Прогнозирование заказов такси

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.

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

Вам нужно:

1. Загрузить данные и выполнить их ресемплирование по одному часу.
2. Проанализировать данные.
3. Обучить разные модели с различными гиперпараметрами. Сделать тестовую выборку размером 10% от исходных данных.
4. Проверить данные на тестовой выборке и сделать выводы.

Данные лежат в файле `taxi.csv`. Количество заказов находится в столбце `num_orders` (от англ. *number of orders*, «число заказов»).

План

1. Подготовка
2. Анализ
3. Обучение
4. Тестирование

1. Подготовка

В [1]:
```
import urllib
import os

import numpy as np
import pandas as pd
import plotly.express as px

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import r2_score
from sklearn.metrics import make_scorer
RMSE = lambda x, y: MSE(x, y) * .5

from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor

from catboost import CatBoostRegressor, cv, Pool

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit

from statsmodels.tsa.seasonal import seasonal_decompose
```

В [2]:
```
name, url = 'datasets/taxi.csv', 'https://code.s3.yandex.net/datasets/taxi.csv'
```

В [3]:
```
os.makedirs(name.split('/')[0], exist_ok=True)
if not os.path.exists(name):
    urllib.request.urlretrieve(url, name)
```

В [4]:
```
data = pd.read_csv(name)
data.sample(3)
```

Out[4]:

	datetime	num_orders
21993	2018-07-31 17:30:00	17
10799	2018-05-14 23:50:00	25
17762	2018-07-02 08:20:00	20

В [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26496 entries, 0 to 26495
Data columns (total 2 columns):
datetime      26496 non-null object
num_orders    26496 non-null int64
dtypes: int64(1), object(1)
memory usage: 414.1+ KB
```

Замечаем, что в первом столбце содержится информация о времени, представленная в текстовом виде. Сделаем из этого столбца индексы для нашего датасета, заранее преобразовав в нужный тип данных. Чтобы проверить являются ли данные временным рядом отсортируем полученный датасет по индексам и проверим на монотонность

В [6]:
```
data = pd.read_csv(name, index_col=[0], parse_dates=[0])
data = data.sort_index()
display(data.sample(3))
```

	num_orders
datetime	
2018-05-25 21:00:00	14
2018-06-12 17:20:00	32
2018-07-26 04:40:00	24

В [7]:
```
data.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):
num_orders    26496 non-null int64
dtypes: int64(1)
memory usage: 414.0 KB
```

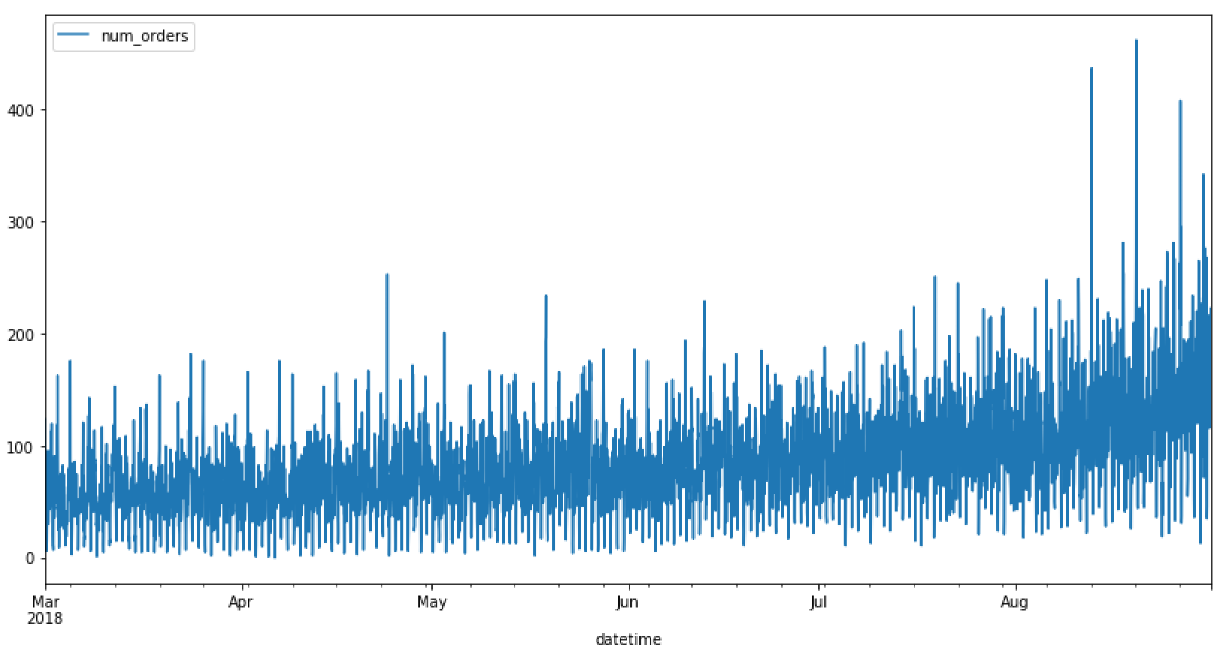
В [8]:
```
display(data.index.is_monotonic)
```

