Data Mining: Polish pizzeria

Jędrzej Kopiszka index: 145304
Bartosz Maślanka index: 144091
Radosław Bodus index: 145457
Hubert Radom index: 145445

Poznan University of Technology, June 2021

1 Introduction

The goal of the project was to solve the problem of forecasting everyday sales using data from Polish pizzeria. Based on the data showing sales in the following days and hours, we wanted our prediction to allow the pizzeria to save money by more accurately estimating the number of pizzas sold, and thus to more accurately estimate the amount of ingredients needed. in this documentation, we will briefly describe the workflow of our project.

2 Exploratory analysis and preprocessing

2.1 Data import and very basic analysis

Our data looks as follows:

	count	year	month	day	hour	working_day	weekend_day	public_holiday
12	0	2016	10	12	0	1	0	0
13	0	2016	10	12	1	1	0	0
14	0	2016	10	12	2	1	0	0
15	0	2016	10	12	3	1	0	0
16	0	2016	10	12	4	1	0	0

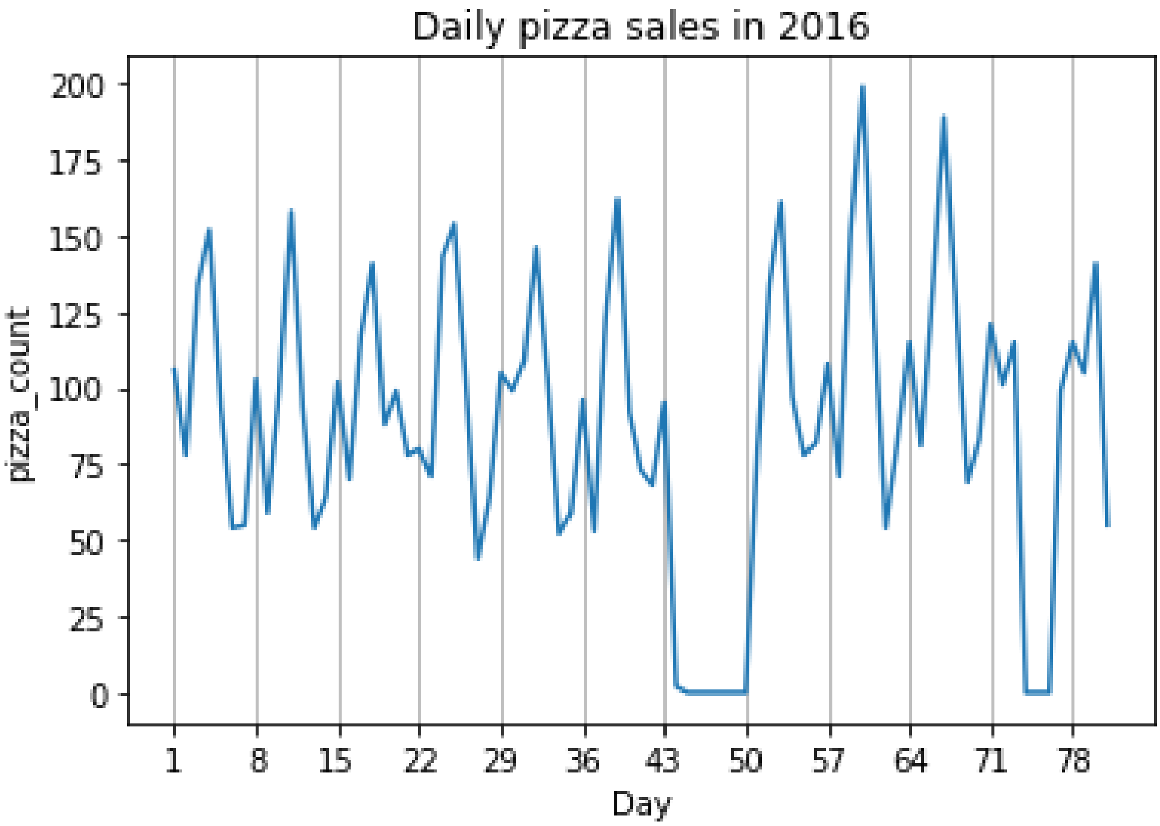
The data range is from 11.10.2016 to 19.06.2018. We drop the first day because it was not full - it started at 12.00 pm not at 12.00 am as every other day. We also remove the information about *weekend day*, because we already have a *working day* column, so it does not provide us with any new data, as it is the opposite

2.2 New DataFrames and deeper understanding of data

Our business goal was to forecast daily sales, so we aggregate data from hourly sales to daily sales. The following DataFrame is the starting point for further experimental changes to our data.

