# Библиотека SCIKIT-TIME (SKTIME)

[Scikit-Time (Sktime )](https://www.sktime.org/en/stable/index.html) — это набор инструментов Python с [открытым исходным кодом](https://github.com/alan-turing-institute/sktime) для работы временными рядами. В основном sktime предполагает использование методов машинного обучения.

Библиотека [Sktime](https://www.sktime.org/en/stable/index.html) расширяет API scikit-learn для задач временных рядов. Он предоставляет необходимые алгоритмы и инструменты преобразования для эффективного решения задач регрессии, прогнозирования и классификации. Библиотека включает специальные алгоритмы обучения временных рядов и методы преобразования.

Если у вас не установленна данная библиотека, давайте установим ее при помощи следующего кода. Рекомендации по установке можно найти в официальной документации [тут](https://www.sktime.org/en/stable/installation.html).

Отметим, что хоть мы и указываем тут код для установки компонент sktime, в общем случае рекомендуем установить полную версию пакета с использованием командной строки. Например, для этого можно использовать следующий скрипт

pip install --upgrade sktime[all\_extras]

try:  
 import sktime  
except:  
# !pip install sktime --user  
# !pip install pmdarima  
# !pip install statsmodels  
# !pip install prophet  
 !pip install --upgrade sktime[all\_extras]  
import sktime

import warnings  
from statsmodels.tools.sm\_exceptions import ConvergenceWarning  
warnings.simplefilter('ignore', ConvergenceWarning)

import sktime  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt

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 MeanAbsolutePercentageError, MeanSquaredError  
from sktime.transformations.series.detrend import Deseasonalizer, Detrender  
from sktime.utils.plotting import plot\_series  
from sktime.forecasting.compose import TransformedTargetForecaster  
from sktime.forecasting.trend import PolynomialTrendForecaster  
from sktime.transformations.panel.tsfresh import TSFreshFeatureExtractor  
from sktime.forecasting.fbprophet import Prophet  
from sktime.forecasting.tbats import TBATS  
smape = MeanAbsolutePercentageError(symmetric = True)  
rmse = MeanSquaredError(square\_root=True)  
  
from sklearn.ensemble import RandomForestRegressor  
  
from sklearn.pipeline import make\_pipeline  
  
from sklearn.metrics import r2\_score  
from sklearn.ensemble import RandomForestClassifier  
  
  
  
r2\_score = lambda y\_pred, y\_test: 1-np.sum(np.square(y\_pred - y\_test))/np.sum(np.square(y\_test - np.mean(y\_test)))  
  
warnings.simplefilter("ignore", FutureWarning)  
%matplotlib inline

В качестве набора данных давайте загрузим рассмотренный ранее набор данных о потреблении электричества в Германии.

path\_ts = 'de\_data.csv'  
df = pd.read\_csv(path\_ts, parse\_dates=['Date'], index\_col="Date")  
df=df.fillna(df.mean())  
df.head()

Consumption Wind Solar Wind+Solar  
Date   
2015-01-01 1088.317 325.165 103.386051 371.25795  
2015-01-02 1246.588 603.554 7.757000 611.31100  
2015-01-03 1117.554 462.955 7.237000 470.19200  
2015-01-04 1081.980 385.023 19.982000 405.00500  
2015-01-05 1325.920 216.540 26.522000 243.06200

В первую очередь в данном уроке мы будем рассматривать однопеременные методы (univariate time series). Поэтому выберем один из столбцов данных в качестве отдельного временного ряда.

Кроме того, в ходе предыдущего анализа было установлено, что ряд имеет как минимум 2 составляющие (быструю, с периодом неделя и медленную с периодом 1 год). Для упрощения анализа и без значительных потерь значимости удалим быструю составляющую при помощи перегруппировки данных.

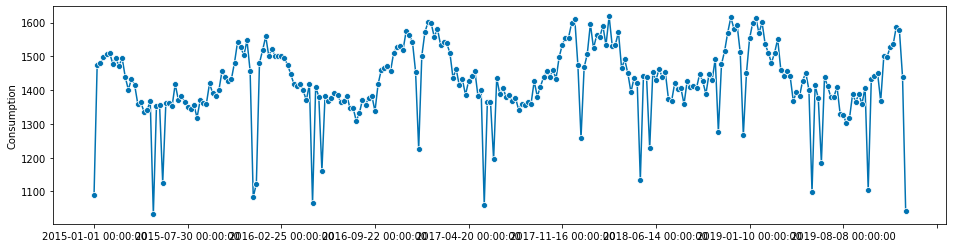
y = df.Consumption.asfreq('7d')  
# y.index = pd.PeriodIndex(y.index) # pd.to\_datetime(y.index)  
y.head()

Date  
2015-01-01 1088.317  
2015-01-08 1474.375  
2015-01-15 1479.245  
2015-01-22 1496.905  
2015-01-29 1508.443  
Freq: 7D, Name: Consumption, dtype: float64

# В качестве альтернативы можно бы было использовать resample('7d'),   
# но в этом случае пришлось бы дополнительно использовать следующую функцию.  
# def add\_freq(idx, freq=None):  
# """Add a frequency attribute to idx, through inference or directly.  
# Returns a copy. If `freq` is None, it is inferred.  
# """  
# idx = idx.copy()  
# if freq is None:  
# if idx.freq is None:  
# freq = pd.infer\_freq(idx)  
# else:  
# return idx  
# idx.freq = pd.tseries.frequencies.to\_offset(freq)  
# if idx.freq is None:  
# raise AttributeError('no discernible frequency found to `idx`. Specify'  
# ' a frequency string with `freq`.')  
# return idx  
# y = df.Consumption.resample('7d').sum()[1:-2]  
# y.index = add\_freq(y.index)

Проведем визуализацию полученных данных при помощи встроенных утилит.

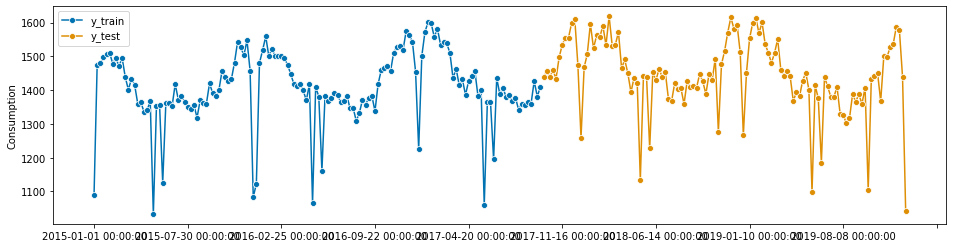
sktime.utils.plotting.plot\_series(y);



Для разделения данных воспользуемся функцией temporal\_train\_test\_split, которая позволяет адаптировать разделение данных не разрушая временные зависимости в данных. Отметим, что данная функция не единственный тип разделения данных. Более подробную информацию можно найти [тут](https://notebooks.githubusercontent.com/view/ipynb?browser=chrome&color_mode=auto&commit=24f6be86f95bfc1ec246dee7dcdd455e0a84a872&device=unknown&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f616c616e2d747572696e672d696e737469747574652f736b74696d652f323466366265383666393562666331656332343664656537646364643435356530613834613837322f6578616d706c65732f77696e646f775f73706c6974746572732e6970796e62&logged_in=false&nwo=alan-turing-institute%2Fsktime&path=examples%2Fwindow_splitters.ipynb&platform=android&repository_id=156401841&repository_type=Repository&version=98).

TEST\_SIZE = int(0.45\*y.size)  
  
y\_train, y\_test = temporal\_train\_test\_split(y, test\_size=TEST\_SIZE)  
  
print(f'Check splitted data size: Train: {y\_train.shape[0]}, Test: {y\_test.shape[0]}')  
  
sktime.utils.plotting.plot\_series(y\_train, y\_test, labels=["y\_train", "y\_test"]);

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



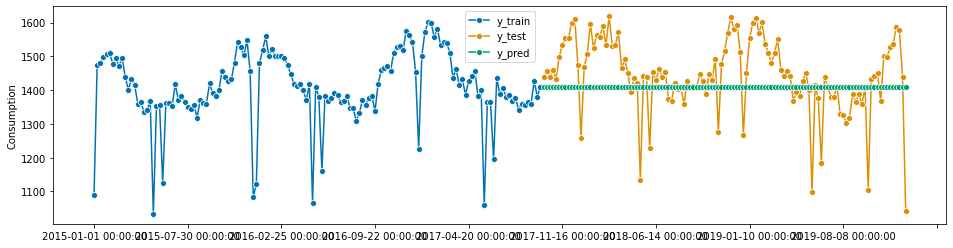
Помимо разделения данных на тренированные и тестовые предсказание в sktime может потребовать т.н. forecasting horizon - то есть непосредственного указания числа предсказываемых значений. Такой горизонт предсказания можно задавать по разному, при помощи массива или объекта класса ForecastingHorizon. Последний способ предоставляет более развитый инструментарий.

Для начала давайте попробуем выполнить наивное предсказание. То есть каждое следующее предсказанное значение будет лишь копией предыдущего, начиная с последнего значение тренировочной выборки.

Мы полагаем, что не точность данного предсказания очевидна, однако, давайте все же проверим ошибку. Для этого воспользуемся симметричной средней процентной ошибкой (symmetry Mean Average Percentage Error, sMAPE)

# ГОРИЗОНТ  
#fh = np.arange(y\_test.size) + 1  
fh = ForecastingHorizon(y\_test.index, is\_relative=False)  
  
# ПРЕДСКАЗАТЕЛЬ  
forecaster = NaiveForecaster(strategy="last")  
forecaster.fit(y\_train)  
  
# ПРЕДСКАЗАНИЕ  
y\_pred = forecaster.predict(fh)  
plot\_series(y\_train, y\_test, y\_pred, labels=["y\_train", "y\_test", "y\_pred"])  
  
# ОШИБКА  
print(f'sMAPE = {smape(y\_pred.values, y\_test.values):.3f}')

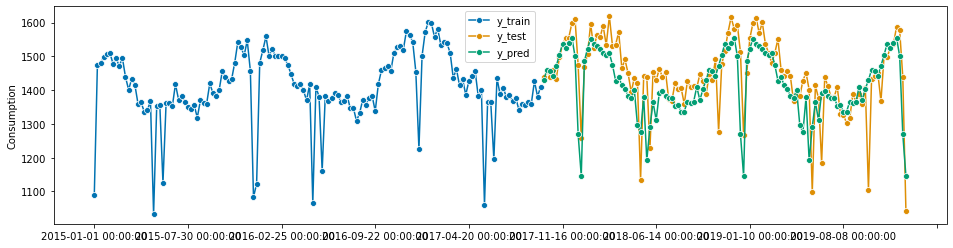
sMAPE = 0.061



Также давайте попробуем оценить точность наивного сезонного предсказания.

SEASON = 52  
  
forecaster = NaiveForecaster(strategy="mean", sp=SEASON)  
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(f'sMAPE = {smape(y\_pred.values, y\_test.values):.3f}')

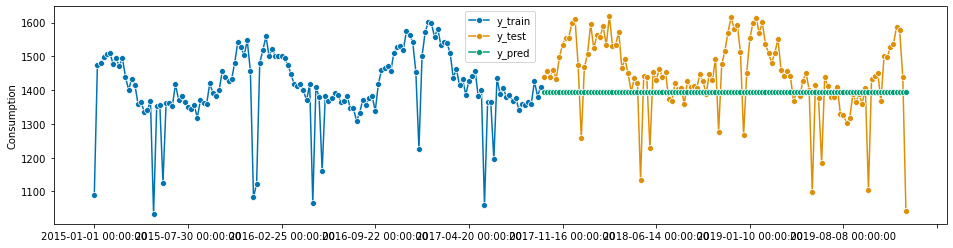
sMAPE = 0.043



Более продвинутым вариантом предсказателей является Семейство методов на основе экспоненциального сглаживания (Exponential Smoothing, Holt-Winter). Давайте посмотрим на их работу. Для начала оценим точность наиболее простого варианта Simple Exponential Smoothing.

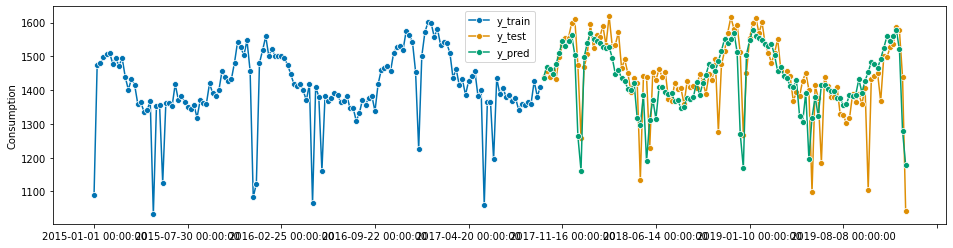
forecaster = ExponentialSmoothing()  
forecaster.fit(y\_train)  
y\_pred = forecaster.predict(fh)  
plot\_series(y\_train, y\_test, y\_pred, labels=["y\_train", "y\_test", "y\_pred"])  
  
print(f'sMAPE = {smape(y\_pred.values, y\_test.values):.3f}')

sMAPE = 0.066



SEASON = 52  
  
# МЕТОДЫ  
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", use\_boxcox =True, sp=SEASON)  
holt\_winter\_mul\_boxcox = ExponentialSmoothing(trend="mul", seasonal="additive", use\_boxcox =True, sp=SEASON)  
holt\_winter\_sadd\_boxcox = ExponentialSmoothing(trend="add", seasonal="mul", use\_boxcox =True, sp=SEASON)  
holt\_winter\_smul\_boxcox = ExponentialSmoothing(trend="mul", seasonal="mul", use\_boxcox =True, sp=SEASON)  
  
  
  
forecaster = 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(f'sMAPE = {smape(y\_pred.values, y\_test.values):.3f}')

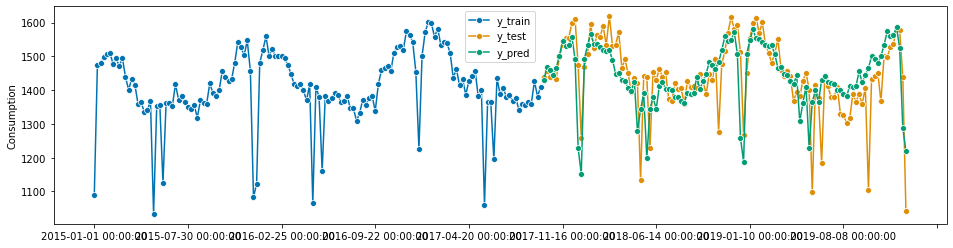
sMAPE = 0.041



Точность в предыдущих случаях оставляла желать лучшего. Однако, теперь давайте посмотрим на работу более сложных методов экспоненциального сглаживания.

SEASON = 52  
  
# МЕТОДЫ  
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", use\_boxcox =True, sp=SEASON)  
holt\_winter\_mul\_boxcox = ExponentialSmoothing(trend="mul", seasonal="additive", use\_boxcox =True, sp=SEASON)  
holt\_winter\_sadd\_boxcox = ExponentialSmoothing(trend="add", seasonal="mul", use\_boxcox =True, sp=SEASON)  
holt\_winter\_smul\_boxcox = ExponentialSmoothing(trend="mul", seasonal="mul", use\_boxcox =True, sp=SEASON)  
  
# ПРЕДСКАЗАТЕЛЬ  
forecaster = EnsembleForecaster(  
 [  
# ("ses", ses),  
# ("holt", holt),  
# ("damped", damped\_holt),  
 ("holt-winter",holt\_winter),  
# ("holt-winter, additive trend, box-cox", holt\_winter\_add\_boxcox),  
 ("holt-winter, multiplicative trend, box-cox", holt\_winter\_mul\_boxcox),  
# ("holt-winter, multiplicative season, box-cox", holt\_winter\_sadd\_boxcox),  
# ("holt-winter, multiplicative both, box-cox", holt\_winter\_smul\_boxcox)   
 ]  
)  
forecaster.fit(y\_train)  
  
# ПРЕДСКАЗАНИЕ  
y\_pred = forecaster.predict(fh)  
plot\_series(y\_train, y\_test, y\_pred, labels=["y\_train", "y\_test", "y\_pred"])  
  
# РЕЗУЛЬТАТ  
print(f'sMAPE = {smape(y\_pred.values, y\_test.values):.3f}')

sMAPE = 0.043



В общем случае пакет sktime позволяет использовать автоматизированный подбор параметров модели экспоненциального сглаживания. Для этого может быть использован пакет AutoETS. Модель, реализуемая данным пакетом также называется Error-Trend-Season (ETS). В случае необходимости исследователь может вручную задать параметры модели, в формате ETS(X,X,X)s, где X может быть N-None, A-additive, M-multiplicative, Ad-additive dumped, s-период сезона или None.

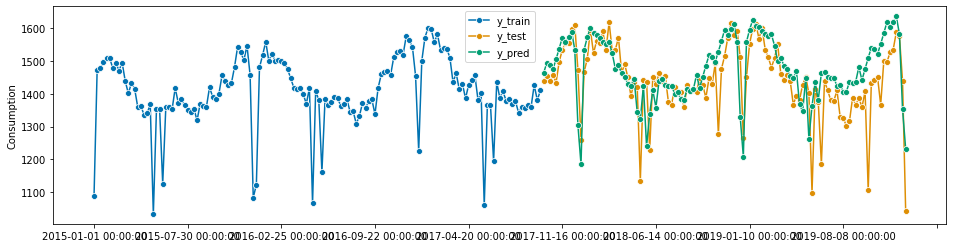
Частными примерами модели ETS являются:

* Simple Exponential smoothing ETS(A,N,N).
* Triple Exponential smoothing ETS(A,A,A).
* и многие другие.

from sktime.forecasting.ets import AutoETS  
  
forecaster = AutoETS(auto=True, sp=SEASON, 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(f'sMAPE = {smape(y\_pred.values, y\_test.values):.3f}')

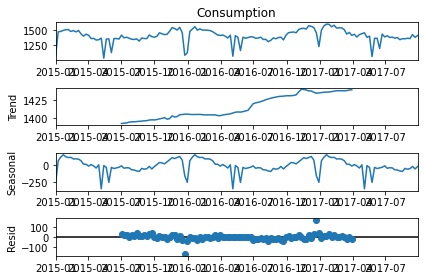
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/base/model.py:604: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle\_retvals  
 warnings.warn("Maximum Likelihood optimization failed to "  
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/base/model.py:604: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle\_retvals  
 warnings.warn("Maximum Likelihood optimization failed to "

sMAPE = 0.045



Другим методом оценки данных с вилянием сезонности и тренда является использование предварительной декомпозиции временного ряда. Одним из наиболее простых методов декомпозиции является разделение на тренд, сезонность и остаток. Давайте посмотрим на то, как будут выглядеть результаты такого разложения.

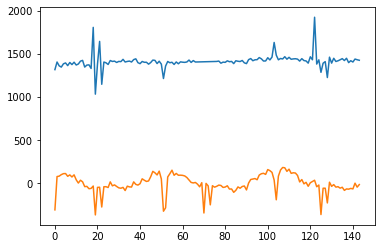
from statsmodels.tsa.seasonal import seasonal\_decompose  
result = seasonal\_decompose(y\_train, model='additive', period = 52)  
result.plot();



В рамках пакета SKTime разложение можно выполнить при помощи объектов класса Deseasonalizer и Detrender.

deseason = Deseasonalizer(model="multiplicative", sp=52)  
detrend = Detrender(forecaster=PolynomialTrendForecaster(degree=1))  
plt.plot(deseason.fit\_transform(y\_train.values))  
plt.plot(detrend.fit\_transform(y\_train.values))

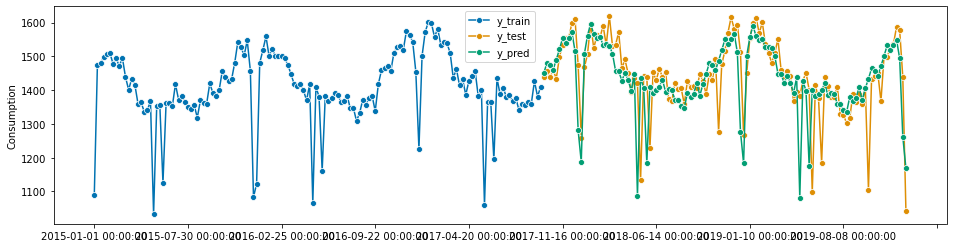
[<matplotlib.lines.Line2D at 0x7fb085895550>]



Посмотрим на работу наиболее простого экспоненциального сглаживания с использованием разложения. Для этого будем использовать контейнер TransformedTargetForecaster - который является некоторым аналогом пайплайна.

SEASON = 52  
  
fh = ForecastingHorizon(y\_test.index, is\_relative=False)  
  
# объек предсказатель  
ses = ExponentialSmoothing()  
  
# пайплайн  
forecaster = TransformedTargetForecaster(  
 steps=[  
 ("deseasonalize", Deseasonalizer(model="multiplicative", sp=SEASON)),  
 ("detrend", Detrender(forecaster=PolynomialTrendForecaster(degree=2))),  
 ("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"])  
  
print(f'sMAPE = {smape(y\_pred.values, y\_test.values):.3f}')

sMAPE = 0.042



# Использование пакета Scikit-learn для предсказаний

Изученные прежде методы предсказания из SKTime скорее следовало бы считать некоторыми предварительными - базовыми оценками. В последующем мы рассмотрим более развитый инструментарий оценок, специализированных для временных рядов. Однако, прежде давайте посмотрим на то, как тут могли быть быть использованы традиционные подходы из [Scikit-learn (sklearn)](https://scikit-learn.org/stable/).

Первая проблема, с которой можно столкнуться при использовании традиционных подходов для анализа временных рядов - это представление выборок данных. Выше в этом уроке мы уже рассматривали термин горизонт предсказания. Такой горизонт можно описать моделью seq2seq когда одна последовательность данных подается на вход модели и на выходе вычисляется новая - предсказанная последовательность. Однако, такой путь не совсем удачная идея для Scikit-learn. Отметим, что в пакете Scikit-learn принято задавать данные в "табличном" виде, то есть вектор входных данных - одно или несколько выходных. Для того, чтобы представить временной ряд в таком виде есть несколько техник. Мы воспользуемся одной из них recursive reduction.

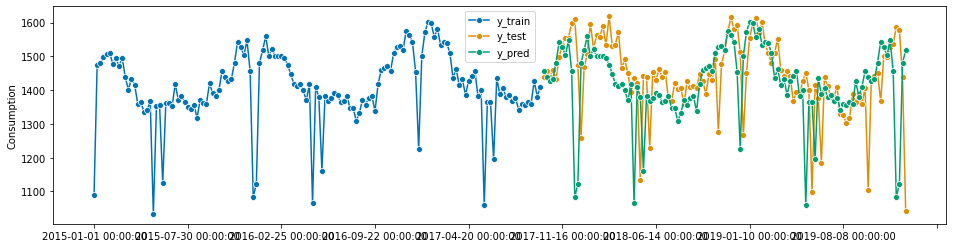
Идея техники recursive reduction достаточно проста, принцип работы показана ниже. Принцип заключается в движении по меткам ряда окна, размером с тренировочную выборку, таким образом, что каждый следующий семпл ответ связан с положением окна. Данная техника может быть реализована в рамках функции make\_reduction. Отметим, что в общем случае [данная функция имеет и другие режимы редукции](https://www.sktime.org/en/stable/api_reference/auto_generated/sktime.forecasting.compose.make_reduction.html).

Также важно заметить, что рекурсивная редукция может быть реализована только в тренировочном режиме. В тестовом режиме такая редукция сводится лишь к 1-step regression или к регрессии с горизонтом при помощи подстановки предсказанных значений.

В качестве примера рассмотрим регрессию ближайших соседей.

from sklearn.neighbors import KNeighborsRegressor  
  
REGRESSION\_WINDOW = 15  
  
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(f'sMAPE = {smape(y\_pred.values, y\_test.values):.3f}')

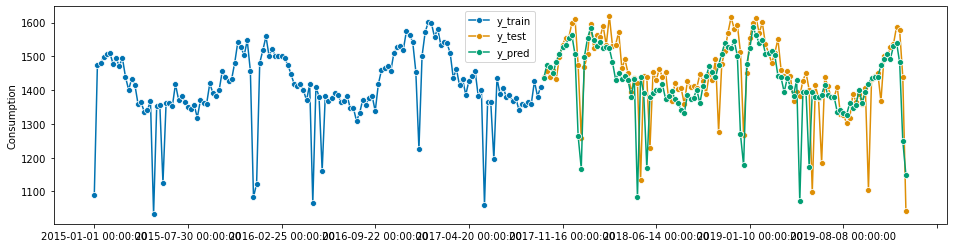
sMAPE = 0.065



Попробуем также провести регрессию с использованием пайплайна.

regressor = KNeighborsRegressor(n\_neighbors=3)  
  
forecaster = TransformedTargetForecaster([  
 ("deseasonalize", Deseasonalizer(model="multiplicative", sp=52)),  
 ("detrend", Detrender(forecaster=PolynomialTrendForecaster(degree=2))),  
 ( "forecast", make\_reduction(  
 regressor,  
 scitype="tabular-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(f'sMAPE = {smape(y\_pred.values, y\_test.values):.3f}')

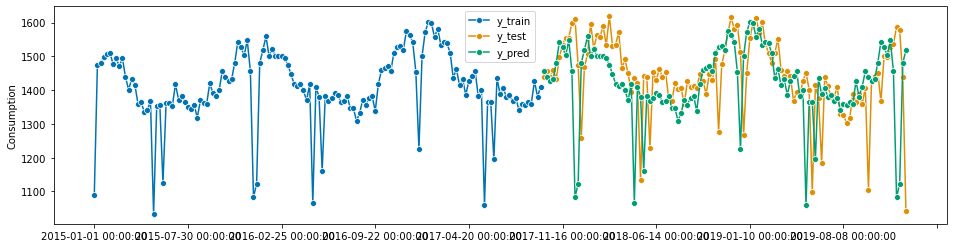
sMAPE = 0.044



В некоторых случаях также можно использовать методы поиска лучших гиперпараметров. Пример ниже позволяет найти наилучшие гиперпараметры как для оценки из sklearn, так и для параметра из SKTime.

from sklearn.neighbors import KNeighborsRegressor  
  
from sktime.forecasting.compose import make\_reduction  
from sktime.forecasting.model\_selection import (  
 ForecastingGridSearchCV,  
 SlidingWindowSplitter,  
)  
  
param\_grid = {"window\_length": [10, 12, 15], "estimator\_\_n\_neighbors": [1,2,3,4]}  
  
regressor = KNeighborsRegressor()  
  
forecaster = make\_reduction(  
 regressor, scitype="tabular-regressor", strategy="recursive"  
)  
  
# Предварительные данные  
cv = SlidingWindowSplitter(initial\_window=int(len(y\_train) \* 0.7), window\_length=30)  
gscv = ForecastingGridSearchCV(forecaster, cv=cv, param\_grid=param\_grid)  
  
gscv.fit(pd.DataFrame(y\_train))  
  
y\_pred = gscv.predict(fh)  
  
plot\_series(y\_train, y\_test, y\_pred, labels=["y\_train", "y\_test", "y\_pred"])  
print(f'sMAPE = {smape(y\_pred.values, y\_test.values):.3f}')  
  
print('Best paramters',gscv.best\_params\_)

sMAPE = 0.065  
Best paramters {'estimator\_\_n\_neighbors': 1, 'window\_length': 15}

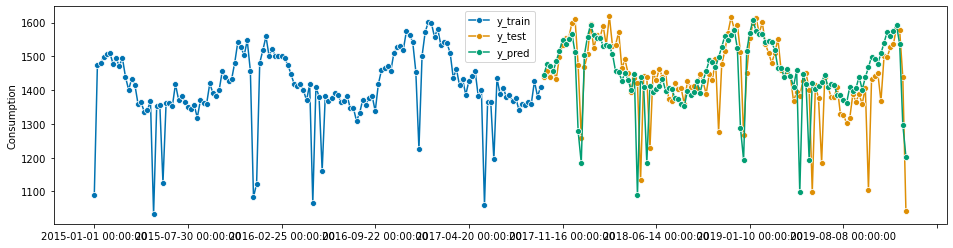


В конце также давайте посмотрим несколько современных методов регрессии, разработанных для временных рядов и представленных в рамках пакета SKTime.

ThetaForecaster - метод предсказания временных рядов на основе экспоненциального сглаживания для модели с дрейфом тренда (случайный тренд).

forecaster = ThetaForecaster(sp=SEASON)  
forecaster.fit(y\_train, fh=fh)  
y\_pred = forecaster.predict(fh)  
  
plot\_series(y\_train, y\_test, y\_pred, labels=["y\_train", "y\_test", "y\_pred"])  
print(f'sMAPE = {smape(y\_pred.values, y\_test.values):.3f}')

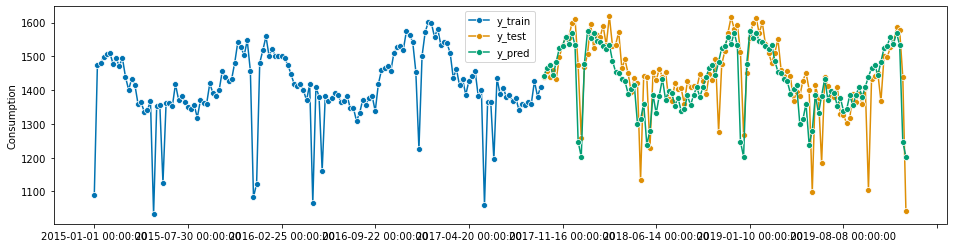
sMAPE = 0.044



TBATS - метод на основе простого преобразования временного ряда (BOX-COX) и затем использования отдельно предсказания для тренда, сезонной части и сложной модели ряда (ARMA). Метод подходит для временных рядов с несколькими сезонными составляющими.

forecaster = TBATS(  
 use\_box\_cox=True,  
 use\_trend=True,  
 use\_damped\_trend=True,  
 sp=SEASON,  
 use\_arma\_errors=True,  
 n\_jobs=-1)  
  
forecaster.fit(y\_train)  
  
fh = ForecastingHorizon(y\_test.index, is\_relative=False)  
y\_pred = forecaster.predict(fh)  
  
plot\_series(y\_train, y\_test, y\_pred, labels=["y\_train", "y\_test", "y\_pred"])  
print(f'sMAPE = {smape(y\_pred.values, y\_test.values):.3f}')

sMAPE = 0.040

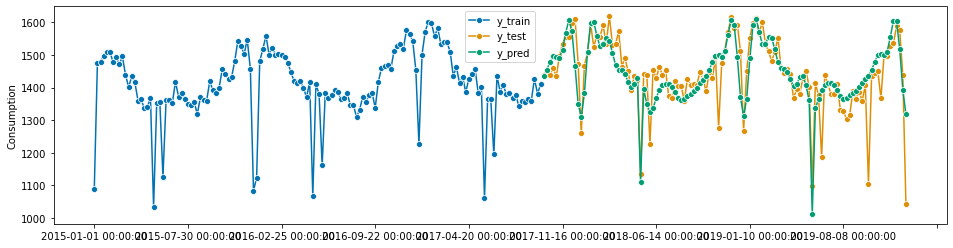


Prophet - метод предсказания временных рядов на основе т.н. обобщенной регрессии для модели бизнес процессов, которая включает тренд с точками перегиба и насыщением, несколько компонент сезонности и редкие, но регулярные события (типа выходных дней).

forecaster = Prophet(  
 seasonality\_mode='additive',  
 n\_changepoints=int(len(y\_train) / 4),  
 add\_country\_holidays={'country\_name': 'Germany'},  
 yearly\_seasonality=True)  
  
forecaster.fit(y\_train)  
fh = ForecastingHorizon(y\_test.index, is\_relative=False)  
y\_pred = forecaster.predict(fh)  
plot\_series(y\_train, y\_test, y\_pred, labels=["y\_train", "y\_test", "y\_pred"])  
print(f'sMAPE = {smape(y\_pred.values, y\_test.values):.3f}')

23:40:25 - cmdstanpy - INFO - Chain [1] start processing  
23:40:25 - cmdstanpy - INFO - Chain [1] done processing

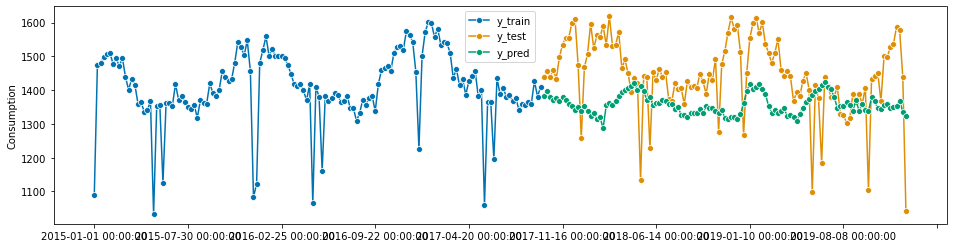
sMAPE = 0.033



Также в некоторых случаях могут быть рассмотрены предсказатели на основе выделения признаков.

regressor = make\_pipeline(  
 TSFreshFeatureExtractor(show\_warnings=False, disable\_progressbar=True),  
 RandomForestRegressor(),  
)  
  
forecaster = make\_reduction(  
 regressor, scitype="time-series-regressor", window\_length=15  
)  
forecaster.fit(y\_train)  
  
fh = ForecastingHorizon(y\_test.index, is\_relative=False)  
y\_pred = forecaster.predict(fh)  
  
plot\_series(y\_train, y\_test, y\_pred, labels=["y\_train", "y\_test", "y\_pred"])  
print(f'sMAPE = {smape(y\_pred.values, y\_test.values):.3f}')

sMAPE = 0.084



**Вопросы:**

1 Для разделения выборки на тренировочную и тестовую мы использовали:

А Случайное разделение в случайные промежутки времени

Б Последовательно создали несколько промежутков, где тест следует за трейном

В Последовательно выбрали тест после всего трейна

**Ответ**: Последовательно выбрали тест после всего трейна

1. Выберите верное выражение для метрики SMAE:

Ответ: b

Метод предсказания, где все последующие значения повторяют аналогичные с заданным шагом сезонности:

NaiveForecaster(strategy="mean", sp=SEASON)

NaiveForecaster(strategy="last")

ExponentialSmoothing()

ответ: NaiveForecaster(strategy="mean", sp=SEASON)

выберите правильное заданный метод Холта-Винтера:

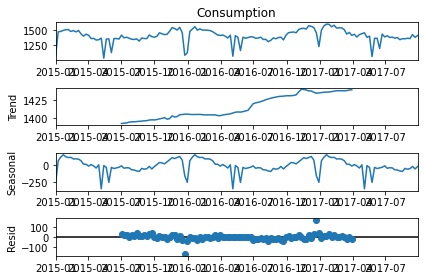
ExponentialSmoothing(sp=SEASON)

ExponentialSmoothing(trend="add", sp=SEASON)

ExponentialSmoothing(trend="add", seasonal="additive", sp=SEASON)

Ответ: ExponentialSmoothing(trend="add", seasonal="additive", sp=SEASON)

Мы провели декомпозицию ряда, выберите где остаток:



Ответ: последний график

Когда мы хотели использоваться предсказания из пакета sklearn, мы использовали функцию make\_redution, зачем?

* Мы хотели свести проблемы горизонта предсказания и проблеме табличной регрессии
* Мы хотели свести проблему предсказания на длинном интервале к предсказанию step-by-step
* Мы хотели преобразовать данные в формат sklearn

Ответ 1

Выберите метод предсказания на основе обобщенной регрессионной модели для бизнесс-процессов:

* Prophet
* AutoETS
* TBATS
* ThetaForecaster
* SES

Ответ Prophet

Выберите метод предсказания на основе модели, скользящего среднего для ряда со случайным трендом:

* Prophet
* AutoETS
* TBATS
* ThetaForecaster
* SES

Ответ ThetaForecaster