```
True
```

Данные представляют собой временной ряд. Для удобства анализа сделаем ресемплинг.

В [10]:
```
data_resampled = data.resample('1H').sum()
```

В [11]:
```
_ = data_resampled.plot(figsize=[14, 7])
```



2. Анализ

В [12]:
```
data_rolled = data_resampled.rolling(10).mean().dropna()
```

В [13]:
```
data_rolled.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 4407 entries, 2018-03-01 09:00:00 to 2018-08-31 23:00:00
Freq: H
Data columns (total 1 columns):
num_orders    4407 non-null float64
dtypes: float64(1)
memory usage: 68.9 KB
```

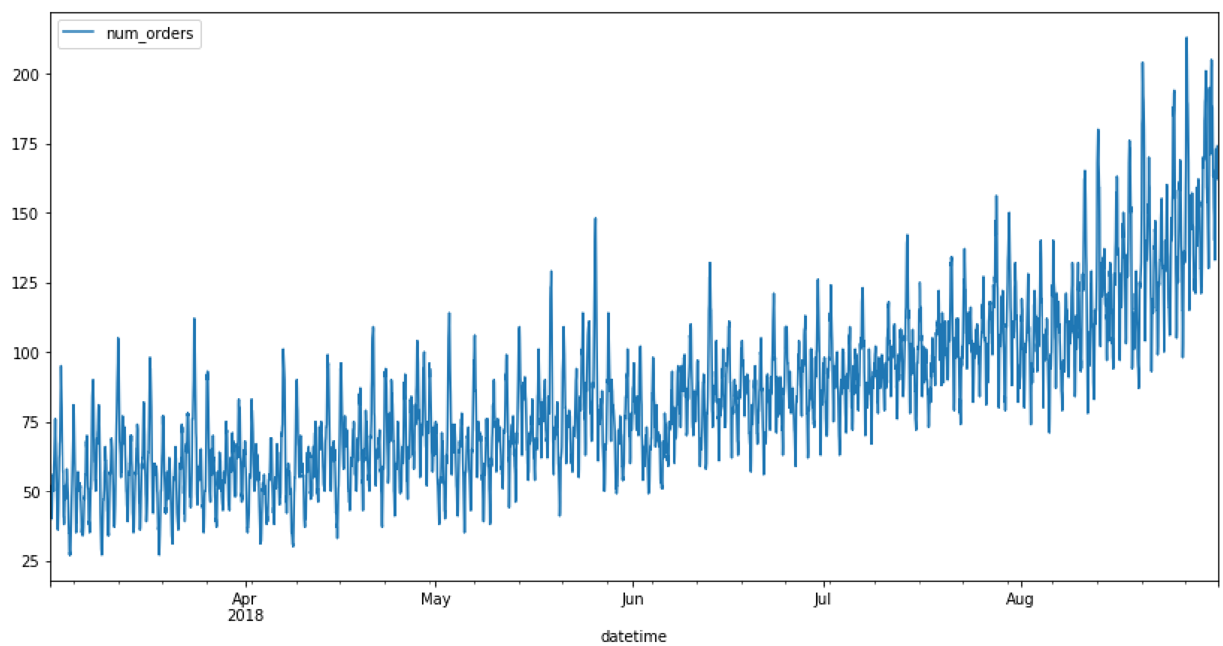
В [14]:
```
data_rolled.describe().T
```

