Machine Learning Project

Tejaswini Dhupad

Chun Han Li

June, 2022

TABLE OF CONTENT

The project code can be viewed and accessed on GitHub with the link here.

INTRODUCTION

In this project, we will be predicting the unit sales for thousands of items sold at different Favorita stores located in Ecuador. The training data includes dates, store and item information, whether that item was being promoted, as well as the unit sales [1].

File Description and Data Field Information:

train.csv: The training data, comprising time series of features '*store_nbr'*, '*family'*, and '*onpromotion'* as well as the target sales. '*store_nbr'* identifies the store at which the products are sold. '*family'* identifies the type of product sold. '*sales'* give the total sales for a product family at a particular store at a given date. Fractional values are possible since products can be sold in fractional units (1.5 kg of cheese, for instance, as opposed to 1 bag of chips). '*onpromotion'* gives the total number of items in a product family that were being promoted at a store at a given date.

test.csv: The test data, having the same features as the training data. You will predict the target sales for the dates in this file. The dates in the test data are for the 15 days after the last date in the training data.

stores.csv: Store metadata, including '*city'*, '*state'*, '*type'*, and '*cluster'*. '*cluster'* is a grouping of similar stores.

oil.csv: Daily oil price. Includes values during both the train and test data timeframes. (Ecuador is an oil-dependent country and its economic health is highly vulnerable to shocks in oil prices).

holidays_events.csv: Holidays and Events, with metadata. NOTE: Pay special attention to the transferred column. A holiday that is transferred officially falls on that calendar day, but was moved to another date by the government. A transferred day is more like a normal day than a holiday. To find the day that it was actually celebrated, look for the corresponding row where type is Transfer. For example, the holiday Independencia de Guayaquil was transferred from 2012-10-09 to 2012-10-12, which means it was celebrated on 2012-10-12. Days that are type Bridge are extra days that are added to a holiday (e.g., to extend the break across a long weekend). These are frequently made up by the type Work Day which is a day not normally scheduled for work (e.g., Saturday) that is meant to pay back the Bridge. Additional holidays are days added to a regular calendar holiday, for example, as typically happens around Christmas (making Christmas Eve a holiday)[1].

Additional Notes: Wages in the public sector are paid every two weeks on the 15th and on the last day of the month. Supermarket sales could be affected by this. A magnitude 7.8 earthquake struck Ecuador on April 16, 2016. People rallied in relief efforts donating water and other first need products which greatly affected supermarket sales for several weeks after the earthquake [1].

The analysis goal for this report is summarised into the following research questions:

• What are the factors that influence daily grocery sales?

• How do these factors impact daily grocery sales?

• How can we model these factors to forecast future grocery sales?

PROBLEM STATEMENT

To predict the unit sales for thousands of items sold at different Favorita stores located in Ecuador. We would be building a time-series model to predict the sales of the store.

PHASE I: DATA SELECTION

The dataset that we are working on is downloaded from a Kaggle competition with a link.

PHASE II: DATA TRANSFORMATION

A. Reading the files: In our dataset for Favorita stores located in Ecuador, we have 5 main files namely:

1. *holiday_events.csv*
2. *oil.csv*
3. *stores.csv*
4. *train.csv*
5. *transactions.csv*

B. Merging all the CSV into *train.csv* based on respective parameters:

Each CSV file is a table and all tables follow database normalisation. In order to use them and do further analysis. There are two main parameters in *train.csv* that can be used as primary keys, '*date'* and '*store_nbr'*, and we merge other CSV files with the help of left join.

```
# Using merge function by setting how='left'
output = pd.merge(data0, data1,
                  on='date',
                  how='left')
```

We also dropped some of the columns that were not necessary like the *'Description'* column. At the end of merging, there are **3054348 rows**(data) and **15 columns**(attributes).

C. Dealing with missing values:

During left joins, we create a lot of missing values. Therefore, we replace empty/missing values with NULL.

```
df = pd.DataFrame(data1)
# Replacing the blank/empty value with NULL
df2=df.fillna("NULL")
```

D. Replace the NULL value with some more meaningful value:

```
In [11]: #Checking for NULL values in the dataset
         data.isnull().sum()
```

```
Out[11]: id                   0
         date                 0
         store_nbr            0
         family               0
         sales                0
         onpromotion          0
         dcoilwtico      955152
         city                 0
         state                0
         type_x               0
         cluster              0
         type_y         2551824
         locale         2551824
         locale_name    2551824
         transferred    2551824
         dtype: int64
```

1. Here, '*dcoilwtico*' (oil price) column has a lot of NULL values, because there are no oil prices during the weekend and holidays. We will need to fill this column using average or shift values depending on our use case.

```
df['lagoil_1_dcoilwtico'] = df['dcoilwtico'].shift(1)
df['lagoil_2_dcoilwtico'] = df['dcoilwtico'].shift(2)
df['lagoil_3_dcoilwtico'] = df['dcoilwtico'].shift(3)
df['lagoil_4_dcoilwtico'] = df['dcoilwtico'].shift(4)
df['oil_week_avg'] = df['dcoilwtico'].rolling(7).mean()
```

2. The last four attributes are related to holidays, not every day is a holiday, therefore, we have a lot of NULL here. All these NULL values can be considered "no holidays". So, using NULL is enough and we don't need to change anything.

