Библиотеки

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.dates import MonthLocator, DateFormatter


sns.set_style('white')
sns.set(rc={'figure.figsize':(14, 4)})
```

Загружаем данные

В качестве дата сета для выполнения заданий я выбрал данные по электроэнергии для Дании. Таймпоинт 60 мин

В дата сете присутствует следующая информация:

Syntax	Description
et_cest_timestamp	таймштамп
load_actual_entsoe_transparency	энергопотребление
load_forecast_entsoe_transparency	прогнозируемое энергопотребление
solar_capacity	вместимость для солнечной энергии
solar_generation_actual	выработанная солнечная энергия
wind_capacity	вместимость для ветряной энергии (общее)
wind_generation_actual	выработання ветряная энергия (общее)
wind_offshore_capacity	вместимость для ветряной энергии (в море)
wind_offshore_generation_actual	выработання ветряная энергия (в море)
wind_onshore_capacity	выработання ветряная энергия (на суше)
wind_onshore_generation_actual	выработання ветряная энергия (на суше)

```
path = "time_series_60min_dk.csv"
df = pd.read_csv(path, index_col='utc_timestamp', parse_dates=True)
df.head()
```

utc_timestamp	cet_cest_timestamp	DK_load_actual_entsoe_transparer
2014-12-31 23:00:00+00:00	2015-01-01T00:00:00+0100	N
2015-01-01 00:00:00+00:00	2015-01-01T01:00:00+0100	N

2015-01-01 01:00:00+00:00	2015-01-01T02:00:00+0100	3100
2015-01-01 02:00:00+00:00	2015-01-01T03:00:00+0100	2980
2015-01-01 03:00:00+00:00	2015-01-01T04:00:00+0100	2933

```
df.index = df.index.strftime('%Y-%m-%d-%H')
```

utc_timestamp	cet_cest_timestamp	DK_load_actual_entsoe_transparenc
2014-12-31-23	2015-01-01T00:00:00+0100	Na
2015-01-01-00	2015-01-01T01:00:00+0100	Na
2015-01-01-01	2015-01-01T02:00:00+0100	3100.0
2015-01-01-02	2015-01-01T03:00:00+0100	2980.3
2015-01-01-03	2015-01-01T04:00:00+0100	2933.4

```
# na's by index
isna_df = df.isna()
df.isna().sum()
```

```
cet_cest_timestamp                        0
DK_load_actual_entsoe_transparency        3
DK_load_forecast_entsoe_transparency      3
DK_solar_capacity                      6602
DK_solar_generation_actual               12
DK_wind_capacity                       6602
DK_wind_generation_actual                 3
DK_wind_offshore_capacity              6602
DK_wind_offshore_generation_actual        3
DK_wind_onshore_capacity               6602
DK_wind_onshore_generation_actual         3
dtype: int64
```

```
nas_by_index = isna_df.sum(axis=1)
print(nas_by_index[nas_by_index != 0])
```

```
utc_timestamp
2014-12-31-23    10
2015-01-01-00     6
2015-01-01-01     1
2015-01-01-02     1
2015-01-01-03     1
                 ..
2020-09-30-19     5
2020-09-30-20     5
2020-09-30-21     5
2020-09-30-22     5
```

```
2020-09-30-23    10
Length: 6606, dtype: int64
```

```
# все пропуски приходятся на несколько дат в начале и
# в конце дата сетак, так что просто уберим их
df = df.loc['2015-01-02-00':'2019-12-29-23']
df.isna().sum()
```

```
cet_cest_timestamp                      0
DK_load_actual_entsoe_transparency      0
DK_load_forecast_entsoe_transparency    0
DK_solar_capacity                       0
DK_solar_generation_actual              0
DK_wind_capacity                        0
DK_wind_generation_actual               0
DK_wind_offshore_capacity               0
DK_wind_offshore_generation_actual      0
DK_wind_onshore_capacity                0
DK_wind_onshore_generation_actual       0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 43752 entries, 2015-01-02-00 to 2019-12-29-23
Data columns (total 11 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   cet_cest_timestamp                    43752 non-null  object
 1   DK_load_actual_entsoe_transparency    43752 non-null  float64
 2   DK_load_forecast_entsoe_transparency  43752 non-null  float64
 3   DK_solar_capacity                     43752 non-null  float64
 4   DK_solar_generation_actual            43752 non-null  float64
 5   DK_wind_capacity                      43752 non-null  float64
 6   DK_wind_generation_actual             43752 non-null  float64
 7   DK_wind_offshore_capacity             43752 non-null  float64
 8   DK_wind_offshore_generation_actual    43752 non-null  float64
 9   DK_wind_onshore_capacity              43752 non-null  float64
 10  DK_wind_onshore_generation_actual     43752 non-null  float64
dtypes: float64(10), object(1)
memory usage: 4.0+ MB
```

Добавим информацию о выходных днях в Дании за указанный срок.

```
danish_holidays = {
    '2015-01-01': 'New Year\'s Day',
    '2015-04-02': 'Maundy Thursday',
    '2015-04-03': 'Good Friday',
    '2015-04-05': 'Easter Sunday',
    '2015-04-06': 'Easter Monday',
    '2015-05-01': 'Labor Day',
    '2015-05-14': 'Ascension Day',
    '2015-05-24': 'Whit Sunday',
    '2015-06-05': 'Constitution Day',
    '2015-12-24': 'Christmas Eve',
```

```
'2015-12-25': 'Christmas Day',
'2015-12-26': 'St. Stephen\'s Day',

'2016-01-01': 'New Year\'s Day',
'2016-03-24': 'Maundy Thursday',
'2016-03-25': 'Good Friday',
'2016-03-27': 'Easter Sunday',
'2016-03-28': 'Easter Monday',
'2016-04-24': 'Great Prayer Day',
'2016-05-01': 'Labor Day',
'2016-05-05': 'Ascension Day',
'2016-05-15': 'Whit Sunday',
'2016-06-05': 'Constitution Day',
'2016-12-24': 'Christmas Eve',
'2016-12-25': 'Christmas Day',
'2016-12-26': 'St. Stephen\'s Day',

'2017-01-01': 'New Year\'s Day',
'2017-04-09': 'Palm Sunday',
'2017-04-10': 'Maundy Thursday',
'2017-04-12': 'Good Friday',
'2017-04-13': 'Easter Sunday',
'2017-05-14': 'Ascension Day',
'2017-05-25': 'Ascension Day',
'2017-06-05': 'Whit Sunday',
'2017-12-24': 'Christmas Eve',
'2017-12-25': 'Christmas Day',
'2017-12-26': 'St. Stephen\'s Day',

'2018-01-01': 'New Year\'s Day',
'2018-03-29': 'Maundy Thursday',
'2018-03-30': 'Good Friday',
'2018-03-31': 'Easter Sunday',
'2018-04-01': 'Easter Monday',
'2018-04-29': 'Great Prayer Day',
'2018-05-01': 'Labor Day',
'2018-05-10': 'Ascension Day',
'2018-05-20': 'Whit Sunday',
'2018-06-05': 'Constitution Day',
'2018-12-24': 'Christmas Eve',
'2018-12-25': 'Christmas Day',
'2018-12-26': 'St. Stephen\'s Day',

'2019-01-01': 'New Year\'s Day',
'2019-04-18': 'Maundy Thursday',
'2019-04-19': 'Good Friday',
'2019-04-21': 'Easter Sunday',
'2019-04-22': 'Easter Monday',
'2019-05-01': 'Labor Day',
'2019-05-30': 'Ascension Day',
'2019-06-05': 'Constitution Day',
'2019-12-24': 'Christmas Eve',
'2019-12-25': 'Christmas Day',
'2019-12-26': 'St. Stephen\'s Day',
```

```
}
holidays = pd.Series(danish_holidays)
holidays.index = pd.to_datetime(holidays.index)

# переименуем столбцы в более удобочитаемый вид
new_cols = {'DK_load_actual_entsoe_transparency': 'total_load',
            'DK_load_forecast_entsoe_transparency': 'load_forecast',
            'DK_solar_capacity': 'solar_capacity',
            'DK_solar_generation_actual': 'solar_generation',
            'DK_wind_capacity': 'wind_capacity',
            'DK_wind_generation_actual': 'wind_generation',
            'DK_wind_offshore_capacity': 'wind_capacity_off',
            'DK_wind_offshore_generation_actual': 'wind_generation_off',
            'DK_wind_onshore_capacity': 'wind_capacity_on',
            'DK_wind_onshore_generation_actual': 'wind_generation_on',
            }

# избавимся от ненужных столбцов
df.drop('cet_cest_timestamp', axis=1, inplace=True)
df.rename(columns=new_cols, inplace=True)
df = df.rename_axis('date')

df.sample(5)
```

date	total_load	load_forecast	solar_capacity	solar_generation
2018-02-11-15	4362.12	4409.0	547.0	6.81
2019-06-29-06	3327.96	3349.4	547.0	238.95
2017-05-15-06	4236.69	4243.3	547.0	133.24
2017-12-09-06	3647.78	3735.5	547.0	0.01
2016-01-13-20	4371.59	4371.4	537.0	0.00

```
# сохраним дата сет
df.to_csv("time_series_60min_dk_mod.csv")
```

Предварительный анализ

Для последующего анализа и предсказаний возьмем значение общего электропотребления

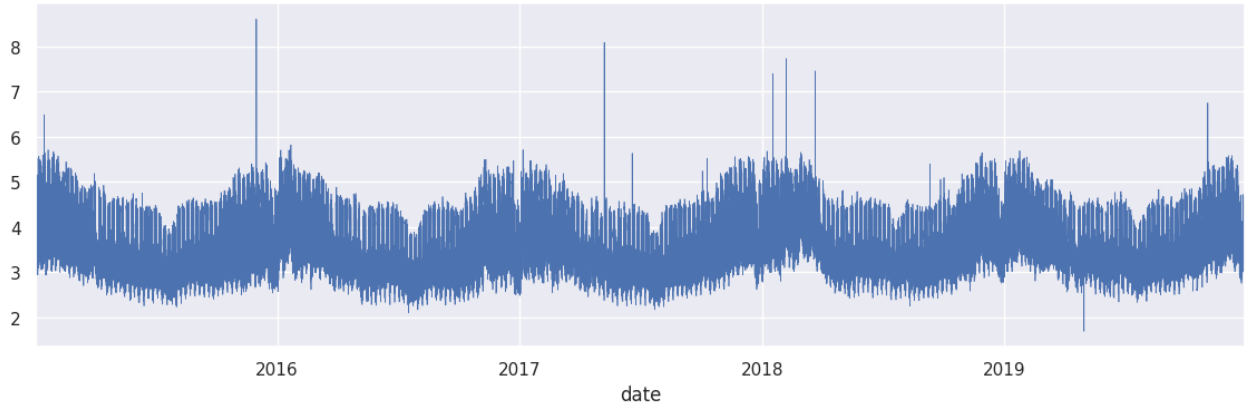
```
data = pd.read_csv("time_series_60min_dk_mod.csv", index_col="date", parse_dat
df = data['total_load'].to_frame()

# добавим значения дня недели
df = df / 1000 # convert MW to GW
```

Визуализация и первые выводы

```
df.describe()
```

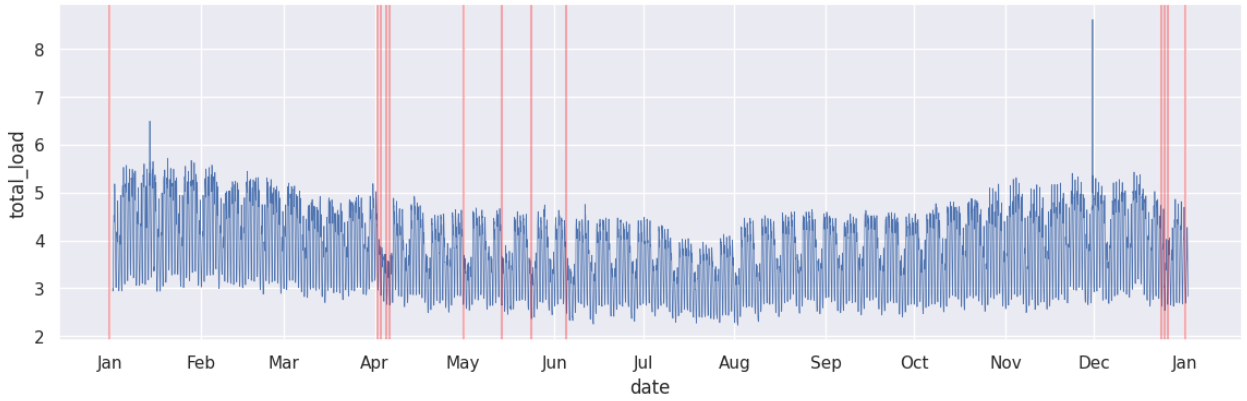
	total_load
count	43752.000000
mean	3.758515
std	0.745868
min	1.692950
25%	3.148113
50%	3.725790
75%	4.350977
max	8.607380

```
df['total_load'].plot(linewidth=0.5);
```

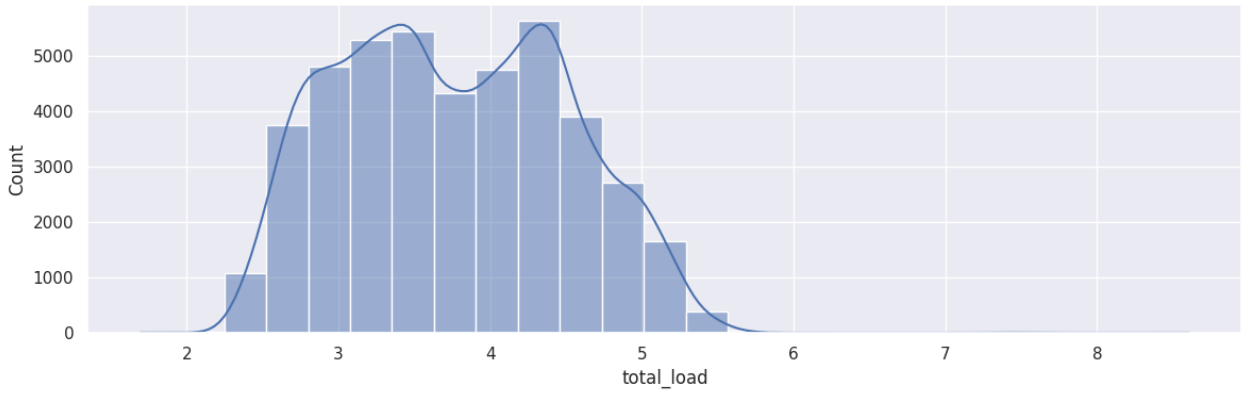



```
# посмотрим как праздничные дни могут влиять на потребление
# электроэнергии в течение календарного года

start_date = '2015-01-01'
end_date = '2016-01-01'
holiday_dates = holidays.loc[start_date: end_date].index

ax = sns.lineplot(df.loc[start date: end date]['total load'], linewidth=0.5);
```

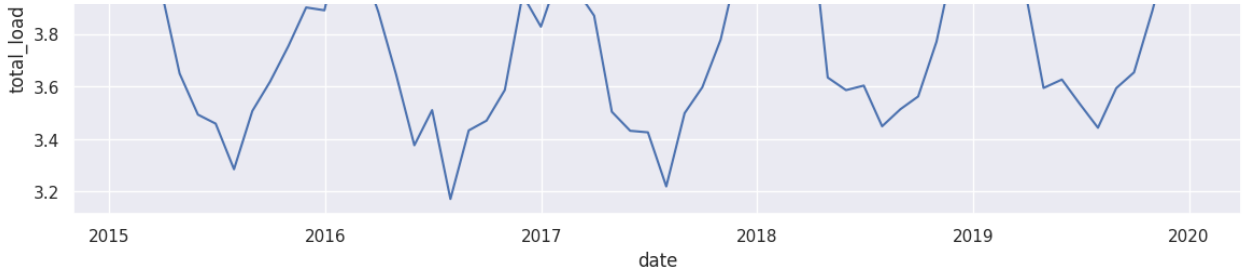
```
for day in holiday_dates:
    ax.axvline(x=day, alpha=0.3, color='red');


months = MonthLocator()
monthsFmt = DateFormatter('%b')

ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(monthsFmt)
```




```
sns.histplot(df['total_load'], bins=25, kde=True);
```




```
df_monthly = df['total_load'].resample('M').mean()
sns.lineplot(data=df_monthly);
```

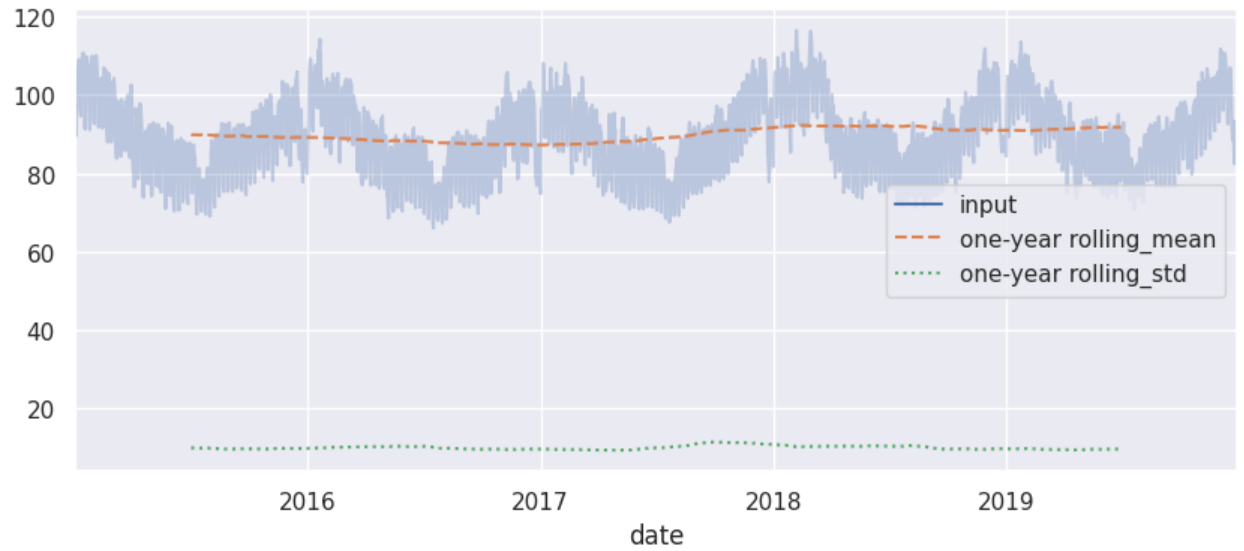


Выводы

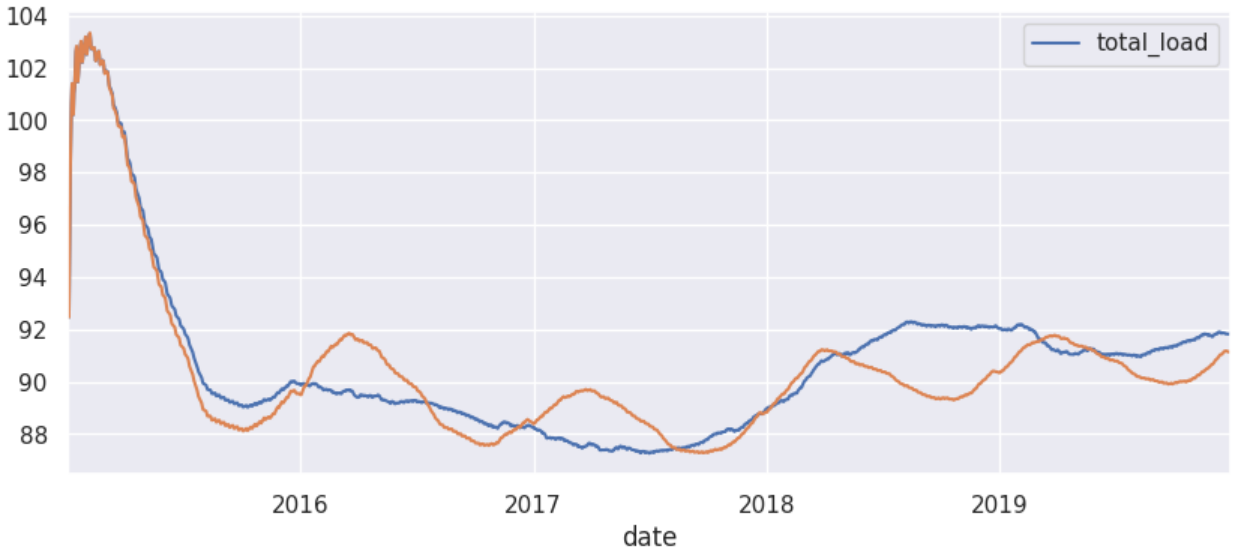
- Из графиков видно, что апрельские и январские праздники вполне объясняют визуально, на первый взгляд, аномальное долгие периоды снижения энергопотребления по сравнению с обычными неделями.
- Так же мы можем отметить явный тренд на снижение при приближении к летнему периоду отпусков (начало августа).
- Сезонные составляющие можно обозначить как недельные и сезонные (зима - лето);

Анализ сезонности и тренда

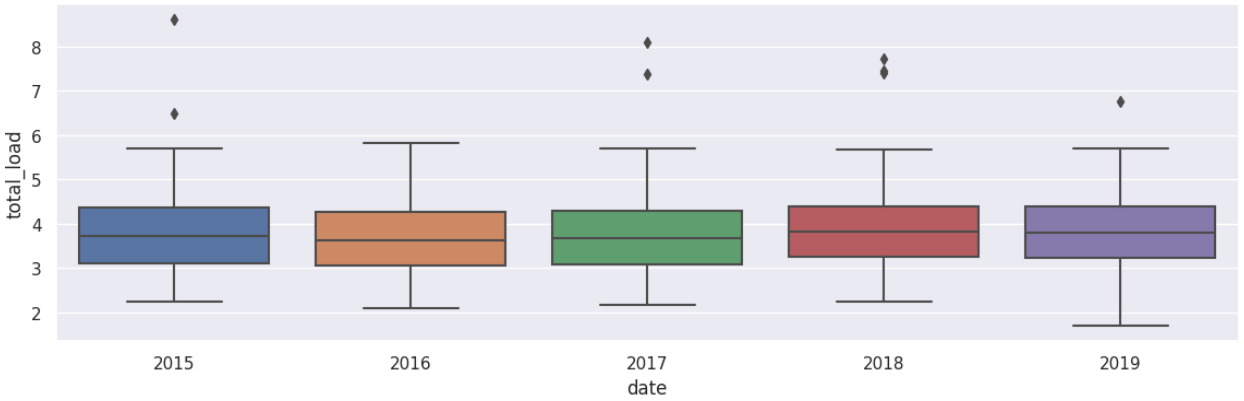
```
rolling = df['total_load'].rolling(365, center=True)

data = pd.DataFrame({'input': df['total_load'],
                     'one-year rolling_mean': rolling.mean(),
                     'one-year rolling_std': rolling.std()})
ax = data.plot(style=['-', '--', ':'])
ax.lines[0].set_alpha(0.3)
```



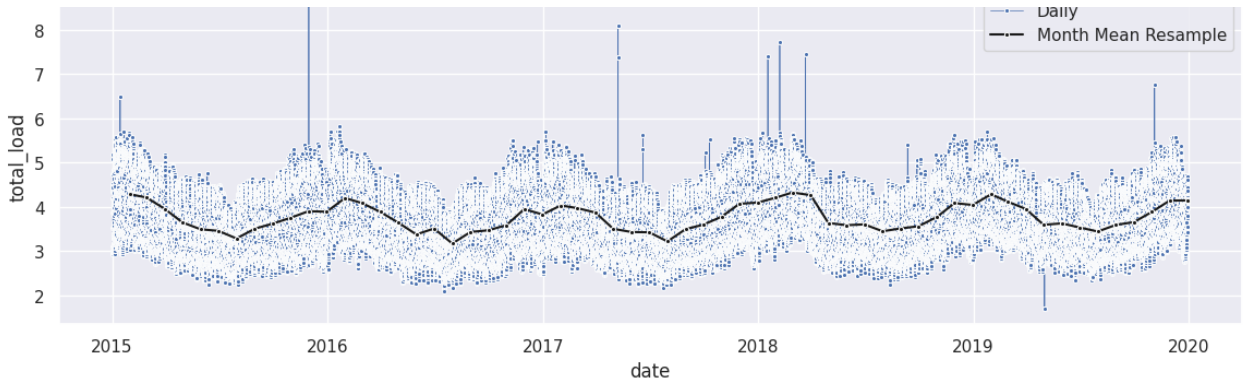
```
df[['total_load']].rolling('365d').mean().plot( linewidth=1.5);
df['total_load'].ewm(halflife=365, min_periods=0, adjust=True).mean().plot();
```



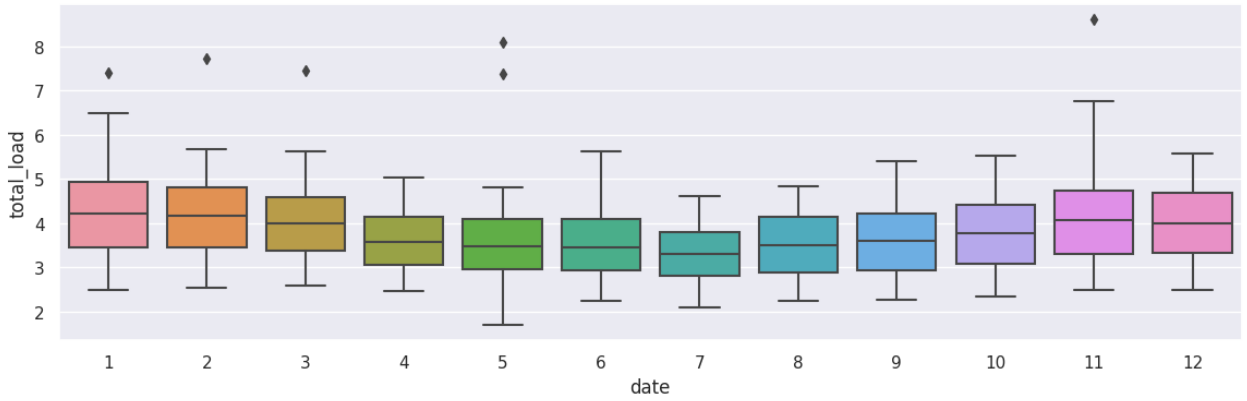

```
sns.boxplot(data=df, x=df.index.year, y='total_load');
```




```
sns.lineplot(data=df['total_load'], marker='.', linestyle='-', linewidth=0.5,
sns.lineplot(data=df_monthly, marker='o', markersize=3, linestyle='-', label='
```

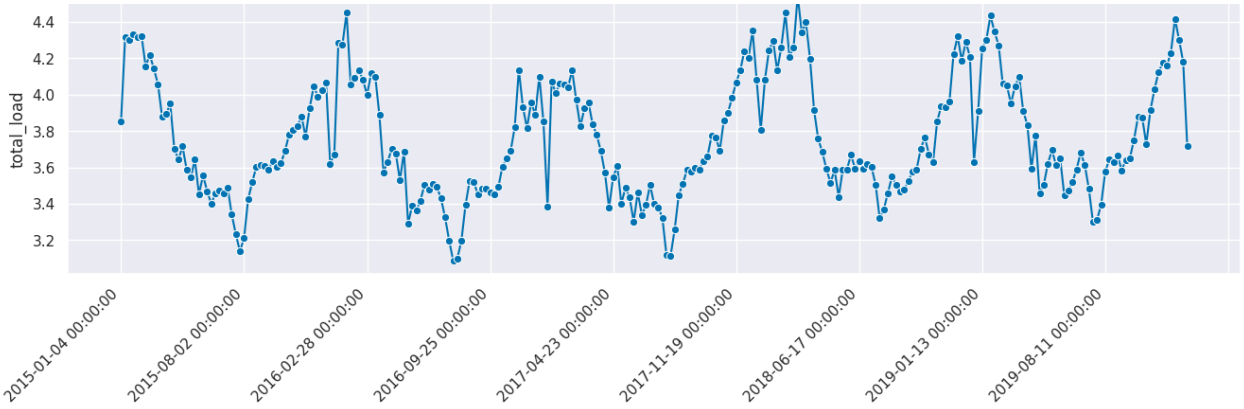



```
sns.boxplot(data=df, x=df.index.month, y='total_load');
```



Анализ стационарности и декомпозиция

```
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose

df_weekly = df['total_load'].resample('W').mean()


plot_series(df_weekly);
plt.xticks(rotation=45, ha='right');
```



Перед декомпозицией нашего временного ряда нам необходимо убедиться, что ряд стационарен

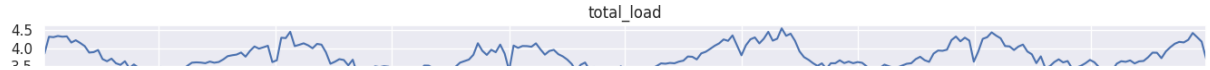
- проверим ряд с помощью adf теста (расширенный тест Дики — Фуллера) с информационным критерием Акаике (стоит по умолчанию);

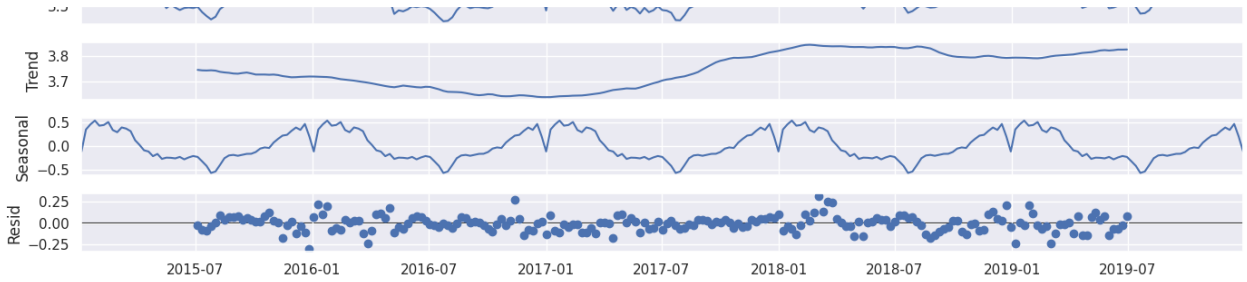
```
result = adfuller(df_weekly)
p_value = result[1]
print(f"P-value from ADF test: {p_value:.8f}")

# проверим с доверительным интервалов .95
if p_value <= 0.05:
    print("ВР стационарен")
else:
    print("ВР нестационарен")
```

```
P-value from ADF test: 0.00000026
ВР стационарен
```

Как видно из результатов анализа, наш временной ряд стационарен и мы можем спокойно переходить его декомпозиции.
Поскольку на визуального анализа мы наблюдали стабильные сезонность и дисперсию, мы можем выбрать аддитивную модель для декомпозиции.

```
# разложим ряд с усреднением по неделе
result = seasonal_decompose(df_weekly, model='additive')
result.plot();
```





По графику декомпозиции нашего временного ряда мы можем сделать следующие выводы:

1. ***Выявление тренда:***

- на графике возможно все же присутствует нелинейный тренд;
- при этом можно отметить некоторые участки, которые можно описать линейно;

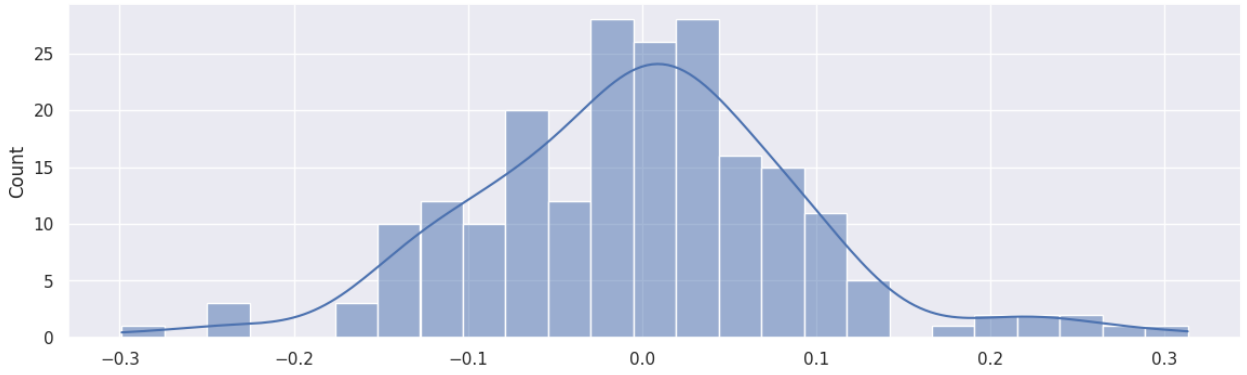
2. ***Сезонность и цикличность:***

- на графики видим четкую сезонность потребления, например, каждый август и конец декабря явный спад;
- цикличности в нашем временном ряду нет, даже несмотря на необычное поведение тренда;

3. ***Шум:***

- после того, как мы декомпозировали наш временной ряд, убрали тренд и сезонность, у нас остался белый шум.

```
# проверка шума на нормальность
sns.histplot(result.resid.values, bins=25, kde=True);
```



Предсказания

```
%%shell
pip install sktime[all_extras] --quiet
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 20.7/20.7 MB 30.6 MB/s eta 0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 122.4/122.4 kB 8.2 MB/s eta (
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 981.7/981.7 kB 37.5 MB/s eta
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.4/10.4 MB 44.0 MB/s eta 0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 645.5/645.5 kB 41.2 MB/s eta
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.5/1.5 MB 41.0 MB/s eta 0:0(
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.7/7.7 MB 58.6 MB/s eta 0:0(
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.0/49.0 kB 5.4 MB/s eta 0:(
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Installing backend dependencies ... done
  Preparing metadata (pyproject.toml) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 244.3/244.3 kB 24.8 MB/s eta
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100.3/100.3 kB 11.2 MB/s eta
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 196.1/196.1 kB 22.3 MB/s eta
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 178.0/178.0 kB 22.1 MB/s eta
  Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 160.4/160.4 kB 16.2 MB/s eta
  Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 160.5/160.5 kB 18.2 MB/s eta
  Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 169.1/169.1 kB 17.3 MB/s eta
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 358.2/358.2 kB 28.0 MB/s eta
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 59.1 MB/s eta 0:0(
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 110.9/110.9 kB 12.8 MB/s eta
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.0/44.0 kB 3.5 MB/s eta 0:(
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 95.3/95.3 kB 11.5 MB/s eta 0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 279.8/279.8 kB 27.4 MB/s eta
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.4/60.4 kB 7.8 MB/s eta 0:(
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 169.2/169.2 kB 21.1 MB/s eta
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 154.7/154.7 kB 17.8 MB/s eta
  Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 144.2/144.2 kB 19.3 MB/s eta
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 135.3/135.3 kB 15.3 MB/s eta
  Building wheel for filterpy (setup.py) ... done
  Building wheel for pycatch22 (pyproject.toml) ... done
  Building wheel for pyod (setup.py) ... done
  Building wheel for keras-self-attention (setup.py) ... done
  Building wheel for fugue-sql-antlr (setup.py) ... done
```

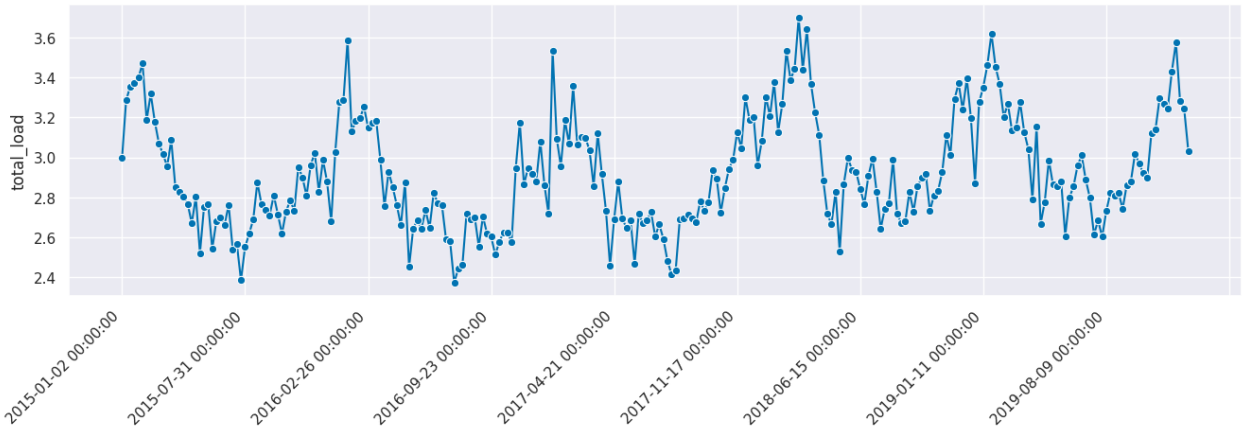
Наивные предсказания. Baseline