Out[14]:

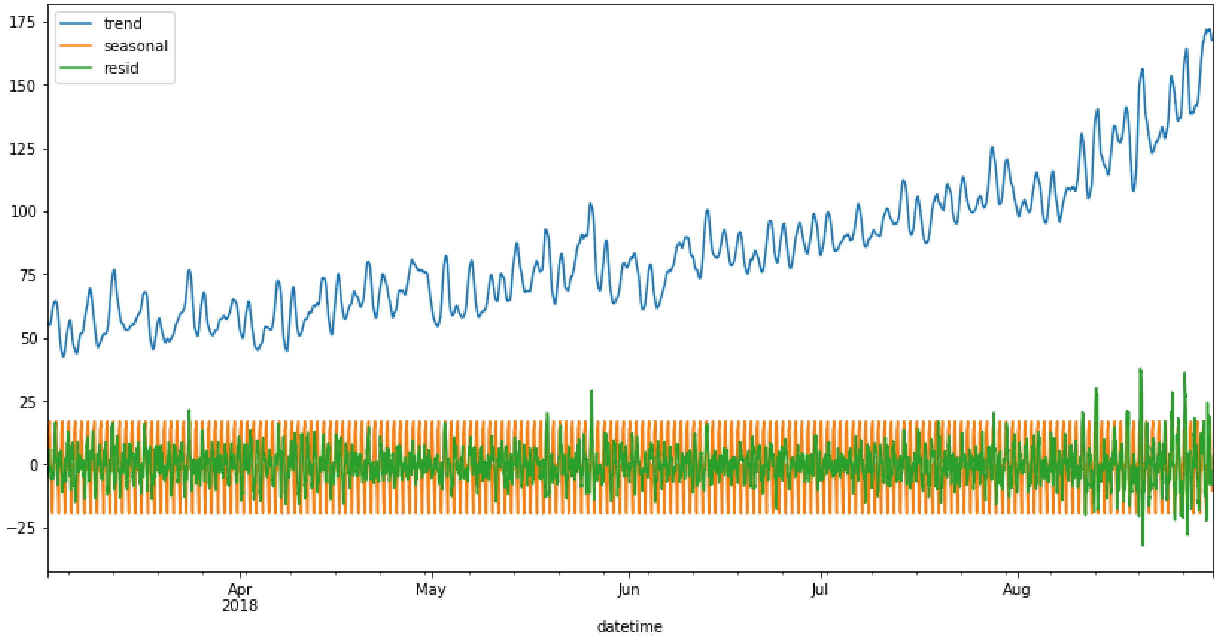
	count	mean	std	min	25%	50%	75%	max
num_orders	4407.0	84.33735	29.72319	27.0	62.6	80.2	100.7	213.4

В [15]:
```
data_rolled = data_rolled.astype(np.uint8)
```

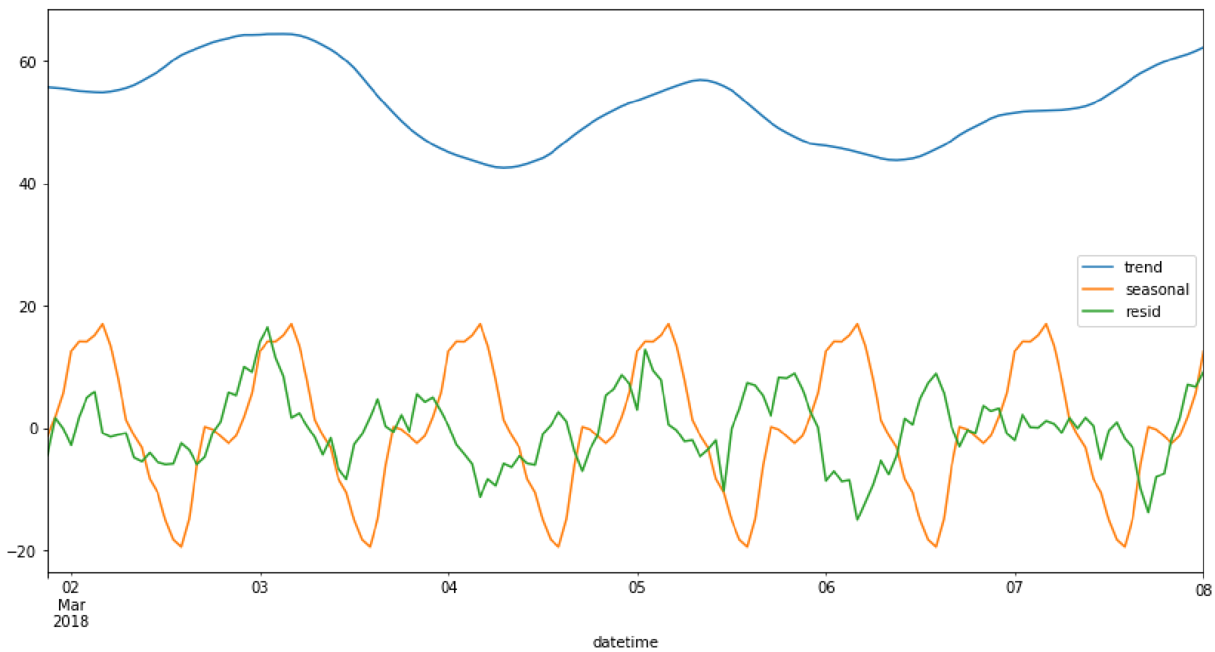
В [16]:
```
_ = data_rolled.plot(figsize=[14, 7])
```



В [17]:
```
data_seas = seasonal_decompose(data_rolled)
data_temp = pd.DataFrame(index=data_rolled.index)
data_temp.loc[:, 'trend'] = data_seas.trend
data_temp.loc[:, 'seasonal'] = data_seas.seasonal
data_temp.loc[:, 'resid'] = data_seas.resid
data_temp = data_temp.dropna()
_ = data_temp.plot(figsize=[14, 7])
# px.line(data_temp)
```



В [18]:
```
_ = data_temp['2018-03-01 00:00:00':'2018-03-08 00:00:00'].plot(figsize=[14, 7])
```



Имеет место посуточная сезонность

3. Обучение

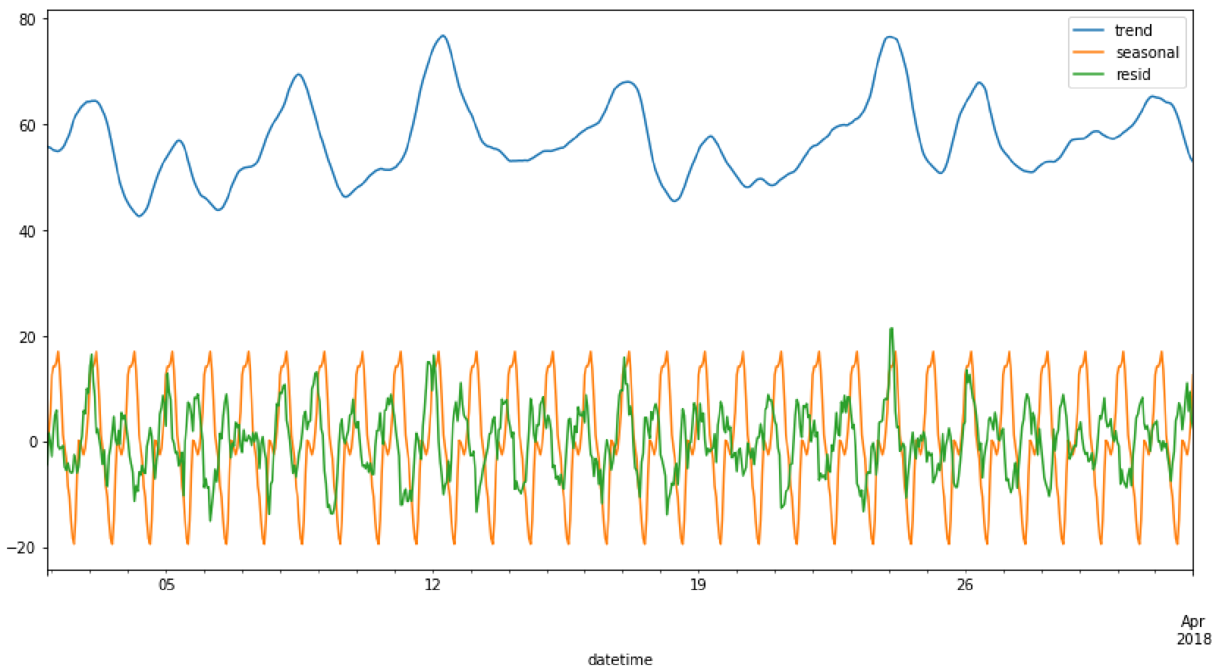
Для самого обучения в исходных данных не хватает признаков, создадим их

В [19]:
```
def make_features(df, max_lag, rolling_mean_size):
    data = df.copy()
#     data['month'] = data.index.month.astype(np.uint8)
#     data['week'] = data.index.isocalendar().week.astype(np.uint8)
#     data['day'] = data.index.day.astype(np.uint8)
    data['hour'] = data.index.hour.astype(np.uint8)
    data['dayofweek'] = data.index.dayofweek.astype(np.uint8)

