

# Прогнозирование количества заказов такси на следующий час

## Table of Contents

- 1 Введение
- 2 Подготовка к проекту
  - 2.1 Импорт библиотек
  - 2.2 Функции для анализа данных
- 3 Подготовка данных
  - 3.1 Изучение данных
  - 3.2 Анализ и предобработка данных
  - 3.3 Вывод
- 4 Создание моделей
  - 4.1 Подготовка к обучению моделей
    - 4.1.1 Создание выборок
    - 4.1.2 Дамми модели
    - 4.1.3 Функция для обучения и оценки моделей методом кросс-валидации
  - 4.2 Обучение моделей
  - 4.3 Тестирование лучшей модели
  - 4.4 Вывод
- 5 Заключение

## Введение

### Описание проекта

Компания «Чётенькое такси» собрала исторические данные о заказах такси в аэропортах. Чтобы привлекать больше водителей в период пиковой нагрузки, нужно спрогнозировать количество заказов такси на следующий час. Постройте модель для такого предсказания.

**Цель:** Значение метрики RMSE на тестовой выборке должно быть не больше 48.

### Описание данных

Данные лежат в файле taxi.csv. Количество заказов находится в столбце 'num\_orders' (от англ. number of orders, «число заказов»).

## Подготовка к проекту

### Импорт библиотек

```
In [1]: !pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\anna\anaconda3\lib\site-packages (1.3.3)
Requirement already satisfied: numpy in c:\users\anna\anaconda3\lib\site-packages (from xgboost) (1.19.2)
```

```
Requirement already satisfied: scipy in c:\users\anna\anaconda3\lib\site-packages (from xgboost) (1.5.2)
```

```
In [2]: # pandas
import pandas as pd
# numpy
import numpy as np
# библиотеки, необходимые для загрузки данных из файла
import os
from pathlib import Path
import urllib

# библиотека для графиков
import matplotlib.pyplot as plt

# библиотека для разбиения данных на выборки
from sklearn.model_selection import train_test_split

# библиотека для разложения ряда на тренд, сезонную составляющую и шум
from statsmodels.tsa.seasonal import seasonal_decompose

import statsmodels.tsa.api as smt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from xgboost import XGBRegressor

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

from sklearn.preprocessing import StandardScaler
```

```
In [3]: import warnings

warnings.filterwarnings('ignore')
```

## ФУНКЦИИ ДЛЯ АНАЛИЗА ДАННЫХ

```
In [5]: def my_plot(data, xl, yl, tit):
    """Построение графика

    data - данные
    xl - подпись оси абсцисс
    yl - подпись оси ординат

    Возвращает:
    ничего
    """
    data.plot()
    plt.xlabel(xl)
    plt.ylabel(yl)
    plt.grid(True)
    plt.title(tit)
    plt.show()
```

```
In [6]: def corr_fun(x,y,t):
    """Рассчитывает значения корреляционной функции

    x - входной параметр
    y - выходной параметр
```

```

t - диапазон сдвига по тaktам

Возвращает:
значения корреляционной функции при сдвиге тактов от 0 до t
"""

CorM=[]
for j in range(1,t+1,1):
    CorM.append(np.corrcoef(x[0:-j],y[j:])[0,1])
return CorM

```

## Подготовка данных

### Изучение данных

```
In [7]: Path('datasets').mkdir(parents=True,exist_ok=True)
get_file('datasets/taxi.csv', 'https://...taxi.csv')
```

```
In [8]: # загрузка данных в dataframe
taxi=pd.read_csv('datasets/taxi.csv',index_col=[0], parse_dates=[0])
```

```
In [9]: taxi.head()
```

```
Out[9]:
      num_orders
datetime
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 [10]: taxi.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
```

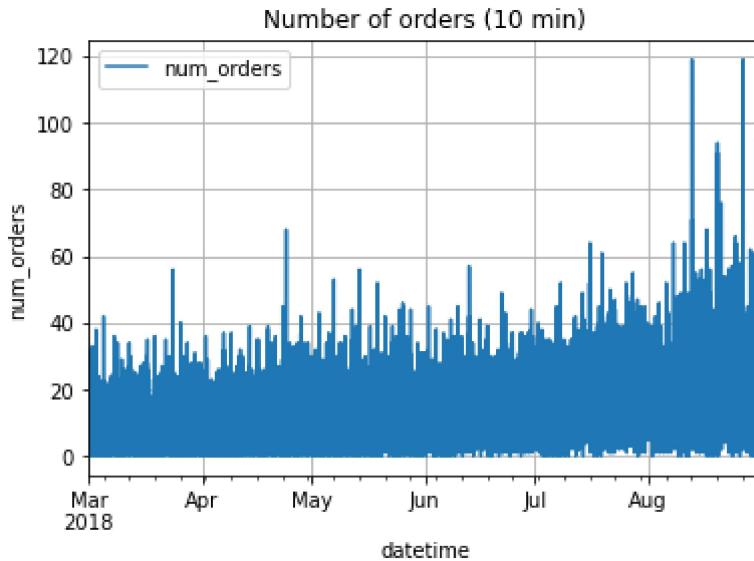
Данные содержат информацию о заказах такси каждые 10 мин. в течение периода с 1 марта 2018 г. по 31 августа 2018 г. (26496 значений)

```
In [11]: taxi.index.is_monotonic
```

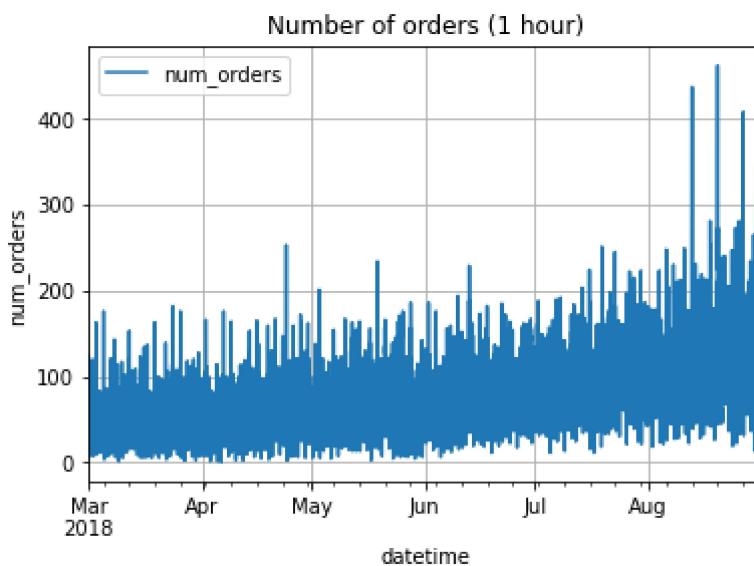
```
Out[11]: True
```

Данные монотонны

```
In [12]: my_plot(taxi, 'datetime', 'num_orders', 'Number of orders (10 min)')
```

