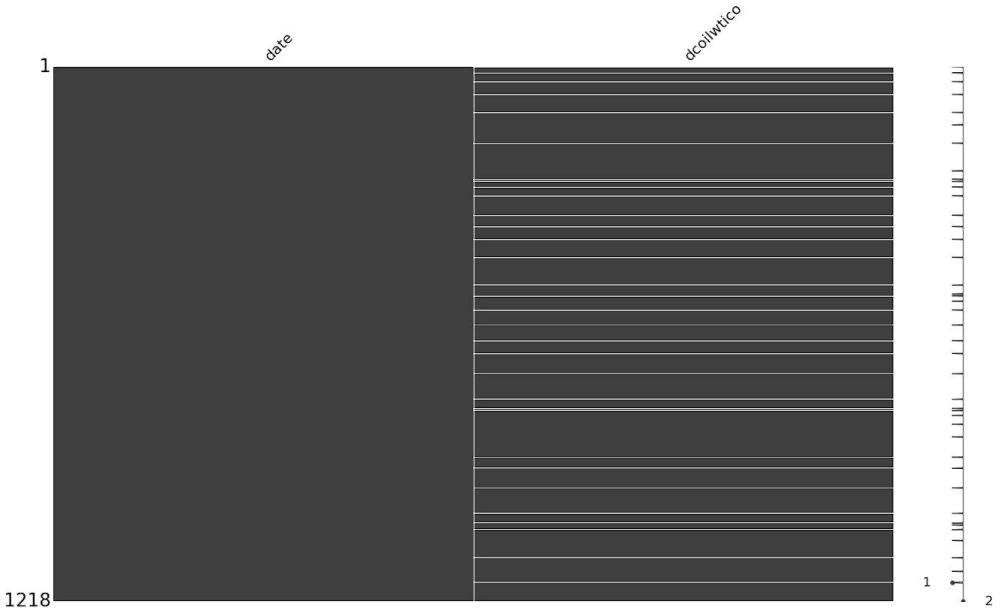

Figure 1: *oil.csv*

Figure 1 displays shows that the dataset has missing values, specifically the column ‘dcoilwtico’ which contains the value of oil over the years. Similar tests were run on the other files and the generated graphs are in the Appendix.

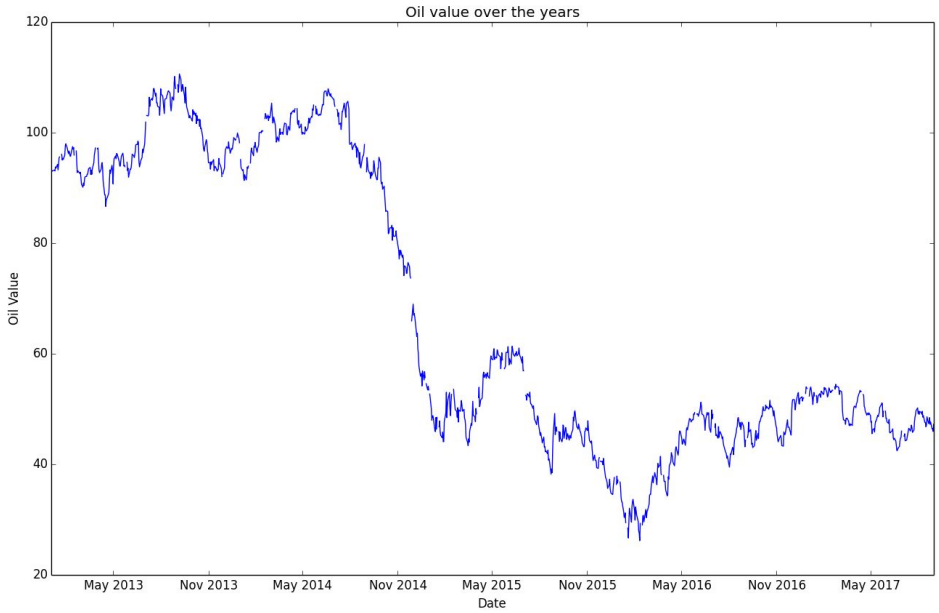

Figure 2: Oil's Value Distribution

Figure 2 shows that the value of oil is trending downwards. Moreover, there is a sharp fall from the third quarter of 2014 to the first quarter of 2015.

We also plotted a graph to show the relationship between oil price and unit sales.

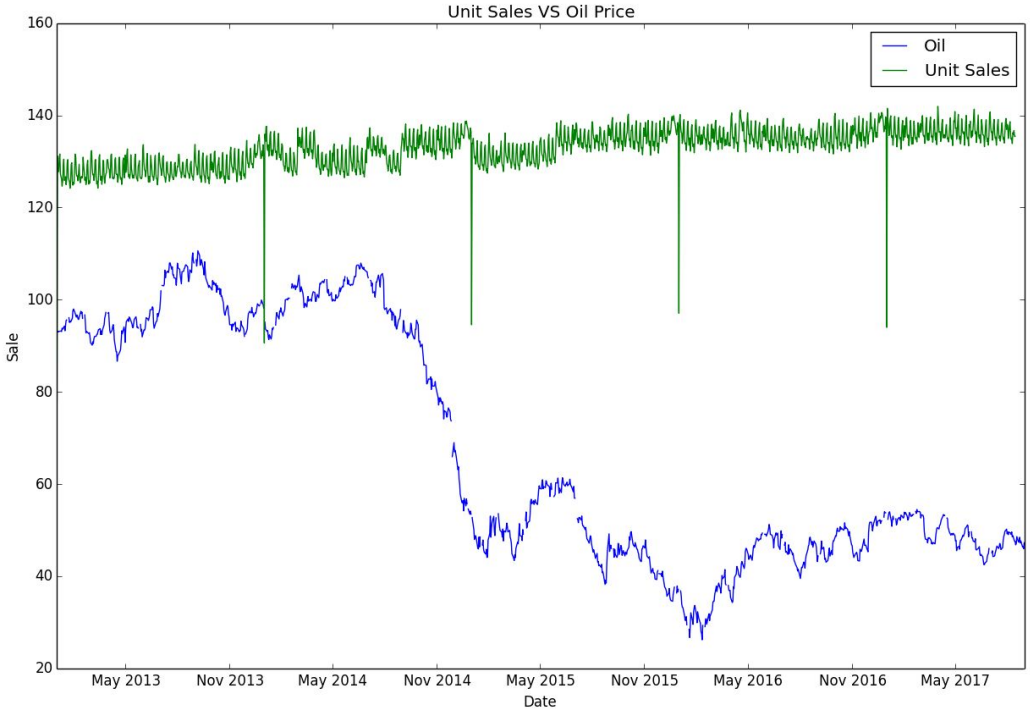

Figure 3: Unit Sales VS Oil Value

Figure 3 shows the distribution of unit sales and oil value. There does not seem to be a high correlation between both features as the distribution of unit sales appear to be consistent over the years. However, the distribution of sales between the first quarter of 2014 and first quarter of 2015 appears to have some fluctuation which is in coherence with the distribution of oil value over the same period. So, we will not be using oil values as features in our forecasting.

