Time Series Project

Time_series_forecasting_for_taxi_company

Table of Contents

Goal

Develop a model for the Sweet Lift Taxi company that predicts the amount of taxi orders for the next hour based on the historical data. It will be used to attract more drivers during peak hours.

The RMSE metric on the test set should not be more than 48.

Data description

Features

- *datetime* — time stamp of the observation.

Target

- *num_orders* - the number of taxi orders at a particular time stamp.

Imports

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib
import numpy as np

from statsmodels.tsa.seasonal import seasonal_decompose

from sklearn.dummy import DummyRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from catboost import CatBoostRegressor
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor

from sklearn.model_selection import train_test_split
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler as ss
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer

import matplotlib.pyplot as plt
%matplotlib inline

import sys
import warnings
if not sys.warnoptions:
       warnings.simplefilter("ignore")

pd.set_option('display.max_rows', None, 'display.max_columns', None)

print("Setup Complete")
```

```
Setup Complete
```

Input data

In [2]:
```
try:
    df = pd.read_csv('taxi.csv', index_col=[0], parse_dates=[0])

except:
    df = pd.read_csv('/datasets/taxi.csv', index_col=[0], parse_dates=[0])
```

Descriptive statistics

In [3]:
```
df.head()
```

Out[3]:

datetime	num_orders
2018-03-01 00:00:00	9
2018-03-01 00:10:00	14
2018-03-01 00:20:00	28
2018-03-01 00:30:00	20
2018-03-01 00:40:00	32

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 26496 entries, 2018-03-01 00:00:00 to 2018-08-31 23:50:00
Data columns (total 1 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   num_orders  26496 non-null  int64
dtypes: int64(1)
memory usage: 414.0 KB
```

Notes for preprocessing:

- There are more than 26k observations with 1 feature and 1 target variable;
- Data is collected for 6 months in 2018;
- Check if the dates and times are in chronological order;
- Resample data by 1 hour because we need to predict orders for the next hour. Take the sum as an aggregation function since we need to identify the total number of orders for a particular hour;
- No missing values;
- Check for duplicates;
- The target is numeric, it's a regression task.