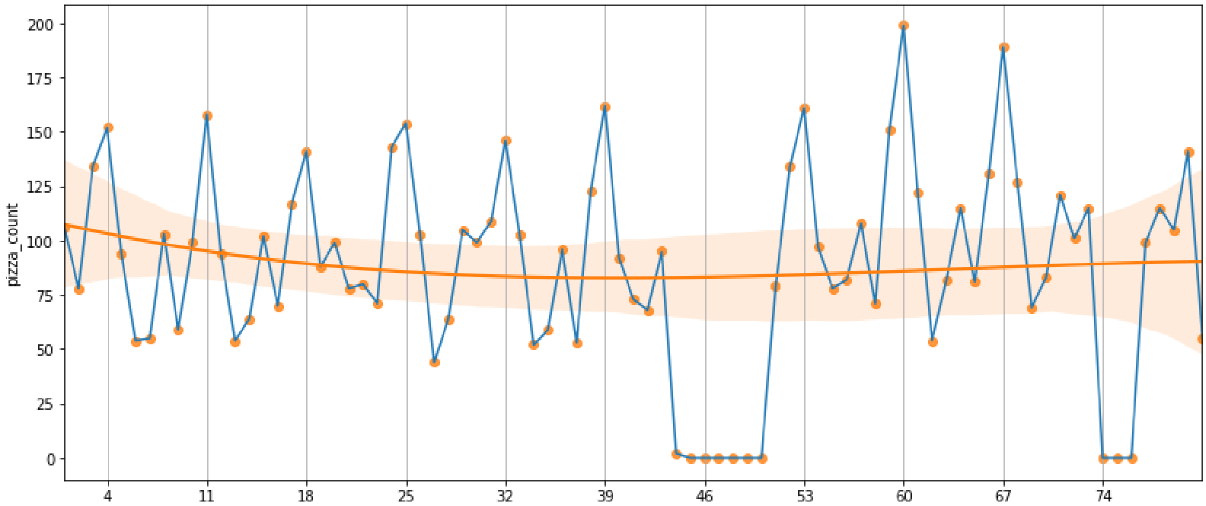
new_index	year	month	day	working_day	public_holiday	pizza_count
1	2016	10	12	1	0	106
2	2016	10	13	1	0	78
3	2016	10	14	1	0	134
4	2016	10	15	0	0	152
5	2016	10	16	0	0	94

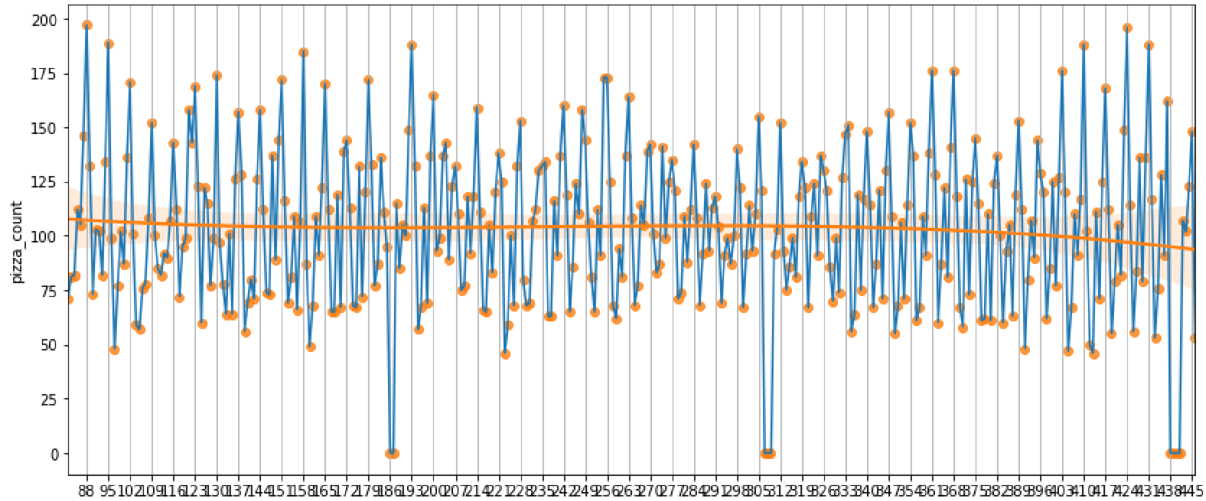
To get a better view of the data, we've shown 2016 sales in the chart.