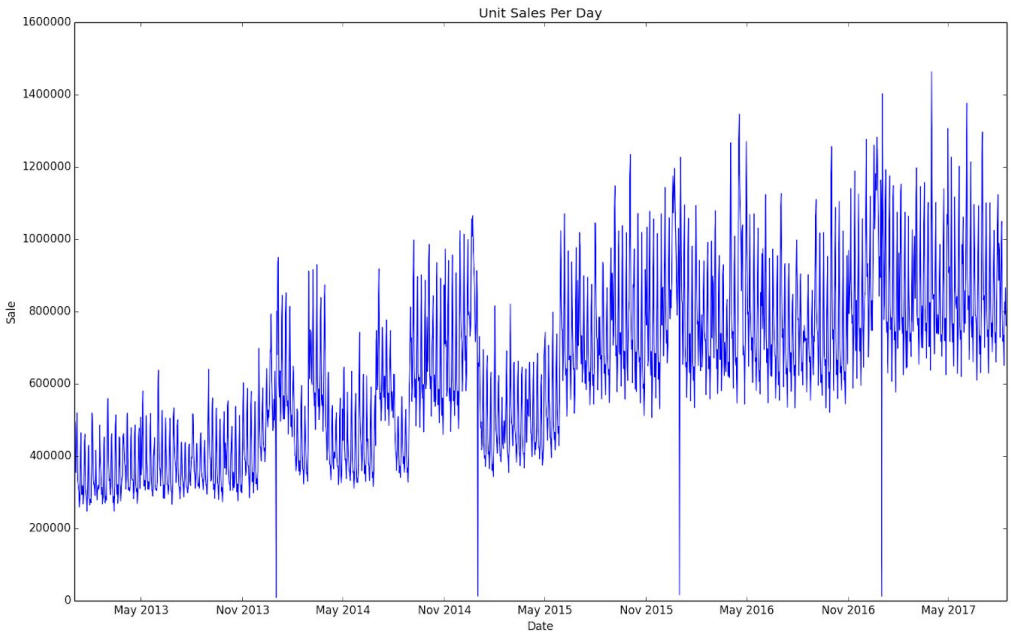

Figure 4: Unit Sales Distribution

Figure 4 shows the distribution of unit sales from all the items from all the branches over the years. Generally, it shows an increasing trend. There are 4 significant spikes which are followed by a sharp fall. This is most likely due to the festival of Christmas in the month of December and the sharp fall is due to holiday where shops are closed. We can observe the upward trend and potential seasonality pattern here. We can clearly see the seasonality pattern between 2014 and 2015. However, there are a lot of spikes during 2016 and 2017.

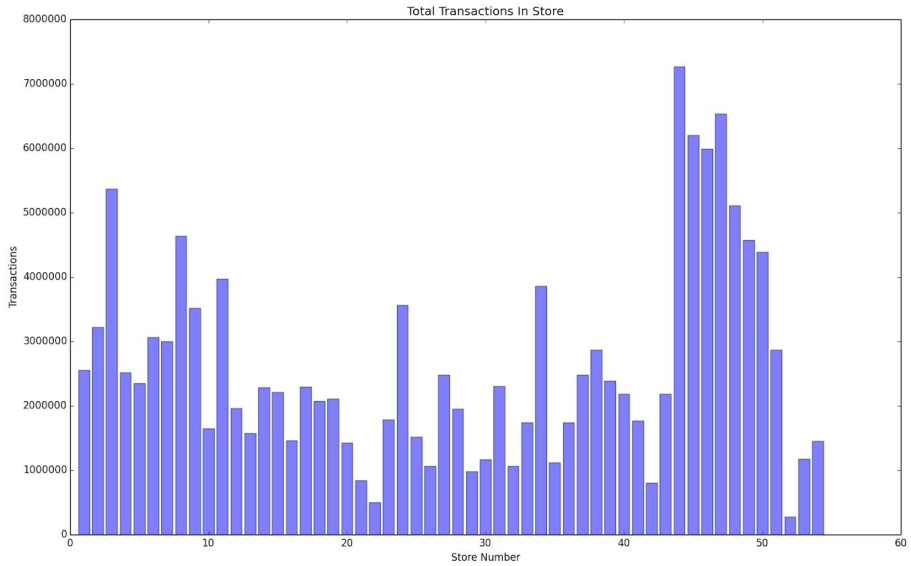

Figure 5: Total Transactions Made Per Store

There are 54 stores in total. We analyzed the behaviour of each store.Distribution of total transactions are quite random. Some stores has high transaction rates and others have low transaction rates. This could depend on the location of the stores. Therefore, we explored the trend in location of the stores.

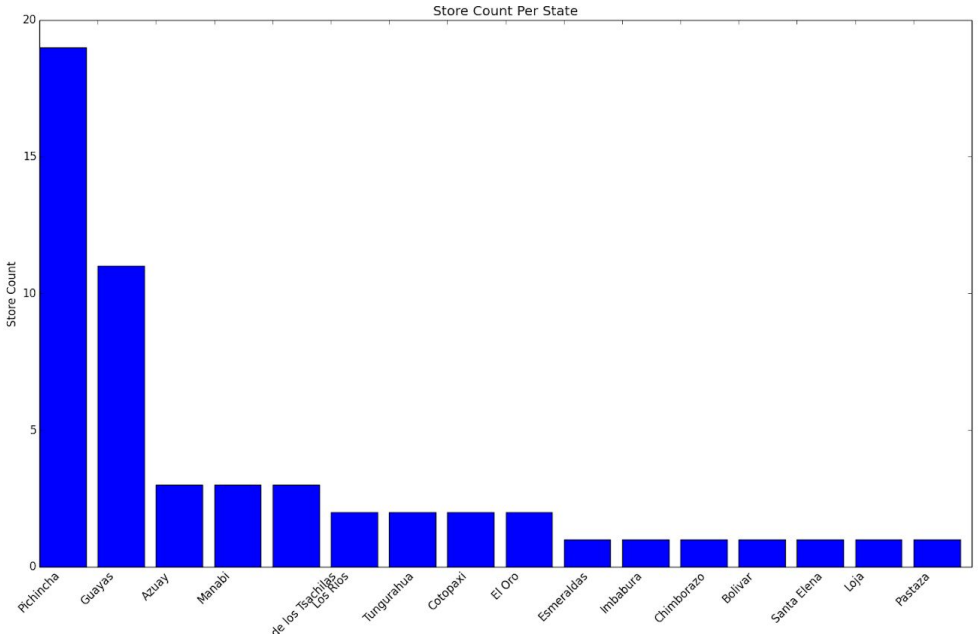

Figure 6: Store Count Per State

The state of Pichincha has a total of 19 stores, which is understandable as it housed Quito, the capital city of Ecuador. The state of Guayas which housed Guayaquil, the

largest city of Ecuador has a total of 11 stores. The rest of the state has a range of 1 to 3 stores.

In evaluation metric, we can see that weight is included. So, we plotted a graph to better visualize the importance of weight.