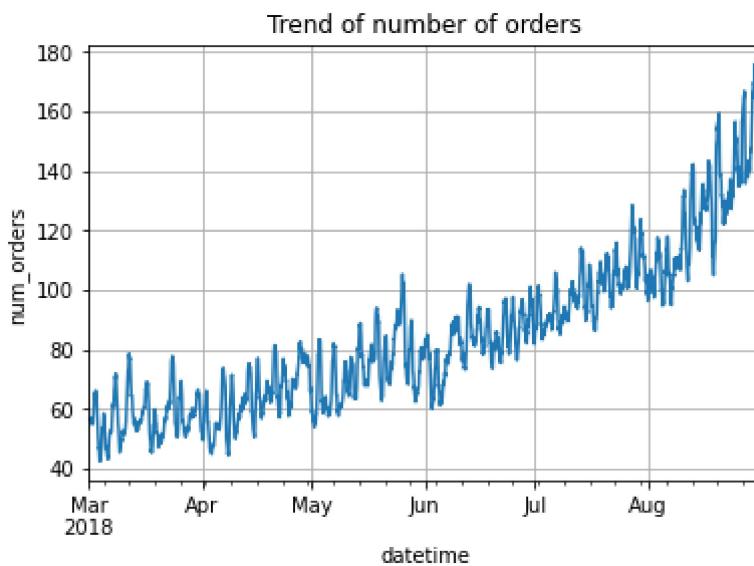


```
In [13]: my_plot(taxi.resample('1H').sum(), 'datetime', 'num_orders', 'Number of orders (1 h
```

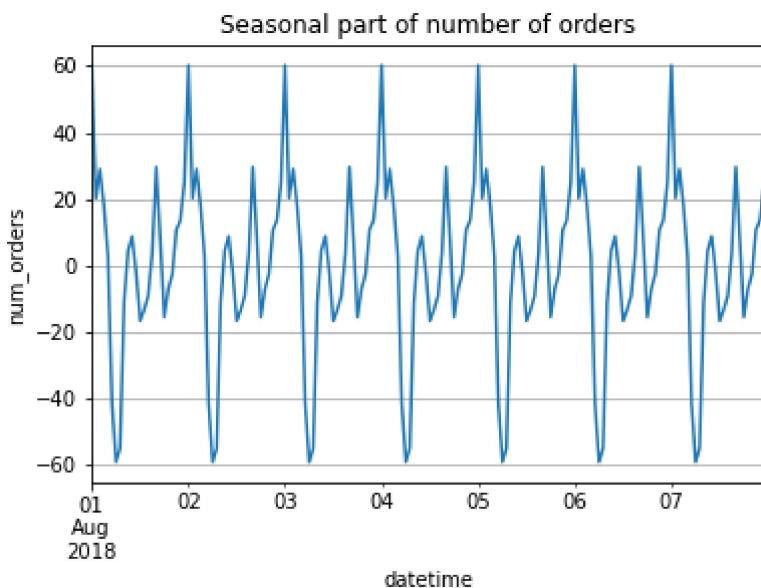


```
In [14]: decomposed = seasonal_decompose(taxi.resample('1H').sum())
```

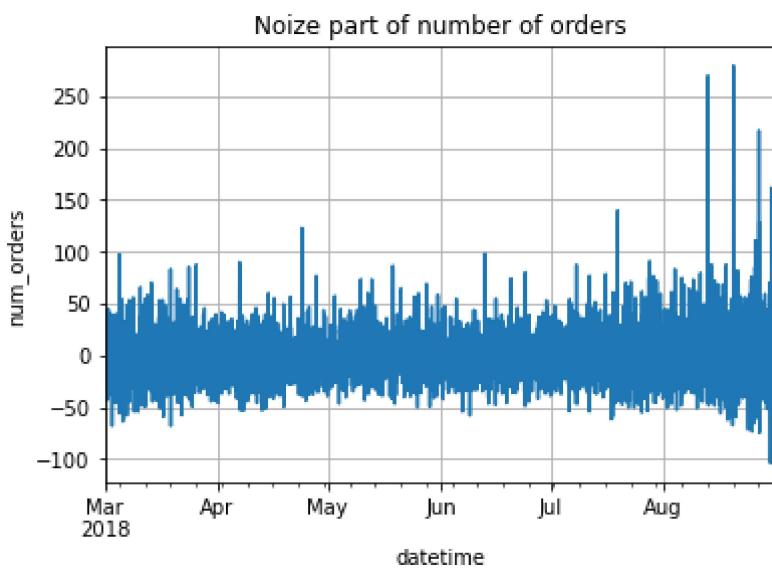
```
In [15]: my_plot(decomposed.trend, 'datetime', 'num_orders', 'Trend of number of orders')
```



```
In [16]: my_plot(decomposed.seasonal['2018-08-01':'2018-08-07'], 'datetime', 'num_orders', 'S
```



```
In [17]: my_plot(decomposed.resid, 'datetime', 'num_orders', 'Noize part of number of orders')
```



Ряд гетероскедастичен – дисперсия данных в августе больше, чем в предыдущие месяцы

Есть тренд – среднее значение растет

Есть сезонная составляющая – периодический процесс с периодом в сутки

## Анализ и предобработка данных

```
In [18]: taxi=taxi.resample('1H').sum()
```

Данные ресэмплированы – раз в час, суммированием заказов за час

```
In [19]: taxi['dayofweek'] = taxi.index.dayofweek
```

```
In [20]: taxi['hour'] = taxi.index.hour
```

Добавлены столбцы:

