



Python. Статьи по языку программирования Пайтон, Питон

Использование библиотеки Pandas для преобразования данных



infozone.group · 23.05.2023

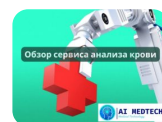


Содержание статьи



1. Использование библиотеки Pandas для преобразования данных
 - 1.1. 1. Фильтрация данных
 - 1.2. 2. Поворот данных (pivot и unpivot)
 - 1.3. 3. Расчет новых столбцов
 - 1.4. 4. Расчет столбца с использованием агрегатного значения по всему столбцу Total
 - 1.5. 5. Аналог оконных функций в Pandas
 - 1.6. 6. Объединение двух и более DataFrame: конкатенация, join, merge
2. Подходы к очистке данных с помощью библиотеки Pandas
 - 2.1. 1. Обнаружение и удаление дубликатов

Latest Posts



Раскрывая
Тайны
Здоровья:
Онлайн
Расшифров
ка
Результатов
Анализа
Крови

○ 14 Min Read



- 2.2. 2. Обработка отсутствующих значений
- 2.3. 3. Обработка выбросов
- 3. Использование group by в Pandas Python для анализа данных
 - 3.1. Основы операции group by
 - 3.2. Группировка данных и вычисление агрегатных значений
 - 3.3. Примеры использования операции group by
 - 3.3.1. Пример 1: Вычисление суммарного значения по группам
 - 3.3.2. Пример 2: Вычисление среднего значения и количества по группам
 - 3.4. Фильтрация данных по группам
 - 3.5. Итоги
- 4. 5 функций Python Pandas: clip() diff() get_dummies() from_dummies() transform()
 - 4.1. clip()
 - 4.2. diff()
 - 4.3. get_dummies()
 - 4.4. from_dummies()
 - 4.5. transform()

Использование библиотеки Pandas для преобразования данных

Pandas — это мощная библиотека для анализа данных в языке программирования Python. Она предоставляет гибкие и эффективные инструменты для обработки и преобразования данных. В этой статье мы рассмотрим несколько ключевых операций, которые можно выполнить с помощью Pandas для преобразования данных.

1. Фильтрация данных

Одна из наиболее распространенных операций с данными — это фильтрация. Pandas предоставляет удобные методы для выбора подмножества данных на основе заданных условий. Для фильтрации данных в Pandas можно использовать операторы сравнения (>, <, == и т. д.) или методы, такие как `isin()` и `str.contains()`, для фильтрации по значениям столбцов или строк.

Пример фильтрации данных по условию:

```
1. import pandas as pd
2.
3. # Создание DataFrame
4. data = {'Name': ['John', 'Emily', 'Ryan', 'Jessica'],
5.         'Age': [25, 30, 35, 28],
6.         'City': ['New York', 'Paris', 'London', 'Sydney']}
7. df = pd.DataFrame(data)
8.
9. # Фильтрация данных по возрасту больше 30
10. filtered_df = df[df['Age'] > 30]
11.
12. print(filtered_df)
```

Вывод:

1.	Name	Age	City
2.	Ryan	35	London

2. Поворот данных (pivot и unpivot)



Путь дата аналитика (Junior Middle Senior Data Analyst)

3 Min Read



Почему люди с депрессией отталкивают других?

3 Min Read



Что такое Cython и CPython? И как они связаны с Python? В чем разница?

5 Min Read

Blog Categories

IT Crypto / Blockchain / API for Developers

IT Аналитика. Big Data. Machine Learning

IT заметки

Artificial Intelligence in Medicine

Nginx. Docker. Docker-compose

Python. Статьи по языку программирования Пайтон, Питон

Архитектура ПО. Чистая архитектура. Подходы и фреймворки для проектирования архитектуры

Искусственный интеллект (Artificial intelligence)

Практики Devops — Docker, Kubernetes. Devops инженер

Статьи по Node.JS

SHARE ON



READ NEXT



Получить количество подписчиков со страницы youtube через Python

Pandas предоставляет методы `pivot()` и `melt()` для поворота данных между «широким» (wide) и «длинным» (long) форматами. Операция `pivot` позволяет преобразовать уникальные значения столбца в новые столбцы, а `melt` — объединить несколько столбцов в один и добавить столбец с их значениями.