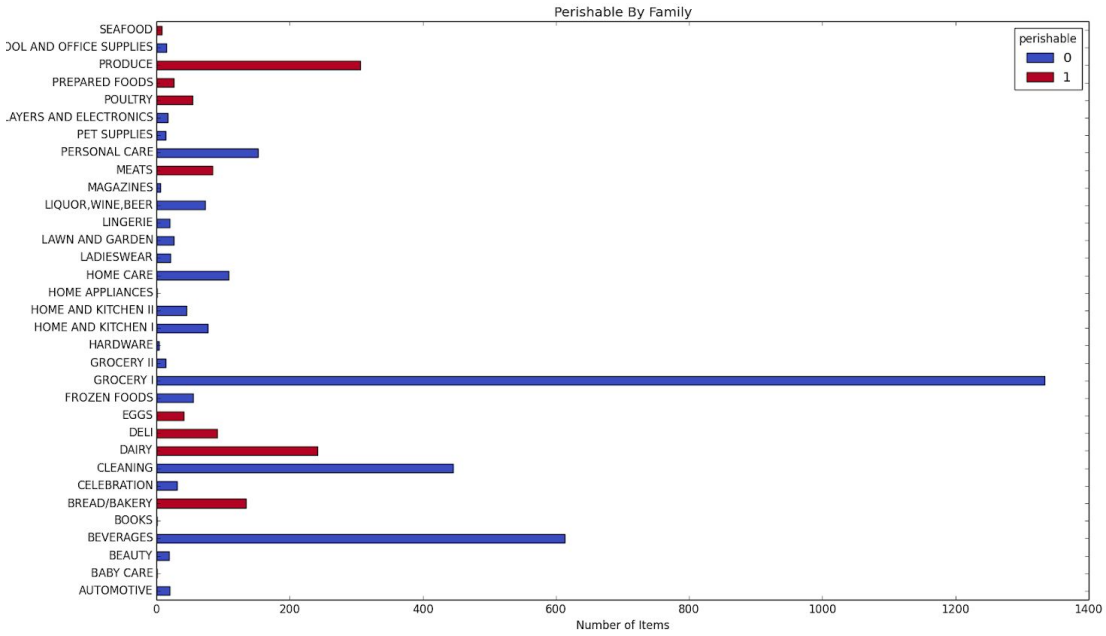

Figure 7: Item Count And Perishable By Family

Figure 7 shows that the top 5 families that consists of the most items are *'GROCERY I', 'BEVERAGES', 'CLEANING', 'PRODUCE', DAIRY'* in descending order. Two of the families, namely *'DAIRY'* and *'PRODUCE',* belong to the class '*perishable*'. There are more non-perishable items compared to the perishable items. In addition, most perishable item categories have total-number of items less than 400.

Lastly, we explored the holiday data to analyze the effect of holidays and events on the sales of products.

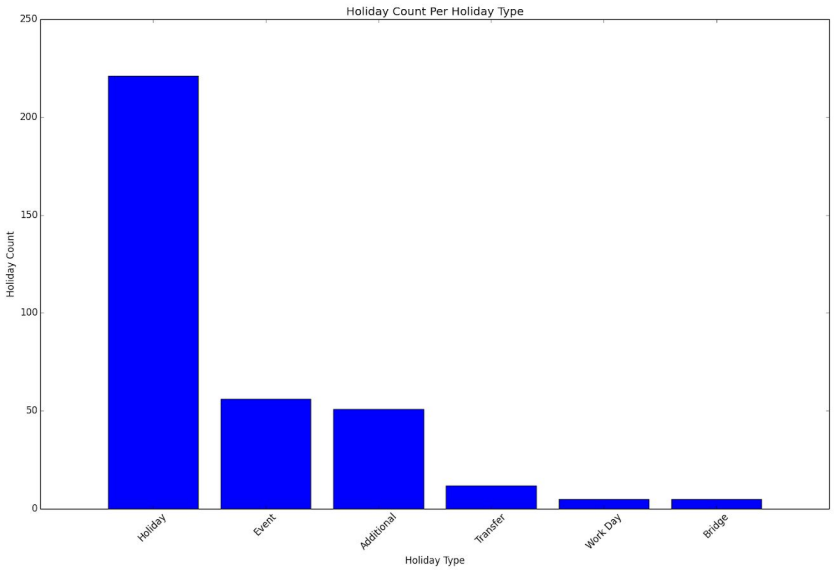

Figure 8: Total number of holidays by category

There are different type of holidays. Dates of holidays are sometimes different year by year. New holidays are added in some years. So, the data is quite random. It is not suitable to use for forecasting.

Proposed Solution

Brief Introduction

We used an approach similar to mean-forecasting, which uses the mean of previous sales to predict the future sales. For our approach, we used not only mean, but also other statistic values like maximum, minimum,median, standard deviation of sales. We also used promotion as an input feature. We also used the weight of perishable items in our model.

We implemented ensemble forecasting which involves two machine learning models, LightGBM and Neural Network. Forecasting results are produced based on weighted average, giving 60% weight to predictions from LightGBM and 40% percent to the predictions from Neural Network.

Data Preprocessing

As the training dataset is quite large, around 4.65 GB, it takes a long time to open the file before we start training anything. In addition, we decided to use maximum one year of data for our forecasting because of inconsistent seasonality we have for every year. We split the data into four different training data sets and one validation set. Before splitting the file, we made sure that there are no NaN values in those files.

train_set_short.csv	Data for 28 days
train_set.csv	Data for half a year
train_set_long.csv	Data for 56 days
valid_set.csv	Data for 1 week
train_set_one_year.csv	Data for 1 year

Table 1. Training files representation

Train_set_short, train_set_long, and valid_set were used for initial stages for exploring different ML algorithms. In later stages, train_set and train_set_long are used because we figured out that we needed more data. Validation was done using data from those training set because having 16 days for validation while we are predicting sales for next

16 days produced accurate results. Codes for data preprocessing are in **preprocessing.py** and **preprocessing_two.py**.

Feature Engineering

This is one of the most important parts of the our project since we have limited features. We calculated the mean, median and standard deviation of all sales which are 3 days, 7 days, 14 days, 30 days, 60 days and 140 days before the next 16 predictions. We also calculated sales mean, maximum and minimum number of promotion before next 16 predictions. We also calculated the mean weekly sales for 4 weeks before and 20 weeks before the next 16 predictions. With that, we hope to capture any pattern that exists before the next 16 days.

```
for i in [3,7,14,30,60,140]:
        X["mean_"+str(i)] = get_timespan(df_train, t2017, i, i).mean(axis=1).values
        X["median_"+str(i)] = get_timespan(df_train, t2017, i, i).median(axis=1).values
        X["std_"+str(i)] = get_timespan(df_train, t2017, i, i).std(axis=1).values
        X["promo_"+str(i)] = get_timespan(promo_2017, t2017, i, i).sum(axis=1).values
        X["max_"+str(i)] = get_timespan(promo_2017, t2017, i, i).max(axis=1).values
        X["min_"+str(i)] = get_timespan(promo_2017, t2017, i, i).min(axis=1).values
for i in range(7):
    X['mean_4_dow{}_2017'.format(i)] = get_timespan(df_train, t2017, 28-i, 4, freq='7D').mean(axis=1).values
    X['mean_20_dow{}_2017'.format(i)] = get_timespan(df_train, t2017, 140-i, 20, freq='7D').mean(axis=1).values
```

Figure 9: Codes for feature engineering

Machine Learning Models

In this section, we will explain about two machine learning model that we used, reasons for using them and implementation.