E. Extracting the value from "*date*" column:

Since this is a time series problem, most of the time, the values follow a certain pattern. Our goal is to find this pattern. When a certain year passes, it never appears again in our testing data, therefore, we can simply discard the year value. However, months have a one-year cycle, and also days of the week occur weekly. We extract the month and day of the week for the date and make it another attribute.

```
date_separation['month'] =
pd.DatetimeIndex(date_separation['date']).month
date_separation['dayofweek'] =
pd.DatetimeIndex(date_separation['date']).dayofweek
```

F. Correcting holidays_events CSV:

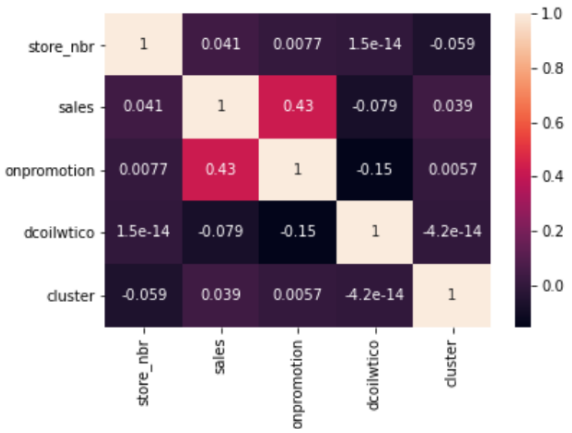
Defined the function for holidays_events and added features to it like - Non-transferable events, removing duplicate events, adding event type, adding New Year, matching holidays_events and stores, adding Easter, adding Closure days and merging the *holidays_events.csv* file with final dataframe.

```
# Non-transferred events
    df.loc[297, 'transferred'] = df.loc[297, 'transferred'] = False
    df = df.query("transferred!=True")

# Adding New Year
    df['firstday'] = df.description_x.apply(lambda x: 1 if x=='Primer
dia del ano' else 0)

# Adding Easter
    df.loc[df.date.isin(['2017-04-16', '2016-03-27', '2015-04-05',
'2014-04-20', '2013-03-31']), 'isevent'] = df.isevent.apply(lambda x:
'y')
    df.loc[df.date.isin(['2017-04-16', '2016-03-27', '2015-04-05',
'2014-04-20', '2013-03-31']), 'event_type'] = df.event_type.apply(lambda
x: 'Holiday')

# Adding closure days
    df['isclosed'] = df.groupby(by=['date',
'store_nbr'])['sales'].transform(lambda x: 1 if x.sum()==0 else 0)
    df.loc[(df.date.dt.year==2017) & (df.date.dt.month==8) &
(df.date.dt.day>=16) , 'isclosed'] = df.isclosed.apply(lambda x: 0)
    df.loc[df.date.isin(['2017-01-01']), 'isevent'] =
df.isevent.apply(lambda x: 'n')
```