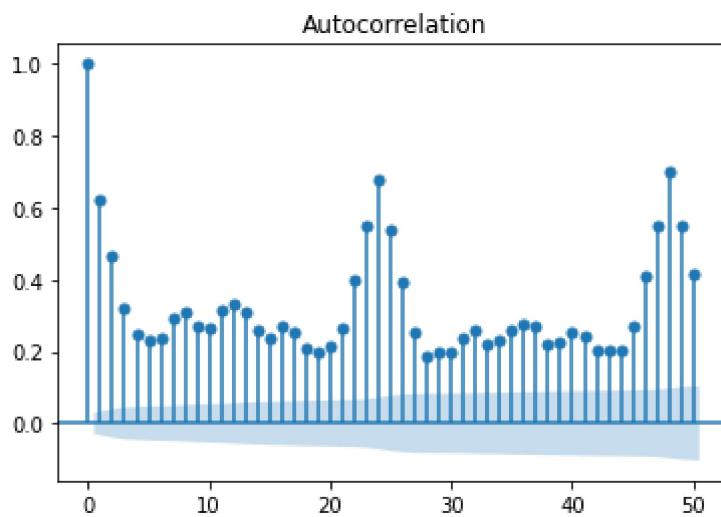
- номер часа (т.к. есть зависимость количества заказов от времени суток)
- день недели (т.к. возможно отличное расписание полетов в зависимости от дня недели, однако это предположение не подтверждено анализом данных)

```
In [21]: taxi.tail()
```

```
Out[21]:
```

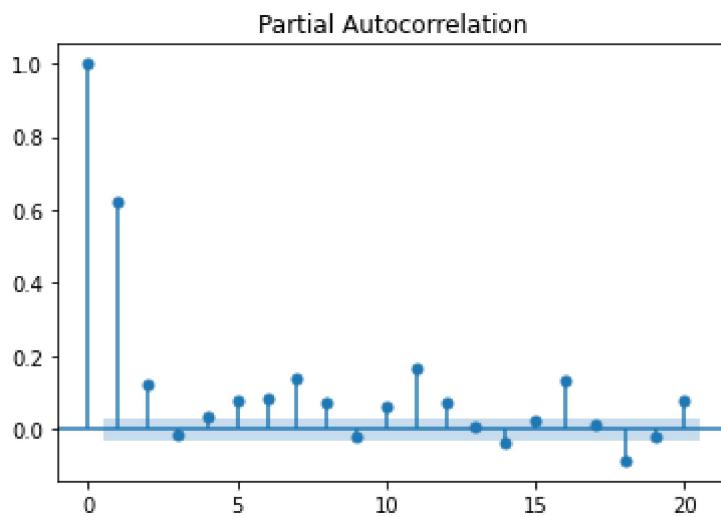
datetime	num_orders	dayofweek	hour
2018-08-31 19:00:00	136	4	19
2018-08-31 20:00:00	154	4	20
2018-08-31 21:00:00	159	4	21
2018-08-31 22:00:00	223	4	22
2018-08-31 23:00:00	205	4	23

```
In [22]: smt.graphics.plot_acf(taxi['num_orders'], lags=50, alpha=0.05)  
plt.show()
```



Построен график автокорреляционной функции для оценки взаимосвязь сигнала от его более раннего значения с глубиной 50 тактов. Все значения автокорреляционной функции можно считать значимыми. Наблюдаются пики с периодом 24.

```
In [23]: smt.graphics.plot_pacf(taxi['num_orders'], lags=20, alpha=0.05)  
plt.show()
```

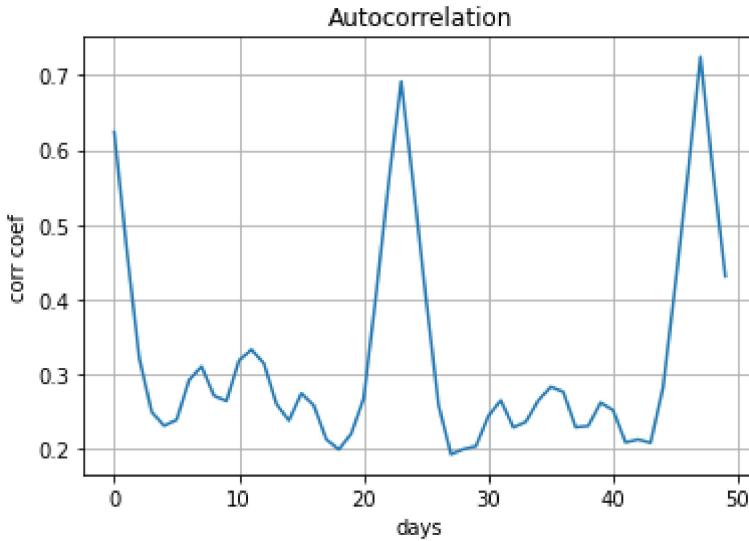


Построен график частичной автокорреляционной функции для оценки взаимосвязь сигнала от его более раннего значения с глубиной 50 тактов. Значения с лагом 1 и 2

автокорреляционной функции можно считать значимыми. Наблюдаются пики с периодом 24.

```
In [24]: Cor=corr_fun(taxi['num_orders'],taxi['num_orders'],50)
```

```
In [25]: my_plot(pd.Series(Cor), 'days', 'corr coef', 'Autocorrelation')
```



Построен график частичной автокорреляционной функции для оценки взаимосвязь сигнала от его более раннего значения с глубиной 50 тактов (собственная функция). Наблюдаются пики с периодом 24.

```
In [26]: def lag_add(data, lags):
    """Добавление признаков-лагов со сдвигом на количество тактов, указанных в списке
    data - данные
    lags - список тактов

    Возвращает:
    ничего
    """
    for i in lags:
        data['lag_{}'.format(i)] = data['num_orders'].shift(i)
```