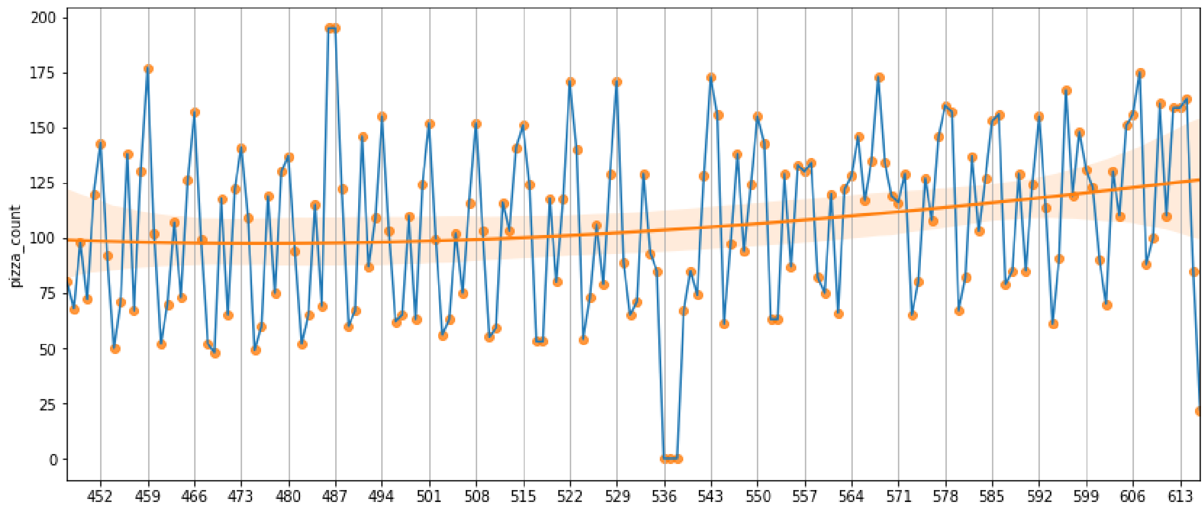


From visual looking at the data from 2016 we can observe days with no sales but after more in-depth analysis (and knowing the data comes from polish pizzeria) we see that pizzeria was closed in these days mostly due to polish national holidays.

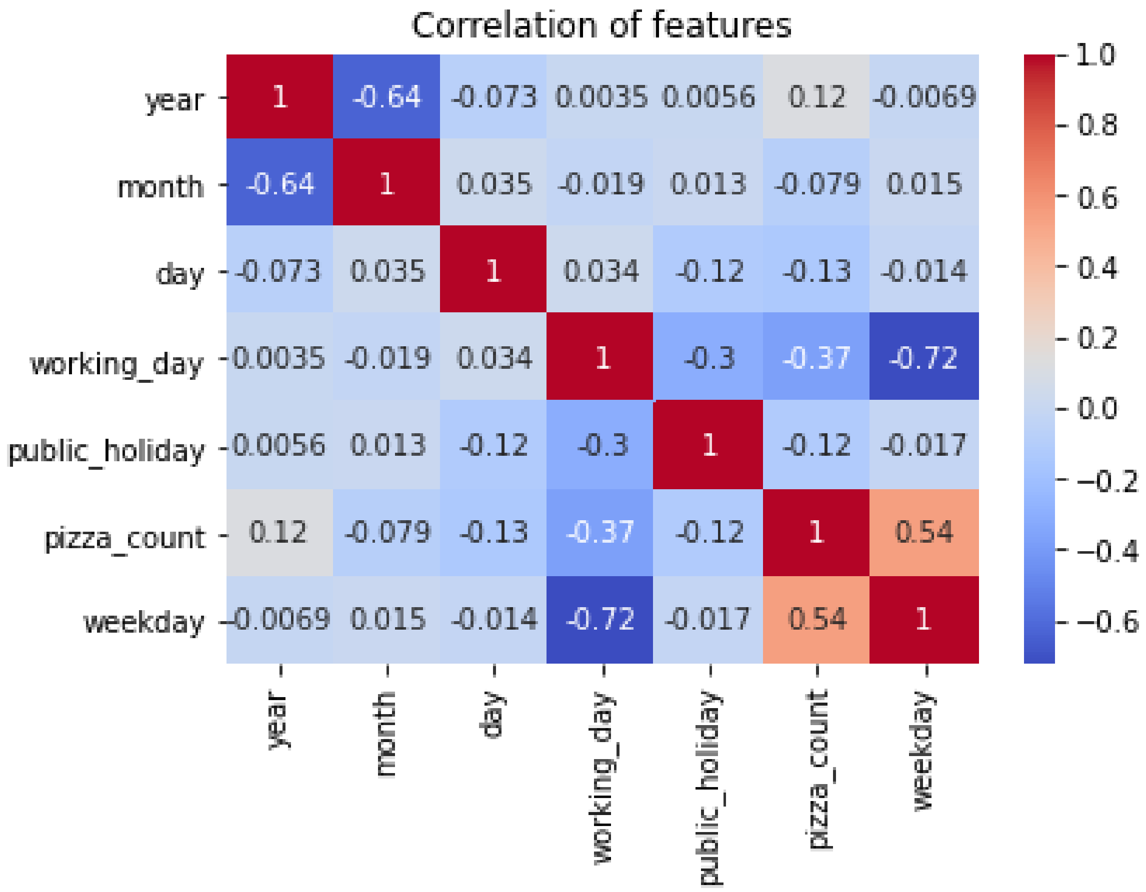
Below are daily sales charts for all 3 years.