```
from sktime.forecasting.naive import NaiveForecaster
from sktime.performance_metrics.forecasting import (
    MeanAbsolutePercentageError,
    MeanSquaredError,
    MeanAbsoluteError
)

from sktime.utils.plotting import plot_series
from sktime.forecasting.model_selection import temporal_train_test_split
from sktime.forecasting.base import ForecastingHorizon
from sklearn.model_selection import train_test_split

from prettytable import PrettyTable

#metrics

smape = MeanAbsolutePercentageError(symmetric = True)
rmse  = MeanSquaredError(square_root=True)
mae = MeanAbsoluteError()

# создадим таблицу для сравнения методов предсказания
predictions = pd.DataFrame(columns=['method', 'smape', 'rmse', 'mae'])

def get_metrics(y_true, y_pred, title=None):
    table = PrettyTable()
    mae_value = mae(y_true, y_pred).round(4)
    rmse_value = rmse(y_true, y_pred).round(4)
    smape_value = smape(y_true, y_pred).round(4)
    global predictions
    predictions = pd.concat([
    predictions,
    pd.Series({
        'method': title,
        'smape': smape_value,
        'rmse': rmse_value,
        'mae': mae_value}
              ).to_frame().T],ignore_index=True)

    table.field_names = ["Metric", "Value"]
    table.title = title
    table.add_row(["MAE", mae_value])
    table.add_row(["RMSE", rmse_value])
    table.add_row(["SMAPE", smape_value])

    print(table)

y = df['total_load'].asfreq('7d')
y.head()
```

```
date
2015-01-02    2.99622
2015-01-09    3.28587
2015-01-16    3.35409
```

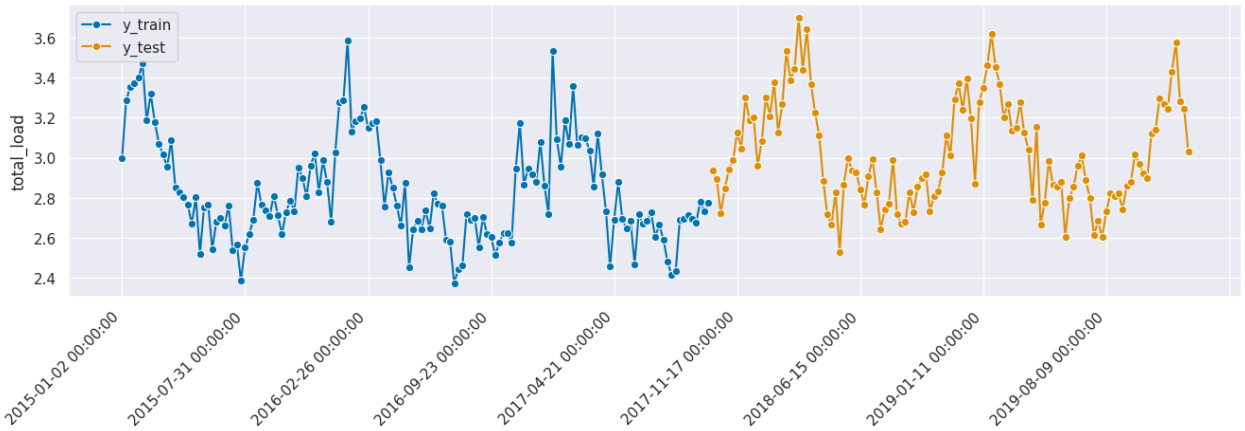
```
2015-01-23    3.37059
2015-01-30    3.39943
Freq: 7D, Name: total_load, dtype: float64
```

```
plot_series(y);
plt.xticks(rotation=45, ha='right');
```

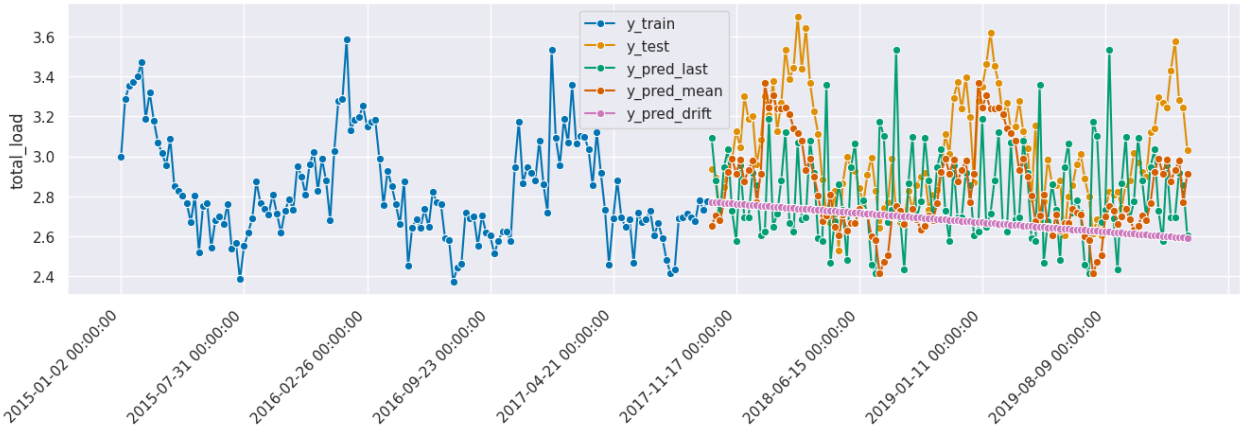



```
test_size = int(0.45 * y.size)

y_train, y_test = temporal_train_test_split(y, test_size=test_size)
fh = ForecastingHorizon(y_test.index, is_relative=False)
SEASON = 52

print(f'Check splitted data size: Train: {y_train.shape[0]}, Test: {y_test.sha

plot_series(y_train, y_test, labels=["y_train", "y_test"]);
plt.xticks(rotation=45, ha='right');
```

```
Check splitted data size: Train: 144, Test: 117
```



```
# last
forecaster = NaiveForecaster(strategy="last", sp=SEASON)
forecaster.fit(y_train)
y_pred_last = forecaster.predict(fh)

# mean
forecaster = NaiveForecaster(strategy="mean", sp=SEASON)
forecaster.fit(y_train)
y_pred_mean = forecaster.predict(fh)

# drift
forecaster = NaiveForecaster(strategy="drift")
forecaster.fit(y_train)
y_pred_drift = forecaster.predict(fh)

plot_series(y_train,
            y_test,
            y_pred_last,
            y_pred_mean,
            y_pred_drift,
            labels=["y_train", "y_test",
                    "y_pred_last", "y_pred_mean",
                    "y_pred_drift"])

plt.xticks(rotation=45, ha='right');
```



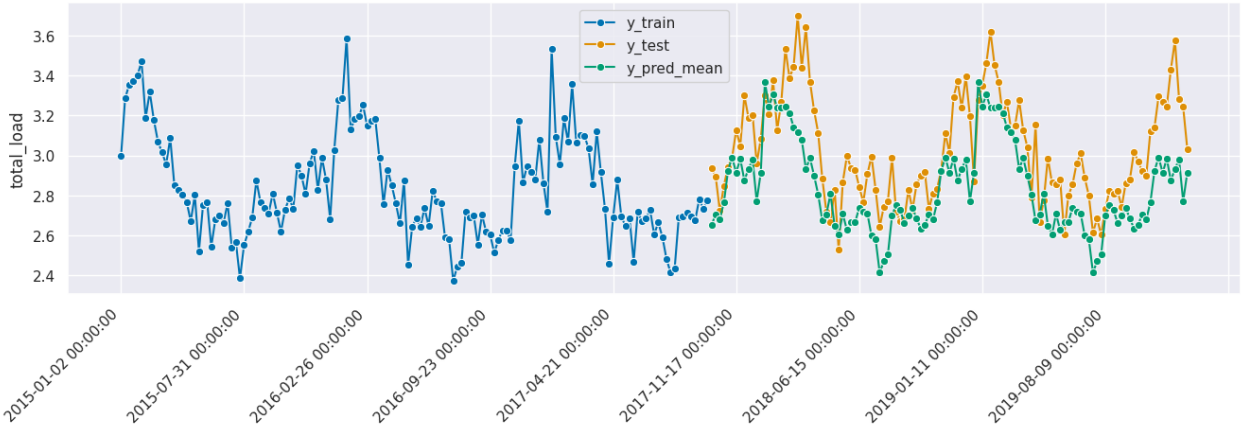

```
get_metrics(y_test, y_pred_last, title="Naive_Last")
get_metrics(y_test, y_pred_mean, title="Naive_Mean_Seasonal")
get_metrics(y_test, y_pred_drift, title="Naive_Drift")
```

```
    +-----------------+
    |    Naive_Last   |
```

```
+--------+--------+
| Metric | Value  |
+--------+--------+
|  MAE   | 0.3443 |
|  RMSE  | 0.4188 |
| SMAPE  | 0.1168 |
+--------+--------+
+----------------------+
| Naive_Mean_Seasonal |
+----------+----------+
|  Metric  |  Value   |
+----------+----------+
|   MAE    |  0.2037  |
|   RMSE   |  0.2508  |
|  SMAPE   |  0.0688  |
+----------+----------+
+-----------------+
|   Naive_Drift   |
+--------+--------+
| Metric | Value  |
+--------+--------+
|  MAE   | 0.3513 |
|  RMSE  | 0.4317 |
| SMAPE  | 0.1197 |
+--------+--------+
```

По данным метрик мы видим, что наиболее лучшие показатели предсказаний получались у Naive Mean Seasonal. Предсказания в целом стараются следовать паттерну реальных значений и мы можем использовать данную модель, как baseline.

```
plot_series(y_train,
            y_test,
            y_pred_mean,
            labels=["y_train", "y_test", "y_pred_mean"])

plt.xticks(rotation=45, ha='right');
```

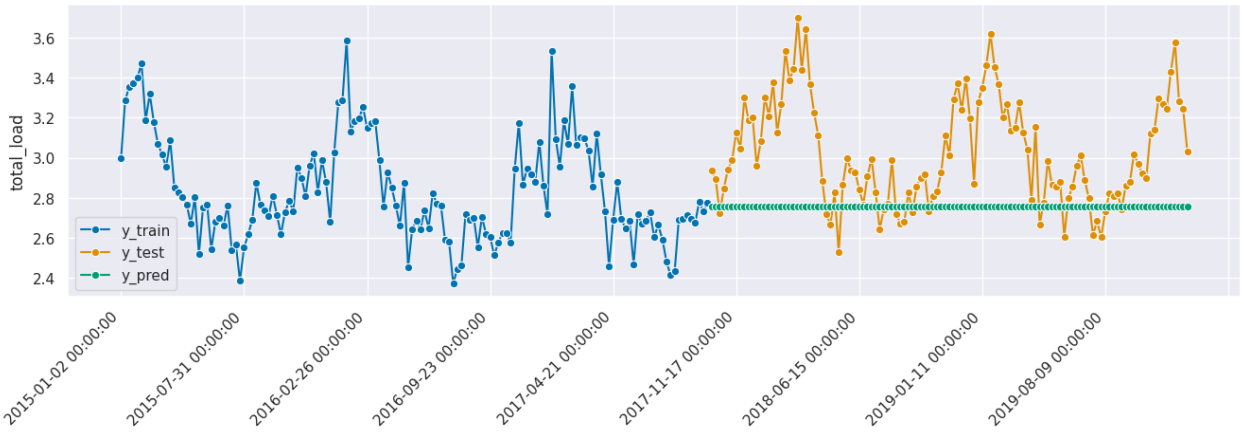


Экспоненциональное сглаживание

```
from sktime.forecasting.exp_smoothing import ExponentialSmoothing


ses = ExponentialSmoothing(sp=SEASON)
holt = ExponentialSmoothing(trend="add", damped_trend=False,  sp=SEASON)
damped_holt = ExponentialSmoothing(trend="add", damped_trend=True,   sp=SEASON
holt_winter = ExponentialSmoothing(trend="add", seasonal="additive", sp=SEASON
holt_winter_add_boxcox  = ExponentialSmoothing(trend="add", seasonal="additive
holt_winter_mul_boxcox  = ExponentialSmoothing(trend="mul", seasonal="additive
holt_winter_sadd_boxcox = ExponentialSmoothing(trend="add", seasonal="mul", us
holt_winter_smul_boxcox = ExponentialSmoothing(trend="mul", seasonal="mul", us
```

Simple Exponential Smoothing

```
forecaster = ses

forecaster.fit(y_train)
y_pred = forecaster.predict(fh)

plot_series(y_train,
            y_test,
            y_pred,
            labels=["y_train",
                    "y_test",
                    "y_pred"])

plt.xticks(rotation=45, ha='right');
```




```
get_metrics(y_test, y_pred, title="Simple Exp Smoothing")
```

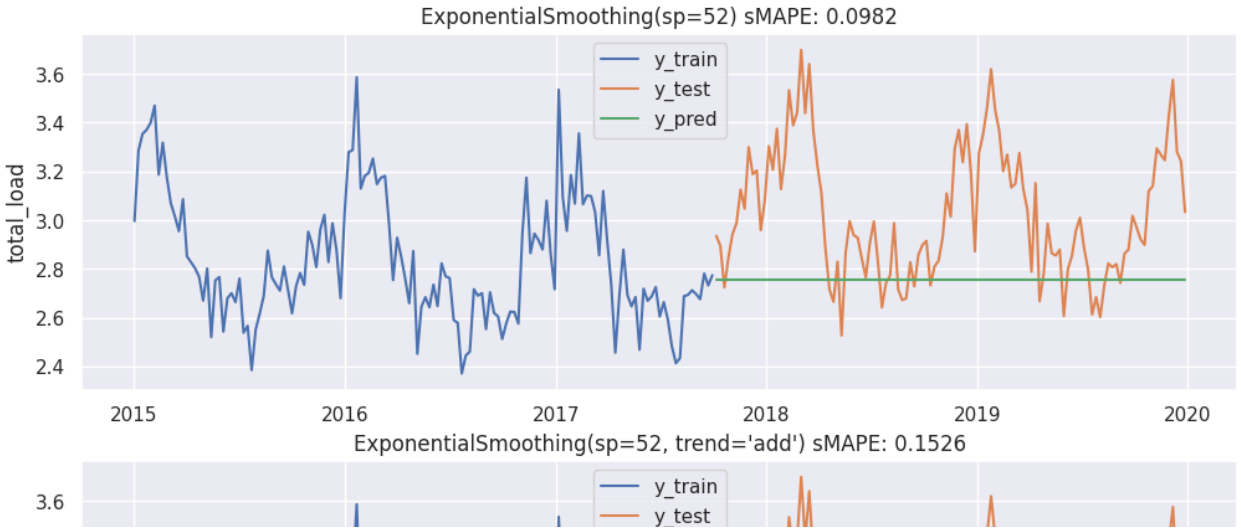
```
+----------------------+
| Simple Exp Smoothing |
+----------+-----------+
|  Metric  |   Value   |
+----------+-----------+
|   MAE    |   0.2932  |
|   RMSE   |   0.3752  |
|  SMAPE   |   0.0982  |
+----------+-----------+
```

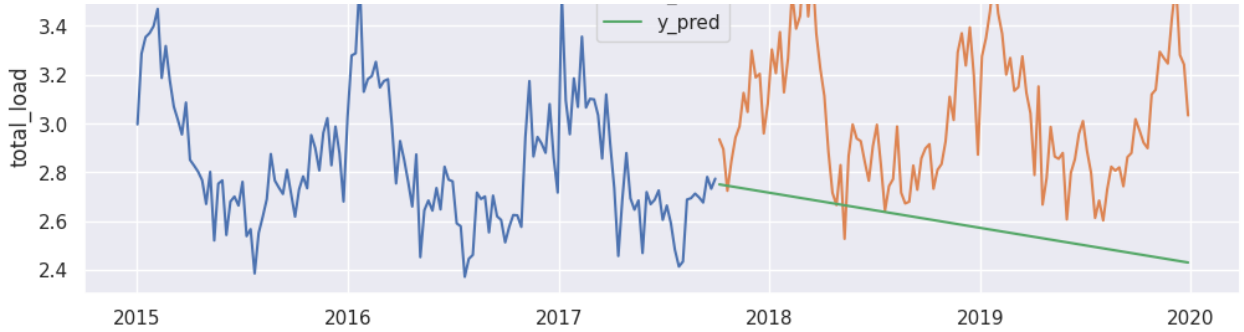
Сравнение методов экспоненционального сглаживания

Проведем сравнение нескольких методов экспоненционального сглаживания

```
# добавим все наши методы в лист
exp_smoothing = [
    ses,
    holt,
    damped_holt,
    holt_winter,
    holt_winter_add_boxcox,
    holt_winter_mul_boxcox,
    holt_winter_sadd_boxcox,
    holt_winter_smul_boxcox,
]
```

```
fig, ax = plt.subplots(len(exp_smoothing), 1, figsize=(12, 35))

for i, col in enumerate(exp_smoothing):
    forecaster = col
    forecaster.fit(y_train)
    y_pred = forecaster.predict(fh)
    sns.lineplot(y_train, ax=ax[i], label='y_train')
    sns.lineplot(y_test, ax=ax[i], label='y_test')
    sns.lineplot(y_pred, ax=ax[i], label='y_pred')
    smape_value = smape(y_test, y_pred).round(4) # метрика
    ax[i].set_title(str(col) + f' sMAPE: {smape_value}')
```

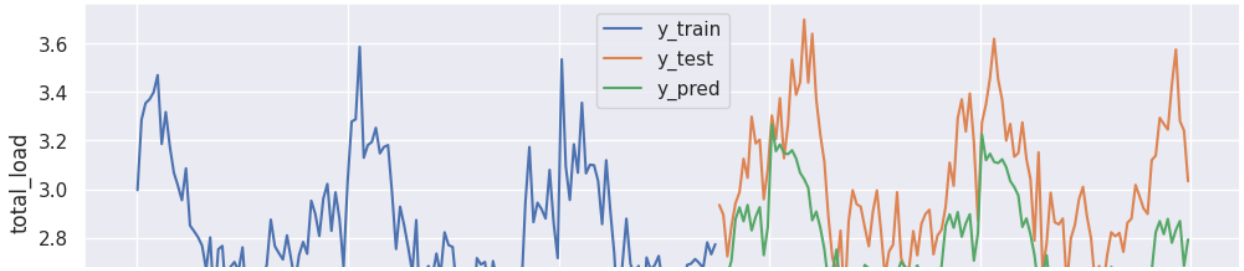




ExponentialSmoothing(damped_trend=True, sp=52, trend='add') sMAPE: 0.0988

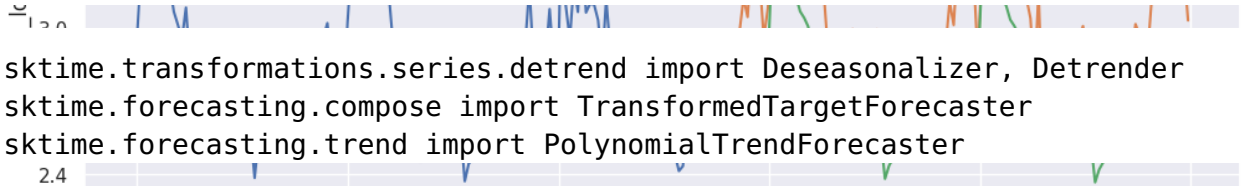
ExponentialSmoothing(seasonal='additive', sp=52, trend='add') sMAPE: 0.0898

ExponentialSmoothing(seasonal='additive', sp=52, trend='add', use_boxcox=True) sMAPE: 0.09



В итоге лучшие результаты среди методов экспоненционального сглаживания получились у методов: holt_winter_sadd_boxcox и holt_winter (sMAPE: 0,089)
Но это все равно меньше наивного предсказания baseline (sMAPE: 0.06)

Экспоненциональное сглаживание с декомпозицией

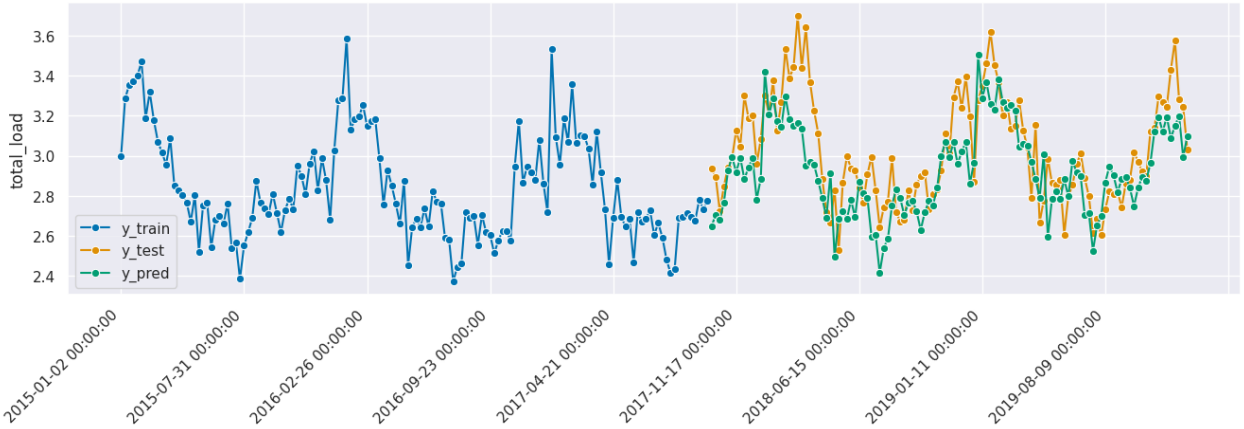


```
from sktime.transformations.series.detrend import Deseasonalizer, Detrender
from sktime.forecasting.compose import TransformedTargetForecaster
from sktime.forecasting.trend import PolynomialTrendForecaster
```

```
forecaster = TransformedTargetForecaster(
```

```
                steps=[
                ("deseasonalize", Deseasonalizer(model="additive", sp=SEASON))
                ("detrend", Detrender(forecaster=PolynomialTrendForecaster(deg
                ("forecaster", ses)
                ])

forecaster.fit(y_train)

# Предсказание
y_pred = forecaster.predict(fh)

# Результаты
plot_series(y_train, y_test, y_pred, labels=["y_train", "y_test", "y_pred"])
plt.xticks(rotation=45, ha='right');

print(f'sMAPE = {smape(y_pred.values, y_test.values):.3f}')
```

```
sMAPE = 0.053
```



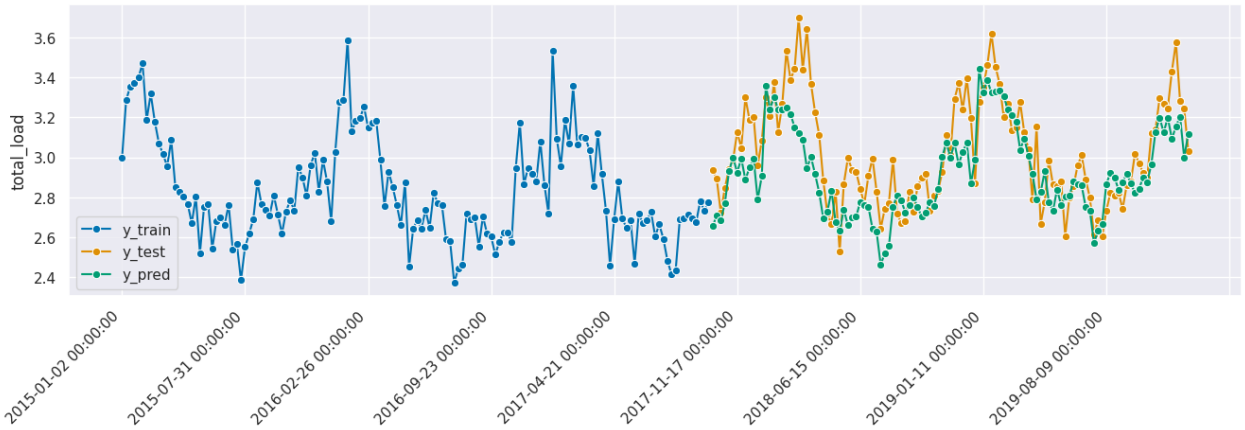

```
get_metrics(y_test, y_pred, title='SimpleExponentialSmoothingDecomposed')
```

```
+---------------------------------------+
| SimpleExponentialSmoothingDecomposed |
+------------------+--------------------+
|      Metric      |       Value        |
+------------------+--------------------+
|       MAE        |       0.1585       |
|       RMSE       |       0.2025       |
|      SMAPE       |       0.0529       |
+------------------+--------------------+
```

```
forecaster = TransformedTargetForecaster(
                steps=[
                ("deseasonalize", Deseasonalizer(model="additive", sp=SEASON))
                ("detrend", Detrender(forecaster=PolynomialTrendForecaster(deg
                ("forecaster", holt_winter)
                ])
```

```
forecaster.fit(y_train)

# Предсказание
y_pred = forecaster.predict(fh)

# Результаты
plot_series(y_train, y_test, y_pred, labels=["y_train", "y_test", "y_pred"])
plt.xticks(rotation=45, ha='right');

print(f'sMAPE = {smape(y_pred.values, y_test.values):.3f}')
```

```
sMAPE = 0.050
```




```
get_metrics(y_test, y_pred, title='ExpSmoothingHoltWinter')
```

```
+-------------------------+
| ExpSmoothingHoltWinter |
+-----------+-------------+
|   Metric  |   Value     |
+-----------+-------------+
|    MAE    |   0.1499    |
|    RMSE   |   0.1943    |
|   SMAPE   |   0.0499    |
+-----------+-------------+
```

SARIMA

```
from sktime.forecasting.sarimax import SARIMAX
from sktime.forecasting.arima import ARIMA
from sktime.forecasting.arima import AutoARIMA
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
```

Поскольку мы имеем дело с временным рядом с четко выраженной сезонной составляющей, а также присутствующим трендом. В качестве регрессионной модели возьмем SARIMA. С учетом проведенного предварительного анализа, нам необходимо установить следующие значения гиперпараметров для нашей модели:

- (p, d, q) - несезонные параметры
- (P, D, Q) - сезонные параметры
- s - периодичность ряда (для нашего ряда с недельными значениями - 52)

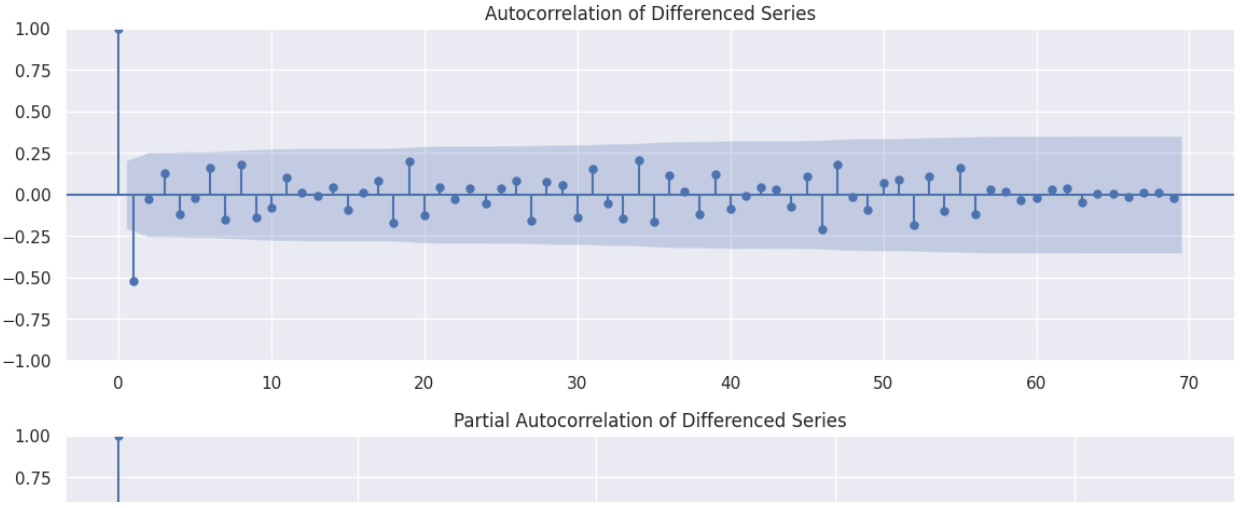
Baseline. Ручной подбор.

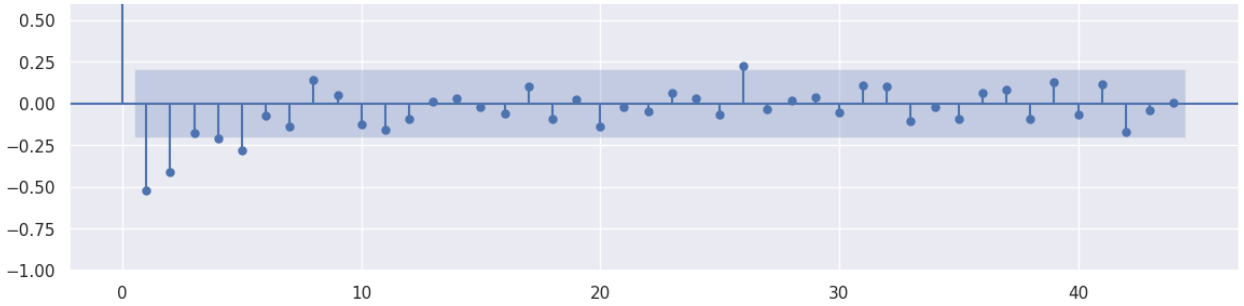
```
y_sdif = y_train[:].diff(1).diff(SEASON).dropna()
```

```
result = adfuller(y_sdif)
p_value = result[1]
print(f"P-value from ADF test: {p_value:.8f}")

# проверим с доверительным интервалов .95
if p_value <= 0.05:
    print("ВР стационарен")
else:
    print("ВР нестационарен")
```

```
P-value from ADF test: 0.00000000
ВР стационарен
```

```
# Usual Differencing
plot_acf(y_sdif,
         title='Autocorrelation of Differenced Series',
         lags=np.arange(70));

# Usual Differencing
plot_pacf(y_sdif,
          title='Partial Autocorrelation of Differenced Series',
          method='ywm',
          lags=np.arange(45));
```





Опираясь на графики автокорреляционной и частичной автокорреляционной функции, Мы можем установить параметры для нашей модели:

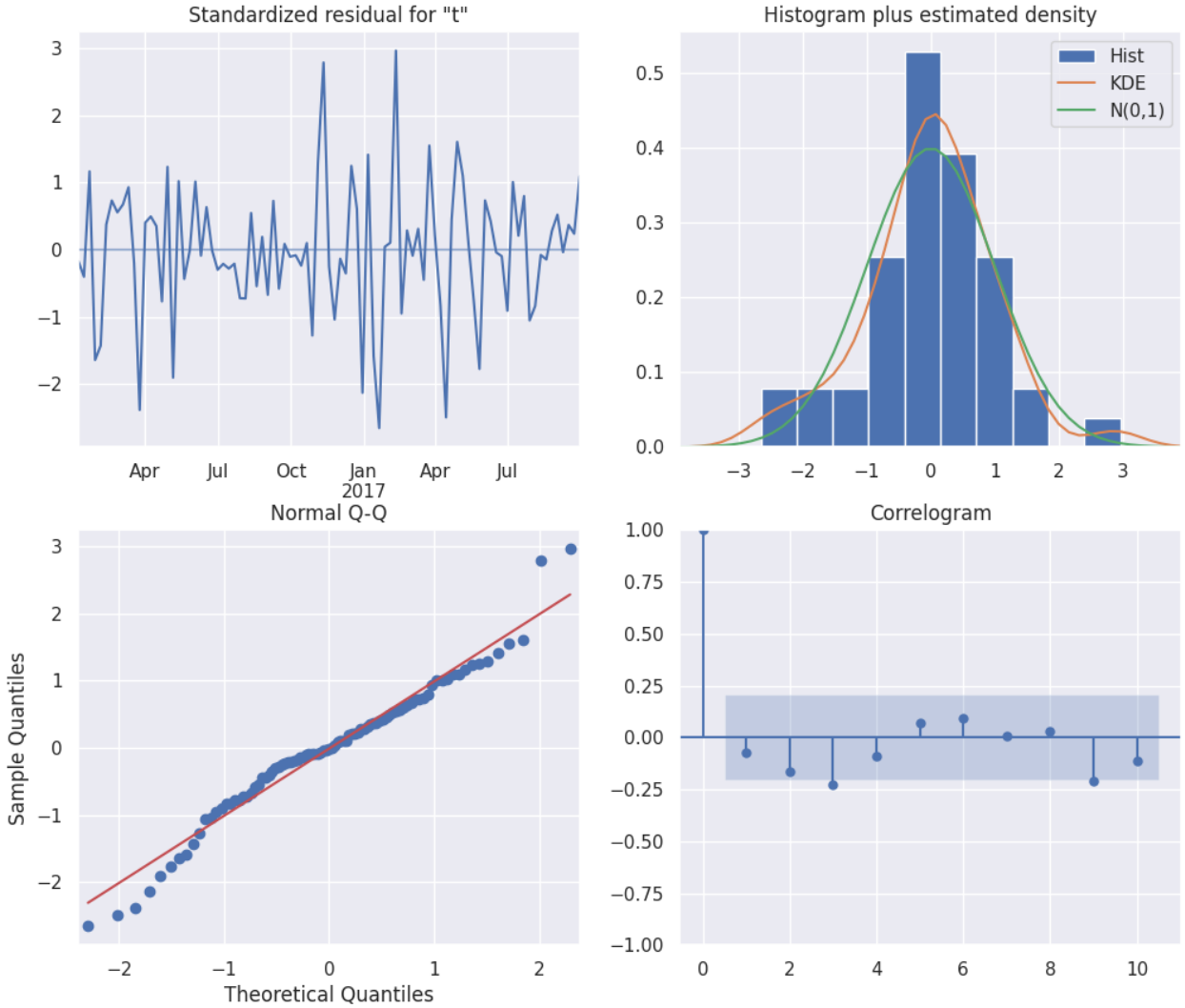
- 2 порядка AR (только 1 и 2 лаг ниже 0 на графике PACF);
- 1 порядок MA (мы выбрали порядок дифференцирования 1);
- 1 порядок SAR (1 лаг выше 0 на графике PACF);
- 0 порядок SMA (на ACF нет значимых лагов ниже 0)

```
forecaster = SARIMAX(order=(2, 1, 0),  seasonal_order=(1, 1, 0, 52))
forecaster.fit(y_train)
print(forecaster.summary())
```

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                           total_load   No. Observations:
Model:             SARIMAX(2, 1, 0)x(1, 1, 0, 52)   Log Likelihood
Date:                            Sat, 18 Nov 2023   AIC
Time:                                    18:41:07   BIC
Sample:                                01-02-2015   HQIC
                                     - 09-29-2017
Covariance Type:                              opg
==========================================================================================
                 coef    std err          z      P>|z|      [0.025      0.9
------------------------------------------------------------------------------------------
intercept      0.0072      0.022      0.327      0.743      -0.036       0
ar.L1         -0.7473      0.100     -7.474      0.000      -0.943      -0
ar.L2         -0.4288      0.104     -4.141      0.000      -0.632      -0
ar.S.L52      -0.5081      0.130     -3.921      0.000      -0.762      -0
sigma2         0.0255      0.004      6.439      0.000       0.018       0
==========================================================================================
Ljung-Box (L1) (Q):                   0.46   Jarque-Bera (JB):
Prob(Q):                              0.50   Prob(JB):
```

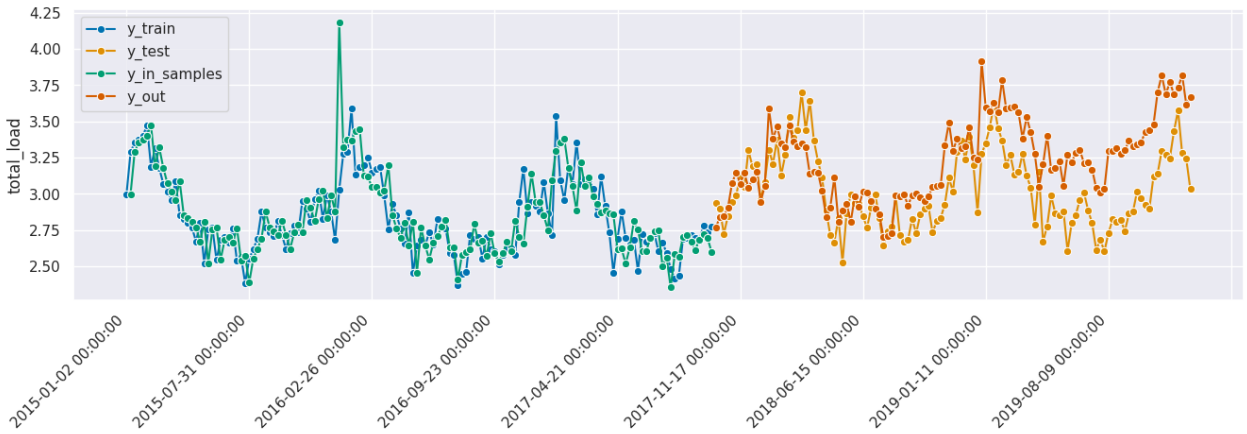
```
Heteroskedasticity (H):                  1.01   Skew:
Prob(H) (two-sided):                     0.98   Kurtosis:
================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (com
```

```
forecaster._fitted_forecaster.plot_diagnostics(figsize=(12, 10));
```



```
fhin = ForecastingHorizon(y_train.index[1:], is_relative=False)
y_in_samples = forecaster.predict(fhin)

fhout = ForecastingHorizon(y_test.index, is_relative=False)
y_out = forecaster.predict(fhout)

# Результаты
plot_series(y_train, y_test, y_in_samples, y_out, labels=["y_train",
                                                          "y_test",
                                                          "y_in_samples",
                                                          "y_out"])
plt.xticks(rotation=45, ha='right');

get_metrics(y_test, y_out, title='SARIMAX(2,1,0)(1,1,0,52)')
```

```
+----------------------------+
| SARIMAX(2,1,0)(1,1,0,52) |
+------------+-------------+
|   Metric   |    Value    |
+------------+-------------+
|    MAE     |    0.258    |
|    RMSE    |    0.3156   |
|   SMAPE    |    0.0821   |
+------------+-------------+
```



AutoARIMA

```
model = AutoARIMA(start_p=1,  # начальный порядок AR
                  start_q=0,  # начальный порядок MA
                  max_p=3,    # конечный порядок AR
                  max_q=0,    # конечный порядок MA
                  d=1,        # Порядок производной
                  max_d=2,
                  seasonal=True,  # Использовать SARIMA
                  start_P=0,  # начальный порядок SAR
```

```
                    start_Q=0,  # начальный порядок SMA
                    max_P=2,
                    max_Q=2,
                    D=1,        # Порядок сезонной производной
                    max_D=2,
                    sp=52,      # Период сезонности
                    max_order = 7, # Максимальный порядок p+q+P+Q
                    trace  = True,   # отчет он-лайн
                    stepwise = True, # метод ускоренного выбора параметров.
                    n_jobs = 1,      # для stepwise парралелизм не доступен.
                    error_action='ignore',
                    suppress_warnings=True)

model.fit(y_train)

model.summary()
```

```
     Performing stepwise search to minimize aic
      ARIMA(1,1,0)(0,1,0)[52] intercept   : AIC=-25.405, Time=1.42 sec
      ARIMA(0,1,0)(0,1,0)[52] intercept   : AIC=1.434, Time=0.95 sec
      ARIMA(1,1,0)(1,1,0)[52] intercept   : AIC=-32.884, Time=16.74 sec
      ARIMA(0,1,0)(0,1,1)[52] intercept   : AIC=inf, Time=14.10 sec
      ARIMA(0,1,0)(0,1,0)[52]             : AIC=-0.556, Time=0.80 sec
      ARIMA(1,1,0)(2,1,0)[52] intercept   : AIC=-30.884, Time=34.74 sec
      ARIMA(1,1,0)(1,1,1)[52] intercept   : AIC=-30.884, Time=9.77 sec
      ARIMA(1,1,0)(0,1,1)[52] intercept   : AIC=inf, Time=17.41 sec
      ARIMA(1,1,0)(2,1,1)[52] intercept   : AIC=-28.884, Time=34.46 sec
      ARIMA(0,1,0)(1,1,0)[52] intercept   : AIC=-5.838, Time=8.35 sec
      ARIMA(2,1,0)(1,1,0)[52] intercept   : AIC=-49.221, Time=11.68 sec
      ARIMA(2,1,0)(0,1,0)[52] intercept   : AIC=-40.210, Time=1.66 sec
      ARIMA(2,1,0)(2,1,0)[52] intercept   : AIC=-47.221, Time=51.25 sec
      ARIMA(2,1,0)(1,1,1)[52] intercept   : AIC=-47.220, Time=25.21 sec
      ARIMA(2,1,0)(0,1,1)[52] intercept   : AIC=inf, Time=24.23 sec
      ARIMA(2,1,0)(2,1,1)[52] intercept   : AIC=-45.220, Time=52.82 sec
      ARIMA(3,1,0)(1,1,0)[52] intercept   : AIC=-48.753, Time=19.98 sec
      ARIMA(2,1,0)(1,1,0)[52]             : AIC=-51.109, Time=8.03 sec
      ARIMA(2,1,0)(0,1,0)[52]             : AIC=-42.152, Time=1.27 sec
      ARIMA(2,1,0)(2,1,0)[52]             : AIC=-49.109, Time=19.99 sec
      ARIMA(2,1,0)(1,1,1)[52]             : AIC=-49.108, Time=13.39 sec
      ARIMA(2,1,0)(0,1,1)[52]             : AIC=inf, Time=24.77 sec
      ARIMA(2,1,0)(2,1,1)[52]             : AIC=-47.109, Time=61.72 sec
      ARIMA(1,1,0)(1,1,0)[52]             : AIC=-34.831, Time=6.85 sec
      ARIMA(3,1,0)(1,1,0)[52]             : AIC=-50.625, Time=12.80 sec

     Best model:  ARIMA(2,1,0)(1,1,0)[52]
     Total fit time: 474.552 seconds
```

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	144
Model:	SARIMAX(2, 1, 0)x(1, 1, 0, 52)	**Log Likelihood**	29.554
Date:	Thu, 16 Nov 2023	**AIC**	-51.109
Time:	03:42:36	**BIC**	-41.066
Sample:	01-02-2015 - 09-29-2017	**HQIC**	-47.057
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.7471	0.099	-7.511	0.000	-0.942	-0.552

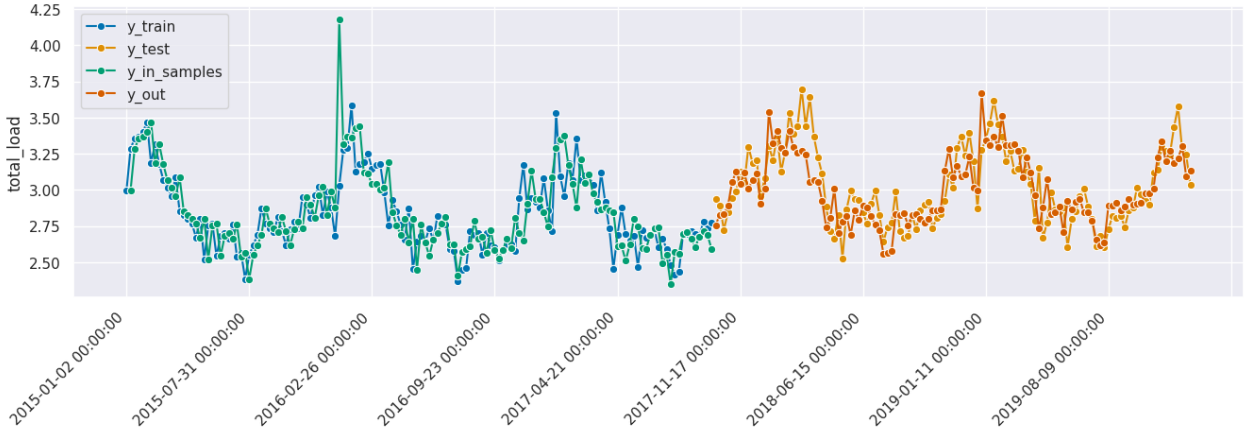
ar.L2	-0.4282	0.100	-4.265	0.000	-0.625	-0.231
ar.S.L52	-0.5070	0.128	-3.974	0.000	-0.757	-0.257
sigma2	0.0256	0.004	6.485	0.000	0.018	0.033

Ljung-Box (L1) (Q):	0.46	**Jarque-Bera (JB):**	3.92
Prob(Q):	0.50	**Prob(JB):**	0.14
Heteroskedasticity (H):	1.02	**Skew:**	-0.13
Prob(H) (two-sided):	0.96	**Kurtosis:**	3.98

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

```
fhin = ForecastingHorizon(y_train.index[1:], is_relative=False)
y_in_samples = model.predict(fhin)

fhout = ForecastingHorizon(y_test.index, is_relative=False)
y_out = model.predict(fhout)

# Результаты
plot_series(y_train, y_test, y_in_samples, y_out, labels=["y_train",
                                                          "y_test",
                                                          "y_in_samples",
                                                          "y_out"])
plt.xticks(rotation=45, ha='right');

get_metrics(y_test, y_out, title='Auto_ARIMA')
```

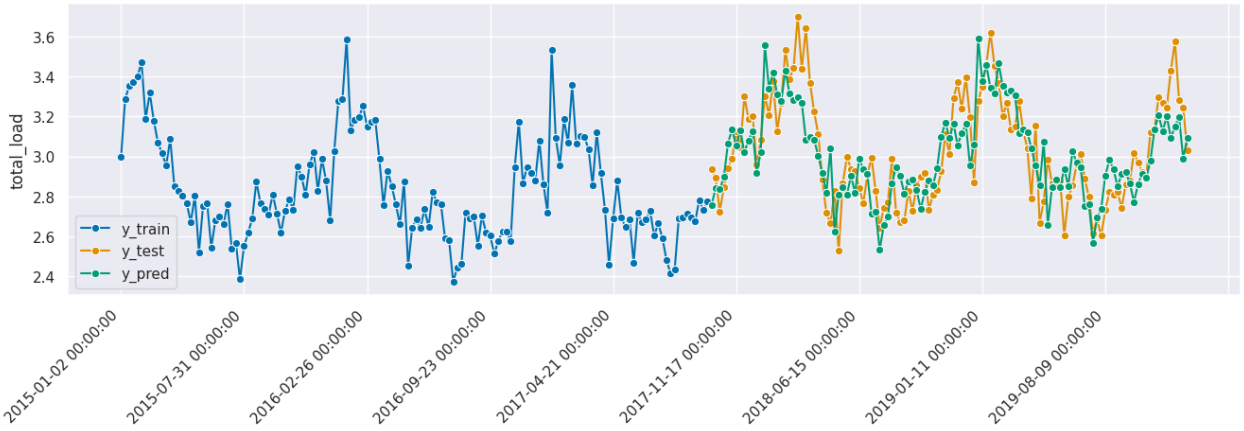
```
+-----------------+
|    Auto_ARIMA   |
+--------+--------+
| Metric | Value  |
+--------+--------+
|  MAE   | 0.1349 |
|  RMSE  | 0.1673 |
| SMAPE  | 0.0444 |
+--------+--------+
```



ARMA с декомпозицией

```
forecaster = TransformedTargetForecaster(
    [
        ("deseasonalize", Deseasonalizer(model="additive", sp=52)),
        ("forecast", ARIMA(order=(2, 1, 0), seasonal_order=(0, 0, 0, 0),  )),
    ]
)

forecaster.fit(y_train)
y_pred = forecaster.predict(fh)

plot_series(y_train, y_test, y_pred, labels=["y_train", "y_test", "y_pred"])
plt.xticks(rotation=45, ha='right');

print(f'sMAPE = {smape(y_pred.values, y_test.values):.3f}')
```

```
sMAPE = 0.045
```




```
get_metrics(y_test, y_pred)
```

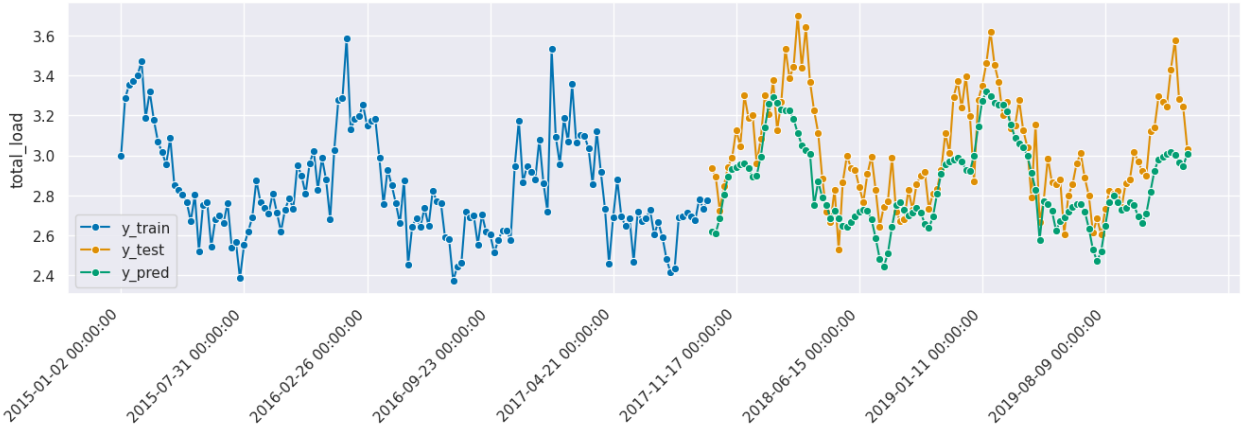
```
+-----------------+
|       None      |
+--------+--------+
| Metric | Value  |
+--------+--------+
|  MAE   | 0.1371 |
|  RMSE  | 0.1718 |
| SMAPE  | 0.0452 |
```

```
+--------+--------+
```

Prophet

```
from sktime.forecasting.fbprophet import Prophet


forecaster = Prophet(
    seasonality_mode='additive',
    freq='1w',
    n_changepoints=int(len(y_train) / 5),
    add_country_holidays={'country_name': 'Denmark'},
    yearly_seasonality=True
)

forecaster.fit(y_train)
y_pred = forecaster.predict(fh)
plot_series(y_train, y_test, y_pred, labels=["y_train", "y_test", "y_pred"])
plt.xticks(rotation=45, ha='right');

print(f'sMAPE = {smape(y_pred.values, y_test.values):.3f}')
```

```
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasona
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonali
DEBUG:cmdstanpy:input tempfile: /tmp/tmplwtcy26y/mfc3a7kr.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmplwtcy26y/98bmka6i.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/pr
03:59:01 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
03:59:01 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
sMAPE = 0.061
```



```
get_metrics(y_test, y_pred, title='Prophet')
```

```
     +-----------------+
     |     Prophet     |
     +--------+--------+
     | Metric | Value  |
     +--------+--------+
     |  MAE   | 0.1823 |
     |  RMSE  | 0.2241 |
     | SMAPE  | 0.0612 |
     +--------+--------+
```

```
predictions.sort_values(by=['smape'])
```

	method	smape	rmse	mae
7	AutoARIMA(2,1,0)(1,1,0,52)	0.0444	0.1673	0.1349
8	ARIMA_Decomposed	0.0452	0.1718	0.1371
5	ExpSmoothingHoltWinter	0.0499	0.1943	0.1499
4	SimpleExponentialSmoothingDecomposed	0.0529	0.2025	0.1585
9	Prophet	0.0612	0.2241	0.1823
1	Naive_Mean_Seasonal	0.0688	0.2508	0.2037
6	SARIMAX(2,1,0)(1,1,0,52)	0.0821	0.3156	0.258
3	Simple Exp Smoothing	0.0982	0.3752	0.2932
0	Naive_Last	0.1168	0.4188	0.3443
2	Naive_Drift	0.1197	0.4317	0.3513

Выводы

- Лучше всего себя показали модели авторегрессии, а именно, ARIMA с параметрами (2,1,0)(1,1,0,52), показавшая наименьшие значения по выбранным нами метрикам (RMSE, MAE, SMAPE). В частности, небольшое значение MAE указывает на то, что модель будет предсказывать будущие значения достаточно точно, без больших ошибок.