LightGBM (Overview)

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It has a faster training speed, higher efficiency and thus, gives a better accuracy. It has support for parallel and GPU learning.

Many boosting tools use pre-sort based algorithms for decision tree learning. It is a simple solution, but not easy to optimize. LightGBM uses histogram-based algorithms which bucket continuous feature (attribute) values into discrete bins. This speeds up training and reduces memory usage.

Light GBM is sensitive to overfitting and can easily overfit small data. It is preferred for handling large-scale data like the datasets provided for Corporación Favorita.

Light GBM grows tree vertically while other algorithm grows trees horizontally. This means that Light GBM grows tree leaf-wise while other algorithm grows level-wise. It will choose the leaf with max delta loss to grow. When growing the same leaf, leaf-wise algorithm can reduce more loss than a level-wise algorithm. The figures below explain the implementation of LightGBM and other boosting algorithms

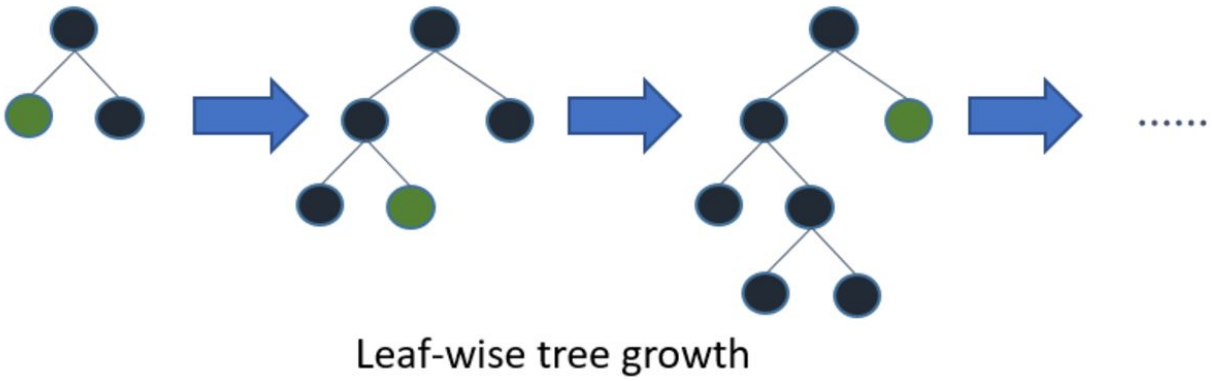

Figure 10: LightGBM tree growth

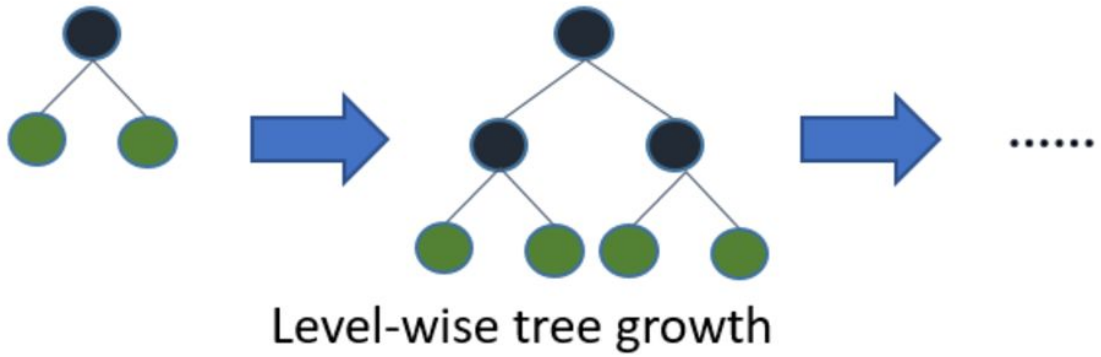

Figure 11: LightGBM tree growth

LightGBM (Implementation)

```
from datetime import date, timedelta
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error
import matplotlib as plp
import lightgbm as lgb

df_train = pd.read_csv(
    'train_set_one_year.csv', usecols=[1, 2, 3, 4, 5],
    converters={'unit_sales': lambda u: np.log1p(
        float(u)) if float(u) > 0 else 0},
    parse_dates=["date"],dtype={'onpromotion': bool}
)

df_test = pd.read_csv(
    "test.csv", usecols=[0, 1, 2, 3, 4],
    dtype={'onpromotion': bool},
    parse_dates=["date"]  # , date_parser=parser
).set_index(
    ['store_nbr', 'item_nbr', 'date']
)

items = pd.read_csv(
    "items.csv",
).set_index("item_nbr") #In order to give weight to item perishable
```

Figure 12: Importing and reading of data

First, we import all the required packages (especially, **panda**, **numpy** and **lightGBM**) and then read in the training data for 1 year that was created during preprocessing. We also read in the test data and items list.

```
promo_train = df_train.set_index(
    ["store_nbr", "item_nbr", "date"])[["onpromotion"]].unstack(
        level=-1).fillna(False)

promo_train.columns = promo_train.columns.get_level_values(1)

promo_test = df_test[["onpromotion"]].unstack(level=-1).fillna(False)
promo_test.columns = promo_test.columns.get_level_values(1)

promo_test = promo_test.reindex(promo_train.index).fillna(False)
promo_2017 = pd.concat([promo_train, promo_test], axis=1)
del promo_test, promo_train

df_train = df_train.set_index(
    ["store_nbr", "item_nbr", "date"])[["unit_sales"]].unstack(
        level=-1).fillna(0)
df_train.columns = df_train.columns.get_level_values(1)

items = items.reindex(df_train.index.get_level_values(1))
```

Figure 13: Filling in NaN values

We fill in the NaN values in the onpromotion column in both the train and test datasets with False values before concatenating the 2 datasets to form the list of promotions in 2017. We have to concatenate promotion of train dataset and test dataset because we required a series of promotion before any prediction date. The NaN values in unit sales for the training dataset are filled with 0.

```
def get_timespan(df, dt, minus, periods, freq='D'):
    return df[pd.date_range(dt - timedelta(days=minus), periods=periods, freq=freq)]

def prepare_dataset(t2017, is_train=True):
    X = pd.DataFrame({
          "day_1_2017": get_timespan(df_train, t2017, 1, 1).values.ravel()
        })
    for i in [3,7,14,30,60,140]:
            X["mean_"+str(i)] = get_timespan(df_train, t2017, i, i).mean(axis=1).values
            X["median_"+str(i)] = get_timespan(df_train, t2017, i, i).median(axis=1).values
            X["std_"+str(i)] = get_timespan(df_train, t2017, i, i).std(axis=1).values
            X["promo_"+str(i)] = get_timespan(promo_2017, t2017, i, i).sum(axis=1).values
            X["max_"+str(i)] = get_timespan(promo_2017, t2017, i, i).max(axis=1).values
            X["min_"+str(i)] = get_timespan(promo_2017, t2017, i, i).min(axis=1).values
    for i in range(7):
        X['mean_4_dow{}_2017'.format(i)] = get_timespan(df_train, t2017, 28-i, 4, freq='7D').mean(axis=1).values
        X['mean_20_dow{}_2017'.format(i)] = get_timespan(df_train, t2017, 140-i, 20, freq='7D').mean(axis=1).values
    for i in range(16):
        X["promo_{}".format(i)] = promo_2017[t2017 + timedelta(days=i)].values.astype(np.uint8)
    if is_train:
        y = df_train[
            pd.date_range(t2017, periods=16)
        ].values
        return X, y
    return X
```