    for lag in range(1, max_lag + 1):
        data['lag_{}'.format(lag)] = data.iloc[:, 0].shift(lag)

    data['rolling_mean'] = data.iloc[:, 0].shift().rolling(rolling_mean_size).mea
    return data.dropna()
```

В [20]:
```
_ = data_temp['2018-03-01 00:00:00':'2018-04-01 00:00:00'].plot(figsize=[14, 7])
```



В [21]:
```
df = make_features(data_rolled, 2, 3)
```

В [22]:
```
train, test = train_test_split(df, shuffle=False, test_size=.1)
```

В [23]:
```
X_train, y_train = train.drop(['num_orders'], axis=1), train.num_orders
X_test, y_test = test.drop(['num_orders'], axis=1), test.num_orders
```

В [24]:
```
X_train
```

Out[24]:

	hour	dayofweek	lag_1	lag_2	rolling_mean
datetime					
2018-03-01 12:00:00	12	3	47.0	46.0	48.333333
2018-03-01 13:00:00	13	3	43.0	47.0	45.333333
2018-03-01 14:00:00	14	3	40.0	43.0	43.333333
2018-03-01 15:00:00	15	3	40.0	40.0	41.000000
2018-03-01 16:00:00	16	3	46.0	40.0	42.000000
...	...	...	...	...	...
2018-08-13 10:00:00	10	0	159.0	164.0	163.333333
2018-08-13 11:00:00	11	0	159.0	159.0	160.666667
2018-08-13 12:00:00	12	0	146.0	159.0	154.666667
2018-08-13 13:00:00	13	0	111.0	146.0	138.666667
2018-08-13 14:00:00	14	0	107.0	111.0	121.333333

3963 rows × 5 columns

Linear Regression

В [25]:
```
model_linreg = LinearRegression()
model_linreg.fit(X_train, y_train)
```

Out[25]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

В [26]:
```
n_split = lambda test_size_perc: int((100 - test_size_perc) / test_size_perc - 1)
tscv = TimeSeriesSplit(n_splits=n_split(10))
```

Cat Boost Regressor

В [27]:
```
%%time