Data preprocessing

Duplicates

In [5]:
```
df.index.duplicated().sum()
```

Out[5]: 0

Chronological order

In [6]:
```
df.sort_index(inplace=True)
```

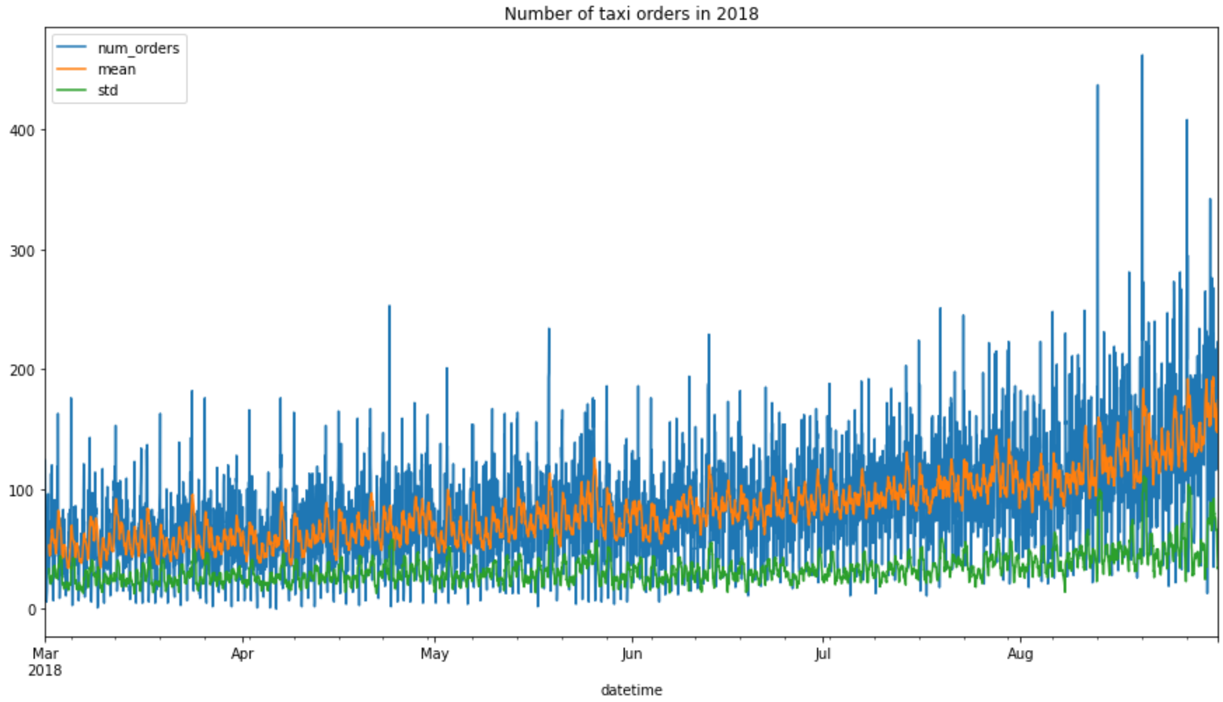
In [7]:
```
df.index.is_monotonic
```

Out[7]: True

The dates and times are in chronological order.

Resampling

In [8]:
```
df = df.resample('1H').sum()
df['mean'] = df['num_orders'].rolling(15).mean()
df['std'] = df['num_orders'].rolling(15).std()
df.plot(figsize=(15,8), title='Number of taxi orders in 2018');
```



We see that the standard deviation almost doesn't change over time. However the average value of this dataset increases by a factor of 2. This stochastic process is slightly nonstationary.

In [9]:
```
df = df.drop(['mean','std'], axis=1)
```

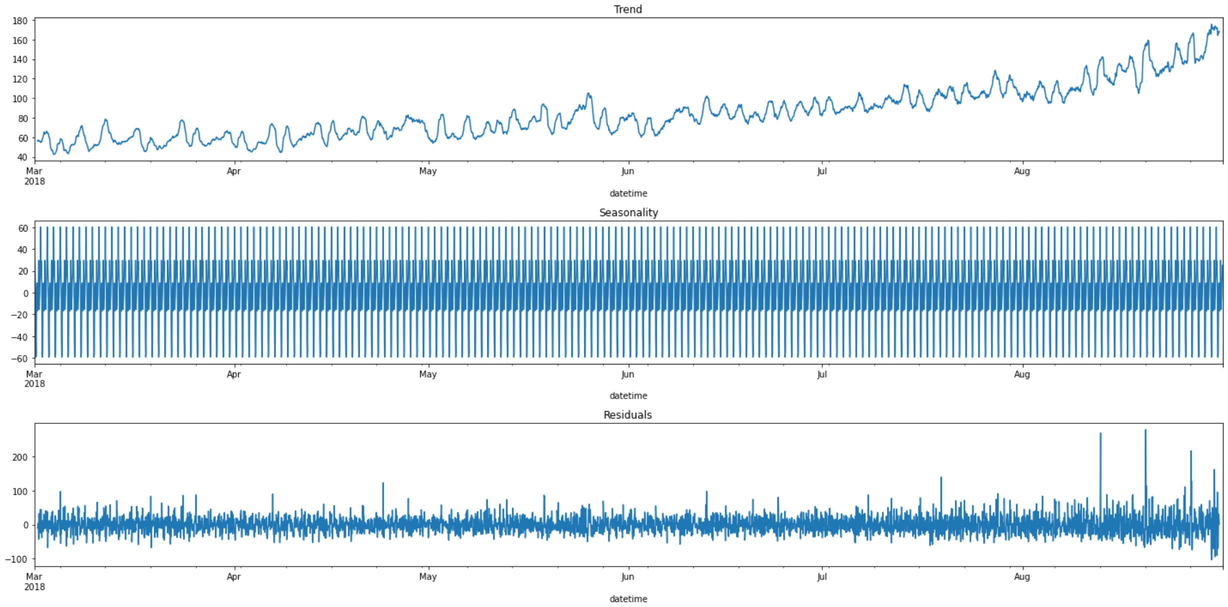
EDA

Let's perform a seasonal decomposition of this dataset to see if there are any trends and seasonality in the data.

