

[https://www.kdnuggets.com/2023/08/leveraging-xgboost-timeseries-forecasting.html#:~:text=Often%20the%20use%20cases%20are,XGBoost%20can%20also%](https://www.kdnuggets.com/2023/08/leveraging-xgboost-timeseries-forecasting.html#:~:text=Often%20the%20use%20cases%20are,XGBoost%20can%20also%20be%20used%20for%20time%20series%20forecasting,)
<https://www.kaggle.com/competitions/store-sales-time-series-forecasting>

```
In [31]: %load_ext autoreload
%autoreload 2

import sys
sys.path.append("../")

import pandas as pd
import numpy as np
import datetime as dt
import statsmodels.api as sm
import statsmodels.graphics as sg

import itertools

from sklearn.model_selection import GridSearchCV

import matplotlib.pyplot as plt

import seaborn as sns
sns.set_theme()

from sklearn.metrics import r2_score, mean_squared_error
from sklearn.linear_model import LinearRegression

import statsmodels.api as sm
import statsmodels.formula.api as smf
import statsmodels.stats as st

from sklearn import metrics

import seaborn as sns
import matplotlib.pyplot as plt

import xgboost as xgb

from IPython.display import display, clear_output

%matplotlib inline
```

The autoreload extension is already loaded. To reload it, use:
%reload_ext autoreload

```
In [32]: df = pd.read_csv('../data/nts.csv', index_col=0, parse_dates=True)
df.shape, df.columns
```

```
Out[32]: ((42, 15),
          Index(['rynek', 'inflacja_r', 'inflacja_q', 'stopa_procentowa',
                'liczba_kredytow', 'tempo_wzrostu', 'ufnosc', 'duze_zakupy',
                'bezrobocie', 'spr_detaliczna', 'pkb', 'wynag', 'inflacja_q_ku
                m',
                'ogolem', 'rpk'],
                dtype='object'))
```

```
In [200]: df["quarter"] = pd.PeriodIndex(df.index, freq='Q').map(lambda r: int(str(
```

drugiej połowie 2023 wynika z uruchomienia rządowego programu kredytów z preferencyjnym oprocentowaniem Bezpieczny Kredyt 2% (program był aktywny w okresie 01-07-2023 do 31-12-2023)

```
In [33]: df.head()
```

```
Out[33]:
```