Пример использования метода `pivot()`:

```
1. import pandas as pd
2.
3. # Создание DataFrame
4. data = {'Name': ['John', 'Emily', 'Ryan', 'Jessica'],
5.         'Subject': ['Math', 'Science', 'Math', 'Science'],
6.         'Score': [90, 85, 92, 88]}
7. df = pd.DataFrame(data)
8.
9. # Поворот данных
10. pivot_df = df.pivot(index='Name', columns='Subject', values='Score')
11.
12. print(pivot_df)
```

Вывод:

	Subject	Math	Science
Name			
Emily		NaN	85.0
Jessica		NaN	88.0
John		90.0	NaN
Ryan		92.0	NaN

Пример использования метода `melt()`:

```
1. import pandas as pd
2.
3. # Создание DataFrame
4. data = {'Name': ['John', 'Emily', 'Ryan', 'Jessica'],
5.         'Math': [90, 85, 92, 88],
6.         'Science': [85, 92, 88, 90]}
7. df = pd.DataFrame(data)
8.
9. # Обратный поворот данных
10. melted_df = pd.melt(df, id_vars=['Name'], value_vars=['Math', 'Science'],
11.                    var_name='Subject', value_name='Score')
12.
13. print(melted_df)
```

Вывод:

		Name	Subject	Score
0	John	Math	90	
1	Emily	Math	85	
2	Ryan	Math	92	
3	Jessica	Math	88	
4	John	Science	85	
5	Emily	Science	92	
6	Ryan	Science	88	
7	Jessica	Science	90	

3. Расчет новых столбцов

Pandas предоставляет простой способ расчета новых столбцов на основе существующих данных. Вы можете использовать арифметические операторы или функции для выполнения вычислений над столбцами.

Пример расчета нового столбца на основе существующих столбцов:

```
1. import pandas as pd
2.
3. # Создание DataFrame
4. data = {'Name': ['John', 'Emily', 'Ryan', 'Jessica'],
5.         'Age': [25, 30, 35, 28],
6.         'Salary': [50000, 60000, 70000, 55000]}
```

SMM. Маркетинг.

Продвижение в социальных сетях

Автоматизация торговли. Торговые системы

Блокчейн. Что такое блокчейн?

Заметки трейдера

Заработок в интернете. Как заработать в сети интернет

Инвестиции. Финансовые рынки. Трейдинг

ИТ технологии. IT technologies

Как создать сайт на WordPress

Карьера в ИТ

Консалтинг. Работа консультанта

Кризис. Работа в кризис

Маркетинг. Системы продаж

Мотивация. Личностный рост. Personal Growth. Motivation

Новости по криптовалюте. Обзоры по крипте

Психология

Руководство командой. TeamLead. Навыки тимлида

Стратегический анализ и планирование. Что такое KPI?

Управление проектами. Product Management

Финансы. Рынок облигаций. Корпоративные финансы

```

7. df = pd.DataFrame(data)
8.
9. # Расчет нового столбца
10. df['Bonus'] = df['Salary'] * 0.1
11.
12. print(df)

```

Вывод:

		Name	Age	Salary	Bonus
2.	0	John	25	50000	5000.0
3.	1	Emily	30	60000	6000.0
4.	2	Ryan	35	70000	7000.0
5.	3	Jessica	28	55000	5500.0

4. Расчет столбца с использованием агрегатного значения по всему столбцу Total

Pandas предоставляет мощные инструменты для агрегации данных. Вы можете использовать методы, такие как `sum()`, `mean()`, `min()`, `max()` и другие, для вычисления агрегатных значений по столбцам. Также можно использовать метод `apply()` для применения пользовательских функций к столбцам или строкам DataFrame.

Пример расчета столбца с использованием агрегатного значения по всему столбцу Total:

```

1. import pandas as pd
2.
3. # Создание DataFrame
4. data = {'Name': ['John', 'Emily', 'Ryan', 'Jessica'],
5.         'Math': [90, 85, 92, 88],
6.         'Science': [85, 92, 88, 90]}
7. df = pd.DataFrame(data)
8.
9. # Расчет столбца с использованием суммы значений по столбцу Total
10. df['Total'] = df[['Math', 'Science']].sum(axis=1)
11.
12. print(df)

```

Вывод:

		Name	Math	Science	Total
2.	0	John	90	85	175
3.	1	Emily	85	92	177
4.	2	Ryan	92	88	180
5.	3	Jessica	88	90	178

5. Аналог оконных функций в Pandas

Оконные функции позволяют выполнять вычисления на группе строк, определенной по некоторым условиям, и возвращать результаты в виде столбца с тем же количеством строк. В Pandas вы можете использовать методы `rolling()`, `expanding()` и `ewm()` для реализации оконных вычислений.

Пример использования метода `rolling()` для вычисления скользящей средней:

```

1. import pandas as pd
2.
3. # Создание DataFrame
4. data = {'Date': ['2022-01-01', '2022-01-02', '2022-01-03', '2022-01-04',
5.                 '2022-01-05'],
6.         'Value': [10, 20, 30, 40, 50]}
7. df = pd.DataFrame(data)

```

```

8. # Вычисление скользящей средней
9. df['Moving Average'] = df['Value'].rolling(window=2).mean()
10.
11. print(df)

```

Вывод:

		Date	Value	Moving Average
2.	0	2022-01-01	10	NaN
3.	1	2022-01-02	20	15.0
4.	2	2022-01-03	30	25.0
5.	3	2022-01-04	40	35.0
6.	4	2022-01-05	50	45.0

6. Объединение двух и более DataFrame: конкатенация, join, merge

Pandas предоставляет несколько методов для объединения двух или более DataFrame. Вы можете использовать методы `concat()`, `join()` и `merge()` для выполнения операций объединения.

Пример использования метода `concat()` для объединения двух DataFrame:

```

1. import pandas as pd
2.
3. # Создание DataFrame 1
4. data1 = {'Name': ['John', 'Emily'],
5.          'Age': [25, 30]}
6. df1 = pd.DataFrame(data1)
7.
8. # Создание DataFrame 2
9. data2 = {'Name': ['Ryan', 'Jessica'],
10.          'Age': [35, 28]}
11. df2 = pd.DataFrame(data2)
12.
13. # Объединение DataFrame
14. merged_df = pd.concat([df1, df2])
15.
16. print(merged_df)

```

Вывод:

		Name	Age
2.	0	John	25
3.	1	Emily	30
4.	0	Ryan	35
5.	1	Jessica	28

Пример использования метода `merge()` для объединения двух DataFrame по общему столбцу:

```

1. import pandas as pd
2.
3. # Создание DataFrame 1
4. data1 = {'Name': ['John', 'Emily'],
5.          'Subject': ['Math', 'Science']}
6. df1 = pd.DataFrame(data1)
7.
8. # Создание DataFrame 2
9. data2 = {'Name': ['John', 'Emily'],
10.          'Score': [90, 85]}
11. df2 = pd.DataFrame(data2)
12.
13. # Объединение DataFrame
14. merged_df = pd.merge(df1, df2, on='Name')
15.
16. print(merged_df)

```

Вывод:

		Name	Subject	Score
2.	0	John	Math	90

Pandas предоставляет множество функций и методов, которые позволяют эффективно обрабатывать, фильтровать и преобразовывать данные. Используйте эти инструменты для улучшения вашего анализа данных и работы с ними.

Подходы к очистке данных с помощью библиотеки Pandas

Очистка данных является важным этапом в процессе анализа данных. Библиотека Pandas предоставляет мощные инструменты для выполнения различных операций по очистке и предварительной обработке данных. В этом разделе мы рассмотрим некоторые подходы к очистке данных с использованием библиотеки Pandas.

1. Обнаружение и удаление дубликатов

Дубликаты данных могут исказить результаты анализа и привести к неточным выводам. Pandas предоставляет методы для обнаружения и удаления дубликатов из DataFrame. Метод `duplicated()` позволяет обнаружить повторяющиеся строки, а метод `drop_duplicates()` удаляет дубликаты.

Пример удаления дубликатов:

```
1. import pandas as pd
2.
3. # Создание DataFrame с дубликатами
4. data = {'Name': ['John', 'Emily', 'John', 'Ryan', 'Emily'],
5.         'Age': [25, 30, 25, 35, 30]}
6. df = pd.DataFrame(data)
7.
8. # Обнаружение и удаление дубликатов
9. df_no_duplicates = df.drop_duplicates()
10.
11. print(df_no_duplicates)
```

Вывод:

	Name	Age
0	John	25
1	Emily	30
3	Ryan	35

2. Обработка отсутствующих значений

Отсутствующие значения (NaN или None) могут привести к проблемам при анализе данных. Pandas предлагает различные методы для обработки отсутствующих значений. Методы `isnull()` и `notnull()` позволяют обнаружить отсутствующие значения, а методы `dropna()` и `fillna()` позволяют удалить или заполнить их соответственно.

Пример заполнения отсутствующих значений:

```
1. import pandas as pd
2.
3. # Создание DataFrame с отсутствующими значениями
4. data = {'Name': ['John', 'Emily', 'Ryan', None, 'Jessica'],
5.         'Age': [25, None, 35, 28, 30]}
```

```

6. df = pd.DataFrame(data)
7.
8. # Заполнение отсутствующих значений средним значением
9. df_filled = df.fillna(df.mean())
10.
11. print(df_filled)

```

Вывод:

1.		Name	Age
2.	0	John	25.0
3.	1	Emily	29.5
4.	2	Ryan	35.0
5.	3	Jessica	28.0
6.	4	Jessica	30.0

3. Обработка выбросов

Выбросы (аномальные значения) могут исказить статистические показатели и привести к неправильным выводам. Pandas предоставляет методы для обработки выбросов, такие как `quantile()` для определения пороговых значений и `clip()` для ограничения значений в заданном диапазоне.

Пример обработки выбросов:

```

1. import pandas as pd
2.
3. # Создание DataFrame с выбросами
4. data = {'Name': ['John', 'Emily', 'Ryan', 'Jessica'],
5.         'Age': [25, 30, 200, 28]}
6. df = pd.DataFrame(data)
7.
8. # Определение порогового значения для выбросов
9. threshold = df['Age'].quantile(0.95)
10.
11. # Замена выбросов на пороговое значение
12. df['Age'] = df['Age'].clip(upper=threshold)
13.
14. print(df)

```

Вывод:

1.		Name	Age
2.	0	John	25.0
3.	1	Emily	30.0
4.	2	Ryan	30.0
5.	3	Jessica	28.0

В зависимости от конкретных требований и характера данных, могут быть применены и другие методы и техники очистки данных, такие как замена значений, удаление столбцов и строк и т. д.

Использование group by в Pandas Python для анализа данных

Анализ данных часто требует группировки данных по определенным категориям или условиям. Библиотека Pandas в Python предоставляет мощный инструментарий для группировки данных с помощью операции `group by`. В этой статье мы рассмотрим основы использования операции `group by` в Pandas и различные аспекты анализа данных, которые можно выполнять с помощью этой операции.

Основы операции group by

Операция `group by` в Pandas позволяет группировать данные по одному или нескольким столбцам и выполнять агрегатные функции на каждой группе.

Синтаксис операции `group by` выглядит следующим образом:

```
1. grouped = df.groupby('column')
```

где `df` — `DataFrame`, а `'column'` — имя столбца, по которому происходит группировка.

После группировки данных можно применять различные агрегатные функции, такие как `sum()`, `mean()`, `count()`, `min()`, `max()`, `median()` и другие.

Группировка данных и вычисление агрегатных значений

Процесс группировки данных в Pandas состоит из нескольких шагов:

1. Группировка данных по выбранным столбцам:

```
1. grouped = df.groupby('column')
```

2. Вычисление агрегатных значений для каждой группы:

```
1. result = grouped.agg({'column1': 'function1', 'column2': 'function2'})
```

Здесь `'column1'` и `'column2'` — это столбцы, для которых вычисляются агрегатные значения, а `'function1'` и `'function2'` — это агрегатные функции, которые нужно применить к этим столбцам.

Примеры использования операции group by

Давайте рассмотрим несколько примеров использования операции `group by` для анализа данных.

Пример 1: Вычисление суммарного значения по группам

```
1. import pandas as pd
2.
3. # Создание DataFrame
4. data = {'Name': ['John', 'Emily', 'Ryan', 'Jessica'],
5.         'Department': ['HR', 'Finance', 'HR', 'Finance'],
6.         'Salary': [50000, 60000, 70000, 55000]}
7. df = pd.DataFrame(data)
8.
9. # Группировка данных по столбцу 'Department' и вычисление суммарного
   значения столбца 'Salary'
10. grouped = df.groupby('Department')
11. result = grouped.agg({'Salary': 'sum'})
12.
13. print(result)
```

Вывод:

1.		Salary
2.	Department	
3.	Finance	115000
4.	HR	120000

Пример 2: Вычисление среднего значения и количества по группам

```
1. import pandas as pd
```



```

2.
3. # Создание DataFrame
4. data = {'Name': ['John', 'Emily', 'Ryan', 'Jessica'],
5.         'Department': ['HR', 'Finance', 'HR', 'Finance'],
6.         'Salary': [50000, 60000, 70000, 55000]}
7. df = pd.DataFrame(data)
8.
9. # Группировка данных по столбцу 'Department' и вычисление среднего
    значения и количества столбца 'Salary'
10. grouped = df.groupby('Department')
11. result = grouped.agg({'Salary': ['mean', 'count']})
12.
13. print(result)

```

Вывод:

	Salary	
	mean	count
Department		
Finance	57500.0	2
HR	60000.0	2

Фильтрация данных по группам

Операция `group by` также позволяет фильтровать данные на основе условий, применяемых к группам. Для этого можно использовать метод `filter()`.

Пример фильтрации данных по группам:

```

1. import pandas as pd
2.
3. # Создание DataFrame
4. data = {'Name': ['John', 'Emily', 'Ryan', 'Jessica'],
5.         'Department': ['HR', 'Finance', 'HR', 'Finance'],
6.         'Salary': [50000, 60000, 70000, 55000]}
7. df = pd.DataFrame(data)
8.
9. # Фильтрация данных по группам с условием, что среднее значение 'Salary'
    в группе больше 55000
10. filtered = df.groupby('Department').filter(lambda x: x['Salary'].mean() >
11.                                             55000)
12. print(filtered)

```

Вывод:

	Name	Department	Salary
0	John	HR	50000
2	Ryan	HR	70000
3	Jessica	Finance	55000

Итоги

Операция `group by` в библиотеке Pandas предоставляет мощный инструмент для группировки данных и выполнения агрегатных вычислений по группам. Вы можете использовать эту операцию для выполнения различных анализов данных, таких как вычисление сумм, средних значений, минимальных и максимальных значений, количества и других. Кроме того, вы можете фильтровать данные на основе условий, применяемых к группам. Используйте операцию `group by` в Pandas для эффективного анализа и обработки ваших данных.