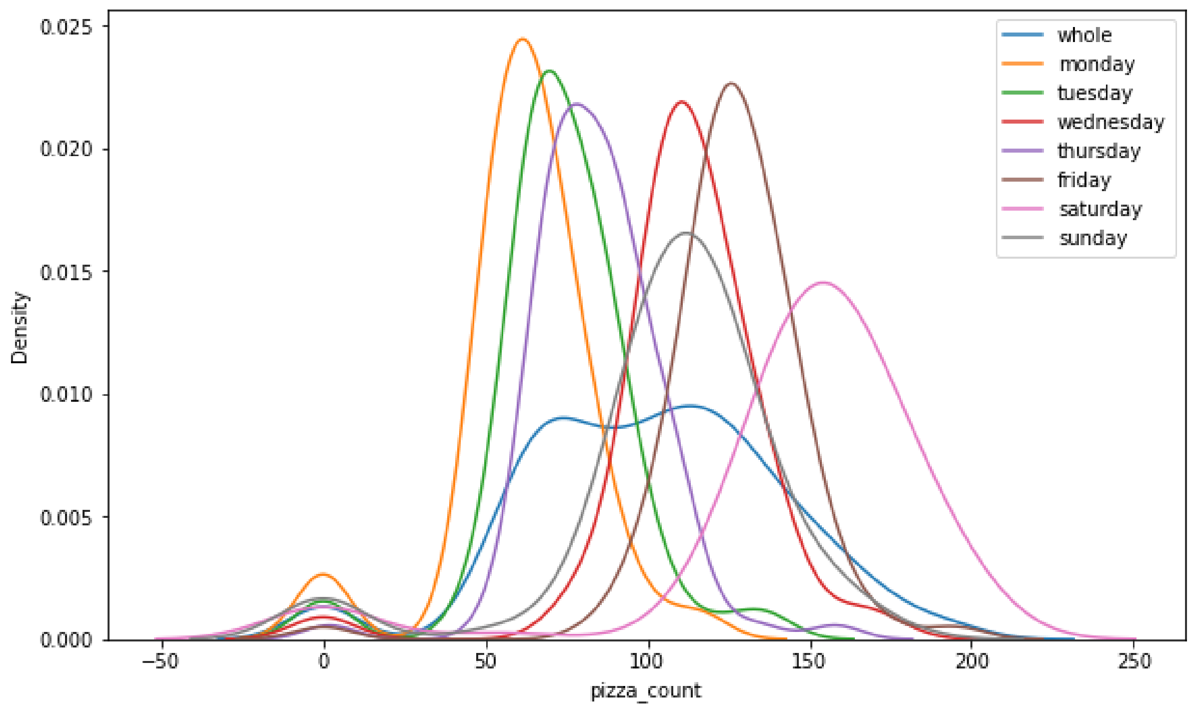




Correlation matrix shows that the number of pizzas sold depends largely on the day of the week, which of course we expected.



For this reason, we checked the mean and standard deviation of the number of pizzas sold for the following weekday and presented it on the chart.



This information is important to our business objective. The pizza owner would want our prediction to be better than just taking the mean and standard deviation into account for a given weekday.
We decided to use the mean to replace zero in our data. We figured this would help us create more efficient models, and in real life, the pizza owner will know when his pizzeria is closed. Before trying the models, we created yet another DataFrames for testing purposes.
In one of them we replaced single columns indicating dates with one of the date type.

	date	weekday	working_day	public_holiday	pizza_count
new_index					
1	2016-10-12	2	1	0	106
2	2016-10-13	3	1	0	78
3	2016-10-14	4	1	0	134
4	2016-10-15	5	0	0	152
5	2016-10-16	6	0	0	94

In the second, we changed the date to unix type.

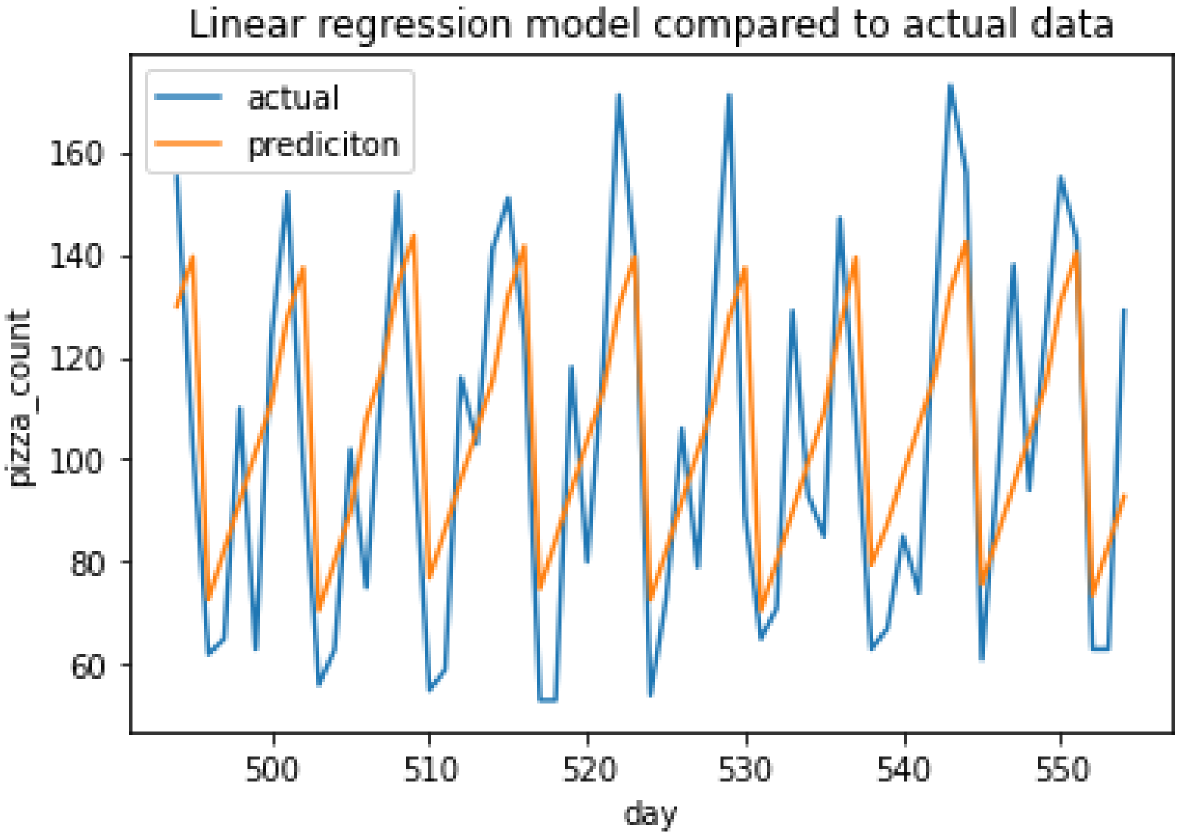
	date	weekday	working_day	public_holiday	pizza_count
new_index					
1	1.476230e+09	2	1	0	106
2	1.476317e+09	3	1	0	78
3	1.476403e+09	4	1	0	134
4	1.476490e+09	5	0	0	152
5	1.476576e+09	6	0	0	94

3 Creating a model

The metric we use to compare models is mean absolute error (MAE). The exact comparisons can be found in the Python notebook, here we include the best MAE results and prediction charts on test data