Анализ ряда на предмет классификации его сегментов

Формулировка задачи

В настоящем разделе, в соответствии с требованиями итогового проекта, проведем анализ ряда на предмет классификации его сегментов. Для выполнения настоящего задания я выбрал временной ряд о количестве вызовов такси в Нью-Йорке. Данный временной ряд отображает количество вызовов такси с шагом в 30 мин, за период с июля 2014 года по 31 января 2015 года.
Для задачи классификации сегментов, выделим 6 временных разделов в течение дня, а именно:

- ночь (с 00 до 04 утра);
- раннее утро (с 04 до 08);
- утро (с 8 до 12);
- день (с 12 до 16);
- вечер (с 16 до 20);
- поздний вечер (с 20 до 24);

И постараемся классифицировать данные сегменты. Решение данной задачи частично может помочь определить начало определенного периода дня, исходя из загруженности сервиса такси, тем самым, мы можем увеличивать или уменьшать автопарк, детектируя сегменты.

Загрузка и подготовка данных. Предварительный анализ

```
from IPython.display import clear_output

from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sktime.datatypes import convert_to

from sklearn.metrics import (
    make_scorer,
    confusion_matrix,
    accuracy_score,
    f1_score,
    precision_score,
    recall_score,
    classification_report,
)

from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_validate

from sktime.classification.shapelet_based import ShapeletTransformClassifier
from sktime.classification.dictionary_based import MUSE

%%shell
```

```
wget https://raw.githubusercontent.com/numenta/NAB/master/data/realKnownCause/
```

```
--2023-11-17 03:04:17--  https://raw.githubusercontent.com/numenta/NAB/mas
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.19
HTTP request sent, awaiting response... 200 OK
Length: 265771 (260K) [text/plain]
Saving to: ‘nyc_taxi.csv’

nyc_taxi.csv        100%[===================>] 259.54K  --.-KB/s    in 0.0

2023-11-17 03:04:17 (9.23 MB/s) - ‘nyc_taxi.csv’ saved [265771/265771]
```

```
taxi_data = pd.read_csv("nyc_taxi.csv", index_col="timestamp", parse_dates=Tru
```

```
taxi_data.head()
```

	value
timestamp	
2014-07-01 00:00:00	10844
2014-07-01 00:30:00	8127
2014-07-01 01:00:00	6210
2014-07-01 01:30:00	4656
2014-07-01 02:00:00	3820

```
taxi_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 10320 entries, 2014-07-01 00:00:00 to 2015-01-31 23:30:00
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   value   10320 non-null  int64
dtypes: int64(1)
memory usage: 161.2 KB
```

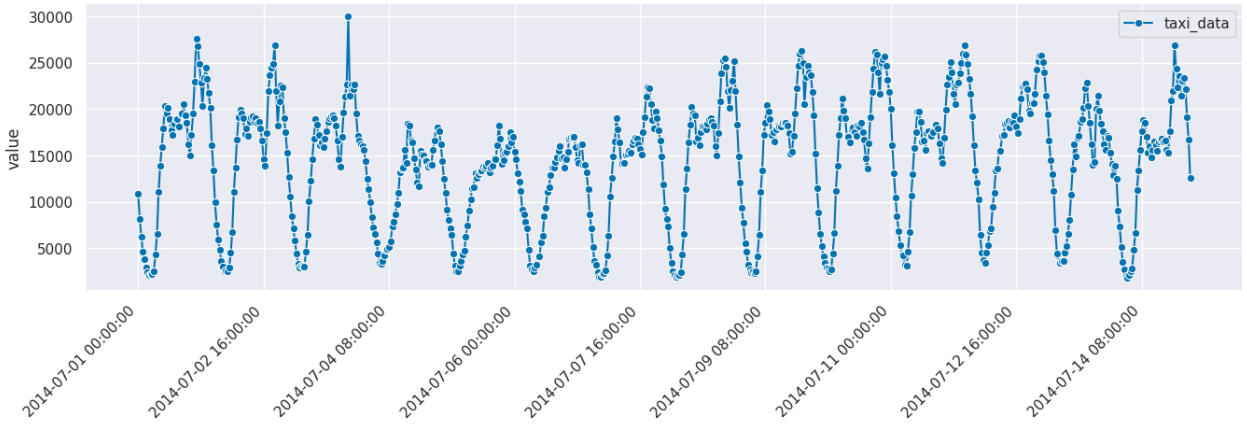
```
taxi_data.describe()
```

	value
count	10320.000000
mean	15137.569380
std	6939.495808
min	8.000000
25%	10262.000000
50%	16778.000000

75% 19838.750000

max 39197.000000

```
plot_series(taxi_data['value'].loc['2014-07-01':'2014-07-14'],
            labels=['taxi_data']
            );
plt.xticks(rotation=45, ha='right');
```




```
# добавь выходной день это или нет
taxi_data['day_of_week'] = taxi_data.index.dayofweek
taxi_data['is_business_day'] = taxi_data['day_of_week'].isin([0, 1, 2, 3, 4])
taxi_data = taxi_data.drop('day_of_week', axis=1)

taxi_data['hour_of_day'] = taxi_data.index.hour

time_segments = {
    'night': (0, 5),
    'morning': (6, 11),
    'afternoon': (12, 17),
    'evening': (18, 24),
}

def to_segments(hour):
    for segment, (start, end) in time_segments.items():
        if start <= hour <= end:
            return segment


taxi_data['time_of_day'] = taxi_data['hour_of_day'].apply(to_segments)
taxi_data.fillna('night', inplace=True)
```

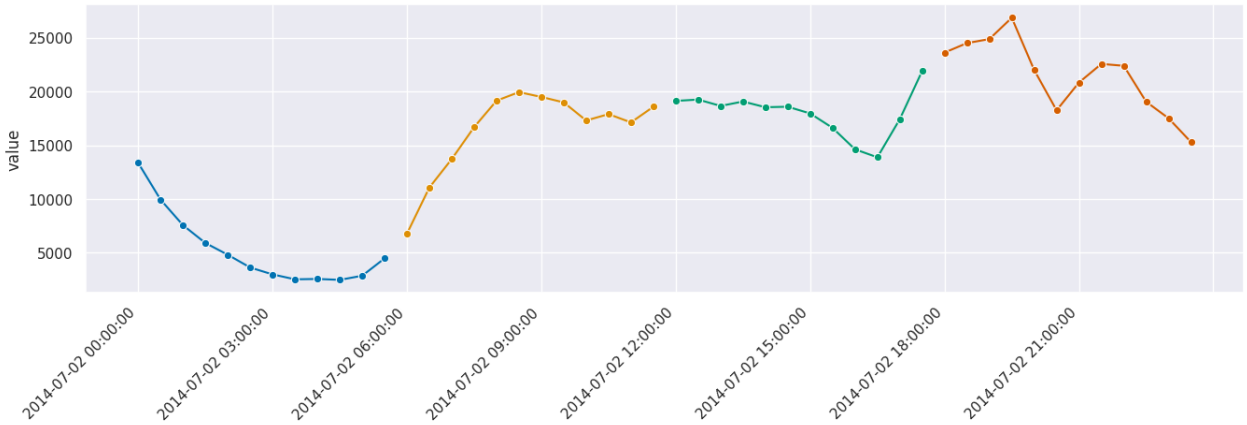
Посмотрим как различаются выбранные нами сегменты.

```
taxi_data['time_of_day'].loc['2014-07-01']
```

```
_          _  _
timestamp
2014-07-01 00:00:00        night
2014-07-01 00:30:00        night
2014-07-01 01:00:00        night
2014-07-01 01:30:00        night
2014-07-01 02:00:00        night
2014-07-01 02:30:00        night
2014-07-01 03:00:00        night
2014-07-01 03:30:00        night
2014-07-01 04:00:00        night
2014-07-01 04:30:00        night
2014-07-01 05:00:00        night
2014-07-01 05:30:00        night
2014-07-01 06:00:00      morning
2014-07-01 06:30:00      morning
2014-07-01 07:00:00      morning
2014-07-01 07:30:00      morning
2014-07-01 08:00:00      morning
2014-07-01 08:30:00      morning
2014-07-01 09:00:00      morning
2014-07-01 09:30:00      morning
2014-07-01 10:00:00      morning
2014-07-01 10:30:00      morning
2014-07-01 11:00:00      morning
2014-07-01 11:30:00      morning
2014-07-01 12:00:00    afternoon
2014-07-01 12:30:00    afternoon
2014-07-01 13:00:00    afternoon
2014-07-01 13:30:00    afternoon
2014-07-01 14:00:00    afternoon
2014-07-01 14:30:00    afternoon
2014-07-01 15:00:00    afternoon
2014-07-01 15:30:00    afternoon
2014-07-01 16:00:00    afternoon
2014-07-01 16:30:00    afternoon
2014-07-01 17:00:00    afternoon
2014-07-01 17:30:00    afternoon
2014-07-01 18:00:00      evening
2014-07-01 18:30:00      evening
2014-07-01 19:00:00      evening
2014-07-01 19:30:00      evening
2014-07-01 20:00:00      evening
2014-07-01 20:30:00      evening
2014-07-01 21:00:00      evening
2014-07-01 21:30:00      evening
2014-07-01 22:00:00      evening
2014-07-01 22:30:00      evening
2014-07-01 23:00:00      evening
2014-07-01 23:30:00      evening
Name: time_of_day, dtype: object
```

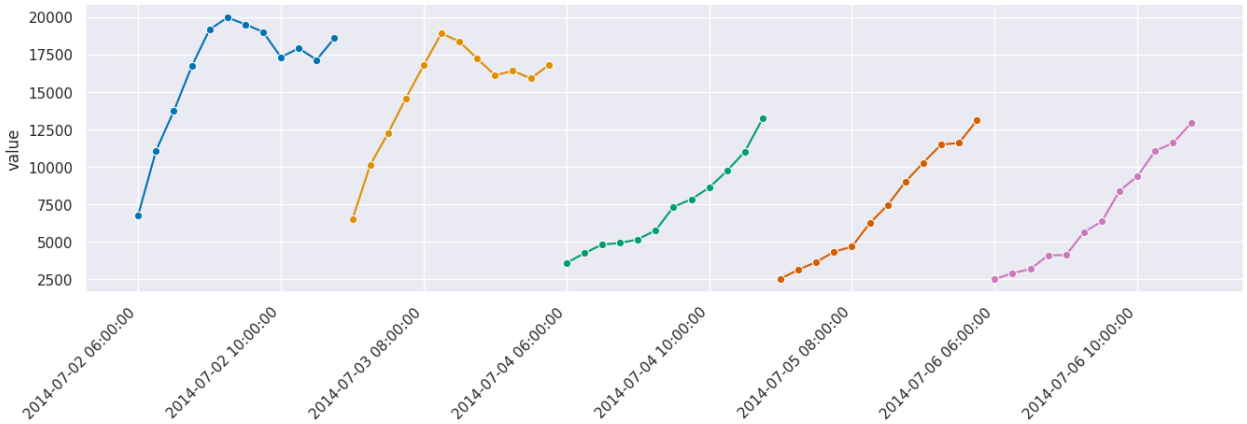
```
late_night = taxi_data[taxi_data[
    'time_of_day'] == "late_night"]['value']
early_morning = taxi_data[taxi_data[
    'time_of_day'] == "early_morning"]['value']
morning = taxi_data[taxi_data[
    'time_of_day'] == "morning"]['value']
```

```
afternoon = taxi_data[taxi_data[
    'time_of_day'] == "afternoon"]['value']
evening = taxi_data[taxi_data[
    'time_of_day'] == "evening"]['value']
night = taxi_data[taxi_data[
    'time_of_day'] == "night"]['value']

plot_series(night.loc['2014-07-02'],
            morning.loc['2014-07-02'],
            afternoon.loc['2014-07-02'],
            evening.loc['2014-07-02'],
);
plt.xticks(rotation=45, ha='right');
```



На графике ниже можно наблюдать как утренние вызовы меняются в зависимости от дня недели.

```
plot_series(morning.loc['2014-07-02'],
            morning.loc['2014-07-03'],
            morning.loc['2014-07-04'],
            morning.loc['2014-07-05'],
            morning.loc['2014-07-06'],
);
plt.xticks(rotation=45, ha='right');
```



Из графиков видно, что загруженность в разные временные периоды отличается. Также отличается их интенсивность. Могут возникнуть проблемы с разделением утреннего и дневного периодов.

Также следует отметить, что все периоды будут отличаться от аналогичных в праздничные и выходные дни

```
# проверим насколько точно распределены данные по сегментам
print(morning.resample('D').count().value_counts())
print(afternoon.resample('D').count().value_counts())
print(evening.resample('D').count().value_counts())
print(night.resample('D').count().value_counts())
```

```
12    215
Name: value, dtype: int64
12    215
Name: value, dtype: int64
12    215
Name: value, dtype: int64
12    215
Name: value, dtype: int64
```

```
# разделим сегменты по дням и преобразуем данные
morning = morning.values.reshape(
    -1, morning.resample('D').count().value_counts().index[0])
afternoon = afternoon.values.reshape(
    -1, afternoon.resample('D').count().value_counts().index[0])
evening = evening.values.reshape(
    -1, evening.resample('D').count().value_counts().index[0])
night = night.values.reshape(
    -1, night.resample('D').count().value_counts().index[0])

# проверим насколько точно распределены данные по сегментам
print(morning.shape)
print(afternoon.shape)
print(evening.shape)
print(night.shape)
```

```
(215, 12)
(215, 12)
(215, 12)
(215, 12)
```

```
X = np.concatenate((
    morning,
    afternoon,
    evening
```

```
    evening,
    night,
    ))
y = np.concatenate((
                    np.ones(night.shape[0]) - 1,
                    np.ones(night.shape[0]) + 0,
                    np.ones(night.shape[0]) + 1,
                    np.ones(night.shape[0]) + 2,
                   ))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(602, 12) (602,) (258, 12) (258,)
```

Классификация методом ближайших соседей

```
clf = KNeighborsClassifier(n_neighbors=10)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

```
print(classification_report(y_test,
                            y_pred,
                            target_names=list(time_segments.keys())))
```

```
                precision    recall  f1-score   support

         night       0.98      0.98      0.98        63
       morning       0.94      0.97      0.95        63
     afternoon       1.00      0.92      0.96        65
       evening       0.96      1.00      0.98        67

      accuracy                           0.97       258
     macro avg       0.97      0.97      0.97       258
  weighted avg       0.97      0.97      0.97       258
```

```
# метрики в виде словаря
scoring_metrics = {
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score, average='weighted'),
    'recall': make_scorer(recall_score, average='weighted'),
    'f1': make_scorer(f1_score, average='weighted')
}
# используем функцию кросс-валидации
scores = cross_validate(clf,
                        X_train, y_train,
                        scoring=scoring_metrics,
                        cv=StratifiedKFold(n_splits=5, shuffle = True, random_
```

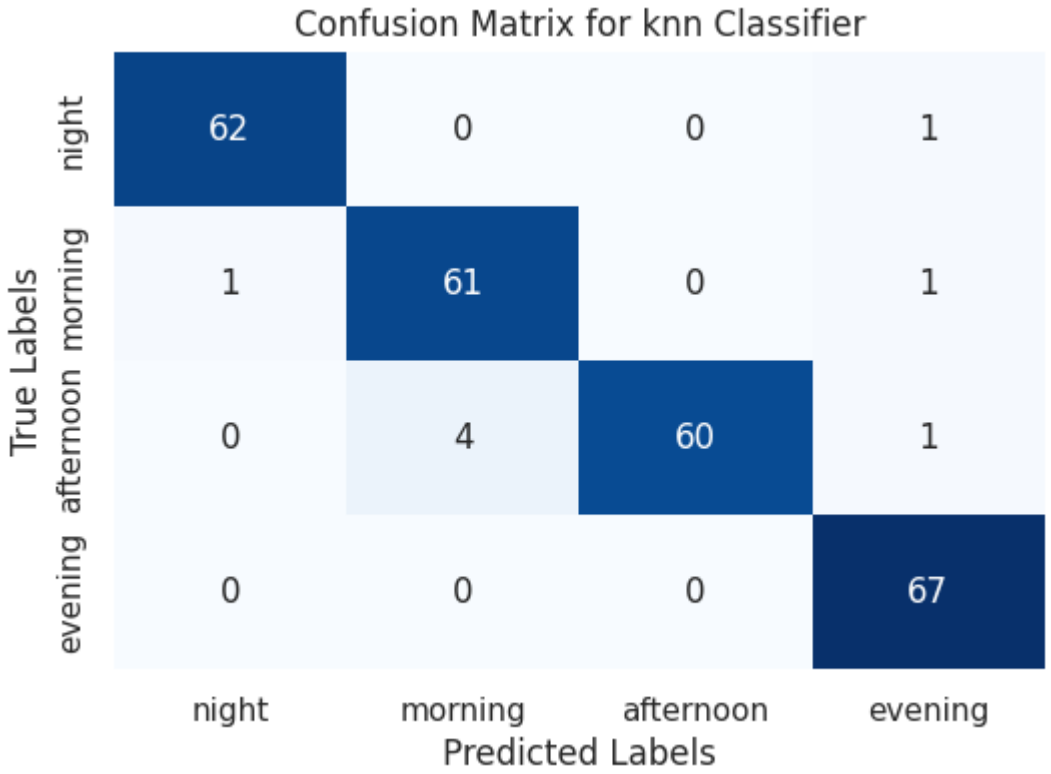
```
print('Результаты Кросс-валидации')
DF_cv_kNN = pd.DataFrame(scores)
```

```
display(DF_cv_kNN)
```

Результаты Кросс-валидации

	fit_time	score_time	test_accuracy	test_precision	test_recall	test_
0	0.001436	0.015796	0.983471	0.983996	0.983471	0.98346
1	0.001202	0.013730	0.966942	0.967901	0.966942	0.96690
2	0.001189	0.013604	0.975000	0.977206	0.975000	0.97491
3	0.001223	0.013780	0.958333	0.959200	0.958333	0.95782
4	0.001183	0.016081	1.000000	1.000000	1.000000	1.00000

```
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=list(time_segments.keys()), yticklabels=list(time_segm
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for knn Classifier')
plt.show()
```



Шейплет классификатор

