Знакомство с SCIKIT-TIME (SKTIME)

Знакомство с библиотекой машинного обучения для анализа временных рядов sktime. Представления временных рядов с точки зрения задач машинного обучения. Преобразования временных раядов. Предсказание временных рядов.

Imports

Scikit-Time (sktime) - это набор инструментов Python с открытым исходным кодом для машинного обучения и работы с временными рядами. Это проект, реализуемый сообществом и финансируемый Советом экономических и социальных исследований Великобритании, Центром исследования потребительских данных и Институтом Алана Тьюринга.

Sktime расширяет API scikit-learn для задач временных рядов. Он предоставляет необходимые алгоритмы и инструменты преобразования для эффективного решения задач регрессии, прогнозирования и классификации временных рядов. Библиотека включает специальные алгоритмы обучения для временных рядов и методы преобразования, не представленные во многих других распространенных библиотеках.

Установим sktime и его зависимости

```
!pip install sktime --user
!pip install pmdarima
!pip install tbats
```

```
import sktime
import pandas as pd
import numpy as np
from warnings import simplefilter
```

```
from sktime.datasets import load_airline
```

```
from sktime.forecasting.model_selection import temporal_train_test_split
```

```
from sktime.forecasting.base import ForecastingHorizon
```

```
from sktime.forecasting.compose import (
    EnsembleForecaster,
    MultiplexForecaster,
    TransformedTargetForecaster,
    make_reduction,
)
```

```
from sktime.forecasting.model_evaluation import evaluate
```

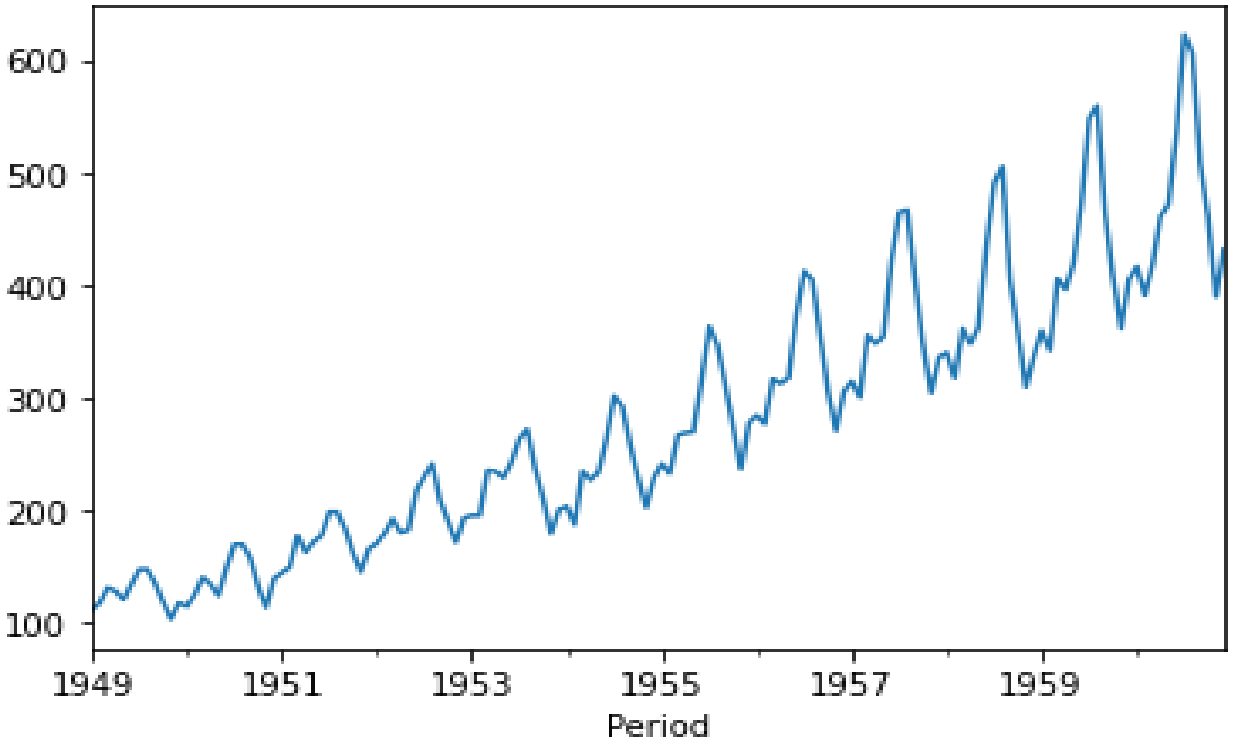
```
from sktime.forecasting.model_selection import (
    ExpandingWindowSplitter,
    ForecastingGridSearchCV,
    SlidingWindowSplitter,
    temporal_train_test_split,
)

from sktime.forecasting.exp_smoothing import ExponentialSmoothing

from sktime.forecasting.naive import NaiveForecaster

from sktime.forecasting.theta import ThetaForecaster

from sktime.forecasting.trend import PolynomialTrendForecaster

from sktime.performance_metrics.forecasting import sMAPE, smape_loss

from sktime.transformations.series.detrend import Deseasonalizer,
Detrender

from sktime.utils.plotting import plot_series

simplefilter("ignore", FutureWarning)

%matplotlib inline
```

Набор данных

Для использования встроенных наборов данных мы можем импортировать их из соответствующего модуля. Для начала импортируем уже знакомый вам набор данных с пассажирами авиакомпаний.

