

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 1

Выполнил:
Бакулин Вадим Романович
2 курс, группа ИТС-б-о-23-1,
11.03.02 «Инфокоммуникационные
технологии и системы связи»,
направленность (профиль)
«Инфокоммуникационные системы и
сети», очная форма обучения

(подпись)

Проверил:
Доцент департамента цифровых,
робототехнических систем и
электроники Воронкин Р.А.

(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.

Ссылка на репозиторий: https://github.com/zepteloid/AI_ML_LR_4

Порядок выполнения работы:

1. Выполнение задания №1.

```
#Задание №1
import pandas as pd
import numpy as np

s = pd.Series([5, 15, 25, 35, 45], index = ['a', 'b', 'c', 'd', 'e'])
print('Результат:\n', s.tolist())
print('Тип:', s.dtype)
```

Рисунок 1. Задание №1

2. Выполнение задания №2.

```
#Задание №2
import pandas as pd
import numpy as np

s = pd.Series([12, 24, 36, 48, 60], index=['A', 'B', 'C', 'D', 'E'])
print(s.loc['C'])
print(s.iloc[2])
```

Рисунок 2. Задание №2

3. Выполнение задания №3.

```
#Задание №3
import pandas as pd
import numpy as np

massiv = np.array([4, 9, 16, 25, 36, 49, 64])
s = pd.Series(massiv)

filtred = s[s > 20]

print('Отфильтрованный массив:\n', filtred.tolist())
```

Рисунок 3. Задание №3

4. Выполнение задания №4.

```
#Задание №4
import pandas as pd
import numpy as np

numbers = np.random.randint(1, 101, 50)
s = pd.Series(numbers)
print('Первые 7 элементов:\n', s.head(7).tolist())
print('Последние 5 элементов:\n', s.tail(5).tolist())
```

Рисунок 4. Задание №4

5. Выполнение задания №5.

```
#Задание №5
import pandas as pd
import numpy as np

s = pd.Series(['cat', 'dog', 'rabbit', 'parrot', 'fish'])
obj = s.dtype

print('Изначальный тип данных:', obj)

categor = s.astype('category')
print('Тип данных после изменения:', categor.dtype)
```

Рисунок 5. Задание №5

6. Выполнение задания №6.

```
#Задание №6
import pandas as pd
import numpy as np

data = pd.Series([1.2, np.nan, 3.4, np.nan, 5.6, 6.8])
Nan = data.isnull()

print('Вывод индексов пропущенных значений', data.index[Nan].tolist())
```

Рисунок 6. Задание №6

7. Выполнение задания №7.

```

#Задание №7
import pandas as pd
import numpy as np

data = pd.Series([1.2, np.nan, 3.4, np.nan, 5.6, 6.8])

sredn = np.mean(data)

filtred = data.fillna(sredn)
print('Измененные данные:\n',filtred.tolist())

```

Рисунок 7. Задание №7

8. Выполнение задания №8.

```

#Задание №8
import pandas as pd
import numpy as np

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

summ = s1 + s2
print('Полученная сумма:',summ.tolist())
print('Nan появляется из-за несовпадения индексов некоторых элементов.')

zamena = summ.fillna(0)
print('Данные после замены "nan" на 0:',zamena.tolist())

```

Рисунок 8. Задание №8

9. Выполнение задания №9.

```

#Задание №9
import pandas as pd
import numpy as np

s = pd.Series([2, 4, 6, 8, 10])
koren = s.apply(np.sqrt)

print('Вычисленный квадрат каждого из чисел:\n',koren.tolist())

```

Рисунок 9. Задание №9

10. Выполнение задания №10.

```
#Задание №10
import pandas as pd
import numpy as np

numbers = np.random.randint(50, 151, 20)

s = pd.Series(numbers)
print('Полученные числа:', s.tolist())

print('Полученная сумма:', np.sum(s))
print('Среднее значение:', np.mean(s))
print('Минимальное значение:', np.min(s))
print('Максимальное значение:', np.max(s))
print('Стандартное отклонение:', np.std(s))
```

Рисунок 10. Задание №10

11. Выполнение задания №11

```
#Задание №11
import pandas as pd
import numpy as np

znach = np.random.randint(10, 101, 10)

s = pd.Series(znach, index = pd.date_range(start='2024-03-01', periods=10, freq='D'))

print(s.loc['2024-03-05': '2024-03-08'])
```

Рисунок 11. Задание №11

12. Выполнение задания №12

```
#Задание №12
import pandas as pd
import numpy as np

s = pd.Series([10, 20, 30, 40, 50, 60], index = ['A', 'B', 'A', 'C', 'D', 'B'])
uniq = s.index.is_unique
print('Проверка на уникальность индексов: ', uniq)

if not uniq:
    group = s.groupby(s.index).sum()
    print('Сложение сгруппированных повторяющих индексов')
    print(group)
```

Рисунок 12. Задание №12

13. Выполнение задания №13

```
#Задание №13
import pandas as pd
import numpy as np

s = pd.Series([100, 200, 300], index = ['2024-03-10', '2024-03-11', '2024-03-12'])
s.index = pd.to_datetime(s.index)
print('Преобразованный индекс:', s.index.dtype)
```

Рисунок 13. Задание №13

14. Выполнение задания №14

```
#Задание №14
import pandas as pd
import numpy as np

df = pd.read_csv("data.csv")
s = pd.Series(df["Цена"].values, index=df["Дата"])
print(s)
```

Рисунок 14. Задание №14

15. Выполнение задания №15

```
#Задание №15
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dates = pd.date_range(start="2024-03-01", periods=30, freq="D")
prices = np.random.randint(50, 151, 30)

s = pd.Series(prices, index=dates)

plt.figure(figsize=(10,6))
plt.plot(s.index, s.values, label="Цена акций", marker="o")
plt.gcf().autofmt_xdate()
plt.ylabel("Цена")
plt.xlabel("Дата")
plt.title("График цены")

plt.grid()
plt.show()
```

Рисунок 15. Задание №15

16. Выполнение индивидуального задания

```

#Индивидуальное задание
import pandas as pd
import matplotlib.pyplot as plt

# Создание Series
dates = ['2024-04-01', '2024-04-02', '2024-04-03', '2024-04-04', '2024-04-05']
values = [120, 130, 110, 140, 135]

series = pd.Series(values, index=dates)

# Преобразование индекса в DatetimeIndex
series.index = pd.to_datetime(series.index)

# Построение графика
plt.figure(figsize=(10, 5))
series.plot()
plt.title('График значений по датам', fontsize=14)
plt.xlabel('Дата', fontsize=12)
plt.ylabel('Значения', fontsize=12)
plt.grid(True)
plt.show()

```

Рисунок 16. Индивидуальное задание

Ответы на контрольные вопросы:

1. Что такое pandas.Series и чем она отличается от списка в Python Pandas. Series — это одномерный массив индексированных данных из библиотеки Pandas для работы с данными на Python. Series можно рассматривать как столбец в таблице, который может хранить данные различных типов.

Некоторые отличия Pandas.Series от списка в Python:

Однородность. В Series все элементы должны быть одного типа данных, в то время как список может содержать элементы разных типов.

Эффективность использования памяти. Series более эффективны, чем списки, так как внутри используют массивы NumPy, которые более компактны и быстрее для числовых вычислений

2. Какие типы данных можно использовать для создания Series?

Для создания Series в библиотеке pandas можно использовать различные типы данных, в том числе:

Словари Python.

Списки Python.

Массивы из numpy: ndarray.

Скалярные величины.

Некоторые основные типы данных, используемые в pandas:

object — текстовые или смешанные числовые и нечисловые значения;

int64 — целые числа;

float64 — числа с плавающей точкой;

bool — булево значение: True/False;

3. Как задать индексы при создании Series

Чтобы задать индексы при создании Series в Pandas, необходимо при вызове конструктора включить параметр `index` и присвоить ему массив строк с метками.

Общий вид синтаксиса: `pd.Series(data, index=index)`

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

В библиотеке Pandas для обращения к элементу Series по индексу используют квадратные скобки языка Python. Индекс должен быть целым числом.

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

`loc` позволяет индексировать по метке, а `iloc` по целому числу позиций, по которым необходимо сделать выборку.

6. Как использовать логическую индексацию в Series

Логическая индексация в Series позволяет отбирать элементы структуры на основе логического выражения. Для этого в квадратных скобках записывается логическое выражение, согласно которому будет произведён отбор.

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

`head(n)`: Этот метод возвращает первые `n` элементов Series. Если `n` не указан, по умолчанию возвращаются первые 5 элементов.

`tail(n)`: Этот метод возвращает последние `n` элементов `Series`. Если `n` не указан, по умолчанию возвращаются последние 5 элементов.

8. Как проверить тип данных элементов `Series`?

В библиотеке `pandas` для проверки типа данных элементов объекта `Series` можно использовать атрибут. `dtype`.

9. Каким способом можно изменить тип данных `Series`?

В библиотеке `pandas` для изменения типа данных объекта `Series` можно использовать метод. `astype()`. Этот метод позволяет преобразовать элементы `Series` в другой тип данных.

10. Как проверить наличие пропущенных значений в `Series`?

В библиотеке `pandas` для проверки наличия пропущенных значений в объекте `Series` можно использовать метод. `isnull()` или `.isna()`, а также метод `any()` для определения, есть ли хотя бы одно пропущенное значение.

11. Методы для заполнения пропущенных значений в `Series`

`.fillna()`: Заполняет пропущенные значения указанным значением, результатом вычисления или методом

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

- `.fillna()`: Заменяет пропущенные значения на указанное значение.

- `.dropna()`: Удаляет строки или столбцы, содержащие пропущенные значения.

13. Математические операции с `Series`

Сложение (+), вычитание (-), умножение (*), деление (/), возведение в степень (**), взятие остатка (%), floor division (//). Эти операции выполняются поэлементно.

14. Преимущество векторизованных операций перед циклами

Python

— Скорость: Векторизованные операции (использующие `NumPy` и `pandas`) выполняются гораздо быстрее, чем циклы Python, так как они реализованы на C/C++ и используют оптимизированные алгоритмы.

– Удобство: Код становится более лаконичным и читаемым, так как не нужно писать циклы для обработки каждого элемента.

15. Применение пользовательской функции к каждому элементу Series

– Использовать метод `.apply()`, передав в него имя пользовательской функции.

16. Агрегирующие функции в Series

`.sum()`, `.mean()`, `.median()`, `.min()`, `.max()`, `.std()`, `.var()`, `.count()`, `.size()`, `.nunique()`

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

- `.min()`: Минимальное значение.
- `.max()`: Максимальное значение.
- `.mean()`: Среднее значение.
- `.std()`: Стандартное отклонение.

18. Сортировка Series

- `.sort_values()`: Сортировка по значениям.
- `.sort_index()`: Сортировка по индексам

19. Проверка уникальности индексов Series

`series.index.is_unique`: возвращает `True`, если все индексы уникальны, и `False` в противном случае.

20. Как сбросить индексы Series и сделать их числовыми?

Для сброса индексов Series и присвоения числовых:

`series.reset_index(drop=True, inplace=True)`

- `drop=True`: удаляет старые индексы.
- `inplace=True`: изменяет Series "на месте".

21. Как можно задать новый индекс в Series ?

1. `series.index = new_index`: Просто присвоить новый список/массив/Index-объект свойству `index`. Длина `new_index` должна совпадать с длиной `series`.

2. `series.reindex(new_index)`: Создаёт новый Series с указанным

`new_index`. Если в `new_index` есть значения, отсутствующие в старом индексе, им присваивается `NaN`.

3. `series.set_axis(new_index, axis=0)`: (менее распространенный) Более общий метод для изменения индексов (и столбцов в `DataFrame`). `axis=0` указывает на изменение индекса. Возвращает новый `Series`.

Выбор зависит от того, нужно ли вам заменить существующий индекс (способ 1) или создать новый `Series` с другим набором индексов (способы 2 и 3).

22. Как работать с временными рядами в `Series` ?

1. Создание: `pd.Series(data, index=pd.to_datetime(dates))` - Индекс должен быть `DatetimeIndex`.

2. Доступ: `series['YYYY-MM-DD']`, `series['YYYY-MM-DD':'YYYY-MM-DD']`, `series.index.dt.year/month/day`

3. `Resample`: `series.resample('D/W/M/A').mean()` - Изменение частоты, агрегация.

4. `Shift`: `series.shift(periods=1)` - Сдвиг данных.

5. `Rolling`: `series.rolling(window=N).mean()` - Скользящее среднее.

6. Пропуски: `series.fillna()`, `series.dropna()`, `series.interpolate()`

23. Как преобразовать строковые даты в формат `DatetimeIndex` ?

1. `pd.to_datetime(серия_строк)`: Самый простой способ. Преобразует `Series` или список строк в `DatetimeIndex`.

2. `pd.DatetimeIndex(серия_строк)`: Создает `DatetimeIndex` напрямую из `Series` или списка строк. Оба способа автоматически распознают большинство распространенных форматов дат. Если формат нестандартный, используйте параметр `format=`, чтобы указать формат строки даты.

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

Если индекс `Series/DataFrame` - `DatetimeIndex`: Слайсинг строками: `df['YYYY-MM-DD':'YYYY-MM-DD']` (включает обе границы диапазона) `loc` со строками: `df.loc['YYYY-MM-DD':'YYYY-MM-DD']` (то же, но более явный)

Если столбец с датами (не индекс):

Логическая индексация: `start_date = 'YYYY-MM-DD'` `end_date = 'YYYY-MM-DD'` `mask = (df['date_column'] >= start_date) & (df['date_column'] <= end_date)` `df.loc[mask]`

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

```
import pandas as pd
```

1. Загрузка CSV в DataFrame

```
df = pd.read_csv('имя_файла.csv', index_col='имя_столбца_с_индексом')
```

2. Преобразование столбца DataFrame в Series

```
series = df['имя_столбца'].squeeze() #squeeze() преобразует DataFrame с одним столбцом в Series
```

Альтернатива (если индекс не нужен из CSV)

```
# series = pd.read_csv('имя_файла.csv', usecols=['имя_столбца']).squeeze()
```

#Если столбец с датами и должен быть индексом

```
#series=pd.read_csv('имя_файла.csv',index_col='имя_столбца_с_индексом',parse_dates=['имя_столбца_с_индексом'])['имя_столбца'].squeeze()
```

#если нужно, `parse_dates`

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

Series?

```
import pandas as pd
```

```
series = pd.read_csv('имя_файла.csv', index_col='имя_столбца_с_индексом')['имя_столбца'].squeeze()
```

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

`.rolling().mean()` используется для вычисления скользящего среднего

(или *moving average*) в Series. Он берёт окно из N последовательных значений и вычисляет их среднее, затем сдвигает окно на одно значение и повторяет процесс. Это сглаживает колебания и показывает тренд.

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

`.pct_change()` вычисляет процентное изменение между текущим и предыдущим элементом в Series/DataFrame.

Задачи:

- 1) Анализ роста: Определение процентного роста или падения во времени (например, изменение цены акции, рост продаж).
- 2) Сравнение изменений: Сравнение скорости изменений между разными периодами или разными временными рядами.
- 3) Нормализация данных: Приведение данных к процентным изменениям, чтобы убрать влияние абсолютных значений.

Кратко: вычисляет процентное изменение между последовательными значениями, что полезно для анализа роста и сравнения изменений.

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

* `.rolling()`:

- 1) Сглаживание временных рядов от шума и случайных колебаний.
 - 2) Выявление трендов и долгосрочных изменений.
 - 3) Фильтрация данных для упрощения анализа.
- 4) `.pct_change()`:
- 5) Измерение темпов роста/падения (экономика, финансы).
 - 6) Сравнение волатильности разных активов.
 - 7) Визуализация изменений в данных относительно предыдущего периода.

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

Почему появляются NaN в Series:

- 1) Отсутствие данных: Явное отсутствие значения в данных (например, в CSV-файле).
- 2) Неопределенные вычисления: Операции, которые не могут быть выполнены (например, деление на ноль).
- 3) Объединение/переиндексация: Объединение Series/DataFrames с разными индексами, где некоторые индексы отсутствуют в другом Series.
- 4) Сдвиг данных (`shift`): Сдвиг временного ряда приводит к появлению NaN в начале или конце.

Как работать с NaN:

1) Обнаружение: `series.isna()` или `series.isnull()` - возвращают Series с True/False.

2) Удаление: `series.dropna()` - удаляет строки с NaN.

3) Заполнение: `series.fillna(value)` - заменяет NaN указанным значением (value может быть числом, средним, предыдущим значением и т.д.).

4) Интерполяция: `series.interpolate()` - заполняет NaN на основе соседних значений (линейно, полиномиально и т.д.).

Выбор метода обработки зависит от контекста и задачи анализа.

Вывод: познакомились с основами работы с библиотекой pandas, в частности, со структурой данных Series