KIVOS

Introduction

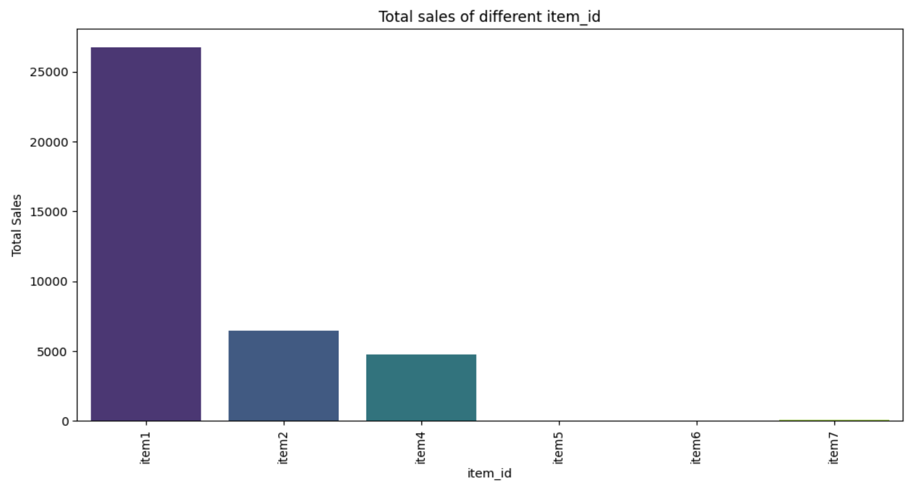
The following demand forecasting problem was developed using the programming language Python3. More specifically, the code was executed in a Jupyter Notebook extension of the VSCodium editor. The python libraries used in this project are the following and can be installed through the package manager pip:

- numpy
- pandas
- matplotlib
- seaborn
- scikit-learn

NOTE: The approach of the solution used in this project was to train a baseline model without many adjustments to the input data and save some metrics from the predictions in order to compare them later on, after the preprocess. Also a seed was used in all the experiments to avoid reproducibility issues.

1. Data loading

In this first step, the goal was to load the data and try to get the first idea of them. The data was processed using *pandas* DataFrames. Using the method *describe()* i was able to see the basic metrics (min, max, mean, std). After merging all the data into one DataFrame some simple plots were created for some visualization and better understanding. A clear note here was that not all items were being sold in all the stores.

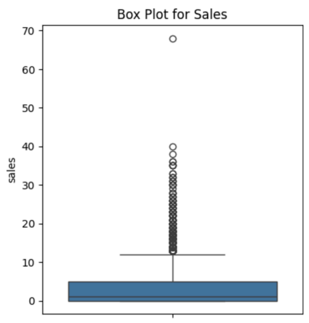


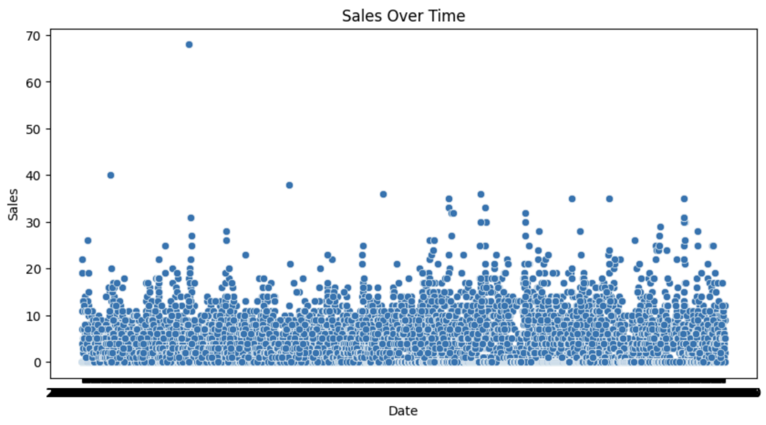
2. Cleansing

For the data cleansing step I tried to identify outliers. As a theoretical approach, below i give a quick explanation of what an outlier could be in our case.

- Sales: Unexpected increases or decreases in sales volumes could be outliers, frequently brought on by mistakes in data input, bad weather, or marketing campaigns.
- Prices: Prices that are very high or low in comparison to normal ranges may indicate exceptional instances, such as clearance prices, or data input errors.

Visual inspection, such as box or scatter plots, was a great way to spot any obvious outliers. Following you will see two simple plots that can easily spot an outlier in sales.





