[Временные ряды 2](#_Toc132876122)

[Структура курса 2](#_Toc132876123)

[Ваши цели 2](#_Toc132876124)

[Анализ временных рядов 2](#_Toc132876125)

[Временные ряды 2](#_Toc132876126)

[Задачи 3](#_Toc132876127)

[Ресемплирование 5](#_Toc132876128)

[Задачи 5](#_Toc132876129)

[Скользящее среднее 6](#_Toc132876130)

[Задача 7](#_Toc132876131)

[Тренды и сезонность 8](#_Toc132876132)

[Задачи 8](#_Toc132876133)

[Стационарные ряды 9](#_Toc132876134)

[Разности временного ряда 13](#_Toc132876135)

[Задача 14](#_Toc132876136)

[Задача прогнозирования 15](#_Toc132876137)

[Задача 16](#_Toc132876138)

[Качество прогноза 16](#_Toc132876139)

[Задачи 17](#_Toc132876140)

# Временные ряды

Вы научитесь решать задачу регрессии для временных рядов.

Узнаете, как из временных рядов получить табличные данные.

#### Структура курса

Временные ряды описывают, как меняются со временем, например, объём потребления электроэнергии или количество заказов в сервисе такси. Вы узнаете, как анализировать ряды, искать в них тренды и сезонность. Затем научитесь создавать признаки и делать на них прогнозы.

В конце курса вас ждёт проект по прогнозированию временных рядов.

#### Ваши цели

* Разобраться в трендах и сезонности временных рядов.
* Научиться создавать признаки из временных рядов.
* Обучить модель регрессии на полученных признаках.

Это короткий и несложный курс: теории будет немного и выполнение проекта займёт мало времени.

## Анализ временных рядов

### Временные ряды

Научимся загружать и строить графики временных рядов.

**Временные ряды** (англ. *time series*) — это последовательности чисел на оси времени. Например:

* цена акций компании в начале торгов;
* объём продаж интернет-магазина по дням;
* количество игроков в онлайне по часам.

Интервал между значениями ряда постоянный: скажем, день, час или 10 минут.

Повторим, как работать с датами и временем в Pandas. Перед вами данные американской энергетической компании *PJM East*. В файле energy\_consumption.csv указано, сколько электроэнергии потребляют её клиенты.

data = pd.read\_csv('/datasets/energy\_consumption.csv')

print(data.head())

Datetime PJME\_MW

0 2002-12-31 01:00:00 26498.0

1 2002-12-31 02:00:00 25147.0

2 2002-12-31 03:00:00 24574.0

3 2002-12-31 04:00:00 24393.0

4 2002-12-31 05:00:00 24860.0

Рассмотрим столбцы таблицы:

* *Datetime* (англ. «дата-время») — значение даты и времени на временной оси. Переменная называется так же, как и тип данных *datetime*. В этой таблице интервал равен одному часу.
* *PJME\_MW* (название компании и *MW*; англ. *megawatt*, «мегаватт») — расход электроэнергии за час.

Нужно привести данные к удобному формату, для их анализа выделить временной промежуток, а затем построить график временного ряда. В этом поможет документация Pandas.



#### Задачи

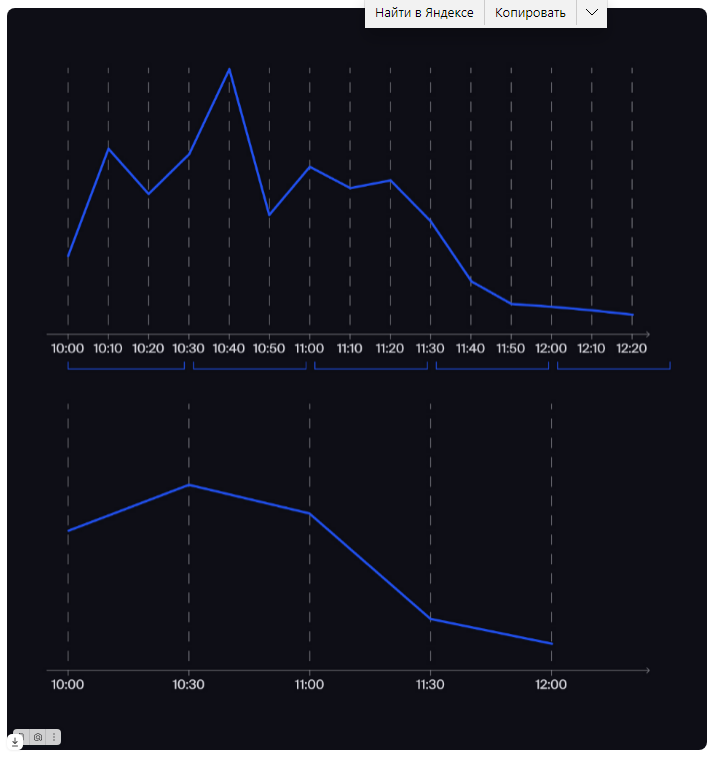
|  |  |
| --- | --- |
| 1.  Измените тип данных *Datetime* с *object* на *datetime64*. Но прежде запустите код и просмотрите общую информацию о данных.  В документации Pandas выберите любой способ преобразования данных. Формат вывода даты указывать не нужно: библиотека определит его самостоятельно.  Напечатайте на экране информацию о таблице (уже в прекоде). | import pandas as pd  data = pd.read\_csv('/datasets/energy\_consumption.csv')  data['Datetime'] = pd.to\_datetime(data['Datetime'])  print(data.info()) |
| 2.  Установите индекс таблицы равным столбцу *Datetime*. В документации Pandas выберите любой способ установки индекса.  Напечатайте на экране информацию о таблице (уже в прекоде). | import pandas as pd  data = pd.read\_csv('/datasets/energy\_consumption.csv', parse\_dates=[0])  data = data.set\_index('Datetime')  print(data.info()) |
| 3.  Чтобы проверить, в хронологическом ли порядке расположены даты и время, посмотрите атрибут индекса таблицы **is\_monotonic** (англ. «монотонный»)*.* Если порядок соблюдён, атрибут вернёт *True*, если нет *—* *False*.  Отсортируйте индекс таблицы. Метод найдите в документации.  Напечатайте на экране значение атрибута *is\_monotonic* (уже в прекоде). Затем вызовом функции *info()* выведите на экран общую информацию о таблице. | import pandas as pd  data = pd.read\_csv('/datasets/energy\_consumption.csv', index\_col=[0], parse\_dates=[0])  data.sort\_index(inplace=True)  print(data.index.is\_monotonic)  print(data.info()) |
| 4.  Из временного ряда выделите данные с января по июнь 2018 года.  Даты во временных рядах можно указывать в срезах. В прекоде выбраны значения с 2016 по 2017 год включительно.  Напечатайте на экране информацию о таблице (уже в прекоде). | import pandas as pd  data = pd.read\_csv('/datasets/energy\_consumption.csv', index\_col=[0], parse\_dates=[0])  data.sort\_index(inplace=True)  #data = data['2016':'2017']  data = data['2018-01':'2018-06']  print(data.info()) |
| 5.  Постройте график временного ряда. | import pandas as pd  data = pd.read\_csv('/datasets/energy\_consumption.csv', index\_col=[0], parse\_dates=[0])  data.sort\_index(inplace=True)  data = data['2018-01':'2018-06']  data.plot() |