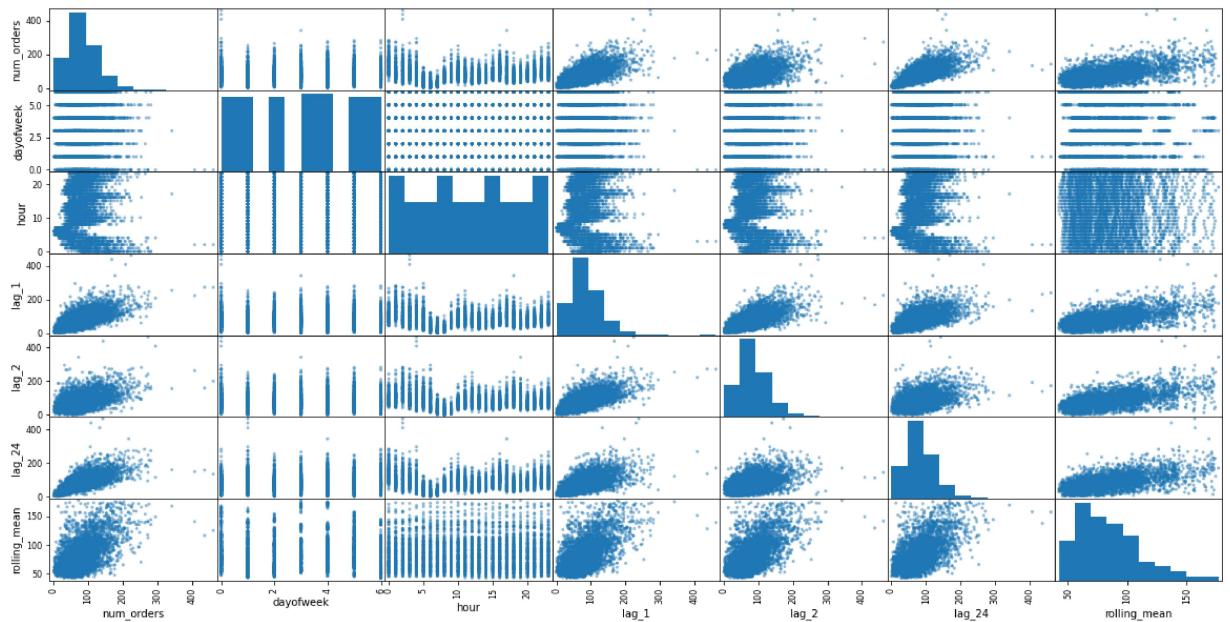
```
In [27]: lag_add(taxi, [1,2,24])
```

```
In [28]: taxi['rolling_mean'] = taxi['num_orders'].shift(1).rolling(24).mean()
```

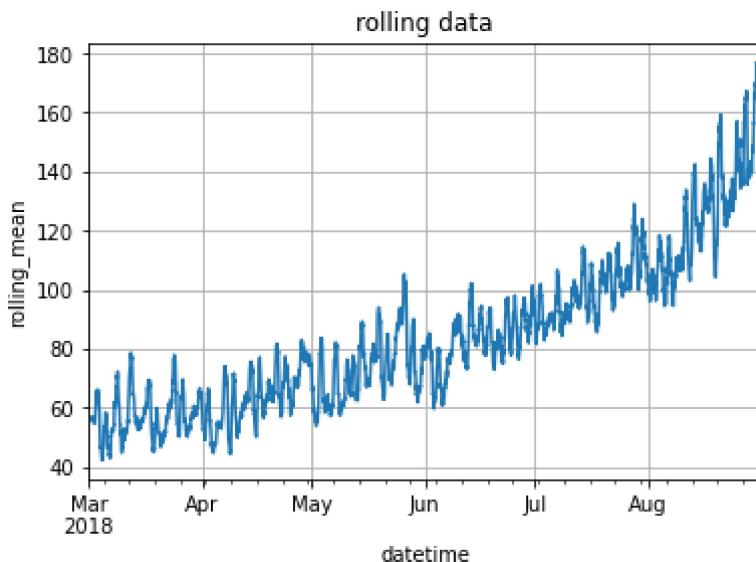
Добавлены столбцы:

- лаг в 1,2 и 24 часа (по данным результатов разложения ряда на составляющие и автокорреляционным функциям)
- сглаженное значение (период сглаживания 24)

```
In [29]: pd.plotting.scatter_matrix(taxi,figsize=(20,10))
plt.show()
```



```
In [30]: my_plot(taxi['rolling_mean'], 'datetime', 'rolling_mean', 'rolling data')
```



## Вывод

### Описание данных:

Данные содержат информацию о заказах такси каждые 10 мин. в течение периода с 1 марта 2018 г. по 31 августа 2018 г. (26496 значений)

Данные монотонны

Ряд гетероскедастичен - дисперсия данных в августе больше, чем в предыдущие месяцы

Есть тренд - среднее значение растет

Есть сезонная составляющая - периодический процесс с периодом в сутки

### Предобработка данных:

1) Данные реэмплированы - раз в час, суммированием заказов за час

2) Добавлены столбцы:

- номер часа (т.к. есть зависимость количества заказов от времени суток)
- день недели (т.к. возможно отличное расписание полетов в зависимости от дня недели, однако это предположение не подтверждено анализом данных)

- лаг в 1,2 и 24 часа (по данным результатов разложения ряда на составляющие и автокорреляционным функциям)
- сглаженное значение (период сглаживания 24) для определения тренда

## Создание моделей

### Подготовка к обучению моделей

#### Создание выборок

```
In [31]: train, test = train_test_split(taxi, shuffle=False, test_size=0.1)
train1, val = train_test_split(train, shuffle=False, test_size=0.1)
```

Данные разделены на обучающую, валидационную и тестовую выборки. В тестовую выборку помещено 10% последних по времени значений рядов.

```
In [32]: taxi['num_orders'].std()
```

```
Out[32]: 45.023853419354054
```

```
In [33]: train['num_orders'].std()
```

```
Out[33]: 38.67181161760075
```

```
In [34]: train1['num_orders'].std()
```

```
Out[34]: 35.99398079032614
```

```
In [35]: val['num_orders'].std()
```

```
Out[35]: 46.43664023042883
```

```
In [36]: test['num_orders'].std()
```

```
Out[36]: 58.61595454245
```

Дисперсия подвыборок отличается - имеет тенденцию к росту, что может негативно отразится на качестве модели.

```
In [37]: train=train.dropna()
```

```
In [38]: train1=train1.dropna()
```

#### Дамми модели

```
In [39]: class Dummy_model_mean:
    def fit(self, train, target):
        self.y = target.mean()

    def predict(self, test):
        return np.zeros(test.shape[0])+(self.y)
```

```
In [40]: class Dummy_model_last:
    def fit(self, train, target):
        self.last = target.shift(1).iloc[-1]

    def predict(self, test):
```

```
    test.loc[test.index[0], 'lag_1']=self.last
    return test['lag_1']
```

Созданы 2 дамми-модели:

- модель Dummy\_model\_mean определяет среднее значение ряда на обучающей выборке и использует его для предсказания;
- модель Dummy\_model\_last всегда предсказывает предыдущее значение.