params = {
    'iterations': [50, 100],
    'depth': [5, 10],
    'learning_rate': [.2, .7],
}

model_catboost = CatBoostRegressor(verbose=False)
gs_cb = model_catboost.grid_search(params, X_train, y_train,\
                                   cv=tscv, verbose=False)
print(gs_cb['params'])
```

```
{'depth': 5, 'iterations': 100, 'learning_rate': 0.2}
CPU times: user 21.6 s, sys: 3.4 s, total: 25 s
Wall time: 45.8 s
```

В [28]:
```
%%time

model_catboost = CatBoostRegressor(verbose=False, **gs_cb['params'])
model_catboost.fit(X_train, y_train)
```

```
CPU times: user 1.27 s, sys: 230 ms, total: 1.5 s
Wall time: 2.52 s
```

Out[28]:
```
<catboost.core.CatBoostRegressor at 0x7f3c31455690>
```

Decision Tree Regressor

В [29]:
```
%%time

params = {
    'max_depth': [10, 20, 50, 75, 100],
}


model_tree = DecisionTreeRegressor(random_state=42)
gs_dt = GridSearchCV(estimator=model_tree, param_grid=params,\
                    scoring=make_scorer(RMSE),\
                     n_jobs=-1, cv=tscv, verbose=1)
gs_dt.fit(X_train, y_train)
print(gs_dt.best_params_, gs_dt.best_score_)
model_tree = gs_dt.best_estimator_
```

```
Fitting 8 folds for each of 5 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend SequentialBackend with 1 concurrent worker
s.

{'max_depth': 50} 17.811221590909092
CPU times: user 467 ms, sys: 0 ns, total: 467 ms
Wall time: 464 ms

[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:    0.4s finished
```

Random Forest Regressor

В [30]:
```
%%time

params = {
    'max_depth': [10, 20, 50],
    'n_estimators': [10, 20]
}

model_forest = RandomForestRegressor(random_state=42)
gs_rf = GridSearchCV(estimator=model_forest, n_jobs=-1,\
                     param_grid=params, cv=tscv,\
                     scoring=make_scorer(RMSE), verbose=1)
gs_rf.fit(X_train, y_train)
print(gs_rf.best_params_, gs_rf.best_score_)
model_forest = gs_rf.best_estimator_
```

```
[Parallel(n_jobs=-1)]: Using backend SequentialBackend with 1 concurrent worker
s.

Fitting 8 folds for each of 6 candidates, totalling 48 fits
{'max_depth': 50, 'n_estimators': 10} 11.815759232954546
CPU times: user 3.98 s, sys: 0 ns, total: 3.98 s
Wall time: 3.99 s

[Parallel(n_jobs=-1)]: Done  48 out of  48 | elapsed:    3.9s finished
```

4. Тестирование

В [32]:
```
models = {}
```

Linear Regression

В [33]:
```
pred_linreg_train = model_linreg.predict(X_train)
pred_linreg_test = model_linreg.predict(X_test)

models['linear'] = {}
models['linear']['rmse_train'] = round(RMSE(y_train, pred_linreg_train), 3)
models['linear']['rmse_test'] = round(RMSE(y_test, pred_linreg_test), 3)
models['linear']['r2'] = round(r2_score(y_test, pred_linreg_test), 3)
```

Decision Tree Regressor

В [35]:
```
pred_tree_train = model_tree.predict(X_train)
pred_tree_test = model_tree.predict(X_test)

models['tree'] = {}
models['tree']['rmse_train'] = round(RMSE(y_train, pred_tree_train), 3)
models['tree']['rmse_test'] = round(RMSE(y_test, pred_tree_test), 3)
models['tree']['r2'] = round(r2_score(y_test, pred_tree_test), 3)
```

Random Forest Regressor

В [36]:
```
pred_forest_train = model_forest.predict(X_train)
pred_forest_test = model_forest.predict(X_test)

