Data mining and text mining 2020 project.

Group composed by: Contini Alessandro, Enrico Voltan, Luca Cellamare

The dataset exploration

First, we did some steps of data exploration in order to have a clear idea of the data, its form and eventual visual patterns.
We will refer to the train set as the product with scope equal to 0 sold in the first 30 months, while the test set will be composed by be the products with scope 1, sold through the whole 3 years period.

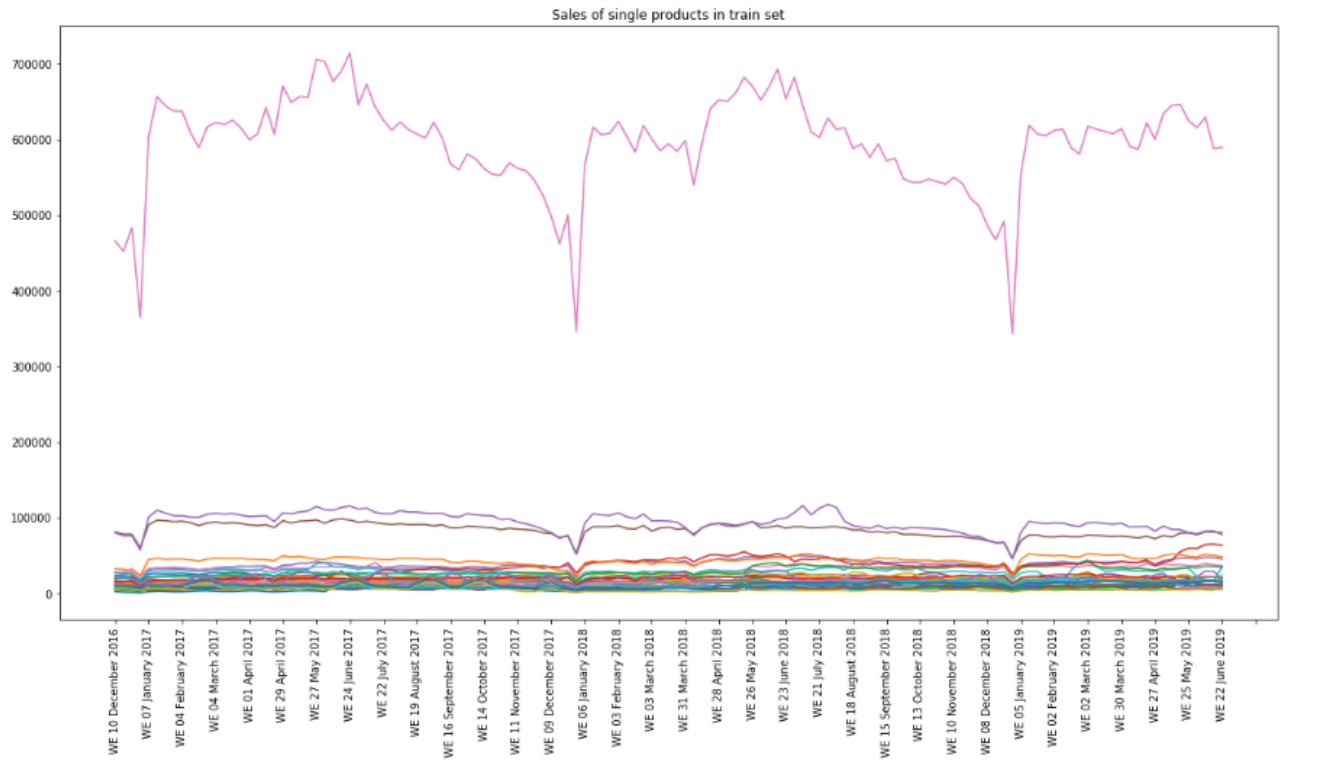


Figure 1, sales of products with scope 0 in the train set

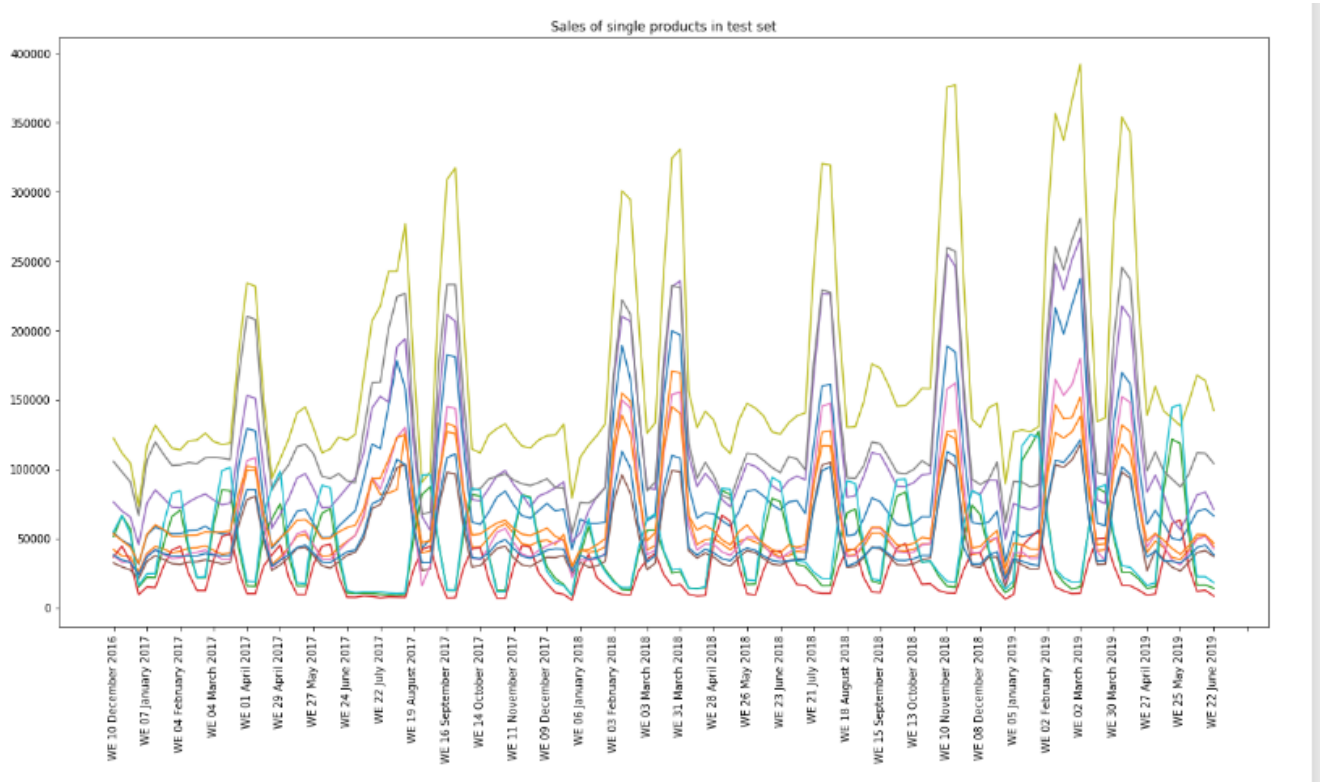


Figure 2, Sales of products with scope equal to 1

Both charts have the sales per week of every single 'sku' (product) as a different line.
We then concentrated our model identification study over the train set, trying to achieve the best performance before moving to the test set.
The first obvious thing is the difference between the train test and the test set. Moreover, we can see a great difference in sales volumes among the products. Another interesting fact noticeable from the train set is that there is some kind of seasonality, a yearly (52 weeks) seasonality exactly.

We then realized a plot over the sum of total sales that shows way better the seasonality.