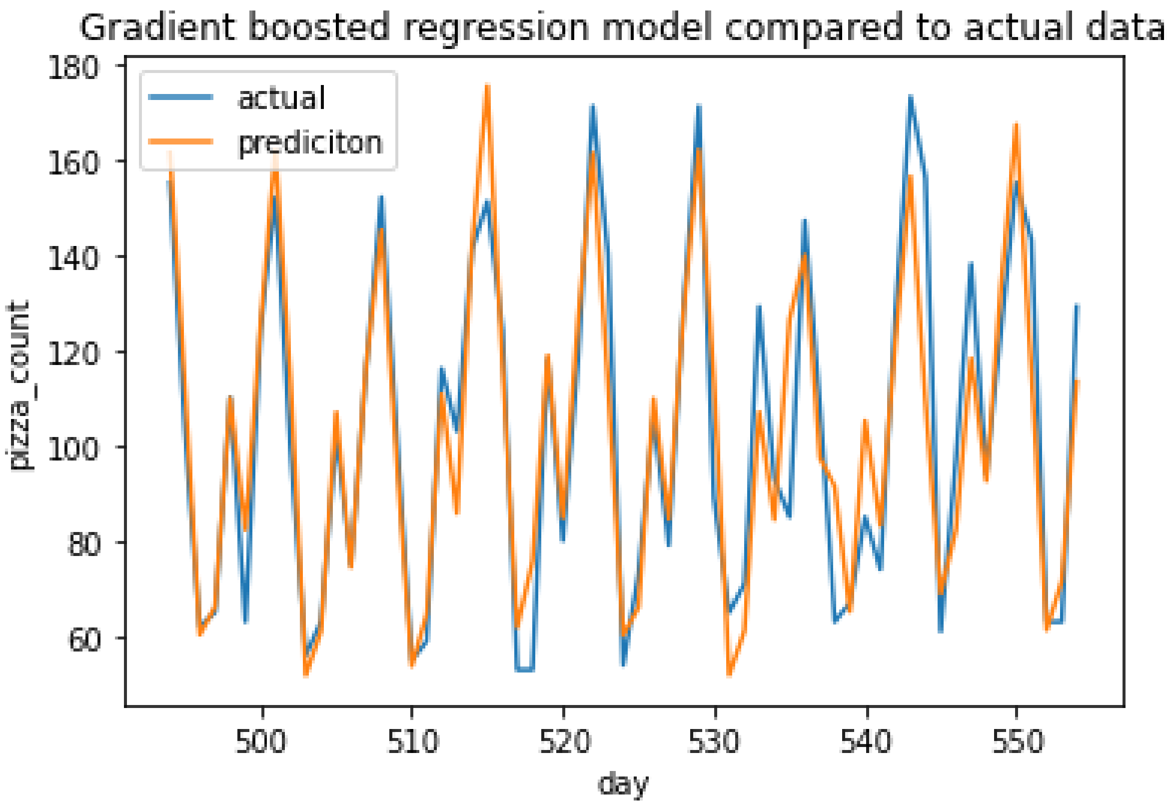
3.1 Linear regression

MAE: 20.871



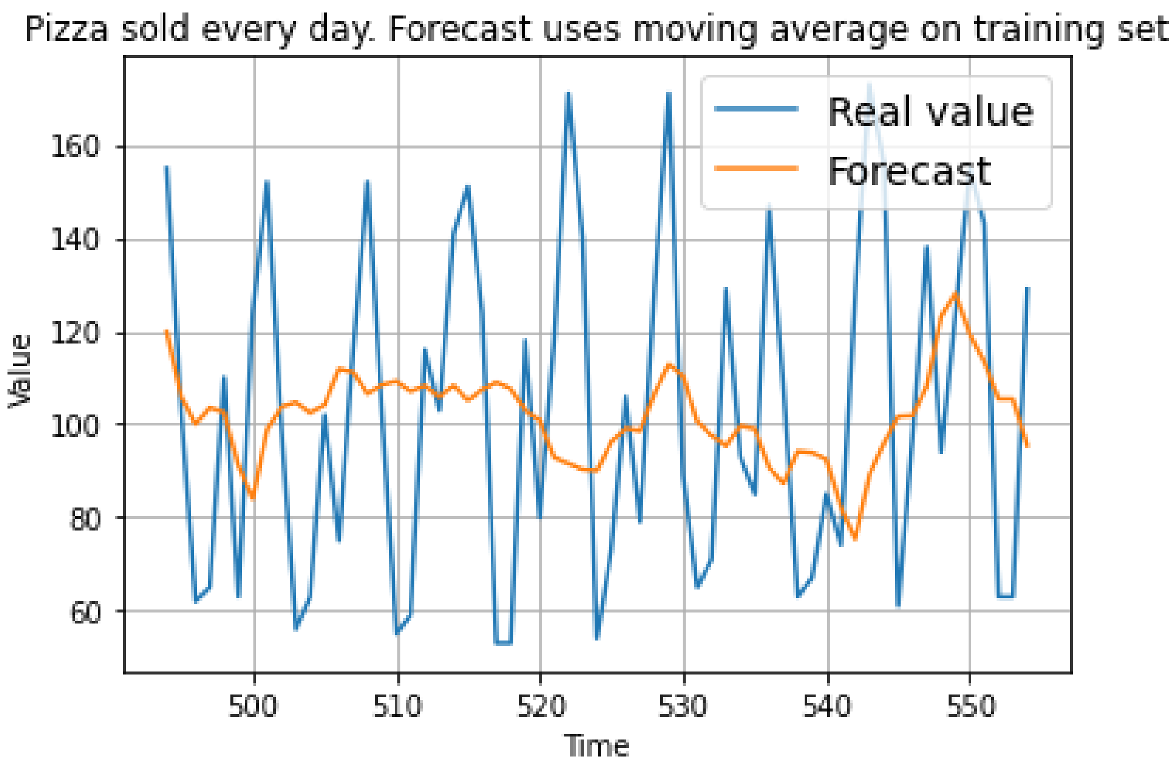
3.2 Gradient boosting regression

MAE: 10.726



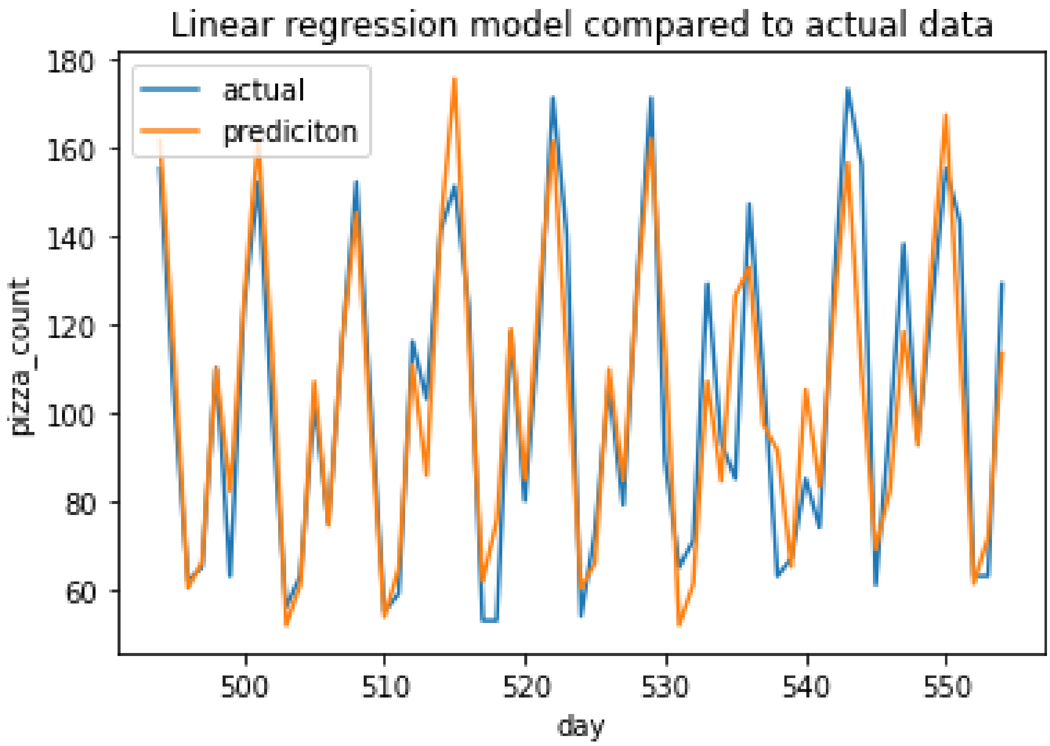
3.3 Moving average

MAE: 30.878



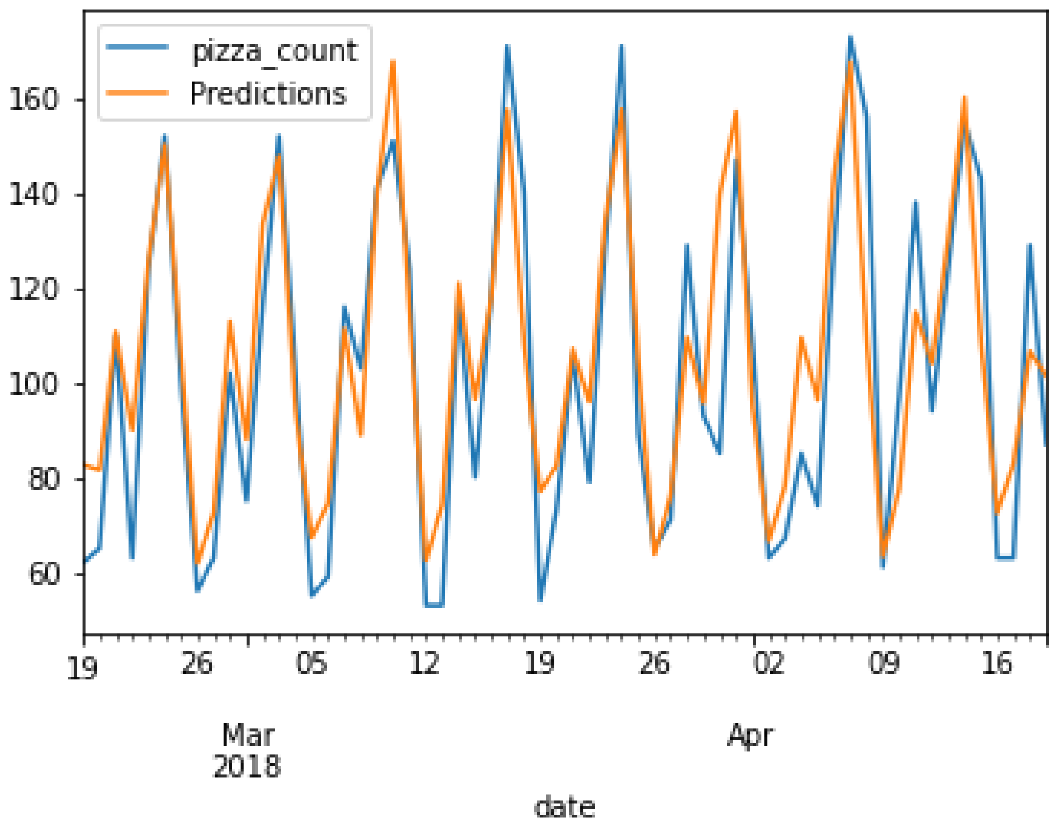
3.4 Gradient Boosting Regressor

MAE: 10.746



3.5 Recurrent Neural Networks

MAE: 13.370



3.6 Azure AutoML

We also experimentally tried to use a service from Microsoft Azure AutoML to create a model. First, we threw a model with a broken date into 3 columns, so the problem was treated as a regression problem. Below are the best models found and the MAE metric for the best model.

Algorithm name	Explained	Normalized root mean squared error ↑
VotingEnsemble	View explanation	0.07870
StackEnsemble		0.07961
MaxAbsScaler, XGBoostRegressor		0.08130
MaxAbsScaler, LightGBM		0.08166
MaxAbsScaler, RandomForest		0.08193
StandardScalerWrapper, XGBoostRegressor		0.08300
StandardScalerWrapper, GradientBoosting		0.08304
MaxAbsScaler, RandomForest		0.08349
MaxAbsScaler, ExtremeRandomTrees		0.08350
StandardScalerWrapper, ElasticNet		0.08411
MaxAbsScaler, DecisionTree		0.08415

