Анализ временных рядов

1 августа 2023 г. 15:00

Введение во временные ряды, основные понятия

Временные ряды - последовательность значений, описывающих протекающий во времени процесс, значения измерены в последовательные промежутки времени, обычно через равные промежутки. Временной ряд может быть проиндексирован как по времени, так и по другой индексированной переменной. Ряд может быть как дискретным, так и непрерывным.

Анализ временных рядов - это набор методов для получения/оценки/прогнозирования параметров и характеристик ряда. Такими характеристиками могут быть дальнейшие значения выборки, некоторые параметры ряда или оценка соответствия ряда некоторой модели. Как правило ряды обрабатываются статистическими методами анализа.

Типы данных (шкал) временных рядов:
1) Номинальная, дихотомическая - данные о принадлежности классу, сравниваются на равенство или неравенство.
2) Порядковая - данные указывают на порядок следования, можно сравнивать на больше-меньше
3) Интервальная - появляется возможность складывать и вычитать данные
4) Шкала отношений - появляется возможность умножать и делить данные, имеется абсолютный ноль шкалы.

Модели временных рядов
Модель временного ряда - математические модели прогнозирования, которые стремятся найти зависимость будущих значений от прошлых внутри самого процесса и на основе этой зависимости вычислить прогноз.

В модели временных рядов выделяют следующие части:
• **Тренд** - медленно меняющаяся часть зависимости временного ряда
• **Цикличность** - относительно медленно меняющиеся периодические составляющие, с нерегулярным периодом и высокой интенсивностью
• **Сезонность** - относительно быстро меняющиеся периодические составляющие, как правило с регулярным периодом.
• **Шум** - некоторое случайное искажение выходных значений, нерегулярные или случайные колебания

Общий вид одномерной аддитивной модели временного ряда:
$y(t) = a0 + trend(t) + seasonal(t) + cyclic(t) + noise(t)$

Общий вид одномерной мультипликативной модели временного ряда:
$y(t) = a0 * trend(t) * seasonal(t) * cyclic(t) * noise(t)$

В общем случае любая модель временного ряда может быть представлена как комбинация нескольких мультипликативных и аддитивных моделей. В более сложных случаях модель может иметь несколько составляющих тренда или других составляющих. Между компонентами модели необязательно линейные отношения.

Задачи анализа временных рядов
• Прогнозирование будущих значений в серии
• Оценка параметров ряда (периода сезонности, периода цикличности, наклон тренда)
• Обнаружение аномалий
• Шумоподавление
• Декомпозиция временного ряда

Типы анализа временных рядов

• Одномерный анализ - простейшая форма анализа временных рядов, когда рассматривается зависимость от времени только одной переменной
• Многомерный анализ - более сложная форма анализа, рассматривается зависимость от времени сразу нескольких признаков.

Особенности представления данных:
• Лаги (запаздывания) - значения на текущем и предыдущих временных шагах.
• Сегменты (окна) - подвыборка значений за фиксированное окно предыдущих временных шагов.

Классификация временных рядов:
• Стационарный ряд - ряд, параметры которого не меняются со временем, в нём есть постоянное среднее и дисперсия, тренда нет как такового.
• Нестационарный ряд - ряд, параметры которого меняются со временем, например параметр монотонно уменьшается, такие ряды имеют выраженный тренд.

• Детерминированный ряд - ряд, не имеющий случайных событий, такой ряд может быть выражен формулой, мы можем проанализировать как параметры вели себя в прошлом и спрогнозировать их поведение в будущем.
• Недетерминированный ряд - ряд, в котором присутствуют случайные события, прогнозирование параметров таких рядов становится сложнее. Анализ происходит благодаря средним значениям и дисперсии.

• Линейный ряд - ряд, в котором компоненты ряда имеют линейные или сведённые к линейным соотношения.
• Нелинейный ряд - ряд, компоненты которого имеют сложные, нелинейные соотношения.

• Структурированный ряд - ряд, с систематическими зависящими от времени закономерностями (явный тренды или сезонность)
• Неструктурированный ряд - ряд без систематической зависящей от времени закономерности (случайный тренд или сезонность)

Статистический подход к анализу временных рядов

Статистические характеристики временного ряда:
• Математическое ожидание
• Стандартное отклонение
• Медиана
• Мода

Автокорреляционная функция
Корреляция - это линейная связь между парой случайных величин. Это степень, в которой два образца линейно зависят друг от друга.

Коэффициент корреляции - коэффициент, отражающий тесноту связи между двумя случайными величинами, например временными рядами.

Автокорреляция - это способность выборок временных рядов быть линейно связанными со своими задержанными отсчётами (лагами). Мера корреляции между значениями одного ряда с разницей во времени.

Ковариация - мера взаимодействия двух случайных величин. Мера взаимосвязи двух случайных величин, измеряющая общее отклонение двух случайных величин от их ожидаемых значений. Эта метрика оценивает, в какой степени переменные изменяются вместе.

Шумы
Белый гауссов шум - ситуация, которая возникает при независимости двух признаков, мат

ожидание гауссового шума равно нулю, а дисперсия в любой точке одинакова и максимальна. Если рассматриваемый временной ряд - белый гауссов шум, то по определению он случайный и мы не можем разумно его моделировать и делать прогнозы.

Если значения, оставшиеся после анализа временного ряда представляют из себя белый шум, значит вся информация о сигнале была использована моделью для прогнозирования и всё что осталось - случайные колебания, которые не могут быть смоделированы.

Цветной шум - некоторый стационарный шум, характеристики которого повторяются через определённый промежуток времени

Методы предсказания временных рядов

Базовые понятия

- Авторегрессионная модель - модель, которая использует информацию из предыдущих временных шагов для прогнозирования значения на следующем шаге
- Автокорреляционная модель - модель, в которой утверждается, что наблюдения за предыдущими значениями целесообразны для прогнозирования значения на следующем шаге. То есть утверждается, что в разные моменты времени между значениями временного ряда есть некоторая связь.

Базовые модели

- Авторегрессионная модель (ARM) - линейная модель, в которой прогнозированная величина является суммой прошлых значений, умноженных на числовой множитель.

- Скользящее среднее (MA) - самый наивный подход к моделированию временных рядов. Согласно этой модели следующее наблюдение является средним значением всех прошлых наблюдений.

- Экспоненциальное сглаживание (SEMA)- продолжает логику скользящего среднего, но теперь с помощью коэффициента альфа мы можем регулировать как быстро модель будет "забывать" предыдущие данные. Чем меньше коэффициент альфа, тем более плавным будет ряд, потому что приближая его к нулю мы приближаемся к модели скользящего среднего.

- Двойное экспоненциальное сглаживание (DEMA, DES) - используется при наличии цикличности в данных, это по сути рекурсивное использование экспоненциального сглаживания дважды, сначала на всём временном ряду, а затем вторым сглаживанием, заходим глубже уже в участки цикличности и сглаживаем уже их.

- Тройное экспоненциальное сглаживание (TEMA) - используется при наличии цикличности и сезонности в данных, сначала сглаживаем весь ряд, затем участки цикличности, затем участки сезонности.

Метрика sMAPE (Symmetric mean absolute percentage error)
Используется для измерения точности по отношению к фактическим значениям. sMAPE является производной метрикой от MAPE.

$$\mathrm{MAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{A_i - F_i}{A_i} \right|$$

Ai - фактическое значение
Fi - полученное значение

Но при малом Ai метрика может резко увеличиться к плюс бесконечности, таким образом MAPE имеет смысл только если все наблюдения имеют достаточно большие фактические значения. Если

же это не так, можно использовать sMAPE - симметричную среднюю абсолютную ошибку.

Существует два определения sMAPE:
В обоих случаях она нормализует относительные ошибку путём деления как на фактическое, так и на предсказанное значение, но во втором случае знаменатель дополнительно делится на 2, расширяя диапазон от 0 до 200%

1-й случай, диапазон от 0 до 100%

$$\mathrm{sMAPE}^{0-100} = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|A_i - F_i|}{|A_i| + |F_i|}$$

2-й случай, диапазон от 0 до 200%

$$\mathrm{sMAPE}^{0-200} = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|A_i - F_i|}{(|A_i| + |F_i|)/2}$$

sMAPE может быть более интерпретируемым, чем ожидалось. Достаточно малые значения sMAPE^{0-200} — особенно менее 30% — можно интерпретировать так же, как значения MAPE. Это также относится к значениям sMAPE^{0-100} после преобразования их в значения sMAPE^{0-200} путем умножения их на 2.

В любом случае, чем меньше sMAPE, тем лучше работает модель.

Предварительная обработка и разведывательный анализ данных

- Колонки - изучить колонки, к какой шкале относятся данные, какого они типа, что обозначают
- Размерность - df.shape
- Общая информация по данным - df.info()
- Статистические параметры - df.describe()
- Несколько первых строк - df.head(x)
- Несколько последних строк - df.tail(x)
- Тип данных - df.dtypes

Обработка пропущенных значений
Удаление пропущенных значений
df.dropna(inplace=True/False)

Обнаружение пропущенных значений
- isnull()
- isna()

Подсчёт количества пропущенных значений
- df.isnull().sum()
- df.isna().sum()

Простейшая замена пропущенных значений конкретным значением
- df.fillna(0, inplace=True) # Заменяем NaN значения нулями

Простейшие методы
```
# Заполнение пропусков средним значением
passengers['target'] = passengers.target.fillna(passengers.target.mean())

# Заполнение пропусков медианным значением
```

```
passengers['target'] = passengers.target.fillna(passengers.target.median())

# Заполнение предыдущим значением
passengers['target'] = passengers.target.ffill()

# Заполнение последующим значением
passengers['target'] = passengers.target.bfill()
```

Хотя эти методы понятные и простые, они могут внести систематическую ошибку, особенно, если заполнять таким образом несколько пропусков подряд. Эту проблему можно попытаться решить, используя параметр limit, который ограничивает количество последовательно заполняемых пропусков одним и тем же значением.

Заполнение скользящим средним и медианой
В этом подходе для заполнения пропусков используется не одно значение, а несколько, то есть считается среднее или медиана по некоторому окну.

Здесь будут важны настройки:
- window - размер окна
- min_periods - минимальное количество периодов для расчёта среднего и медианы

```
# Заполнение средним значением по окну
passengers['target'] = passengers.target.fillna(
    passengers.target.rolling(window = 5, min_periods = 1).mean())
    )

# Заполнение медианным значением по окну
passengers['target'] = passengers.target.fillna(
    passengers.target.rolling(window = 5, min_periods = 1).median())
    )
```

Интерполяция
Различные методы интерполяции подойдут для временных рядов, у которых есть тренд. Суть метода - по имеюшимся наблюдениям вычислить проходящую через них функцию.

- Функция, найденная в процессе интерполяции, проходит строго через известные точки (этим она отличается от аппроксимации (которая стремится с помощью простой функции описать более сложную) и регрессии (которая стремится найти минимизирующую ошибку функцию)

- Найденная функция описывает только заданный известными точками интервал и не выходит за его пределы (этим она отличается от экстраполяции)

Некоторые термины интерполяции:

- Значения каждой из известных точек, по которым строится интерполирующая функция, называются узлами (knots)
- Совокупность точек — интерполяционной сеткой (grid)
- Сама функция называется интерполянтом (interpolant)

Способы интерполяции
- Линейная интерполяция - между каждой парой известных точек строит линейную функцию. Считает что все три точки (две известные и одна неизвестная) лежат на одной прямой, поэтому коэффициент наклона будет одинаков. Соответственно через систему линейных уравнений можно рассчитать координаты неизвестной точки.

- Полиномиальная интерполяция - вместо линейных уравнений, пытается описать функцию в

виде полинома второй или большей степени, график которого проходил бы через все известные точки. Заметим, что для всех точек строится единая функция, которая бы описывала сразу всё. Но с ростом числа точек могут возникнуть осцилляции, которые негативно скажутся на точности заполнения пропусков (феномен Рунге).

- Сплайн-интерполяция - между каждой парой известных точек строит полиномиальные функции. Это позволяет использовать полиному меньшей степени по сравнению с полиномиальной интерполяцией, что позволяет избежать феномена Рунге.

Реализация в Python
```
# Линейная интерполяция
passengers['linear'] = passengers.target.interpolate(method = 'linear')

# Полиномиальная интерполяция
passengers['polynomial'] = passengers.target.interpolate(method = 'polynomial', order = 3)

# Сплайн-интерполяция (порядок от 1 до 5)
passengers['spline'] = passengers.target.interpolate(method = 'spline', order = 5)
```

Сравнение различных методов по отношению ко временным рядам разных типов показывают, что более сложные методы интерполяции хорошо работают на нестационарных рядах с относительно большим шагом. При этом в стационарных временных рядах с меньшим шагом большую эффективность показывают простые методы.

Особенности и возможности индексов в формате datetime

Перевод в формат дата-время
Переводим столбец даты-времени в специальный формат DateTime и превращаем этот столбец в столбец индексов

```
df.Date = pd.to_datetime(df.Date) # Переводим столбец даты в специальный формат даты-времени
df.set_index('Date', inplace=True) # Заменяем столбец индексов столбцом даты-времени
df.sample(15, random_state=0) # Выводим образец набора данных, первые 15 строк
```

Фильтрация по loc-iloc
Используя даты как индексы, мы можем гибко фильтровать данные, например взять данные только по одному году или по диапазону лет, по конкретному месяцу конкретного года и так далее.

Для такой фильтрации данных используются методы loc() и iloc()
- loc - выбирает строки и столбцы на основе "меток" (значение индекса, название столбца)
- iloc - выбирает строки и столбцы на основе их целочисленных позиций (то есть выборка происходит не по названию столбца или символа индекса, а по его порядковому номеру)

Примеры:
```
df.iloc[10] - возьмём 11-ю строку по порядковому номеру
df.iloc[0:10] - возьмём первые 10 строк по их порядковым номерам
df.loc['2017-08'] - возьмём все столбцы данных, и только те строки, которые относятся к августу 2017 года.
df.loc['2014-01-20':'2014-01-22'] - возьмём все столбцы, и только те строки, которые относятся к указанному диапазону
```

Варианты обращения к отдельным столбцам:
```
# В дополнение к диапазону:
df.loc['2014-01-20':'2014-01-25', 'Wind']
# Обращение к столбцам как к методам:
df.Wind.loc['2014-01-20':'2014-01-25']
# Обращение к столбцам как к ключевым словам:
```

df[['Wind']].loc['2014-01-20':'2014-01-25']

Возможности получения временных меток
df.index.day - возвращает значение дня по всему столбцу даты
df.index.weekday - возвращает день недели, где 0 - понедельник, 6 - воскресенье
df.index.month - возвращает значения года
df.index.year - возвращает значения года

Частотное представление данных

Данные можно представлять с разной частотой, например с частотой D - для дней, W,M,Y для недель, месяцев и лет соответственно.
df[['Wind']].loc['2014-01-20':'2014-02-25'].asfreq('W') - в указанном диапазоне получим один столбец с шагом в неделю

Можно взять месяц из временных меток и посмотреть, как он будет выглядеть в недельном представлении.
df.loc['2012-02'].asfreq('W') - выбрали месяц одного года, вывели все столбцы с шагом в неделю

Можно также использовать диапазонный тип обращений, как для списков в питоне.
df.loc['2012':].asfreq('Y') - начиная с указанного года выводим все столбцы с шагом в год

Группировка по частотам
Когда индексы имеют формат дат, их можно сгруппировать по заданным частотам.
Для группировки могут быть использованы методы groupby, resample и asfreq.
После группировки можно использовать групповые операции.

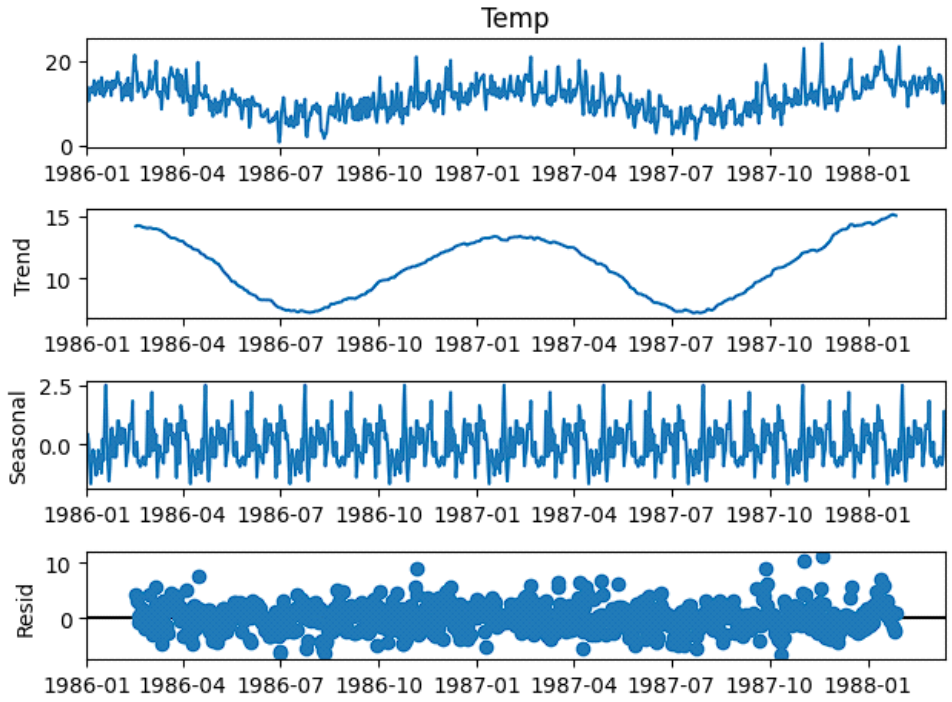
Пример:
df.groupby(pd.Grouper(freq='1y')).sum() - получаем таблицу суммы данных каждого столбца за год
df.resample('1w').median().head(3) - получаем первые три строки медианных значений за неделю
df.asfreq('1w').head(3) - получаем первые три строки значений для каждой недели

Получение начальной информации о данных

Предварительная декомпозиция временного ряда

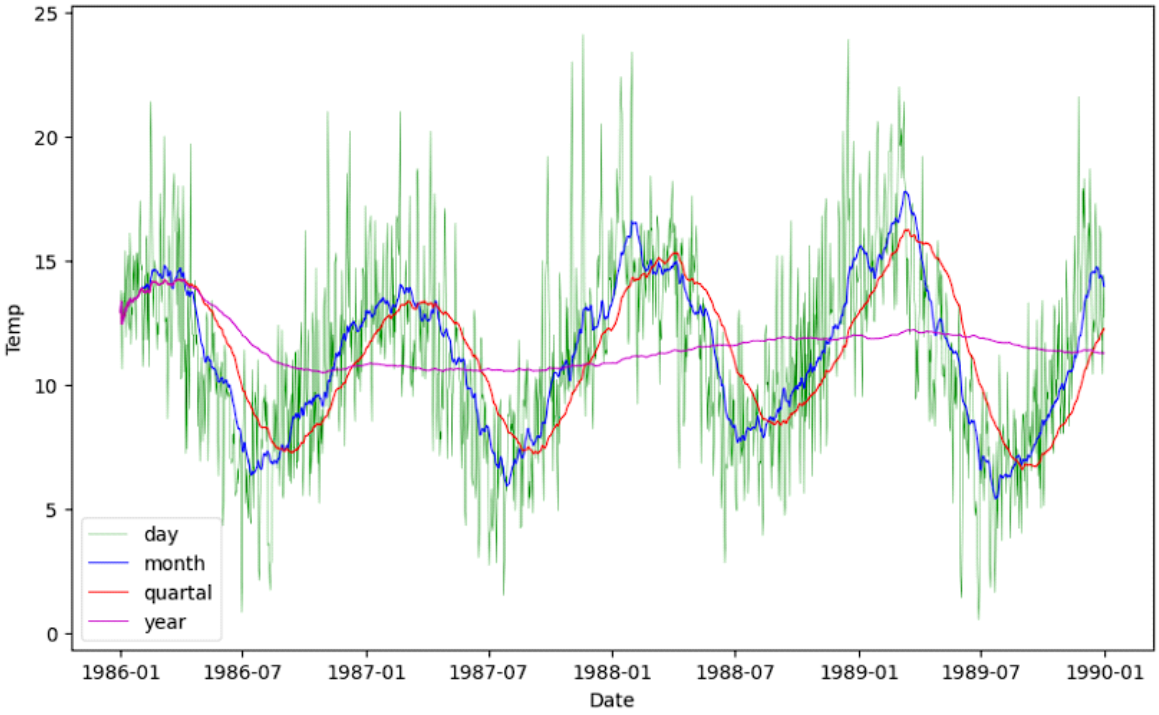
Декомпозиция временного ряда - разделение временного ряда на компоненты: тренд, цикличность, сезонность и шум.
С помощью функции seasonal_decompose декомпозицию можно сделать автоматически.
(Но судя по всему это не самый точный метод, возможно он больше подходит для начального анализа, или же, вместо тренда здесь показывается цикличность, хотя функция не подразумевает её вывод)

```
# Импорт функции для декомпозиции
from statsmodels.tsa.seasonal import seasonal_decompose

# Передаём в функцию ряд, который хотим разделить, модель ряда (аддитивная или мультипликативная) и окно сглаживания в единицах измерения ряда (т.е. в минимальной рассматриваемой величине ряда)
result = seasonal_decompose(y_train, model='additive', period = 93)

# Строим графики декомпозиции
result.plot();
```



Декомпозиция с помощью скользящего среднего "вручную"

```
# Сглаживание по дням
df_rol_d = df[['Temp']].rolling('1d').mean() # Непосредственно функция для MA (Moving average)
fig, ax = plt.subplots(nrows=1, ncols=1, figsize = (10, 6))
sns.lineplot(data = df_rol_d, x = "Date", y = "Temp", color = "g", linewidth = 0.3, label = "day")

# Сглаживание по месяцам
df_rol_m = df[['Temp']].rolling('31d').mean()
sns.lineplot(data = df_rol_m, x = "Date", y = "Temp", color = "b", linewidth = 0.7, label = "month")

# Сглаживание по кварталам
df_rol_q = df[['Temp']].rolling('93d').mean()
sns.lineplot(data = df_rol_q, x = "Date", y = "Temp", color = "r", linewidth = 0.7, label = "quartal")

# Сглаживание по году
df_rol_y = df[['Temp']].rolling('365d').mean()
sns.lineplot(data = df_rol_y, x = "Date", y = "Temp", color = "m", linewidth = 0.7, label = "year")
```



Проверка автокорреляции

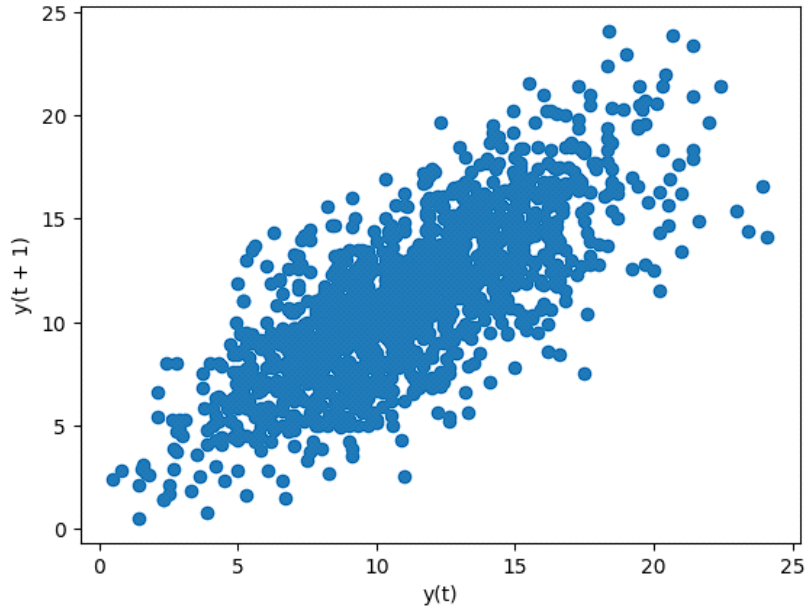
Автокорреляция - линейная зависимость текущего значения параметра от значения этого же параметра в другой промежуток времени.

Чтобы проверить автокорреляцию данных, нужно построить график зависимости текущих значений параметра от предыдущего, то есть предыдущего лага. Для этого в pandas есть специальная функция lag_plot(), которая строит точечный график соответствующей зависимости.

lag_plot(series, lag)

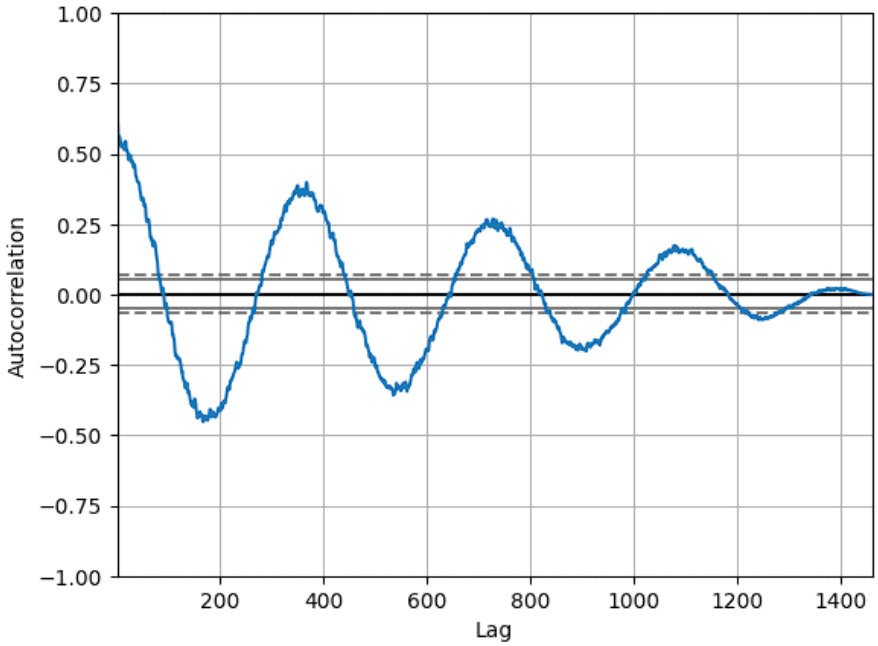
- series - временной ряд, который проверяем на автокорреляцию
- lag - показывает с каким шагом назад мы сравниваем текущее значение, lag = 1 - значит текущее сравниваем с предыдущим, lag = 2 - значит текущее сравниваем с пред-предыдущим и тд.

```
# Импортируем нужные библиотеки и функции
from pandas import Series
from matplotlib import pyplot
from pandas.plotting import lag_plot

# Передаём столбец данных для которого проверяем автокорреляцию
series = df_86_89
# Вызываем функцию, которая построит график, передаём в неё столбец и величину лага
lag_plot(series, lag = 1)
```



Но таким образом удобно проверять только конкретные лаги, чтобы сразу проверить корреляцию столбца при всех возможных значениях лага, следует воспользоваться функцией autocorrelation_plot

```
# Импортируем нужную функцию
from pandas.plotting import autocorrelation_plot
# Задаём проверяемый столбец
series = df_86_89
# Передаём в функцию столбец и получаем график коэффициента корреляции в зависимости от величины лага
autocorrelation_plot(series)
```

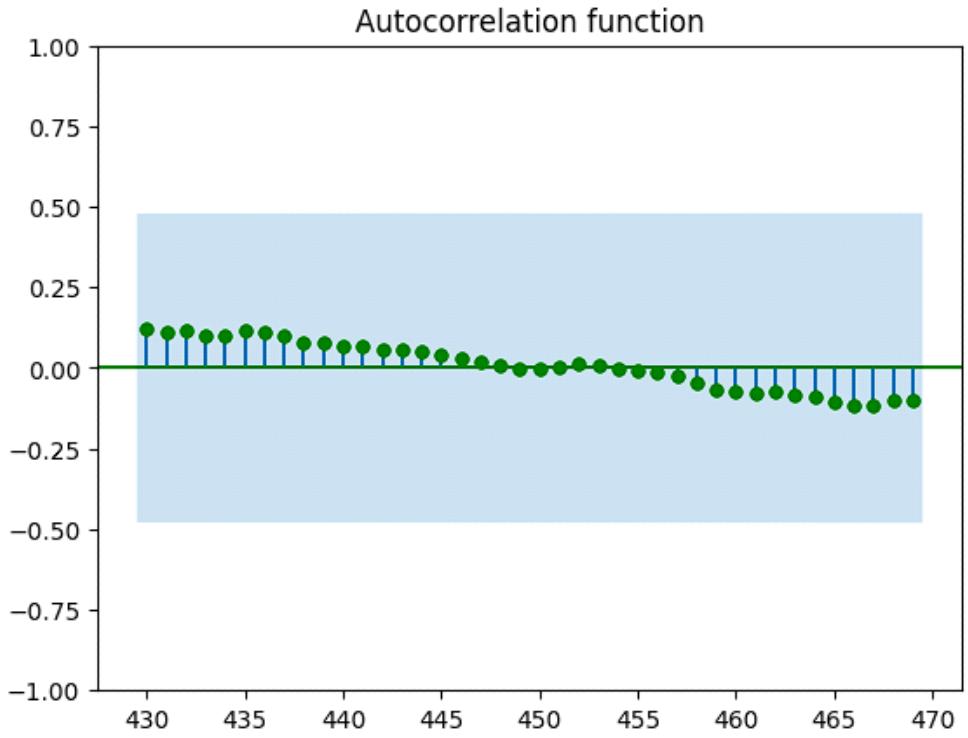


Этот график показывает зависимость коэффициента корреляции от величины лага, мы рассматриваем датасет колебаний температуры, видно, что чем больший лаг мы возьмём, тем меньше будет автокорреляция, а так как к текущему значению (например летней температуре) мы можем сопоставить зимнюю температуру (выбрав соответствующий лаг), то есть моменты, когда коэффициент корреляции уходит в минус

Эту же зависимость мы можем представить в виде линейного графика

```
# Импортируем нужную функцию
from statsmodels.graphics.tsaplots import plot_acf
```

```
# Передаём исследуемый столбец
series = df_86_89
# Строим график, мы можем передать или одно значение лага, до которого будет строиться график, или задать диапазон, график построит зависимость внутри этого диапазона
plot_acf(series, lags=(range(430, 470)), color='g', title='Autocorrelation function')
```



Проверка на стационарность

Стационарный временной ряд - ряд, который с течением времени не меняет свои статистические характеристики (постоянство мат ожидания, постоянство дисперсии (гомоскедантичность), независимость ковариации от времени)

Стационарность важна, потому что по стационарному ряду легко строить прогноз, так как мы полагаем что его будущие статистические характеристики не будут отличаться от наблюдаемых текущих. Большинство моделей так или иначе моделируют и предсказывают эти характеристики, поэтому если ряд нестационарен, такие прогнозы окажутся неверными.

Один из способов проверить, является ли временной ряд стационарным - это выполнить тест Дики-Фуллера

H0 - Временной ряд является нестационарным

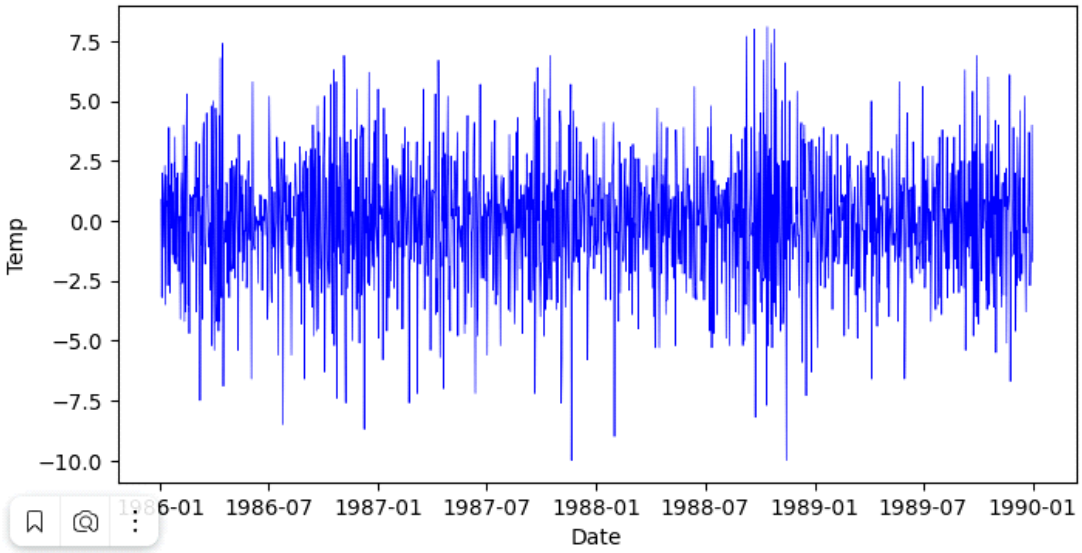
```
# Импортируем нужную функцию
from statmodels.tsa.stattools import adfuller

# Передаём исследуемый столбец
result = adfuller(df_86_89)

# Получаем нужные выходные данные
stats = result[0]
p_value   = result[1]

# Выводим результат
print("Статистика", stats, "p-value", p_value )
```

Метод возвращает несколько значений, нас интересует p_value, с индексом [1]. Соответственно, если p_value меньше уровня значимости, значит нулевая гипотеза отвергается, то есть ряд стационарен.

Чтобы сделать ряд более стационарным его нужно продифференцировать, это делается с помощью метода diff():

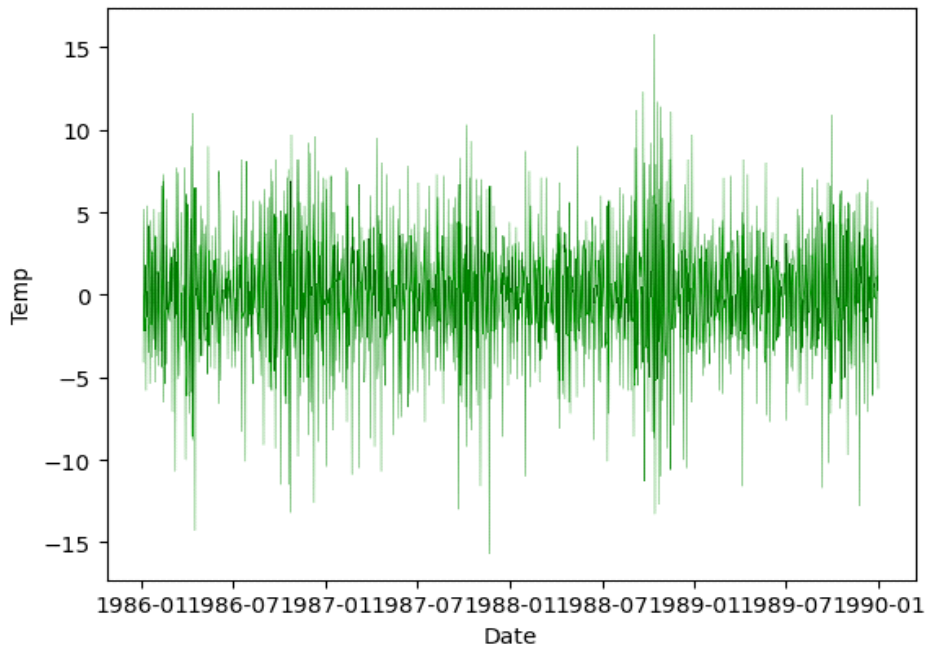
df_1_dif = df.diff() - первая производная исходного столбца
df_2_dif = df.diff().diff() - вторая производная исходного столбца

```
# Пример графика первой производной
sns.lineplot(data = df_86_89.diff(), x = df_86_89.index, y = "Temp", color = "b", linewidth = 0.5)
```




```
# Пример графика второй производной
sns.lineplot(data = df_86_89.diff().diff(), x = df_86_89.index, y = "Temp", color = "g", linewidth = 0.3)
```

```
<Axes: xlabel='Date', ylabel='Temp'>
```



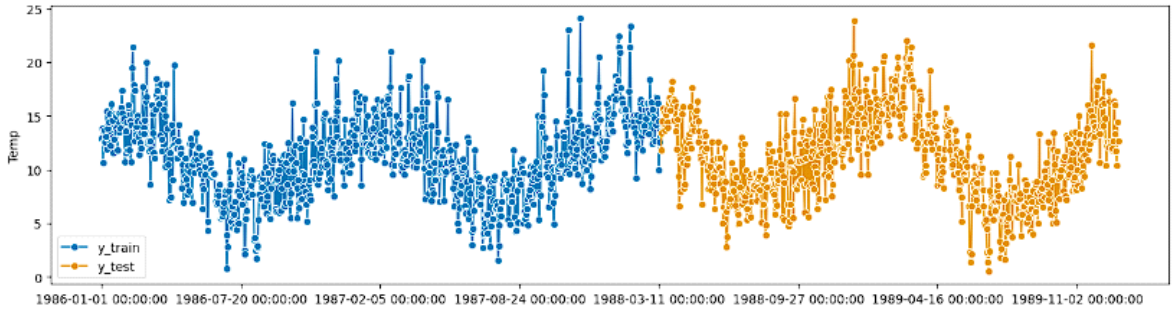
Но пока возникают вопросы, а что делать дальше, тест дики-фуллера выдаёт ошибку на дифференцированном столбце, возможно из за нулевых значений, и из за нулевых же значений многие методы предсказания не станут работать

Обучение моделей

Разделение данных на тренировочные и тестовые

Разделим данные на тренировочные и тестовые, для этого используется функция temporal_train_test_split

```
# Импорт функции
from sktime.forecasting.model_selection import temporal_train_test_split
# Задаём процент разделения на тренировочную и тестовую
%_test = int(0.45 * df.Temp.size)
# Делим исходный датасет с помощью импортированной функции, в функции указываем какой признак разделяем и какой размер деления, получившиеся два фрагмента датасета сохраняем в две соответствующие переменные
y_train, y_test = temporal_train_test_split(df.Temp, test_size = %_test)
# Строим получившийся разделённый график
sktime.utils.plotting.plot_series(y_train, y_test, labels = ["y_train", "y_test"], markers = [None, None])
```



Построение графиков в sktime
Мы всегда можем построить график с помощью известных matplotlib и seaborn, но для лаконичности это лучше сделать через основную библиотеку sktime.

```
# Импорт функции для построения графиков
from sktime.utils.plotting import plot_series
# Построение графика с указанными параметрами
plot_series(y_train, y_test, labels=['name_1', 'name_2'], markers=None, colors=None, title=None, x_label=подпись_оси_x, y_label=подпись_оси_y, ax = [0, 1])
```

Использование метрик в sktime
sktime также содержит собственный модуль для использования метрик

```
# Импорт модуля для использования метрик (такой импорт опционален)
import sktime.performance_metrics.forecasting
# Импорт непосредственно метрик
from sktime.performance_metrics.forecasting import  MeanAbsolutePercentageError as MAPE
```

Функции и модели анализа временных рядов
ForecastingHorizon(values, is_relative) - определяет количество элементов, значения которых мы хотим предсказать

- values - непосредственно индексы точек, которые хотим спрогнозировать
- is_relative - True/False

NaiveForecaster(strategy, window_length, sp) - предсказатель, использующий простые стратегии прогнозирования. Может выступать в качестве базовой модели, относительно которой будут проводиться улучшения.

- strategy
  Есть три стратегии: last, mean, drift:
  - last - предсказывает весь горизонт как последнее значение в тренировочных данных.
  - mean - предсказывает весь горизонт как среднее значение среди тренировочных данных
  - drift - берёт первую и последнюю точку из выбранного размера окна, соединяет их и экстраполирует эту закономерность на горизонт.

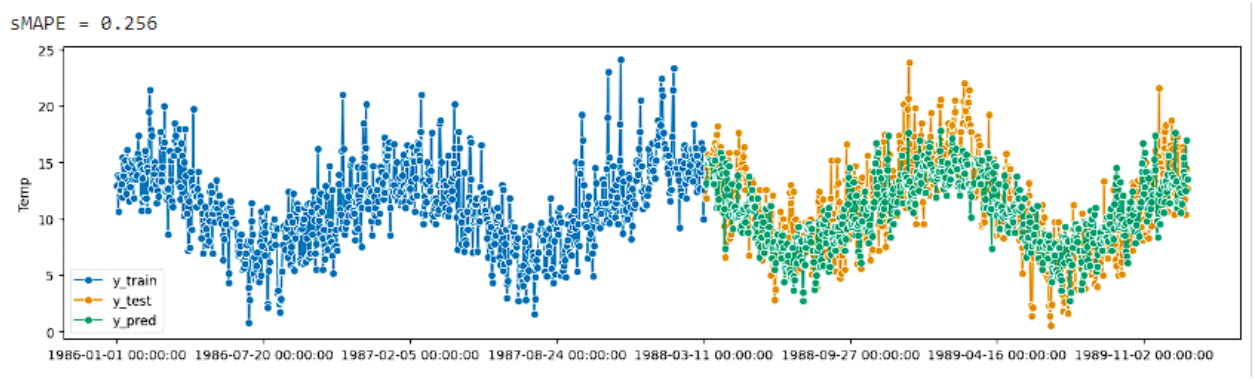
- window_length - ширина окна, которым будем проходиться по ряду

- sp - период сезонности, правильно указанный период сезонности хорошо влияет на качество модели.

  Не все вариации параметров хорошо сочетаются друг с другом.

Пример

```
# Задаём горизонт предсказаний
fh = ForecastingHorizon(y_test.index, is_relative=False)
# Выбираем стратегию и устанавливаем период сезонности
forecaster = NaiveForecaster(strategy="mean", sp=365)
# Обучаем модель
forecaster.fit(y_train)
# Делаем прогноз, передавая горизонт предсказаний
y_pred = forecaster.predict(fh)
# Строим график
plot_series(y_train, y_test, y_pred, labels=["y_train", "y_test", "y_pred"])
# Выводим значение метрики
mse = MeanSquaredError()
mse(y_test, y_pred)
```



ExponentialSmoothing(trend, damped_trend, seasonal, use_boxcox, sp) - экспоненциальное сглаживание, принцип аналогичен скользящему среднему, но более ранние значения, на которых основывается усреднение, постепенно стираются из памяти метода, то есть он берёт среднее не по всему ряду, а по некоторому окну.

- trend - add/mul - указываем тип тренда
- damped_trend - True/False - указываем, должен ли тренд быть затухающим
- seasonal - add/mul - указываем тип сезонности
- use_boxcox - True/False
- sp - период сезонности

Возможны различные комбинации указанных аргументов.

Преобразование Бокса-Кокса - метод преобразования ненормально распределённых данных в как можно более нормально распределённые.

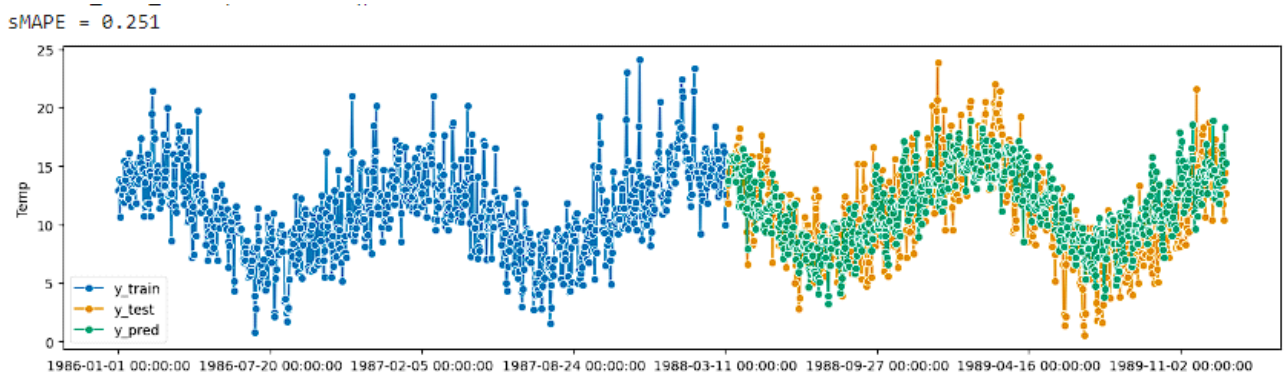
Различают три разновидности экспоненциального сглаживания:

1. Линейное экспоненциальное сглаживание (SES) - применяется для данных без явного тренда или сезонности.
2. Двойное экспоненциальное сглаживание (DES, метод Хольта) - применяется для данных с явным трендом, но без сезонности.
3. Тройное экспоненциальное сглаживание (TES, метод Хольта-Винтерса) - применяется для

данных с явным трендом и сезонностью.

Пример

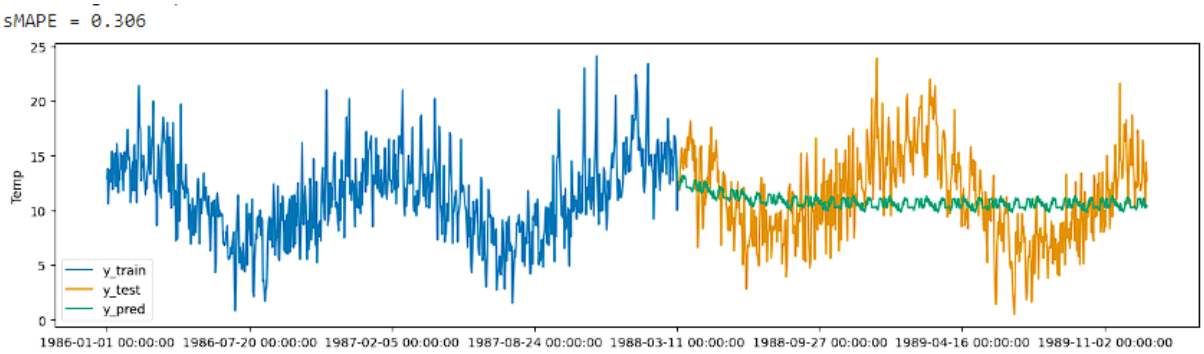
```
# Импорт модели предсказания
from sktime.forecasting.exp_smoothing import ExponentialSmoothing
# Задаём параметры модели предсказания
forecaster = ExponentialSmoothing(trend="additive", seasonal="additive", use_boxcox =True, sp=365)
# Обучаем модель
forecaster.fit(y_train)
# Предсказываем на горизонте
y_pred = forecaster.predict(fh)
# Строим график
plot_series(y_train, y_test, y_pred, labels=["y_train", "y_test", "y_pred"])
# Выводим метрику
mse = MeanSquaredError(square_root = True)
mse(y_test, y_pred)
```



- Ансамбли моделей предсказания - ансамбль из разных или одинаковых моделей с различными параметрами

```
# Импортируем функцию для создания ансамблей
from sktime.forecasting.compose import EnsembleForecaster

# Определяем используемые методы
ses = ExponentialSmoothing(sp=365)
holt_winter = ExponentialSmoothing(trend="add", seasonal="additive", sp=365)
holt_winter_mul_boxcox  = ExponentialSmoothing(trend="mul", seasonal="additive", use_boxcox =True,
sp=365)

# Строим предсказатель из указанных моделей
forecaster = EnsembleForecaster(
    [
        ("ses", ses),
        ("holt-winter", holt_winter)
        ("holt-winter, multiplicative trend, box-cox", holt_winter_mul_boxcox),
    ]
)

# Обучаем предсказатель
forecaster.fit(y_train)

# Делаем прогноз
y_pred = forecaster.predict(fh)

# Строим график
```

```
plot_series(y_train, y_test, y_pred, labels=["y_train", "y_test", "y_pred"], markers = [None, None, None])

# Получаем метрику
mse = MeanSquaredError(square_root = True)
mse(y_test, y_pred)
```

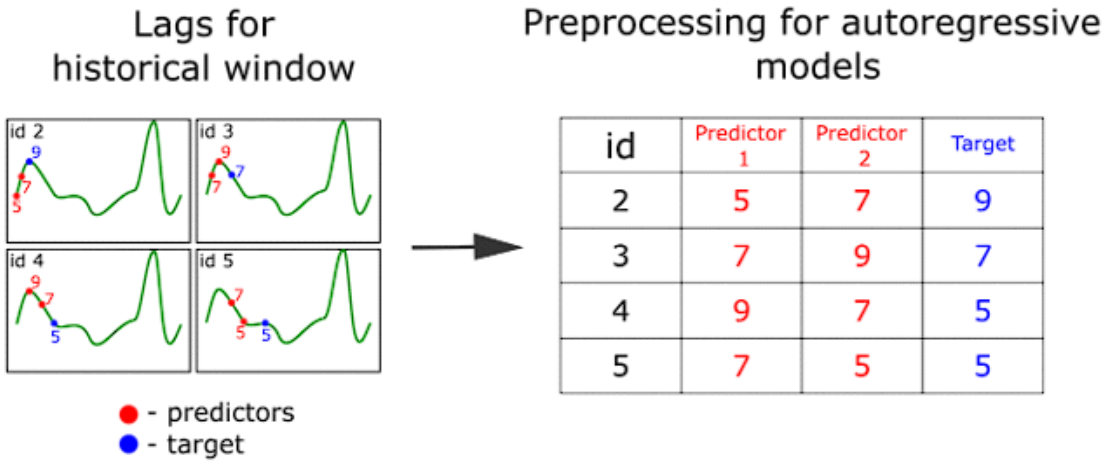


Применение классических алгоритмов машинного обучения для прогнозирования временных рядов

Сложность применения классических алгоритмов для прогнозирования временных рядов состоит в том, что признаки и целевая переменная представляют из себя единый столбец, то есть у нас нет набора признаков, которые бы предсказывали искомую переменную, потому что ряд должен предсказывать сам себя.

Для того, чтобы привести данные к удобному для классических алгоритмов виду, необходимо построить траекторную матрицу. Это матрица, состоящая из отрезков временного ряда так, что часть точек становятся признаками, а следующая за ними - искомой переменной.

Так как временной ряд — это последовательность, где последующие значения обычно зависят от предыдущих, то мы можем использовать текущие и предыдущие элементы временного ряда для прогнозирования будущих.



id	Predictor 1	Predictor 2	Target
2	5	7	9
3	7	9	7
4	9	7	5
5	7	5	5

Код для перевода временного ряда в траекторную матрицу

```
# transform a time series dataset into a supervised learning dataset
def gen_features(data, n_in=1, n_out=1, dropnan=True):
	n_vars = 1 if type(data) is list else data.shape[1]
	df = pd.DataFrame(data)
	cols = list()
	# input sequence (t-n, ... t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
```

```
        # forecast sequence (t, t+1, ... t+n)
        for i in range(0, n_out):
                cols.append(df.shift(-i))
        # put it all together
        agg = concat(cols, axis=1)
        # drop rows with NaN values
        if dropnan:
                agg.dropna(inplace=True)
        return agg.values


# example feature gen
f = gen_features(values, n_in=12, dropnan=False)

np.set_printoptions(edgeitems=30, linewidth=1000,
    formatter=dict(float=lambda x: "%.3g" % x))
print("Features:", f[0:15])
```

```
Features: [[nan nan nan nan nan nan nan nan nan nan nan nan 12.9]
 [nan nan nan nan nan nan nan nan nan nan nan 12.9 13.8]
 [nan nan nan nan nan nan nan nan nan nan 12.9 13.8 10.6]
 [nan nan nan nan nan nan nan nan nan 12.9 13.8 10.6 12.6]
 [nan nan nan nan nan nan nan nan 12.9 13.8 10.6 12.6 13.7]
 [nan nan nan nan nan nan nan 12.9 13.8 10.6 12.6 13.7 12.6]
 [nan nan nan nan nan nan 12.9 13.8 10.6 12.6 13.7 12.6 13.1]
 [nan nan nan nan nan 12.9 13.8 10.6 12.6 13.7 12.6 13.1 15.4]
 [nan nan nan nan 12.9 13.8 10.6 12.6 13.7 12.6 13.1 15.4 11.9]
 [nan nan nan 12.9 13.8 10.6 12.6 13.7 12.6 13.1 15.4 11.9 13.8]
 [nan nan 12.9 13.8 10.6 12.6 13.7 12.6 13.1 15.4 11.9 13.8 14.4]
 [nan 12.9 13.8 10.6 12.6 13.7 12.6 13.1 15.4 11.9 13.8 14.4 15.2]
 [12.9 13.8 10.6 12.6 13.7 12.6 13.1 15.4 11.9 13.8 14.4 15.2 12.5]
 [13.8 10.6 12.6 13.7 12.6 13.1 15.4 11.9 13.8 14.4 15.2 12.5 12.2]
 [10.6 12.6 13.7 12.6 13.1 15.4 11.9 13.8 14.4 15.2 12.5 12.2 16.1]]
```

Семейство моделей ARMA

Базовая модель ARMA состоит из двух компонентов:

- Авторегрессия (AR) - прогнозирование будущих значений на основании предыдущих значений ряда
- Модель скользящего среднего (MA) - помогает учесть случайные колебания и отклонения истинного значения от прогнозируемого

Модель ARMA с параметрами p и q может описать любой стационарный ряд (ряд, в котором отсутствует тренд и сезонность)

Если же данные не стационарны, следует использовать более сложные модели этого семейства:

- ARIMA, здесь добавляется компонент Integrated (I), который отвечает за удаление тренда (сам процесс называется дифференцированием);
- SARIMA, эта модель учитывает сезонность (Seasonality, S)
- SARIMAX включает еще и внешние или экзогенные факторы (eXogenous factors, X), которые напрямую не учитываются моделью, но влияют на нее.

Определение параметров модели

Для ARIMA:

- p: (AR) Количество наблюдений запаздывания, включенных в модель, также называемое порядком запаздывания.
- d: (I) Количество раз, когда исходные наблюдения различаются, также называемое степенью разности.
- q: (MA) Размер окна скользящей средней, также называемый порядком скользящей средней.

Коэффициенты p и q модели ARIMA находятся с помощью графиков ACF (Автокорреляционная функция) и PACF (Частичная автокорреляционная функция), параметр d - есть порядок интегрирования временного ряда, для его перехода к стационарному состоянию.

ACF поможет нам определить q, т. к. по ее коррелограмме можно определить количество автокорреляционных коэффициентов сильно отличных от 0 в модели MA. Сама ACF показывает зависимость текущего значения от запаздывающих.

PACF поможет нам определить p, т. к. по ее коррелограмме можно определить максимальный номер коэффициента сильно отличный от 0 в модели AR. График PACF измеряет корреляцию переменной после удаления эффектов предыдущих временных задержек.

Параметров у модели SARIMAX больше. Их полная версия выглядит как SARIMAX(p, d, q) x (P, D, Q, s). В данном случае, помимо известных параметров p и q, у нас появляется параметр d, отвечающий за тренд, а также набор параметров (P, D, Q, s), отвечающих за сезонность.

Проблема подбора гиперпараметров

На самом деле подбор параметров не является тривиальной задачей, нет возможности автоматического подбора, а ручной подбор является довольно кропотливым. Поэтому хорошим решением может быть применение модели auto_arima, это arima-модель, которая осуществляет автоматический подбор гиперпараметров, используя перебор из заданного диапазона.

Определение лучшего решения происходит на основе критерия Акаике. Он оценивает, насколько хорошо модель подходит под данные. И чем критерий меньше - тем точнее модель.

Пример использования ARIMA

```
# Импорт библиотек
import statsmodels.api as sm

# Отключим предупреждения системы
import warnings
warnings.simplefilter(action = 'ignore', category = Warning)
```

Чтение данных и начальное преобразование

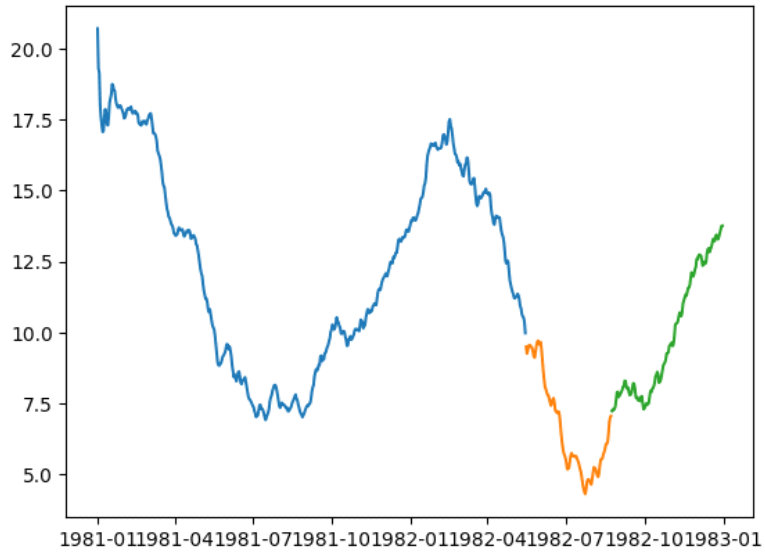
```
# Чтение файла
df = pd.read_csv('Temp.csv')
# Преобразуем дату к виду date-time
df.Date = pd.to_datetime(df.Date, infer_datetime_format=True)
# Переносим преобразованную дату в индекс
df.set_index("Date", inplace = True)
df.head(5)
```

Выделяем тренировочные и тестовые данные

```
train = df[:500]
val = df[500:600]
test = df[600:]
```

```
# Визуализируем для наглядности
plt.plot(train)
plt.plot(val)
plt.plot(test)
```

```
[<matplotlib.lines.Line2D at 0x7ddcb42d3760>]
```

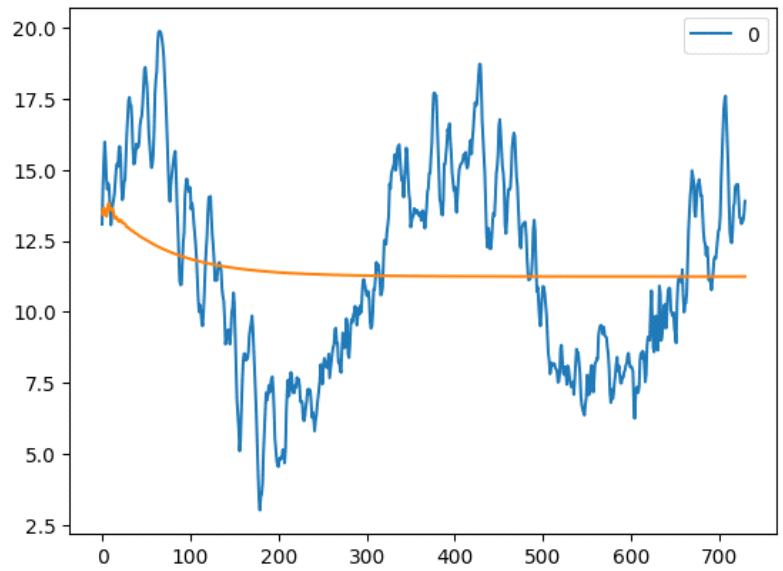



```
# Определение модели с заданными гиперпараметрами и её обучение
model = sm.tsa.ARIMA(train_data, order=(9,0,1)).fit()

# Для валидации определим границы валидационной выборки
start = len(train)
end = len(train) + len(test) -1

# Получаем результат прогнозирования
pred = model.predict(start, end)
```

Здесь надо отметить, что предсказанные данные уже не имеют привязки к индексам-датам, поэтому нельзя для сравнения визуализировать в чистом виде предсказанные данные и их валидационный исходник. Поэтому из обоих наборов данных извлечём только значения, без индексов, и визуализируем их.

```
# Визуализация
import seaborn as sns
sns.lineplot(data = test_data.values)
sns.lineplot(data = pred.values)
```

```
<Axes: >
```

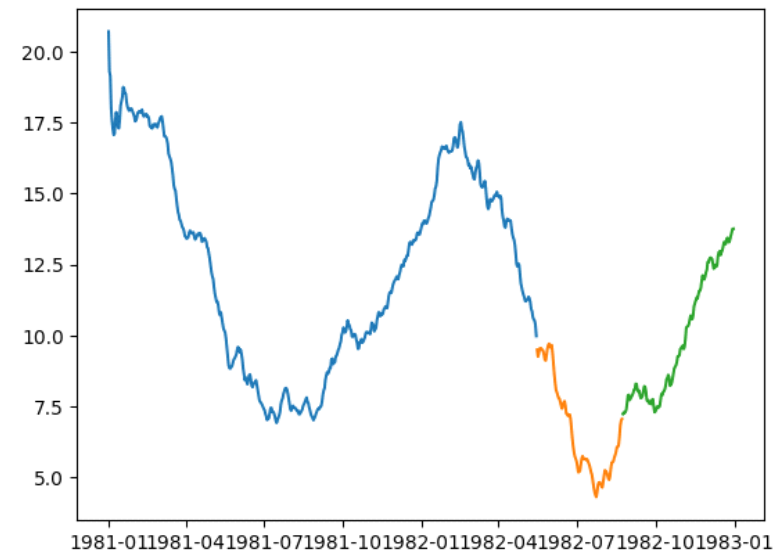



```
# Расчёт метрики
r2 = r2_score(test_data.values, pred.values)
print('Метрика R2 = ', r2)
```

Пример использования Auto-arima

```
# Импорт основной библиотеки
from pmdarima.arima import auto_arima

# Выделяем тренировочные и тестовые
train = df[:500]
val = df[500:600]
test = df[600:]

# Визуализируем для наглядности
plt.plot(train)
plt.plot(val)
plt.plot(test)
```

```
[<matplotlib.lines.Line2D at 0x7ddcb42d3760>]
```



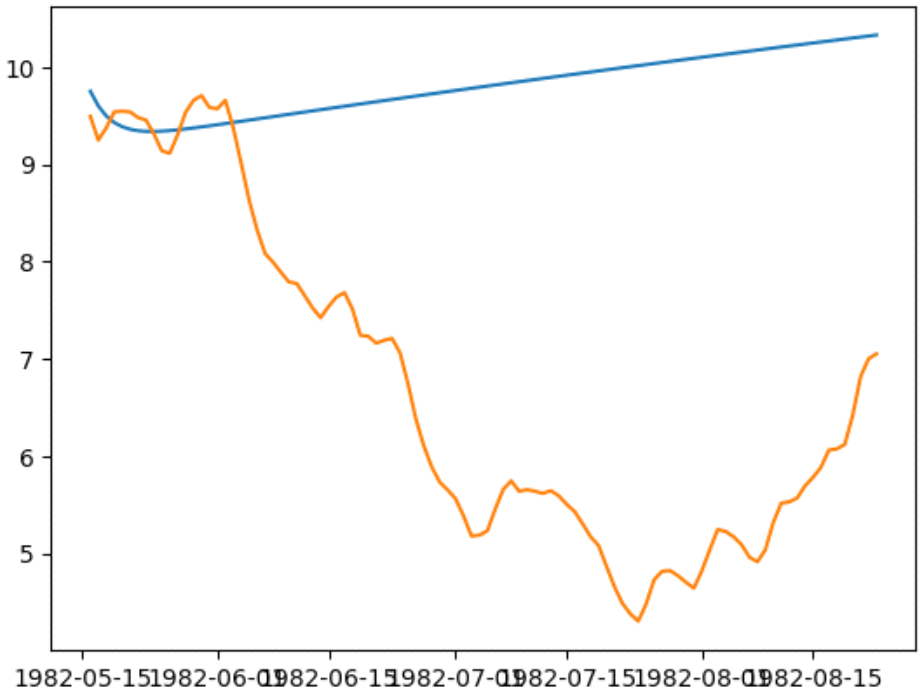

```
# Задаём модель и её гиперпараметры
# Тренировочный набор, начальные и максимальные значения основных параметров,
m - период сезонности (4 - квартальная, 12- ежемесячная, 1 - ежегодная, т.е. сезонности нет),
seasonal - есть ли сезонность в данных, trace - нужен ли вывод результатов подбора
```

```
arima_model =  auto_arima(train, start_p=0, d=0, start_q=0,
                          max_p=8, max_d=5, max_q=8, start_P=0,
                          D=0, start_Q=0, max_P=8, max_D=5,
                          max_Q=8, m=12, seasonal=True,
                          error_action='warn', trace = True,
                          supress_warnings=True, stepwise = True)
```

```
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=2687.482, Time=0.02 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=inf, Time=2.24 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=inf, Time=1.04 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=3983.897, Time=0.03 sec
 ARIMA(0,0,0)(1,0,0)[12] intercept   : AIC=inf, Time=1.26 sec
 ARIMA(0,0,0)(0,0,1)[12] intercept   : AIC=inf, Time=1.03 sec
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=1350.582, Time=5.15 sec
 ARIMA(0,0,0)(2,0,1)[12] intercept   : AIC=inf, Time=11.60 sec
 ARIMA(0,0,0)(1,0,2)[12] intercept   : AIC=1317.859, Time=7.44 sec
 ARIMA(0,0,0)(0,0,2)[12] intercept   : AIC=inf, Time=1.77 sec
```

```
# Делаем предсказание
prediction = pd.DataFrame(arima_model.predict(n_periods = 100), index=val.index)
prediction.columns = ['Temp']

# Построим предсказанные и фактические результаты
plt.plot(prediction)
plt.plot(val)
```

```
[<matplotlib.lines.Line2D at 0x7ddcb4377d90>]
```




```
# Считаем метрику
r2_score(val, prediction)
```

Проверка оптимальности модели

Проверить оптимальность модели можно при помощи встроенных процедур. С помощью метода summary() можно получить следующую таблицу.

Здесь больше всего нас интересует таблица коэффициентов. Столбец coef показывает влияние каждого параметра на временной ряд, а P>|z| — значимость. Чем ближе значение P>|z| к нулю, тем

выше значимость.

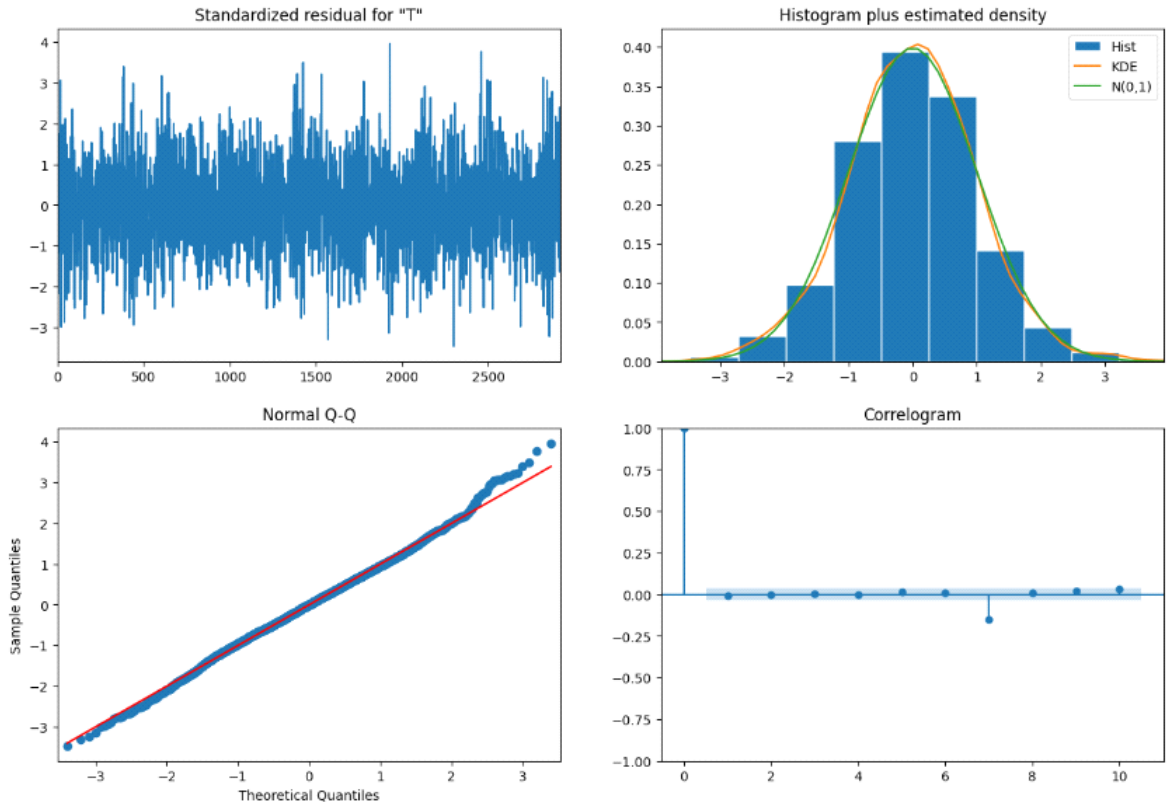
```
print(model.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                   Temp   No. Observations:                 2920
Model:                 ARIMA(9, 0, 1)   Log Likelihood               -1601.040
Date:                Fri, 26 Jan 2024   AIC                           3226.080
Time:                        07:57:05   BIC                           3297.832
Sample:                             0   HQIC                          3251.925
                                - 2920
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         11.2418      0.674     16.670      0.000       9.920      12.564
ar.L1          1.3856      0.069     19.957      0.000       1.250       1.522
ar.L2         -0.4193      0.100     -4.206      0.000      -0.615      -0.224
ar.L3          0.0470      0.045      1.055      0.291      -0.040       0.134
ar.L4          0.0022      0.029      0.075      0.940      -0.055       0.059
ar.L5         -0.0108      0.027     -0.395      0.693      -0.065       0.043
ar.L6         -0.0020      0.027     -0.075      0.940      -0.055       0.051
ar.L7         -0.4936      0.027    -18.541      0.000      -0.546      -0.441
ar.L8          0.6827      0.039     17.714      0.000       0.607       0.758
ar.L9         -0.2038      0.032     -6.316      0.000      -0.267      -0.141
ma.L1          0.1426      0.070      2.030      0.042       0.005       0.280
sigma2         0.1749      0.004     41.490      0.000       0.167       0.183
===================================================================================
Ljung-Box (L1) (Q):                   0.08   Jarque-Bera (JB):                20.92
Prob(Q):                              0.78   Prob(JB):                         0.00
Heteroskedasticity (H):               0.98   Skew:                             0.02
Prob(H) (two-sided):                  0.75   Kurtosis:                         3.41
===================================================================================

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

Другой метод для оценки модели - plot_diagnostics(), с его помощью мы получаем 4 графика, которые говорят об остатках модели (то есть о тех данных, которые не имели вклад в обучение)

Модель работает достаточно хорошо, если остатки модели некоррелированы и распределяются с нулевым средним значением

```
model.plot_diagnostics(figsize=(15, 10))
plt.show()
```



Рассмотрим каждый из графиков:

- График стандартизированных остатков - показывает линейный график остатков, модель обучилась хорошо, если остатки колеблются около нуля и в их графике не наблюдается закономерностей.

- Гистограмма остатков - позволяет сравнить распределение остатков со стандартным нормальным распределением. Видимо, чем ближе распределение остатков к нормальному распределению, тем лучше.

- График зависимости теоретических квантилей от фактических, чем больше теснота связи, то есть чем лучше построенный график совпадает с красной линией, тем лучше.

- Коррелограмма - показывает автокорреляцию остатков, в идеале корреляции остатков не должно быть вовсе.