| | rynek | inflacja_r | inflacja_q | stopa_procentowa | liczba_kredytow | ten |
|------------|-------------|------------|------------|------------------|-----------------|-----|
| 2014-03-31 | 6193.211456 | 100.6 | 100.2 | 0.0446 | 41942.0 | |
| 2014-06-30 | 6227.984614 | 100.3 | 100.0 | 0.0447 | 45499.0 | |
| 2014-09-30 | 6357.260998 | 99.7 | 99.5 | 0.0443 | 43653.0 | |
| 2014-12-31 | 6438.874574 | 99.3 | 99.6 | 0.0388 | 42993.0 | |
| 2015-03-31 | 6473.616559 | 98.5 | 99.5 | 0.0348 | 42169.0 | |

```
In [59]: bn, fn = 35, 42
         ldf = df[['rynek', 'wynag', 'inflacja_q_kum']]
         train, test = ldf.iloc[:bn], ldf.iloc[bn:fn]
         train.index, test.index, len(train), len(test)
```

```
Out[59]: (DatetimeIndex(['2014-03-31', '2014-06-30', '2014-09-30', '2014-12-31',
                        '2015-03-31', '2015-06-30', '2015-09-30', '2015-12-31',
                        '2016-03-31', '2016-06-30', '2016-09-30', '2016-12-31',
                        '2017-03-31', '2017-06-30', '2017-09-30', '2017-12-31',
                        '2018-03-31', '2018-06-30', '2018-09-30', '2018-12-31',
                        '2019-03-31', '2019-06-30', '2019-09-30', '2019-12-31',
                        '2020-03-31', '2020-06-30', '2020-09-30', '2020-12-31',
                        '2021-03-31', '2021-06-30', '2021-09-30', '2021-12-31',
                        '2022-03-31', '2022-06-30', '2022-09-30'],
                        dtype='datetime64[ns]', freq=None),
         DatetimeIndex(['2022-12-31', '2023-03-31', '2023-06-30', '2023-09-30',
                        '2023-12-31', '2024-03-31', '2024-06-30'],
                        dtype='datetime64[ns]', freq=None),
         35,
         7)
```

```
In [67]: X_train = train.drop('rynek', axis=1)
         y_train = train['rynek']

         X_test = test.drop('rynek', axis=1)
         y_test = test['rynek']
```

```
In [66]: reg = xgb.XGBRegressor(
         max_depth=4,
         learning_rate=0.02,
         n_estimators=300,
```

```

    )
    reg.fit(X_train, y_train, verbose=True)
    test['rynek_Prediction'] = reg.predict(X_test)

    fig, axes = plt.subplots(nrows=1, ncols=2)
    xgb.plot_importance(reg, ax=axes[0])
    train['rynek'].plot(style='k', figsize=(10,5), label='train', ax=axes[1])
    test['rynek'].plot(style='b', figsize=(10,5), label='test', ax=axes[1])
    test['rynek_Prediction'].plot(style='r', figsize=(10,5), label='prediction')
    plt.title('train and prediction')
    plt.legend()

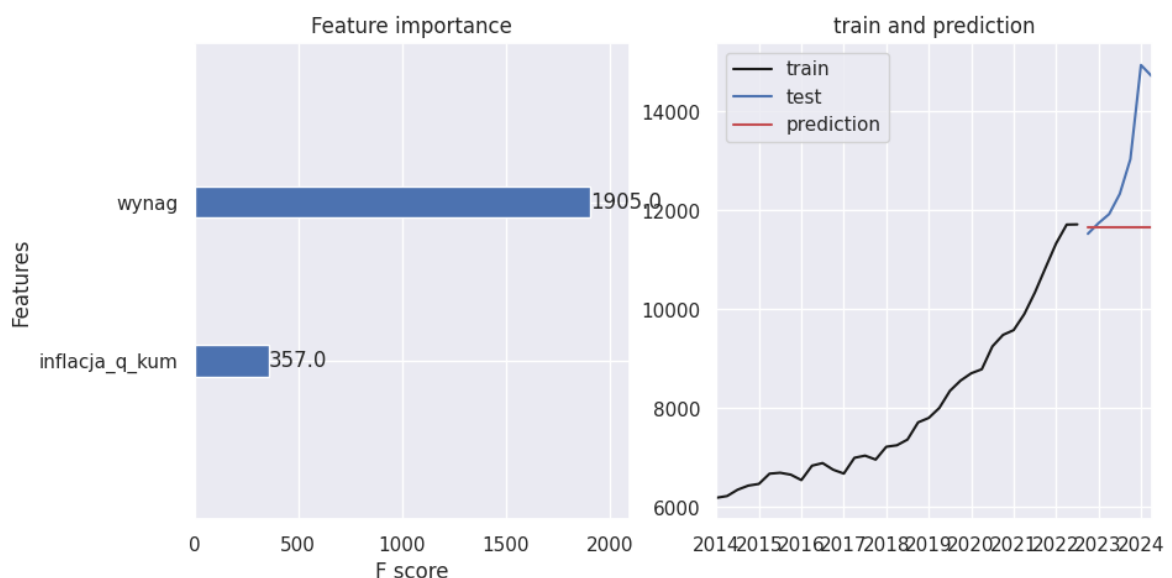
```

/tmp/ipykernel_98298/2262089887.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test['rynek_Prediction'] = reg.predict(X_test)
```

Out[66]: <matplotlib.legend.Legend at 0x7e369df03050>



```

In [68]: print('RMSE: ', round(metrics.mean_squared_error(y_true=test['rynek'], y_p
print('MAE: ', round(metrics.mean_absolute_error(y_true=test['rynek'], y_p
print('MAPE: ', round(metrics.mean_absolute_percentage_error(y_true=test[

```

RMSE: 3252215.671

MAE: 1275.807

MAPE: 0.09

<https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/>

In []:

<https://medium.com/@rithpansanga/optimizing-xgboost-a-guide-to-hyperparameter-tuning-77b6e48e289d>

```

In [143... bn, fn = 39, 42
ldf = df[['rynek', 'wynag', 'inflacja_q_kum', 'rpk']]
train, test = ldf.iloc[:bn], ldf.iloc[bn:fn]
train.index, test.index, len(train), len(test)

```

```
X_train = train.drop('rynek', axis=1)
y_train = train['rynek']

X_test = test.drop('rynek', axis=1)
y_test = test['rynek']
X_train.columns,
```

Out[143... (Index(['wynag', 'inflacja_q_kum', 'rpk'], dtype='object'),)

```
In [144... param_grid = {
    'max_depth': [4, 5, 7],
    'max_leaves': [0, 9],
    'learning_rate': [0.2, 0.1, 0.01,],
    'subsample': [0.6, 0.8, 1],
    'n_estimators': [240, 600, 100]
}
```