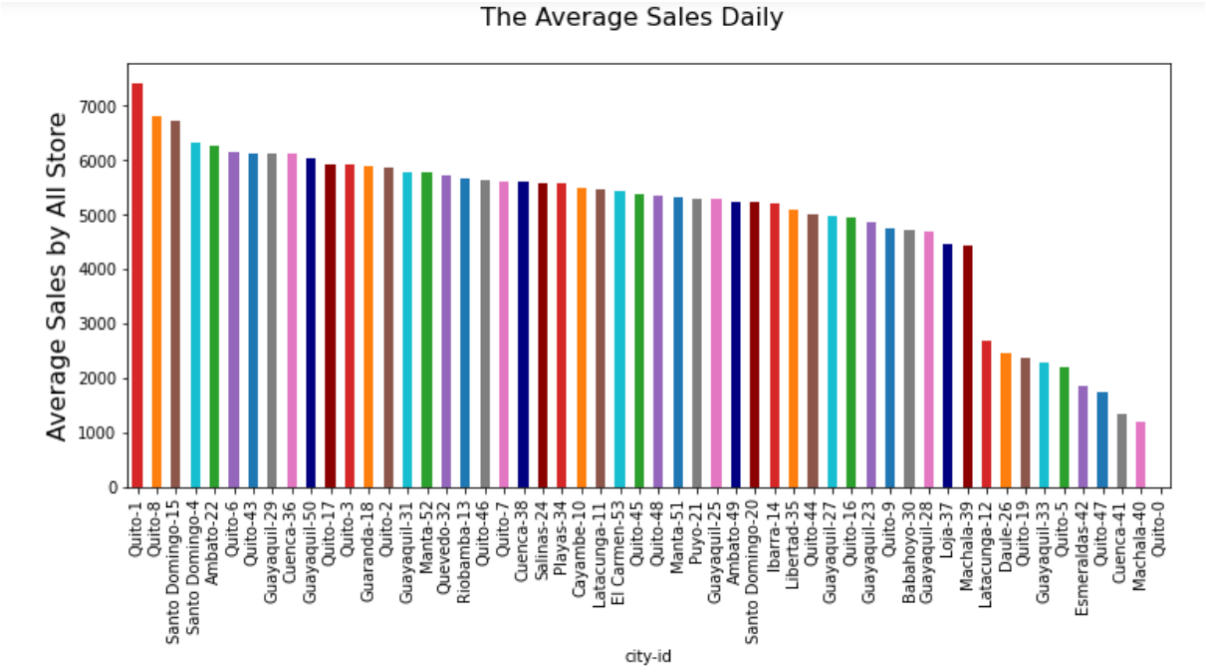
PHASE III: DATA PRE-PROCESSING

A. Print out some basic analysis chart, and general average value:

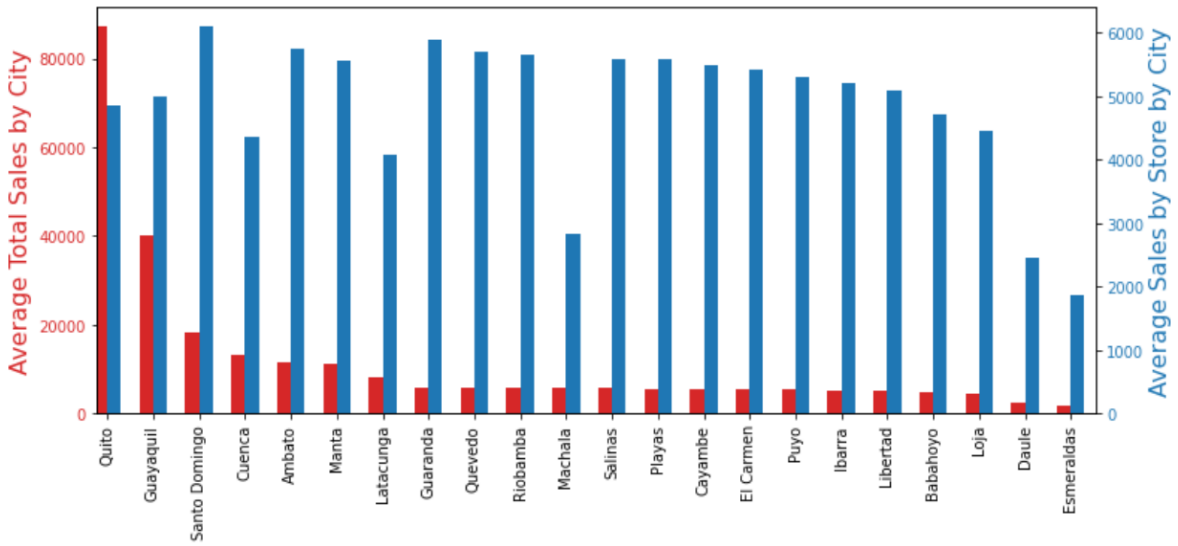
1. Relationship analysis using correlation matrix: The correlation matrix shows a strong relation between *'sales'* and *'onpromotion'*. It infers that when there is a promotional sale going on in a store, the sales are also high.



2. Daily Average Sales by all stores: There are 54 stores at different locations (including different cities and states). However, the averages of sales per day are not similar. Most effective stores are located in Quito city. There are also about 9 stores running badly. An available suggestion is to shut down these stores or look for effective solutions to improve the business.

