

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №4
дисциплины
«Искусственный интеллект и машинное обучение»
Вариант 12

Выполнил:
Рябинин Егор Алексеевич
2 курс, группа ИВТ-б-о-23-2,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Проверил:
Доцент департамента цифровых,
робототехнических систем и
электроники института перспективной
инженерии
Воронкин Роман Александрович

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2025 г

Тема: Введение в pandas: изучение структуры Series и базовых операций.

Цель: познакомить с основами работы с библиотекой pandas, в частности, со структурой данных Series.

Порядок выполнения работы:

Ссылка на репозиторий GitHub:

https://github.com/EgorGorilla/Lab4_Artificial-Intelligence-and-Machine-Learning

1. Задания практической работы.

✓ **Задание №1. Создание Series из списка**

Создайте Series из списка чисел [5, 15, 25, 35, 45] с индексами ['a', 'b', 'c', 'd', 'e']. Выведите его на экран и определите его тип данных.

```
✓ [219] import numpy as np
0      import pandas as pd
сек
arr = pd.Series([5, 15, 25, 35, 45], ['a', 'b', 'c', 'd', 'e'])
print("Массив:\n", arr)
print("\nТип данных:", arr.dtype)
```

Рисунок 1 – Задание №1. Создание Series из списка

```
⇒ Массив:
a      5
b     15
c     25
d     35
e     45
dtype: int64

Тип данных: int64
```

Рисунок 2 – Результат работы программы к заданию №1

✓ **Задание №2. Получение элемента Series**

Дан Series с индексами ['A', 'B', 'C', 'D', 'E'] и значениями [12, 24, 36, 48, 60]. Используйте .loc[] для получения элемента с индексом 'C' и .iloc[] для получения третьего элемента.

```
✓ [43] A = pd.Series([12,24,36,48,60], ['A', 'B', 'C', 'D', 'E'])
0      print("Элемент с индексом 'C':", A.loc['C'])
сек      print("\nТретий элемент:", A.iloc[2])
```

Рисунок 3 – Задание №2. Получение элемента Series

```
⇒ Элемент с индексом 'C': 36

Третий элемент: 36
```

Рисунок 4 – Результат работы программы к заданию №2

✓ Задание №3. Фильтрация данных с помощью логической индексации

Создайте Series из массива NumPy `np.array([4, 9, 16, 25, 36, 49, 60])`. Выберите только те элементы массива, которые больше 20, и выведите результат.

```
✓ 0 сек.
▶ B = np.array([4, 9, 16, 25, 36, 48, 60])
  K = pd.Series(B)
  print("Исходный массив:\n",K)
  F = K[K>20]
  print("\nМассив с элементами больше 20:\n",F)
```

Рисунок 5 – Задание №3. Фильтрация данных с помощью логической индексации

```
⇒ Исходный массив:
  0      4
  1      9
  2     16
  3     25
  4     36
  5     48
  6     60
dtype: int64

Массив с элементами больше 20:
  3     25
  4     36
  5     48
  6     60
dtype: int64
```

Рисунок 6 – Результат работы программы к заданию №3

✓ Задание №4. Просмотр первых и последних элементов

Создайте Series, содержащий 50 случайных чисел от 1 до 100 (используйте `np.random.randint`). Выведите первые 7 и последние 5 элементов с помощью `.head()` и `.tail()`.

```
✓ 0 сек.
[65] W = pd.Series(np.random.randint(1, 101, size=50))
      print("Первые 7 элементов:\n",W.head(7))
      print("\nПоследние 5 элементов:\n",W.tail(5))
```

Рисунок 7 – Задание №4. Просмотр первых и последних элементов

```

⇒ Первые 7 элементов:
  0    15
  1    71
  2    95
  3    48
  4    98
  5    75
  6    39
dtype: int64

Последние 5 элементов:
  45    4
  46   94
  47   92
  48   11
  49   88
dtype: int64

```

Рисунок 8 – Результат работы программы к заданию №4

✓ Задание №5. Определение типа данных Series

Создайте Series из списка ['cat', 'dog', 'rabbit', 'parrot', 'fish']. Определите тип данных с помощью .dtype, затем преобразуйте его в category с помощью .astype().

```

[69] animals = pd.Series(['cat', 'dog', 'rabbit', 'parrot', 'fish'])
      print("Тип данных до преобразования:", animals.dtype)
      animals = animals.astype('category')
      print("Тип данных после преобразования:", animals.dtype)

```

Рисунок 9 – Задание №5. Определение типа данных Series

```

⇒ Тип данных до преобразования: object
  Тип данных после преобразования: category

```

Рисунок 10 – Результат работы программы к заданию №5

✓ Задание №6. Проверка пропущенных значений

Создайте Series с данными [1.2, np.nan, 3.4, np.nan, 5.6, 6.8]. Напишите код, который проверяет, есть ли в Series пропущенные значения (NaN), и выведите индексы таких элементов.

```

✓ [100] nan = pd.Series([1.2, np.nan, 3.4, np.nan, 5.6, 6.8])
    0
    cec
      print("Исходный массив:\n",nan)
      filtered_nan = nan[nan.isna()].index
      print("\nИндексы пропущенных значений: ",list(filtered_nan))

```

Рисунок 11 – Задание №6. Проверка пропущенных значений

```

Исходный массив:
0    1.2
1    NaN
2    3.4
3    NaN
4    5.6
5    6.8
dtype: float64

Индексы пропущенных значений: [1, 3]

```

Рисунок 12 – Результат работы программы к заданию №6

✓ Задание №7. Заполнение пропущенных значений

Используйте Series из предыдущего задания и замените все NaN на среднее значение всех непустых элементов. Выведите результат.

```

nan = pd.Series([1.2, np.nan, 3.4, np.nan, 5.6, 6.8])
print("Исходный массив:\n",nan)
average = nan.mean()
print("\nСреднее значение всех непустых элементов: ",average)
filtered_nan = nan.fillna(average)
print("\nМассив с замененными NaN на среднее значение всех непустых элементов:\n",filtered_nan)

```

Рисунок 13 – Задание №7. Заполнение пропущенных значений

```

Исходный массив:
0    1.2
1    NaN
2    3.4
3    NaN
4    5.6
5    6.8
dtype: float64

Среднее значение всех непустых элементов: 4.25

Массив с замененными NaN на среднее значение всех непустых элементов:
0    1.20
1    4.25
2    3.40
3    4.25
4    5.60
5    6.80
dtype: float64

```

Рисунок 14 – Результат работы программы к заданию №7

✓ Задание №8. Арифметические операции с Series

Создайте два Series:

```

s1 = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
s2 = pd.Series([5, 15, 25, 35], index=['b', 'c', 'd', 'e'])

```

Выполните сложение s1 + s2. Объясните, почему в результате появляются NaN, и замените их на 0.

```

s1 = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
s2 = pd.Series([5, 15, 25, 35], index=['b', 'c', 'd', 'e'])
s3 = s1+s2
print("Массив после сложения элементов:\n",s3)
s4 = s3.fillna(0)
print("\nМассив после замены NaN на 0:\n", s4)
print("\nВ результате сложения мы получили NaN, т.к операция сложения выполняется по индексам, поскольку индекс 'a' и 'e' отсутствовали в первом и втором массиве соответственно, они получили значение NaN.")

```

Рисунок 15 – Задание №8. Арифметические операции с Series

```

➡ Массив после сложения элементов:
a      NaN
b    25.0
c    45.0
d    65.0
e      NaN
dtype: float64

Массив после замены NaN на 0:
a      0.0
b    25.0
c    45.0
d    65.0
e      0.0
dtype: float64

В результате сложения мы получили NaN, т.к операция сложения выполняется по индексам,
поскольку индекс 'a' и 'e' отсутствовали в первом и втором массиве соответственно, они получили значение NaN.

```

Рисунок 16 – Результат работы программы к заданию №8

✓ Задание №9. Применение функции к Series

Создайте Series из чисел [2, 4, 6, 8, 10]. Напишите код, который применяет к каждому элементу функцию вычисления квадратного корня с помощью `.apply(np.sqrt)`.

```

✓ [117] s = pd.Series([2, 4, 6, 8, 10])
      0 000 print("Исходный массив:\n",s)
      000 s_sqrt = s.apply(np.sqrt)
      000 print("\nМассив после вычисления квадратного корня для каждого элемента:\n",s_sqrt)

```

Рисунок 17 – Задание №9. Применение функции к Series

```

➡ Исходный массив:
0      2
1      4
2      6
3      8
4     10
dtype: int64

Массив после вычисления квадратного корня для каждого элемента:
0      1.414214
1      2.000000
2      2.449490
3      2.828427
4      3.162278
dtype: float64

```

Рисунок 18 – Результат работы программы к заданию №9

✓ Задание №10. Основные статистические методы

Создайте Series из 20 случайных чисел от 50 до 150 (используйте `np.random.randint`). Найдите сумму, среднее, минимальное и максимальное значение. Выведите также стандартное отклонение.

```

▶ a = pd.Series(np.random.randint(50, 151, size=20))
  print("Массив:\n",a)
  print("\nСумма элементов:", a.sum())
  print("\nСреднее значение:", a.mean())
  print("\nМинимальное значение:", a.min())
  print("\nМаксимальное значение:", a.max())
  print("\nСтандартное отклонение:", a.std())
  print("\n",a.describe())

```

Рисунок 19 – Задание №10. Основные статистические методы

```

➡ Массив:
0      148
1       62
2       57
3       60
4      121
5      111
6       85
7       78
8      144
9      111
10      57
11      90
12     145
13      70
14      88
15      61
16     125
17      80
18     147
19     104
dtype: int64

Сумма элементов: 1944

Среднее значение: 97.2

Минимальное значение: 57

Максимальное значение: 148

Стандартное отклонение: 32.57154069364689

```

Рисунок 20 – Результат работы программы к заданию №10

✓ Задание №11. Работа с временными рядами

Создайте Series, где индексами будут даты с 1 по 10 марта 2024 года (`pd.date_range(start='2024-03-01', periods=10, freq='D')`), а значениями - случайные числа от 10 до 100. Выберите данные за 5-8 марта.

```

✓ 0 сек. ▶ date = pd.date_range(start='2024-03-01', periods=10, freq='D')
values = np.random.randint(10, 101, size=10)
d = pd.Series(values, index=date)
date_new = d['2024-03-05':'2024-03-08']
print("Исходные данные:\n", d)
print("\nДанные за 5-8 марта:\n", date_new)

```

Рисунок 21 – Задание №11. Работа с временными рядами

```

➡ Исходные данные:
2024-03-01    70
2024-03-02    10
2024-03-03    35
2024-03-04   100
2024-03-05    18
2024-03-06    61
2024-03-07    16
2024-03-08    16
2024-03-09    97
2024-03-10    50
Freq: D, dtype: int64

Данные за 5-8 марта:
2024-03-05    18
2024-03-06    61
2024-03-07    16
2024-03-08    16
Freq: D, dtype: int64

```

Рисунок 22 – Результат работы программы к заданию №11

✓ Задание №12. Проверка уникальности индексов

Создайте Series с индексами ['A', 'B', 'A', 'C', 'D', 'B'] и значениями [10, 20, 30, 40, 50, 60]. Проверьте, являются ли индексы уникальными. Если нет, сгруппируйте повторяющиеся индексы и сложите их значения.

```
✓ [145] s = pd.Series([10, 20, 30, 40, 50, 60], ['A', 'B', 'A', 'C', 'D', 'B'])
сек.    print("Исходный массив:\n",s)
        print("\nПроверка уникальности индексов:",s.index.is_unique)
        s = s.groupby(level=0).sum()
        print("\nМассив после группировки:\n",s)
```

Рисунок 23 – Задание №12. Проверка уникальности индексов

```
⇒ Исходный массив:
  A    10
  B    20
  A    30
  C    40
  D    50
  B    60
dtype: int64

Проверка уникальности индексов: False

Массив после группировки:
  A    40
  B    80
  C    40
  D    50
dtype: int64
```

Рисунок 24 – Результат работы программы к заданию №12

✓ Задание №13. Преобразование строковых дат в DatetimeIndex

Создайте Series, где индексами будут строки ['2024-03-10', '2024-03-11', '2024-03-12'], а значениями [100, 200, 300]. Преобразуйте индексы в DatetimeIndex и выведите тип данных индекса.

```
✓ [157] s = pd.Series([100, 200, 300], index=['2024-03-10', '2024-03-11', '2024-03-12'])
сек.    print("Тип данных индекса до преобразования:",s.index.dtype)
        s.index = pd.to_datetime(s.index)
        print("\nТип данных индекса после преобразования:",s.index.dtype)
```

Рисунок 25 – Задание №13. Преобразование строковых дат в DatetimeIndex

```
⇒ Тип данных индекса до преобразования: object

Тип данных индекса после преобразования: datetime64[ns]
```

Рисунок 26 – Результат работы программы к заданию №13

✓ Задание №14. Чтение данных из CSV-файла

Создайте CSV-файл data.csv со следующими данными:

1 Дата, Цена

2 2024-03-01, 100

3 2024-03-02, 110

4 2024-03-03, 105

5 2024-03-04, 120

6 2024-03-05, 115

Прочитайте файл и создайте Series, используя "Дата" в качестве индекса.

```
✓ [218] data = {  
0      'Дата': ['2024-03-01', '2024-03-02', '2024-03-03', '2024-03-04', '2024-03-05'],  
сек    'Цена': [100, 110, 105, 120, 115]  
      }  
      df = pd.DataFrame(data)  
      df.to_csv('data.csv', index=False)  
      df = pd.read_csv('data.csv', parse_dates=['Дата'])  
      s = pd.Series(df['Цена'].values, index=df['Дата'])  
      print(s)
```

Рисунок 27 – Задание №14. Чтение данных из CSV-файла

```
⇒ Дата  
2024-03-01    100  
2024-03-02    110  
2024-03-03    105  
2024-03-04    120  
2024-03-05    115  
dtype: int64
```

Рисунок 28 – Результат работы программы к заданию №14

✓ Задание №15. Построение графика на основе Series

Создайте Series, где индексами будут даты с 1 по 30 марта 2024 года, а значениями - случайные числа от 50 до 150. Постройте график значений с помощью matplotlib. Добавьте заголовок подписи осей и сетку

```
✓ [218] import matplotlib.pyplot as plt  
0      s = pd.Series(np.random.uniform(50,150,size=30),index=pd.date_range(start='2024-03-01', periods=30, freq='D'))  
сек    s.plot(kind='bar',color='red')  
      plt.title('Продажи (март 2025)')  
      plt.xlabel('Дата')  
      plt.ylabel('Выручка (тыс. рублей)')  
      plt.grid()  
      plt.show()
```

Рисунок 29 – Задание №15. Построение графика на основе Series

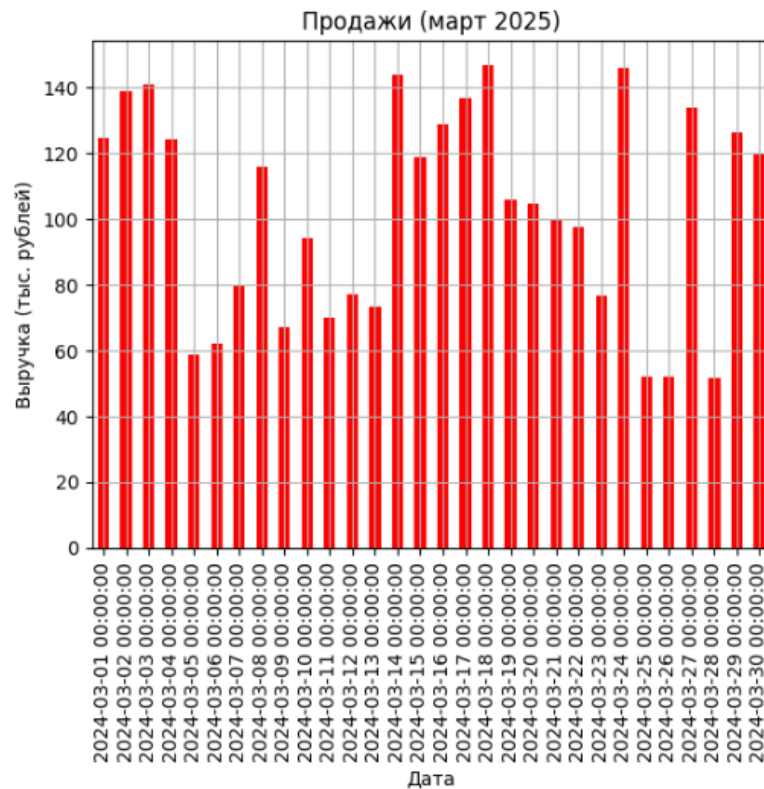


Рисунок 30 – Результат работы программы к заданию №15

2. Индивидуальное задание.

✓ Вариант 12. Динамика спроса на товар в интернет-магазине

Создайте CSV-файл demand.csv со следующими данными:

1 Дата, Спрос

2 2024-09-01, 120

3 2024-09-02, 135

4 2024-09-03, 150

5 2024-09-04, 170

6 2024-09-05, 160

Прочитайте файл, установите DatetimeIndex, вычислите скользящее среднее (rolling(3).mean()) и постройте два графика, фактического спроса и его сглаженного значения.

```
data = {
    'Дата': ['2024-09-01', '2024-09-02', '2024-09-03', '2024-09-04', '2024-09-05'],
    'Спрос': [120, 135, 150, 170, 160]
}
df = pd.DataFrame(data)
df.to_csv('demand.csv', index=False)
df = pd.read_csv('demand.csv', parse_dates=['Дата'], index_col='Дата')
df['Скользящее среднее'] = df['Спрос'].rolling(3).mean()
plt.figure(figsize=(8, 5))
plt.plot(df.index, df['Спрос'], marker='o', label='Фактический спрос', color='blue')
plt.plot(df.index, df['Скользящее среднее'], marker='s', linestyle='dashed', label='Сглаженный спрос', color='red')
plt.title('Спрос на товар (сентябрь 2024)')
plt.xlabel('Дата')
plt.ylabel('Спрос')
plt.legend()
plt.grid()
plt.xticks(rotation=45)
plt.show()
```

Рисунок 31 – Индивидуальное задание

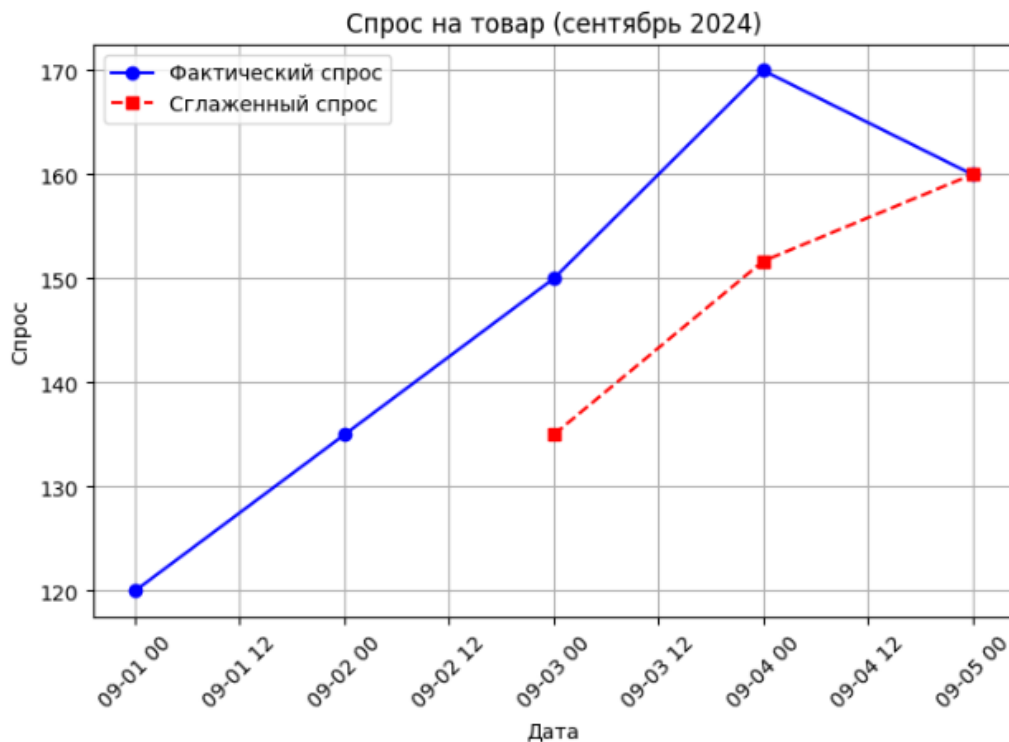


Рисунок 32 – Результат работы программы к индивидуальному заданию

Контрольные вопросы:

1. Что такое `pandas.Series` и чем она отличается от списка в Python?

- `pandas.Series` — это одномерный массив с метками (индексами), поддерживающий различные типы данных. В отличие от списка, `Series` обладает индексами, поддерживает векторные операции и работает быстрее.

2. Какие типы данных можно использовать для создания `Series`? - Числовые (`int`, `float`), строковые (`str`), булевы (`bool`), временные (`datetime`), объекты (`object`) и категории (`category`).

3. Как задать индексы при создании `Series`? - Указать параметр `index` при создании `Series`: `pd.Series(data, index=custom_index)`.

4. Каким образом можно обратиться к элементу `Series` по его индексу? - Использовать `series[index]` или `series.loc[index]`.

5. В чем разница между `.iloc[]` и `loc[]` при индексации `Series`? - `.iloc[]` работает с порядковыми номерами (0, 1, 2), а `.loc[]` использует заданные индексы.

6. Как использовать логическую индексацию в Series? - Применять условия: ``series[series > 10]`` вернет только элементы больше 10.

7. Какие методы можно использовать для просмотра первых и последних элементов Series? - ``series.head(n)`` и ``series.tail(n)``.

8. Как проверить тип данных элементов Series? - ``series.dtype``.

9. Каким способом можно изменить тип данных Series? - ``series.astype(new_type)``.

10. Как проверить наличие пропущенных значений в Series? - ``series.isna()`` или ``series.isnull()``.

11. Какие методы используются для заполнения пропущенных значений в Series? - ``series.fillna(value)``.

12. Чем отличается метод `.fillna()` от `.dropna()`? - `.fillna()` заменяет NaN значением, а `.dropna()` удаляет строки с NaN.

13. Какие математические операции можно выполнять с Series? - Сложение, вычитание, умножение, деление, логарифм, возведение в степень и др.

14. В чем преимущество векторизованных операций по сравнению с циклами Python? - Они работают быстрее, так как используют оптимизированные C-библиотеки.

15. Как применить пользовательскую функцию к каждому элементу Series? - Использовать ``series.apply(func)``.

16. Какие агрегирующие функции доступны в Series? - ``sum()``, ``mean()``, ``median()``, ``std()``, ``min()``, ``max()``.

17. Как узнать минимальное, максимальное, среднее и стандартное отклонение в Series? - ``series.min()``, ``series.max()``, ``series.mean()``, ``series.std()``.

18. Как сортировать Series по значениям и по индексам? - ``series.sort_values()`` для значений, ``series.sort_index()`` для индексов.

19. Как проверить, являются ли индексы Series уникальными? - ``series.index.is_unique``.

20. Как сбросить индексы Series и сделать их числовыми? -
``series.reset_index(drop=True)``.

21. Как можно задать новый индекс в Series? -
``series.set_index(new_index)``.

22. Как работать с временными рядами в Series? - Преобразовать индекс в ``DatetimeIndex`` и использовать ``resample()``, ``rolling()``.

23. Как преобразовать строковые даты в формат DatetimeIndex? -
``pd.to_datetime(series)``.

24. Каким образом можно выбрать данные за определенный временной диапазон? - ``series['2023-01-01':'2023-12-31']``.

25. Как загрузить данные из CSV-файла в Series? -
``pd.read_csv('file.csv', usecols=['column'], squeeze=True)``.

26. Как установить один из столбцов CSV-файла в качестве индекса Series? - ``pd.read_csv('file.csv', index_col='column')``.

27. Для чего используется метод `.rolling().mean()` в Series? - Для вычисления скользящего среднего.

28. Как работает метод `.pct_change()`? Какие задачи он решает? - Вычисляет процентное изменение между соседними значениями, полезно в анализе трендов.

29. В каких ситуациях полезно использовать `.rolling()` и `.pct_change()`? - ``rolling()`` для сглаживания данных, ``pct_change()`` для анализа динамики изменений.

30. Почему NaN могут появляться в Series, и как с ними работать? - NaN появляются из-за отсутствующих данных, преобразований типов, деления на 0. Удалять ``dropna()``, заполнять ``fillna()``.

Вывод: в ходе практической работы мы познакомились с основами работы с библиотекой pandas, в частности, со структурой данных Series.