5 функций Python Pandas: `clip()` `diff()` `get_dummies()`

from_dummies() transform()

Pandas — очень популярная библиотека для обработки данных, которая широко используется в науке о данных и аналитике. Он предоставляет широкий спектр функций, которые ускоряют выполнение задач анализа и обработки данных.

Есть некоторые функции, которые используются не очень часто, но весьма полезны для определенных задач. Эти функции могут помочь вам сэкономить время и усилия при работе с данными в Python Pandas.

clip()

Функцию `clip` можно использовать для ограничения значений в `DataFrame` или `Series` указанным диапазоном. Это означает, что любое значение ниже указанного нижнего предела будет установлено на нижний предел, а любое значение выше указанного верхнего предела будет установлено на верхний предел. Это удобный способ обрезать значения до нужного диапазона, эффективно обрабатывая выбросы.

Давайте сначала создадим простой `DataFrame`:

```
1. import numpy as np
2. import pandas as pd
3.
4. df = pd.DataFrame(
5.     np.random.randint(-10, 10, size=(5, 5)),
6.     columns=list("ABCDE")
7. )
8.
9. df
10. # output
11.    A  B  C  D  E
12. 0   3 -1 -1 -3  1
13. 1  -2 -5 -4  3  2
14. 2  -2 -4 -8  6 -8
15. 3  -1 -5 -2  1 -6
16. 4   4 -7  4  8  1
```

Мы можем ограничить нижние значения до -4, поэтому любое значение меньше -4 будет установлено на -4.

```
1. df.clip(lower=-4)
2. # output
3.    A  B  C  D  E
4. 0   3 -1 -1 -3  1
5. 1  -2 -4 -4  3  2
6. 2  -2 -4 -4  6 -4
7. 3  -1 -4 -2  1 -4
8. 4   4 -4  4  8  1
```

Точно так же мы можем ограничить верхнее и нижнее значения одновременно:

```
1. df.clip(lower=-4, upper=4)
2. # output
3.    A  B  C  D  E
4. 0   3 -1 -1 -3  1
5. 1  -2 -4 -4  3  2
6. 2  -2 -4 -4  4 -4
7. 3  -1 -4 -2  1 -4
8. 4   4 -4  4  4  1
```

diff()

Функция `diff` вычисляет разницу между текущим элементом и элементом в предыдущей позиции (или любой другой указанной позиции) в `DataFrame` или `Series`.

По умолчанию функция вычисляет разницу между последовательными строками (т. е. функция использует период, равный 1), но вы можете указать другой период. Мы также можем рассчитать разницу между последовательными столбцами, установив значение параметра `axis` равным 1.

Эта функция особенно полезна для данных временных рядов, чтобы найти изменение во времени.

```
1. df
2. # output
3.    A B C D E
4. 0 2 1 1 0 4
5. 1 5 6 0 4 7
6. 2 9 2 8 8 2
7. 3 6 5 3 4 2
8. 4 7 2 5 6 2
9.
10.
11. # difference between consecutive rows
12. df.diff()
13. # output
14.    A    B    C    D    E
15. 0 NaN NaN NaN NaN NaN
16. 1 3.0 5.0 -1.0 4.0 3.0
17. 2 4.0 -4.0 8.0 4.0 -5.0
18. 3 -3.0 3.0 -5.0 -4.0 0.0
19. 4 1.0 -3.0 2.0 2.0 0.0
20.
21. # difference between the row before
22. df.diff(periods=2)
23. # output
24.    A    B    C    D    E
25. 0 NaN NaN NaN NaN NaN
26. 1 NaN NaN NaN NaN NaN
27. 2 7.0 1.0 7.0 8.0 -2.0
28. 3 1.0 -1.0 3.0 0.0 -5.0
29. 4 -2.0 0.0 -3.0 -2.0 0.0
30.
31. # difference between consecutive columns
32. df.diff(axis=1)
33. # output
34.    A    B    C    D    E
35. 0 NaN -1  0 -1  4
36. 1 NaN  1 -6  4  3
37. 2 NaN -7  6  0 -6
38. 3 NaN -1 -2  1 -2
39. 4 NaN -5  3  1 -4
```

get_dummies()