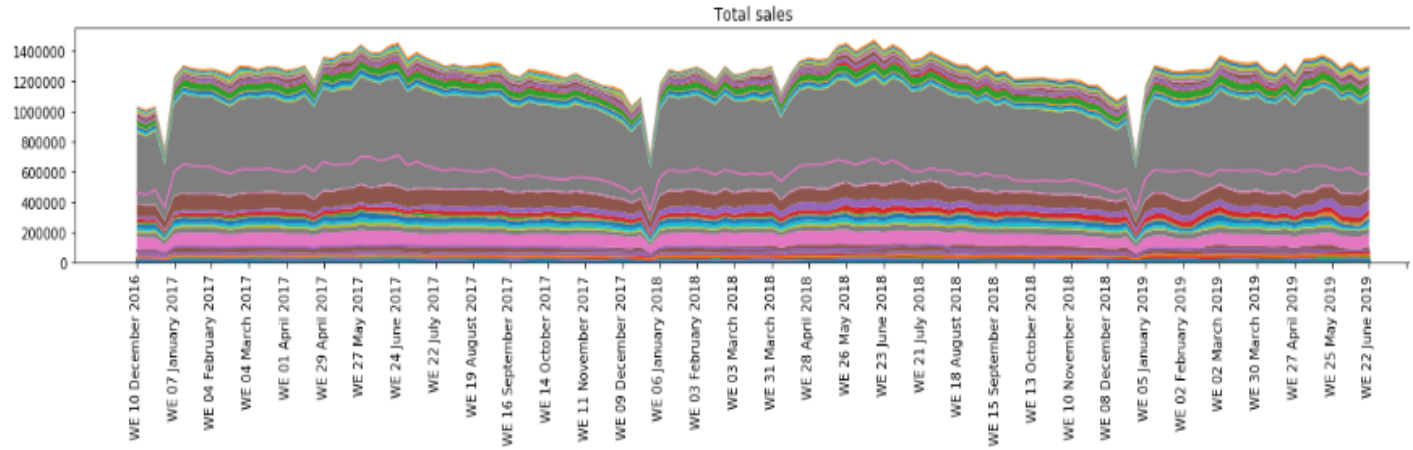


Figure 3, Total sales over the train set

We then supposed another common claim in this kind of problems, that price was related to sales and that there would be complementary products that get sold together often. As it will be shown later, we looked for these correlations through the data set.

	p-value
1356	3.78772e-07
144	3.14009e-07
546	8.10594e-05
549	0.0047431
554	0.006132
686	0.00599379
688	0.00012009
1027	3.21448e-05
1035	0.000200964
1051	1.08838e-08
1058	8.01482e-05
1065	5.96089e-05
1206	0.000108246
1371	8.14682e-05
1516	0.00211623
1554	0.0125335
1603	0.00273161
1608	0.00288056
1633	0.00218606
1732	0.00482241
2249	0.00332108
2396	0.0341242
2401	2.79094e-06
2410	2.40056e-05
2689	0.0404963
2704	0.000705305
2705	0.0410633
2718	0.0164291

Another aspect we decided to focus on was to check whether the sales of the different products were stationary (their statistical properties like mean or standard deviation do not change over time). So we applied the Dickey-Fuller Augmented test and we found out that for only 24 over 43 products it is possible to reject the non stationary hypothesis . We also tried to apply the functions log() and sqrt() to the non stationary products but this transformation didn't make them stationary apart from one product(the one with the 'sku' equal to 2705), for the code please refer to *stationary_check.ipynb.*
Interestingly, we were able to make them stationary, applying the diff() function. But the fact of having for some product, as the target feature, the sales as they are and for others the sales but with the diff() function applied, resulted in difficulties in managing the models. Also because as we will explain later, finally we decided to normalize the features.