### Ресемплирование

Как поменять интервал временного ряда? Выполним его ресемплирование.

**Ресемплирование**, или **ресемплинг** (англ. *resample*, «повторная выборка»), — это изменение интервала со значениями ряда. Его выполняют в два этапа:

1. Выбирают новую длину интервала. Причём значения из текущего интервала группируются. Допустим, интервал равен 10 минутам. Возьмём новый интервал — 30 минут. В него попадут значения из трёх 10-минутных интервалов.
2. В каждой группе вычисляется агрегированное значение ряда. Это может быть медиана, среднее, максимум или минимум.



Чтобы поменять интервал и сгруппировать значения, вызовем функцию *resample()*. В аргументе укажем новый интервал. Например:

Скопировать кодPYTHON

*# 1H англ. hour, 1 час*

data.resample('1H')

*# 2W англ. week, 2 недели*

data.resample('2W')

Функция *resample()* похожа на *groupby()*. После группировки вызовем функции *mean()* и *max()* для агрегации значений:

Скопировать кодPYTHON

*# среднее по каждому часу*

data.resample('1H').mean()

*# максимум по каждым двум неделям*

data.resample('2W').max()

#### Задачи

|  |  |
| --- | --- |
| 1.  Постройте график среднего потребления электроэнергии по годам. | import pandas as pd  data = pd.read\_csv('/datasets/energy\_consumption.csv', index\_col=[0], parse\_dates=[0])  data.sort\_index(inplace=True)  data = data.resample('1Y').mean()  data.plot() |
| 2.  Постройте график энергопотребления с января по июнь 2018 года. Выберите интервал в один день, по каждому — вычислите суммарное энергопотребление. | import pandas as pd  data = pd.read\_csv('/datasets/energy\_consumption.csv', index\_col=[0], parse\_dates=[0])  data.sort\_index(inplace=True)  data = data['2018-01':'2018-06'].resample('1D').sum()  data.plot() |
|  |  |

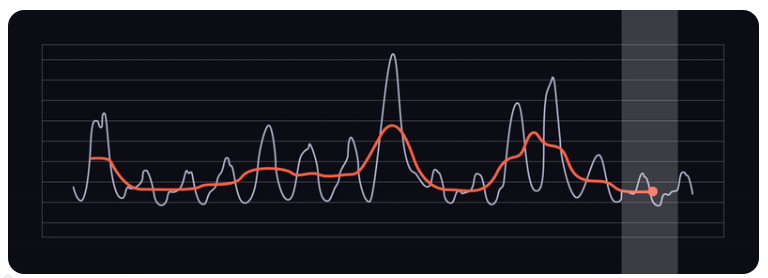
### Скользящее среднее

Чтобы во временном ряду снизить колебания, применим скользящее среднее.

**Скользящее среднее** (англ. rolling mean), или **движущееся среднее** (англ. moving average), — метод сглаживания временных рядов. Его суть заключается в поиске значений, которые меньше всего подвержены колебаниям, то есть средних арифметических.

Метод работает так: экспериментально подбирается **размер окна** (англ. window size) — интервал, в котором выполняют усреднение. Чем интервал больше, тем сильнее сглаживание. Затем окно начинает «скользить» почти от начала ряда к его концу, в каждой точке вычисляя среднее значение ряда и тем самым сглаживая его.

В скользящем среднем окна «наслаиваются» друг на друга и не могут выходить за пределы ряда. Поэтому средних будет чуть меньше, чем исходных значений ряда.