Since it was only one outlier with a value of 68, I pursued to totally remove it from the dataset.

3. Preprocessing and feature engineering

For this step I tested different features in the preprocess of the data. Some of them caused reasonable results and some others did not. So eventually I only kept the features that of course benefited the model. Firstly the *daydt* column was splitted into day, month and year. Since the stores and the items were only a few, the values were transformed to integers in order to have only numeric values before I feed the data to the model. I replaced NaN values from *promo_price* with the *regular_price* and added an extra field of *has_promo* with a value of 0 or 1. Eventually NaN values were kept in the dataset since it was better for the train. Finally a field *discount* was added with the value of the *regular_price* minus the *promo_price*. Some more features that were tested but didn't seem to give better results were *day_of_week* and *is_weekend* since it's known that stores/companies usually have more consumption on weekends.
In case i had access to more data, some suggestions for additional features that could be used to improve the forecast quality are the following:

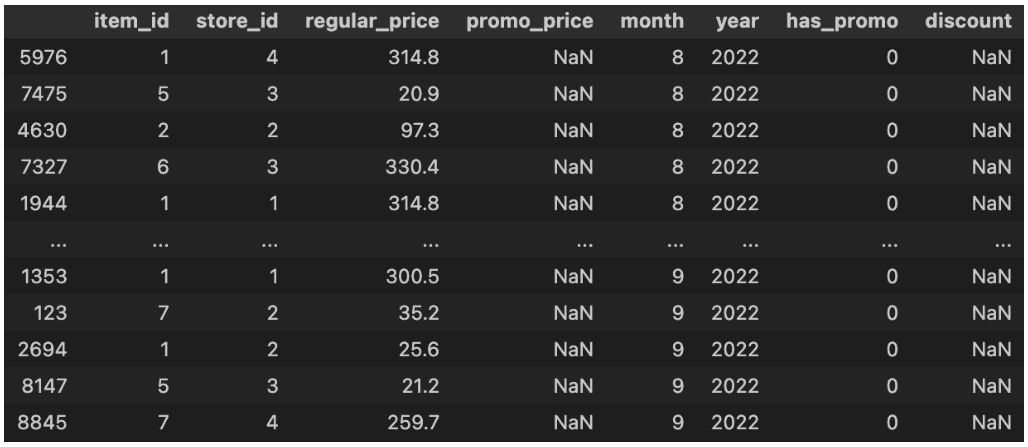
- Weather data, since they could influence the retail demand
- Holiday Indicators. Depending on the category, the demand could be increased or decreased.

- Prices of Competitors could give additional insights into promotions.

Of course keeping in mind the assumption, that all the stores operate every day and there are no stock outs.

4. Selection of models, training, prediction and evaluation

For the specific set of data, RandomForestRegressor from scikit-learn was used for the train of the model, since we are trying to predict units sold, which is a numeric value. After sorting the dataset by date, i splitted the dataset into test and train. Data after the 15th of August 2022 were used for testing and the rest for training. A screenshot of testing data is shown below.



	item_id	store_id	regular_price	promo_price	month	year	has_promo	discount
5976	1	4	314.8	NaN	8	2022	0	NaN
7475	5	3	20.9	NaN	8	2022	0	NaN
4630	2	2	97.3	NaN	8	2022	0	NaN
7327	6	3	330.4	NaN	8	2022	0	NaN
1944	1	1	314.8	NaN	8	2022	0	NaN
...	...	...	...	...	...	...	...	...
1353	1	1	300.5	NaN	9	2022	0	NaN
123	7	2	35.2	NaN	9	2022	0	NaN
2694	1	2	25.6	NaN	9	2022	0	NaN
8147	5	3	21.2	NaN	9	2022	0	NaN
8845	7	4	259.7	NaN	9	2022	0	NaN

Apart from the *model.score()* method, RMSE was used as a good measure for the evaluation of the model. Below are the results with room for improvement.

model.score()	RMSE
0.211684	1.81756

For the predictions, a function was created in order to take all the necessary input data and add the predictions to a CSV file. In all the cases of the input data, the *promo_price* was set to *NaN* and I used the same *regular_price* as the one in the latest date. Also, of course i excluded the predictions for the combinations of items-stores that didn't exist in the dataset.

5. Saving the results

The saving of the results took place in the previous mentioned function. The prediction results can be seen in *forecast_vas.csv.*