```
clf = ShapeletTransformClassifier(estimator=RandomForestClassifier(n_estimator
                                  n_shapelet_samples=100,
                                  max_shapelets=100,
                                  batch_size=20)
```

```
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print(classification_report(y_test,
                            y_pred,
                            target_names=list(time_segments.keys())))
```

```
                  precision    recall  f1-score   support

           night       0.98      1.00      0.99        63
         morning       0.97      0.94      0.95        63
       afternoon       0.94      0.97      0.95        65
         evening       0.98      0.97      0.98        67

        accuracy                           0.97       258
       macro avg       0.97      0.97      0.97       258
    weighted avg       0.97      0.97      0.97       258
```

```
# метрики в виде словаря
scoring_metrics = {
    'accuracy': make_scorer(accuracy_score),
    'precision': make_scorer(precision_score, average='weighted'),
    'recall': make_scorer(recall_score, average='weighted'),
    'f1': make_scorer(f1_score, average='weighted')
}
# используем функцию кросс-валидации
scores = cross_validate(clf,
                        X_train, y_train,
                        scoring=scoring_metrics,
                        cv=StratifiedKFold(n_splits=5, shuffle = True, random_
```

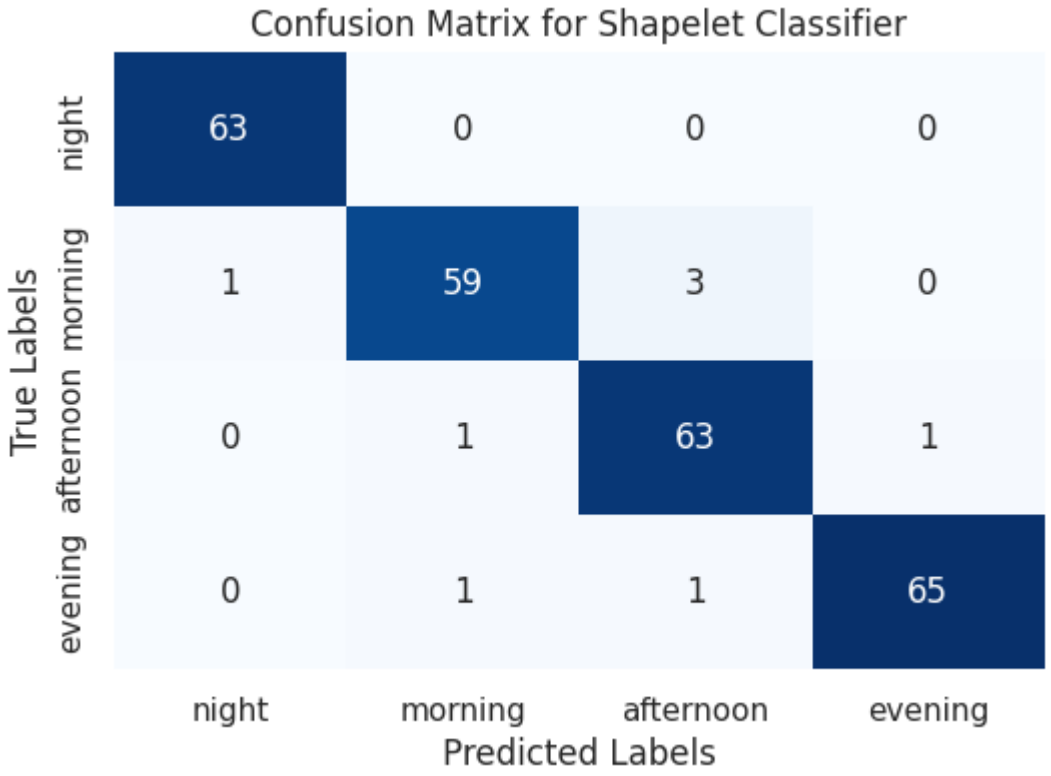
```
print('Результаты Кросс-валидации')
DF_cv_kNN = pd.DataFrame(scores)
display(DF_cv_kNN)
```

Результаты Кросс-валидации

	fit_time	score_time	test_accuracy	test_precision	test_recall	test_f
0	1.840186	0.152752	0.950413	0.958678	0.950413	0.95071
1	2.711891	0.163172	0.966942	0.970831	0.966942	0.96725
2	2.636560	0.175226	0.975000	0.975260	0.975000	0.97500
3	2.463008	0.088763	0.991667	0.991935	0.991667	0.99166
4	1.618674	0.094272	0.983333	0.983602	0.983333	0.98333

```
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=list(time_segments.keys()), yticklabels=list(time_segm
plt.xlabel('Predicted Labels')
```

```
plt.ylabel('True Labels')
plt.title('Confusion Matrix for Shapelet Classifier')
plt.show()
```



Dictionary-based. MUltivariate Symbolic Extension

```
clf = MUSE(random_state=50)
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
y_pred = clf.predict(X_test)

print(classification_report(y_test,
                            y_pred,
                            target_names=list(time_segments.keys())))
```

```
                precision    recall  f1-score   support

         night       0.98      0.98      0.98        63
       morning       0.98      0.94      0.96        63
     afternoon       0.93      0.98      0.96        65
       evening       1.00      0.99      0.99        67

      accuracy                           0.97       258
     macro avg       0.97      0.97      0.97       258
  weighted avg       0.97      0.97      0.97       258
```

```
# метрики в виде словаря
scoring_metrics = {
    'accuracy': make_scorer(accuracy_score),
```

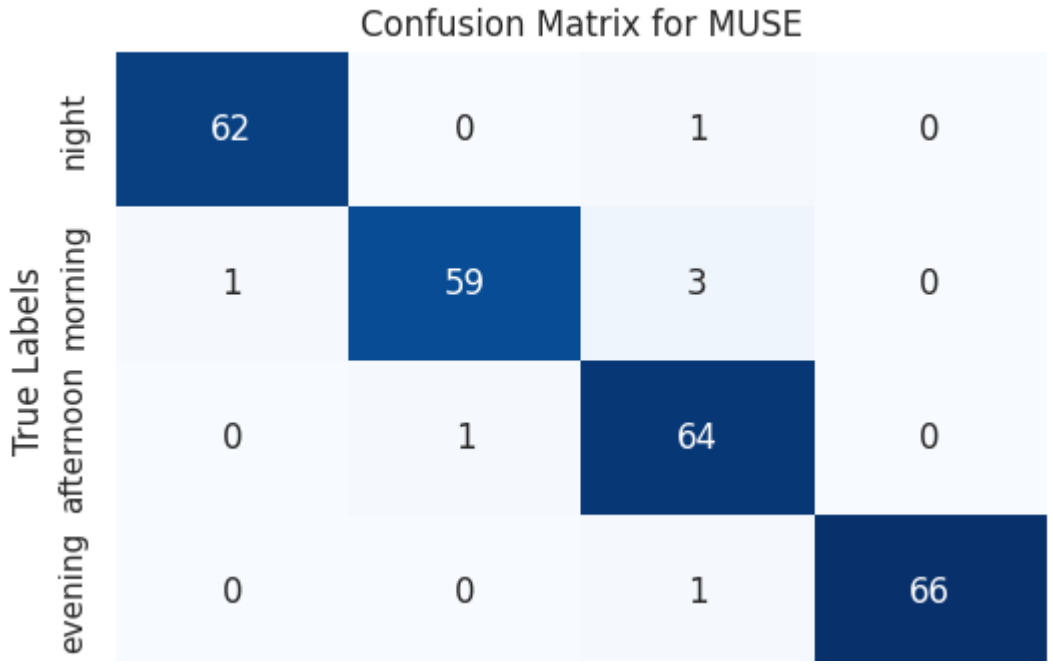
```
    'precision': make_scorer(precision_score, average='weighted'),
    'recall': make_scorer(recall_score, average='weighted'),
    'f1': make_scorer(f1_score, average='weighted')
}
# используем функцию кросс-валидации
scores = cross_validate(clf,
                        X_train, y_train,
                        scoring=scoring_metrics,
                        cv=StratifiedKFold(n_splits=5, shuffle = True, random_
```

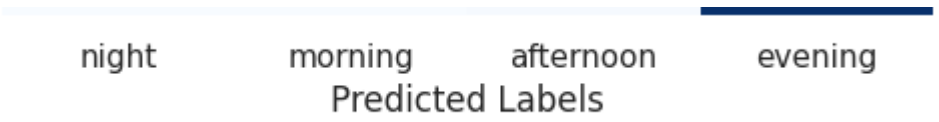
```
print('Результаты Кросс-валидации')
DF_cv_kNN = pd.DataFrame(scores)
display(DF_cv_kNN)
```

Результаты Кросс-валидации

	fit_time	score_time	test_accuracy	test_precision	test_recall	test_f
0	1.411691	0.180949	0.991736	0.992002	0.991736	0.9917
1	1.302919	0.146048	0.991736	0.992002	0.991736	0.9917
2	1.248949	0.112060	0.975000	0.975538	0.975000	0.9751
3	1.359977	0.146124	0.975000	0.976288	0.975000	0.9746
4	0.890492	0.070745	0.983333	0.983602	0.983333	0.9833

```
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=list(time_segments.keys()), yticklabels=list(time_segm
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for MUSE')
plt.show()
```





TSAI

```
%%shell
pip install tsai --quiet
```

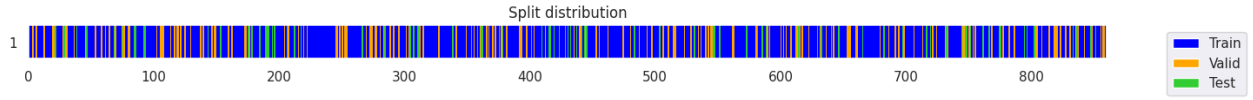
```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 324.2/324.2 kB 6.8 MB/s eta (
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.5/2.5 MB 38.3 MB/s eta 0:0(
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 235.6/235.6 kB 5.2 MB/s eta (
```

```
from tsai.all import *
import warnings
import torch
from torch import nn

computer_setup()
```

```
os              : Linux-5.15.120+-x86_64-with-glibc2.35
python          : 3.10.12
tsai            : 0.3.8
fastai          : 2.7.13
fastcore        : 1.5.29
torch           : 2.1.0+cu118
device          : cpu
cpu cores       : 1
threads per cpu : 2
RAM             : 12.68 GB
GPU memory      : N/A
```

```
X_ = np.atleast_3d(X).transpose(0, 2, 1)
X_ = X_.astype('float32')

class_map = {
    0:'late_night',
    1:'earlymorning',
    2:'morning',
    3:'afternoon',
    4:'evening',
    5:'night',
    }
class_map

labeler = ReLabeler(class_map)
y_ = labeler(y)



splits = get_splits(y_,
                    n_splits=1
```

```
                    n_splits=1,
                    valid_size=0.2,
                    test_size=0.1,
                    shuffle=True,
                    balance=False,
                    stratify=True,
                    random_state=42,
                    show_plot=True,
                    verbose=True)
splits
```




```
((#602) [108,535,712,39,243,385,656,351,699,605...],
 (#172) [303,580,102,269,172,750,540,661,118,219...],
 (#86) [632,38,748,178,361,761,375,84,441,423...])
```

```
tfms  = [None, [Categorize()]]
dsets = TSDatasets(X_, y_, tfms=tfms, splits=splits)

bs = 128
dls   = TSDataLoaders.from_dsets(dsets.train, dsets.valid, bs=[bs, bs*2])

archs = [
    (ResNet, {}),
    (InceptionTime, {}),
    (XceptionTime,  {}),
    (TCN, {}),
    (RNNPlus, {'n_layers':3, 'bidirectional': True} ),
    (LSTM,    {'n_layers':3, 'bidirectional': False}),
        ]

results = pd.DataFrame(columns=['arch',
                                'hyperparams',
                                'total params',
                                'train loss',
                                'valid loss',
                                'accuracy',
                                'time'])

for i, (arch, k) in enumerate(archs):

    model = create_model(arch, dls=dls, **k)

    print(model.__class__.__name__)

    learn = Learner(dls, model,  metrics=accuracy)
    start = time.time()
    learn.fit_one_cycle(20, 1e-3)
    elapsed = time.time() - start
    vals = learn.recorder.values[-1]
    results.loc[i] = [arch.__name__, k, count_parameters(model), vals[0], vals
    results.sort_values(by='accuracy', ascending=False, ignore_index=True, inp
```

```
results.sort_values(by='accuracy', ascending=False, ignore_index=True, inp
clear_output()
display(results)
```

	arch	hyperparams	total params	train loss	valid loss	accuracy	time
0	InceptionTime	{}	388868	0.090774	0.024650	1.000000	28
1	XceptionTime	{}	399480	0.478904	0.431051	0.994186	87
2	ResNet	{}	478724	0.106801	0.039970	0.988372	15
3	TCN	{}	66754	2.932964	0.298348	0.883721	41
4	RNNPlus	{'n_layers': 3, 'bidirectional': True}	142204	1.393069	1.388252	0.250000	5
5	LSTM	{'n_layers': 3, 'bidirectional':	203204	1.383635	1.378964	0.250000	9

Выводы и рекомендации по выбору модели предсказаний

- С нашей синтетической задачей классификации сегментов временного ряда отлично справились классические методы машинного обучения. Оценки по метрике accuracy приближались к 1 при кросс-валидации. При этом с помощью нейросетевых моделей нам удалось добиться даже более высокой точности.
- Можно также отметить, что с задачей не смогли справиться семейство рекуррентных моделей.

Отчет по аномалиям

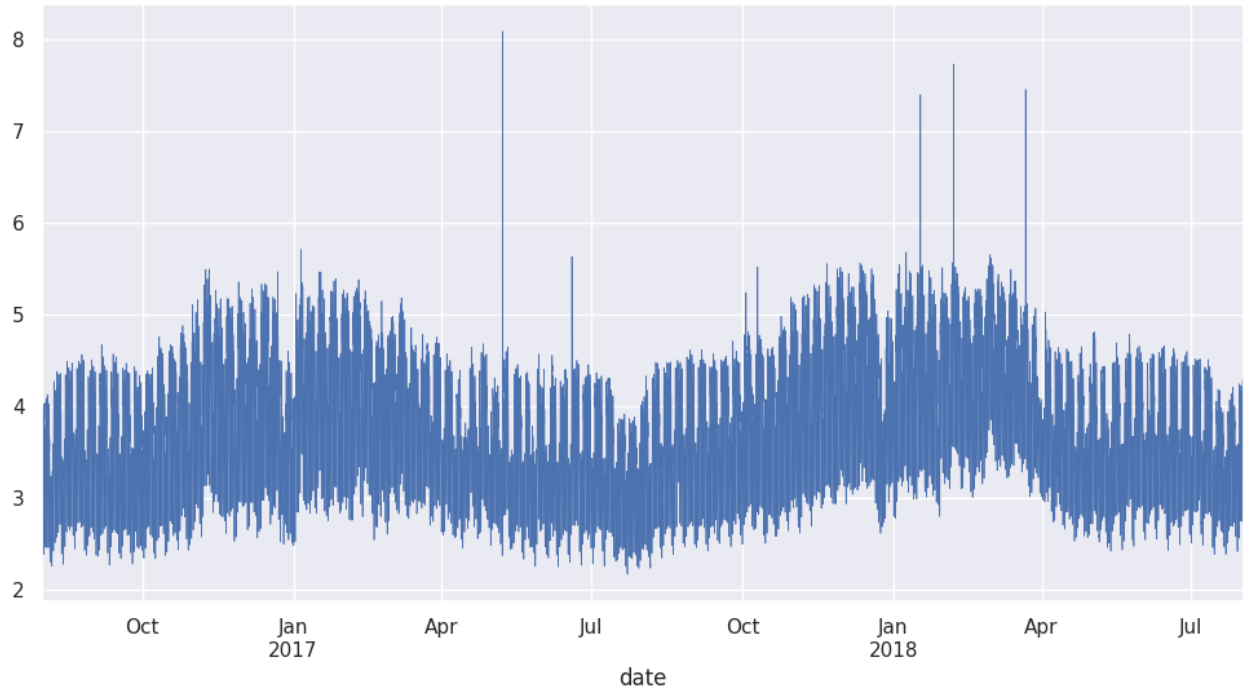
Для отчета по аномалиям возьмем, уже использовавшийся нами, датасет о потреблении электроэнергии в Дании. Однако, сократим период наблюдений, и возьмем период с августа 2016 года по август 2019 год.