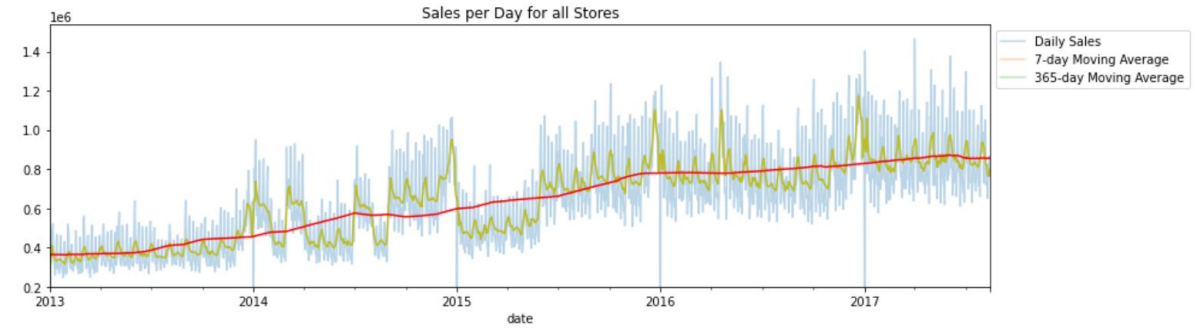


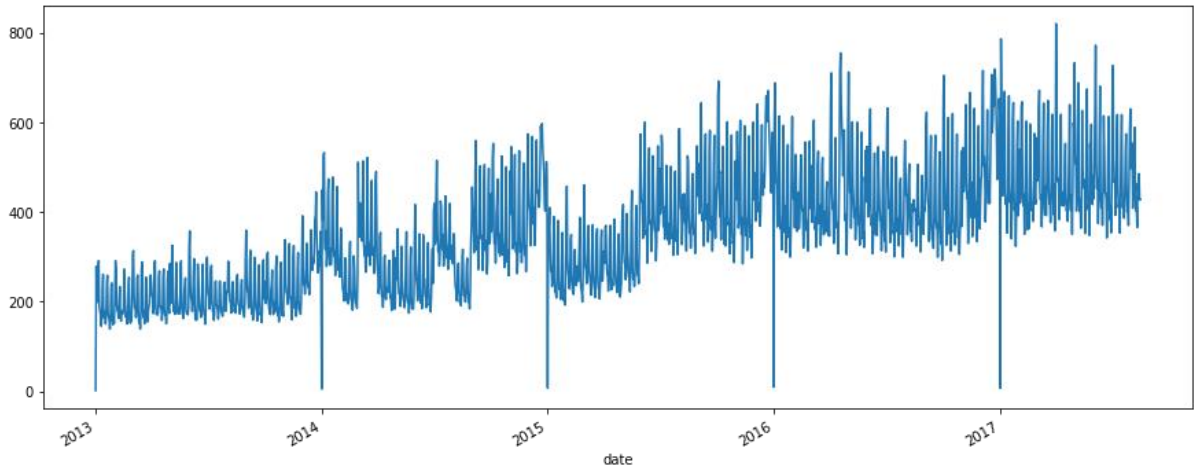
3. Average Sales by City: Looking at the graph, we can see that Santo Domingo has the highest daily average sales w.r.t city. It is followed by Guaranda city. Whereas, Quito also has the high daily-average sales both per city and per store.



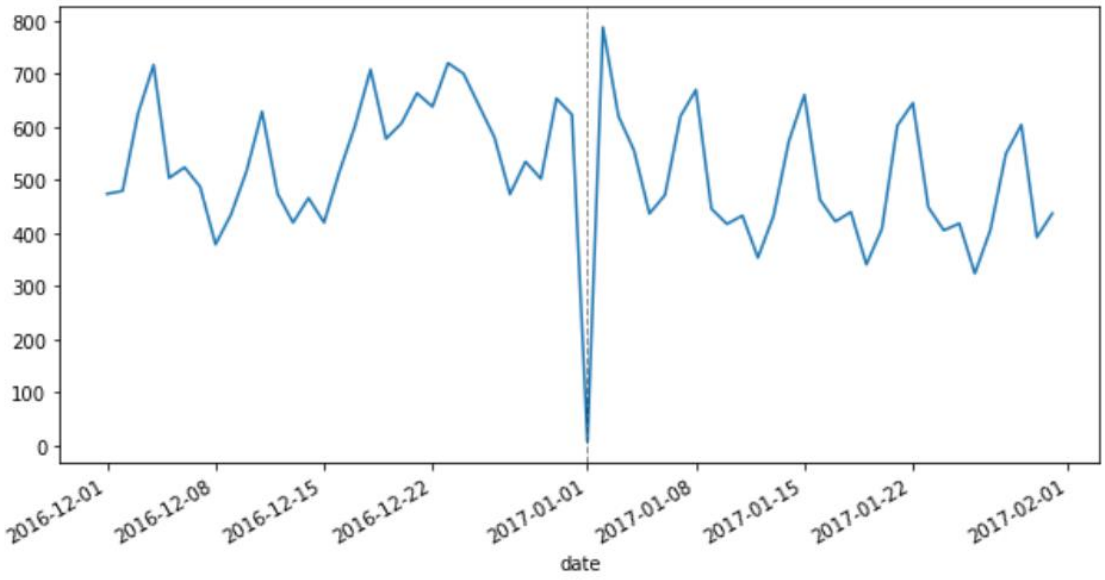
B. Analysis base on different attribute values by time:

1. **Analysing the Sales over the years**: Based on the graph, we could see that the sales are always peaking during New Year's Eve.