In [10]:
```
decomposed = seasonal_decompose(df)

plt.figure(figsize=(20, 10))

plt.subplot(311)
decomposed.trend.plot(ax=plt.gca())
plt.title('Trend')
plt.subplot(312)
decomposed.seasonal.plot(ax=plt.gca())
plt.title('Seasonality')
plt.subplot(313)
decomposed.resid.plot(ax=plt.gca())
plt.title('Residuals')
plt.tight_layout()
```



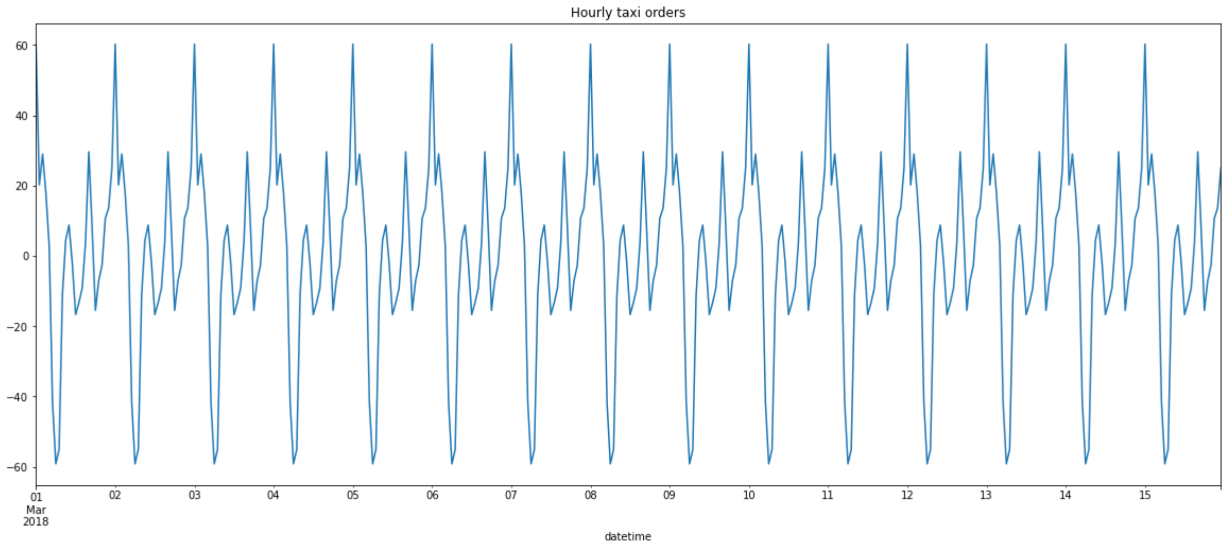
We can see the overall trend that the number of orders is growing in this company over the time.

We don't see any clear seasonality in terms of months of the year in this dataset but we should remember that the data is collected for only 6 months.

The residuals graph shows a stationary stochastic process, the level of residuals doesn't change over time, so it shouldn't influence the predictions. It is essentially just noise in the data. The residuals level is mostly around 0, which means that the data was well decomposed.

Let's look at the hourly data and see whether there are any patterns.

In [11]:
```
plt.figure(figsize=(20, 8))
decomposed.seasonal['2018-03-01':'2018-03-15'].plot()
plt.title('Hourly taxi orders');
```



We can see a clear pattern of the number of orders during each day. The lowest peak is around 6 am and the highest one is around 12 am. We will create a new feature in the next section that will help a model better learn this pattern.

Feature engineering

Time series differences

First of all, to make our data more stationary, let's shift it by 1 hour and then try to predict the difference between these 2 series.

In [12]:
```
df -= df.shift()
df.head()
```

Out[12]:

datetime	num_orders
2018-03-01 00:00:00	NaN
2018-03-01 01:00:00	-39.0
2018-03-01 02:00:00	-14.0
2018-03-01 03:00:00	-5.0
2018-03-01 04:00:00	-23.0

New features

Let's create some new features from our data, it should help the model make better predictions.

In [13]:
```
def make_features(data, max_lag, rolling_mean_size):
    data['month'] = data.index.month
    data['day'] = data.index.day
    data['dayofweek'] = data.index.dayofweek
    data['hourofday'] = data.index.hour

    for lag in range(1, max_lag + 1):
        data['lag_{}'.format(lag)] = data['num_orders'].shift(lag)

    data['rolling_mean'] = data['num_orders'].shift().rolling(rolling_mean_si
    data['rolling_std'] = data['num_orders'].shift().rolling(rolling_mean_siz

make_features(df, 10, 5)
```