```
In [145... pairs = [(k, _v) for _v in v] for k,v in param_grid.items()]
num = np.prod([len(p) for p in pairs])
num
```

Out[145... 162

```
In [146... pairs = [(k, _v) for _v in v] for k,v in param_grid.items()]
num = np.prod([len(p) for p in pairs])
res = {}
counter = 0
for it in itertools.product(*pairs):
    display(str(round(counter / num, 4) * 100) + " %")
    counter += 1
    d = dict(it)
    model = xgb.XGBRegressor(**d)
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    rmse = metrics.mean_squared_error(y_true=test['rynek'], y_pred=pred)
    mae = metrics.mean_absolute_error(y_true=test['rynek'], y_pred=pred)
    mape = metrics.mean_absolute_percentage_error(y_true=test['rynek'], y_
    res[tuple(d.values())] = (rmse, mae, mape)
    clear_output(wait=True)
```

'99.38 %'

```
In [147... for k, v in sorted(res.items(), key=lambda r: r[1][0]):

    d = {
        'max_depth': k[0],
        'max_leaves': k[1],
        'learning_rate': k[2],
        'subsample': k[3],
        'n_estimators': k[4]
    }
    print(k, v)
    break
```

(4, 0, 0.2, 0.6, 600) (4322908.863174175, 1897.4589588425151, 0.1300807872314772)

```

In [148... # d = {
#         'max_depth': 4,
#         'max_leaves': 0,
#         'learning_rate': 0.2,
#         'subsample': 0.6,
#         'n_estimators': 600
# }
model = xgb.sklearn.XGBRegressor(**d)
model.fit(X_train, y_train)

pred = model.predict(X_test)

rmse = metrics.mean_squared_error(y_true=test['rynek'], y_pred=pred)
mae = metrics.mean_absolute_error(y_true=test['rynek'], y_pred=pred)
mape = metrics.mean_absolute_percentage_error(y_true=test['rynek'], y_pred=pred)
res[tuple(d.values())] = (rmse, mae, mape)
test['rynek_Prediction'] = pred

fig, axes = plt.subplots(nrows=1, ncols=2)
xgb.plot_importance(model, ax=axes[0])
train['rynek'].plot(style='k', figsize=(10,5), label = 'train', ax=axes[1])
test['rynek'].plot(style='b', figsize=(10,5), label = 'test', ax=axes[1])
test['rynek_Prediction'].plot(style='r', figsize=(10,5), label = 'predict')
plt.title('train and prediction')
plt.legend()

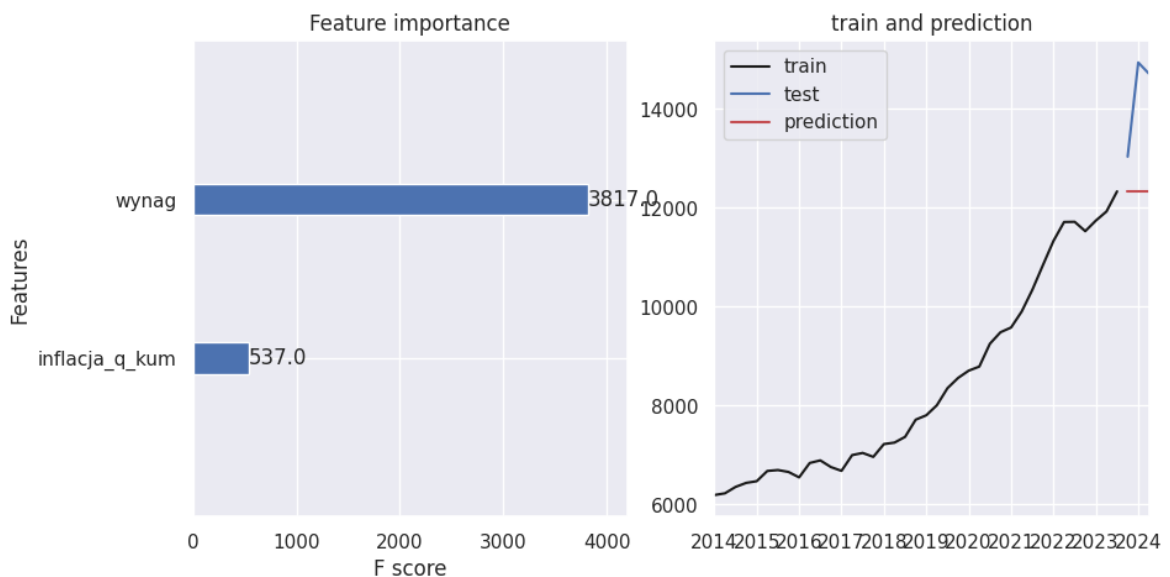
```

/tmp/ipykernel_98298/1260853665.py:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
test['rynek_Prediction'] = pred
```