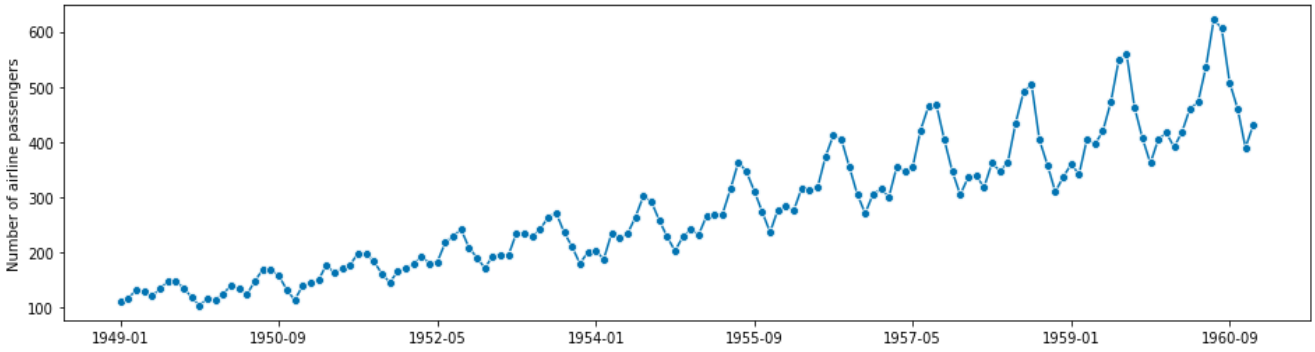
```
y = sktime.datasets.load_airline()

print('output data type: ',type(y))

y.plot();


print(y.describe())
```

```
output data type:  <class 'pandas.core.series.Series'>
count    144.000000
mean     280.298611
std      119.966317
min      104.000000
25%      180.000000
50%      265.500000
75%      360.500000
max      622.000000
Name: Number of airline passengers, dtype: float64
```



Мы также можем использовать встроенный метод plot_series для визуализации данных.

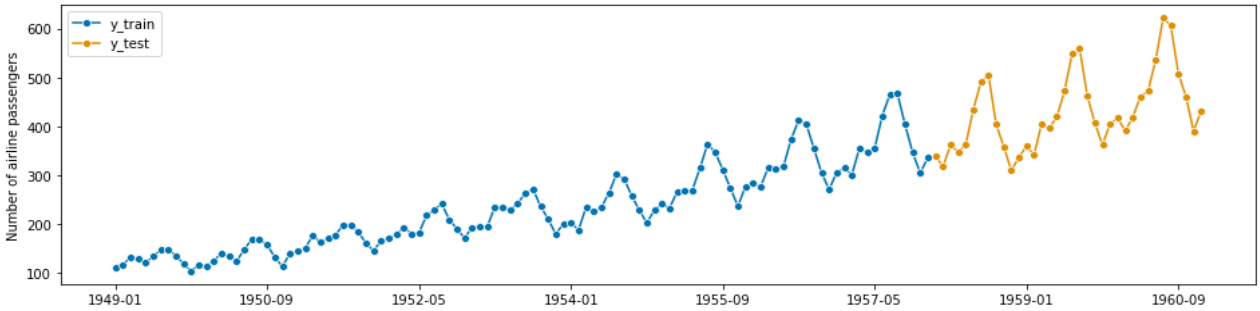
```
sktime.utils.plotting.plot_series(y);
```



Для разделения массивов или матриц на последовательные обучающие и тестовые подмножества мы будем использовать метод temporal_train_test_split. Мы будем использовать прогнозирование с заранее определенным горизонтом, используя переменную TEST_SIZE.

```
TEST_SIZE = 36

y_train, y_test =
sktime.forecasting.model_selection.temporal_train_test_split(y,
test_size=TEST_SIZE)

print('check splitted data size: ', y_train.shape[0], y_test.shape[0])
```

```
sktime.utils.plotting.plot_series(y_train, y_test, labels=["y_train",
"y_test"]);


check splitted data size:  108 36
```



После разделения мы можем указать горизонт прогнозирования, используя абсолютные значениях отсчетов времени.

```
fh = ForecastingHorizon(y_test.index, is_relative=False)
print(fh)
```

```
ForecastingHorizon(['1958-01', '1958-02', '1958-03', '1958-04', '1958-05',
'1958-06',
             '1958-07', '1958-08', '1958-09', '1958-10', '1958-11', '1958-
12',
             '1959-01', '1959-02', '1959-03', '1959-04', '1959-05', '1959-
06',
             '1959-07', '1959-08', '1959-09', '1959-10', '1959-11', '1959-
12',
             '1960-01', '1960-02', '1960-03', '1960-04', '1960-05', '1960-
06',
             '1960-07', '1960-08', '1960-09', '1960-10', '1960-11', '1960-
12'],
            dtype='period[M]', name='Period', freq='M', is_relative=False)
```