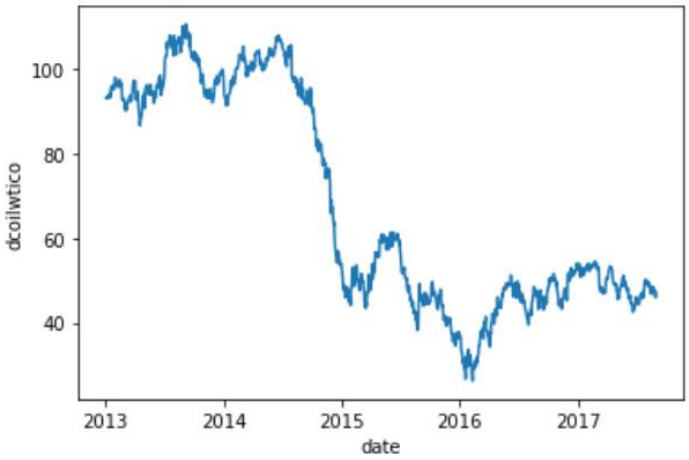




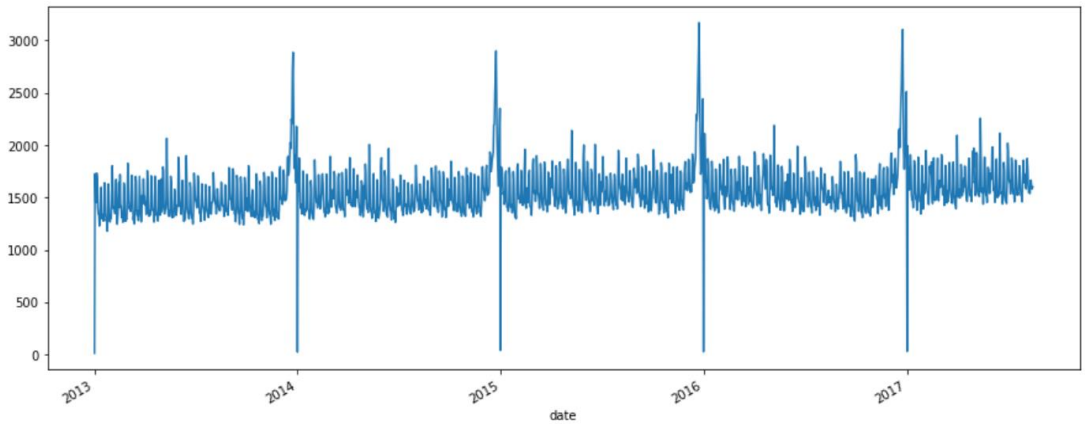
2. **Analysing the Sales data from 2016-12-01 to 2017-02-01**: Based on the previous step, we want to know the time the peak happened.



3. Analysing the oil prices over the years: Ecuador is heavily dependent on oil; therefore, changes in oil price may strongly affect its economy and impact grocery sales. The line plot of oil price below shows that there is a dramatic drop from Jun 2014 to Jan 2015, as well as a small increase from Feb 2015 to Jun 2015. Interestingly, this trend of oil price negatively correlates (-0.67 by spearman method) with grocery sales. It is likely that the prices of some grocery items (i.e., imported items) could tend to increase when oil price drops, which may drive people to do more grocery shopping before the prices start to rise.



4. Analysing the Transactions over the years: Just to see the transaction pattern of the sales in the stores from 2013 to 2017.



C. GroupBy value with different attributes:

```
str_type
A    1.180273e+08
D    1.179447e+08
C    5.380120e+07
B    5.065891e+07
E    2.264005e+07
Name: sales, dtype: float64
```

```
cluster
14    5.076277e+07
6     4.127568e+07
11    3.679107e+07
8     3.581573e+07
10    3.064024e+07
13    2.522077e+07
3     2.430911e+07
5     2.010697e+07
1     1.787748e+07
15    1.675057e+07
4     1.543991e+07
2     1.170915e+07
17    1.036647e+07
9     9.642782e+06
12    6.443996e+06
7     6.297535e+06
16    3.621918e+06
Name: sales, dtype
```

```
family
GROCERY I                     1.076903e+08
BEVERAGES                     7.980141e+07
PRODUCE                       5.471677e+07
CLEANING                      2.868959e+07
DAIRY                         2.145715e+07
BREAD/BAKERY                  1.292329e+07
POULTRY                       9.196818e+06
MEATS                         8.696725e+06
PERSONAL CARE                 7.498256e+06
DELI                          7.272648e+06
HOME CARE                     7.039159e+06
EGGS                          4.605960e+06
FROZEN FOODS                  4.059254e+06
PREPARED FOODS                2.431409e+06
LIQUOR,WINE,BEER              2.312182e+06
HOME AND KITCHEN I            7.806920e+05
HOME AND KITCHEN II           6.555480e+05
GROCERY II                    5.559580e+05
SEAFOOD                       5.341888e+05
CELEBRATION                   3.304340e+05
LAWN AND GARDEN               2.829960e+05
LADIESWEAR                    2.761220e+05
PLAYERS AND ELECTRONICS       2.626510e+05
PET SUPPLIES                  1.854080e+05
AUTOMOTIVE                    1.663070e+05
SCHOOL AND OFFICE SUPPLIES    1.654760e+05
MAGAZINES                     1.520660e+05
LINGERIE                      1.509240e+05
BEAUTY                        1.295810e+05
HARDWARE                      3.081800e+04
HOME APPLIANCES               9.644000e+03
BOOKS                         6.438000e+03
BABY CARE                     5.912000e+03
Name: sales, dtype: float64
```

PHASE IV: MODEL SELECTION AND TRAINING

In order to use the models here, we have to change all the non-numeric data to numeric data. One way to do this is applying One-Hot Encoding to all the non-numeric data. Hence, we need to use the '***astype'*** and '***get_dummies'*** function to achieve this. First, we call the '*astype'* function to make sure all the numeric data is either int, float, or bool. Meanwhile, use the same function to make sure all the non-numeric data is in 'str' type. Next, call '*get_dummies'* it will transfer the data attribute into One-Hot Encoding style. To illustrate this, see the example below:

	A	B
x	a1	b1
y	a2	b1
z	a2	b2

After calling '*get_dummies'* function, it will transform the input data in following way.

	A_a1	A_a2	B_b1	B_b2
x	1	0	1	0
y	0	1	1	0
z	0	1	0	1

The code should like this:

```
df['year'] = df.index.year.astype('int')
df['quarter'] = df.index.quarter.astype('int')
```

```
df = pd.get_dummies(df, columns=['year'], drop_first=True)
df = pd.get_dummies(df, columns=['quarter'], drop_first=True)
```

A. Linear Regression (Ridge Regression)

LR is basically a model that explains the relationship between a response variable and one or more explanatory variable a.k.a features. For example, demand and supply, higher the demand higher should be the supply.

The equation of a LR is given by Y= AX + c,
where Y is the dependent variable,
X is the explanatory variable,
A is the slope and
c is the intercept.

Linear-regression models have become a proven way to scientifically and reliably predict the future. Because linear regression is a long-established statistical procedure, the properties of linear-regression models are well understood and can be trained very quickly[2].

Hence start from the LR (easiest) model; the total training time is **6.125 seconds**.

```
from sklearn.linear_model import Ridge
ri = Ridge(alpha=1.0)
model_ri = ri.fit(x_train, y_train)
prediction_ri = model_ri.predict(x_test)
prediction_ri[prediction_ri < 0] = 0
```

The result for this model, **Root Mean Squared Logarithmic Error = 1.98722.** Which is quite unacceptable, obviously this kind of linear regression is too simple to our dataset.

B. Decision Tree Regressor

Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values [2].

The model combines decision trees and linear regression. It first applies a decision tree and then uses linear regression for each node to output the predicted value.

```
from sklearn.tree import DecisionTreeRegressor
model_dt = DecisionTreeRegressor()
model_dt.fit(x_train, y_train)
prediction_dt = model_dt.predict(x_test)
prediction_dt[prediction_dt < 0] = 0
```

The result for this model, **Root Mean Squared Logarithmic Error = 0.75789,** and the total training time is **127.105 seconds**. This is a big improvement over the last one, which means we're heading in the right direction to make our predictions more accurate.

C. Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned [3].

This model is an advanced version of DecisionTreeRegressor. The differences between the two models are the same as the differences between decision trees and random forests. However, the training took much longer than we expected, more than an hour and we couldn't get the result, and it doesn't look like we can train the model using our own laptop.

```
from sklearn import ensemble
rf = ensemble.RandomForestRegressor(
    n_estimators=300,
    max_depth=2,
    random_state=42
)
model_rf = rf.fit(x_train, y_train)
prediction_rf = model_rf.predict(x_test)
prediction_rf[prediction_rf < 0] = 0
```

We concluded that to use RandomForestRegressor, we need to pre-process the data again using another method. Since we have over 3 million rows of data in our files, we need to somehow split the data and train the model separately to reduce the time.

To find the best way to split the data, we need to use the result of the *GroupBy* value with different attributes in the data pre-processing part.

D. Linear Regression with Random Forest Regressor

As mentioned before, in this model we will split our dataset, and train a lot of small models based on different values. Also, we would like to try to combine two models we have used before and combine the prediction of these two models using weighted average.

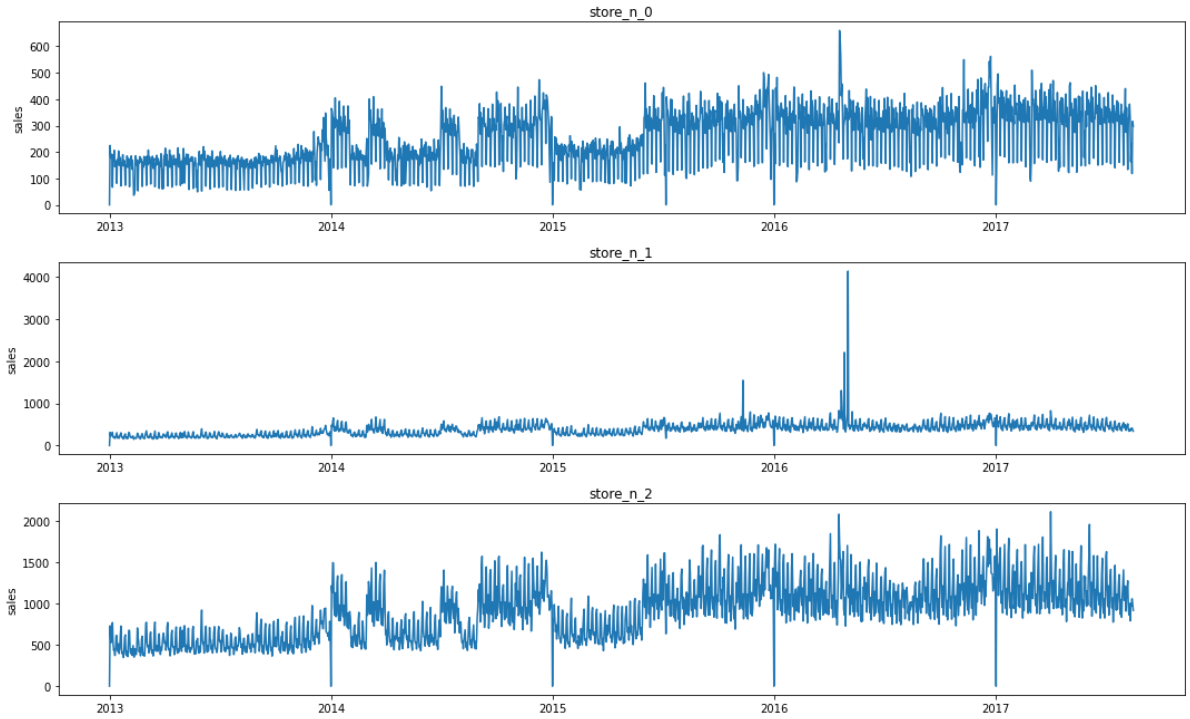
Step 1: Setting the Dates: We decided to make the training start on 2016-06-01 because of the following reasons:

1. In the early years the patterns are clearly different from the ones in the last years regarding '*Sales*' and '*Oil*';
2. We wanted to avoid some peaks during May and April'2016, since we encountered performance degradation when including them;
3. We wanted to include the same period for the test dataset.

```
date_list = {}
date_list['date_start_train'] = '2013-01-01'
date_list['date_end_train'] = '2017-08-15'
date_list['date_start_test'] = '2017-08-16'
date_list['date_end_test'] = '2017-08-31'
date_list['date_start_fore'] = '2016-06-01'
```

```
diff_train = (pd.Timestamp(date_list['date_end_train']) -
pd.Timestamp(date_list['date_start_fore'])).days
diff_test = (pd.Timestamp(date_list['date_end_test']) -
pd.Timestamp(date_list['date_start_fore'])).days
```

Step 2: Analysing stores sales by different '*store_nbr'*, as shown below, each store has a very different pattern. Therefore, we decided to split our training by different '*store_nbr'*.



We tried to analyse the sales of each '*store_nbr'* over the years starting from 2013 to 2017. Hence, the above graph helps us understand the sales at each store over the time period.

```
# Adding features to orig_stores
df['uniquestore'] = df.city.apply(lambda x: 0 if x in ['Quito',
'Guayaquil', 'Santo Domingo', 'Cuenca', 'Manta', 'Machala',
'Latacunga', 'Ambato'] else 1)
df['newstore'] = df.store_nbr.apply(lambda x: 1 if x in [19, 20, 21,
28, 35, 41, 51, 52] else 0)

# Merging orig_stores, orig_test and orig_train
df = pd.concat([orig_train, orig_test], axis=0).merge(df,
on=['store_nbr'], how='left')
df = df.rename(columns={'type' : 'store'})
```

Step 3: The model implementation

A. Splitting the training and test dataset

```
idx_train, idx_test = train_test_split(orig_df.index,
test_size=test_size, shuffle=False)
```

```
    X_train, X_test = X.loc[idx_train, :], X.loc[idx_test, :]
    y_train, y_test = y.loc[idx_train], y.loc[idx_test]

    return X_train, y_train, X_test, y_test
```

B. Adjust type
C. One-Hot encoding
D. Defining Deterministic process

```
fourierA = CalendarFourier(freq='A', order=5)
fourierM = CalendarFourier(freq='M', order=2)
fourierW = CalendarFourier(freq='W', order=4)

dp = DeterministicProcess(index=df.index,
                          order=1,
                          seasonal=False,
                          constant=False,
                          additional_terms=[fourierA, fourierM,
fourierW],
                          drop=True)
dp_df = dp.in_sample()
df = pd.concat([df, dp_df], axis=1)
```

E. Handling the potential outlier values

```
df['outliers'] = df.sales.apply(lambda x: 1 if x>30000 else 0)
df.drop(columns=['daysinmonth', 'month', 'city'], inplace=True)
```

F. Selecting the features for the model

```
cols =
df.columns[df.columns.str.match(r'year|quarter|event|isevent|isclos
ed|oil\_week|dcoilwtico|week|isweek|startschool|sin|cos|trend')]
X = df.loc[:,cols]
X = X.groupby(by='date').first()
```

G. Splitting the train, test and log transformation data

```
X_train, y_train, X_test, y_test = split_func(y, X, np.log1p(y),
end_df, n)
```

H. Training Linear Regression model for each store (training by different '*store_nbr'*)

```
for i in orig_df.store_nbr.unique():
    model = LinearRegression()
    model.fit(X_train, y_train)
    lr_pred_train_y = model.predict(X_train)
    lr_pred_test_y = model.predict(X_test)
```

I. Training Random Forest Regressor model for each store (training by different '*store_nbr'*)

```
for i in orig_df.store_nbr.unique():
    model = RandomForestRegressor(n_estimators=320, random_state=0)
    model.fit(X_train, y_train)
    rf_pred_train_y = model.predict(X_train)
    rf_pred_test_y = model.predict(X_test)
```

J. Collecting the result of the two models using weighted average.

```
st_pred_train_y = lr_pred_train_y * 0.4 + rf_pred_train_y * 0.6
st_pred_test_y = lr_pred_test_y * 0.4 + rf_pred_test_y * 0.6
```

The result for this model, **Root Mean Squared Logarithmic Error = 0.40356**, 54 small models have been trained, and the total training time is **173.84 seconds**.

Conclusion: We started with the Linear Regression model for implementation but it was not giving us satisfactory results. Hence, we tried with the Decision Tree Regressor, indeed the result improved a lot. Next, we think we can improve more by applying more complex models. There is the Random Forest model which by comparison is far better than Decision Tree, we tried implementing it but the execution was very slow given that we had merged all our data in one single file. Therefore, to reduce the execution time and improve accuracy we implemented the Linear Regression with Random Forest with a different approach wherein we added the files to the dataframes with the required constraints so that the execution time would be improved and we made a progress from **0.75789 to 0.40356** using almost the same time.