Our solution, features explanations

Please refer to files *single_and_average_REPORT.ipynb* and correlations.ipynb. **The first one contains the script used for the final prediction, with a M.A.P.E. of 9.660,** the second file contains scripts used to find correlations through data, motivating our feature selection.

First of all, we saw a deep difference in scale of numbers, for example through sales and prices etc... So, we implemented functions to scale variables. This approach actually improved our M.A.P.E. sensibly.

SCALING FUNCTIONS

In [4]:
```
quantile_transformer = preprocessing.QuantileTransformer(output_distribution='normal', random_state=123)
min_max_scaler = preprocessing.MinMaxScaler()

def scale(feature, scaler = min_max_scaler):
    size = len(feature)
    return scaler.fit_transform(np.array([feature]).reshape(size, 1)).T[0]
def unscale(scaled, original, scaler = min_max_scaler):
    size2 = len(scaled)
    size1 = len(original)
    return scaler.fit(np.array([original]).reshape(size1, 1)).inverse_transform(np.array([scaled]).reshape(size2, 1)).T[0]
```

Figure 4, functions used in both directions for scaling

This preproccesing step also allowed us to investigate better the correlations between the products.
In fact, we were able to discover pattern existing among different products that at first seemed to behave differently but were highly correlated Please refer to *The_importance_of_scaled_features.ipynb.*

Figure 5, on the left you can notice how the product with sku equal to 1608 have a very different scale with respect to the other products. While on the right you can observe how the product 1608 follows the same trends of the other products and shows that rescaling the sales allows to discover existing correlations.

After this step we created a huge data set with many variables we found useful during the training. The features here are obtained after many analysis, for example, we found out that the sales of previous weeks were good predictors of the current sales. On the other hand, prices work in the opposite way, there was a negative correlation between prices and sales. It makes sense, because in the offer/demand laws when prices decrease sales rise.
Another important feature were the groups of products among the train set. As said before, we found out that there were many groups of products whose sales were correlated in the observed years.

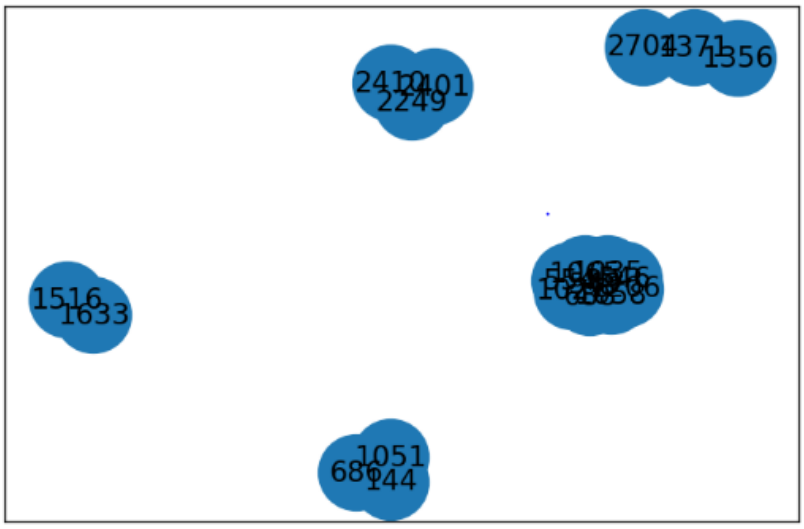


Figure 6, group of products and their skus

Operating an algorithm based over Pearson correlation we found 5 groups among the train set. Later we found out that even the test set was dividable in groups, 2 exactly. Our solution uses the groups predicting them together.

