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 6

Выполнил:
Якушенко Антон Андреевич
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:
Ассистент департамента цифровых, робототехнических систем и электроники Хацукова А.И

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г.

ТЕМА: ВВЕДЕНИЕ В PANDAS: ИЗУЧЕНИЕ СТРУКТУРЫ SERIES И БАЗОВЫХ ОПЕРАЦИЙ

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

Ссылка на репозиторий: https://github.com/Yakush766/LB4.git

Порядок выполнения работы:

Задание 1: Создание Series из списка

```python
import pandas as pd

# Создаем Series из списка чисел и индексов
data = [5, 15, 25, 35, 45]
index = ['a', 'b', 'c', 'd', 'e']
series = pd.Series(data, index=index)

# Выводим Series на экран
print(series)

# Определяем и выводим тип данных Series
print("\nТип данных Series:")
print(series.dtype)
```

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

Тип данных Series:
int64
```

Рисунок 1. Задание 1

Задание 2: Получение элемента Series

```python
import pandas as pd

# Создаем Series с указанными индексами и значениями
data = [12, 24, 36, 48, 60]
index = ['A', 'B', 'C', 'D', 'E']
series = pd.Series(data, index=index)

# Используем .loc для получения элемента с индексом 'C'
element_loc = series.loc['C']
print(f"Элемент с индексом 'C' (используя .loc): {element_loc}")

# Используем .iloc для получения третьего элемента (индекс 2)
element_iloc = series.iloc[2]
print(f"Третий элемент (используя .iloc): {element_iloc}")
```

```
Элемент с индексом 'C' (используя .loc): 36
Третий элемент (используя .iloc): 36
```

Рисунок 2. Задание 2

Задание 3: Фильтрация данных с помощью логической индексации

```python
import pandas as pd
import numpy as np

# Создаем Series из массива NumPy
data = np.array([4, 9, 16, 25, 36, 49, 64])
series = pd.Series(data)

# Используем логическую индексацию для выбора элементов, которые больше 20
filtered_series = series[series > 20]

# Выводим отфильтрованный Series
print(filtered_series)
```

```
3    25
4    36
5    49
6    64
dtype: int32
```

Рисунок 3. Задание 3

Задание 4: Просмотр первых и последних элементов

```python
import pandas as pd
import numpy as np

# Создаем Series, содержащий 50 случайных целых чисел от 1 до 100
data = np.random.randint(1, 101, 50)  # От 1 (включительно) до 101 (не включительно)
series = pd.Series(data)

# Выводим первые 7 элементов
print("Первые 7 элементов:")
print(series.head(7))

# Выводим последние 5 элементов
print("\nПоследние 5 элементов:")
print(series.tail(5))
```

```
Первые 7 элементов:
0     3
1    22
2    83
3    12
4    76
5    61
6     3
dtype: int32

Последние 5 элементов:
45    67
46    10
47    59
48    78
49    81
dtype: int32
```

Рисунок 4. Задание 4

Задание 5: Определение типа данных Series

```python
import pandas as pd

# Создаем Series из списка строк
data = ['cat', 'dog', 'rabbit', 'parrot', 'fish']
series = pd.Series(data)

# Определяем тип данных Series
print("Исходный тип данных Series:")
print(series.dtype)

# Преобразуем тип данных в category
series_category = series.astype('category')

# Определяем тип данных преобразованной Series
print("\nТип данных Series после преобразования в 'category':")
print(series_category.dtype)
```

```
Исходный тип данных Series:
object

Тип данных Series после преобразования в 'category':
category
```

Рисунок 5. Задание 5

Задание 6: Проверка пропущенных значений

```python
import pandas as pd
import numpy as np

# Создаем Series с данными, содержащими пропущенные значения (np.nan)
data = [1.2, np.nan, 3.4, np.nan, 5.6, 6.8]
series = pd.Series(data)

# Проверяем, есть ли в Series пропущенные значения (NaN)
nan_mask = series.isnull()

# Выводим индексы элементов, которые являются NaN
nan_indices = nan_mask[nan_mask].index
print("Индексы элементов, являющихся NaN:")
print(nan_indices)
```

```
Индексы элементов, являющихся NaN:
Index([1, 3], dtype='int64')
```

Рисунок 6. Задание 6

Задание 7: Заполнение пропущенных значений

```python
import pandas as pd
import numpy as np

# Создаем Series с данными, содержащими пропущенные значения (np.nan)
data = [1.2, np.nan, 3.4, np.nan, 5.6, 6.8]
series = pd.Series(data)

# Вычисляем среднее значение всех непустых элементов
mean_value = series.mean()

# Заполняем все NaN значения средним значением
filled_series = series.fillna(mean_value)

# Выводим результат
print("Series после заполнения NaN средним значением:")
print(filled_series)
```

```
Series после заполнения NaN средним значением:
0    1.20
1    4.25
2    3.40
3    4.25
4    5.60
5    6.80
dtype: float64
```

Рисунок 7. Задание 7

Задание 8: Арифметические операции с Series

```python
import pandas as pd

# Создаем 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'])

# Выполняем сложение s1 + s2
result = s1 + s2

# Выводим результат
print("Результат сложения:")
print(result)

# Заменяем NaN на 0
result_filled = result.fillna(0)

# Выводим результат после замены NaN
print("\nРезультат после замены NaN на 0:")
print(result_filled)

# Объяснение появления NaN
print("\nОбъяснение:")
print("NaN появляются в результате сложения Series, когда индексы не совпадают.")
print("В таких случаях pandas не может найти соответствующее значение для сложения,")
print("и в результате получается NaN.")
```

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

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

Объяснение:
NaN появляются в результате сложения Series, когда индексы не совпадают.
В таких случаях pandas не может найти соответствующее значение для сложения,
и в результате получается NaN.
```

Рисунок 8. Задание 8

Задание 9: Применение операции с Series

```python
import pandas as pd
import numpy as np

# Создаем Series из чисел
data = [2, 4, 6, 8, 10]
series = pd.Series(data)

# Применяем функцию вычисления квадратного корня к каждому элементу
sqrt_series = series.apply(np.sqrt)

# Выводим результат
print("Series после применения функции вычисления квадратного корня:")
print(sqrt_series)
```

```
Series после применения функции вычисления квадратного корня:
0    1.414214
1    2.000000
2    2.449490
3    2.828427
4    3.162278
dtype: float64
```

Рисунок 9. Задание 9

Задание 10: Основные статические методы

```python
import pandas as pd
import numpy as np

# Создаем Series из 20 случайных чисел от 50 до 150 (включительно)
data = np.random.randint(50, 151, 20) # От 50 (включительно) до 151 (не включительно)
series = pd.Series(data)

# Находим сумму
series_sum = series.sum()
print(f"Сумма: {series_sum}")

# Находим среднее значение
series_mean = series.mean()
print(f"Среднее значение: {series_mean}")

# Находим минимальное значение
series_min = series.min()
print(f"Минимальное значение: {series_min}")

# Находим максимальное значение
series_max = series.max()
print(f"Максимальное значение: {series_max}")

# Находим стандартное отклонение
series_std = series.std()
print(f"Стандартное отклонение: {series_std}")
```

```
Сумма: 2094
Среднее значение: 104.7
Минимальное значение: 54
Максимальное значение: 149
Стандартное отклонение: 29.263053767327044
```

Рисунок 10. Задание 10

Задание 11: Работа с временными рядами

```python
import pandas as pd
import numpy as np

# Создаем Series с датами в качестве индексов и случайными числами в качестве значений
dates = pd.date_range(start='2024-03-01', periods=10, freq='D')
data = np.random.randint(10, 101, 10)  # От 10 (включительно) до 101 (не включительно)
series = pd.Series(data, index=dates)

# Выбираем данные за 5-8 марта
selected_data = series['2024-03-05':'2024-03-08']

# Выводим выбранные данные
print("Данные за 5-8 марта:")
print(selected_data)
```

```
Данные за 5-8 марта:
2024-03-05    64
2024-03-06    55
2024-03-07    16
2024-03-08    12
Freq: D, dtype: int32
```

Рисунок 11. Задание 11

Задание 12: Проверка уникальности индексов

```python
import pandas as pd

# Создаем Series с повторяющимися индексами
index = ['A', 'B', 'A', 'C', 'D', 'B']
data = [10, 20, 30, 40, 50, 60]
series = pd.Series(data, index=index)

# Проверяем уникальность индексов
if series.index.is_unique:
    print("Индексы уникальны.")
else:
    print("Индексы не уникальны.")

    # Группируем повторяющиеся индексы и суммируем их значения
    grouped_series = series.groupby(series.index).sum()

    # Выводим сгруппированную Series
    print("\nСгруппированная Series с суммированными значениями:")
    print(grouped_series)
```

```
Индексы не уникальны.

Сгруппированная Series с суммированными значениями:
A    40
B    80
C    40
D    50
dtype: int64
```

Рисунок 12. Задание 12

Задание 13: Преобразование строковых дат в DatetimeIndex

```python
import pandas as pd

# Создаем Series с индексами в виде строк дат
dates = ['2024-03-10', '2024-03-11', '2024-03-12']
data = [100, 200, 300]
series = pd.Series(data, index=dates)

# Преобразуем индексы в DatetimeIndex
series.index = pd.to_datetime(series.index)

# Выводим тип данных индекса
print("Тип данных индекса:")
print(series.index.dtype)

# Выводим Series с DatetimeIndex
print("\nSeries с DatetimeIndex:")
print(series)
```

```
Тип данных индекса:
datetime64[ns]

Series с DatetimeIndex:
2024-03-10    100
2024-03-11    200
2024-03-12    300
dtype: int64
```

Рисунок 13. Задание 13

Задание 14: Построение графика на основе Series

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

# Создаем диапазон дат с 1 по 30 марта 2024 года.
dates = pd.date_range(start='2024-03-01', end='2024-03-30')

# Создаем случайные целые числа от 50 до 150 (включительно).
values = np.random.randint(50, 151, size=len(dates))

# Создаем объект Series из Pandas, где даты - это индексы, а значения - случайные числа.
series = pd.Series(values, index=dates)

# Создаем график Series.
plt.figure(figsize=(12, 6))  # Необязательно: Настраиваем размер фигуры
plt.plot(series)

# Добавляем заголовок и подписи к осям.
plt.title('График случайных значений в марте 2024 года')
plt.xlabel('Дата')
plt.ylabel('Значение')

# Добавляем сетку для лучшей читаемости.
plt.grid(True)

# Поворачиваем метки оси x, чтобы они не перекрывались (необязательно).
plt.xticks(rotation=45)

# Отображаем график.
plt.tight_layout() # Автоматически регулирует параметры подграфиков, чтобы они помещались в область фигуры.
plt.show()
```

Рисунок 14. Задание 14

Индивидуальное задание:

1. Сначала создадим CSV-файл

```
import csv

# Данные для записи в CSV-файл
data = [
    ['Дата', 'Продажи'],
    ['2024-07-01', 120],
    ['2024-07-02', 150],
    ['2024-07-03', 170],
    ['2024-07-04', 160],
    ['2024-07-05', 180]
]

# Открываем файл для записи, 'w' означает запись, newline='' предотвращает лишние пустые строки
with open('sales.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)  # Создаем объект writer
    writer.writerows(data)  # Записываем все строки данных

print("Файл sales.csv успешно создан.")
```

```
Файл sales.csv успешно создан.
```

Рисунок 15. Создал файл CSV

2. Далее читаем с этого файла данные и строим графики

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

# 1. Чтение данных из CSV-файла
try:
    df = pd.read_csv('sales.csv', encoding='windows-1251')  # Попробуем кодировку windows-1251
except UnicodeDecodeError:
    df = pd.read_csv('sales.csv', encoding='cp1251')  # Если windows-1251 не сработает, пробуем cp1251

# 2. Установка DatetimeIndex
df['Дата'] = pd.to_datetime(df['Дата']) # Преобразуем столбец 'Дата' в формат datetime
df = df.set_index('Дата') # Устанавливаем столбец 'Дата' в качестве индекса

# 3. Расчет скользящего среднего (rolling mean)
rolling_mean = df['Продажи'].rolling(window=2).mean()

# 4. Построение графиков
plt.figure(figsize=(12, 6)) # Создаем фигуру для графиков

# График 1: Реальные продажи
plt.plot(df['Продажи'], label='Реальные продажи')

# График 2: Скользящее среднее
plt.plot(rolling_mean, label='Скользящее среднее (window=2)')

# Добавляем заголовок, подписи осей и легенду
plt.title('Продажи и скользящее среднее')
plt.xlabel('Дата')
plt.ylabel('Продажи')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

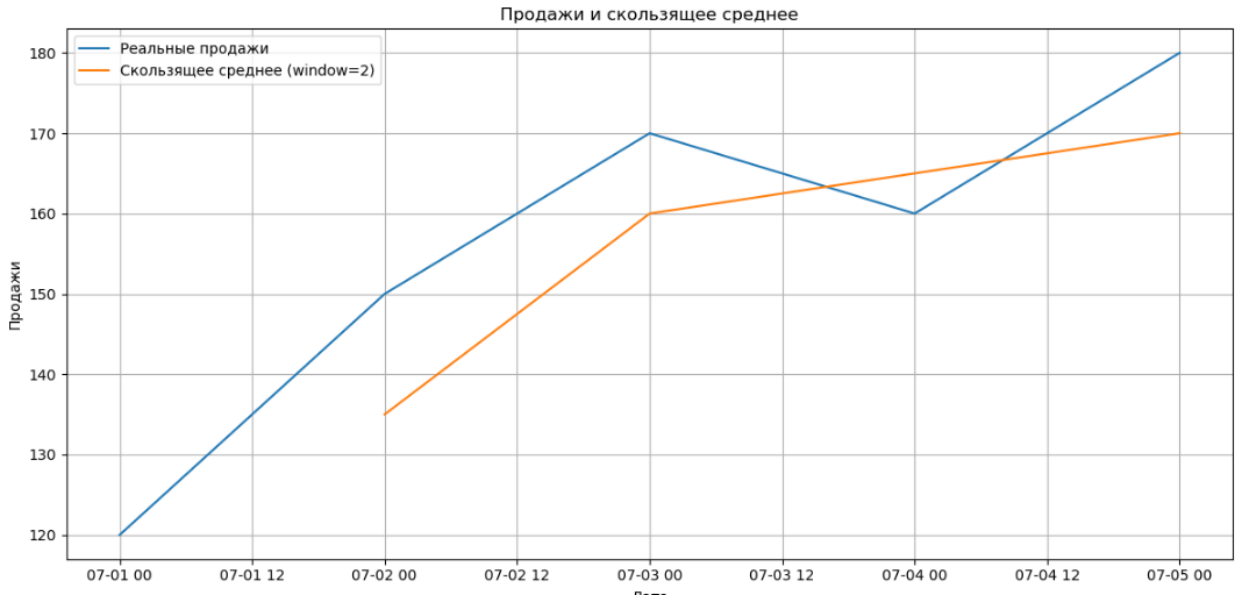

Рисунок 16. Выполненное задание

Ответы на контрольные вопросы:

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

pandas.Series - это одномерный массив данных с индексами. В отличие от списков Python, Series имеют явно определенные индексы, которые могут быть любого неизменяемого типа (например, числа, строки, даты). Series также поддерживают векторизованные операции и методы pandas для анализа данных, чего нет у обычных списков.

2. **Какие типы данных можно использовать для создания Series?**

Можно использовать любые типы данных, поддерживаемые NumPy, включая числа (целые, с плавающей точкой), строки, логические значения, даты и время. Все элементы Series должны быть одного типа данных (dtype).

3. Как задать индексы при создании Series?

Индексы можно задать с помощью параметра index при создании Series. Например: pd.Series(data, index=my_index_list). Если индекс не задан, pandas автоматически создаст числовой индекс, начиная с 0.

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

Можно использовать квадратные скобки [] с индексом, как в словаре: my_series['my_index'] или my_series[0].

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

– .iloc[] использует *числовую* позицию (целочисленный индекс) элемента, начиная с 0. Это похоже на индексацию списка.

– .loc[] использует *значение индекса* (метка индекса). Это позволяет обращаться к элементам по их явно заданным именам индексов.

6. Как использовать логическую индексацию в Series?

Можно создать логическую маску (Series с булевыми значениями) на основе некоторого условия и использовать эту маску для выбора элементов, удовлетворяющих условию: my_series[my_series > 10].

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

– .head(n): Возвращает первые n элементов Series. По умолчанию n=5.

– .tail(n): Возвращает последние n элементов Series. По умолчанию n=5.

8. Как проверить тип данных элементов Series?

Можно использовать атрибут .dtype:my_series.dtype.

9. Каким способом можно изменить тип данных Series?

Можно использовать метод .astype(): my_series.astype('float64').

10. Как проверить наличие пропущенных значений в Series?

Можно использовать метод .isnull() (или .isna()), который возвращает Series с булевыми значениями, где True означает пропущенное значение (NaN). Чтобы посчитать количество пропусков, можно использовать .isnull().sum().

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

– .fillna(value): Заменяет пропущенные значения на указанное значение.

– .fillna(method='ffill'): Заполняет пропущенные значения предыдущим не-пропущенным значением (forward fill).

– .fillna(method='bfill'): Заполняет пропущенные значения следующим не-пропущенным значением (backward fill).

– .interpolate(): Заполняет пропущенные значения, используя интерполяцию.

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

– .fillna(): Заполняет пропущенные значения.

– .dropna(): Удаляет строки (или столбцы в DataFrame), содержащие пропущенные значения.

13. Какие математические операции можно выполнять с Series? Можно выполнять любые стандартные математические операции (+, -, *, /, **, %), а также логические операции (>, <, ==, !=) между Series и скалярными значениями или между двумя Series. Эти операции применяются поэлементно (векторизованно).

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

Векторизованные операции выполняются значительно быстрее, чем циклы Python, так как они используют оптимизированные библиотеки NumPy, написанные на C. Векторизация также упрощает код, делая его более читаемым.

15. Как применить пользовательскую функцию к каждому элементу Series? Можно использовать метод .apply(my_function). my_function должна принимать один элемент Series в качестве аргумента и возвращать результат.

16. Какие агрегирующие функции доступны в Series? .sum(), .mean(), .median(), .min(), .max(), .std(), .var(), .count(), .unique(), .nunique().

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

Использовать методы: .min(), .max(), .mean(), .std().

18. Как сортировать Series по значениям и по индексам?

- .sort_values(): Сортирует Series по значениям.
- .sort_index(): Сортирует Series по индексу.

19. Как проверить, являются ли индексы Series уникальными?

Использовать атрибут .is_unique у объекта индекса: my_series.index.is_unique.

20. Как сбросить индексы Series и сделать их числовыми?

Использовать метод .reset_index(): my_series.reset_index(). Это преобразует индекс в обычный столбец и создаст новый числовой индекс.

21. Как можно задать новый индекс в Series?

Можно присвоить новый список или Series атрибуту .index: my_series.index = new_index_list. Длина new_index_list должна совпадать с длиной Series.

22. Как работать с временными рядами в Series?

Pandas предоставляет мощные инструменты для работы с временными рядами, включая:

- Создание Series с DatetimeIndex.
- Выбор данных по дате или диапазону дат.
- Передискретизацию (resampling) временных рядов.
- Расчет скользящих средних.
- Обработку сезонности.

23. Как преобразовать строковые даты в формат DatetimeIndex?

Использовать функцию pd.to_datetime(): dates = pd.to_datetime(date_strings). Затем использовать dates в качестве индекса при создании Series.

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

– Использовать .loc[] с диапазоном дат: my_series.loc['2024-01-01':'2024-01-31']

– Использовать булеву индексацию: my_series[(my_series.index >= '2024-01-01') & (my_series.index <= '2024-01-31')]

25. Как загрузить данные из CSV-файла в Series? Сначала загрузить данные в DataFrame с помощью pd.read_csv(), а затем выбрать нужный столбец, чтобы создать Series: df = pd.read_csv('my_data.csv'); my_series = df['my_column'].

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

После загрузки в DataFrame: df = pd.read_csv('my_data.csv', index_col='date_column'). Если уже загрузили, можно использовать .set_index(): df = df.set_index('date_column'). Затем создать Series: my_series = df['my_column']

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

Для расчета скользящего среднего (moving average). Он сглаживает временные ряды, уменьшая шум и выделяя тренды.

28. Как работает метод .pct_change()? Какие задачи он решает? Метод .pct_change() вычисляет процентное изменение между текущим и

предыдущим элементом Series. Он используется для анализа темпов роста, волатильности и других изменений в данных.

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

– .rolling(): Полезен для анализа трендов и сезонности во временных рядах, сглаживания шумов и выделения основных закономерностей.

– .pct_change(): Полезен для анализа темпов роста, волатильности, изменений цен, продаж и других экономических показателей.

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

NaN (Not a Number) появляются в Series в результате:

– Отсутствующих данных в исходном наборе данных.

– Неопределенных математических операций (например, деление на ноль).

– Преобразования типов данных, когда невозможно представить значение (например, преобразование строки "hello" в число).

– .fillna() или .dropna() для обработки, используя либо присвоение значения по умолчанию, либо удаление.

Вывод: в ходе лабораторной работы познакомились с основами работы с библиотекой pandas, в частности, а также ознакомились со структурой данных Series.