В Pandas скользящее среднее вычисляют в два этапа:

1. Вызовом функции *rolling()* создают скользящее окно. В аргументе указывают его размер:

Скопировать кодPYTHON

*# размер окна 7*

data.rolling(7)

1. Для агрегации значений вызывают функцию *mean()*:

Скопировать кодPYTHON

*# скользящее среднее с окном размером 7*

data.rolling(7).mean()

#### Задача

Добавьте в столбец 'rolling\_mean' скользящее среднее с размером окна, равным 10. Выведите на экран графики энергопотребления с января по июнь 2018 года и скользящего среднего (уже в прекоде).

import pandas as pd

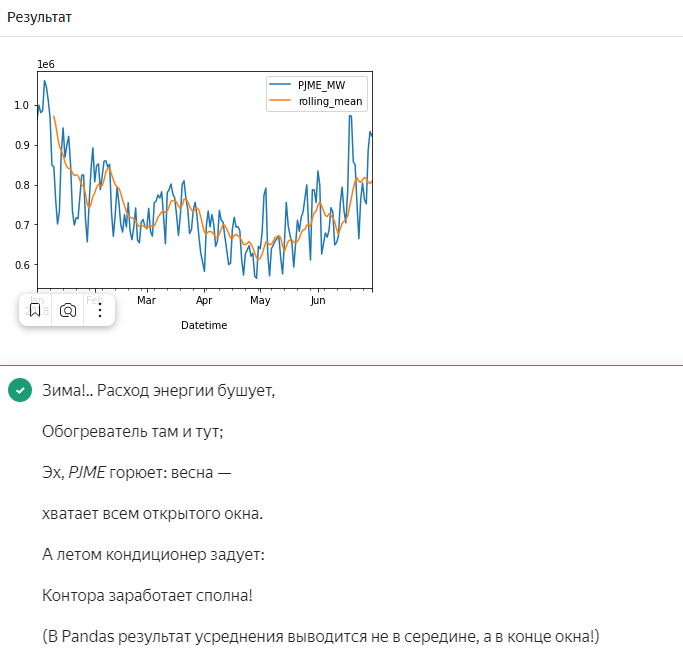
data = pd.read\_csv('/datasets/energy\_consumption.csv', index\_col=[0], parse\_dates=[0])

data.sort\_index(inplace=True)

data = data['2018-01':'2018-06'].resample('1D').sum()

data['rolling\_mean'] = data.rolling(10).mean()

data.plot()



### Тренды и сезонность

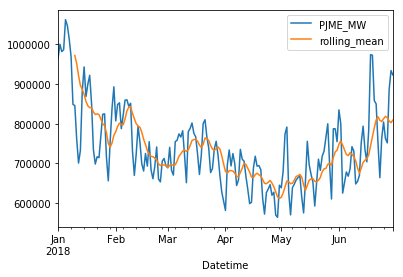
Чтобы лучше понимать временной ряд, разберём тренды и сезонность.

**Тренд** (англ. *trend*) — плавное изменение среднего значения ряда без повторяющихся закономерностей. Например, ежегодное увеличение объёма продаж авиабилетов.

**Сезонность** (англ. *seasonality*) — циклично повторяющиеся закономерности во временном ряду. Допустим, рост продаж авиабилетов летом.

Тренды и сезонность зависят от масштаба данных. Нельзя увидеть закономерности, повторяющиеся каждое лето, если есть данные только за год.

Посмотрим на график *rolling\_mean*. Увеличение энергопотребления зимой и летом — это тренд.



Если эти данные анализировать в масштабе нескольких лет, рост энергопотребления зимой и летом — это уже сезонные изменения.

В модуле **tsa.seasonal** (от англ. *time series analysis*, «анализ временных рядов») библиотеки **statsmodels** (англ. «статистические модели») есть функция **seasonal\_decompose()** (англ. «разбить на части»). Она раскладывает временной ряд на три составляющие: тренд, сезонность и **остаток** (англ. *residuals*). Это компонента, которая не объясняется трендом и сезонностью, это шум.

Скопировать кодPYTHON

from statsmodels.tsa.seasonal import seasonal\_decompose

decomposed = seasonal\_decompose(data)

Функция *seasonal\_decompose()* принимает временной ряд, а возвращает объект структуры **DecomposeResult** (англ. «результат разделения»). В нём есть нужные атрибуты:

* *decomposed.trend* — тренд;
* *decomposed.seasonal* — сезонная составляющая;
* *decomposed.resid* — остаток декомпозиции.

#### Задачи

|  |  |  |
| --- | --- | --- |
| 1.  Разложите временной ряд на тренд и сезонную компоненту. Допишите код вывода графиков этих составляющих ряда. | | import pandas as pd  from statsmodels.tsa.seasonal import seasonal\_decompose  import matplotlib.pyplot as plt  data = pd.read\_csv('/datasets/energy\_consumption.csv', index\_col=[0], parse\_dates=[0])  data.sort\_index(inplace=True)  data = data['2018-01':'2018-06'].resample('1D').sum()  decomposed = seasonal\_decompose(data)  plt.figure(figsize=(6, 8))  plt.subplot(311)  # Чтобы график корректно отобразился, указываем его  # оси ax, равными plt.gca() (англ. get current axis,  # получить текущие оси)  decomposed.trend.plot(ax=plt.gca())  plt.title('Trend')  plt.subplot(312)  decomposed.seasonal.plot(ax=plt.gca())  plt.title('Seasonality')  plt.subplot(313)  decomposed.resid.plot(ax=plt.gca())  plt.title('Residuals')  plt.tight\_layout() |
| 2.  Постройте график сезонной составляющей за первые 15 дней января 2018 года. | import pandas as pd  from statsmodels.tsa.seasonal import seasonal\_decompose  import matplotlib.pyplot as plt  data = pd.read\_csv('/datasets/energy\_consumption.csv', index\_col=[0], parse\_dates=[0])  data.sort\_index(inplace=True)  data = data['2018-01':'2018-06'].resample('1D').sum()  #data = data['2018-01-01':'2018-01-15']  decomposed = seasonal\_decompose(data)  decomposed.seasonal['2018-01-01':'2018-01-15'].plot(ax=plt.gca()) | |
|  |  | |

### Стационарные ряды

Прогнозировать данные помогают стационарные ряды. Разберёмся, что это такое.

