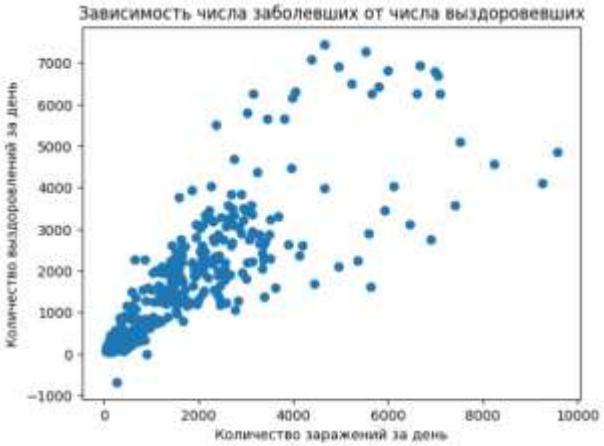
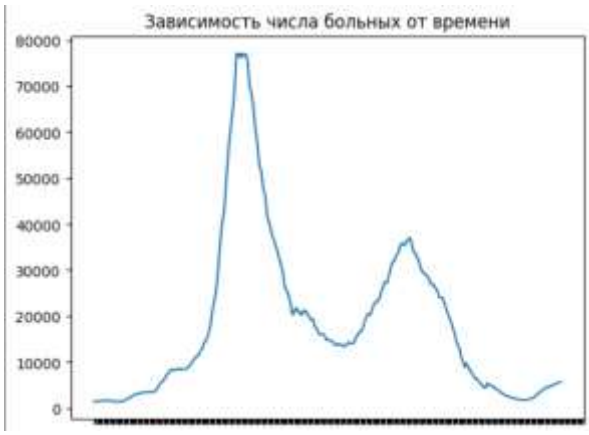
**Временной ряд (time series)** - это данные, последовательно собранные в регулярные промежутки времени. Каждое измерение в временном ряду связано с определенным моментом времени.

Рассмотрим на примере данных  о заболеваемости ковидом в Австрии:




1. перекрестные данные
2. временной ряд

Основное отличие между этими двумя типами данных заключается в зависимости между наблюдениями. Временные ряды имеют временную структуру, где значения зависят от предыдущих значений, в то время как в структурных/перекрестных данных наблюдения считаются независимыми.

При работе с временными рядами обычно выделяют два основных аспекта:

1. Анализ временного ряда:

Цель анализа временного ряда - понять свойства исследуемого процесса, его зависимость от времени

2. Моделирование и прогнозирование временного ряда:

Этот аспект включает в себя создание математических моделей, которые описывают поведение временного ряда в прошлом, чтобы предсказать его поведение в будущем. Для построения прогноза часто используются методы статистики и машинного обучения.
Основной задачей моделирования и прогнозирования является предсказание будущих значений временного ряда на основе его исторических данных.

**Терминология**

<u>Периоды:</u>

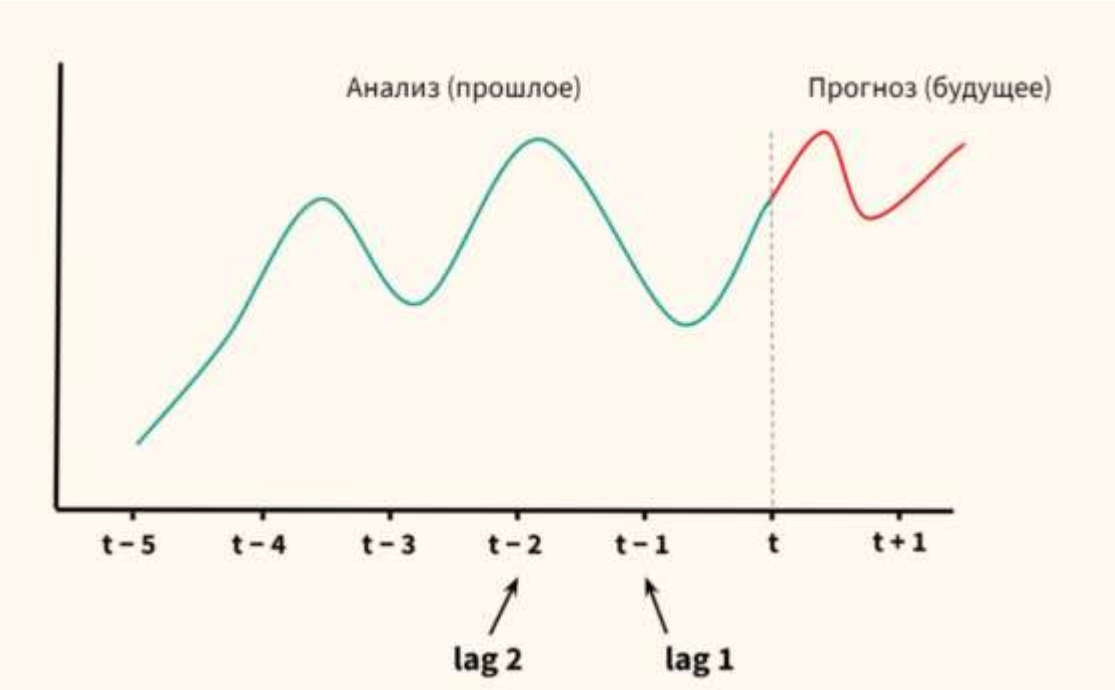
Период t обозначает текущий момент времени.

t−1, t−2,… представляют прошлые моменты времени.

t+1, t+2,… представляют будущие моменты времени.

Временной лаг:

Временной лаг представляет собой задержку или отставание в данных по сравнению с определенным периодом времени.
Например, временной лаг t-1 означает данные, которые предшествуют текущему моменту времени t на один период.




Для начала импортируем необходимые библиотеки:

```
import pandas as pd
import matplotlib.pyplot as plt
```

Далее импортируем данные:

```
data = pd.read_excel('Австрия.xlsx')
data.head()
```

| | Страна | Дата | Заражений | Выздоровлений | Смертей | Заражений за день | Выздоровлений за день | Смертей за день | Население страны | Тестов | Тестов за день |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Австрия | 20.07.2020 | 19743 | 17659.0 | 711 | 88 | 60.0 | 0 | 8725931 | 805600 | 7164 |
| 1 | Австрия | 21.07.2020 | 19827 | 17716.0 | 710 | 84 | 57.0 | -1 | 8725931 | 814681 | 9081 |
| 2 | Австрия | 22.07.2020 | 19929 | 17849.0 | 711 | 102 | 133.0 | 1 | 8725931 | 826031 | 11350 |
| 3 | Австрия | 23.07.2020 | 20099 | 17943.0 | 711 | 170 | 94.0 | 0 | 8725931 | 833841 | 7810 |
| 4 | Австрия | 24.07.2020 | 20214 | 18042.0 | 711 | 115 | 99.0 | 0 | 8725931 | 840890 | 7049 |

Вычислим общее число больных, вычитая из общего числа заражений количество выздоровлений и количество смертей.

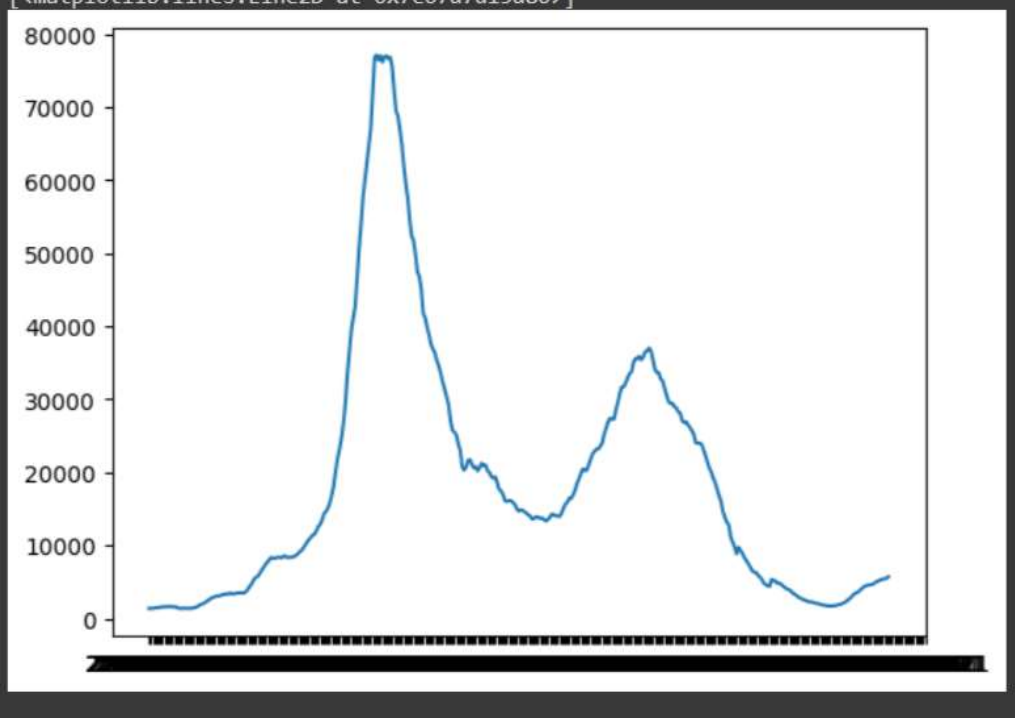
```
[4] data['Больных'] = data['Заражений'] - data['Выздоровлений'] - data['Смертей']
    df = data[['Дата', 'Больных']].copy()
    df.head()
```

| | Дата | Больных |
|---|---|---|
| 0 | 20.07.2020 | 1373.0 |
| 1 | 21.07.2020 | 1401.0 |
| 2 | 22.07.2020 | 1369.0 |
| 3 | 23.07.2020 | 1445.0 |
| 4 | 24.07.2020 | 1461.0 |

Сделаем дату индексом (это позволит упростить анализ и обработку данных) и построим зависимость количества больных от периода их заражения. pd.set_index()

```
df.set_index('Дата', inplace = True)
plt.plot(df)
```

```
[<matplotlib.lines.Line2D at 0x7e07d7d15d80>]
```




Питон воспринимает дату как число. Это не очень удобно, если мы хотим делать срезы и в целом изменять данные во времени. Дату можно преобразовать в специальный объект datetime. (с помощью функции pd.to_datetime())

Превращаем наш индекс в объект datetime и посмотрим на первые пять дат и на тип данных.

```
df.index = pd.to_datetime(df.index)
df.index[5:]
```

```
<ipython-input-9-7bf5e7e74a09>:1: UserWarning: Parsing dates in %d.%m.%Y format when dayfirst=False (the default) was specified. Pass
  df.index = pd.to_datetime(df.index)
DatetimeIndex(['2020-07-25', '2020-07-26', '2020-07-27', '2020-07-28',
               '2020-07-29', '2020-07-30', '2020-07-31', '2020-08-01',
               '2020-08-02', '2020-08-03',
               ...
               '2022-03-11', '2022-03-12', '2022-03-13', '2022-03-14',
               '2022-03-15', '2022-03-16', '2022-03-17', '2022-03-18',
               '2022-03-19', '2022-03-20'],
              dtype='datetime64[ns]', name='Дата', length=604, freq=None)
```

**Изменение шага временного ряда, сдвиг и скользящее среднее**

Мы можем изменить шаг (resample) нашего временного ряда, и посмотреть средние показатели, например, за месяц или год.

pd.resample() в библиотеке pandas используется для изменения частоты выборки временного ряда. Это позволяет агрегировать или изменять данные на основе новой частоты временного ряда. В основном, pd.resample() используется совместно с методами агрегации, такими как sum(), mean(), max(), min() и другими.

```
df_sum = df.resample('M').sum()
df_sum.head()
```

| Дата | Больных |
|---|---|
| 2020-07-31 | 18155.0 |
| 2020-08-31 | 68442.0 |
| 2020-09-30 | 184969.0 |
| 2020-10-31 | 489474.0 |
| 2020-11-30 | 1947070.0 |

получили сумму больных за каждый месяц

Кроме того, мы можем сдвинуть (shift) наши данные на n периодов вперед или назад. pd.shift() - основные передаваемые параметры:

- periods - по умолчанию = 1, количество периодов для сдвига. Может быть положительным или отрицательным

- axis - {0 or ‘index’, 1 or ‘columns’, None}, default None - Направление сдвига

```
# произведем сдвиг на два периода вперед
df.shift(2, axis = 0).head()
```

| Дата | Больных | Exp_smoothing |
|---|---|---|
| 2020-07-20 | NaN | NaN |
| 2020-07-21 | NaN | NaN |
| 2020-07-22 | 1373.0 | 1373.00 |
| 2020-07-23 | 1401.0 | 1378.60 |
| 2020-07-24 | 1369.0 | 1376.68 |

Первые два значения определяются как пропущенные (NaN или Not a number)

Мы также можем рассчитать скользящее среднее (moving average, rolling average) за n предыдущих периодов.

Метод скользящего среднего (Moving Average) — это статистический метод анализа временных рядов, который используется для сглаживания данных и выявления трендов. Он заключается в вычислении среднего значения последовательных подотрезков данных определенной длины, который затем перемещается (скользит) по временному ряду


Теперь рассчитаем его для нашего датасета
pd.rolling() - создаёт скользящее окно, которое перемещается по временному ряду или другому набору данных с одним шагом за раз.

Передаваемые параметры:

- window - размер скользящего окна

Для каждого положения окна вычисляется статистика, например, среднее, медиана, сумма и т. д.
При использовании метода rolling() нужно указать размер окна с помощью параметра window. Затем можно применять различные функции к этому скользящему окну, например, mean() для вычисления скользящего среднего для конкретного элемента.

head() выводит первые 5 строк получившейся таблицы

```
df.rolling(window = 3).mean().head()
```

| Дата | Больных |
|---|---|
| 2020-07-20 | NaN |
| 2020-07-21 | NaN |
| 2020-07-22 | 1381.0 |
| 2020-07-23 | 1405.0 |
| 2020-07-24 | 1425.0 |

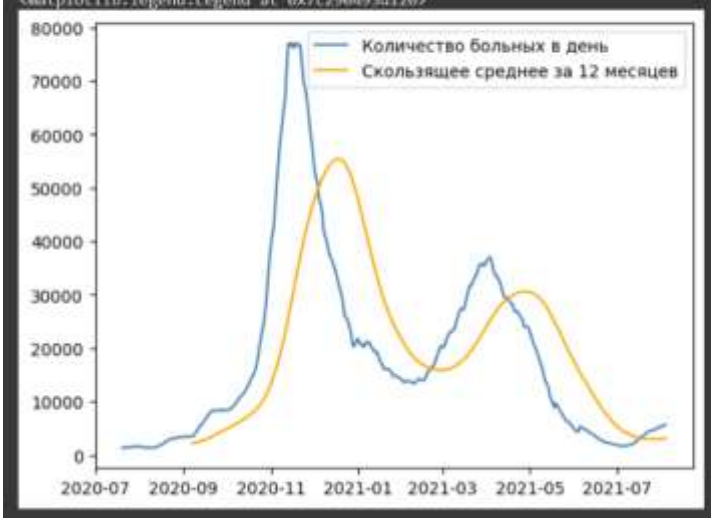
Период, за который рассчитывается скользящее среднее, также называется окном (window)

Визуализируем:

Для того чтобы построить график временного ряда мы можем воспользоваться инструментами, которые уже содержатся в библиотеке Pandas. Воспользуемся методом plot() и немного усложним наш график:

```
plt.plot(df, label = 'Количество больных в день', color = 'steelblue')
plt.plot(df.rolling(window = 50).mean(), label = 'Скользящее среднее за 12 месяцев', color = 'orange')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7c290493d120>
```




Подробнее про построение графиков на Python:
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

Как мы видим, скользящее среднее сильно сглаживает показатели, особенно это видно, если используется большая длина окна.

**Разложение временного ряда на компоненты:**

Выявление компонентов временного ряда, или анализ временных рядов на компоненты (time series decomposition), предполагает разложение ряда на основные составляющие, такие как тренд, сезонность и случайные колебания.

Тренд: Тренд представляет собой долгосрочное изменение уровня ряда. Это может быть рост или спад в течение длительного временного периода.

Сезонность: Сезонность подразумевает циклические изменения уровня ряда с постоянным периодом, который обычно связан с определенным временем года, месяцем или днем недели. Например, сезонность может проявляться в ежегодных циклических колебаниях продаж товаров к праздникам.

Случайные колебания: Случайные колебания представляют собой непрогнозируемое случайное изменение ряда, которое не подчиняется никакому определенному закону или шаблону. Эти колебания могут быть вызваны случайными факторами или внешними воздействиями.

Процесс декомпозиции временного ряда обычно осуществляется с использованием одного из двух основных методов: аддитивного или мультипликативного.

Аддитивная декомпозиция:

Предполагает, что изменение во времени является суммой тренда, сезонности и остатка.
Тренд добавляется к сезонной компоненте и остатку для получения оригинального временного ряда.
Формально это выглядит как: $x_t = T_t + S_t + E_t$, где
$x_t$ - исходное значение временного ряда в момент времени t,
$T_t$ - тренд в момент времени t,
$S_t$ - сезонная компонента в момент времени t,
$E_t$ - остаточная компонента (или ошибка) в момент времени t.

Мультипликативная декомпозиция:

Предполагает, что изменение во времени является произведением тренда, сезонности и остатка.
Тренд умножается на сезонную компоненту и на остаток для получения оригинального временного ряда.
Формально это выглядит как:
$x_t = T_t * S_t * E_t$, где
$x_t$ - исходное значение временного ряда в момент времени t,
$T_t$ - тренд в момент времени t,
$S_t$ - сезонная компонента в момент времени t,
$E_t$ - остаточная компонента (или ошибка) в момент времени t.

Процесс декомпозиции может быть реализован с использованием различных методов, таких как скользящее среднее, экспоненциальное сглаживание и другие. После того как тренд и сезонность были оценены, остаток вычисляется путем вычитания тренда и сезонной компоненты из оригинального временного ряда.

В библиотеке pandas функция pd.seasonal_decompose() реализует эти концепции, пытаясь разложить временной ряд на трендовую, сезонную и остаточную компоненты с учетом выбранной модели (аддитивной или мультипликативной). Это делается путем применения соответствующих методов оценки тренда и сезонности к исходному временному ряду.

**Вот как работает эта функция:**

Входные параметры:

- *x:* временной ряд, который вы хотите проанализировать.
- *model:* модель декомпозиции. По умолчанию 'additive', но также может быть 'multiplicative'.
- *filt*: длина сезонного фильтра. Если не указано, используется 13 для моделирования сезонных колебаний.
- *period:* длина сезонного периода. Если не указано, функция попытается автоматически обнаружить период.

Процесс декомпозиции:

- *Тренд:* Оценивается общий тренд данных. Это может быть линейным или нелинейным.
- *Сезонность:* Выявляются сезонные колебания, которые повторяются через определенные интервалы времени.
- *Остаток (или ошибка)*: Оставшаяся часть, которая не объясняется ни трендом, ни сезонностью. Это то, что остается после удаления тренда и сезонности из исходных данных.

Выходные данные:

- *trend*: трендовая компонента временного ряда.
- *seasonal:* сезонная компонента временного ряда.
- *residual:* остаточная (ошибка) компонента временного ряда.

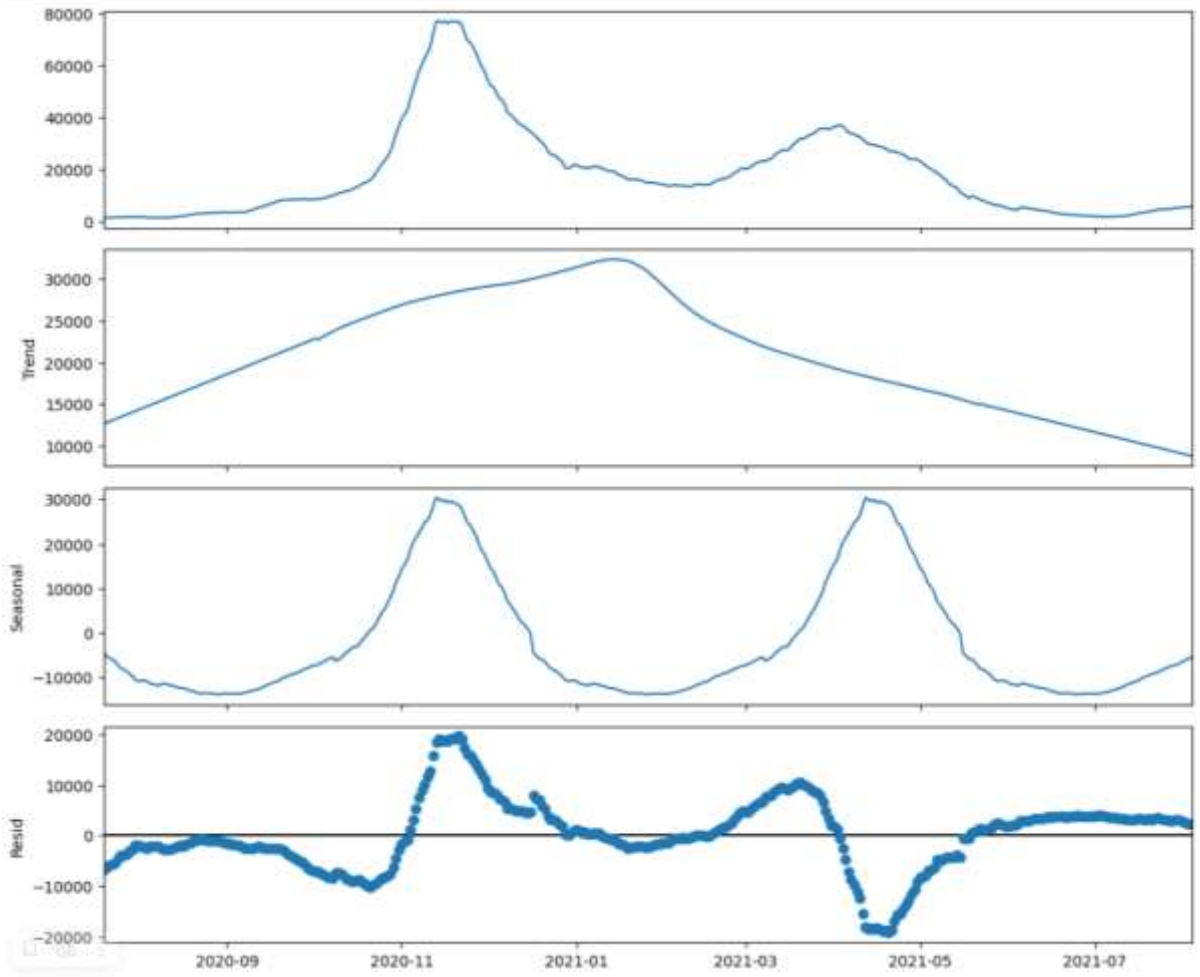
Разложим наш временной ряд на компоненты:

```
# импортируем функцию seasonal_decompose из statsmodels
from statsmodels.tsa.seasonal import seasonal_decompose

# задаем размер графика
from pylab import rcParams
rcParams['figure.figsize'] = 11, 9

# применяем функцию к данным
decompose = seasonal_decompose(df, extrapolate_trend = 30, period=150)
decompose.plot()

plt.show()
```


80000
60000
40000
20000
0
Trend
30000
25000
20000
15000
10000
Seasonal
30000
20000
10000
0
−10000
Resid
20000
10000
0
−10000
−20000
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07