In [14]:
```
df.head()
```

Out[14]:

datetime	num_orders	month	day	dayofweek	hourofday	lag_1	lag_2	lag_3	lag_4	lag_5
2018-03-01 00:00:00	NaN	3	1	3	0	NaN	NaN	NaN	NaN	NaN
2018-03-01 01:00:00	-39.0	3	1	3	1	NaN	NaN	NaN	NaN	NaN
2018-03-01 02:00:00	-14.0	3	1	3	2	-39.0	NaN	NaN	NaN	NaN
2018-03-01 03:00:00	-5.0	3	1	3	3	-14.0	-39.0	NaN	NaN	NaN

	num_orders	month	day	dayofweek	hourofday	lag_1	lag_2	lag_3	lag_4	lag_5
datetime										
2018-03-01 04:00:00	-23.0	3	1	3	4	-5.0	-14.0	-39.0	NaN	NaN

Splitting the data

In [15]:
```
df = df.dropna(how='any', axis=0)

X = df.drop('num_orders', axis=1)
y = df['num_orders']

X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=False, test_
```

In [16]:
```
print('Train set from', X_train.index.min(), 'until', X_train.index.max())
print('Test set from', X_test.index.min(), 'until', X_test.index.max())
```

```
Train set from 2018-03-01 11:00:00 until 2018-08-13 14:00:00
Test set from 2018-08-13 15:00:00 until 2018-08-31 23:00:00
```

Standard scaling

Let's scale the features before modeling to be able to compare their coefficients in the later sections.

In [17]:
```
sc = ss()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Model selection

We will be using the RMSE metric for best model selection. Let's create this function and the respective scorer to use it in the GridSearch and cross-validation further.

In [18]:
```
def rmse(actual, predict):
    predict = np.array(predict)
    actual = np.array(actual)

    distance = predict - actual

    square_distance = distance ** 2

    mean_square_distance = square_distance.mean()

    score = np.sqrt(mean_square_distance)

    return score

rmse_scorer = make_scorer(rmse, greater_is_better = False)
```

Let's instantiate a TimeSeriesSplit that we will use for different models to split train set into train and validation parts.

In [19]:
```
tscv = TimeSeriesSplit()
```

Baseline

In [20]:
```
model = DummyRegressor(strategy='median')
baseline = np.mean(abs(cross_val_score(model,  X_train, y_train, cv=tscv, sco
baseline
```

Out[20]: 37.03855837988927

Random Forest

Base model

In [21]:
```
RF = RandomForestRegressor(random_state=12345)
RF_base = np.mean(abs(cross_val_score(RF,  X_train, y_train, cv=tscv, scoring
RF_base
```

Out[21]: 27.695984384784133

Hyperparameters tuning

In [22]:
```
params = {"n_estimators" : [500, 700],
          "max_depth" : [6, 7, 8, 9, 10]}

gsSVR = GridSearchCV(estimator=RF, cv=tscv, param_grid=params, n_jobs=-1, ver
gsSVR.fit(X_train, y_train)
SVR_best = gsSVR.best_estimator_
print(abs(gsSVR.best_score_))
```

27.685830844756293

In [23]:
```
best_param = pd.DataFrame(gsSVR.best_params_, index=[0])
RF_score_tuned = abs(gsSVR.best_score_)
best_param['score'] = RF_score_tuned

best_param
```

Out[23]:

	max_depth	n_estimators	score
0	10	500	27.685831

XGBoost

In [24]:
```
XGB = XGBRegressor(n_jobs=-1, random_state=12345)
XGB_base = np.mean(abs(cross_val_score(XGB,  X_train, y_train, cv=tscv, scori
XGB_base
```

Out[24]: 27.84081524120368

Hyperparameters tuning

In [25]:
```
params = {"n_estimators" : [500, 700],
          "max_depth" : [6, 7, 8, 9, 10]}

gsSVR = GridSearchCV(estimator=XGB, cv=tscv, param_grid=params, n_jobs=-1, ve
gsSVR.fit(X_train, y_train)
SVR_best = gsSVR.best_estimator_
print(abs(gsSVR.best_score_))
```

27.87332446489183

In [26]:
```
best_param = pd.DataFrame(gsSVR.best_params_, index=[0])
```

```
XGB_score_tuned = abs(gsSVR.best_score_)
best_param['score'] = XGB_score_tuned

best_param
```

Out[26]:

	max_depth	n_estimators	score
0	6	500	27.873324

LightGBM

In [27]:

```
LGB = LGBMRegressor(random_state=12345)
LGB_base = np.mean(abs(cross_val_score(LGB,  X_train, y_train, cv=tscv, scori
LGB_base
```

Out[27]: 26.927415627655176

Hyperparameters tuning

In [28]:

```
params = {"n_estimators" : [500, 700],
          "max_depth" : [6, 7, 8, 9, 10]}

gsSVR = GridSearchCV(estimator=LGB, cv=tscv, param_grid=params, n_jobs=-1, ve
gsSVR.fit(X_train, y_train)
SVR_best = gsSVR.best_estimator_
print(abs(gsSVR.best_score_))
```

```
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_l
eaves OR 2^max_depth > num_leaves. (num_leaves=31).
26.922160988250802
```

In [29]:

```
best_param = pd.DataFrame(gsSVR.best_params_, index=[0])
LGB_score_tuned = abs(gsSVR.best_score_)
best_param['score'] = LGB_score_tuned

best_param
```

Out[29]:

	max_depth	n_estimators	score
0	7	500	26.922161

CatBoost

In [30]:

```
CB = CatBoostRegressor(verbose=0, loss_function="RMSE", random_state=12345)
CB_base = np.mean(abs(cross_val_score(CB,  X_train, y_train, cv=tscv, scoring
CB_base
```

Out[30]: 25.98269431164594

Hyperparameters tuning

In [31]:

```
params = {"iterations" : [500, 700],
          "depth" : [6, 7, 8, 9, 10]}

gsSVR = GridSearchCV(estimator=CB, cv=tscv, param_grid=params, n_jobs=-1, verl
gsSVR.fit(X_train, y_train)
SVR_best = gsSVR.best_estimator_
print(abs(gsSVR.best_score_))
```

```
26.061997143628922
```

In [32]:

```
best_param = pd.DataFrame(gsSVR.best_params_, index=[0])
```

```
CB_score_tuned = abs(gsSVR.best_score_)
best_param['score'] = CB_score_tuned

best_param
```

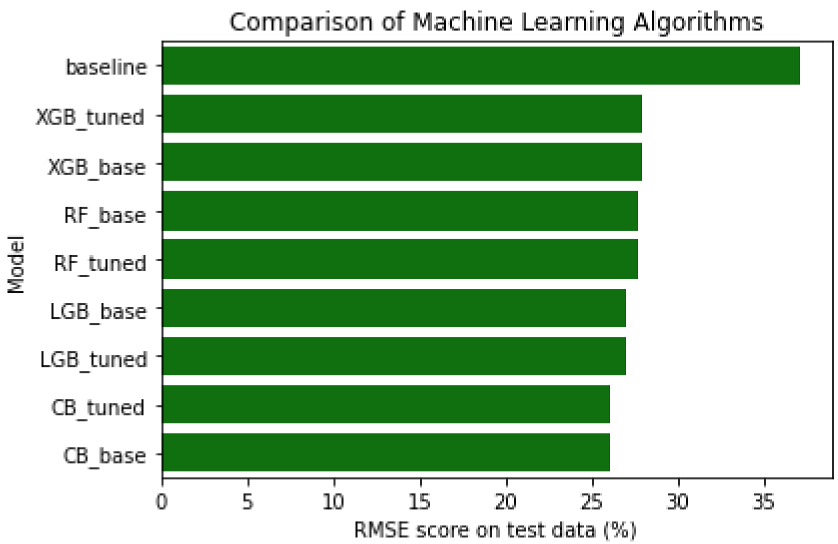
Out[32]:

	depth	iterations	score
0	6	700	26.061997

Results

In [33]:
```
models = pd.DataFrame({
    'Model': ['baseline', 'RF_base', 'XGB_base', 'LGB_base', 'CB_base',
              'RF_tuned', 'XGB_tuned', 'LGB_tuned', 'CB_tuned'],
    'Score': [baseline, RF_base, XGB_base, LGB_base, CB_base,
              RF_score_tuned, XGB_score_tuned, LGB_score_tuned, CB_score_tuned
sorted_by_score = models.sort_values(by='Score', ascending=False)
```

In [34]:
```
sns.barplot(x='Score', y = 'Model', data = sorted_by_score, color = 'g')
plt.title('Comparison of Machine Learning Algorithms')
plt.xlabel('RMSE score on test data (%)')
plt.ylabel('Model');
```



As we can see, the **base CatBoost model** showed the best train RMSE score (25.98). Based on that, we will be recommending this model to be the final model for this project. In terms of future development, we see that hyperparameter tuning didn't change the CB model's score much. Probably other parameters should also be tuned to reach any significant improvement.

Feature importances

Let's see what features were the most important for prediction.

In [35]:
```
CB = CatBoostRegressor(verbose=0, loss_function="RMSE", random_state=12345)
CB.fit(X_train, y_train)