Figure 14: Feature engineering functions

The get_timespan function returns the data within the date range given. The prepare_dataset function loops through the dataset and does feature engineering as mentioned in a previous section.

```
print("Preparing dataset...")
t2017 = date(2017, 6, 21)
X_l, y_l = [], []
for i in range(4):
    delta = timedelta(days=7 * i)
    X_tmp, y_tmp = prepare_dataset(t2017 + delta)
    X_l.append(X_tmp)
    y_l.append(y_tmp)
X_train = pd.concat(X_l, axis=0)
y_train = np.concatenate(y_l, axis=0)
del X_l, y_l

X_val, y_val = prepare_dataset(date(2017, 7, 26))

X_test = prepare_dataset(date(2017, 8, 16), is_train=False)
```

Figure 15: Prepare dataset for training model and test model

```
params = {
    'num_leaves': 80,
    'objective': 'regression',
    'min_data_in_leaf': 200,
    'learning_rate': 0.02,
    'feature_fraction': 0.8,
    'bagging_fraction': 0.7,
    'bagging_freq': 1,
    'metric': 'l2',
    'num_threads': 16
}
```

Figure 16: Parameters for LGBM model

```
MAX_ROUNDS = 5000
val_pred = []
test_pred = []
cate_vars = []
for i in range(16):
    print("=" * 50)
    print("Step %d" % (i+1))
    print("=" * 50)
    dtrain = lgb.Dataset(
        X_train, label=y_train[:, i],
        categorical_feature=cate_vars,
        weight=pd.concat([items["perishable"]] * 4) * 0.25 + 1
    )

    dval = lgb.Dataset(
        X_val, label=y_val[:, i], reference=dtrain,
        categorical_feature=cate_vars,
        weight=items["perishable"] * 0.25 + 1
    )

    bst = lgb.train(
        params, dtrain, num_boost_round=MAX_ROUNDS,
        valid_sets=[dtrain, dval], early_stopping_rounds=50, verbose_eval=100
    )

    print("\n".join(("%s: %.2f" % x) for x in sorted(
        zip(X_train.columns, bst.feature_importance("gain")),
        key=lambda x: x[1], reverse=True
    )))
    val_pred.append(bst.predict(
        X_val, num_iteration=bst.best_iteration or MAX_ROUNDS))
    test_pred.append(bst.predict(
        X_test, num_iteration=bst.best_iteration or MAX_ROUNDS))
```

Figure 17: Training LGBM model

This model is trained with new Date-set every time before predicting sale for one day. So, that is the reason by it needs to go through 16 iterations since we are making prediction for 16 days. Every 1 round, cross-validation is done on both training set and validation set. The model is trained until it reaches 5000 rounds(epouches) or it's cross-validation score cannot be improved anymore.

Neural Network (Overview)

Neural Network is a tree based network where it works based on the weight of each node. The algorithm consists of three portions, the input, activation function and the output. In this case, the efficiency of the algorithm depends on the number of layers inside the activation function. The larger the activation function, the longer it will take to compute the result.

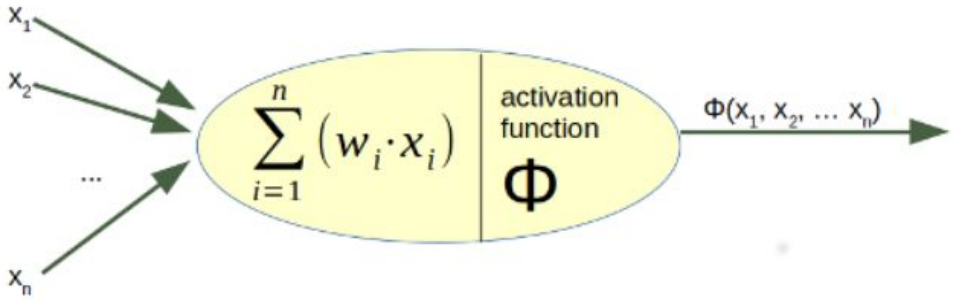

Figure 18: Activation function for Neural Network

During the processing of the weights, it also includes a bias weight which will determine a line cross y axis when splitting the data into multiple categories. In the computation, it consists of hidden layers where it would reorganize or rearrange the input data.

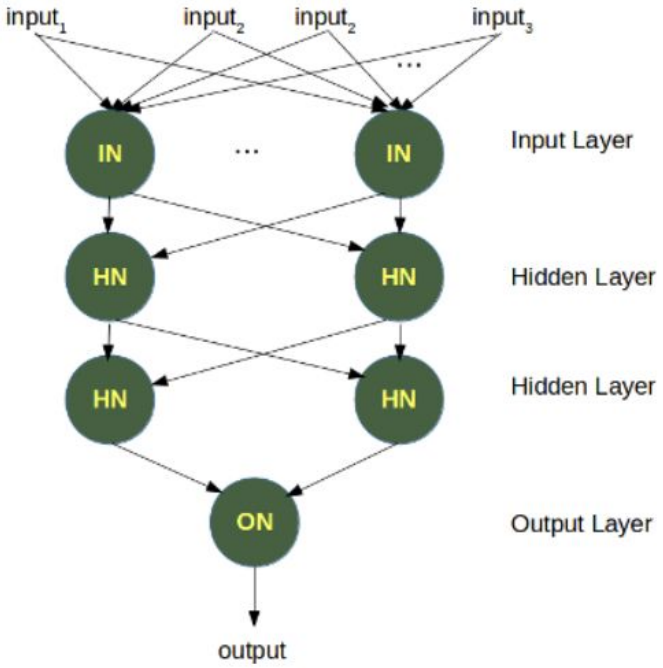

Figure 19: Neural network model

However, when compared against LGBM, the algorithm is not sensitive to over fitting. Hence, it gives a slightly better result but at a slower speed when compared against LGBM which we do not want as it will affect the overall score in kaggle result.

Neural Network (Implementation)

For our project, we will be using 7 hidden layers, each layer having 512,256,128,64,32,16 and 1 neurons respectively.

