

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»

Выполнил:
Хохлачев Вадим Александрович
2 курс, группа ИТС-б-о-23-1,
11.03.02 «Инфокоммуникационные
технологии и системы связи», 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: познакомиться с основами работы с библиотекой pandas, в частности, со структурой данных Series

Порядок выполнения работы:

1. Ознакомились с теоретическим материалом
2. Создание репозитория

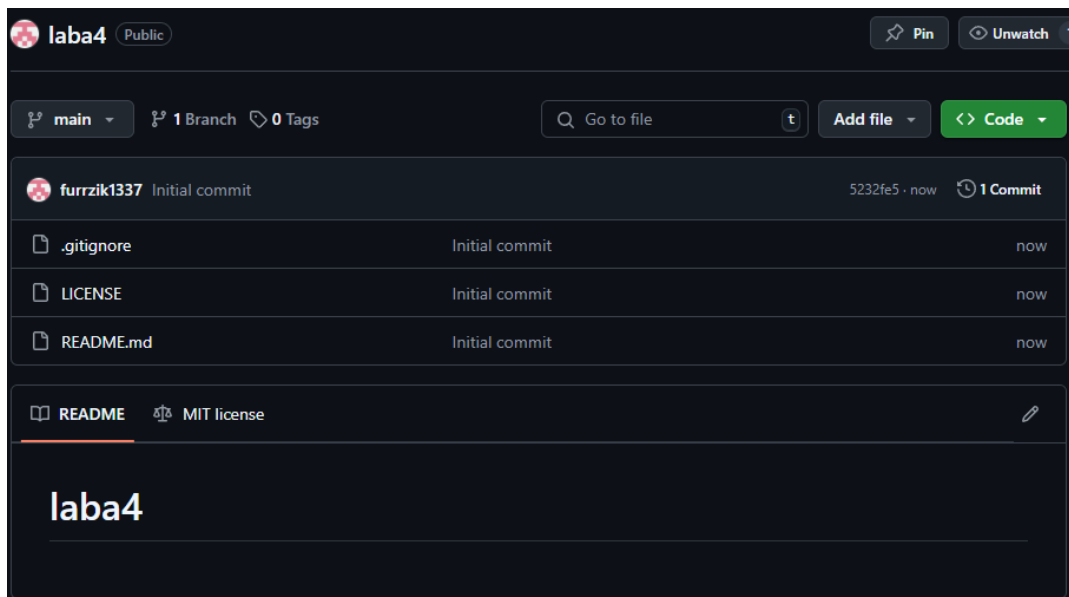


Рисунок 1. Репозиторий

3. Выполнено клонирование репозитория

```
C:\Users\furrzik>git clone https://github.com/furrzik1337/laba4.git
Cloning into 'laba4'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (5/5), done.
C:\Users\furrzik>
```

Рисунок 2. Клонирование

4. Выполнено практическое задание 1

```
import pandas as pd

# Создаем Series
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:", series.dtype)
```

a	5
b	15
c	25
d	35
e	45

dtype: int64

Тип данных Series: int64

Рисунок 3. Создание Series из списка

5. Выполнено практическое задание 2

```
import pandas as pd

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

# Получаем элемент с индексом 'C' с помощью .loc[]
element_loc = series.loc['C']
print("Элемент с индексом 'C' (.loc[]):", element_loc)

# Получаем третий элемент с помощью .iloc[]
element_iloc = series.iloc[2]
print("Третий элемент (.iloc[]):", element_iloc)
```

Элемент с индексом 'C' (.loc[]): 36

Третий элемент (.iloc[]): 36

Рисунок 4. Получение элемента series

6. Выполнено практическое задание 3

```
import pandas as pd
import numpy as np

# Создаем массив NumPy
numpy_array = np.array([4, 9, 16, 25, 36, 49, 64])

# Создаем Series из массива NumPy
series = pd.Series(numpy_array)

# Выбираем элементы, которые больше 20
fseries = series[series > 20]

# Выводим результат
print(fseries)
```

3 25
4 36
5 49
6 64
dtype: int32

Рисунок 5. Фильтрация данных

7. Выполнено практическое задание 4

```
import pandas as pd
import numpy as np

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

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

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

Первые 7 элементов:
0 48
1 55
2 50
3 73
4 42
5 86
6 65
dtype: int32

Последние 5 элементов:
45 93
46 95
47 47
48 45
49 21
dtype: int32

Рисунок 6. Просмотр первых и последних элементов

8. Выполнено практическое задание 5

```
import pandas as pd

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

# Определяем тип данных Series
print("Тип данных до преобразования:", series.dtype)

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

# Определяем тип данных после преобразования
print("Тип данных после преобразования:", categorical_series.dtype)
```

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

Рисунок 7. Определение типа данных

9. Выполнено практическое задание 6

```
import pandas as pd
import numpy as np

# Создаем Series с пропущенными значениями (NaN)
data = [1.2, np.nan, 3.4, np.nan, 5.6, 6.8]
series = pd.Series(data)

# Проверяем наличие пропущенных значений (NaN) с помощью .isna() или .isnull()
# is_na = series.isna()
is_na = series.isnull() # Эквивалентно series.isna()

# Выводим индексы элементов с пропущенными значениями
nan_indices = is_na[is_na].index
print("Индексы элементов с пропущенными значениями (NaN):", nan_indices)
```

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

Рисунок 8. Проверка пропущенных значений

10. Выполнено практическое задание 7

```
import pandas as pd
import numpy as np

# Создаем Series с пропущенными значениями (NaN)
data = [1.2, np.nan, 3.4, np.nan, 5.6, 6.8]
series = pd.Series(data)

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

# Заменяем NaN на среднее значение
series_filled = series.fillna(mean_value)

# Выводим результат
print(series_filled)
```

0	1.20
1	4.25
2	3.40
3	4.25
4	5.60
5	6.80

dtype: float64

Рисунок 9. Заполнение пропущенных значений

11. Выполнено практическое задание 8

```
import pandas as pd

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

# Создаем Series s2
s2 = pd.Series([5, 15, 25, 35], index=['b', 'c', 'd', 'e'])

# Выполняем сложение s1 + s2
result = s1 + s2
print("Результат сложения (с NaN):")
print(result)

# Заменяем NaN на 0
result_filled = result.fillna(0)
print("\nРезультат сложения (NaN заменены на 0):")
print(result_filled)
```

Результат сложения (с NaN):

a	NaN
b	25.0
c	45.0
d	65.0
e	NaN

dtype: float64

Результат сложения (NaN заменены на 0):

a	0.0
b	25.0
c	45.0
d	65.0
e	0.0

dtype: float64

Рисунок 10. Арифметические операции

12. Выполнено практическое задание 9

```

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 с помощью .apply(np.sqrt)
sqrt_series = series.apply(np.sqrt)

# Выводим результат
print(sqrt_series)

```

0 1.414214
1 2.000000
2 2.449490
3 2.828427
4 3.162278
dtype: float64

Рисунок 11. Применение функции

13. Выполнено практическое задание 10

```

import pandas as pd
import numpy as np

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

# Находим сумму
total_sum = series.sum()
print("Сумма:", total_sum)

# Находим среднее
mean_value = series.mean()
print("Среднее:", mean_value)

# Находим минимальное значение
min_value = series.min()
print("Минимум:", min_value)

# Находим максимальное значение
max_value = series.max()
print("Максимум:", max_value)

# Находим стандартное отклонение
std_deviation = series.std()
print("Стандартное отклонение:", std_deviation)

```

Сумма: 2098
Среднее: 104.9
Минимум: 54
Максимум: 150
Стандартное отклонение: 34.223568364198194

Рисунок 12. Листинг программы

14. Выполнено практическое задание 11


```
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) # Случайные числа от 10 до 100
series = pd.Series(data, index=dates)

# Выбираем данные за 5-8 марта включительно с помощью .loc
selected_data = series.loc['2024-03-05':'2024-03-08']

# Выводим выбранные данные
print(selected_data)
```

2024-03-05	37
2024-03-06	48
2024-03-07	97
2024-03-08	55

Freq: D, dtype: int32

Рисунок 13. Работа с временными рядами

15. Выполнено практическое задание 12

```
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, являются ли индексы уникальными
is_unique = series.index.is_unique
print("Индексы уникальны:", is_unique)

# Если индексы не уникальны, группируем и складываем значения
if not is_unique:
    grouped_series = series.groupby(series.index).sum()
    print("\nСгруппированный Series (повторяющиеся индексы сложены):")
    print(grouped_series)
```

Индексы уникальны: False

Сгруппированный Series (повторяющиеся индексы сложены):

A	40
B	80
C	40
D	50

dtype: int64

Рисунок 14. Проверка уникальности индексов

16. Выполнено практическое задание 13

```
import pandas as pd

# Создаем Series со строковыми индексами
index = ['2024-03-10', '2024-03-11', '2024-03-12']
data = [100, 200, 300]
series = pd.Series(data, index=index)

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

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

Тип данных индекса: datetime64[ns]
```

Рисунок 15. Преобразование строковых дат

17. Выполнено практическое задание 14

```
: import pandas as pd

# Данные для записи в CSV-файл
data = {'Дата': ['2024-03-01', '2024-03-02', '2024-03-03', '2024-03-04', '2024-03-05'],
        'Цена': [100, 110, 105, 120, 115]}

# Создаем DataFrame из данных
df = pd.DataFrame(data)

# Записываем DataFrame в CSV-файл с разделителем запятой
df.to_csv('data.csv', index=False, sep=',')

print("Файл data.csv успешно создан.")

Файл data.csv успешно создан.
```

Рисунок 16. Листинг программы

	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 17. Результат

18. Выполнено практическое задание 15

```

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

# Создаем Series с датами в качестве индексов и случайными числами в качестве значений
dates = pd.date_range(start='2024-03-01', periods=30, freq='D')
data = np.random.randint(50, 151, 30)
series = pd.Series(data, index=dates)

# Создаем график
plt.figure(figsize=(12, 6)) # Задаем размер графика
plt.plot(series)

# Добавляем заголовок
plt.title('Изменение значений в марте 2024 года', fontsize=16)

# Добавляем подписи осей
plt.xlabel('Дата', fontsize=12)
plt.ylabel('Значение', fontsize=12)

# Добавляем сетку
plt.grid(True)

# Форматируем подписи оси x для лучшей читаемости
plt.xticks(rotation=45, ha='right')

# Отображаем график
plt.tight_layout() # Автоматически корректирует параметры подграфика, чтобы график поместился в область Figure
plt.show()

```

Рисунок 18. Листинг программы



Рисунок 19. Результат

19. Выполнено индивидуальное задание

```

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

# Создаем Series с датами в качестве индексов и случайными значениями популярности
dates = pd.date_range('2024-06-01', periods=30, freq='D')
data = np.random.randint(0, 101, 30) # Значения от 0 до 100
series = pd.Series(data, index=dates)

# Находим даты, когда популярность превышала 80
high_popularity_dates = series[series > 80].index

# Создаем график
plt.figure(figsize=(12, 6))
plt.plot(series, label='Популярность')

# Выделяем точки на графике, где популярность превышала 80
plt.plot(high_popularity_dates, series[high_popularity_dates], 'ro', label='Популярность > 80') # 'ro' - красные кружки

# Добавляем заголовок и подписи осей
plt.title('Популярность запроса за последние 30 дней', fontsize=16)
plt.xlabel('Дата', fontsize=12)
plt.ylabel('Популярность', fontsize=12)

# Добавляем сетку
plt.grid(True)

# Форматируем подписи оси x для лучшей читаемости
plt.xticks(rotation=45, ha='right')

# Добавляем легенду
plt.legend()

# Отображаем график
plt.tight_layout()
plt.show()

```

Рисунок 20. Листинг программы



Рисунок 21. Результат

20. Зафиксировал изменения на репозитории






 furrzik1337 notebooks	f121d64 · now	🕒 2 Commits
 notebooks	notebooks	now
 .gitignore	Initial commit	1 hour ago
 LICENSE	Initial commit	1 hour ago
 README.md	Initial commit	1 hour ago

Рисунок 22. Репозиторий

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

Ссылка на GitHub: <https://github.com/furrzik1337/laba4>