В задаче потребления электроэнергии найдём **скользящее стандартное отклонение** (англ. rolling standard deviation, rolling std) — стандартное отклонение по скользящему окну:

Скопировать кодPYTHON

import pandas as pd

data = pd.read\_csv('energy\_consumption.csv', index\_col=[0], parse\_dates=[0])

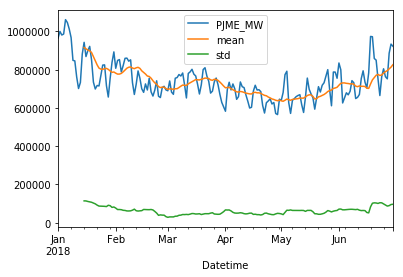
data.sort\_index(inplace=True)

data = data['2018-01':'2018-06'].resample('1D').sum()

data['mean'] = data['PJME\_MW'].rolling(15).mean()

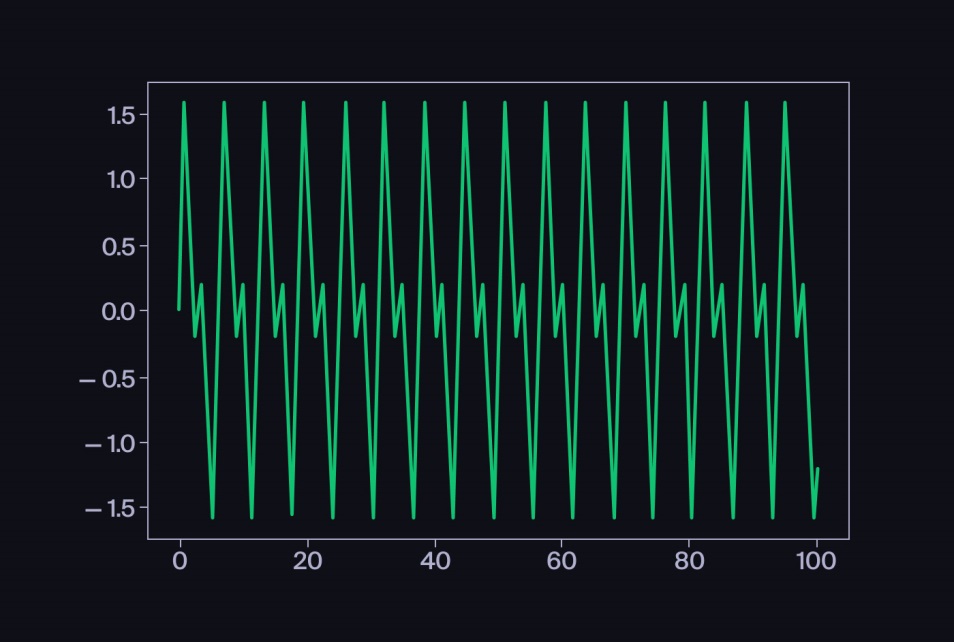
data['std'] = data['PJME\_MW'].rolling(15).std()

data.plot()



В статистике временной ряд описывается **стохастическим процессом** (англ. stochastic process). Это случайная величина, у которой со временем меняется её распределение. У этой величины есть среднее и дисперсия, которые тоже меняются.

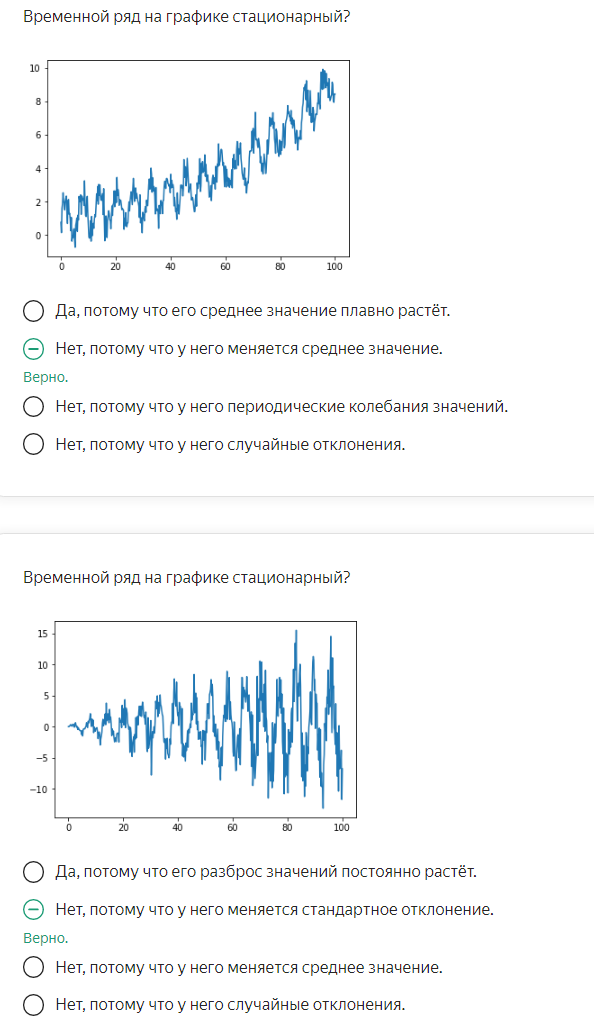
Стохастический процесс **стационарный** (англ. stationary stochastic process), если его распределение со временем не меняется. Например, к такому процессу относятся периодические колебания значений.



Если распределение меняется, то процесс называется **нестационарным** (англ. nonstationary stochastic process).

Узнать распределение временного ряда нельзя. Поэтому **стационарные временные ряды** (англ. stationary time series) — это ряды, у которых среднее и стандартное отклонение не меняются. Когда среднее и стандартное отклонение первого ряда меняется медленнее второго, то первый ряд «более стационарный», чем второй.

**Нестационарные ряды** (англ. nonstationary time series) прогнозировать сложнее: их свойства меняются слишком быстро.





### Разности временного ряда

Чтобы ряд сделать стационарным, найдём разности его значений.

**Разности временного ряда** (англ. time series difference) — это набор разностей между соседними элементами временного ряда, т. е. из каждого значения вычитается предыдущее.

Для поиска разностей временного ряда применяется метод **shift()** (англ. «сдвиг»). Все значения он сдвигает вдоль временной оси на один шаг вперёд:

Скопировать кодPYTHON

data = pd.Series([0.5, 0.7, 2.4, 3.2])

print(data)

print(data.shift())

Скопировать код

0 0.5

1 0.7

2 2.4

3 3.2

dtype: float64

0 NaN

1 0.5

2 0.7

3 2.4

dtype: float64

Последнее значение ряда пропадает: его сдвигать некуда. На месте нулевого — NaN, потому что для него значения нет. Добавим аргумент, чтобы заполнить недостающие значения:

Скопировать кодPYTHON

import pandas as pd

data = pd.Series([0.5, 0.7, 2.4, 3.2])

print(data)

*# англ. заполнить значением*

print(data.shift(fill\_value=0))

Скопировать код

0 0.5

1 0.7

2 2.4

3 3.2

dtype: float64

0 0.0

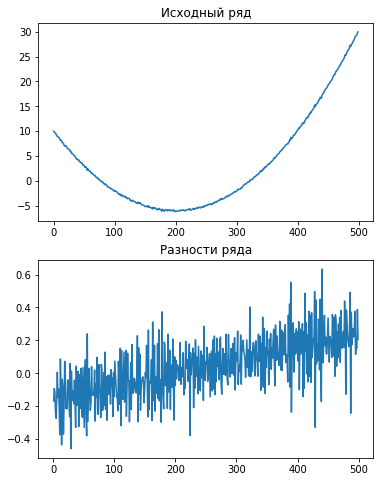
1 0.5

2 0.7

3 2.4

dtype: float64

Разности временного ряда более стационарны, чем сам ряд. Например, нелинейный тренд преобразуется в линейный:



#### Задача

Вычислите разности временного ряда. Пропущенные значения заполнять не нужно.

На графике изобразите скользящее среднее и скользящее стандартное отклонение (уже в прекоде).

Подсказка

Из данных за январь-июнь 2018 нужно вычесть результат вызова метода *shift()*.

import pandas as pd

data = pd.read\_csv('/datasets/energy\_consumption.csv', index\_col=[0], parse\_dates=[0])

data.sort\_index(inplace=True)

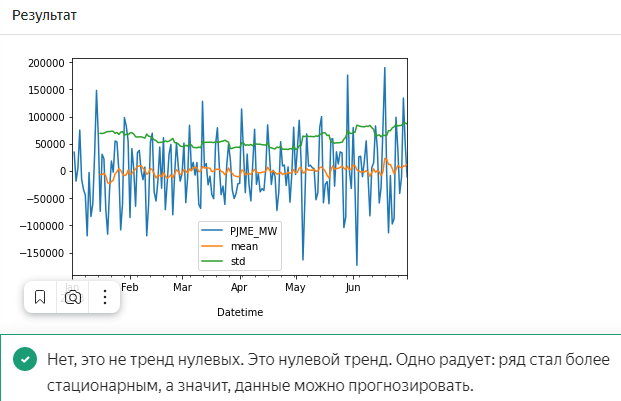
data = data['2018-01':'2018-06'].resample('1D').sum()

data = data - data.shift()

data['mean'] = data['PJME\_MW'].rolling(15).mean()

data['std'] = data['PJME\_MW'].rolling(15).std()

data.plot()



### Задача прогнозирования

Разберём задачу прогнозирования временных рядов.

Задача **прогнозирования временного ряда** состоит в построении модели, которая по историческим данным предскажет будущие значения временного ряда.

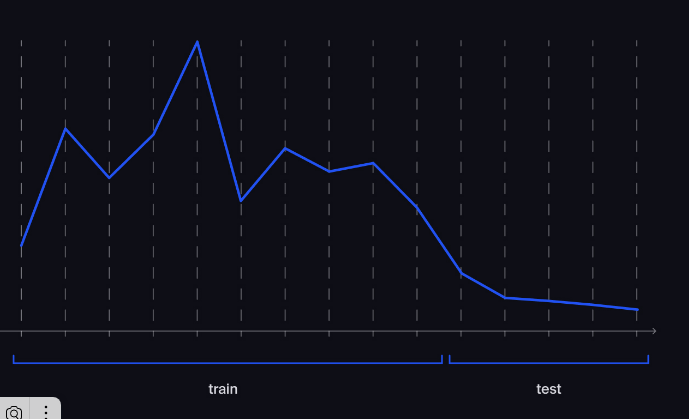
Промежуток времени в будущем, на который строится прогноз, называется **горизонтом прогнозирования** (англ. *forecast horizon*). В задачах этой темы он будет равен одному шагу.

Если значения временного ряда, или функция x(t), где t — время, — это числа, то перед вами задача регрессии для временных рядов; если категории — задача классификации.

Например, к задачам прогнозирования рядов относятся предсказания:

* количества заказов такси на следующий час,
* объёма доставок курьерской службы на следующий день,
* объёма спроса на товары в интернет-магазине на следующей неделе.

По исходным данным создадим обучающую (*train*) и тестовую (*test*) выборки. Перемешивать выборки в задаче прогнозирования временного ряда нельзя. Данные обучающей выборки должны предшествовать данным тестовой. Иначе тестирование модели будет некорректным: модель не должна обучаться на данных из будущего.



Функция train\_test\_split() из модуля sklearn.model\_selection по умолчанию перемешивает данные. Поэтому укажем аргумент shuffle (с англ. «перетасовывать») равным *False*, чтобы разделить данные корректно:

import pandas as pd

from sklearn.model\_selection import train\_test\_split

data = pd.Series([0.1, 0.5, 2.3, 1.2, 1.5])

train, test = train\_test\_split(data, shuffle=False, test\_size=0.2)

print('Обучающая выборка:')

print(train)

print('Тестовая выборка:')

print(test)

Обучающая выборка:

0 0.1

1 0.5

2 2.3

3 1.2

dtype: float64

Тестовая выборка:

4 1.5

dtype: float64

Таким же образом данные делятся на три выборки: обучающую, валидационную и тестовую.

#### Задача

Разбейте датасет о потреблении электроэнергии на обучающую и тестовую выборки в соотношении 4:1. Возьмите данные за доступное время.

Напечатайте на экране минимальные и максимальные значения индексов выборок (уже в прекоде). Они нужны, чтобы убедиться в корректности деления

import pandas as pd

from sklearn.model\_selection import train\_test\_split

data = pd.read\_csv('/datasets/energy\_consumption.csv', index\_col=[0], parse\_dates=[0])

data.sort\_index(inplace=True)

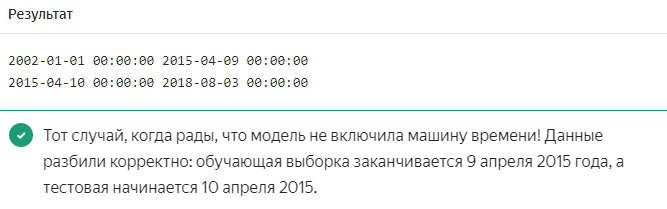
data = data.resample('1D').sum()

train, test = train\_test\_split(data, shuffle=False, test\_size=0.2)

# < напишите код здесь >

print(train.index.min(), train.index.max())

print(test.index.min(), test.index.max())



### Качество прогноза

Научимся измерять качество предсказания временных рядов и проверять модели на адекватность.

Обучим модель с горизонтом прогнозирования в один день. Такие модели пригодятся в автоматизации принятия технических решений. Например, в задаче энергопотребления модель поможет изменять режим работы генераторов автоматически.

Чтобы проверять качество моделей в наших задачах, возьмём метрику *MAE*. Её можно легко интерпретировать.

Спрогнозировать временные ряды без обучения можно двумя способами:

1. Все значения тестовой выборки предсказываются одним и тем же числом (константой). Для метрики *MAE* — это медиана.
2. Новое значение *x(t)* прогнозируется предыдущим значением ряда, то есть *x(t-1)*. Этот способ не зависит от метрики.

#### Задачи

|  |  |  |
| --- | --- | --- |
| 1.  Оцените модель первым способом — прогнозом константой. Дневной объём электропотребления предскажите медианой, сохраните значения в переменной pred\_median и найдите для этого прогноза значение *MAE*.  В прекоде указан средний объём электропотребления, чтобы вы смогли соотнести его со значением метрики *MAE*.  Напечатайте на экране значения среднего объёма электропотребления и метрики *MAE* (уже в прекоде).  **Подсказка**  Допишите код построения прогноза:  Скопировать кодPYTHON  pred\_median = np.ones(test.shape) \* *# < напишите код здесь >* | | import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn. metrics import mean\_absolute\_error  data = pd.read\_csv('/datasets/energy\_consumption.csv', index\_col=[0], parse\_dates=[0])  data.sort\_index(inplace=True)  data = data.resample('1D').sum()  train, test = train\_test\_split(data, shuffle=False, test\_size=0.2)  print("Средний объём электропотребления в день:", test['PJME\_MW'].mean())  pred\_median = np.ones(test.shape) \* train['PJME\_MW'].median()  # < напишите код здесь >  print("MAE:", mean\_absolute\_error(test['PJME\_MW'], pred\_median)) |
| 2.  Оцените модель вторым способом — предыдущим значением ряда. Предскажите дневной объём электропотребления и найдите для этого прогноза значение *MAE*.  В прекоде указан средний объём электропотребления, чтобы вы смогли соотнести его со значением метрики *MAE*.  Напечатайте на экране значения среднего объёма электропотребления и метрики *MAE* (уже в прекоде). | import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import mean\_absolute\_error  data = pd.read\_csv('/datasets/energy\_consumption.csv', index\_col=[0], parse\_dates=[0])  data.sort\_index(inplace=True)  data = data.resample('1D').sum()  train, test = train\_test\_split(data, shuffle=False, test\_size=0.2)  print("Средний объём электропотребления в день:", test['PJME\_MW'].mean())  pred\_previous = test.shift(fill\_value=train.iloc[-1,-1])  print("MAE:",mean\_absolute\_error(test['PJME\_MW'], pred\_previous)) | |

### Создание признаков

Создадим признаки для горизонта прогнозирования в один шаг.

Напишем функцию для создания признаков. Разберём каждый тип признаков:

**1. Календарные признаки** (англ. calendar features)

Во многих данных тренды и сезонность привязаны к конкретной дате. Тип datetime64 в Pandas уже содержит нужную информацию, осталось лишь представить её как отдельные столбцы. Рассмотрим пример:

data = pd.read\_csv('energy\_consumption.csv', index\_col=[0], parse\_dates=[0])

data.sort\_index(inplace=True)

data = data.resample('1D').sum()

# признак, в котором хранится год как число

data['year'] = data.index.year

# признак, в котором хранится день недели как число

data['dayofweek'] = data.index.dayofweek

print(data.head(10))

PJME\_MW year dayofweek

Datetime

2002-01-01 714857.0 2002 1

2002-01-02 822277.0 2002 2

2002-01-03 828285.0 2002 3

2002-01-04 809171.0 2002 4

2002-01-05 729723.0 2002 5

2002-01-06 727766.0 2002 6

2002-01-07 800012.0 2002 0

2002-01-08 824710.0 2002 1

2002-01-09 810628.0 2002 2

2002-01-10 755317.0 2002 3

**2. «Отстающие значения»** (англ. lag features)

Предыдущие значения временного ряда подскажут, будет ли функция x(t) расти или уменьшаться. Получим отстающие значения знакомой функцией shift():

data['lag\_1'] = data['PJME\_MW'].shift(1)

data['lag\_2'] = data['PJME\_MW'].shift(2)

data['lag\_3'] = data['PJME\_MW'].shift(3)

print(data.head(10))

PJME\_MW lag\_1 lag\_2 lag\_3

Datetime

2002-01-01 714857.0 NaN NaN NaN

2002-01-02 822277.0 714857.0 NaN NaN

2002-01-03 828285.0 822277.0 714857.0 NaN

2002-01-04 809171.0 828285.0 822277.0 714857.0

2002-01-05 729723.0 809171.0 828285.0 822277.0

2002-01-06 727766.0 729723.0 809171.0 828285.0

2002-01-07 800012.0 727766.0 729723.0 809171.0

2002-01-08 824710.0 800012.0 727766.0 729723.0

2002-01-09 810628.0 824710.0 800012.0 727766.0

2002-01-10 755317.0 810628.0 824710.0 800012.0

Для первых дат есть не все отстающие значения, поэтому в этих строках стоят NaN.

**3. Скользящее среднее**

Скользящее среднее как признак задаёт общий тренд временного ряда. Повторим, как его вычислять:

data['rolling\_mean'] = data['PJME\_MW'].rolling(5).mean()

print(data.head(10))

PJME\_MW rolling\_mean

Datetime

2002-01-01 714857.0 NaN

2002-01-02 822277.0 NaN

2002-01-03 828285.0 NaN

2002-01-04 809171.0 NaN

2002-01-05 729723.0 780862.6

2002-01-06 727766.0 783444.4

2002-01-07 800012.0 778991.4

2002-01-08 824710.0 778276.4

2002-01-09 810628.0 778567.8

2002-01-10 755317.0 783686.6

Скользящее среднее в моменте t учитывает текущее значение ряда x(t). Это некорректно: целевой признак «убежал» в признаки. Вычисление скользящего среднего не должно включать в себя текущее значение ряда.

#### Задачи

|  |  |  |
| --- | --- | --- |
| 1.  Напишите функцию **make\_features()** (англ. «создать признаки»), чтобы прибавить к таблице четыре новых календарных признака: год, месяц, день и день недели. Имена столбцов должны быть такие: *'year'*, *'month'*, *'day'*, *'dayofweek'*.  Примените функцию к таблице и напечатайте на экране её первые пять строк (уже в прекоде). | | import pandas as pd  import numpy as np  data = pd.read\_csv('/datasets/energy\_consumption.csv', index\_col=[0], parse\_dates=[0])  data.sort\_index(inplace=True)  data = data.resample('1D').sum()  # < напишите код здесь >  def make\_features(data) :  data['year'] = data.index.year  data['month'] = data.index.month  data['day'] = data.index.day  data['dayofweek'] = data.index.dayofweek  make\_features(data)  print(data.head()) |
| 2.  Вычислите отстающие значения. В функцию *make\_features()* добавьте новый аргумент **max\_lag**, который задаст максимальный размер отставания. Новые признаки назовите: *'lag\_1', 'lag\_2' —* и до величины *max\_lag*.  Примените функцию к таблице и напечатайте на экране её первые пять строк (уже в прекоде). | | import pandas as pd  import numpy as np  data = pd.read\_csv('/datasets/energy\_consumption.csv', index\_col=[0], parse\_dates=[0])  data.sort\_index(inplace=True)  data = data.resample('1D').sum()  def make\_features(data, max\_lag):  data['year'] = data.index.year  data['month'] = data.index.month  data['day'] = data.index.day  data['dayofweek'] = data.index.dayofweek  for i in range(1,max\_lag+1):  col = 'lag\_' + str(i)  data[col] = data['PJME\_MW'].shift(i)    # < напишите код здесь >  make\_features(data, 4)  print(data.head()) |
| 3.  Вычислите скользящее среднее и добавьте его как признак '*rolling\_mean*'. В функцию *make\_features()* добавьте новый аргумент *rolling\_mean\_size*, который задаст ширину окна. Текущее значение ряда для расчёта скользящего среднего применять нельзя.  Примените функцию к таблице и напечатайте на экране её первые пять строк (уже в прекоде). | import pandas as pd  import numpy as np  data = pd.read\_csv('/datasets/energy\_consumption.csv', index\_col=[0], parse\_dates=[0])  data.sort\_index(inplace=True)  data = data.resample('1D').sum()  def make\_features(data, max\_lag, rolling\_mean\_size):  data['year'] = data.index.year  data['month'] = data.index.month  data['day'] = data.index.day  data['dayofweek'] = data.index.dayofweek    for lag in range(1, max\_lag + 1):  data['lag\_{}'.format(lag)] = data['PJME\_MW'].shift(lag)  data['rolling\_mean'] = data['PJME\_MW'].shift().rolling(rolling\_mean\_size).mean()  make\_features(data, 4, 4)  print(data.head()) | |

### Обучение модели

Обучим линейную регрессию с учётом новых признаков.

Разделим данные на обучающую и тестовую выборки. Отстающие значения и скользящее среднее вычисляются по прошлым данным. Признаки для первых значений тестовой выборки находятся в конце обучающей выборки.

Получить признаки для первых значений обучающей выборки нельзя: исторических данных по ним нет. В предыдущих задачах значения этих признаков были NaN. Их нужно удалить.

|  |  |  |
| --- | --- | --- |
| 1.  Разбейте датасет о потреблении электроэнергии на обучающую и тестовую выборки в соотношении 4:1. Вам нужны данные за всё время. Из обучающей выборки удалите строки с пропусками.  Напечатайте на экране размеры обучающей и тестовой выборки (уже в прекоде). | | import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  data = pd.read\_csv('/datasets/energy\_consumption.csv', index\_col=[0], parse\_dates=[0])  data.sort\_index(inplace=True)  data = data.resample('1D').sum()  def make\_features(data, max\_lag, rolling\_mean\_size):  data['year'] = data.index.year  data['month'] = data.index.month  data['day'] = data.index.day  data['dayofweek'] = data.index.dayofweek    for lag in range(1, max\_lag + 1):  data['lag\_{}'.format(lag)] = data['PJME\_MW'].shift(lag)  data['rolling\_mean'] = data['PJME\_MW'].shift().rolling(rolling\_mean\_size).mean()  # мы выбрали произвольные значения аргументов  make\_features(data, 1, 1)  # < напишите код здесь >  train,test = train\_test\_split(data, shuffle=False, test\_size=0.2)  train = train.dropna()  print(train.shape)  print(test.shape) |
| 2.  В выборке выделите признаки и целевой признак. На них обучите линейную регрессию и сохраните её в переменной *model*. Затем напечатайте на экране значения *MAE* для обучающей и тестовой выборок (уже в прекоде). Подберите аргументы функции *make\_features()* так, чтобы значение *MAE* на тестовой выборке было не больше 37 000. | import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LinearRegression  from sklearn.metrics import mean\_absolute\_error  data = pd.read\_csv('/datasets/energy\_consumption.csv', index\_col=[0], parse\_dates=[0])  data.sort\_index(inplace=True)  data = data.resample('1D').sum()  def make\_features(data, max\_lag, rolling\_mean\_size):  data['year'] = data.index.year  data['month'] = data.index.month  data['day'] = data.index.day  data['dayofweek'] = data.index.dayofweek    for lag in range(1, max\_lag + 1):  data['lag\_{}'.format(lag)] = data['PJME\_MW'].shift(lag)  data['rolling\_mean'] = data['PJME\_MW'].shift().rolling(rolling\_mean\_size).mean()  make\_features(data, 6, 1)  train, test = train\_test\_split(data, shuffle=False, test\_size=0.2,random\_state=12345)  train = train.dropna()  # < напишите код здесь >  train\_features = train.drop(['PJME\_MW'],axis=1)  train\_target = train['PJME\_MW']  test\_features = test.drop(['PJME\_MW'],axis=1)  test\_target = test['PJME\_MW']  model = LinearRegression()  model.fit(train\_features, train\_target)  predict\_train = model.predict( train\_features)  predict\_test = model.predict(test\_features)  # < напишите код здесь >  print("MAE обучающей выборки:", mean\_absolute\_error(train\_target, predict\_train))  print("MAE тестовой выборки: ", mean\_absolute\_error (test\_target, predict\_test)) | |
|  | |  |

### Проект: Временные ряды

#### Прогнозирование заказов такси

Поздравляем! Вы прошли курс в тренажёре. Теперь можете самостоятельно решить новую задачу машинного обучения.

Когда закончите, отправьте работу на проверку ревьюеру: он пришлёт комментарии в течение суток. После этого нужно доработать проект и пройти повторную проверку.

Скорее всего, вы будете дорабатывать кейс по комментариям ещё несколько раз. Это нормально.

Проект завершён, когда ревьюер одобрил все доработки.

#### Описание проекта

Компания «Чётенькое такси» собрала исторические данные о заказах такси в аэропортах. Чтобы привлекать больше водителей в период пиковой нагрузки, нужно спрогнозировать количество заказов такси на следующий час. Постройте модель для такого предсказания.

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

#### Инструкция по выполнению проекта

1. Загрузите данные и выполните их ресемплирование по одному часу.
2. Проанализируйте данные.
3. Обучите разные модели с различными гиперпараметрами.

Сделайте тестовую выборку размером 10% от исходных данных.

1. Проверьте данные на тестовой выборке и сделайте выводы.

### Описание данных

Данные лежат в файле /datasets/taxi.csv. [Скачать датасет](https://code.s3.yandex.net/datasets/taxi.csv).

Количество заказов находится в столбце 'num\_orders' (от англ. number of orders, «число заказов»).

#### Как будут проверять проект?

Мы подготовили критерии оценки проекта, которыми руководствуются ревьюеры. Прежде чем приступить к решению кейса, внимательно их изучите.

На что обращают внимание ревьюеры, проверяя проект:

* Все ли шаги по инструкции выполняете?
* Как готовите данные?
* Как анализируете данные?
* Какие модели и гиперпараметры рассматриваете?
* Не дублируете ли код?
* Какие выводы делаете?
* Следите ли за структурой проекта?
* Поддерживаете аккуратность кода?