**Практическое задание 2**

Необходимо загрузить файл Google Colab **Задание\_1.ipynb** на Google диск

**Задание 1**. Изучить методы одномерных массивов Pandas из данного файла. В colab ноутбуке Задание\_1.ipynb добавьте код примеров из данного файла.

Для этого в colab ноутбуке для каждого метода добавьте текстовую ячейку с названием метода и ячейку кода, в которую поместите пример кода этого метода. Выполните код.

**Задание 2**. Далее, из документации Pandas <https://pandas.pydata.org/docs/reference/series.html> выберите любой метод для Series и добавьте в colab ноутбук из документации описание этого метода и пример кода.

**Задание 3.** В ноутбуке решите следующую задачу.

Напишите функцию, которая принимает на вход N чисел и возвращает Series, в которой каждый элемент является результатом деления этого элемента на среднее значение всех элементов.

На вход программе подается число N, после чего каждое число на отдельной строке.

**Sample Input:**

3

1

2

3

**Sample Output:**

0 0.5

1 1.0

2 1.5

dtype: float64

За основу возьмите следующий код:

import pandas as pd

def solution():

list\_ = []

n =

s1 =

mean =

result =

print(result)

solution()

**Одномерные массивы Pandas**

Одномерные массивы Pandas - это объекты Series, которые представляют собой одномерные массивы с метками. Они могут содержать данные разных типов, таких как целые числа, числа с плавающей точкой и строки.

Создания одномерного массива Pandas:

import pandas as pd

data = [1, 2, 3, 4, 5]

s = pd.Series(data)

print(s)

Вывод:

0 1

1 2

2 3

3 4

4 5

dtype: int64

В этом примере мы импортировали библиотеку Pandas и создали список данных. Затем мы использовали функцию pd.Series (), чтобы создать объект Series из списка данных. Наконец, мы напечатали объект Series.

Обратите внимание на то, что объект Series содержит метки индекса (0, 1, 2, 3 и 4), которые соответствуют каждому элементу в списке данных.

Теперь давайте рассмотрим пример получения доступа к элементам одномерного массива Pandas:

import pandas as pd

data = [1, 2, 3, 4, 5]

s = pd.Series(data)

print(s[0])

print(s[1:4])

Вывод:

1

1 2

2 3

3 4

dtype: int64

В этом примере мы использовали квадратные скобки для доступа к элементу одномерного массива Pandas по индексу. Мы также использовали квадратные скобки и диапазон для доступа к нескольким элементам одномерного массива Pandas.

# Методы Series

В этом разделе мы рассмотрим различные методы, доступные для работы с сериями Pandas.

Серии Pandas - это одномерные маркированные массивы, которые могут содержать данные любого типа. Они похожи на списки или массивы в Python, но обладают дополнительной функциональностью и возможностями, которые делают их идеальными для анализа данных. Эти методы позволят вам манипулировать данными и анализировать их различными способами, включая сортировку, фильтрацию и агрегирование данных. Эффективно используя эти методы, вы сможете получить ценные сведения о своих данных и принять более обоснованные решения.

Метод **concat()**

Объединение объектов pandas .

Аргументы для этого метода:

objs: обязательный аргумент - последовательность или объектов Series или DataFrame.

пример:

s1 = pd.Series([1, 2])

s2 = pd.Series([3, 4])

s3 = concat(s1, s2)

print(s3)

Выводом для этого кода будет

0 1

1 2

0 3

1 4

dtype: int64

# Метод mean()

вычисляет среднее значение всех элементов Series. Например, если у нас есть Series s, то чтобы вычислить среднее значение всех его элементов, нужно написать следующее:

import pandas as pd

s = pd.Series([10, 20, 30])

mean\_value = s.mean()

print(mean\_value)

Вывод:

20.0

# Метод fillna()

Используется для заполнения значений NA/NaN в DataFrame или Series. DataFrame – двумерная таблица Pandas

 Синтаксис этого метода такой:

dataframe.fillna(value, method, axis, inplace, limit, downcast)

Аргументы этого метода:

* value - скалярное значение, словарь, Series или DataFrame, которое используется для заполнения пропусков (например, 0). Можно также указать разные значения для разных индексов (для Series) или столбцов (для DataFrame). Значения, которых нет в словаре/Series/DataFrame, не будут заполнены. Это значение не может быть списком.
* method - строка, которая определяет метод заполнения пропусков. Может быть одним из следующих значений: backfill, bfill, pad, ffill или None (по умолчанию). Эти значения означают следующее:
  + pad / ffill: заполняет пропуски последним действительным наблюдением вперед до следующего действительного.
  + backfill / bfill: заполняет пропуски следующим действительным наблюдением.
* axis - ось, вдоль которой заполняются пропуски. Может быть 0 или ‘index’ (по умолчанию) для индекса или 1 или ‘columns’ для столбцов. Для Series этот аргумент не используется.
* inplace - логическое значение, которое определяет, изменять ли исходный объект или возвращать новый. По умолчанию False, то есть возвращается новый объект.
* limit - целое число или None (по умолчанию), которое определяет максимальное количество последовательных NaN значений для заполнения вперед/назад (если указан метод) или максимальное количество значений вдоль всей оси, где будут заполнены NaN значения (если метод не указан). Должно быть больше 0, если не None.
* downcast - словарь или None (по умолчанию), который определяет, какие типы данных пытаться привести к более низким типам (например, float64 к int64, если возможно).

Пример кода:

import pandas as pd

import numpy as np

# Создаем DataFrame с пропусками

df = pd.DataFrame([[np.nan, 2, np.nan, 0],

[3, 4, np.nan, 1],

[np.nan, np.nan, np.nan, np.nan],

[np.nan, 3, np.nan, 4]],

columns=list('ABCD'))

# Выводим исходный DataFrame

print(df)

A B C D

0 NaN 2.0 NaN 0.0

1 3.0 4.0 NaN 1.0

2 NaN NaN NaN NaN

3 NaN 3.0 NaN 4.0

# Заполняем все пропуски нулями

df1 = df.fillna(0)

print(df1)

Выводом будет:

A B C D

0 0.0 2.0 0.0 0.0

1 3.0 4.0 0.0 1.0

2 0.0 0.0 0.

# Метод apply()

apply(func, convert\_dtype=True, args=(), \*\*kwds): Используйте этот метод для применения любой функции к каждому элементу в вашем объекте Series

Аргументы для этого метода:

* **func**: Функция, которую нужно применить.
* **convert\_dtype**: Преобразовать или нет тип данных результата.
* **args**: Дополнительные аргументы, которые нужно передать функции.

Вот пример:

import pandas as pd

# create a series

s = pd.Series([1, 2, 3])

# apply a function to each element in the series

s = s.apply(lambda x: x \* 2)

print(s)

Выводом для этого кода будет

0 2

1 4

2 6

dtype: int64

# Так же есть схожий **метод** transform()

метод transform() применяет функцию к каждой группе в объекте DataFrame или Series и возвращает объект того же размера, что и исходный. Он также может использоваться для заполнения пропущенных значений в группах. В отличие от метода apply(), метод transform() всегда сохраняет форму исходного объекта.

Вот пример кода для метода apply():

import pandas as pd

s = pd.Series([1, 2, 3])

def add\_one(x):

return x + 1

s.apply(add\_one)

Результат:

0 2

1 3

2 4

dtype: int64

А вот пример кода для метода transform():

import pandas as pd

s = pd.Series([1, None, 3])

def fillna\_mean(x):

return x.fillna(x.mean())

s.transform(fillna\_mean)

Результат:

0 1.0

1 2.0

2 3.0

dtype: float64

# Метод drop()

drop(labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise'): Используйте этот метод для удаления указанных меток из строк или столбцов.

Аргументы для этого метода:

* **labels**: Метки, которые нужно удалить.
* **axis**: Удалить строки или столбцы.
* **index**: Псевдоним для axis=0.
* **columns**: Псевдоним для axis=1.
* **level**: Для MultiIndex уровень, с которого будут удалены метки.
* **inplace**: Изменять или нет исходный объект Series.

Вот пример:

import pandas as pd

# создадим фрейм данных

df = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})

# отбросим колонку 'B'

df = df.drop(columns=['B'])

print(df)

Выводом для этого кода будет

A

0 1

1 2

# Метод drop\_duplicates()

drop\_duplicates(subset=None, keep=‘first’, inplace=False, ignore\_index=False): Используйте этот метод для удаления дублирующихся значений из вашего объекта Series.

Аргументы для этого метода:

* **subset**: Столбцы, которые нужно учитывать при определении дубликатов.
* **keep**: Какие дубликаты (если есть) нужно сохранить.
* **inplace**: Изменять или нет исходный объект Series.

Вот пример:

import pandas as pd

# создадим серию с дублирующимися значениями

s = pd.Series([1, 2, 2])

# удалим дубликаты

s = s.drop\_duplicates()

print(s)

Выводом для этого кода будет

0 1

1 2

dtype: int64

# Метод equals()

equals(other):Определит, равны ли два Series.

Аргументы для этого метода:

* **other**: Другой объект Series для сравнения.

Вот пример:

import pandas as pd

# создадим две серии с одинаковыми значениями

s1 = pd.Series([1, 2])

s2 = pd.Series([1, 2])

# проверим, одинаковы ли они

result = s1.equals(s2)

print(result)

Выводом для этого кода будет

True

# Метод isin()

isin(values): Определите, содержится ли каждый элемент серии в значениях.

Аргументы для этого метода:

* **values**: Список значений для проверки.

Вот пример:

import pandas as pd

# создадим серию и список значений для проверки

s = pd.Series([1, 2])

values = [1]

# проверим, есть ли каждый элемент в серии в списке значений

result = s.isin(values)

print(result)

Выводом этого кода будет

0 True

1 False

dtype: bool

# Метод map()

map(arg, na\_action=None): Сопоставить значения серии в соответствии с входным соответствием.

Аргументы для этого метода:

* **arg**: Функция, словарь или объект Series, который отображает входные значения.
* **na\_action**: Какое действие следует предпринять при обнаружении нулевого значения.

Вот пример:

import pandas as pd

# создать серию и словарь для сопоставления с новыми значениями

s = pd.Series(['cat', 'dog'])

mapping = {'cat': 'feline', 'dog': 'canine'}

# сопоставить значения в серии с новыми значениями с помощью .map()

s = s.map(mapping)

print(s)

Выводом этого кода будет

0 feline

1 canine

dtype: object

# Метод replace()

replace(to\_replace=None, value=None, inplace=False, limit=None, regex=False, method='pad'): Заменяет значения, указанные в to\_replace, на value.

Аргументы для этого метода:

* **to\_replace**: Значение(я), которое(ые) будет заменено(ы).
* **value**: Значение(я), которое(ые) заменят to\_replace.

Вот пример:

import pandas as pd

# создайте ряд с некоторыми значениями для замены их другими значениями.

s = pd.Series([1, 2])

to\_replace = [1]

value = [3]

# заменим значения в серии на новое значение

s = s.replace(to\_replace=to\_replace,value=value)

print(s)

Выводом для этого кода будет

0 3

1 2

dtype: int64

# Метод sort\_values()

sort\_values(axis=0, ascending=True): Сортировка по значениям вдоль любой оси.

Аргументы для этого метода:

* **axis**: Хотите отсортировать по строкам (0) или столбцам (1).
* **ascending**: Задает порядок сортировки. Чтобы задать порядок «по убыванию», следует указать ascending=False. (По умолчанию sort\_values() сортирует по убыванию)

Вот пример:

import pandas as pd

# Создаем series

s = pd.Series([2, 1])

# сортировка значений в серии

s = s.sort\_values()

print(s)

Выводом для этого кода будет

1 1

0 2

dtype: int64

### Дополнительно.

# Есть схожий метод sort\_index()

 sort\_index() используется для сортировки объекта Series по индексу, а функция sort\_values() используется для сортировки объекта Series по значениям.

Если вы хотите отсортировать объект Series по индексу, то используйте функцию sort\_index(). Если вы хотите отсортировать объект Series по значениям, то используйте функцию sort\_values().

Вот пример кода для объекта Series:

import pandas as pd

s = pd.Series([3, 1, 2])

s.sort\_index()

Результат:

0 3

1 1

2 2

dtype: int64

# Метод div()

1. делит каждый элемент Series на заданное число или на элементы другой Series. Например, если у нас есть Series s1 и число n, то чтобы получить новую Series, в которой каждый элемент s1 разделен на n, нужно написать следующее:

import pandas as pd

s1 = pd.Series([10, 20, 30])

n = 5

s\_new = s1.div(n)

print(s\_new)

Вывод:

0 2.0

1 4.0

2 6.0

dtype: float64

# Метод sub()

1. вычитает из каждого элемента Series заданное число или элементы другой Series. Например, если у нас есть Series s1 и число n, то чтобы получить новую Series, в которой из каждого элемента s1 вычтено n, нужно написать следующее:

import pandas as pd

s1 = pd.Series([10, 20, 30])

n = 5

s\_new = s1.sub(n)

print(s\_new)

Вывод:

0 5

1 15

2 25

dtype: int64

# Метод mul()

умножает каждый элемент Series на заданное число или на элементы другой Series. Например, если у нас есть Series s1 и s2, то чтобы получить новую Series, в которой каждый элемент s1 умножен на соответствующий элемент s2, нужно написать следующее:

import pandas as pd

s1 = pd.Series([1, 2, 3])

s2 = pd.Series([4, 5, 6])

s\_new = s1.mul(s2)

print(s\_new)

Вывод:

0 4

1 10

2 18

dtype: int64

# Метод pow()

возводит каждый элемент Series в заданную степень. Например, если у нас есть Series s, то чтобы получить новую Series, в которой каждый элемент s возведен в квадрат, нужно написать следующее:

import pandas as pd

s = pd.Series([1, 2, 3])

s\_new = s.pow(2)

print(s\_new)

Вывод:

0 1

1 4

2 9

dtype: int64