Data preprocessing for the neural network model is almost the same as LightGBM model. So, we will just highlight neural network we built. We used the keras package from tensorflow for neural network.Figure 20 shows the code implementation.

```
from datetime import date, timedelta
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
import matplotlib as plp
import tensorflow as tf
from tensorflow import keras

import warnings
warnings.filterwarnings('ignore')
```

Figure 20: Importing of packages

Another minor difference from LightGBM is that we have to transform the generated dataset into matrix which is appropriate for keras dense layer as shown below.

```
scaler = StandardScaler()
scaler.fit(pd.concat([X_train, X_val, X_test]))
X_train[:] = scaler.transform(X_train)
X_val[:] = scaler.transform(X_val)
X_test[:] = scaler.transform(X_test)

X_train = X_train.as_matrix()
X_test = X_test.as_matrix()
X_val = X_val.as_matrix()
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
X_val = X_val.reshape((X_val.shape[0], 1, X_val.shape[1]))
```

Figure 21: Data Transformation

Figure 22 shows the structure of neural network. We used Relu activation function (**tf.nn.relu**) for each neutron.

```
model = keras.Sequential([
 keras.layers.Flatten(input_shape=(X_train.shape[1],X_train.shape[2])),
 keras.layers.Dense(512, activation=tf.nn.relu),
 keras.layers.Dense(256, activation=tf.nn.relu),
 keras.layers.Dense(128, activation=tf.nn.relu),
 keras.layers.Dense(64, activation=tf.nn.relu),
 keras.layers.Dense(32, activation=tf.nn.relu),
 keras.layers.Dense(16, activation=tf.nn.relu),
 keras.layers.Dense(1)
 ])
 return model
```

Figure 22: Model Activation Function

```
N_EPOCHS = 1000

val_pred = []
test_pred = []
for i in range(16):
    print("=" * 50)
    print("Step %d" % (i+1))
    print("=" * 50)
    y = y_train[:, i]
    y_mean = y.mean()
    xv = X_val
    yv = y_val[:, i]
    model = build_model()
    model.compile(optimizer= tf.train.AdamOptimizer(0.001), loss='mse', metrics=['mse'])
    callbacks = [
      tf.keras.callbacks.EarlyStopping(patience=10, monitor='val_loss')
    ]
    model.fit(X_train, y - y_mean, batch_size = 65536,epochs = N_EPOCHS, verbose=2,callbacks=callbacks, validation
    val_pred.append(model.predict(X_val)+y_mean)
    test_pred.append(model.predict(X_test)+y_mean)
```

Figure 23: Training NN Model

Just like LightGBM model, we have to do 16 iterations in order to predict for 16 days. Adam optimizer is used for optimisation of neural network. Early stopping is also implemented here so this model will train until it's cross-validation score does not improve anymore or it reaches 1000 EPOCHs.

Ensemble Forecasting

We will be using weighted average in order to produce the final prediction result. Initially, we planned to use 50-50. However, predictions produced by LightGBM model always give better results compared to the predictions produced by the NN model. So, after many trials, we figured out that giving 60% percent weight to the LightGBM model and 40% weight to the NN model produces the best result and that will be the final result.

The code for this implementation is in the **NN+LGBM.ipynb** file.

Cross-Validation

The way both NN and LightGBM training is done is that they will train till cross-validation score is no longer improved. As mentioned before, validation set is created based on the sales 16 days after the training data. This is because we are predicting sales for 16 days in test set.

During model training, we did the cross-validation on mean squared error(mse) on both training set and validation set for every iteration. We got the very low MSE score for final LightGBM model and NN model. In this way, we can assure that our model is suitable for forecasting the sale.

Experiments

In this section, all the steps taken to reach the final steps will be described. First, we started by doing simple linear regression using all the previous unit_sales data. However, cross-validation result was so bad that it was not even submitted to the Kaggle.

After we found out about LightGBM model, we started using it only by inputting the mean sale of 3,7,14 and 16 days before. That is the first model we submitted to the kaggle. We got the score of 0.58086 for private and 0.54792 for public.

After that, we added promotion as another feature. We added weight for “perishable” items. We added more and more engineered features into the LightGBM model. However, adding more features does not always give you the better score. So, we had to adjust the engineered features in order to get the optimal model. Other than input features, LightGBM model also has a few parameters that we can tune. So, we also adjusted those parameters to get the optimal prediction.

After analysis of other solutions, we found out that neural network can be used for this forecasting problem. So, we built the simple neural network with few hidden layers. We gradually added more and more layers and neurons to the network. The prediction from neural network gets better and better. However, they are always worse than the predictions from LightGBM.

Finally, we decided to do ensemble forecasting by combining the prediction result of two models. Initially, we used averaging method. However, since LightGBM method always does better than NN, we decided to give more weight to the LightGBM model. As a result, we use weighted average ensemble method.

After all the experiments, we have reached the final LightGBM model and NN model.

The complete history of experiment and submission can be found in **Appendix**.

Kaggle Evaluation Score and Rank

For the Corporación Favorita sales prediction Kaggle competition, we submitted our solutions multiple times after fine tuning certain parameters to see improvement in our score and rank. Our overall best score was for our prediction models, LightGBM and Neural network. The table below shows our best public and private score and ranking.

	Public Leaderboard	Private Leaderboard
Score	0.51276	0.52013
Rank	358 out of 1675	99 out of 1675

Table 2 : Score and Ranking for Kaggle competition

Challenges

Forecasting sales of the individual store alone is challenging enough. For this competition, we are required to predict the sales of each item sold in different stores, which are located across the country. Therefore, a lot of factors such as location of stores, demographics of customers in each store location, holidays, signature events like disaster, war, elections, trend of certain items have effects on the sale of items. Therefore, it is challenging to include all of those features in forecasting sales.

The very first problem we have is not having enough data. Although holidays, locations of the stores are given, it will be better if we can have dates of significant events and demographics of population for each store. We are given oil data but during data analysis, we found out that it does not seem to have effect on unit sales.

The second problem will be relating those special events with sales. We are given dates of holidays. It is hard to determine the effect of those holidays on the sale data. Moreover, dates of holidays are not the same for every year. So, it will be hard to use holiday as feature for our forecasting.

Based on two problems above, the third problem we can have is not having enough features for training. The only reliable features we have unit_sales, promotion, and weight for perishable items. Even for promotion, certain amount of promotion columns are missing for 2013 datas in train data. Although, huge sale data set is given, we cannot use them all because there are a lot of spikes in data and we cannot assure whether seasonality exists in the sale or not.

One of the most time consuming tasks was actually fine tuning the models we built in order to improve our scores and ranking in the Kaggle competition. This includes various parameters that are defined for the models. Fine tuning the model parameters requires a lot of trial and error and multiple runs but it must be done to look for the best possible outcome and prediction.

The training dataset provided by the Kaggle competition takes up 4.65GB and based on the algorithm that we used, it requires large amount of RAM. As most of our laptops only have 8GB RAM, we could not even open the training data file to view the contents. This is why the data preprocessing was so important. Due to the overload of memory, our laptops hanged multiple times during program execution. Luckily some group members did not face this issue, thanks to their computer's increased RAM capacity of 16GB. Therefore, we were able to share the preprocessed files with each other for further processing and building models to predict the sales.

Conclusion

There are a lot of things we learnt from the project. Firstly, we have learnt about time-series forecasting problems which is outside our lecture scope. We have learned out about many different time-series algorithms like ARIMA, ETS, LightGBM, and so. We have also learnt about LightGBM and Neural Network, the way they work and how they can be used not only for time-series forecasting but also for other machine learning problems. We have learned that having a lot of input features will not always produce the best result. We have learned how to fine-tune the model so that they will give better result. We have also learned that how to analyse the data before applying any Machine Learning Algorithms.

To conclude, this is undoubtedly, one of the more challenging projects we have done during our study in NTU.