models['forest'] = {}
models['forest']['rmse_train'] = round(RMSE(y_train, pred_forest_train), 3)
models['forest']['rmse_test'] = round(RMSE(y_test, pred_forest_test), 3)
models['forest']['r2'] = round(r2_score(y_test, pred_forest_test), 3)
```

Cat Boost Regressor (gread_searched)

В [37]:
```
pred_catboost_train = model_catboost.predict(X_train)
pred_catboost_test = model_catboost.predict(X_test)

models['catboost_grid_searched'] = {}
models['catboost_grid_searched']['rmse_train'] = round(RMSE(y_train, pred_catboos
models['catboost_grid_searched']['rmse_test'] = round(RMSE(y_test, pred_catboost_
models['catboost_grid_searched']['r2'] = round(r2_score(y_test, pred_catboost_tes
```

В [38]:
```
model_catboost = CatBoostRegressor(verbose=False)
model_catboost.fit(X_train, y_train)
pred_catboost_train = model_catboost.predict(X_train)
pred_catboost_test = model_catboost.predict(X_test)

models['catboost'] = {}
models['catboost']['rmse_train'] = round(RMSE(y_train, pred_catboost_train), 3)
models['catboost']['rmse_test'] = round(RMSE(y_test, pred_catboost_test), 3)
models['catboost']['r2'] = round(r2_score(y_test, pred_catboost_test), 3)
```

В [39]:
```
pd.DataFrame(models).T
```

Out[39]:

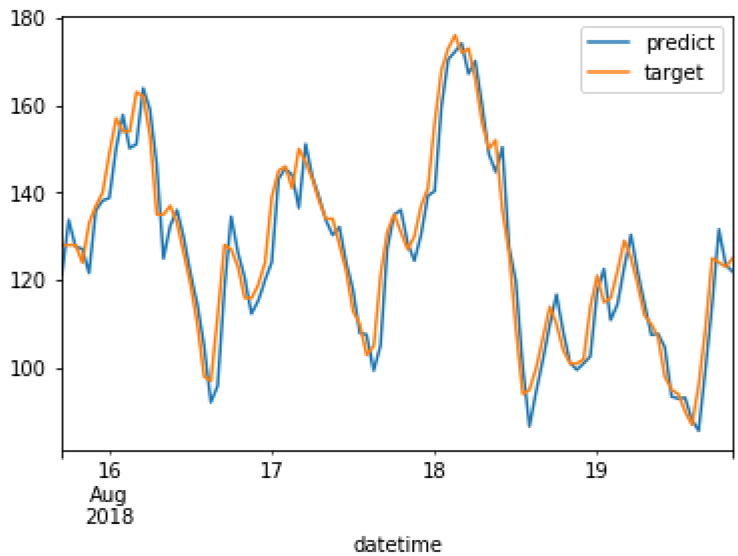
	rmse_train	rmse_test	r2
linear	9.712	29.631	0.910
tree	0.005	64.388	0.805
forest	1.223	52.109	0.842
catboost_grid_searched	4.451	114.808	0.652
catboost	3.526	112.950	0.658

Итоговый вывод

Для данной задачи лучше использовать `Linear Regression`. Все остальные рассматриваемые модели переобучились. Неожиданно плохо показала себя модель `CatBoostRegressor`.

В [40]:
```
a, b = 50, 150
check = pd.DataFrame(index=X_test.iloc[a:b, :].index)
check.loc[:, 'predict'] = pred_linreg_test[a:b]
check.loc[:, 'target'] = y_test[a:b]
# px.line(check)
check.plot()
```

Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3c31169c90>



В [41]:
```
a, b = 10, 30
check = pd.DataFrame(index=X_test.iloc[a:b, :].index)
check.loc[:, 'predict'] = pred_linreg_test[a:b]
check.loc[:, 'target'] = y_test[a:b]
# px.line(check)
_ = check.plot()
```