mean_absolute_error

11.482

The second experiment was performed on data with date type, treating it as a time series. Below are also the models and the MAE metric for the best model.

Algorithm name	Explained	Normalized root mean squared error ↑
VotingEnsemble		0.06270
RobustScaler, GradientBoosting		0.06857
RobustScaler, GradientBoosting		0.07746
RobustScaler, GradientBoosting		0.07892
StandardScalerWrapper, RandomForest		0.07927
MaxAbsScaler, RandomForest		0.07979
MinMaxScaler, DecisionTree		0.08121
MaxAbsScaler, RandomForest		0.08153
RobustScaler, DecisionTree		0.08169
MaxAbsScaler, GradientBoosting		0.08221
TruncatedSVDWrapper, GradientBoosting		0.08283

mean_absolute_error

10.261

4 DVC

We've created the pipeline which made it possible to version our learning models. Thanks to DVC we are able to execute whole pipeline as a one commend. A file params.yaml makes it possible to easily adjust/manipulate the parameters of the given model

```yaml
stages:
  train:
    cmd: python src/train.py
    deps:
      - src/train.py
      - src/prepare_dataset.py
    params:
      - train.model_file_name
      - train.seed
    outs:
      - model.sav
  evaulate:
    cmd: python src/evaluate.py
    deps:
      - src/evaluate.py
      - model.sav
    params:
      - evaluate.model_file_name
      - evaluate.results_file_name
```

```
  +-------+
  | train |
  +-------+
      *
      *
      *
+----------+
| evaulate |
+----------+
+-------------------------+
| data\pizza_data.csv.dvc |
+-------------------------+
```

```yaml
schema: '2.0'
stages:
  train:
    cmd: python src/train.py
    deps:
    - path: src/prepare_dataset.py
      md5: f012d1168226c749bd3b2ad3492fb727
      size: 3565
    - path: src/train.py
      md5: da516131218de439201c0611c18e9afa
      size: 1265
    params:
      params.yaml:
        train.model_file_name: model.sav
        train.seed: 124532
    outs:
    - path: model.sav
      md5: df2ded9b24b4e99f76b59205f2198997
      size: 165747
  evaluate:
    cmd: python src/evaluate.py
    deps:
    - path: model.sav
      md5: df2ded9b24b4e99f76b59205f2198997
      size: 165747
    - path: src/evaluate.py
      md5: 1f893b21fb9f8330d9e43476b17c08cc
      size: 1006
    params:
      params.yaml:
        evaluate.model_file_name: model.sav
        evaluate.results_file_name: results.json
```