```
data = df['total_load']['2016-08-01':'2018-08-01'].copy().to_frame()
data.describe()
```

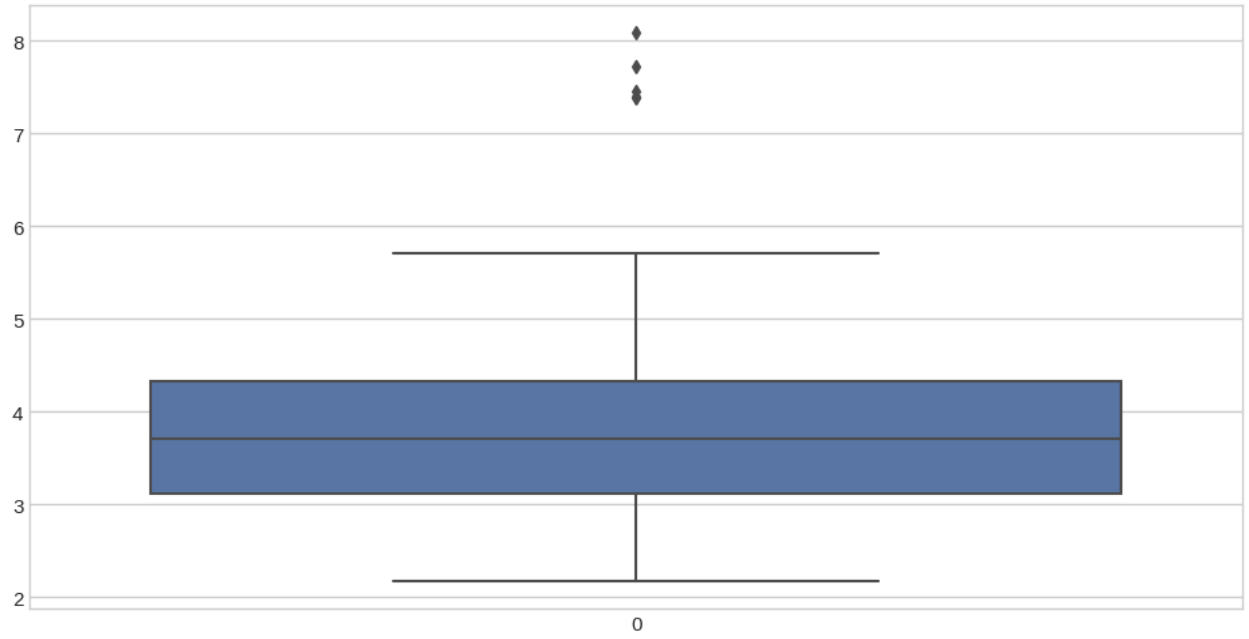
	total_load
count	17544.000000
mean	3.740572
std	0.748068
min	2.173110
25%	3.125612
50%	3.710220
75%	4.323433

max 8.088420

```
data['total_load'].plot(linewidth=0.5);
```

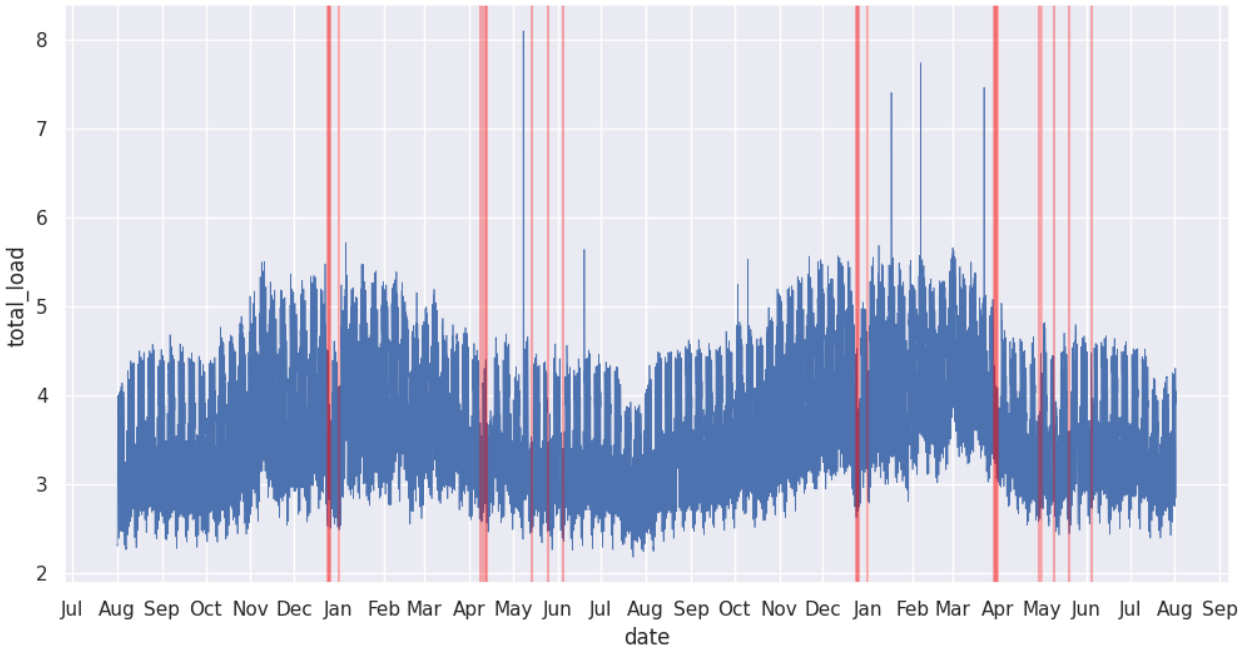



```
sns.boxplot(data['total_load']);
```

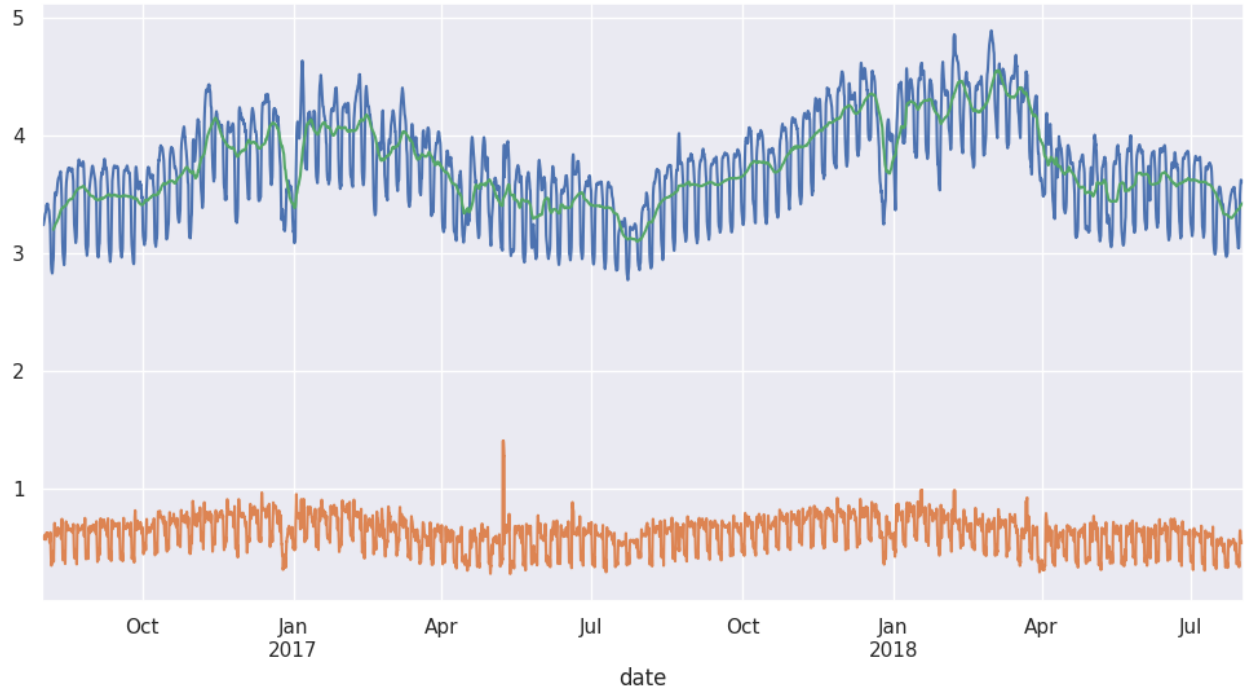


Как видно из визуализации временного ряда, а также графика ящик с усами, в данных присутствуют выбросы (значения выше 0.75 квантиля)

```
# посмотрим как праздничные дни могут влиять на потребление
# электроэнергии в течение календарного года

start_date = '2015-01-01'
end_date = '2016-01-01'
holiday_dates = holidays.loc[data.index[0]: data.index[-1]].index

ax = sns.lineplot(data['total_load'], linewidth=0.5);
for day in holiday_dates:
    ax.axvline(x=day, alpha=0.3, color='red');


months = MonthLocator()
monthsFmt = DateFormatter('%b')

ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(monthsFmt)
```



```
plt.rc('figure',figsize=(12,6))
plt.rc('font',size=15)
# create moving-averages
data['total_load'].rolling(24).mean().plot()
data['total_load'].rolling(24).std().plot()
data['total_load'].rolling(24*7).mean().plot();
```



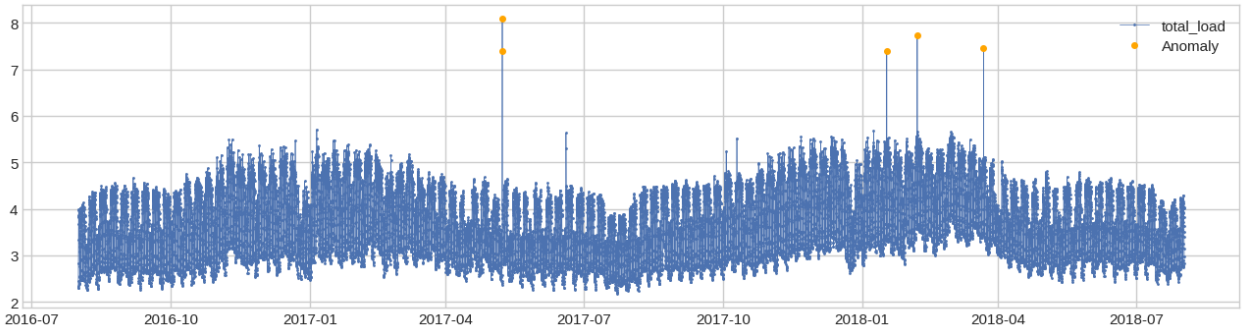
Anomaly Detection Toolkit

```
%%shell
pip install adtk
```

```
Collecting adtk
  Downloading adtk-0.6.2-py3-none-any.whl (60 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.0/61.0 kB 1.8 MB/s eta 0:(
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.10/dis
Requirement already satisfied: pandas>=0.23 in /usr/local/lib/python3.10/d
Requirement already satisfied: matplotlib>=3.0 in /usr/local/lib/python3.1(
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python
Requirement already satisfied: statsmodels>=0.9 in /usr/local/lib/python3.
Requirement already satisfied: packaging>=17.0 in /usr/local/lib/python3.1(
Requirement already satisfied: tabulate>=0.8 in /usr/local/lib/python3.10/(
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/d
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3
```

```
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/pyth
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/d
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/pyth
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/d
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packa
Installing collected packages: adtk
Successfully installed adtk-0.6.2
```

```
from adtk.detector import SeasonalAD
from adtk.visualization import plot

seasonal_vol = SeasonalAD()
anomalies = seasonal_vol.fit_detect(data['total_load'])
anomalies.value_counts()

plot(data['total_load'], anomaly=anomalies, anomaly_color="orange", anomaly_ta
```



Здесь мы видим, что на основе сезонности (определяется по умолчанию из тренировочных данных), модель правильно уловила выбросы в данных.

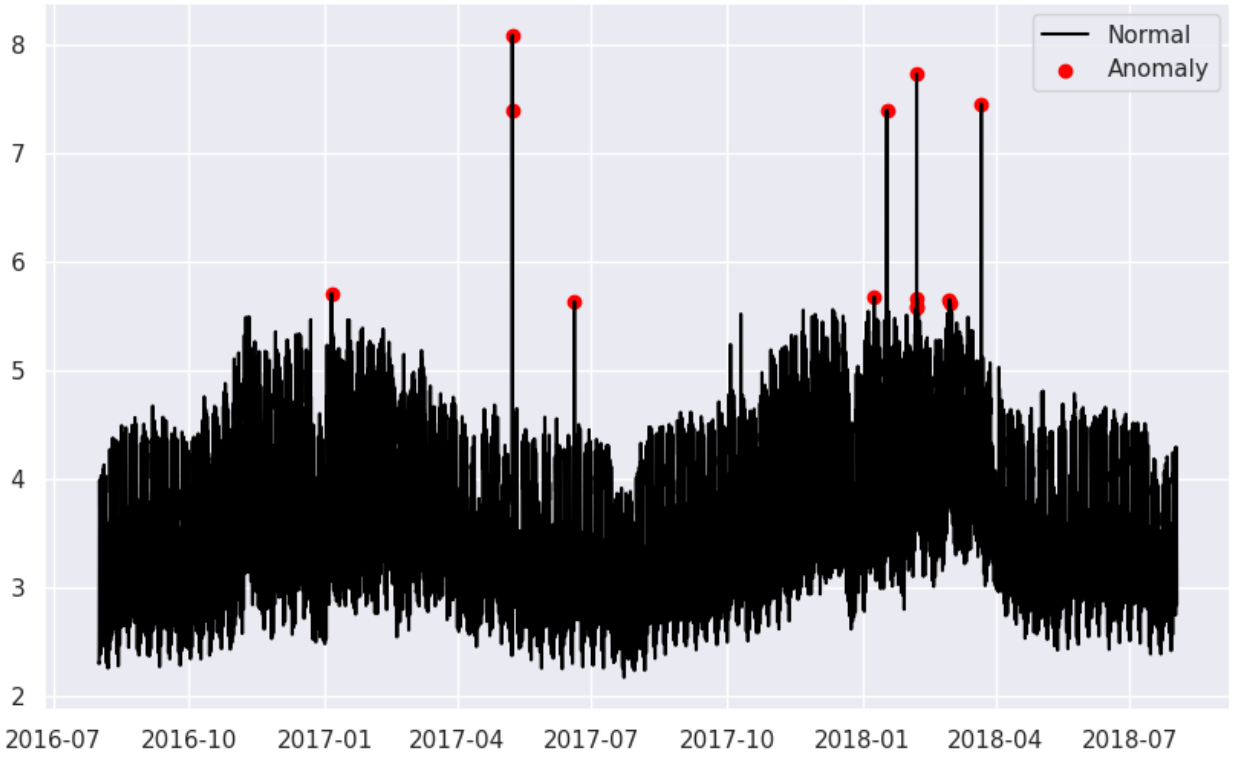
Isolation Forest

```
from sklearn.ensemble import IsolationForest

y = data['total_load'].values.reshape(-1, 1)
```

Установим значение outliers_fraction на уровне 0.001

```
outliers_fraction = 0.001

model =  IsolationForest(contamination=outliers_fraction)
```

```
model.fit(y)

data['anomaly_IF'] = model.predict(y)

# visualization
fig, ax = plt.subplots(figsize=(10,6))

a = data.loc[data['anomaly_IF'] == -1, ['total_load']] #anomaly

ax.plot(data.index, data['total_load'], color='black', label = 'Normal')
ax.scatter(a.index,a['total_load'], color='red', label = 'Anomaly')
plt.legend()
plt.show();
```



Как видно из графика, модель IsolationForest четко определила, отмеченные ранее нами аномалии, однако, также включила в них несколько новых значений, поскольку мы определили параметр выбросов в 0.1 %

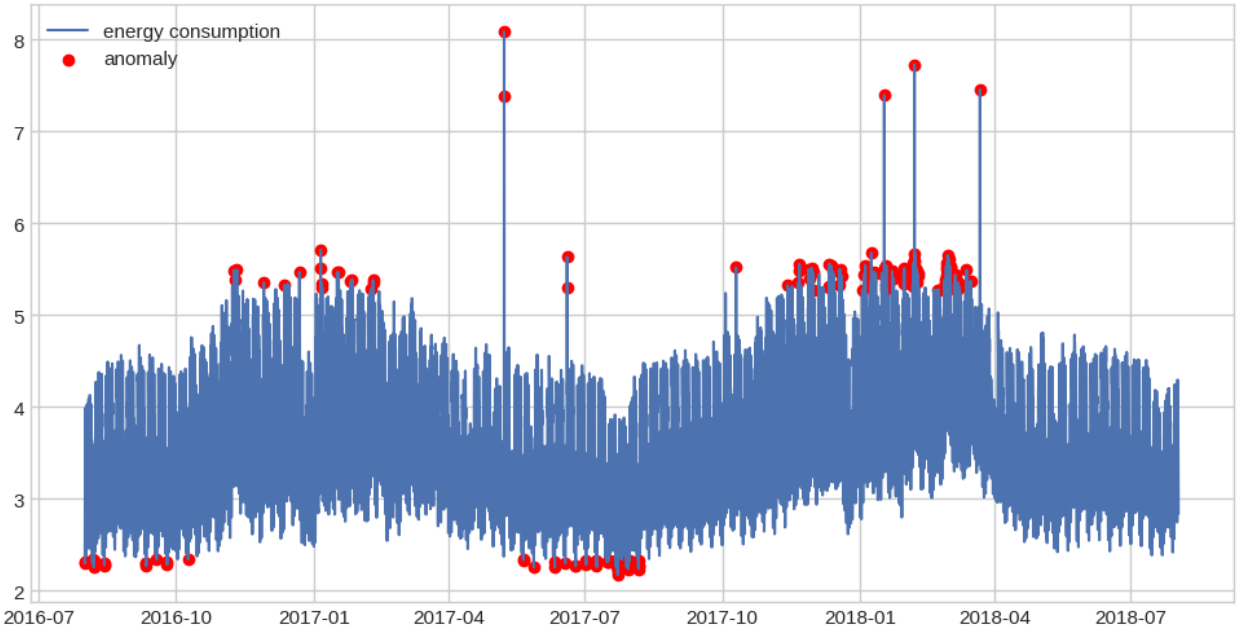
Метод ближайших соседей

```
from pyod.models.knn import KNN
```

```
# Initialize a KNN model
clf = KNN(contamination=0.01)  # 0.1 is the proportion of outliers you expect

# Fit the model
clf.fit(y)

# Get the prediction labels of the training data
y_train_pred = clf.labels_  # binary labels (0: inliers, 1: outliers)

# Outlier scores
y_train_scores = clf.decision_scores_  # raw outlier scores

y_train_pred
```

```
array([1, 1, 0, ..., 0, 0, 0])
```

```
anomalies_mask = y_train_pred == 1
fig, ax = plt.subplots()
ax.plot(data.index, data['total_load'], 'b-', label='energy consumption')
ax.scatter(data.index[anomalies_mask], data['total_load'][anomalies_mask], col
ax.legend(loc='upper left')
plt.show()
```



Выводы

- Все представленные модели хорошо справились с задачей определения аномалий. В нашем случае аномалии представляли собой выбросы. В зависимости от установленного ограничения модели четко определили

превышающие значения. При это можно отметить, что для определения выбросов можно использовать и визуализацию с предварительным анализом.

- В наших данных также присутствовали моменты снижения энергопотребления, что моделями не было задетектировано, как аномалия, либо новизна. Скорее всего, потому что данные изменения имеют закономерность, например, связаны с периодом отпуском или выходными днями.
- Что касается выбросов, и причины их возникновения, можно отметить несколько вариантов:

1. погодные аномалии (холодная погода);
2. сбой измерительных систем, или ошибки;
3. ремонтные работы;