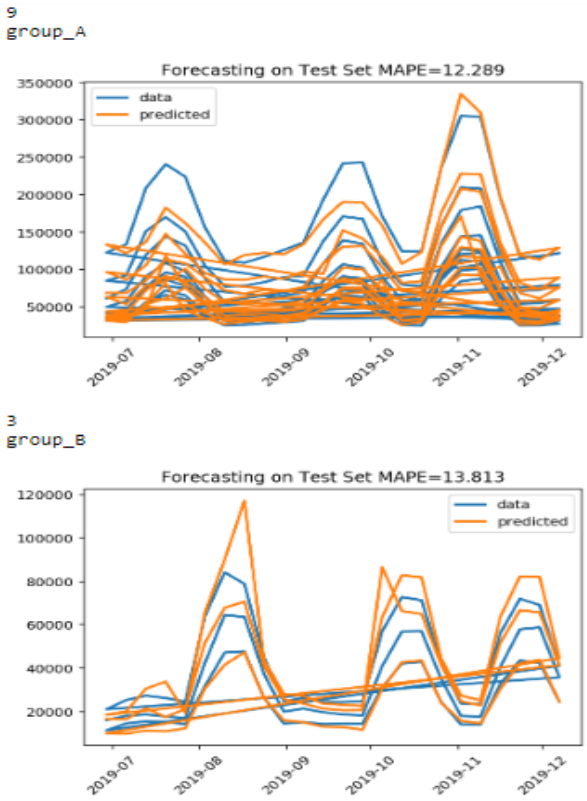


Figure 7, here you can observe an intermediate prediction over the 2 groups of the test set. On the top the name of each group and the number of elements.

```
RangeIndex: 288 entries, 0 to 287
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   scaled_month       288 non-null    float64
 1   group_A            288 non-null    bool
 2   group_B            288 non-null    bool
 3   BRAND2             288 non-null    bool
 4   BRAND4             288 non-null    bool
 5   scaled_dayofyear   288 non-null    float64
 6   scaled_dayofmonth  288 non-null    float64
 7   scaled_weekofyear  288 non-null    float64
 8   scaled_sales1      288 non-null    float64
 9   scaled_sales2      288 non-null    float64
 10  scaled_sales3      288 non-null    float64
 11  scaled_promo       288 non-null    float64
 12  scaled_diff1       288 non-null    float64
 13  scaled_diff2       288 non-null    float64
 14  percentage_diff1   288 non-null    float64
 15  scaled_price       288 non-null    float64
 16  scaled_rolling2    288 non-null    float64
dtypes: bool(4), float64(13)
memory usage: 30.5 KB
```

Figure 8, features used in the test set prediction, moslty the same used for the train set

We will explain some significant features now.
Scaled_month: this feature contains the month without year or week information. This highlighted the annual seasonality. This variable was used together *with scaled_dayofyear/dayofmonth/weekofyear* for reconstructing the exact moment.

Group_A, group_B: those 2 variables contains a boolean that indicates whose correlation group the current 'sku' appertains.
Scaled_sales1/2/3: Sales of the previous weeks, the number indicates how many weeks before the number consider. We found out using autocorrelations and actual predictions M.A.P.E. that sales were relevant until 3 days before, on average.
Scaled_price: As it will be shown later, it is the most important feature in our prediction. We saw a deep correlation between price and sales during the same week: when price decreased sales tended to rise.
Scaled_diffN: variables with the 'diff' contains the difference between the previous week of the one predicted and N weeks before.

Our solution, the actual algorithm

We started by simply predicting the sales for the last six months over the whole dataset and by group of correlation, but the predictions weren't accurate.
We decided to switch to a different approach computing the predictions week-by-week.
This means that our model predicts only the first week of the test set and then integrate the real value on the train set taking that row out the test set. Doing this until the test set is empty gave us a good result, this is observable in the method "predict week" in the solution notebook.
We decided to apply this procedure with three algorithms: XGBoost, Random Forest and Extra Trees and averaging the predictions to obtain a better final result.
We decided to try something more complex and challenging: stacking the models.
Stacking means predicting also the train set, and use these predictions to model how we choose the best prediction on the test set.
Predicting the train set is done following these steps:
- Split the train set in groups of rows, called folds
- Take out the train set the fold to predict
- Fit the models on the remaining rows
- Predict the chosen fold
- Do this for every fold and concatenate the predictions
- Fit a final model on these predictions and use it on the models prediction of the test set