В качестве альтернативы мы можем использовать относительные отсчеты, испльзуя `np.array`

```
fh = np.arange(len(y_test)) + 1

fh = np.array([2, 5])  # 2nd and 5th step ahead
```

Задача предсказания

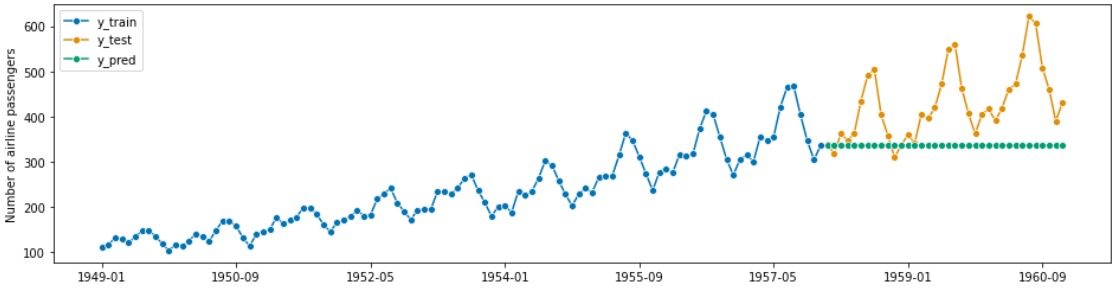
Традиционная постановка

Сделаем наивный прогноз, как на предыдущем занятии (по statsmodels.tsa). SKTime позволяет сделать это в общем стиле, совместимом с другими scikit библиотеками. Для оценки здесь будет использоваться так называемая мера Symmetry MAPE:

$$sMAPE = \frac{1}{H}\sum_{i=1}^{H}\frac{|y(h_i) - \hat{y}(h_i)|}{|y(h_i)| + |\hat{y}(h_i)|}$$

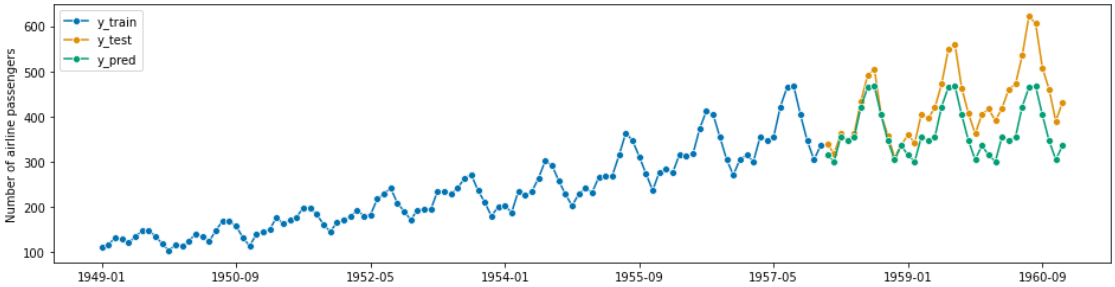
Для наивного прогноза:

```
fh = ForecastingHorizon(y_test.index, is_relative=False)

forecaster = NaiveForecaster(strategy="last")

forecaster.fit(y_train)

y_pred = forecaster.predict(fh)

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

print('score = ',smape_loss(y_pred, y_test))

score =  0.2319577038795143
```



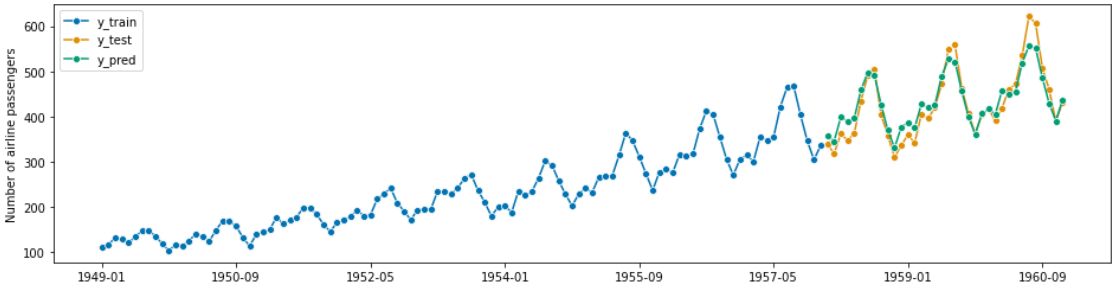
Для сезонного-наивного прогноза:

```
forecaster = NaiveForecaster(strategy="last", sp=12)

forecaster.fit(y_train)

y_pred = forecaster.predict(fh)

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

print('score = ',smape_loss(y_pred, y_test))

score =  0.145427686270316
```



