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 ПРАКТИЧЕСКОЙ РАБОТЕ №4
дисциплины
«Искусственный интеллект и машинное обучение»
Вариант 15

Выполнил:
Степанов Артем Сергеевич
2 курс, группа ИВТ-б-о-23-2,
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.

Порядок выполнения работы:

Ссылка на репозиторий GitHib:

https://github.com/ArtemStepanovNkey/AI-university/tree/main

1. Задания практической работы.

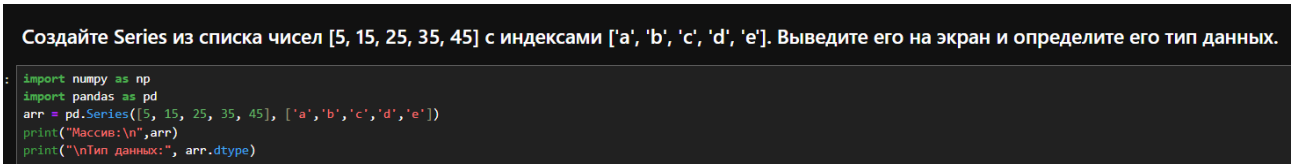

Рисунок 1 – Задание №1. Создание Series из списка

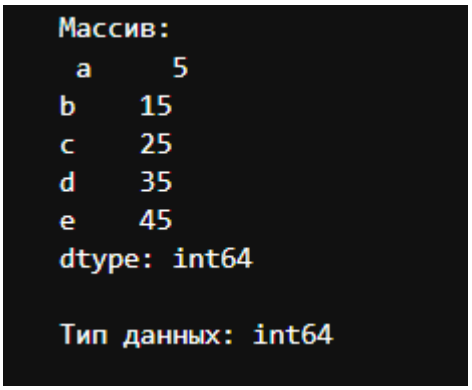

Рисунок 2 – Результат работы программы к заданию №1

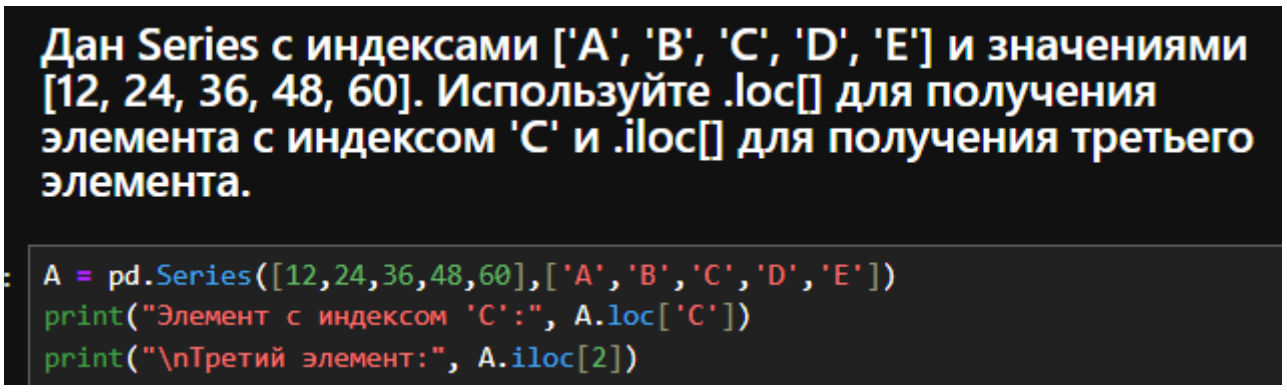

Рисунок 3 – Задание №2. Получение элемента Series

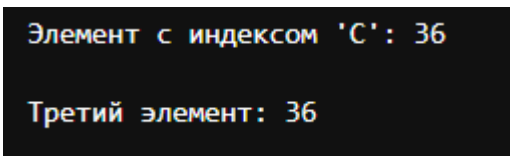

Рисунок 4 – Результат работы программы к заданию №2

Создайте Series из массива NumPy np.array([4, 9, 16, 25, 36, 49, 60]). Выберите только те элементы массива, которые больше 20, и выведите результат. ¶

```
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)
```

Рисунок 5 – Задание №3. Фильтрация данных с помощью логической индексации

```
Исходный массив:
 0     4
1     9
2    16
3    25
4    36
5    48
6    60
dtype: int64

Массив с элементами больше 20:
 3    25
4    36
5    48
6    60
dtype: int64
```

Рисунок 6 – Результат работы программы к заданию №3

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

```
W = pd.Series(np.random.randint(1, 101, size=50))
print("Первые 7 элементов:\n",W.head(7))
print("\nПоследние 5 элементов:\n",W.tail(5))
```

Рисунок 7 – Задание №4. Просмотр первых и последних элементов

```
Первые 7 элементов:
 0    15
1    71
2    95
3    48
4    98
5    75
6    39
dtype: int64

Последние 5 элементов:
 45     4
46    94
47    92
48    11
49    88
dtype: int64
```

Рисунок 8 – Результат работы программы к заданию №4

Создайте Series из списка ['cat', 'dog', 'rabbit', 'parrot', 'fish']. Определите тип данных с помощью .dtype, затем преобразуйте его в category с помощью .astype().

```python
animals = pd.Series(['cat', 'dog', 'rabbit', 'parrot', 'fish'])
print("Тип данных до преобразования:", animals.dtype)
animals = animals.astype('category')
print("Тип данных после преобразования:", animals.dtype)
```

Рисунок 9 – Задание №5. Определение типа данных Series

```
Тип данных до преобразования: object
Тип данных после преобразования: category
```

Рисунок 10 – Результат работы программы к заданию №5

Создайте Series с данными [1.2, np.nan, 3.4, np.nan, 5.6, 6.8]. Напишите код, который проверяет, есть ли в Series пропущенный значения (NaN), и выведите индексы таких элементов.

```python
nan = pd.Series([1.2, np.nan, 3.4, np.nan, 5.6, 6.8])
print("Исходный массив:\n",nan)
filtered_nan = nan[nan.isna()].index
print("\nИндексы пропущенных значений: ",list(filtered_nan))
```

Рисунок 11 – Задание №6. Проверка пропущенных значений

```
Исходный массив:
 0    1.2
1    NaN
2    3.4
3    NaN
4    5.6
5    6.8
dtype: float64

Индексы пропущенных значений:  [1, 3]
```

Рисунок 12 – Результат работы программы к заданию №6

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

```python
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",filte
```

Рисунок 13 – Задание №7. Заполнение пропущенных значений

```
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 14 – Результат работы программы к заданию №7

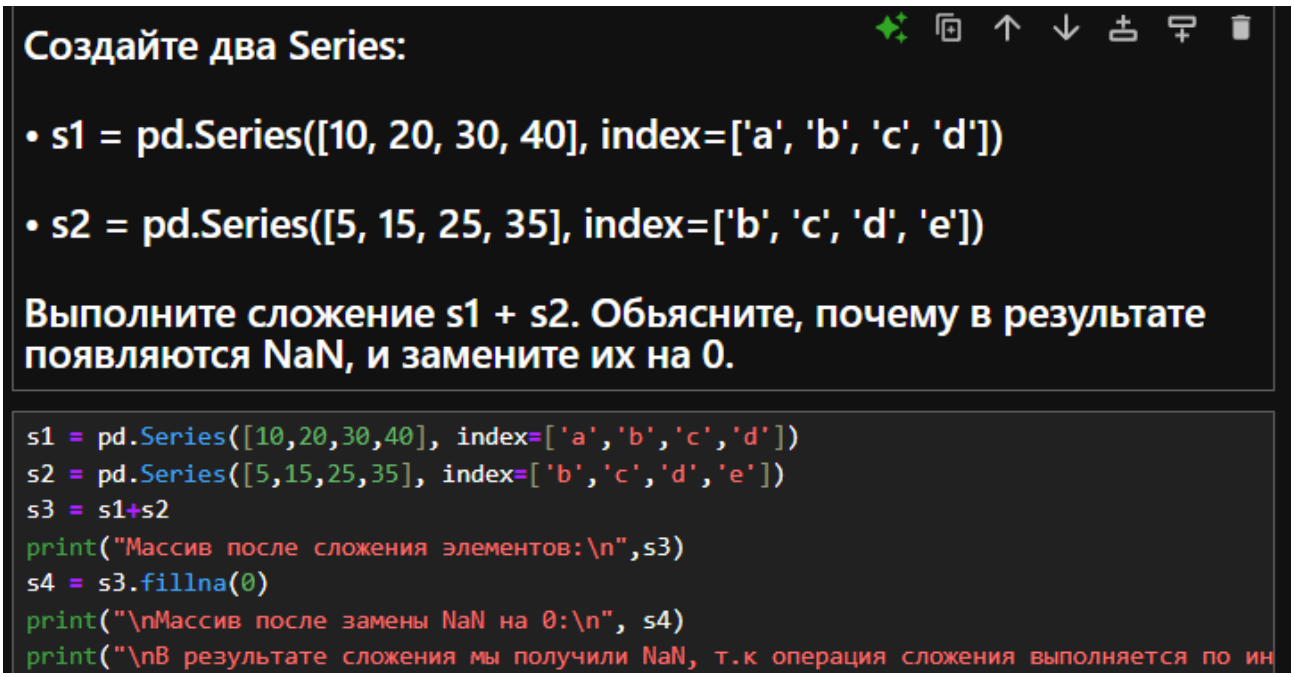

Рисунок 15 – Задание №8. Арифметические операции с Series

```
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

В результате сложения мы получили NaN, т.к операция сложения выполняется по индексам,
поскольку индекс 'a' и 'e' отсутствовали в первом и втором массиве соответственно, они
получили значение NaN.
```

Рисунок 16 – Результат работы программы к заданию №8

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). ¶

```python
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)
```

Рисунок 17 – Задание №9. Применение функции к Series

```
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 18 – Результат работы программы к заданию №9

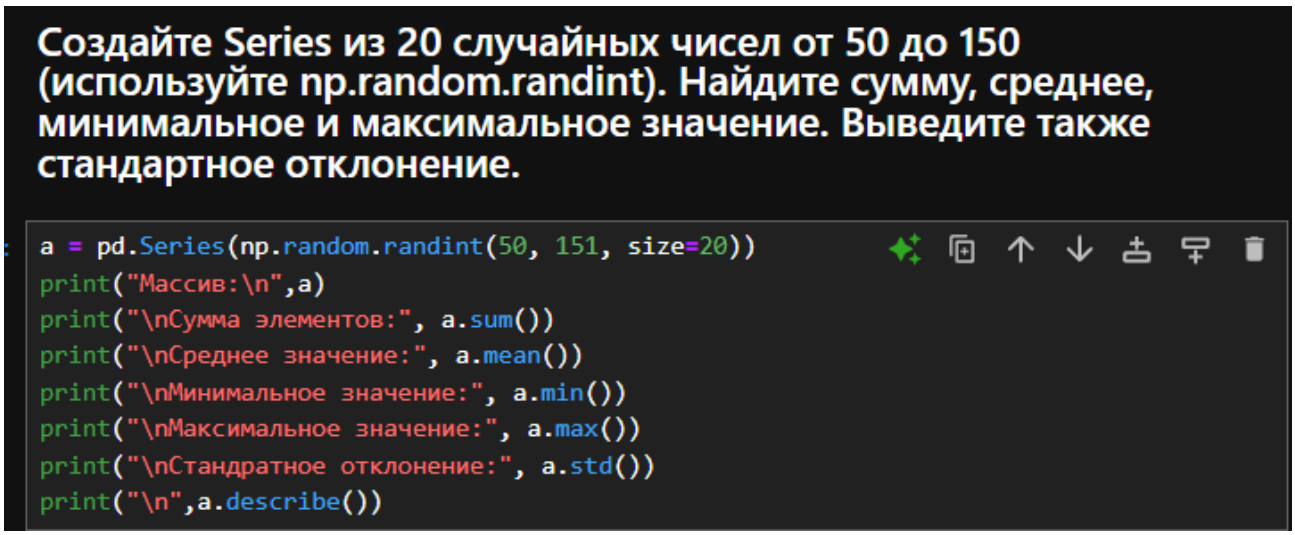

Рисунок 19 – Задание №10. Основные статистические методы

```
Массив:
 0     148
1      62
2      57
3      60
4     121
5     111
6      85
7      78
8     144
9     111
10     57
11     90
12    145
13     70
14     88
15     61
16    125
17     80
18    147
19    104
dtype: int64

Сумма элементов: 1944

Среднее значение: 97.2

Минимальное значение: 57

Максимальное значение: 148

Стандратное отклонение: 32.57154069364689

 count     20.000000
mean      97.200000
std       32.571541
min       57.000000
25%       68.000000
50%       89.000000
75%      122.000000
max      148.000000
dtype: float64
```

Рисунок 20 – Результат работы программы к заданию №10

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)
```

Рисунок 21 – Задание №11. Работа с временными рядами

```
Исходные данные:
 2024-03-01     70
2024-03-02     10
2024-03-03     35
2024-03-04    100
2024-03-05     18
2024-03-06     61
2024-03-07     16
2024-03-08     16
2024-03-09     97
2024-03-10     50
Freq: D, dtype: int64

Данные за 5-8 марта:
 2024-03-05    18
2024-03-06    61
2024-03-07    16
2024-03-08    16
Freq: D, dtype: int64
```

Рисунок 22 – Результат работы программы к заданию №11

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)
```

Рисунок 23 – Задание №12. Проверка уникальности индексов

```
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 24 – Результат работы программы к заданию №12

Создайте Series, где индексами будут строки ['2024-03-10', '2024-03-11', '2024-03-12'], а значениями [100, 200, 300]. Преобразуйте индексы в DatatimeIndex и выведите тип данных индекса.

```
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)
```

Рисунок 25 – Задание №13. Преобразование строковых дат в DatetimeIndex

```
Тип данных индекса до преобразования: object

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

Рисунок 26 – Результат работы программы к заданию №13

Создайте CSV-файл data.csv со следующими данными:

1 Дата, Цена

2 2024-03-01, 100

3 2024-03-02, 110

4 2024-03-03, 105

5 2024-03-04, 120

6 2024-03-05, 115

Прочитайте файл и создайте Series, используя "Дата" в качестве индекса.

```
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)
```

Рисунок 27 – Задание №14. Чтение данных из CSV-файла

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

Рисунок 28 – Результат работы программы к заданию №14

Создайте Series, где индексами будут даты с 1 по 30 марта 2024 года, а значениями - случайные числа от 50 до 150. Постройте график значений с помощью matplotlib. Добавьте заголовок подписи осей и сетку

```
import matplotlib.pyplot as plt
s = pd.Series(np.random.uniform(50,150,size=30),index=pd.date_range(start='2024-03-01',
s.plot(kind='bar',color='green')
plt.xlabel('Дата')
plt.ylabel('Выручка')
plt.title('Продажи за март 2025')
plt.grid()
plt.show()
```

Рисунок 29 – Задание №15. Построение графика на основе Series

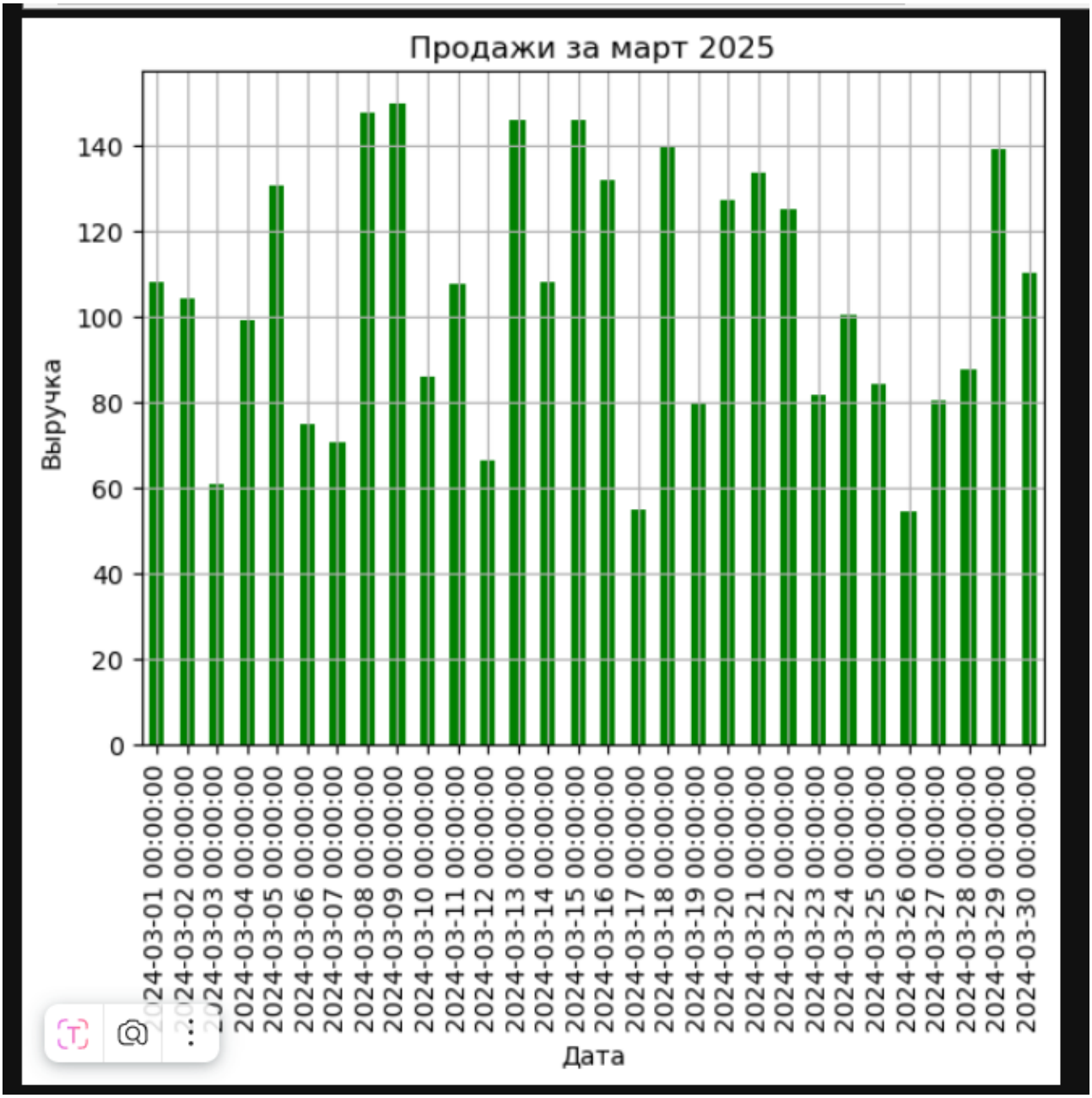

Рисунок 30 – Результат работы программы к заданию №15

2. Индивидуальное задание.

Индивидуальное задание

Вариант 15.

Создайте CSV-файл water_usage.csv со следующими данными:

Прочитайте файл, установите DatetimeIndex , найдите среднее дневное потребление и постройте график с выделением дней выше среднего.

Дата,Потребление воды (м³)

2024-10-01,12000

2024-10-02,12500

2024-10-03,11800

2024-10-04,13000

2024-10-05,13500

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


data = {
    'Дата': ['2024-10-01', '2024-10-02', '2024-10-03', '2024-10-04', '2024-10-05'],
    'Потребление': [12000, 12500, 11800, 13000, 13500]
}
pd.DataFrame(data).to_csv('water_usage.csv', index=False)

df = pd.read_csv('water_usage.csv', parse_dates=['Дата'])
series = df.set_index('Дата')['Потребление']

mean_cons = series.mean()
plt.figure(figsize=(10, 6))

# Создаем список цветов для каждого столбца
colors = ['red' if val > mean_cons else 'green' for val in series]

bars = plt.bar(series.index, series.values, color=colors, width=0.6)

# Добавляем линию среднего
plt.axhline(y=mean_cons, color='red', linestyle='--',
           linewidth=2, label=f'Среднее: {mean_cons} м³')
plt.title('Потребление воды в городе', pad=20, fontsize=14)
plt.xlabel('Дата', labelpad=10, fontsize=12)
plt.ylabel('Потребление воды (м³)', labelpad=10, fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.legend(fontsize=10)
plt.xticks(rotation=45, ha='right')


plt.tight_layout()
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2., height,
            f'{height}',
            ha='center', va='bottom')

plt.show()
```

Рисунок 31 – Индивидуальное задание

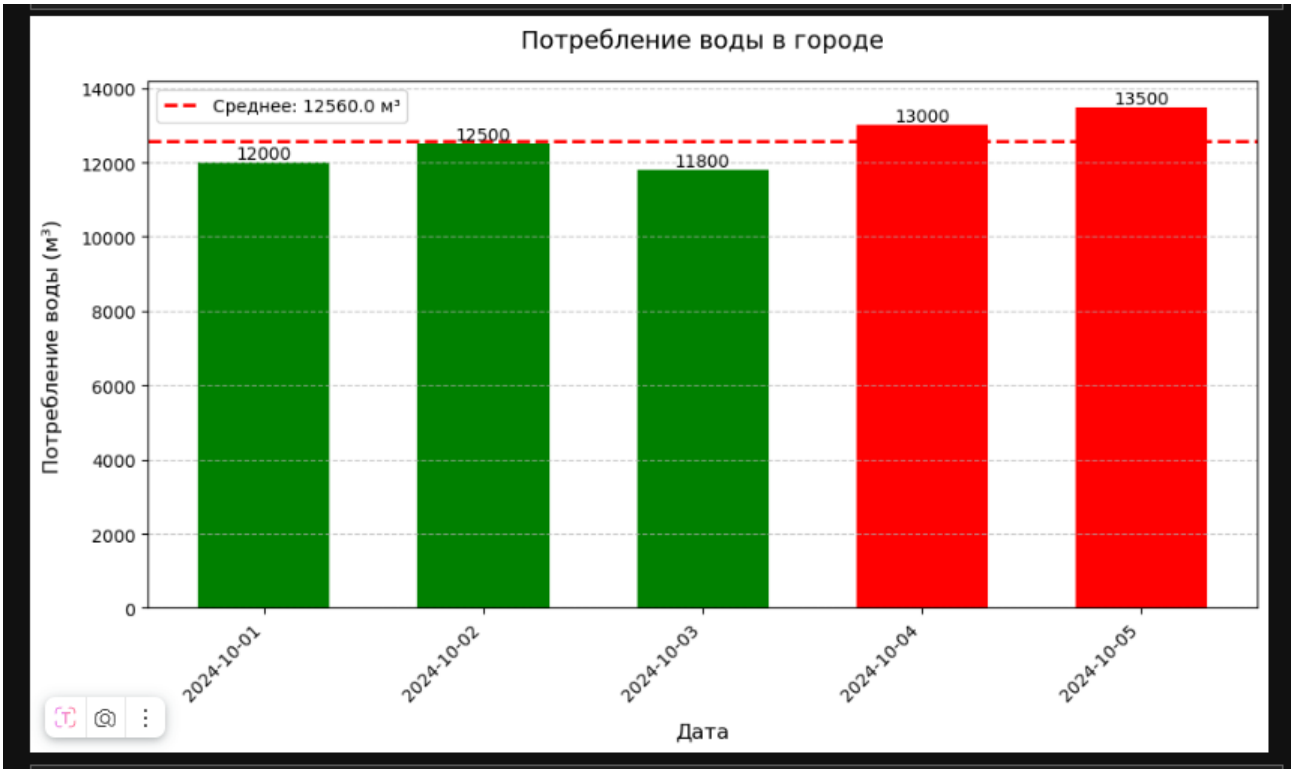

Рисунок 32 – Результат работы программы к индивидуальному заданию

Контрольные вопросы:

1. Что такое pandas.Series и чем она отличается от списка в Python? - pandas.Series — это одномерный массив с метками (индексами), поддерживающий различные типы данных. В отличие от списка, Series обладает индексами, поддерживает векторные операции и работает быстрее.

2. Какие типы данных можно использовать для создания Series? - Числовые (int, float), строковые (str), булевы (bool), временные (datetime), объекты (object) и категории (category).

3. Как задать индексы при создании Series? - Указать параметр index при создании Series: `pd.Series(data, index=custom_index)`.

4. Каким образом можно обратиться к элементу Series по его индексу? - Использовать `series[index]` или `series.loc[index]`.

5. В чем разница между .iloc[] и loc[] при индексации Series? - `.iloc[]` работает с порядковыми номерами (0, 1, 2), а `.loc[]` использует заданные индексы.

6. Как использовать логическую индексацию в Series? - Применять условия: `series[series > 10]` вернет только элементы больше 10.

7. Какие методы можно использовать для просмотра первых и последних элементов Series? - `series.head(n)` и `series.tail(n)`.

8. Как проверить тип данных элементов Series? - `series.dtype`.

9. Каким способом можно изменить тип данных Series? - `series.astype(new_type)`.

10. Как проверить наличие пропущенных значений в Series? - `series.isna()` или `series.isnull()`.

11. Какие методы используются для заполнения пропущенных значений в Series? - `series.fillna(value)`.

12. Чем отличается метод .fillna() от .dropna()? - `.fillna()` заменяет NaN значением, а `.dropna()` удаляет строки с NaN.

13. Какие математические операции можно выполнять с Series? - Сложение, вычитание, умножение, деление, логарифм, возведение в степень и др.

14. В чем преимущество векторизированных операций по сравнению с циклами Python? - Они работают быстрее, так как используют оптимизированные C-библиотеки.

15. Как применить пользовательскую функцию к каждому элементу Series? - Использовать `series.apply(func)`.

16. Какие агрегирующие функции доступны в Series? - `sum()`, `mean()`, `median()`, `std()`, `min()`, `max()`.

17. Как узнать минимальное, максимальное, среднее и стандартное отклонение в Series? - `series.min()`, `series.max()`, `series.mean()`, `series.std()`.

18. Как сортировать Series по значениям и по индексам? - `series.sort_values()` для значений, `series.sort_index()` для индексов.

19. Как проверить, являются ли индексы Series уникальными? - `series.index.is_unique`.

20. Как сбросить индексы Series и сделать их числовыми? - `series.reset_index(drop=True)`.

21. Как можно задать новый индекс в Series? - `series.set_index(new_index)`.

22. Как работать с временными рядами в Series? - Преобразовать индекс в `DatetimeIndex` и использовать `.resample()`, `.rolling()`.

23. Как преобразовать строковые даты в формат DatetimeIndex? - `pd.to_datetime(series)`.

24. Каким образом можно выбрать данные за определенный временной диапазон? - `series['2023-01-01':'2023-12-31']`.

25. Как загрузить данные из CSV-файла в Series? - `pd.read_csv('file.csv', usecols=['column'], squeeze=True)`.

26. Как установить один из столбцов CSV-файла в качестве индекса Series? - `pd.read_csv('file.csv', index_col='column')`.

27. Для чего используется метод .rolling().mean() в Series? - Для вычисления скользящего среднего.

28. Как работает метод .pct_change()? Какие задачи он решает? - Вычисляет процентное изменение между соседними значениями, полезно в анализе трендов.

29. В каких ситуациях полезно использовать .rolling() и .pct_change()? - `.rolling()` для сглаживания данных, `.pct_change()` для анализа динамики изменений.

30. Почему NaN могут появляться в Series, и как с ними работать? - NaN появляются из-за отсутствующих данных, преобразований типов, деления на 0. Удалять `.dropna()`, заполнять `.fillna()`.

Вывод: в ходе практической работы мы познакомились с основами работы с библиотекой pandas, в частности, со структурой данных Series.