We used XGBoost and Extra Trees as initial models and Random Forest as final model and we did the stacking for every week.
The result was slightly better with respect to the averaging technique but it took a lot more time to compute it so we decided to stick with the latter for the final predictions.

```
XGB MAPE: 10.41891

FOREST MAPE: 10.59336

EXTRA MAPE: 9.73299
```

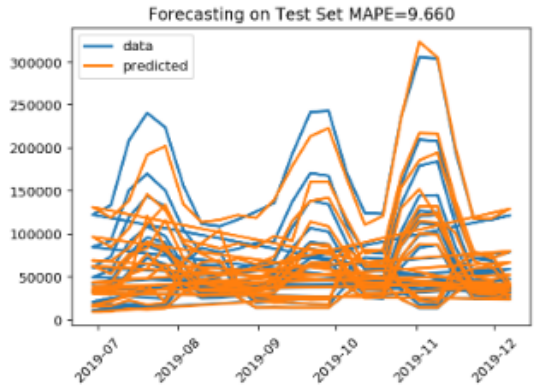
AVARAGE PREDICTIONS:

```
In [20]: final_prediction = 0.3 * forest_pred + 0.3 * xgb_pred + 0.4 *  extra_pred

         mape = mean_absolute_percentage_error(y_test.target, final_prediction)
         mape

Out[20]: 9.659513084535472
```

```
In [21]: plot_scaled_results(final_prediction, y_test, rescale_df)
```



In this extract of our script we can see the predictions score of the three methods. The peculiarity is that combining them led to a better prediction then their single predictions, this because the weighted average that follows gave the importance to a model related to his utility to the final result.
The last chart shows our prediction.
The next plots are made with the SHAP library that aims to explain how our Boost and Trees models compute the predictions. They contains the following information:

Feature importance: Variables are ranked in descending order.
Impact: The horizontal location shows whether the effect of that value is associated with a higher or lower prediction.
Original value: Color shows whether that variable is high (in red) or low (in blue) for that observation.
Correlation: Price as expected works in reverse correlation, the low values (blue) are on the right side,the same for the diff1 (that contains the previous week sales – sales of 2 weeks before) while the sales of the previous week are positively related to the model output.

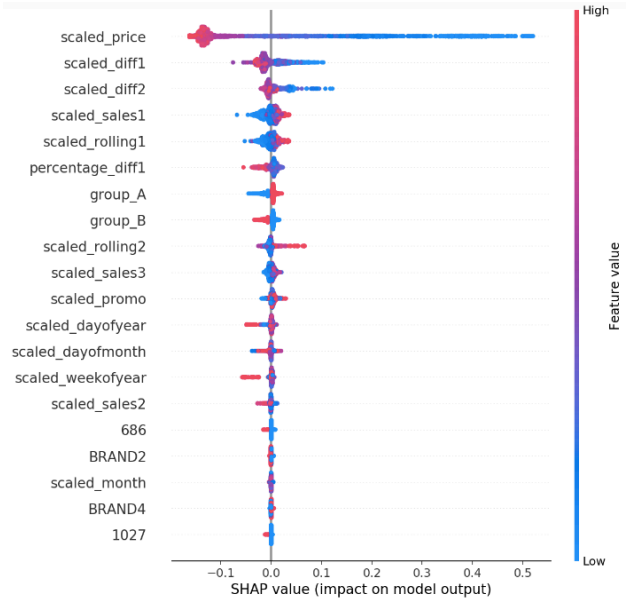


Figure 9, features values for RandomForest.

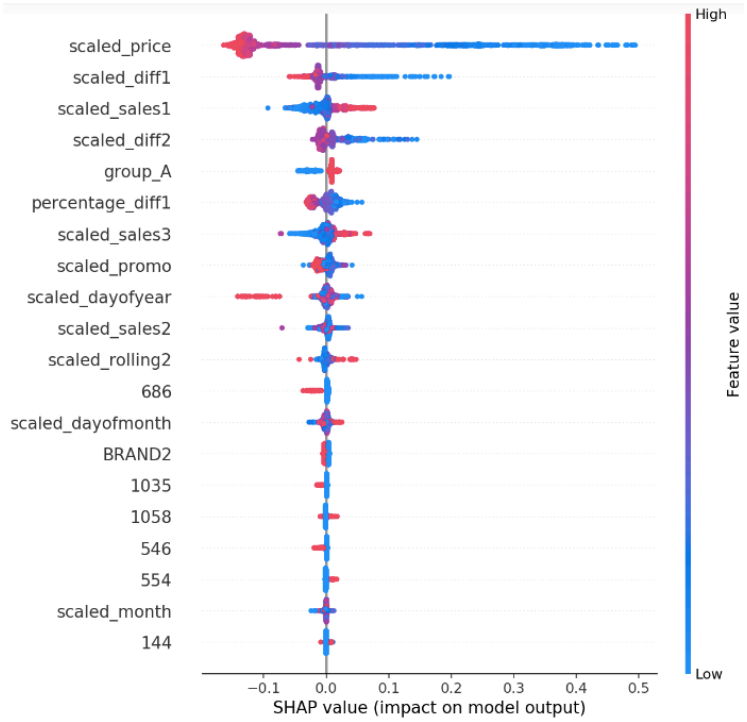


Figure 10. features values for XGBoost.

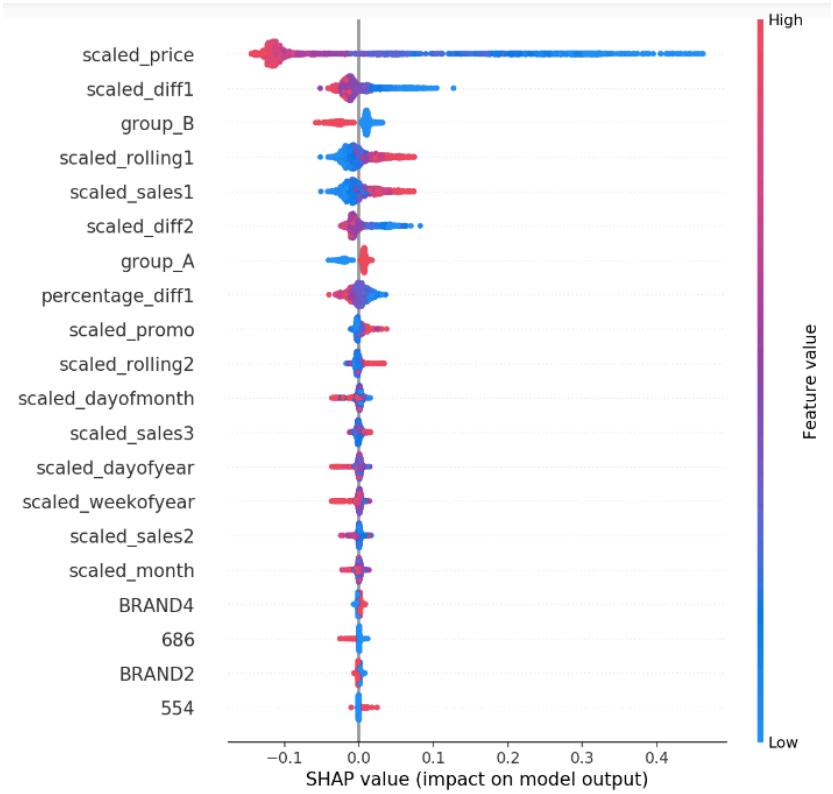


Figure 11, features values for Extra Trees.