## Функция для обучения и оценки моделей методом кросс-валидации

```
In [41]: def model_cv(Type_of_model, params, target, cv, show_res=True, scaler=None):
    """Методом кросс-валидации определяет среднее значение RMSE на валидационных, те
    Обучает модель на всем датасете

    Type_of_model - тип модели
    params - данные
    target - целевой параметр
    cv - количество батчей для кросс-валидации
    show_res - печать результата
    scaler - метод масштабирования (Если None, не масштабировать)

    Возвращает:
    модель, объект для масштабирования данных, среднее значение RMSE на валидационны
    тестовых подвыборках, RMSE обученной модели на тестовой выборке.
    """

    # копирование датасета, т.к. передан по ссылке
    data=params.copy()
    # создание объекта для масштабирования
    scale=scaler
    # создание объекта модели
    m=Type_of_model
    # определение длины батча
    bathc_l=data.shape[0]//(cv+1)

    RMSE_train=0
    RMSE_val=0

    # перебор батчей
    for i in range(1,cv+1):
        # разделение датасета на тренировочный и тестовый по принципу - подвыборка р
        # остальное (то, что было до тестовой подвыборки) в обучающую
        cv_train, cv_val = train_test_split(data.iloc[0:data.shape[0]-(bathc_l*(i-1))]

        # масштабирование, если задан способ масштабирования
        if scale!=None:
            scale.fit(X=cv_train.drop([target],axis=1))
            cv_train.loc[:,cv_train.drop([target],axis=1).columns]=scale.transform(X
            cv_val.loc[:,cv_val.drop([target],axis=1).columns]=scale.transform(X=cv_

        # обучение модели
        m.fit(cv_train.drop([target],axis=1),cv_train[target])
        # вычисление RMSE на тренировочной подвыборке
        RMSE_train+=np.sqrt(mean_squared_error(m.predict(cv_train.drop([target],axis=1))

        # вычисление RMSE на валидационной подвыборке
        RMSE_val+=np.sqrt(mean_squared_error(m.predict(cv_val.drop([target],axis=1)))

    # разделение RMSE на количество экспериментов
    RMSE_train/=(cv)
    RMSE_val/=(cv)

    # масштабирование всего датасета, если задан способ масштабирования
    data=params.copy()
```

```

if scale!=None:
    scale.fit(X=data.drop([target],axis=1))
    data.loc[:,data.drop([target],axis=1).columns]=scale.transform(X=data.drop([
        target],axis=1))

# обучение модели на всем датасете
m.fit(data.drop([target],axis=1),data[target])

# вычисление RMSE на тренировочной подвыборке
RMSE_=np.sqrt(mean_squared_error(m.predict(data.drop([target],axis=1)),data[targ

# вывод графиков
if show_res:
    print("RMSE on training sets (CV): ", RMSE_train)
    print("RMSE on validating sets (CV): ", RMSE_val)
    print("RMSE on whole training set: ", RMSE_)

return m, scale, RMSE_train, RMSE_val, RMSE_

```

Создана функция для обучения модели заданного типа и определения средних оценок модели. Методом кросс-валидации определяет среднее значение RMSE на валидационных, тестовых подвыборках. Затем обучает модель на всем датасете. Возвращает модель, объект для масштабирования данных (если задан), среднее значение RMSE на валидационных, тестовых подвыборках, RMSE обученной модели на тестовой выборке.

```

In [42]: def model_plot(m, scale, data, target):
    """Строит график предсказанных и целевых значений

    m - модель
    scaler - объект для масштабирования данных (Если None, не масштабировать)
    data - данные
    target - целевой параметр

    Возвращает:
    ничего
    """
    param=data.copy()
    if scale!=None:
        param.loc[:,param.drop([target],axis=1).columns]=scale.transform(X=param.dro

    pred=pd.Series(m.predict(param.drop([target],axis=1))).reset_index(drop=True)
    targ=pd.Series(param[target]).reset_index(drop=True)

    plt.plot(targ[0:])
    plt.plot(pred[0:])

    plt.xlabel('time')
    plt.ylabel('Number of orders')
    plt.grid(True)
    plt.title('Number of orders model and real output')
    plt.legend(['targ', 'pred'])
    plt.show()

```

Создана функция для построения графика предсказанных и целевых значений

## Обучение моделей функцией model\_cv

```

In [43]: Res=pd.DataFrame(columns=['dummy1', 'dummy2', 'LR','Ridg','XGB','XGB_b','XGB_gscv'],

```

### Dummy1

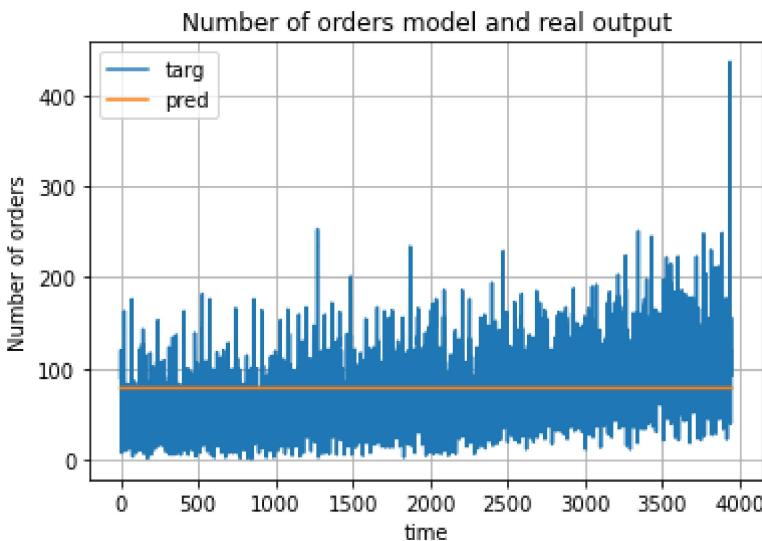
```

In [44]: model1, scale1, *Res.dummy1=model_cv(Dummy_model_mean(), train,'num_orders',cv=3)

```

```
RMSE on training sets (CV): 31.754037017356126  
RMSE on validating sets (CV): 42.61870914835257  
RMSE on whole training set: 38.68289405726269
```

```
In [45]: model_plot(model1, scale1, train, 'num_orders')
```