Out[148... <matplotlib.legend.Legend at 0x7e369c3d7080>



```
In [126... model.feature_names_in_, X_test
```

```
Out[126...] (array(['ufnosc', 'wynag', 'inflacja_q_kum', 'bezrobocie', 'rpk'],
      dtype='<U14'),
            ufnosc      wynag      inflacja_q_kum      bezrobocie      rpk
2022-12-31    -43.8    6850.330727         1.322818         5.2      0.0
2023-03-31    -36.6    7062.406358         1.370440         5.4      0.0
2023-06-30    -30.1    7278.507128         1.429369         5.1      0.0
2023-09-30    -22.5    7498.418749         1.456527         5.0      1.0
2023-12-31    -16.1    7684.873442         1.452157         5.1      1.0
2024-03-31    -12.5    7954.361153         1.459418         5.3      0.0
2024-06-30    -12.4    8144.805614         1.472553         4.9      0.0)
```

In [127...]

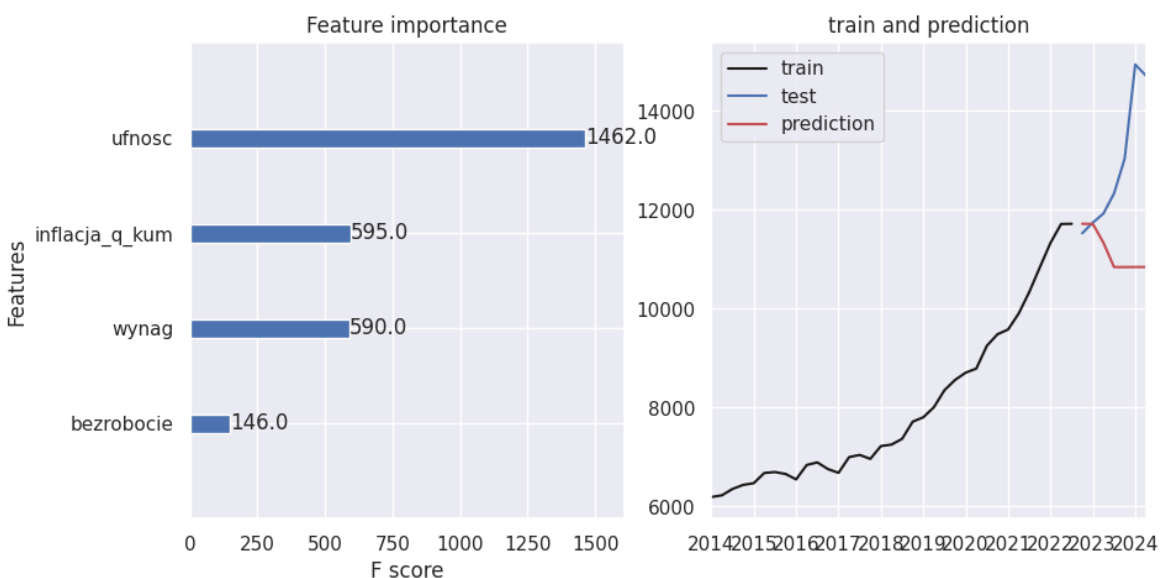
```
In [129...] rmse = metrics.mean_squared_error(y_true=test['rynek'], y_pred=pred)
mae = metrics.mean_absolute_error(y_true=test['rynek'], y_pred=pred)
mape = metrics.mean_absolute_percentage_error(y_true=test['rynek'], y_pred=pred)
res[tuple(d.values())] = (rmse, mae, mape)
test['rynek_Prediction'] = pred

fig, axes = plt.subplots(nrows=1, ncols=2)
xgb.plot_importance(model, ax=axes[0])
train['rynek'].plot(style='k', figsize=(10,5), label = 'train', ax=axes[1])
test['rynek'].plot(style='b', figsize=(10,5), label = 'test', ax=axes[1])
test['rynek_Prediction'].plot(style='r', figsize=(10,5), label = 'predict')
plt.title('train and prediction')
plt.legend()
```

/tmp/ipykernel_98298/3665934210.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
test['rynek_Prediction'] = pred

Out[129...] <matplotlib.legend.Legend at 0x7e369d7f7f20>



In []: