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-3

Выполнил:
Пугачев Кирилл Дмитриевич
2 курс, группа ИТС-б-о-23-1,
11.03.02 Инфокоммуникационные
технологии и системы связи,
очная форма обучения

_____
(подпись)

Проверила:
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/chillkirill/LABA4AI

Ход работы:

1. Выполнение практических заданий.

```
import pandas as pd

data = [5, 15, 25, 35, 45]
index = ['a', 'b', 'c', 'd', 'e']

series = pd.Series(data, index=index)

print(series)
print(series.dtype)
```

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

Рисунок 1. Практическое задание 1

```
import pandas as pd

data = [12, 24, 36, 48, 60]
index = ['A', 'B', 'C', 'D', 'E']

series = pd.Series(data, index=index)

# Получение элемента с индексом 'C'
element_c = series.loc['C']
print(f"Элемент с индексом 'C': {element_c}")

# Получение третьего элемента
third_element = series.iloc[2]
print(f"Третий элемент: {third_element}")
```

```
Элемент с индексом 'C': 36
Третий элемент: 36
```

Рисунок 2. Практическое задание 2

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

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

# Фильтрация данных с помощью логической индексации
filtered_series = series[series > 20]

print(filtered_series)
```

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

Рисунок 3. Практическое задание 3

1]:

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

# Создание Series со 50 случайными целыми числами от 1 до 100
data = np.random.randint(1, 101, 50)
series = pd.Series(data)

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

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

```
Первые 7 элементов:
0    48
1    15
2    53
3    90
4     2
5    56
6    60
dtype: int32

Последние 5 элементов:
45    21
46    41
47    92
48    82
49    44
dtype: int32
```

Рисунок 4. Практическое задание 4

```
1]: import pandas as pd

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

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

# Преобразование типа данных
series_categorical = series.astype('category')

# Определение типа данных после преобразования
print("Тип данных после преобразования:", series_categorical.dtype)

print("\nSeries после преобразования:")
print(series_categorical)
```

```
Исходный тип данных: object
Тип данных после преобразования: category

Series после преобразования:
0       cat
1       dog
2    rabbit
3    parrot
4      fish
dtype: category
Categories (5, object): ['cat', 'dog', 'fish', 'parrot', 'rabbit']
```

Рисунок 5. Практическое задание 5

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

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

# Проверка на наличие пропущенных значений
nan_mask = series.isnull()

nan_indices = nan_mask[nan_mask].index
print("Индексы элементов NaN:", nan_indices)
```

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

Рисунок 6. Практическое задание 6

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

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

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

    # Замена всех nan на среднее значение
    filled_series = series.fillna(mean_value)

    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. Практическое задание 7

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

     # 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'])

     result = s1 + s2

     print("Результат сложения до замены NaN:\n", result)

     # Замена nan на 0
     result_filled = result.fillna(0)

     print("\nРезультат сложения после замены NaN на 0:\n", result_filled)

     # Объяснение появления nan
     print("\nПочему появляются NaN:")
     print("При сложении Series, Pandas выравнивает данные по индексам.")
     print("Если индекс присутствует только в одном из Series, то в результате сложения получается NaN для этого индекса.")
     print("В данном случае, индекс 'a' есть только в s1, а индекс 'e' есть только в s2, поэтому при сложении на этих позициях получаются NaN.")
```

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

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

Почему появляются NaN:
При сложении Series, Pandas выравнивает данные по индексам.
Если индекс присутствует только в одном из Series, то в результате сложения получается NaN для этого индекса.
В данном случае, индекс 'a' есть только в s1, а индекс 'e' есть только в s2, поэтому при сложении на этих позициях получаются NaN.
```

Рисунок 8. Практическое задание 8

```
import pandas as pd
import numpy as np

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

# Применение функции вычисления квадратного корня
sqrt_series = series.apply(np.sqrt)

print("Исходный Series:\n", series)
print("\nSeries с квадратными корнями:\n", sqrt_series)
```

```
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 9. Практическое задание 9

```
import pandas as pd
import numpy as np

random_series = pd.Series(np.random.randint(50, 151, size=20)) #randint принимает верхнюю границу не включительно, поэтому 151

# Вычисление статистик
total_sum = random_series.sum()
average = random_series.mean()
minimum = random_series.min()
maximum = random_series.max()
standard_deviation = random_series.std()


print("Сумма:", total_sum)
print("Среднее:", average)
print("Минимум:", minimum)
print("Максимум:", maximum)
print("Стандартное отклонение:", standard_deviation)
```

```
Сумма: 1973
Среднее: 98.65
Минимум: 50
Максимум: 141
Стандартное отклонение: 22.35661917760893
```

Рисунок 10. Практическое задание 10

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

date_index = pd.date_range(start='2024-03-01', periods=10, freq='D')
random_values = np.random.randint(10, 101, size=10)  # randint принимает верхнюю границу не включительно, поэтому 101
date_series = pd.Series(random_values, index=date_index)

#  Выбор данных за 5-8 марта
start_date = '2024-03-05'
end_date = '2024-03-08'
selected_data = date_series[start_date:end_date]

print("Исходный Series:")
print(date_series)
print("\nДанные за 5-8 марта:")
print(selected_data)
```

```
Исходный Series:
2024-03-01    15
2024-03-02    62
2024-03-03    19
2024-03-04    73
2024-03-05    16
2024-03-06    84
2024-03-07    95
2024-03-08    19
2024-03-09    16
2024-03-10    56
Freq: D, dtype: int32

Данные за 5-8 марта:
2024-03-05    16
2024-03-06    84
2024-03-07    95
2024-03-08    19
Freq: D, dtype: int32
```

Рисунок 11. Практическое задание 11

[1]:

```python
import pandas as pd

index = ['A', 'B', 'A', 'C', 'D', 'B']
values = [10, 20, 30, 40, 50, 60]
my_series = pd.Series(values, index=index)

#  Проверка уникальности индексов
is_unique = my_series.index.is_unique
print("Индексы уникальны:", is_unique)

#  Группировка и суммирование, если индексы не уникальны
if not is_unique:
    grouped_series = my_series.groupby(my_series.index).sum()
    print("\nSeries после группировки и суммирования:")
    print(grouped_series)
else:
    print("\nSeries с уникальными индексами:")
    print(my_series)
```

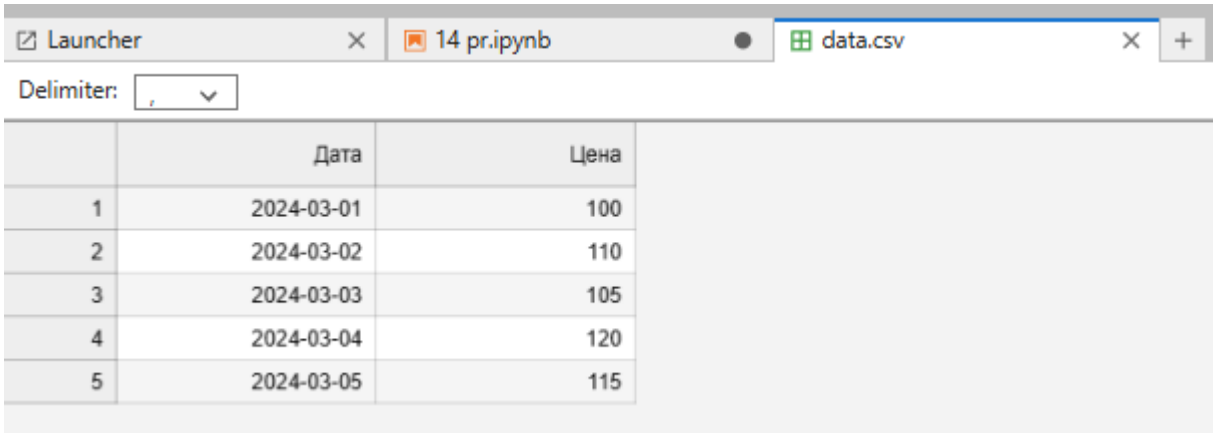
```
Индексы уникальны: False

Series после группировки и суммирования:
A    40
B    80
C    40
D    50
dtype: int64
```

Рисунок 12. Практическое задание 12

[1]:
```python
import pandas as pd

#  Создание Series со строковыми индексами
index_strings = ['2024-03-10', '2024-03-11', '2024-03-12']
values = [100, 200, 300]
my_series = pd.Series(values, index=index_strings)

#  Преобразование индексов в DatetimeIndex
my_series.index = pd.to_datetime(my_series.index)

print("Тип данных индекса:", my_series.index.dtype)

print("\nSeries с DatetimeIndex:")
print(my_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. Практическое задание 13



	Дата	Цена
1	2024-03-01	100
2	2024-03-02	110
3	2024-03-03	105
4	2024-03-04	120
5	2024-03-05	115

Рисунок 14. Практическое задание 14, дальше не получилось

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

date_index = pd.date_range(start='2024-03-01', periods=30, freq='D')
random_values = np.random.randint(50, 151, size=30)  # randint принимает верхнюю границу не включительно
date_series = pd.Series(random_values, index=date_index)

#  график
plt.figure(figsize=(12, 6))
plt.plot(date_series)

plt.title("Изменение случайных значений в марте 2024", fontsize=16)
plt.xlabel("Дата", fontsize=12)
plt.ylabel("Значение", fontsize=12)

#  сетка
plt.grid(True)

plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

Рисунок 15. Практическое задание 15

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

date_index = pd.date_range(start='2024-04-01', periods=15, freq='D')
random_values = np.random.randint(500, 1001, size=15)  #randint принимает верхнюю границу не включительно, поэтому 1001
date_series = pd.Series(random_values, index=date_index)

#  Построение графика и отображение тренда

plt.figure(figsize=(12, 6))

#  график значений
plt.plot(date_series, label='Значения')

#  среднее значение
mean_value = date_series.mean()

#  пунктирная линия среднего значения
plt.axhline(mean_value, color='red', linestyle='--', label=f'Среднее: {mean_value:.2f}') #Добавляем горизонтальную линию.  f-строка для форматирования среднего

plt.title("Изменение случайных значений в апреле 2024", fontsize=16)
plt.xlabel("Дата", fontsize=12)
plt.ylabel("Значение", fontsize=12)

#  сетка
plt.grid(True)

plt.xticks(rotation=45)
plt.tight_layout()

plt.legend()

plt.show()
```

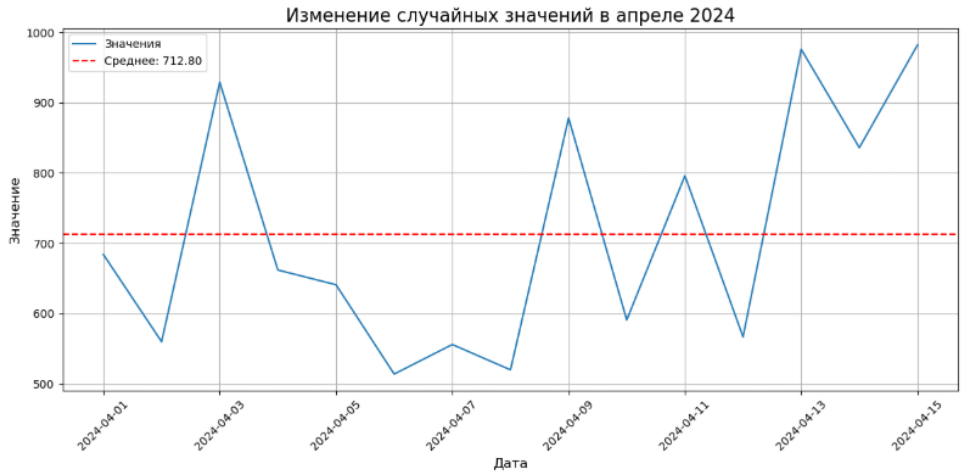


Рисунок 16. Индивидуальное задание 1

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

Ответы на контрольные вопросы:

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

Это одномерный массив с метками (индексами). От списка отличается наличием индекса, который может быть произвольным (не только числовым).

2. Какие типы данных можно использовать для создания Series?

Любые типы данных Python (числа, строки, булевы значения, даты и т.д.).

3. Как задать индексы при создании Series?

При создании Series передать список индексов аргументом index. pd.Series(data, index=index_list).

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

series['index_name'] или series[index_position] (но лучше использовать .loc[] и .iloc[] для однозначности).

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

iloc[]- обращение по числовой позиции (как список), .loc[] - обращение по значению индекса.

6. Как использовать логическую индексацию в Series?

series[series > 0] (возвращает элементы, где условие истинно).

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

Просмотр: .head(n) - первые n элементов, .tail(n) - последние n элементов.

8. Как проверить тип данных элементов Series?

Тип данных: series.dtype

9. Каким способом можно изменить тип данных Series?

Изменение типа данных: .astype(new_type)

10. Как проверить наличие пропущенных значений в Series?

.isnull() (возвращает Series с True/False), .isna() - alias .isnull(), .isnull().any() - есть ли хотя бы один пропуск, .isnull().sum() - количество пропусков

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

Заполнение пропусков:.fillna(value) - заполняет пропуски значением, .fillna(method='ffill'/'bfill') - заполняет предыдущим/следующим значением.

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

fillna() заполняет пропущенные значения, .dropna() удаляет строки (или столбцы в DataFrame) с пропущенными значениями.

13. Какие математические операции можно выполнять с Series?

Математические операции: +, -, , /, , %, сравнения (>, <, ==), и т.д.

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

Векторизация: Векторизированные операции выполняются быстрее и эффективнее, чем циклы Python, т.к. используют оптимизированные библиотечные функции, а не перебор поэлементно.

15. Как применить пользовательскую функцию к каждому элементу Series ?

Применение функции: .apply(function) или .map(function)

16. Какие агрегирующие функции доступны в Series?

В Pandas Series доступны следующие агрегирующие функции:

• sum(): Сумма всех элементов.

• mean(): Среднее значение.

• median(): Медиана.

• min(): Минимальное значение.

• max(): Максимальное значение.

• std(): Стандартное отклонение.

• var(): Дисперсия.

• count(): Количество не-NA/null элементов.

• nunique(): Количество уникальных элементов.

• describe(): Выводит сводную статистику (count, mean, std, min, 25%, 50%, 75%, max).

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

Используйте следующие методы:

• s.min(): Минимальное значение.

• s.max(): Максимальное значение.

• s.mean(): Среднее значение.

• s.std(): Стандартное отклонение.

18. Как сортировать Series по значениям и по индексам?

• По значениям: s.sort_values(ascending=True) (по возрастанию) или s.sort_values(ascending=False) (по убыванию).

• По индексам: s.sort_index(ascending=True) (по возрастанию) или s.sort_index(ascending=False) (по убыванию).

19. Как проверить, являются ли индексы Series уникальными?

Использовать s.index.is_unique. Вернуть True, если индексы уникальны, и False в противном случае.

20. Как сбросить индексы Series и сделать их числовыми?

Использовать s.reset_index(drop=False) (индекс станет столбцом) или s.reset_index(drop=True) (индекс будет потерян и создан новый числовой).

21. Как можно задать новый индекс в Series?

• Присвоить новый список индексов s.index = new_index_list.

• Использовать s.reindex(new_index_list). Этот метод создает новый Series с указанным индексом. Значения будут скопированы, если индексы совпадают. Если индекс отсутствует, будет добавлено NaN.

22. Как работать с временными рядами в Series?

• Индексы должны быть типа DatetimeIndex.

• Можно использовать методы resample(), shift(), diff(), rolling() для анализа и обработки временных рядов.

• Функция pd.date_range() полезна для создания последовательностей дат.

23. Как преобразовать строковые даты в формат DatetimeIndex?

С помощью pd.to_datetime(series_или_список_строк)

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

Если индекс Series типа DatetimeIndex, можно использовать срезы.

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

Использовать pd.read_csv('filename.csv', squeeze=True, index_col='имя_столбца_индекса'). squeeze=True преобразует DataFrame в Series, если в CSV только один столбец данных.

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

При чтении CSV использовать index_col='имя_столбца'

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

Метод .rolling() создает "скользящее окно" по данным, а .mean() вычисляет среднее значение в этом окне. Полезно для сглаживания временных рядов и выявления трендов. Например, .rolling(window=7).mean() вычислит скользящее среднее за последние 7 периодов.

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

Метод .pct_change() вычисляет процентное изменение между текущим и предыдущим элементом в Series. Он решает задачу анализа относительных изменений данных, например, для анализа роста или падения финансовых показателей.

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

• .rolling(): Анализ трендов во временных рядах, сглаживание шума, выявление сезонности.

• .pct_change(): Анализ изменений в процентах (рост, падение), сравнение относительных изменений, финансовый анализ.

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

NaN (Not a Number) появляются в Series, когда:

• Данные отсутствуют или повреждены.

• В результате математических операций (например, деление на ноль).

• При объединении Series с разными индексами, где нет соответствующих значений. • При использовании reindex() с новыми индексами.