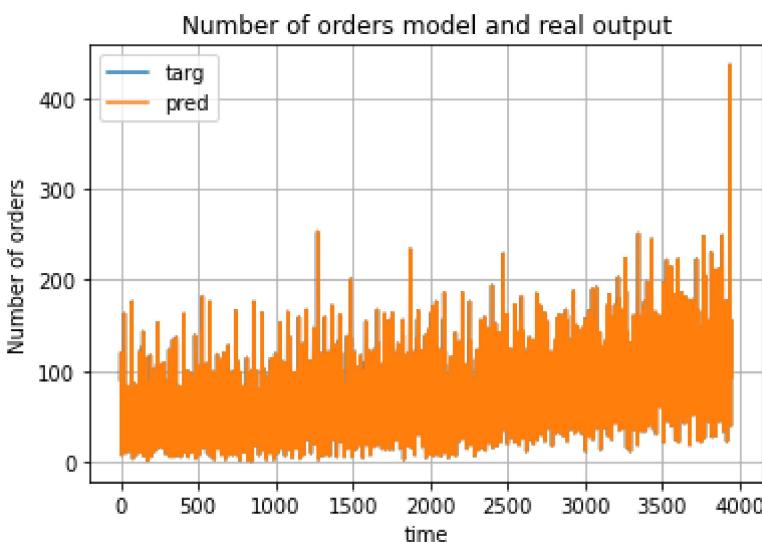


## Dummy2

```
In [46]: model2, scale2, *Res.dummy2=model_cv(Dummy_model_last(), train, 'num_orders', cv=3)
```

RMSE on training sets (CV): 31.27031430051109  
RMSE on validating sets (CV): 38.01673634463308  
RMSE on whole training set: 36.180598339476525

```
In [47]: model_plot(model2, scale2, train, 'num_orders')
```

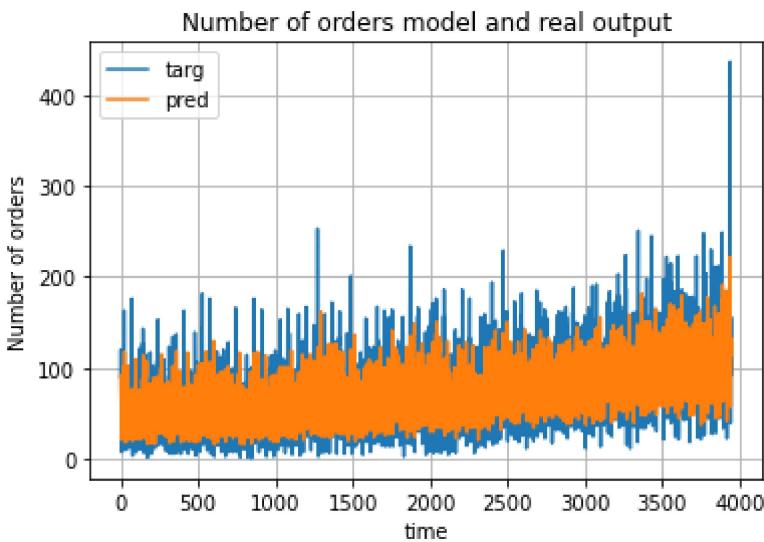


## LinearRegression

```
In [48]: model3, scaler3, *Res.LR=model_cv(LinearRegression(), train, 'num_orders', cv=5, scale
```

RMSE on training sets (CV): 23.2143459605552  
RMSE on validating sets (CV): 27.52525950413577  
RMSE on whole training set: 26.794417845173726

```
In [49]: model_plot(model3, scaler3, train, 'num_orders')
```



```
In [50]: best_par=[100, 0]
for a in [0.5, 1.0, 2]:
    _, _, _, scr, _=model_cv(Ridge(alpha=a), train, 'num_orders', cv=3, show_res=False, sc
    if scr<best_par[0]:
        best_par=[scr, a]
```

```
In [51]: best_par
```

```
Out[51]: [28.34953304821271, 0.5]
```

## Ridge

```
In [52]: model4, scale4, *Res.Ridg=model_cv(Ridge(alpha=best_par[1]), train, 'num_orders', cv=3)

RMSE on training sets (CV): 23.03651587750798
RMSE on validating sets (CV): 28.34953304821271
RMSE on whole training set: 26.794417974647786
```

## XGBRegressor

```
In [53]: best_par=[100, 0, 0, '']
for i in range(80,120,5):
    for a in [0.5, 0.8, 1, 1.5, 2]:
        for s in ['gbtree']:
            param_dist = {'n_estimators':i, 'reg_alpha':a, 'booster':s}
            _, _, _, scr, _=model_cv(XGBRegressor(objective='reg:squarederror'), random_
            if scr<best_par[0]:
                best_par=[scr, param_dist['n_estimators'], param_dist['reg_alpha'], par
```

```
In [54]: best_par
```

```
Out[54]: [28.238755435627088, 80, 0.5, 'gbtree']
```

```
In [55]: param_dist = {'n_estimators':best_par[1], 'reg_alpha':best_par[2], 'booster':best_par
model5, _, *Res.XGB=model_cv(XGBRegressor(objective='reg:squarederror'), random_state=4

RMSE on training sets (CV): 5.568767607618656
RMSE on validating sets (CV): 28.238755435627088
RMSE on whole training set: 9.848313395761686
```

## Обучение модели на разностях

```
In [56]: taxi_d=taxi.copy()
taxi_d['cheak']=taxi_d['num_orders']
```

```

taxi_d['num_orders']=taxi_d['num_orders']-taxi_d['num_orders'].shift(1)
taxi_d['lag_1']=taxi_d['lag_1']-taxi_d['lag_1'].shift(1)
taxi_d['lag_2']=taxi_d['lag_2']-taxi_d['lag_2'].shift(1)
taxi_d['lag_24']=taxi_d['lag_24']-taxi_d['lag_24'].shift(1)
taxi_d['rolling_mean']=taxi_d['rolling_mean']-taxi_d['rolling_mean'].shift(1)

```

Создан датасет, содержащий разницу в количестве заказов вместо абсолютных величин

```
In [57]: train_d, test_d = train_test_split(taxi_d, shuffle=False, test_size=0.1)
train_d=train_d.dropna()
```

```
In [58]: best_par=[100, 0, 0, '']
for i in range(80,120,5):
    for a in [0.5, 0.8, 1, 1.5, 2]:
        for s in ['gbtree']:
            param_dist = {'n_estimators':i, 'reg_alpha':a,'booster':s}
            _, _, _,scr,_=model_cv(XGBRegressor(objective='reg:squarederror',random_
            if scr<best_par[0]:
                best_par=[scr,param_dist['n_estimators'],param_dist['reg_alpha'],par
```

```
In [59]: best_par
```

```
Out[59]: [27.247627904342952, 80, 0.8, 'gbtree']
```

```
In [72]: param_dist = {'n_estimators':best_par[1], 'reg_alpha':best_par[2],'booster':best_par[3]
model6,_, *Res.XGB_b=model_cv(XGBRegressor(objective='reg:squarederror',random_state=42,
RMSE on training sets (CV): 5.600238362472016
RMSE on validating sets (CV): 27.247627904342952
RMSE on whole training set: 10.499436887181469
```

## Обучение моделей GridSearchCV

```
In [61]: from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit

param_dist = {'n_estimators':range(80,120,5), 'reg_alpha':[0.5, 0.8, 1, 1.5, 2],'objective':['reg:squarederror'],
m=XGBRegressor()
model7 = GridSearchCV(m, cv=TimeSeriesSplit(n_splits = 3),param_grid=param_dist, scoring='neg_root_mean_squared_error')
model7.fit(train.drop(['num_orders']),axis=1,train['num_orders'])
```

```
Out[61]: GridSearchCV(cv=TimeSeriesSplit(max_train_size=None, n_splits=3),
estimator=XGBRegressor(base_score=None, booster=None,
colsample_bylevel=None,
colsample_bynode=None,
colsample_bytree=None, gamma=None,
gpu_id=None, importance_type='gain',
interaction_constraints=None,
learning_rate=None, max_delta_step=None,
max_depth=None, min_child_weight=None,
missing=nan, mono...,
n_estimators=100, n_jobs=None,
num_parallel_tree=None, random_state=None,
reg_alpha=None, reg_lambda=None,
scale_pos_weight=None, subsample=None,
tree_method=None, validate_parameters=None,
verbosity=None),
param_grid={'n_estimators': range(80, 120, 5),
'objective': ['reg:squarederror'],
'random_state': [42],
'reg_alpha': [0.5, 0.8, 1, 1.5, 2]},
scoring='neg_root_mean_squared_error')
```

```
In [62]: Res.loc['RMSE_val_cv','XGB_gscv']=model7.best_score_
```

```
In [63]: model7.best_params_
```

```
Out[63]: {'n_estimators': 80,  
          'objective': 'reg:squarederror',  
          'random_state': 42,  
          'reg_alpha': 0.5}
```

```
In [73]: Res
```

```
Out[73]:
```

	dummy1	dummy2	LR	Ridg	XGB	XGB_b	XGB_gscv
RMSE_train_cv	31.754037	31.270314	23.214346	23.036516	5.568768	5.600238	NaN
RMSE_val_cv	42.618709	38.016736	27.525260	28.349533	28.238755	27.247628	28.2388
RMSE_train	38.682894	36.180598	26.794418	26.794418	9.848313	10.499437	NaN

### Промежуточный вывод:

Созданы две дамми модели для оценки полученных результатов.

Подобраны гиперпараметры и обучены 3 модели на датасете с абсолютными величинами - типа LinearRegression(), Ridge(), XGBRegressor().

Обучена модель типа XGBRegressor() на датасете с разностями. **лучший результат**

Значение RMSE на валидационных выборках 27,2

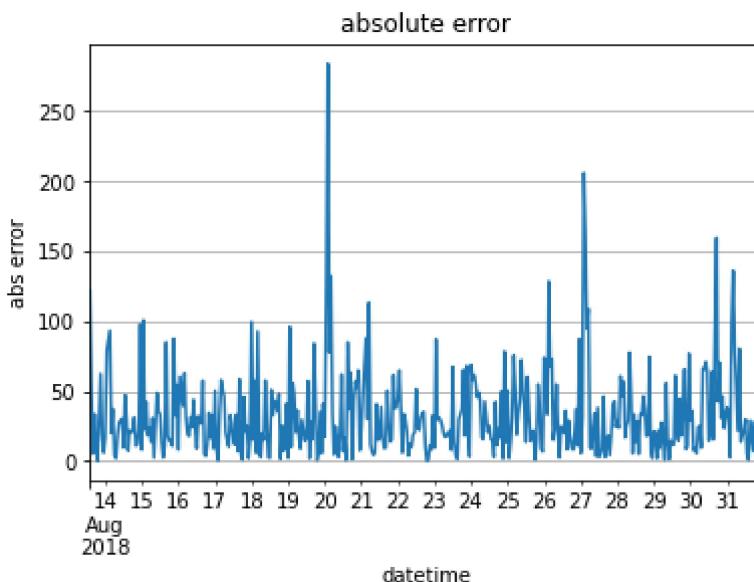
Обучена модель типа XGBRegressor() на датасете с абсолютными величинами с использованием GridSearchCV().

## Тестирование лучшей моделей

```
In [91]: RMSE_=np.sqrt(mean_squared_error(model6.predict(test_d.drop(['cheak','num_orders']),axis=1),  
                                         test_d['abs_error']))  
print("RMSE on test sets: ", RMSE_)
```

RMSE on test sets: 44.465009107170154

```
In [90]: my_plot(np.abs(model6.predict(test_d.drop(['cheak','num_orders'],axis=1))+test_d['abs_error']))
```



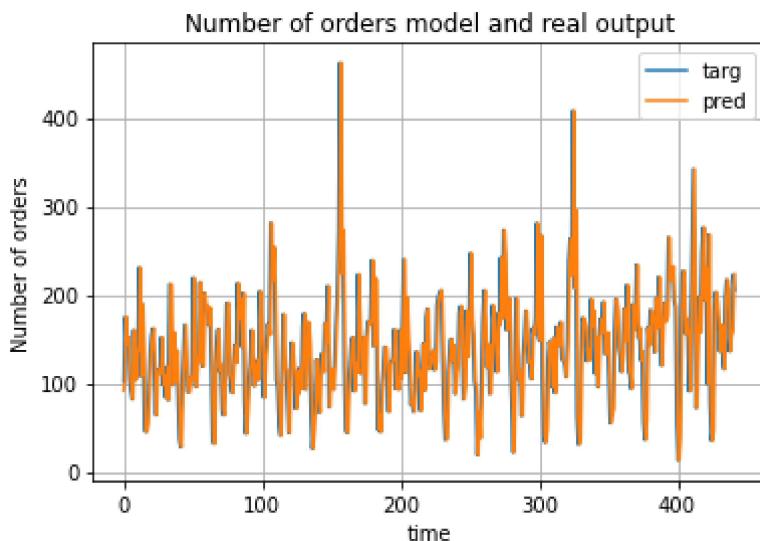
## Сравнение с дамми-моделью

```
In [69]: RMSE_=np.sqrt(mean_squared_error(model2.predict(test_d.drop(['num_orders'],axis=1)),te
```

```
print("RMSE on test sets: ", RMSE_)
```

```
RMSE on test sets: 58.85285355053503
```

```
In [70]: model_plot(model2, _, test, 'num_orders')
```



