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
дисциплины
«Искусственный интеллект и машинное обучение»
Вариант 5

Выполнил:
Беков Шамиль Расулович
2 курс, группа ИВТ-б-о-23-1,
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.

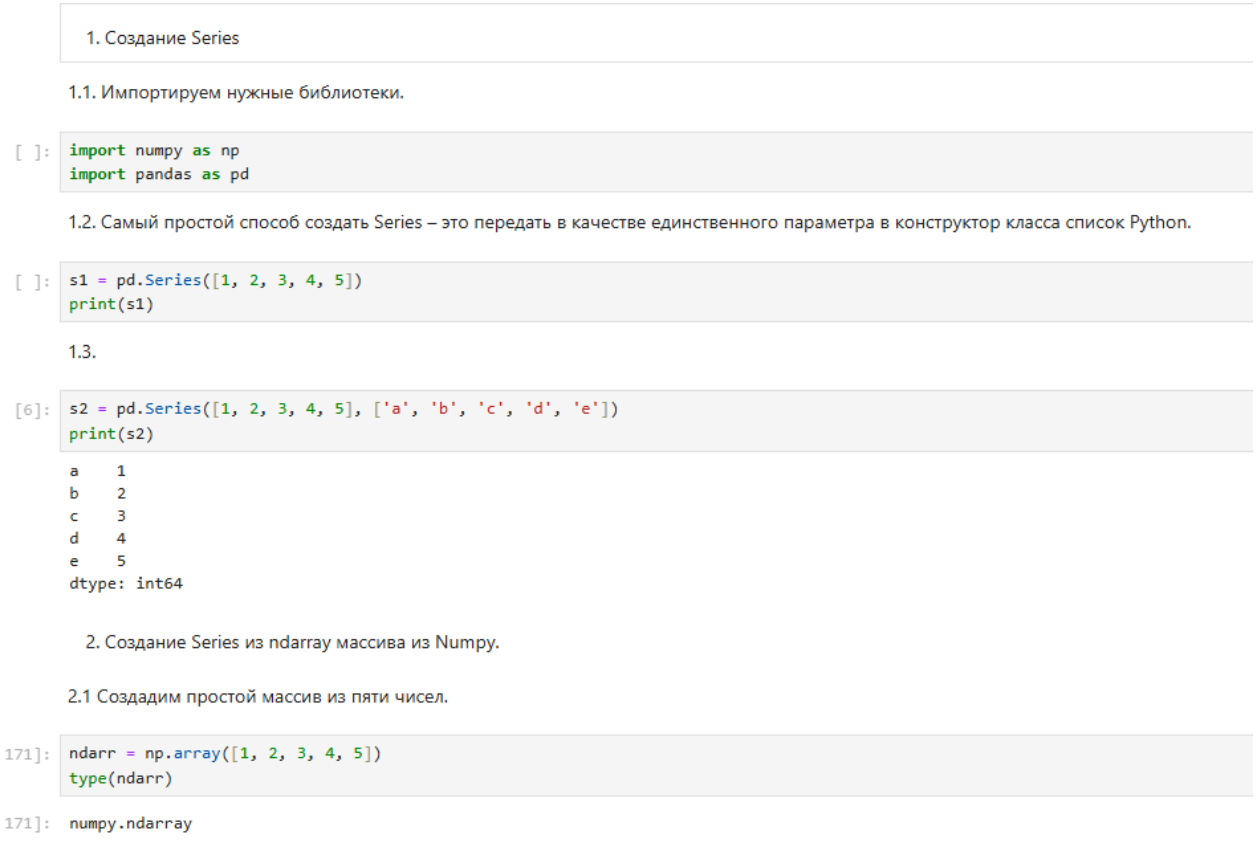
Тема: Введение в pandas: изучение структуры Series и базовых операций.

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

Github: https://github.com/ShamilBekov/Laba-4/tree/main

Порядок выполнения работы:

1. Проработали примеры практической работы:



1. Создание Series

1.1. Импортируем нужные библиотеки.

```
[ ]: import numpy as np
     import pandas as pd
```

1.2. Самый простой способ создать Series – это передать в качестве единственного параметра в конструктор класса список Python.

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

1.3.

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

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

2. Создание Series из ndarray массива из Numpy.

2.1 Создадим простой массив из пяти чисел.

```
171]: ndarr = np.array([1, 2, 3, 4, 5])
      type(ndarr)

171]: numpy.ndarray
```

Рисунок 1. Создание Series разными способами

5. Работа с элементами Series.

5.1. обращение к элементам Series по численному индексу.

[190]:
```
s6 = pd.Series([1, 2, 3, 4, 5], ['a', 'b', 'c', 'd', 'e'])
print(s6[2])
print(s6['d'])
print(s6[:2])
print(s6[s6 <= 3])
```

```
3
4
a    1
b    2
dtype: int64
a    1
b    2
c    3
dtype: int64
C:\Users\User\AppData\Local\Temp\ipykernel_19188\3822981573.py:2: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. I
bels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  print(s6[2])
```

5.2. Со структурами Series можно работать как с векторами: складывать, умножать вектор на число и т.п.

[28]:
```
s7 = pd.Series([10, 20, 30, 40, 50], ['a', 'b', 'c', 'd','e'])
print(s6 + s7)
print(s6 * 3)
```

```
a    11
b    22
c    33
d    44
e    55
dtype: int64
a     3
b     6
c     9
d    12
e    15
dtype: int64
```

Рисунок 2. Работа с элементами Series

9. Использование логической индексации для фильтрации данных в Series.

9.1. При логической индексации создается булевый массив.

[41]:
```
import pandas as pd
s = pd.Series([10, 25, 8, 30, 15], index=['a', 'b', 'c', 'd', 'e'])
filtered_s = s[s > 10]
print(filtered_s)
```

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

9.1. Создание логического массива. При применении логического выражения к Series создается булевый массив, который можно вывести отдельно:

[44]:
```
print(s > 10)
```

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

9.2. Комбинированные условия. Можно задавать **несколько условий** с логическими операторами & (и), | (или), ~ (не):

[47]:
```
filtered_s = s[(s >= 10) & (s <= 30)]
print(filtered_s)
```

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

Рисунок 3. Индексация для фильтрации данных Series

10. Изменение значений элементов в Series.

10.1. Изменение значений по индексу. Значения в pandas.Series можно изменять, обращаясь к элементу по его индексу с помощью .loc[] (меточная индексация) или .iloc[] (позиционная индексация).

[57]:
```
import pandas as pd
s = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
s.loc['b'] = 25
print(s)
```

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

10.2. Изменение значений по позиции.

[60]:
```
s.iloc[1] = 50
print(s)
```

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

10.3. Изменение значений с помощью логической индексации.

[63]:
```
s[s > 30] += 10
print(s)
```

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

Рисунок 4. Изменение значений элементов в Series

11. Основные методы работы с Series.

11.1. Метод `.head(n)` позволяет получить первые `n` элементов из объекта `Series`. Если `n` не указан, по умолчанию возвращается первые 5 элементов.

76]:
```
import pandas as pd
s = pd.Series([10, 20, 30, 40, 50, 60, 70], index=['a', 'b', 'c', 'd','e', 'f', 'g'])
print(s.head(3))
```

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

11.1.1. Если вызвать `.head()` без аргумента, по умолчанию будет выведено 5 элементов:

79]:
```
print(s.head())
```

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

11.2. Метод `.teil(n)` работает аналогично `.head(n)`, но возвращает последние n элементов `Series`.

32]:
```
print(s.tail(3))
```

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

11.2.1. Если вызвать `.teal()` без аргумента, по умолчанию будет выведено 5 элементов:

Рисунок 5. Основные методы работы с Series

12.1. Получение индексов с помощью `.INDEX`.

[89]:
```
import pandas as pd
s = pd.Series([10, 20, 30, 40, 50], index=['a', 'b', 'c', 'd', 'e'])
print(s.index)
```

```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

12.1.1. К индексу можно обращаться по элементам, так как он работает как массив:

[92]:
```
print(s.index[0])
print(s.index[-1])
```

```
a
e
```

12.1.1. Можно проверить, содержит ли индекс определенное значение:

[95]:
```
print('b' in s.index)
print('z' in s.index)
```

```
True
False
```

12.2. Получение индексов с помощью `.VALUES`.

[98]:
```
print(s.values)
```

```
[10 20 30 40 50]
```

12.2.1. Значения можно использовать в операциях, как с обычным массивом:

[101]:
```
print(s.values[0])
print(s.values.mean())
```

```
10
30.0
```

Рисунок 6. Получение индексов с помощью .INDEX

13. `.dtype` - определение типа данных в pandas Sedries.

13.1. Проверка типа данных в `Series`.

105]:
```
import pandas as pd
s = pd.Series([10, 20, 30, 40, 50])
print(s.dtype)
```

```
int64
```

13.2. Автоматическое определение типа.

При сизданиие `Series` pandas автоматически определяет тип данных на основе переданных значений:

108]:
```
s1 = pd.Series([1.5, 2.3, 3.7])
s2 = pd.Series(["apple", "banana", "cherry"])
s3 = pd.Series([True, False, True])
print(s1.dtype)
print(s2.dtype)
print(s3.dtype)
```

```
float64
object
bool
```

13.3. Приведение типа с `.ASTYPE()`.

111]:
```
s1_int = s1.astype(int)
print(s1_int)
print(s1_int.dtype)
```

```
0    1
1    2
2    3
dtype: int32
int32
```

Рисунок 7. Определение типов данных в pandas Series

14. `.insull()` и `.notnull()` - проверка наличия пропущенных значений в pandas Series.

14.1. `.ISNULL()` - выявление пропущенных значений.

[118]:
```
import pandas as pd
import numpy as np
s = pd.Series([10, np.nan, 30, None, 50])
print(s.isnull())
```

```
0    False
1     True
2    False
3     True
4    False
dtype: bool
```

14.2. `.NOTNULL()` - проверка на ненулевые значения.

[121]:
```
print(s.notnull())
```

```
0     True
1    False
2     True
3    False
4     True
dtype: bool
```

14.3. Использование логической индексации для фильтрации непустых значенииий.

[124]:
```
filtered_s = s[s.notnull()]
print(filtered_s)
```

```
0    10.0
2    30.0
4    50.0
dtype: float64
```

14.3.1. Выбор только пропущенных значений:

[127]:
```
missing_values = s[s.isnull()]
print(missing_values)
```

```
1   NaN
3   NaN
dtype: float64
```

Рисунок 8. Проверка наличия пропущенных значений в pandas Series

15. `.fillna(value)`, `.dropna()` - работа с пропущенными значениями в pandas Series.

15.1. `.FILLNA(VALUE) - замена пропущенных значений.

[131]:
```
import pandas as pd
import numpy as np
s = pd.Series([10, np.nan, 30, None, 50])
s_filled = s.fillna(0)
print(s_filled)
```

```
0    10.0
1     0.0
2    30.0
3     0.0
4    50.0
dtype: float64
```

15.2. Замена пропущенных значений средним, медианой или другим вычисленным значением.

[134]:
```
s_filled = s.fillna(s.mean())
print(s_filled)
```

```
0    10.0
1    30.0
2    30.0
3    30.0
4    50.0
dtype: float64
```

15.3. `.DROPNA()` - удаление пропущенных значений.

[137]:
```
s_cleaned = s.dropna()
print(s_cleaned)
```

```
0    10.0
2    30.0
4    50.0
dtype: float64
```

Рисунок 9. Работа с пропущенными значениями в pandas Series

16. Арифметические операции с числом.

[140]:
```
import pandas as pd
s = pd.Series([10, 20, 30, 40, 50])
s_multiplied = s * 2
print(s_multiplied)
```

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

17. Арифметические операции между двумя Series.

[143]:
```
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'])
s_sum = s1 + s2
print(s_sum)
```

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

Рисунок 10. Арифметические операции

20. Применение функций с помощью `.apply(func)` для Series.

20.1. Использование `.APPLY()` с анонимной функцией `LAMBDA`.

[153]:
```
import pandas as pd
s = pd.Series([1, 2, 3, 4, 5])
s_squared = s.apply(lambda x: x ** 2)
print(s_squared)
```

```
0     1
1     4
2     9
3    16
4    25
dtype: int64
```

20.2. Использование `.APPLY()` с обычной функцией.

[156]:
```
def custom_function(x):
    return f"Value: {x}"
s_transformed = s.apply(custom_function)
print(s_transformed)
```

```
0    Value: 1
1    Value: 2
2    Value: 3
3    Value: 4
4    Value: 5
dtype: object
```

20.3. Использование `.APPLY()` для сложных вычислений.

[159]:
```
import numpy as np
s_log = s.apply(np.log)
print(s_log)
```

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

Рисунок 11. Применение функций с помощью .apply(func) для Series

21. Методы агрегирования Series.

21.1. Сумма элеентов `.sum()`.

[194]:
```
import pandas as pd
s = pd.Series([10, 20, 30, 40, 50])
print(s.sum())
```

```
150
```

21.1.1. При наличии NaN метод `sum()` их игнорирует.

[199]:
```
s_with_nan = pd.Series([10, 20, None, 40, 50])
print(s_with_nan.sum())
```

```
120.0
```

21.2. Среднее значение `.MEAN()`.

[204]:
```
print(s.mean())
```

```
30.0
```

21.2.1. При наличии NaN метод игнорирует их.

[209]:
```
print(s_with_nan.mean())
```

```
30.0
```

21.3. Минимум и максимум `.MIN()` и `.MAX()`.

[212]:
```
print(s.min())
print(s.max())
```

```
10
50
```

Рисунок 12. Методы агрегирования Series

22. Совместимость с NumPy.

22.1. Логарифм `np.log(s)`.

[223]:
```
import pandas as pd
import numpy as np
s = pd.Series([1, 2, 3, 4, 5])
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
```

22.2. Экспонента `np.exp(s)`.

[226]:
```
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
```

22.3. Квадратный корень `np.sqrt(s)`.

[229]:
```
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
```

Рисунок 13. Совместимость с NumPy

23. Работа с индексами.

23.1. Изменение индексов с помощью `.SET_INDEX()`.

[239]:
```
import pandas as pd
s = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
s.index = ['x', 'y', 'z', 'w']
print(s)
```

```
x    10
y    20
z    30
w    40
dtype: int64
```

23.2. Сброс индексов с помощью `.RESER_INDEX()`.

[242]:
```
s_reset = s.reset_index()
print(s_reset)
```

```
  index   0
0     x  10
1     y  20
2     z  30
3     w  40
```

23.2.1. Если требуется удалить старый индекс, можно передать аргумент `drop=True`.

[245]:
```
s_reset = s.reset_index(drop=True)
print(s_reset)
```

```
0    10
1    20
2    30
3    40
dtype: int64
```

Рисунок 14. Работа с индексами

24. Проверка уникальности индексов Series.

[248]:
```
import pandas as pd
s_unique = pd.Series([10, 20, 30], index=['a', 'b', 'c'])
print(s_unique.index.is_unique)
```

```
True
```

24.2. Проверка при наличии повторяющихся индексов.

[251]:
```
s_duplicate = pd.Series([10, 20, 30, 40], index=['a', 'b', 'a', 'c'])
print(s_duplicate.index.is_unique)
```

```
False
```

24.3. Доступ к элементам при повторяющихся индексах.

[254]:
```
print(s_duplicate.loc['a'])
```

```
a    10
a    30
dtype: int64
```

24.4. Обработка неуникальных индексов.

24.4.1. Сбросить индекс и создать числовой индекс.

[258]:
```
s_reset = s_duplicate.reset_index(drop=True)
print(s_reset)
```

```
0    10
1    20
2    30
3    40
dtype: int64
```

Рисунок 15. Проверка уникальности индексов Series

25. Сортировка данных в Series.

25.1. Сортировка по индексу с `.sort_index()`.

25.1.1. По умолчанию индексы сортируются в **в порядке возрастания**.

[269]:
```
import pandas as pd
s = pd.Series([10, 20, 30, 40], index=['d', 'b', 'a', 'c'])
s_sorted = s.sort_index()
print(s_sorted)
```

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

25.1.2. Сортировка в порядке убывания.

[273]:
```
s_sorted_desc = s.sort_index(ascending=False)
print(s_sorted_desc)
```

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

25.2. Сортировка по значениям с .sort_values().

25.2.1. По умолчанию индексы сортируются в в порядке возрастания.

[277]:
```
s_sorted_values = s.sort_values()
print(s_sorted_values)
```

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

Рисунок 16. Сортировка данных в Series

26. Обработка `NaN` при сортировке.

```
[283]: s_nan = pd.Series([10, None, 30, None, 20], index=['a', 'b', 'c', 'd','e'])
print(s_nan.sort_values())
```

```
a    10.0
e    20.0
c    30.0
b     NaN
d     NaN
dtype: float64
```

26.1. Можно изменить поведение с помощью параметра `na_position`.

```
[286]: print(s_nan.sort_values(na_position='first'))
```

```
b     NaN
d     NaN
a    10.0
e    20.0
c    30.0
dtype: float64
```

Рисунок 17. Обработка NaN при сортировке

27. Применение Series для представления временных рядов.

27.1. Создание временного ряда в `Series`.

Временные ряды строятся с использованием индекса типа `DatetimeIndex`.

```
[293]: import pandas as pd
dates = pd.date_range(start='2024-03-01', periods=5, freq='D')
s = pd.Series([100, 105, 102, 98, 110], index=dates)
print(s)
```

```
2024-03-01    100
2024-03-02    105
2024-03-03    102
2024-03-04     98
2024-03-05    110
Freq: D, dtype: int64
```

27.2. Доступ к данным по дате.

```
[296]: print(s['2024-03-03'])
```

```
102
```

27.2.1. Филтрация по диапозону дат.

```
[299]: print(s['2024-03-02':'2024-03-04'])
```

```
2024-03-02    105
2024-03-03    102
2024-03-04     98
Freq: D, dtype: int64
```

Рисунок 18. Применение Series для представления временных рядов

28. Процентный прирост.

Метод `.pct_change()` вычисляет процентное изменение.

Каждое значени показывает насколько процентно изменился текущий элемент относительно предыдущего.

[342]:
```
import pandas as pd
s = pd.Series([100, 110, 120, 90, 150])
print(s.pct_change())
```

```
0         NaN
1    0.100000
2    0.090909
3   -0.250000
4    0.666667
dtype: float64
```

28.1. Процентное изменение в процентах.

По умолчанию `.pct_change()` возвращает **долю изменений (0.1 = 10%)**, но можно умножить на 100, чтобы получить проценты.

[347]:
```
print(s.pct_change() * 100)
```

```
0          NaN
1    10.000000
2     9.090909
3   -25.000000
4    66.666667
dtype: float64
```

28.2. Изменение с другим интервалом (периодом).

По умолчанию `.pct_change()` сравнивает каждое значение с предыдущим (`periods=1`). Можно задать больший шаг (`periods=n`), чтобы вычислить изменение **по сравнению с n шагов назад**.

[351]:
```
print(s.pct_change(periods=2))
```

```
0         NaN
1         NaN
2    0.200000
3   -0.181818
4    0.250000
dtype: float64
```

Рисунок 19. Процентный прирост

29. Создание Series на основе данных из внешних файлов.

29.1. Загрузка данных из CSV-файла.

[368]:
```
import pandas as pd
df = pd.read_csv("data.csv", header=None, names=["Дата", "Цена"])
s=df["Цена"]
print(s)
```

```
0    Цена
1     100
2     120
3     140
4     125
5     115
Name: Цена, dtype: object
```

29.2. Чтение данных из файла с указанием индекса.

[397]:
```
s = pd.read_csv("data.csv", index_col="Дата")["Цена"]
print(s)
```

```
Дата
2024-05-01    100
2024-05-02    120
2024-05-03    140
2024-05-04    125
2024-05-05    115
Name: Цена, dtype: int64
```

Рисунок 20. Создание Series на основе данных из внешних файлов

2. Решили практические задания работы.

Задание №1. 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.

Результат выполнения задания:

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.

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

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

Тип данных: int64
```

Рисунок 21. Создание Series из списка

Задание №2. 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.

Результат выполнения задания:

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 индеком `'C'` и `.iloc[]` для получения третьего элемента.

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

```
Элемент с индексом 'C': 36

Третий элемент: 36
```

Рисунок 22. Получение элемента Series

Задание №3. Создайте Series из массива NumPy np.array([4, 9, 16, 25, 36, 49, 64]). Выберите только те элементы, которые больше 20, и выведите результат.

Результат выполнения задания:

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.

```
[54]: 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)
```

```
Исходный массив:
 0     4
1     9
2    16
3    25
4    36
5    48
6    60
dtype: int32

Массив с элементами больше 20:
 3    25
4    36
5    48
6    60
dtype: int32
```

Рисунок 23. Фильтрация данных с помощью логической индексации

Задание №4. 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().

Результат выполнения задания:

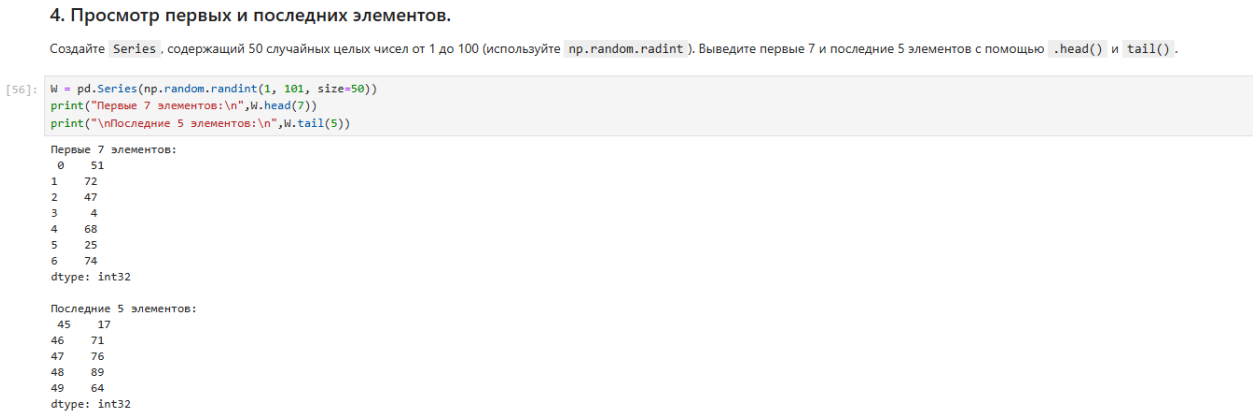


Рисунок 24. Просмотр первых и последних элементов

Задание №5. Создайте Series из списка ['cat', 'dog', 'rabbit', 'parrot', 'fish']. Определите тип данных с помощью .dtype , затем преобразуйте его в category с помощью .astype().Результат выполнения задания:

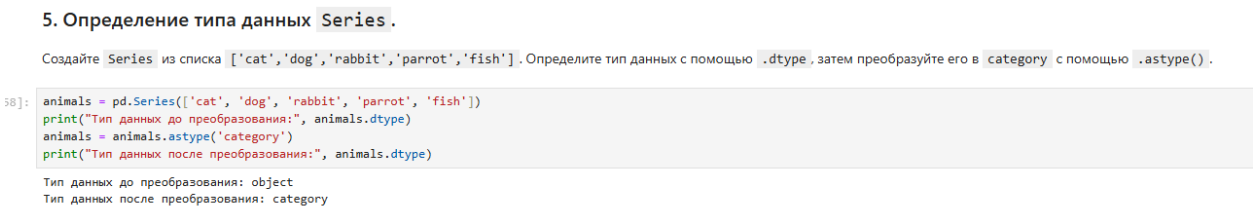


Рисунок 25. Определение типа данных Series

Задание №6. 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.

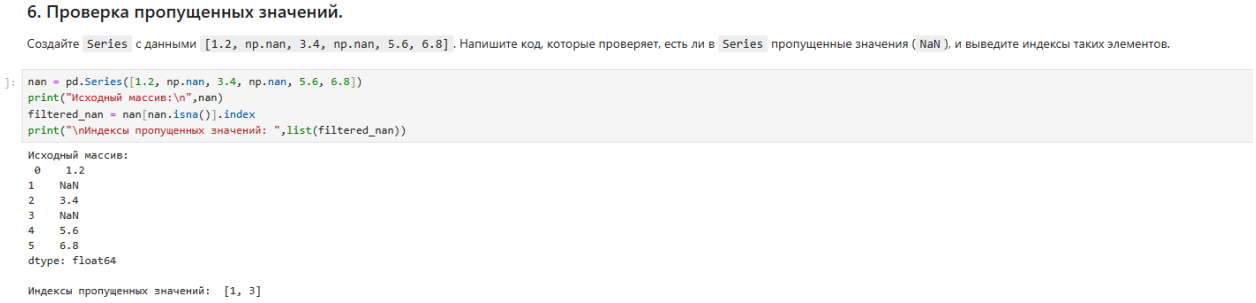


Рисунок 26. Проверка пропущенных значений

Задание №7. Используйте Series из предыдущего задания и замените все NaN на среднее значение всех непустых элементов. Выведите результат.

7. Заполнение пропущенных значений.

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

[62]:
```
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)
```

```
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 27. Заполнение пропущенных значений

Задание №8. Создайте два Series:

1. s1 = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])

2. 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.

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`.

64]:
```
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,\n поскольку индекс 'a' и 'e' отсутствовали в первом и втором массиве соответственно, они
```

```
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
```

Рисунок 28. Арифметические операции с Series

Задание №9. 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).

Результат выполнения задания:

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.sq`

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

```
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 29. Применение функции к Series

Задание №10. 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.

Результат выполнения задания:

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("\nСтандратное отклонение:", a.std())
print("\n",a.describe())
```

```
Массив:
 0     69
1    109
2    107
3    109
4     77
5     70
6     55
7     77
8     99
9    126
10    77
11    74
12    89
13    58
14    73
15    96
16    95
17   110
18   139
19    67
dtype: int32

Сумма элементов: 1776

Среднее значение: 88.8

Минимальное значение: 55

Максимальное значение: 139

Стандратное отклонение: 22.818736530362614
```

Рисунок 30. Основные статистические методы

Задание №11. 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.

Результат выполнения задания:

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.

```
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)
```

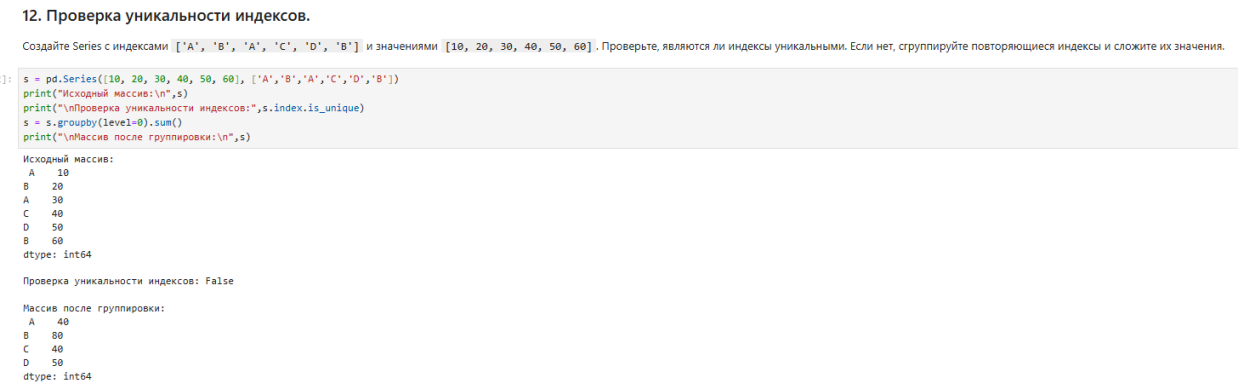
```
Исходные данные:
 2024-03-01    59
2024-03-02    10
2024-03-03    71
2024-03-04    55
2024-03-05    69
2024-03-06    96
2024-03-07    17
2024-03-08    68
2024-03-09    45
2024-03-10    42
Freq: D, dtype: int32

Данные за 5-8 марта:
 2024-03-05    69
2024-03-06    96
2024-03-07    17
2024-03-08    68
Freq: D, dtype: int32
```

Рисунок 31. Работа с временными рядами

Задание №12. 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.

Результат выполнения задания:



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'])
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)
```

```
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 32. Проверка уникальности индексов

Задание №13. 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.

Результат выполнения задания:

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.

[74]:
```
s = pd.Series([100, 200, 300], index=['2024-03-10','2024-03-11','2024-03-12'])
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)
```

```
Тип данных индекса до преобразования: object

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

Рисунок 33. Преобразование строковых дат в DatetimeIndex

Задание №14. Создайте CSV-файл data.csv со следующими данными:

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

Результат выполнения задания:

14. Чтение данных из CSV-файла.

Создайте CSV-файл `data.csv` . Прочитайте файл и создайте `Series` , используя `"Дата"` в качестве индекса.

6]:
```
data = {
    'Дата': ['2024-03-01', '2024-03-02', '2024-03-03', '2024-03-04', '2024-03-05'],
    '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)
```

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

Рисунок 34. Чтение данных из CSV-файла

Задание №15. 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.

Результат выполнения задания:

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.

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

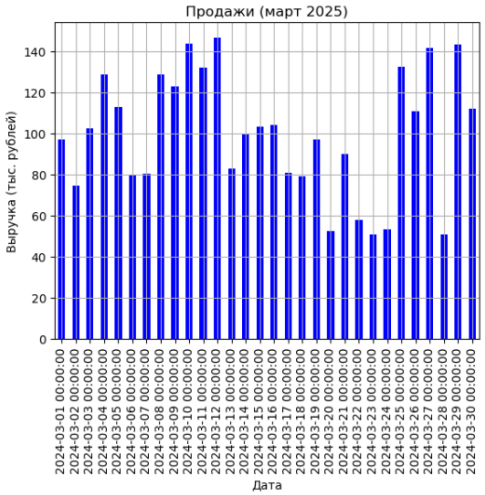


Рисунок 35. Построение графика на основе Series

3. Выполнение индивидуального задания.

Создайте Series, где индексами будут даты с 10 по 20 марта 2024 года, а значениями - случайные температуры от -5 до +10 градусов. Выделите данные за 15-17 марта, преобразуйте индекс в DatetimeIndex, а затем постройте график температуры.

5. Преобразование и анализ погодных данных.

Создайте `Series`, где индексами будут даты с 10 по 20 марта 2024 года, а значениями – случайные температуры от -5 до +10 градусов. Выделите данные за 15-17 марта, преобразуйте индекс в `DatetimeIndex`, а затем постройте график температуры.

```
[18]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

s = pd.Series(np.random.uniform(-5, 10, size=11),index=pd.date_range(start='2024-03-10', end='2024-03-20', freq='D'))

print("Исходный Series:")
print(s)

subset = s['2024-03-15':'2024-03-17']

print("\nДанные за 15-17 марта:")
print(subset)

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

plt.figure(figsize=(10, 6))
plt.plot(s.index, s.values, marker='o', linestyle='-')
plt.plot(subset.index, subset.values, marker='o', linestyle='-', color='red', linewidth=2, label='15-17 March')
plt.title("Температура в марте 2024")
plt.xlabel("Дата")
plt.ylabel("Температура (°C)")
plt.grid(True)
plt.legend()
plt.show()
```

Рисунок 36. Решение индивидуального задания

```
Исходный Series:
2024-03-10    3.135649
2024-03-11    4.147180
2024-03-12    9.112508
2024-03-13    3.484698
2024-03-14    9.511020
2024-03-15    2.166255
2024-03-16   -2.307010
2024-03-17    7.240249
2024-03-18    2.971627
2024-03-19   -2.867183
2024-03-20   -2.922918
Freq: D, dtype: float64

Данные за 15-17 марта:
2024-03-15    2.166255
2024-03-16   -2.307010
2024-03-17    7.240249
Freq: D, dtype: float64
```

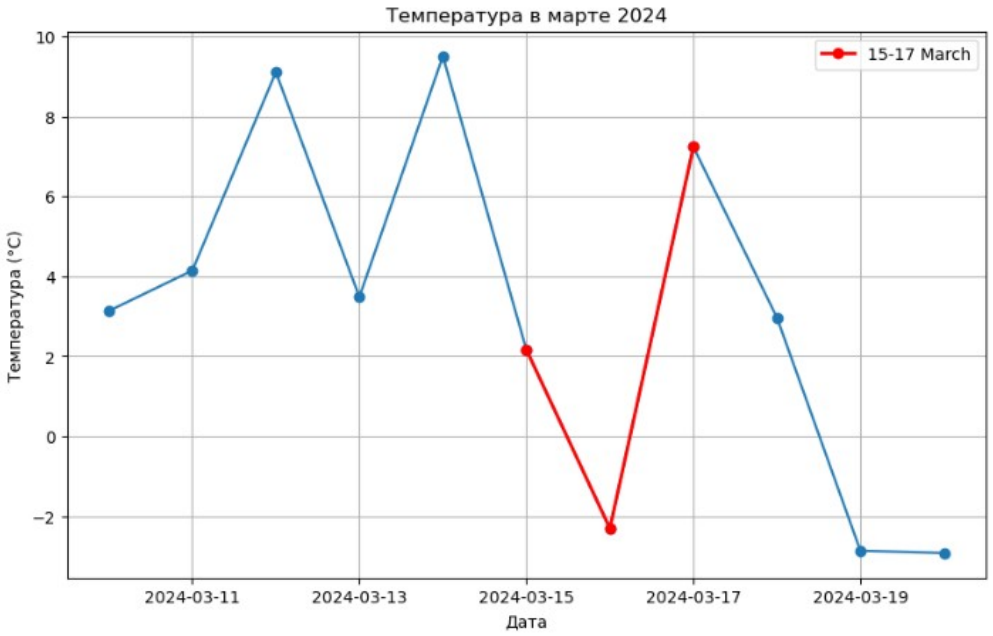


Рисунок 37. Результат решения индивидуального задания

Контрольные вопросы:

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

Series – это маркированная одномерная структура данных, ее можно представить, как таблицу с одной строкой. С Series можно работать как с обычным массивом (обращаться по номеру индекса), и как с ассоциированным массивом, когда можно использовать ключ для доступа к элементам данных.

2. Какие типы данных можно использовать для создания Series?

Словари Python; списки Python; массивы из numpy: ndarray; скалярные величины.

3. Как задать индексы при создании Series?

pandas.Series(data=None, index=None, dtype=None, name=None,

copy=False, fastpath=False), где index – список меток, который будет использоваться для доступа к элементам Series. Длина списка должна быть равна длине data;

4. Каким образом можно обратиться к элементу Series по его индексу?

К элементам Series можно обращаться по численному индексу, Можно использовать метку, Метод .iloc[] используется для доступа к элементам Series по их порядковому номеру, начиная с 0. Метод .loc[] позволяет обращаться к элементам Series по их пользовательским меткам (индексам).

5. В чём разница между .iloc[] и .loc[] при индексации Series?

- iloc следует использовать, когда важно обращаться к элементам по их порядковому номеру.

- loc удобен, когда метки индексов несут смысловую нагрузку и их важно учитывать при доступе к данным.

При работе со срезами iloc исключает верхнюю границу, а loc включает её.

6. Как использовать логическую индексацию в Series?

При логической индексации создаётся булевый массив (массив True и False), который применяется к Series, возвращая только те элементы, для которых значение True.

7. Какие методы можно использовать для просмотра первых и последних элементов Series?

Метод .head(n) позволяет получить первые n элементов из объекта Series. Если n не указан, по умолчанию возвращаются первые 5 элементов.

Метод .tail(n) работает аналогично .head(n) , но возвращает последние n элементов Series .

8. Как проверить тип данных элементов Series?

Атрибут .dtype в pandas.Series используется для определения типа данных элементов внутри серии. Он возвращает объект numpy.dtype, который показывает, к какому типу принадлежат данные.

9. Каким способом можно изменить тип данных Series ?

При необходимости можно изменить тип данных Series с помощью метода .astype().

s = s.astype('int')

10. Как проверить наличие пропущенных значений в Series?

Метод .isnull() возвращает True для элементов, которые являются пропущенными (NaN), и False для всех остальных.

11. Какие методы используются для заполнения пропущенных значений в Series?

Метод .fillna(value) заменяет все NaN в Series на указанное значение.

Метод .dropna() удаляет все элементы, содержащие NaN, и возвращает новую Series без пропущенных значений.

12. Чем отличается метод .fillna() от .dropna()?

Метод .fillna(value) заменяет все NaN в Series на указанное значение.

Метод .dropna() удаляет все элементы, содержащие NaN, и возвращает новую Series без пропущенных значений.

13. Какие математические операции можно выполнять с Series?

+, -, *, /, //, %, **, а также сравнения: ==, <, >, и т.д. Применяются поэлементно.

14. В чём преимущество векторизированных операций по сравнению с циклами Python?

Они быстрее, чем циклы for, т.к. выполняются на уровне библиотеки

15. Как применить пользовательскую функцию к каждому элементу Series?

Метод .apply(func) позволяет применять функцию к каждому элементу Series , выполняя преобразование или анализ данных без необходимости использования циклов for. Это особенно полезно при обработке больших наборов данных.

16. Какие агрегирующие функции доступны в Series?

sum(), mean(), min(), max(), std(), median(), count()

17. Как узнать минимальное, максимальное, среднее и стандартное отклонение Series?

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

18. Как сортировать Series по значениям и по индексам?

По значениям: s.sort_values();

По индексам: s.sort_index().

19. Как проверить, являются ли индексы Series уникальными?

Атрибут .is_unique возвращает True, если все индексы уникальны, и False, если есть повторяющиеся значения.

20. Как сбросить индексы Series и сделать их числовыми?

s_reset = s_duplicate.reset_index(drop=True)

21. Как можно задать новый индекс в Series?

s.index = новый_список

22. Как работать с временными рядами в Series ?

Временные ряды строятся с использованием индекса типа DatetimeIndex. Можно выбирать данные по конкретной дате, как по обычному индексу: print(s['2024-03-03']).

23. Как преобразовать строковые даты в формат DatetimeIndex?

pd.to_datetime(s)

24. Каким образом можно выбрать данные за определённый временной диапазон?

s['2022-01-01':'2022-03-01']

25. Как загрузить данные из CSV-файла в Series?

Для чтения CSV-файла используется функция pd.read_csv() . Обычно данные загружаются в виде DataFrame, но можно извлечь отдельный столбец в Series

26. Как установить один из столбцов CSV-файла в качестве индекса Series?

pd.read_csv('file.csv', index_col='название_столбца')

27. Для чего используется метод .rolling().mean() в Series?

Метод .rolling(window).mean() создаёт окно фиксированной длины (window), внутри которого вычисляется среднее значение. Окно перемещается по Series, обновляя расчёты на каждом шаге.

28. Как работает метод .pct_change()? Какие задачи он решает?

Метод .pct_change() вычисляет процентное изменение между текущим и предыдущим значением Series . Он широко используется в финансовом анализе, анализе временных рядов, а также для оценки относительных изменений в данных.

29. В каких ситуациях полезно использовать .rolling() и .pct_change()?

.rolling() – для сглаживания и анализа трендов;

.pct_change() – для анализа относительных изменений.

30. Почему NaN могут появляться в Series, и как с ними работать?

Причины: отсутствующие значения, ошибки при чтении данных. Методы:

.fillna() – заменить

.dropna() – удалить

.isna() – найти

Вывод: в ходе лабораторной работы были получены навыки работы с основами библиотеки pandas, в частности, со структурой данных Series.