PHASE V: MODEL EVALUATION

The evaluation metric used for comparing the models was Root Mean Squared Logarithmic Error, which is calculated as stated below:

The RMSLE is calculated as:
[\sqrt{ \frac{1}{n} \sum_{i=1}^n \left(\log (1 + \hat{y}_i) - \log (1 + y_i)\right)^2}]
where:

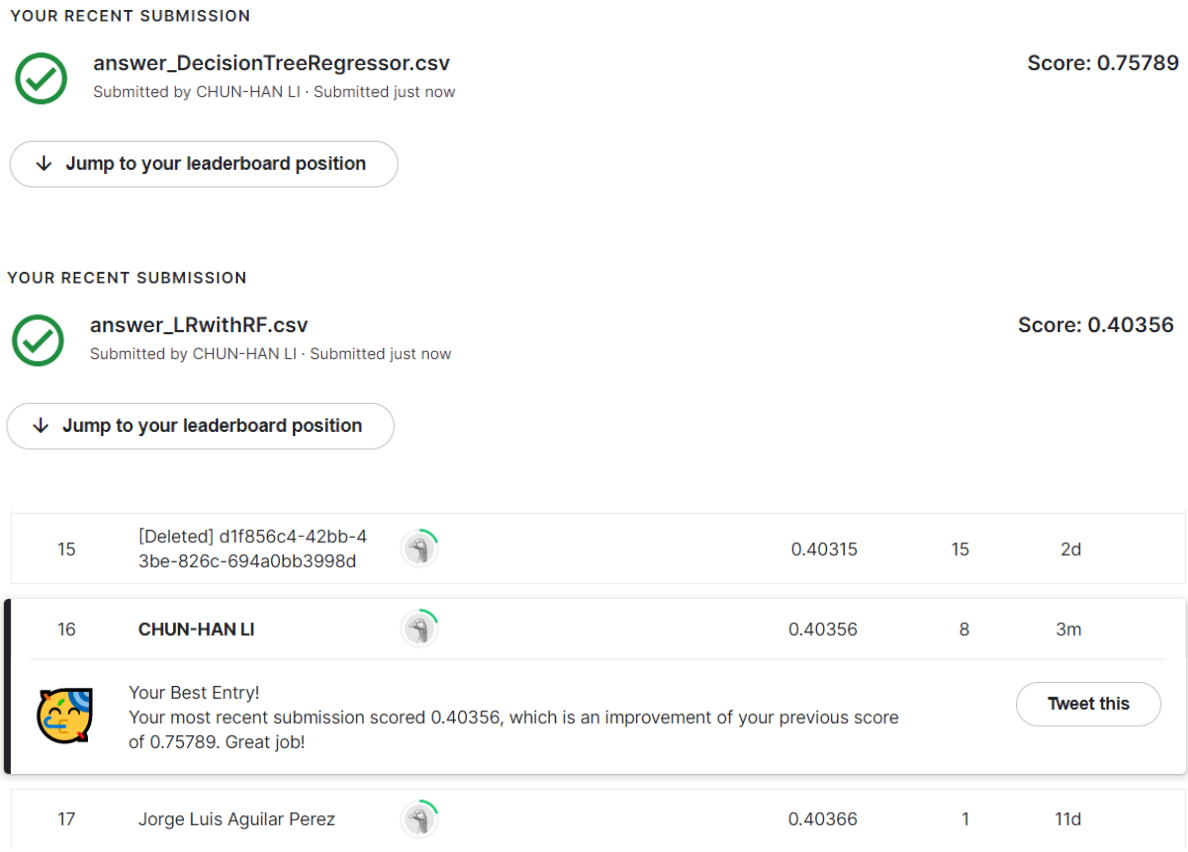
- n is the total number of instances,
- $\hat{y}_i$ is the predicted value of the target for instance (i),
- y_i is the actual value of the target for instance (i), and,
- $\log$ is the natural logarithm.

Below is the submission history and score based on the different models :

YOUR RECENT SUBMISSION

answer_RidgeRegressor.csv — Score: 1.98722
Submitted by CHUN-HAN LI · Submitted 2 days ago

↓ Jump to your leaderboard position



Final Result: Accuracy and execution time.

Model	Execution Time	RMSLE (error)
Ridge Regression	6.125 seconds	1.98722
Decision Tree Regressor	127.105 seconds	0.75789
Random Forest Regressor	> 1 hour	-
LR with RF	173.84 seconds	0.40356

References:

[1] Paul, A. (2022, March 30). *Linear Regression and Random Forest - Analytics Vidhya*. Medium. https://medium.com/analytics-vidhya/linear-regression-and-random-forest-33d4297a186a

[2] Store Sales - Time Series Forecasting | Kaggle. (2021). Store Sales - The Series Forecasting. https://www.kaggle.com/competitions/store-sales-time-series-forecasting/data?select=train.csv

[3] GeeksforGeeks. (2022, May 18). *Python | Decision Tree Regression using sklearn*. https://www.geeksforgeeks.org/python-decision-tree-regression-using-sklearn/

[4] Sathiyanarayanan, M. (2018, October 6). Data Analysis and Forecasting of Grocery Sales in Ecuador. Mithileysh Sathiyanarayanan - Academia.Edu. https://www.academia.edu/37542292/Data_Analysis_and_Forecasting_of_Grocery_Sales_in_Ecuador