Appendix

Null-Checking for data file

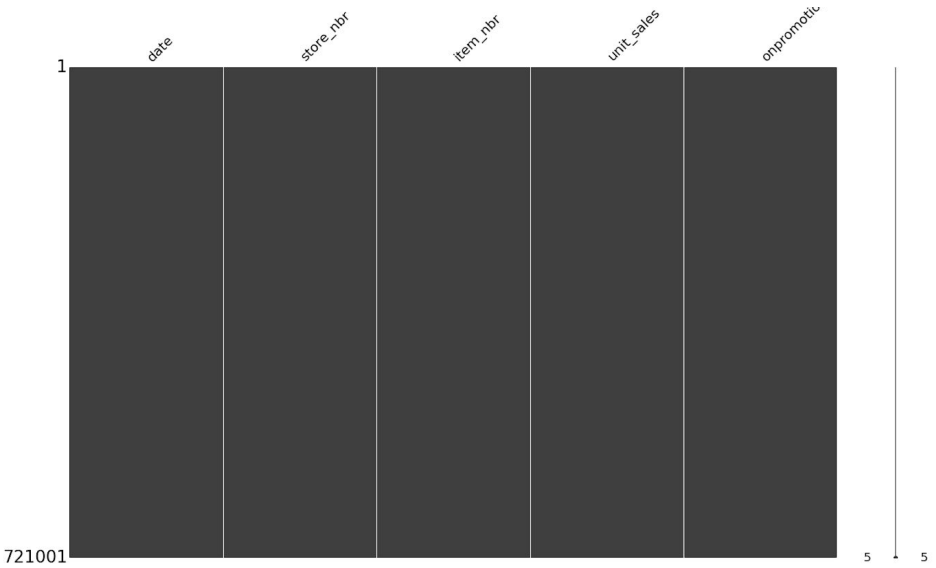

Figure 24: *train.csv*

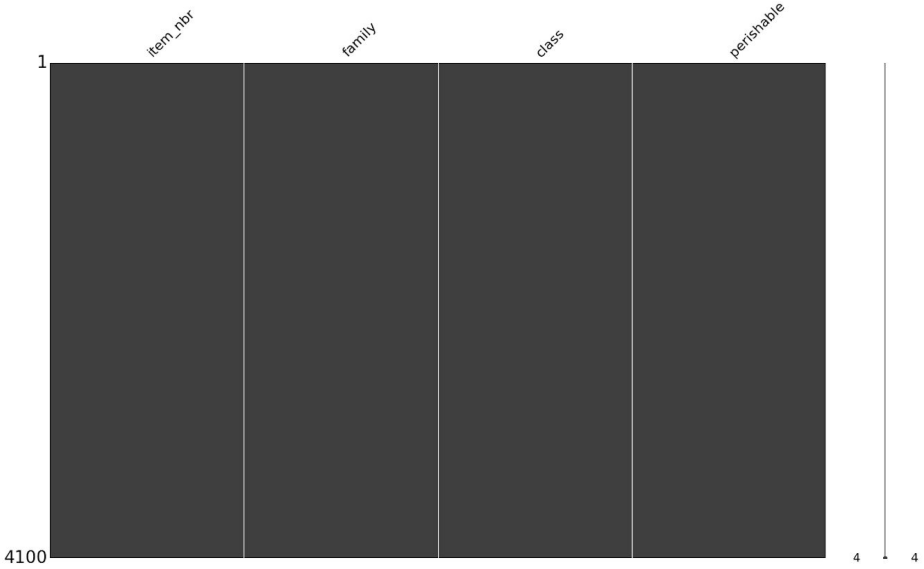

Figure 25: *items.csv*

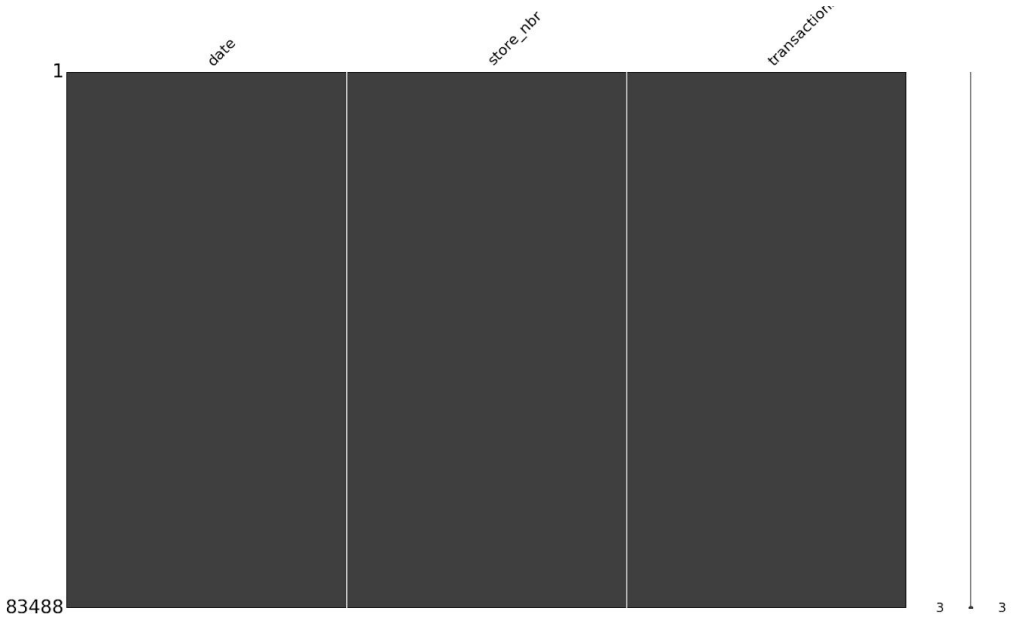

Figure 26: *transactions.csv*

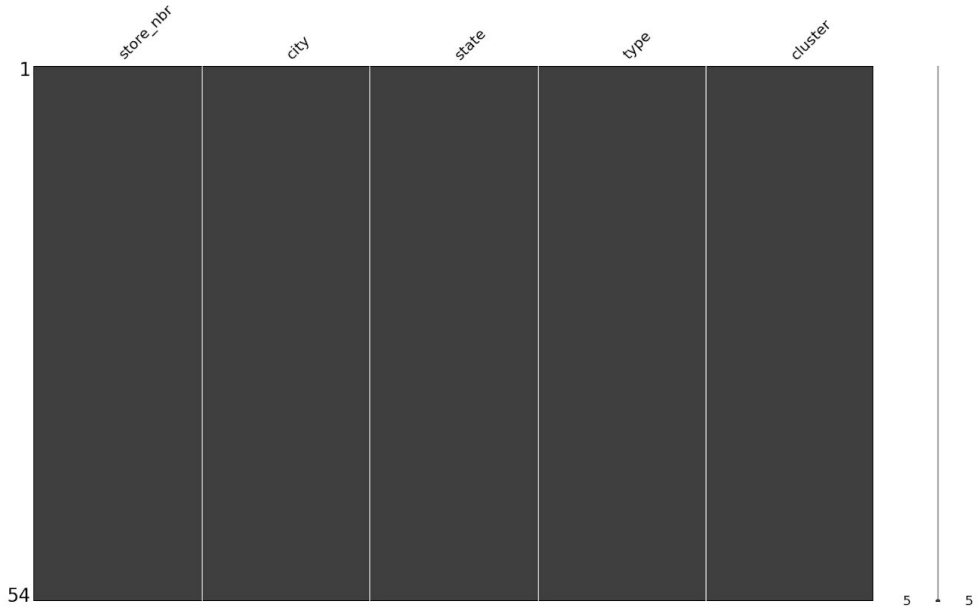

Figure 27: *stores.csv*

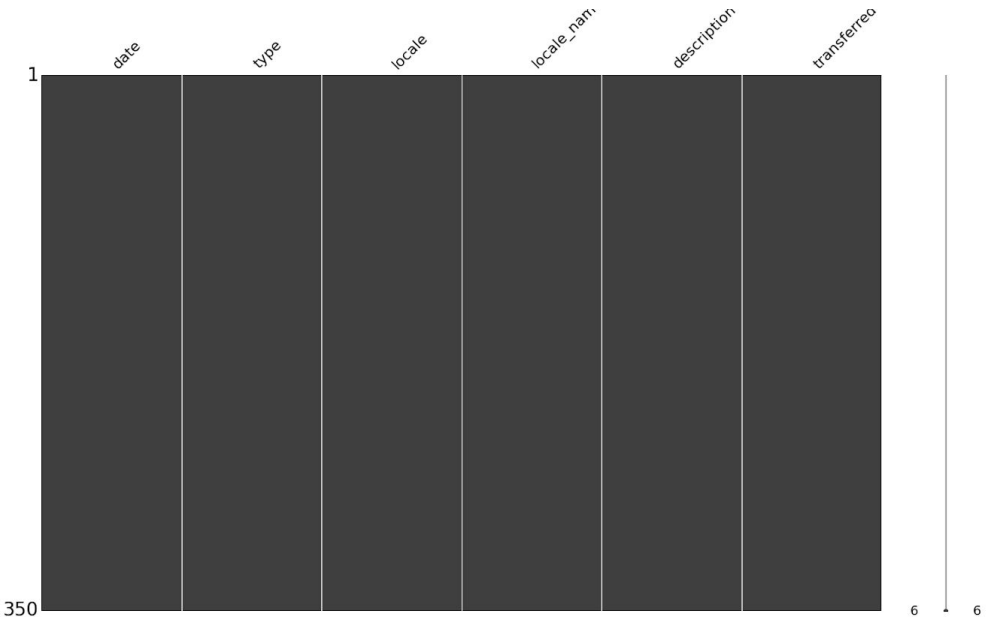

Figure 28: *holidays_events.csv*

Kaggle Submission History (From Latest to Earliest)

nn_lgm_22_1.csv a month ago by Arkar Min add submission details	0.52078	0.51301	☐
nn_lgm_22.csv a month ago by Arkar Min add submission details	0.52028	0.51259	☐
nn_lgm_7_2.csv a month ago by Arkar Min 40-60 (nn_year, lgbm_year)	0.52036	0.51238	☐
nn_lgm_8.csv a month ago by Arkar Min add submission details	0.52028	0.51259	☐
nn_lgm_8.csv a month ago by Arkar Min add submission details	0.52017	0.51252	☐
nn_promo_weight_year_LSTM.csv a month ago by Arkar Min add submission details	0.52389	0.51723	☐
nn_promo_weight_year_4.csv a month ago by Arkar Min add submission details	0.54145	0.52981	☐
modified_weight_promo_year_2.csv a month ago by Arkar Min add submission details	0.52106	0.51339	☐

nn_promo_weight_year_3.csv a month ago by Arkar Min add submission details	0.52323	0.51590	☐
nn_promo_weight_year_2.csv a month ago by Arkar Min add submission details	0.52280	0.51477	☐
nn_lgm_7_1.csv a month ago by Arkar Min add submission details	0.52088	0.51278	☐
nn_lgm_7.csv a month ago by Arkar Min add submission details	0.52036	0.51238	☐
modified_weight_promo_year_1.csv a month ago by Arkar Min add submission details	0.52110	0.51394	☐
nn_lgm_6.csv a month ago by Arkar Min add submission details	0.52053	0.51250	☐
nn_promo_weight_year.csv a month ago by Arkar Min add submission details	0.52340	0.51505	☐
modified_weight_promo_year.csv a month ago by Arkar Min add submission details	0.52099	0.51320	☐
nn_lgm_5.csv a month ago by Arkar Min add submission details	0.52297	0.51386	☐
nn_promo_weight_week_2.csv a month ago by Arkar Min add submission details	0.52543	0.51581	☐
nn_promo_weight_week_2.csv a month ago by Arkar Min add submission details	0.52543	0.51581	☐
modified_weight_promo_week_2.csv a month ago by Arkar Min add submission details	0.52366	0.51474	☐
nn_lgm_4.csv a month ago by Arkar Min add submission details	0.52654	0.51642	☐
nn_promo_weight_week_1.csv a month ago by Arkar Min add submission details	0.52904	0.51830	☐
modified_weight_promo_week_1.csv a month ago by Arkar Min add submission details	0.52694	0.51742	☐
nn_lgm_3.csv a month ago by Arkar Min add submission details	0.52654	0.51677	☐

nn_promo_weight_week.csv a month ago by Arkar Min add submission details	0.52891	0.51864	☐
modified_weight_promo_week.csv a month ago by Arkar Min add submission details	0.52711	0.51761	☐
lgb_one_step.csv a month ago by Arkar Min add submission details	0.52356	0.51563	☐
nn_promo_weight_2000.csv a month ago by Arkar Min add submission details	0.53712	0.52922	☐
modified_weight_promo_3.csv a month ago by Arkar Min add submission details	0.53520	0.52821	☐
nn_lgm_2.csv a month ago by Arkar Min add submission details	0.53603	0.52743	☐
nn_lgm_1.csv a month ago by Arkar Min add submission details	0.53562	0.52717	☐
nn_promo_weight_1000.csv a month ago by Arkar Min add submission details	0.53751	0.52854	☐
nn_promo_weight.csv a month ago by Arkar Min add submission details	0.53759	0.52863	☐
nn_weight.csv a month ago by Arkar Min add submission details	0.58507	0.55121	☐
nn.csv a month ago by Arkar Min add submission details	0.58513	0.55117	☐
modified_weight_promo_2.csv a month ago by Arkar Min LGBM	0.53585	0.52771	☐
nn_weight.csv a month ago by Arkar Min add submission details	Error	Error	☐
modified_weight_promo_1.csv a month ago by Arkar Min add submission details	0.54592	0.53510	☐
modified_weight_promo.csv a month ago by Arkar Min add submission details	0.54834	0.53727	☐
modified_weight.csv a month ago by Arkar Min add submission details	0.58512	0.55203	☐

modified_1.csv a month ago by Arkar Min add submission details	0.58508	0.55214	☐
nn_lgm.csv a month ago by Arkar Min add submission details	0.58487	0.55123	☐
nn.csv a month ago by Arkar Min add submission details	0.58531	0.55152	☐
modified.csv a month ago by Arkar Min LSTM Code without weight without promo	0.58520	0.55249	☐
nopromo.csv a month ago by Arkar Min LSTM Code with Weight without promo	0.58086	0.54792	☐
simple_lgbm.csv a month ago by Arkar Min add submission details	1.54103	1.51410	☐
lgb.csv a month ago by Arkar Min Testing the result got by igbo-starter code	0.54032	0.52988	☐