Для Holt-Winter сглаживающего прогнзоа:

```
forecaster = ExponentialSmoothing(trend="add", seasonal="additive", sp=12)
forecaster.fit(y_train)
y_pred = forecaster.predict(fh)
plot_series(y_train, y_test, y_pred, labels=["y_train", "y_test",
"y_pred"])

print('score = ',smape_loss(y_pred, y_test))

score =  0.05027652903776341
```



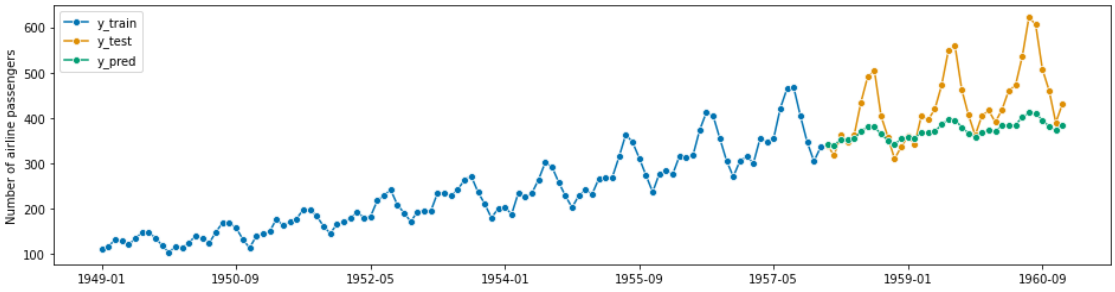
Примечание для получения простого экспоненциального сглаживания (SES) и двойного экспоненциального сглаживания (HOLT, DEA) используйте

```
ses = ExponentialSmoothing(sp=12)
holt = ExponentialSmoothing(trend="add", damped_trend=False, sp=12)
```

Для построения ансамбля методов используйте

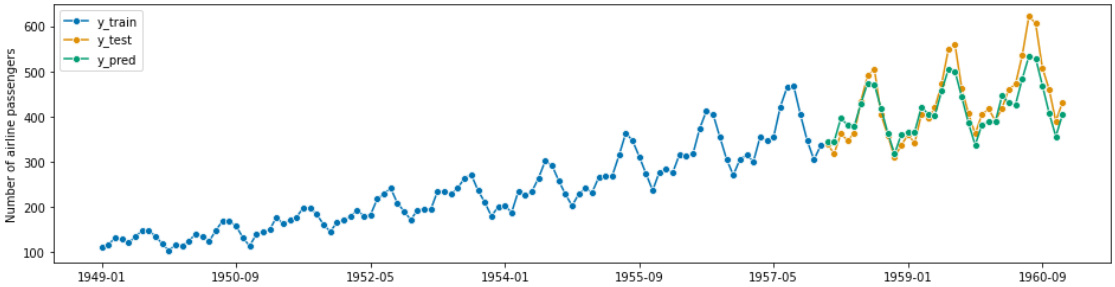
```
ses = ExponentialSmoothing(sp=12)

holt = ExponentialSmoothing(trend="add", damped_trend=False, sp=12)

damped_holt = ExponentialSmoothing(trend="add", damped_trend=True, sp=12)

holt_winter = ExponentialSmoothing(trend="add", seasonal="additive",
sp=12)
```

```
forecaster = EnsembleForecaster(
    [
        ("ses", ses),
        ("holt", holt),
        ("damped", damped_holt),
        ("holt-winter",holt_winter)
    ]
)
forecaster.fit(y_train)
y_pred = forecaster.predict(fh)
plot_series(y_train, y_test, y_pred, labels=["y_train", "y_test",
"y_pred"])

print('score = ',smape_loss(y_pred, y_test))

score =  0.1393047282084183
```



Для случая автоматизированного подбора гиперпараметров экспоненциального сглаживания (Холта-Винтера):

```
from sktime.forecasting.ets import AutoETS

forecaster = AutoETS(auto=True, sp=12, n_jobs=-1)

forecaster.fit(y_train)

y_pred = forecaster.predict(fh)

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

print('score = ',smape_loss(y_pred, y_test))

score =  0.06318722677034444
```



Упражнение 1

1. Сравнить точность предсказания различных вариантов модели ETS, таких как:

   - `trend : str{"add", "mul", "additive", "multiplicative", None}, optional (default=None)`
     `Type of trend component.`

   - `damped_trend : bool, optional (default=None)`
     `Should the trend component be damped.`

   - `seasonal : {"add", "mul", "additive", "multiplicative", None}, optional (default=None)`
     `Type of seasonal component.`

     Примечание. В обобщенном виде методы сглаживания могут быть объединены в так называемую модель Error-Trend_Seasonality (ETS). Модель может быть описана как Ошибка, Тренд, Сезонность (ETS): s = ETS (X, X, X) s, где X может быть N-None, A-аддитивным, M-мультипликативным, Ad-аддитивным затухающим, s-период сезонности, если S не равно None.

     Известные вам случае могут соответствовать следующим вариантам модели ETS: Простое экспоненциальное сглаживание соответствует ETS (A, N, N). Тройное экспоненциальное сглаживание соответствует ETS (A, A, A).

2. Поробуйте провести прогноз моделью ETS с учетом части `error`.
   `error : str, optional`
   `The error model. "add" (default) or "mul".`

3. Сравните результаты прогноза ETS с использованием (или без) трансформации временного ряда методом boxcox.
   `use_boxcox : {True, False, 'log', float}, optional`
   `Should the Box-Cox transform be applied to the data first? If 'log' then apply the log. If float then use lambda equal to float.`

Использование предсказаний с использованием библиотеки scikit-learn

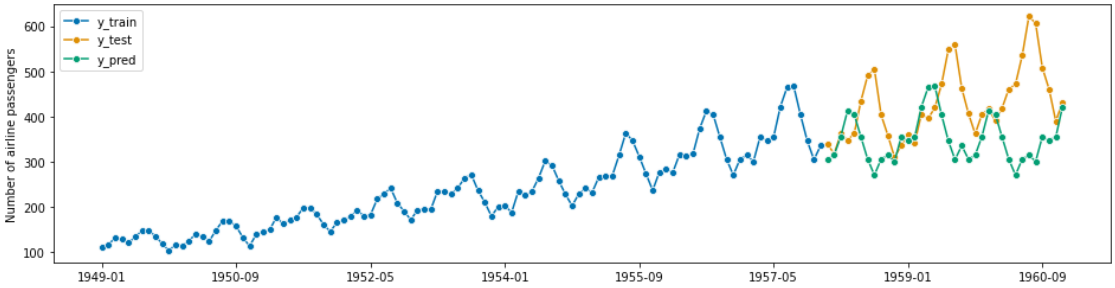
Помимо встроенных методов, SKTime позволяет применять подходы, основанные на scikit-learn. Например, задачу прогнозирования можно сравнить с задачей регрессии в sklearn. Однако, прямое использование стандартной регрессии sklearn, требуют наличия данных и меток, которые не нужны во временных рядах. Как будет показано ниже эта проблема может быть сравнительно просто решена.

Лучший способ составить прогноз с использованием обычной регрессии - это использовать так называемую технику “сокращения прогнозирования”, которая представляет собой преобразование неявно имеющейся задачи долгосрочной регрессии в задачу на основе скользящего окна. Такое преобразование можно сделать с помощью метода make_reduction, как показано ниже.

Идея редукции состоит в том, чтобы свести задачу регрессии в отношении временных рядов к проблеме табличной регрессии, как показано ниже. Обратите внимание, что в задаче табличной регрессии или регрессии скользящего окна на этапе обучения вы можете использовать предсказанные значения вместо известных в соответствующих позициях. Это показано серыми стрелками.

Посмотрим, как работает метод make_reduction. Существуют «прямые», «рекурсивные» и «многовыводные» стратегии прогнозирования. В прямой стратегии мы используем разные прогнозы для каждого результата (цели) (без серых стрелок). В рекурсивной стратегии мы используем предыдущие результаты в прогнозах для каждого следующего результата (с серыми стрелками). В стратегии с несколькими выходами мы напрямую прогнозируем несколько шагов.

```
from sklearn.neighbors import KNeighborsRegressor

REGRESSION_WINDOW = 5

regressor = KNeighborsRegressor(n_neighbors=1)
forecaster = make_reduction(regressor, window_length=REGRESSION_WINDOW,
strategy="recursive")

forecaster.fit(y_train)

y_pred = forecaster.predict(fh)

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

print('score = ',smape_loss(y_pred, y_test))

score =  0.2329490896108744
```



В make_reduction аргументы window_length и стратегии являются гиперпараметрами, которые мы, можем оптимизировать. В приведенном ниже примере мы используем поиск по сетке для проверки наилучшей длины окна. Для этого мы используется следующие шаги:

- создаем сетку длин окна
- создание начального предсказателя с помощью регрессора KNeighborsRegressor.
- Разделение выборки на тренировочную и для проверки (мы сделали это, сдвинув начало окна до 80% размера выборки и продвинувшись с длиной 25 точек до конца),
- производим поиск по сетке прогнозов с оценкой на каждой итерации и функцией SMAPE в качестве меры.

```
grid = {"window_length": [5, 7, 10, 12, 15,17,20]}

#initial forecaster
regressor  = KNeighborsRegressor(n_neighbors=1)
forecaster = make_reduction(regressor,
                            window_length=grid["window_length"][-1],
                            strategy="recursive")

# use temporal cross-validation to find the optimal parameter.
cros_val = SlidingWindowSplitter( initial_window=int(len(y_train) * 0.7),
window_length=25)

#grid search
gscv = ForecastingGridSearchCV(forecaster, strategy="refit", cv=cros_val,
param_grid=grid, scoring=sMAPE())

gscv.fit(y_train)


print('best window size = ',gscv.best_params_)

best window size =  {'window_length': 12}
```

to see the full protocol use the following code

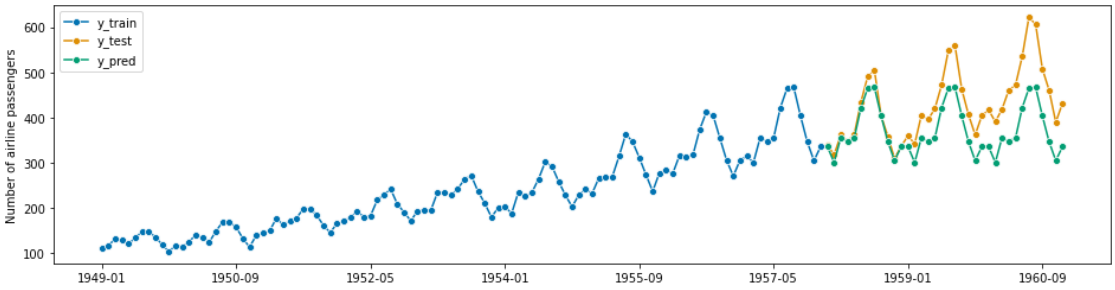
```
pd.DataFrame(gscv.cv_results_)

   mean_test_sMAPE  mean_fit_time  mean_pred_time                 params
\
0         0.133554       0.001182        0.002694   {'window_length': 5}
1         0.117308       0.001151        0.002576   {'window_length': 7}
2         0.095433       0.001213        0.002512  {'window_length': 10}
3         0.090602       0.001333        0.002454  {'window_length': 12}
4         0.095508       0.001121        0.002542  {'window_length': 15}
5         0.095508       0.001181        0.002361  {'window_length': 17}
6         0.095508       0.001150        0.002240  {'window_length': 20}

   rank_test_sMAPE
0              7.0
1              6.0
2              2.0
3              1.0
4              4.0
5              4.0
6              4.0
```

после поиска вы можете сделать прогноз с наилучшими результатами

```
y_pred = gscv.predict(fh)

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

```
print('score = ',smape_loss(y_pred, y_test))
```

```
score =  0.14008272913734346
```



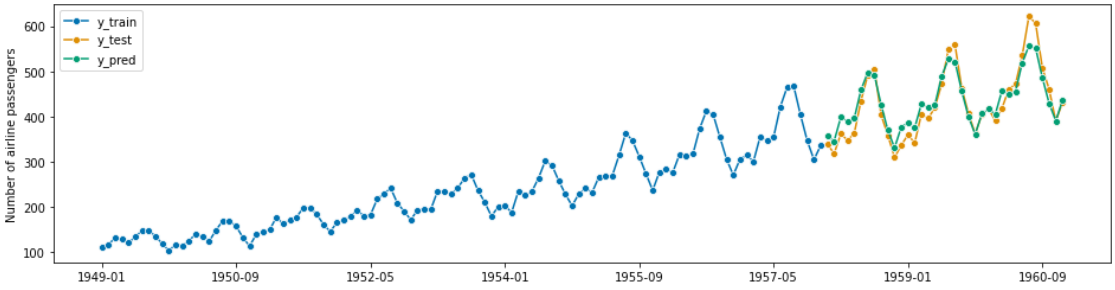
Также возможен поиск по сетке другими методами.

```
ses = ExponentialSmoothing(sp=12)

holt = ExponentialSmoothing(trend="add", damped_trend=False, sp=12)

holt_winter = ExponentialSmoothing(trend="add", seasonal="additive",
sp=12)


forecaster = MultiplexForecaster(
    forecasters=[
        ("ses", ses),
        ("holt", holt),
        ("holt_winter", holt_winter),
    ]
)

cv = SlidingWindowSplitter(initial_window=int(len(y_train) * 0.5),
window_length=30)

forecaster_grid = {"selected_forecaster": ["ses", "holt", "holt_winter"]}

gscv = ForecastingGridSearchCV(forecaster, cv=cv,
param_grid=forecaster_grid)

gscv.fit(y_train)

print(gscv.best_params_, "\n\n\n", gscv.best_forecaster_)
```

```
{'selected_forecaster': 'holt_winter'}
```

```
 MultiplexForecaster(forecasters=[('ses', ExponentialSmoothing(sp=12)),
                                  ('holt',
                                   ExponentialSmoothing(sp=12,
```

```
trend='add')),
                                    ('holt_winter',

ExponentialSmoothing(seasonal='additive',
                                                        sp=12,
trend='add'))],
                    selected_forecaster='holt_winter')

y_pred = gscv.predict(fh)
plot_series(y_train, y_test, y_pred, labels=["y_train", "y_test",
"y_pred"])
print('score = ',smape_loss(y_pred, y_test))

0.05027652903776341
```



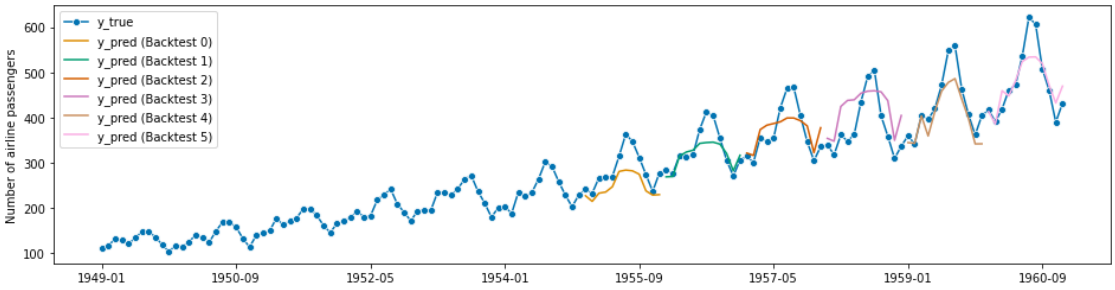
Помимо простого деления train-test-split мы можем сделать ExpandingWindowSplitter.

```
from sklearn.ensemble import RandomForestRegressor

regressor  = RandomForestRegressor()

forecaster = make_reduction(regressor,
                            window_length=12,
                            strategy="recursive")

cv = ExpandingWindowSplitter(
    step_length=12, fh=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
initial_window=72)

df = evaluate(forecaster=forecaster, y=y, cv=cv, strategy="refit",
return_data=True)
df.iloc[:, :5]
```

	cutoff	fit_time	len_train_window	pred_time	test_sMAPE
0	1954-12	0.125927	72	0.071958	0.132538
1	1955-12	0.123929	84	0.074957	0.055439
2	1956-12	0.126938	96	0.078944	0.078507
3	1957-12	0.142903	108	0.084950	0.127128
4	1958-12	0.146924	120	0.083951	0.070665
5	1959-12	0.140903	132	0.072958	0.053559

теперь вы можете видеть результат этого ExpandingWindowSplitter.

```
# visualization of a forecaster evaluation
fig, ax = plot_series(
    y,
    df["y_pred"].iloc[0],
    df["y_pred"].iloc[1],
    df["y_pred"].iloc[2],
    df["y_pred"].iloc[3],
    df["y_pred"].iloc[4],
    df["y_pred"].iloc[5],
    markers=["o", "", "", "", "", "", ""],
    labels=["y_true"] + ["y_pred (Backtest " + str(x) + ")" for x in
range(6)],
)
ax.legend();
```

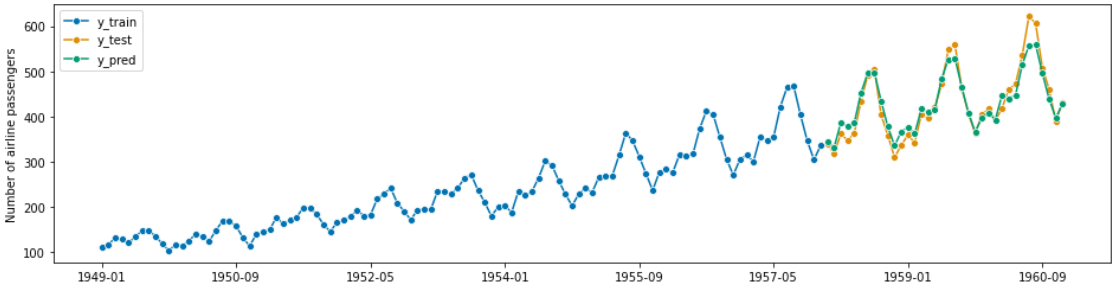


Упражнение 2

1. Сделайте EnsembleForecaster как минимум двумя методами из sktime и sklearn.
2. Сделайте ForecastingGridSearchCV для поиска оптимального количества ближайших соседей для метода KNN.
3. Сделайте MultiplexForecaster для Naive, Holt-Winter, Random Forest Regressor, AdaBoostRegressor.

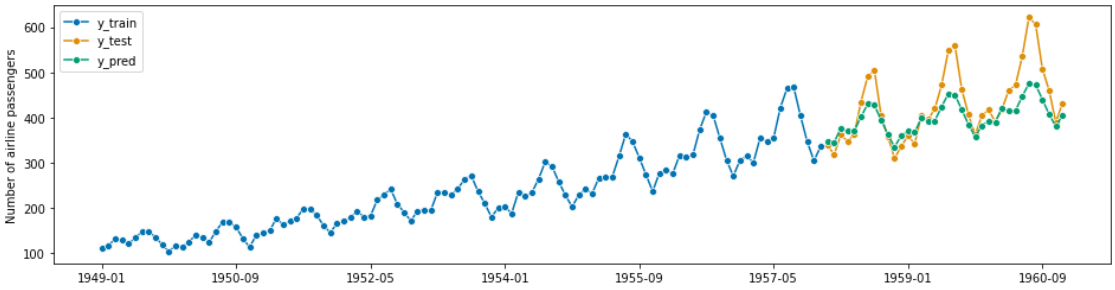
Обратите внимание, что sktime содержит множество заимствований из других пакетов. В некоторых случаях их необходимо будет установить дополнительно.

```
from sktime.forecasting.arima import ARIMA, AutoARIMA

forecaster = AutoARIMA(sp=12, suppress_warnings=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"])
print('score = ',smape_loss(y_pred, y_test))

0.04117062367046532
```




```
from sktime.forecasting.tbats import TBATS

forecaster = TBATS(sp=12, use_trend=True, use_box_cox=False)
forecaster.fit(y_train)
y_pred = forecaster.predict(fh)
plot_series(y_train, y_test, y_pred, labels=["y_train", "y_test",
"y_pred"])
print('score = ',smape_loss(y_pred, y_test))
```

```
0.08493353477049963
```

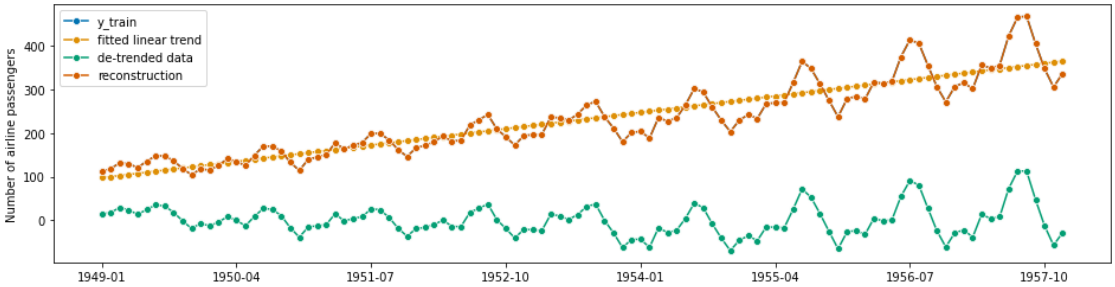


Преобразование данных

Sktimes содержит ряд инструментов для декомпозиции временных рядов и прогнозирования компонентов. Один из универсальных инструментов - преобразование детрендинг. Мы можем использовать его с методом прогнозирования тренда PolynomialTrendForecaster. В примере ниже мы будем использовать линейный тренд (степень полинома равна 1).

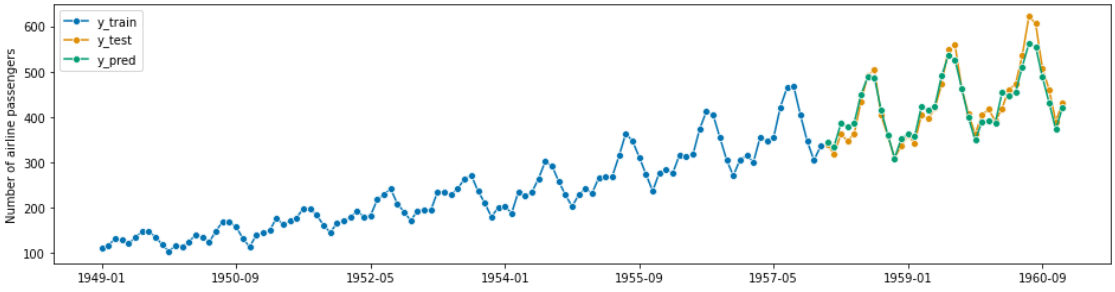
```
# liner detrending
forecaster = PolynomialTrendForecaster(degree=1)

transformer = Detrender(forecaster=forecaster)

y_detrend = transformer.fit_transform(y_train)

fh_ins = -np.arange(len(y_train))  # in-sample forecasting horizon

y_trend = forecaster.fit(y_train).predict(fh=fh_ins)

reconstructed = y_trend + y_detrend
```

```
plot_series(y_train, y_trend, y_detrend, reconstructed,
            labels=["y_train", "fitted linear trend", "de-trended data",
"reconstruction"]);
```



Мы можем сделать конвейер трансформаций для прогноза, как это показано ниже.

```
regressor  = KNeighborsRegressor(n_neighbors=3)

forecaster = TransformedTargetForecaster(
    [
        ("deseasonalize", Deseasonalizer(model="multiplicative", sp=12)),
        ("detrend",
Detrender(forecaster=PolynomialTrendForecaster(degree=1))),
        ( "forecast",
make_reduction(regressor,window_length=15,strategy="recursive",),
         ),
    ]
)
forecaster.fit(y_train)
y_pred = forecaster.predict(fh)
plot_series(y_train, y_test, y_pred, labels=["y_train", "y_test",
"y_pred"])
print('score = ',smape_loss(y_pred, y_test))

score =  0.03983950097748029
```



Упражнение 3

1. Создайте конвейер с Deseasonalizer и Naive Forecast для остаточной части, сравните и объясните разницу со случаем без преобразования.
2. Сделайте тренд-сезонность-остаток-декомпозицию, проанализируйте результаты.