### Промежуточный вывод:

Значение RMSE на тестовой выборке у выбранной по результатом исследования на обучающей выборке модели, типа XGBRegressor() (44,5), что в 1.5 раза хуже оценок, полученных по результатом исследования модели на обучающей выборке методом кросс-валидации, однако лучше в сравнении с дамми-моделью.

## Вывод

Данные разделены на обучающую и тестовую выборки. В тестовую выборку помещено 10% последних по времени значений рядов.

Созданы 2 дамми-модели:

- модель Dummy\_model\_mean определяет среднее значение ряда на обучающей выборке и использует его для предсказания;
  - модель Dummy\_model\_last всегда предсказывает предыдущее значение.
- Создана функция для обучения модели заданного типа и определения средних оценок модели. Методом кросс-валидации определяет среднее значение RMSE на валидационных, тестовых подвыборках. Затем обучает модель на всем датасете. Возвращает модель, объект для масштабирования данных (если задан), среднее значение RMSE на валидационных, тестовых подвыборках, RMSE обученной модели на тестовой выборке. Создана функция для построения графика предсказанных и целевых значений

Подобраны гиперпараметры и обучены 3 модели - типа LinearRegression(), Ridge(), XGBRegressor() и дамми модели для оценки полученных результатов.

Лучшее (наименьшее) значение RMSE на валидационных выборках у модели XGBRegressor() с гиперпараметрами {'n\_estimators':100, 'reg\_alpha':1,5,'booster':'gbtree'} (26.4)

Значение RMSE на тестовой выборке у выбранной по результатом исследования на обучающей выборке модели, типа XGBRegressor(), с гиперпараметрами {'n\_estimators':100, 'reg\_alpha':1,5,'booster':'gbtree'} (46,5), что в 1.5 раза хуже оценок, полученных по результатом исследования модели на обучающей выборке методом кросс-валидации.

Плохое качество модели вероятно вызвано нестационарность данных, особенно в последней части, попавшей в тестовую выборку:

Ряд гетероскедастичен - дисперсия данных в августе больше, чем в предыдущие месяцы  
Есть тренд - среднее значение растет

## Заключение

### Прогнозирование количества заказов такси на следующий час

**Описание проекта** Компания «Чётенко такси» собрала исторические данные о заказах такси в аэропортах. Чтобы привлекать больше водителей в период пиковой нагрузки, нужно спрогнозировать количество заказов такси на следующий час. Постройте модель для такого предсказания.

#### Описание данных:

Данные лежат в файле taxi.csv. Количество заказов находится в столбце 'num\_orders' (от англ. number of orders, «число заказов»). Данные содержат информацию о заказах такси каждые 10 мин. в течение периода с 1 марта 2018 г. по 31 августа 2018 г. (26496 значений)  
Данные монотонны

Ряд гетероскедастичен - дисперсия данных в августе больше, чем в предыдущие месяцы

Есть тренд - среднее значение растет

Есть сезонная составляющая - периодический процесс с периодом в сутки

#### Предобработка данных:

1) Данные реэмплированы - раз в час, суммированием заказов за час

2) Добавлены столбцы:

- номер часа (т.к. есть зависимость количества заказов от времени суток)
- день недели (т.к. возможно отличное расписание полетов в зависимости от дня недели, однако это предположение не подтверждено анализом данных)
- лаг в 1-2 и 24 часа (по данным результатов разложения ряда на составляющие и автокорреляционным функциям)
- сглаженное значение (период сглаживания 24) для определения тренда

**Подготовка к обучению моделей:** 1) Данные разделены на обучающую и тестовую выборки. В тестовую выборку помещено 10% последних по времени значений рядов.

2) Созданы 2 дамми-модели:

- модель Dummy\_model\_mean определяет среднее значение ряда на обучающей выборке и использует его для предсказания;
  - модель Dummy\_model\_last всегда предсказывает предыдущее значение.
- 3) Создана функция для обучения модели заданного типа и определения средних оценок модели. Методом кросс-валидации определяет среднее значение RMSE на валидационных, тестовых подвыборках. Затем обучает модель на всем датасете. Возвращает модель, объект для масштабирования данных (если задан), среднее значение RMSE на валидационных, тестовых подвыборках, RMSE обученной модели на тестовой выборке.
- 4) Создана функция для построения графика предсказанных и целевых значений

**Обучение моделей:** Подобраны гиперпараметры и обучены 3 модели - типа LinearRegression(), Ridge(), XGBRegressor() и дамми модели для оценки полученных результатов.

Лучшее (наименьшее) значение RMSE на валидационных выборках у модели XGBRegressor() с гиперпараметрами {'n\_estimators':100, 'reg\_alpha':1,5,'booster':'gbtree'} (26.4), что превышает качество дамми моделей

**Тестирование лучшей модели:** Значение RMSE на тестовой выборке у выбранной по результатом исследования на обучающей выборке модели, типа XGBRegressor(), с гиперпараметрами {'n\_estimators':100, 'reg\_alpha':1,5,'booster':'gbtree'} (46,5), что в 1.5 раза хуже оценок, полученных по результатом исследования модели на обучающей выборке методом кросс-валидации.

### **Заключение:**

В результате, полученное значение RMSE удовлетворяет, предъявляемым требованиям (<48).

Плохое качество модели в сравнении с качеством, ожидаемым по результатом кросс-валидации, вероятно вызвано нестационарность данных, особенно в последней части, попавшей в тестовую выборку:

Ряд гетероскедастичен - дисперсия данных в августе больше, чем в предыдущие месяцы  
Есть тренд - среднее значение растет

In [ ]: