

**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 лабораторной работе № 4  
«Работа с JupyterNotebook, JupyterLab, GoogleColab»  
по дисциплине «Искусственный интеллект и машинное обучение»**

Выполнил:

Евдокимов Станислав Алексеевич 2  
курс, группа ИВТ-б-о-23-1, 09.03.01  
«Информатика и вычислительная  
техника», направленность (профиль)  
«Автоматизированные системы  
обработки информации и  
управления», очная форма обучения

Руководитель практики:

Воронкин Роман Александрович

Ставрополь 2025

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

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

**Порядок выполнения лабораторной работы:**

**[StanislavEvdokimov/Lab4](#)**

**Задание 1. Установка pandas.**

```
[3]: !pip install pandas

Requirement already satisfied: pandas in c:\users\hesko\anaconda3\lib\site-packages (2.2.2)
Requirement already satisfied: numpy>=1.26.0 in c:\users\hesko\anaconda3\lib\site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\hesko\anaconda3\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\hesko\anaconda3\lib\site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\hesko\anaconda3\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\hesko\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)

[6]: import pandas as pd

[15]: import pandas as pd
      print(pd.__version__)

2.2.2
```

Рисунок 1 – Установка pandas

**Задание 2. Создание Series.**

```
[19]: In [3]: s1 = pd.Series([1, 2, 3, 4, 5])
      In [4]: print(s1)

      0    1
      1    2
      2    3
      3    4
      4    5
      dtype: int64

[21]: In [5]: s2 = pd.Series([1, 2, 3, 4, 5], ['a', 'b', 'c', 'd', 'e'])
      In [6]: print(s2)

      a    1
      b    2
      c    3
      d    4
      e    5
      dtype: int64

[27]: import numpy as np
      In [3]: ndarr = np.array([1, 2, 3, 4, 5])
      In [4]: type(ndarr)
      Out[4]: np.ndarray

[29]: In [5]: s3 = pd.Series(ndarr, ['a', 'b', 'c', 'd', 'e'])
      In [6]: print(s3)

      a    1
      b    2
      c    3
      d    4
      e    5
      dtype: int32

[31]: In [7]: d = {'a':1, 'b':2, 'c':3}
      In [8]: s4 = pd.Series(d)
      In [9]: print(s4)

      a    1
      b    2
      c    3
```

Рисунок 2 – Series

### Задание 3. Работа с элементами Series.

```

[39]: In [13]: s6 = pd.Series([1, 2, 3, 4, 5], ['a', 'b', 'c', 'd', 'e'])
      In [14]: s6.iloc[2]
      Out[14]: 3

[37]: In [15]: s6['d']
      Out[15]: 4

[51]: In [16]: s6[:2]

[51]: a    1
      b    2
      dtype: int64

[49]: In [17]: s6[s6 <= 3]

[49]: a    1
      b    2
      c    3
      dtype: int64

[53]: In [18]: s7 = pd.Series([10, 20, 30, 40, 50], ['a', 'b', 'c', 'd', 'e'])
      In [19]: s6 + s7

[53]: a    11
      b    22
      c    33
      d    44
      e    55
      dtype: int64

[55]: In [20]: s6 * 3

[55]: a    3
      b    6

```

Рисунок 3 – Работа с элементами Series

#### Задание 4. Позиционная индексация (iloc).

```
[57]: import pandas as pd
      # Создадим Series с пользовательскими индексами
      s = pd.Series([10, 20, 30, 40, 50], index=['a', 'b', 'c', 'd', 'e'])
      # Доступ к элементам по порядковому номеру
      print(s.iloc[0]) # Первый элемент (10)
      print(s.iloc[2]) # Третий элемент (30)
      print(s.iloc[-1]) # Последний элемент (50)

10
30
50

[59]: # Доступ к элементам по метке индекса
      print(s.loc['a']) # Первый элемент (10)
      print(s.loc['c']) # Третий элемент (30)
      print(s.loc['e']) # Последний элемент (50)

10
30
50

[67]: # Срез с использованием iloc (позиционный индекс, невключительный stop)
      print(s.iloc[1:3]) # Выведет элементы с индексами 1 и 2 (20, 30)
      # Срез с использованием loc (меточная индексация, включительный stop)
      print(s.loc['b':'d']) # Выведет элементы с индексами 'b', 'c' и 'd' (20, 30, 40)

b    20
c    30
dtype: int64
b    20
c    30
d    40
dtype: int64
```

Рисунок 4 – iloc

**Задание 5.** Использование логической индексации для фильтрации данных в Series.

```
[69]: import pandas as pd
      # Создаём Series с числовыми значениями
      s = pd.Series([10, 25, 8, 30, 15], index=['a', 'b', 'c', 'd', 'e'])
      # Фильтруем значения больше 10
      filtered_s = s[s > 10]
      print(filtered_s)

      b    25
      d    30
      e    15
      dtype: int64
```

```
[71]: print(s > 10)

      a    False
      b     True
      c    False
      d     True
      e     True
      dtype: bool
```

```
[73]: # Фильтрация значений в диапазоне от 10 до 30
      filtered_s = s[(s >= 10) & (s <= 30)]
      print(filtered_s)

      a    10
      b    25
      d    30
      e    15
      dtype: int64
```

```
[75]: # Выбираем только элементы, у которых индекс 'b' или 'd'
      filtered_s = s[s.index.isin(['b', 'd'])]
      print(filtered_s)

      b    25
      d    30
      dtype: int64
```

```
[77]: s_with_nan = pd.Series([10, None, 8, 30, None], index=['a', 'b', 'c', 'd', 'e'])
      # Фильтрация только непустых значений
      filtered_s = s_with_nan[s_with_nan.notnull()]
      print(filtered_s)

      a    10.0
      c     8.0
      d    30.0
      dtype: float64
```

Рисунок 5 – Логическая индексация

## Задание 6. Изменение значений элементов в Series.

```
[79]: import pandas as pd
      # Создаём Series
      s = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
      # Изменяем значение элемента с индексом 'b'
      s.loc['b'] = 25
      print(s)

a    10
b    25
c    30
d    40
dtype: int64
```

```
[81]: # Изменяем второй элемент (позиция 1)
      s.iloc[1] = 50
      print(s)

a    10
b    50
c    30
d    40
dtype: int64
```

```
[83]: # Увеличиваем все значения больше 30 на 10
      s[s > 30] += 10
      print(s)

a    10
b    60
c    30
d    50
dtype: int64
```

```
[85]: # Изменяем значения элементов с индексами 'a' и 'c'
      s.loc[['a', 'c']] = [100, 200]
      print(s)

a    100
b     60
c    200
d     50
dtype: int64
```

```
[87]: # Умножаем все элементы на 2
      s = s.apply(lambda x: x * 2)
      print(s)

a    200
b    120
c    400
d    100
dtype: int64
```

Рисунок 6 – Изменение значений

## Задание 7. Основные методы работы с Series.

```
[91]: s_with_nan = pd.Series([10, None, 30, None], index=['a', 'b', 'c', 'd'])
      # Заполняем пропущенные значения числом 0
      s_filled = s_with_nan.fillna(0)
      print(s_filled)

a    10.0
b     0.0
c    30.0
d     0.0
dtype: float64
```

```
[97]: import pandas as pd
      # Создаём Series
      s = pd.Series([10, 20, 30, 40, 50, 60, 70], index=['a', 'b', 'c', 'd', 'e', 'f', 'g'])
      # Вывод первых 3 элементов
      print(s.head(3))

a    10
b    20
c    30
dtype: int64
```

```
[99]: print(s.head())

a    10
b    20
c    30
d    40
e    50
dtype: int64
```

```
[101]: # Вывод последних 3 элементов
       print(s.tail(3))

e    50
f    60
g    70
dtype: int64
```

```
[103]: print(s.tail())

c    30
d    40
e    50
f    60
g    70
dtype: int64
```

Рисунок 7 – Методы работы с Series

## Задание 8. Операции над Series.



```
[139]: import pandas as pd
# Создаём Series
s = pd.Series([10, 20, 30, 40, 50])
# Умножаем все элементы на 2
s_multiplied = s * 2
print(s_multiplied)
```

```
0    20
1    40
2    60
3    80
4   100
dtype: int64
```

```
[141]: s1 = pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e'])
s2 = pd.Series([10, 20, 30, 40, 50], index=['a', 'b', 'c', 'd', 'e'])
# Поэлементное сложение двух Series
s_sum = s1 + s2
print(s_sum)
```

```
a    11
b    22
c    33
d    44
e    55
dtype: int64
```

```
[143]: s3 = pd.Series([100, 200, 300], index=['a', 'b', 'f'])
# Сложение Series с несовпадающими индексами
s_result = s1 + s3
print(s_result)
```

```
a    101.0
b    202.0
c      NaN
d      NaN
e      NaN
f      NaN
dtype: float64
```

Рисунок 8 – Операции над Series

## Задание 9. Совместимость с NumPy.

```
[171]: import pandas as pd
import numpy as np
# Создаём Series
s = pd.Series([1, 2, 3, 4, 5])
# Применяем натуральный логарифм
s_log = np.log(s)
print(s_log)

0    0.000000
1    0.693147
2    1.098612
3    1.386294
4    1.609438
dtype: float64

[173]: s_exp = np.exp(s)
print(s_exp)

0     2.718282
1     7.389056
2    20.085537
3    54.598150
4   148.413159
dtype: float64

[175]: s_sqrt = np.sqrt(s)
print(s_sqrt)

0    1.000000
1    1.414214
2    1.732051
3    2.000000
4    2.236068
dtype: float64

[177]: print(np.sin(s)) # Синус каждого элемента
print(np.cos(s)) # Косинус каждого элемента
print(np.abs(s)) # Модуль значений

0    0.841471
1    0.909297
2    0.141120
3   -0.756802
4   -0.958924
dtype: float64
0    0.540302
```

Рисунок 9 – Совместимость с NumPy

## Задание 10. Задание 1. Создание Series из списка

### 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.

```
[5]: import pandas as pd
s2 = pd.Series([5, 15, 25, 35, 45], ['a', 'b', 'c', 'd', 'e'])
print(s2)

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

Рисунок 10 – Создание Series из списка

## Задание 11. Задание 2. Получение элемента Series

## 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.

```
[19]: s = pd.Series([12, 24, 36, 48, 60], ['A', 'B', 'C', 'D', 'E'])
      e11 = s.loc['C']
      e12 = s.iloc[2]
      print("Исходный Series:")
      print(s)
      print("Элемент с индексом 'C':", e11)
      print("Третий элемент:", e12)
```

```
Исходный Series:
A    12
B    24
C    36
D    48
E    60
dtype: int64
Элемент с индексом 'C': 36
Третий элемент: 36
```

Рисунок 11 – Получение элемента Series

**Задание 12.** Задание 3. Фильтрация данных с помощью логической индексации

Создайте Series из массива NumPy `np.array([4, 9, 16, 25, 36, 49, 64])` . Выберите только те элементы, которые больше 20, и выведите результат.

```
[33]: s = pd.Series([4, 9, 16, 25, 36, 49, 64])
      print("Исходный Series:")
      print(s)
      filtered_s = s[s > 20]
      print("Фильтрованный Series:\n", filtered_s)
```

```
Исходный Series:
0     4
1     9
2    16
3    25
4    36
5    49
6    64
dtype: int64
Фильтрованный Series:
3     25
4     36
5     49
6     64
dtype: int64
```

Рисунок 12 – Фильтрация данных

**Задание 13.** Задание 4. Просмотр первых и последних элементов

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

```
[39]: import numpy as np
ser = pd.Series(np.random.randint(1, 101, size=50))
print("Первые 7 элементов:")
print(ser.head(7))
print("\nПоследние 5 элементов:")
print(ser.tail(5))
```

Первые 7 элементов:

0	93
1	57
2	75
3	45
4	43
5	57
6	93

dtype: int32

Последние 5 элементов:

45	16
46	33
47	75
48	19
49	56

dtype: int32

Рисунок 13 – Просмотр первых и последних элементов

### Задание 14. 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() .**

```
[47]: anim = pd.Series(['cat', 'dog', 'rabbit', 'parrot', 'fish'])
print("Исходный тип данных:", anim.dtype)
anim2 = anim.astype('category')
print("Преобразование series:")
print(anim2)
```

Исходный тип данных: object  
Преобразование series:

0	cat
1	dog
2	rabbit
3	parrot
4	fish

dtype: category  
Categories (5, object): ['cat', 'dog', 'fish', 'parrot', 'rabbit']

Рисунок 14 – Просмотр первых и последних элементов

### Задание 15. 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. .**

```
[65]: prop = pd.Series([1.2, np.nan, 3.4, np.nan, 5.6, 6.8])
nan = prop.isna().any()
ind = prop[prop.isna()].index.tolist()
print("Series:")
print(prop)
print("Есть ли пропущенные значения?", "Да" if nan else "Нет")
print("Индексы пропущенных значений:", ind)
```

Series:

0	1.2
1	NaN
2	3.4
3	NaN
4	5.6
5	6.8

dtype: float64  
Есть ли пропущенные значения? Да  
Индексы пропущенных значений: [1, 3]

Рисунок 15 – Просмотр первых и последних элементов

### Задание 16. Задание 7. Заполнение пропущенных значений

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

```
[71]: print("Исходный Series:")
      print(prop)
      sred = prop.mean()
      zam = prop.fillna(sred)
      print("Среднее значение:", sred)
      print("Результат:")
      print(zam)
```

```
Исходный Series:
0    1.2
1    NaN
2    3.4
3    NaN
4    5.6
5    6.8
dtype: float64
Среднее значение: 4.25
Результат:
0    1.20
1    4.25
2    3.40
3    4.25
4    5.60
5    6.80
dtype: float64
```

Рисунок 16 – Пропущенные значения

## Задание 17. 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

```
[77]: s1 = pd.Series([10, 20, 30, 40], ['a', 'b', 'c', 'd'])
      s2 = pd.Series([5, 15, 25, 35], ['b', 'c', 'd', 'e'])
      result = s1 + s2
      print(result)
```

```
a    NaN
b    25.0
c    45.0
d    65.0
e    NaN
dtype: float64
```

```
[79]: nul = result.fillna(0)
      print("Результат:")
      print(nul)
```

```
Результат:
a    0.0
b    25.0
c    45.0
d    65.0
e    0.0
dtype: float64
```

Рисунок 17 – Арифметические операции

## Задание 18. 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)` .

```
[83]: series = pd.Series([2, 4, 6, 8, 10])
sqr = series.apply(np.sqrt)
print("Исходный Series:")
print(series)
print("\nSeries с квадратными корнями:")
print(sqr)
```

Исходный Series:

```
0    2
1    4
2    6
3    8
4   10
dtype: int64
```

Series с квадратными корнями:

```
0    1.414214
1    2.000000
2    2.449490
3    2.828427
4    3.162278
dtype: float64
```

## Рисунок 18 – Применение функции к Series

### Задание 19. 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.

```
[90]: ran = pd.Series(np.random.randint(50, 151, size=20))
summa = ran.sum()
sred = ran.mean()
minim = ran.min()
maxim = ran.max()
otkl = ran.std()
print("Series:")
print(ran)
print("Показатели:")
print("Сумма", summa)
print("Среднее", sred)
print("Минимум", minim)
print("Максимум", maxim)
print("Стандартное отклонение", otkl)
```

Series:

```
0    136
1    141
2     64
3    133
4     82
5    120
6    142
7     74
8    130
9     54
10    142
11    104
12     88
13     62
14     77
15     65
16     84
17    118
18     88
19    104
dtype: int32
```

Показатели:

```
Сумма 2008
Среднее 100.4
Минимум 54
Максимум 142
Стандартное отклонение 30.262709378827406
```

## Рисунок 19 – Основные статистические методы

### Задание 20. 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.

```
[96]: data = pd.date_range(start='2024-03-01', periods=10, freq='D')
ind = np.random.randint(10, 101, size=10)
series = pd.Series(ind, data)
res = series['2024-03-05':'2024-03-08']
print("Исходный вариант:")
print(series)
print("Данные за 5-8 марта 2024 года:")
print(res)
```

```
Исходный вариант:
2024-03-01    52
2024-03-02    13
2024-03-03    99
2024-03-04    74
2024-03-05    63
2024-03-06    77
2024-03-07    96
2024-03-08    67
2024-03-09    75
2024-03-10    44
Freq: D, dtype: int32
Данные за 5-8 марта 2024 года:
2024-03-05    63
2024-03-06    77
2024-03-07    96
2024-03-08    67
```

## Рисунок 20 – Временные ряды

### Задание 21. 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.

```
]: s = pd.Series([10, 20, 30, 40, 50, 60], ['A', 'B', 'A', 'C', 'D', 'B'])
if s.index.has_duplicates:
    grouped_s = s.groupby(s.index).sum()
    print("Исходный Series:")
    print(s)
    print("Группировка:")
    print(grouped_s)
else:
    print("Все индексы уникальные")
```

```
Исходный Series:
A    10
B    20
A    30
C    40
D    50
B    60
dtype: int64
Группировка:
A    40
B    80
C    40
D    50
dtype: int64
```

## Рисунок 21 – Уникальность индексов

### Задание 22. 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.


```
120]: ser = pd.Series([100, 200, 300], ['2024-03-10', '2024-03-11', '2024-03-12'])
ser.index = pd.to_datetime(ser.index)
print("Series:")
print(ser)
print("Тип данных индекса:", type(ser.index))

Series:
2024-03-10    100
2024-03-11    200
2024-03-12    300
dtype: int64
Тип данных индекса: <class 'pandas.core.indexes.datetimes.DatetimeIndex'>
```

Рисунок 22 –DatetimeIndex

## Задание 23. 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

```
.60]: data = """Дата,Цена
2024-03-01,100
2024-03-02,110
2024-03-03,105
2024-03-04,120
2024-03-05,115"""
with open('data.csv', 'w', encoding='utf-8') as file:
    file.write(data)
seeries = pd.read_csv('data.csv', index_col="Дата")
print("Series:")
print(seeries)

Series:
          Цена
Дата
2024-03-01  100
2024-03-02  110
2024-03-03  105
2024-03-04  120
2024-03-05  115
```

Рисунок 23 – CSV-файл

## Задание 24. 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.

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

dates = pd.date_range(start='2024-03-01', end='2024-03-30', freq='D')
values = np.random.randint(50, 151, size=len(dates))
time_series = pd.Series(values, index=dates)

plt.figure(figsize=(10, 4))
time_series.plot(linewidth=2, marker='o', markersize=5)

plt.title('График', fontsize=20, pad=20)
plt.xlabel('Дата', fontsize=16)
plt.ylabel('Значение', fontsize=16)
plt.grid(True, linestyle='--', alpha=0.7)

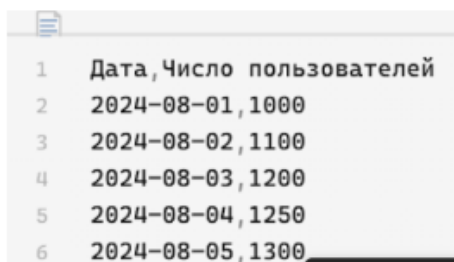
plt.tight_layout()
plt.show()
```



Рисунок 24 – Построение графика

**Задание 25.** Индивидуальное задание 1. Работа с временными рядами

Создайте CSV-файл `users.csv` со следующими данными:



1	Дата, Число пользователей
2	2024-08-01, 1000
3	2024-08-02, 1100
4	2024-08-03, 1200
5	2024-08-04, 1250
6	2024-08-05, 1300

Прочитайте файл, установите `DatetimeIndex`, найдите процентный прирост пользователей (`pct_change()`), отобразите его на графике и выделите дни с приростом выше 5%.

```
2]: users = """Дата,Число пользователей
2024-08-01,1000
2024-08-02,1100
2024-08-03,1200
2024-08-04,1250
2024-08-05,1300"""

with open('users.csv', 'w', encoding='utf-8') as file:
    file.write(users)

df = pd.read_csv('users.csv', parse_dates=['Дата'])
df.set_index('Дата', inplace=True)

df['Прирост, %'] = df['Число пользователей'].pct_change() * 100

plt.figure(figsize=(10, 6))
bars = plt.bar(df.index, df['Прирост, %'])

for i, bar in enumerate(bars):
    if df['Прирост, %'].iloc[i] > 5:
        bar.set_color('red')

plt.title('Ежедневный прирост пользователей (август 2024)', fontsize=20)
plt.xlabel('Дата', fontsize=16)
plt.ylabel('Прирост, %', fontsize=16)
plt.grid(axis='y', linestyle='--', alpha=0.7)
```

Рисунок 25 – Индивидуальное задание

## Ответы на контрольные вопросы:

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

`pandas.Series` — это одномерный помеченный массив, похожий на столбец в таблице.

Отличия от списка Python:

Серия имеет индексы (можно задавать вручную).

Поддерживает разные типы данных в одном массиве (но лучше однородные).

Оптимизирована для векторизованных операций (быстрее циклов).

Есть встроенные методы для анализа данных (например, `.mean()`, `.fillna()`).

### 2. Какие типы данных можно использовать для создания `Series`?

Любые типы:

Числовые (int, float).

Строковые (str).

Булевы (bool).

Даты (datetime).

Категории (category).

Пример:

```
import pandas as pd  
s = pd.Series([1, 2.5, "текст", True])
```

### **3. Как задать индексы при создании Series?**

Через параметр index:

```
s = pd.Series([10, 20, 30], index=['a', 'b', 'c'])
```

### **4. Как обратиться к элементу Series по индексу?**

По метке: `s['a']` → 10.

По позиции: `s[0]` → 10.

### **5. Разница между .iloc[] и .loc[]**

`.loc[]` — доступ по метке индекса (например, `s.loc['a']`).

`.iloc[]` — доступ по позиции (например, `s.iloc[0]`).

### **6. Логическая индексация в Series**

```
s = pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])
```

```
s[s > 2] # Вывод: c:3, d:4
```

### **7. Просмотр первых и последних элементов**

`.head(n)` — первые n элементов.

`.tail(n)` — последние n элементов.

### **8. Проверка типа данных элементов**

```
s.dtype # Тип элементов
```

```
s.astype('float') # Пример изменения типа
```

### **9. Изменение типа данных Series**

```
s = s.astype('float64') # Преобразование во float
```

### **10. Проверка пропущенных значений (NaN)**

```
s.isnull().sum() # Количество пропусков
```

```
s.isna() # Булев массив
```

## 11. Заполнение пропущенных значений

`.fillna(0)` — заполнить нулями.

`.fillna(s.mean())` — заполнить средним.

## 12. Разница между `.fillna()` и `.dropna()`

`.fillna()` — заменяет NaN указанным значением.

`.dropna()` — удаляет строки с NaN.

## 13. Математические операции с Series

Арифметика: `s + 2`, `s * s`.

Статистика: `s.sum()`, `s.mean()`.

## 14. Преимущество векторизованных операций

Быстрее циклов (оптимизированы под C).

Пример: `s * 2` выполняется мгновенно, а цикл — медленно.

## 15. Применение функции к каждому элементу

`s.apply(lambda x: x**2)` # Квадрат каждого элемента

## 16. Агрегирующие функции

`s.sum()`, `s.mean()`, `s.min()`, `s.max()`, `s.std()`.

## 17. Минимум, максимум, среднее, стандартное отклонение

`print(s.min(), s.max(), s.mean(), s.std())`

## 18. Сортировка Series

По значениям: `s.sort_values()`.

По индексу: `s.sort_index()`.

## 19. Проверка уникальности индексов

`s.index.is_unique` # True/False

## 20. Сброс индексов

`s.reset_index(drop=True)` # Старые индексы удаляются

## 21. Задание нового индекса

`s.index = ['x', 'y', 'z']` # Новые метки

## 22. Работа с временными рядами

`s = pd.Series([1, 2, 3], index=pd.to_datetime(['2023-01-01', '2023-01-02', '2023-01-03']))`

`s.resample('D').mean()` # Ресемплирование

### 23. Преобразование строк в даты

```
s.index = pd.to_datetime(s.index)
```

### 24. Выбор данных по временному диапазону

```
s['2023-01-01':'2023-01-02']
```

### 25. Загрузка данных из CSV в Series

```
s = pd.read_csv('data.csv', usecols=['column_name'], squeeze=True)
```

### 26. Установка столбца CSV как индекса

```
s = pd.read_csv('data.csv', index_col='date')
```

### 27. Метод .rolling().mean()

Скользящее среднее (например, для сглаживания временного ряда).

```
s.rolling(window=3).mean() # Среднее за 3 периода
```

### 28. Метод .pct\_change()

Процентное изменение между элементами.

```
s.pct_change() # (x[i] - x[i-1]) / x[i-1]
```

### 29. Применение .rolling() и .pct\_change()

Финансовый анализ (изменение цен акций).

Анализ трендов (например, рост продаж).

### 30. Почему появляются NaN и как с ними работать?

Причины:

Пропущенные данные в источнике.

Операции с разными индексами (например,  $s1 + s2$ ).

Решение:

Удалить: `.dropna()`.

Заполнить: `.fillna(value)`.

**Вывод:** в ходе лабораторной работы мы исследовали основы работы с библиотекой pandas, в частности, со структурой данных Series