coeff_df = pd.DataFrame()
coeff_df['Feature'] = df.drop(['num_orders'], axis=1).columns.values
coeff_df["Correlation"] = pd.Series(CB.feature_importances_)

coeff_df.sort_values(by='Correlation', ascending=False)
```

Out[35]:

	Feature	Correlation
3	hourofday	30.721629
14	rolling_mean	17.810282
4	lag_1	9.291658
9	lag_6	6.341063
0	month	4.093035
5	lag_2	3.837391
11	lag_8	3.715292
13	lag_10	3.332679
10	lag_7	3.315139
8	lag_5	3.313232
2	dayofweek	3.211398
12	lag_9	2.846059
6	lag_3	2.716146
7	lag_4	2.426004
15	rolling_std	2.008833
1	day	1.020160

`hourofday` and `rolling_mean` turned out to be much more important in this model than any other feature.

Test the model

In [36]:
```
y_pred_test = CB.predict(X_test)
CB_test_score = rmse(y_test, y_pred_test)
CB_test_score
```

Out[36]: 40.570918937457144

In [37]:
```
round((40.57-25.98)/40.57 * 100, 2)
```

Out[37]: 35.96

The test score is 35.96% higher than the train score which probably suggests some degree of overfitting. This gives some room for future improvement. However the target metric of 48 or lower has been reached.

Sanity check

In [38]:
```
model = DummyRegressor(strategy='median')
model.fit(X_train, y_train)
y_base_test = model.predict(X_test)
base_test_score = rmse(y_test, y_base_test)
base_test_score
```

Out[38]: 58.92214010119039

In [39]:
```
round((58.92-40.57)/58.92 * 100, 2)
```

Out[39]: 31.14

The test score of the Dummy regression model, that we use as a baseline to analyze the model quality, is 31.14% higher than our final chosen model. It means that the modeling was useful.

Conclusion

The **goal** of this project was to develop a model to determine the amount of taxi orders for the next hour based on the historical data. The RMSE metric on the test set had to be no more than **48**.

We have completed the following steps in this project:

1.Descriptive statistics

2.Data preprocessing

We made sure the dates and times were in chronological order, resample data by 1 hour because we needed to predict orders for the next hour and took the sum as an aggregation function since we needed to identify the total number of orders for a particular hour. We also plotted the data and noticed that this set is quite stationary. Finally, we checked the data for duplicates.

3.EDA

We have performed a seasonal decomposition of this dataset to see if there are any trends and seasonality in the data. We noticed the overall trend that the number of orders is growing in this company over the time. We didn't see any clear seasonality in terms of months of the year in this dataset but the data was collected for only 6 months. The level of residuals doesn't change over time, so it shouldn't influence the predictions. The residuals level is mostly around 0, which means that the data was well decomposed.

Finally, we've decided to take a look at the hourly data and noticed a pattern of the number of orders during each day. Based on that we've decided to create a new feature that will help a model better learn this pattern.

4.Splitting the data

Data was split into train and test sets with the ration 1:5.

5. Standard scaling

It was performed to be able to compare feature importances.

6.Model selection

We have compared Linear Regression, Random Forest, XGBoost, LightGBM and CatBoost models. We have also tuned a few hyperparameters for these algorithms. We have chosen the tuned CatBoost model based on RMSE score.

7.Feature importances

Based on the features importances attribute, we found that the `hourofday` and `rolling_mean` turned out to be much more important in this model than any other feature.

8.Sanity check

The test score of the Dummy regression model, that we use as a baseline to analyze the model quality, is 30.51% higher than our final chosen model. It means that the modeling was useful.

Results

The base CatBoost model has shown the best results (**test RMSE of 40.57**) in terms of quality. The target metric of 48 or lower has been reached. It showed some degree of overfitting that can be dealt with in the future. Besides, the model with tuned hyperparameters gave slightly worse results than the base model with default parameters, probably other parameters should be adjusted. This can be done in the future.