Функцию `get_dummies` в `pandas` можно использовать для преобразования категориальных переменных в фиктивные переменные. Он возвращает новый `DataFrame` с двоичными столбцами для каждой категории (или отдельного значения), присутствующими в исходных данных, где 1 означает наличие этой категории, а 0 — отсутствие.

Давайте сначала создадим простой `DataFrame` с категориальной переменной.

```
1. import pandas as pd
2.
```

```

3.     df = pd.DataFrame(
4.
5.         {
6.             "age": [45, 53, 60, 42, 34],
7.             "gender": ["Female", "Male", "Female", "Female", "Male"]
8.         }
9.     )
10.
11. df
12. # output
13.   age gender
14. 0  45 Female
15. 1  53 Male
16. 2  60 Female
17. 3  42 Female
18. 4  34 Male

```

Давайте применим `get_dummies` функцию к столбцу пола.

```

1. pd.get_dummies(df)
2.
3. # output
4.   age gender_Female gender_Male
5. 0  45             1           0
6. 1  53             0           1
7. 2  60             1           0
8. 3  42             1           0
9. 4  34             0           1

```

Это удобно при предварительной обработке данных для моделей машинного обучения, поскольку некоторые алгоритмы не принимают категориальные переменные как есть.

from_dummies()

Эта `from_dummies` функция была представлена в Pandas версии 1.5.0. Он выполняет противоположную функцию `get_dummies`.

Мы можем создать категориальную переменную из фиктивных переменных, используя `from_dummies` функцию следующим образом:

```

1. df_new = pd.get_dummies(df)
2.
3. df_new
4. # output
5.   age gender_Female gender_Male
6. 0  45             1           0
7. 1  53             0           1
8. 2  60             1           0
9. 3  42             1           0
10. 4  34             0           1
11.
12.
13. pd.from_dummies(df_new[["gender_Female", "gender_Male"]])
14. # output
15. 0 gender_Female
16. 1 gender_Male
17. 2 gender_Female
18. 3 gender_Female
19. 4 gender_Male

```

transform()

Функцию `transform` можно использовать для выполнения функции над `DataFrame` или `Series`, которая возвращает объект с аналогичным индексом. Он может работать с одним столбцом или с несколькими столбцами.

Это похоже на `apply` функцию, но в отличие от `apply` функции, `transform` может возвращать серию, которая имеет тот же размер, что и входные данные, что делает ее полезной при выполнении векторных операций и широковещании, особенно при использовании с объектами `groupby`.

```
1. import pandas as pd
2. import numpy as np
3.
4. df = pd.DataFrame(
5.     np.random.randint(0, 10, size=(5, 3)),
6.     columns=list("ABC")
7. )
8.
9. df
10. # output
11.   A B C
12. 0 6 8 1
13. 1 8 1 6
14. 2 8 5 0
15. 3 2 5 5
16. 4 5 3 8
17.
18.
19. df.transform([np.sqrt, np.square])
20. # output
```

	A		B		C	
	sqrt	square	sqrt	square	sqrt	square
0	2.449490	36	2.828427	64	1.000000	1
1	2.828427	64	1.000000	1	2.449490	36
2	2.828427	64	2.236068	25	0.000000	0
3	1.414214	4	2.236068	25	2.236068	25
4	2.236068	25	1.732051	9	2.828427	64

5

Рейтинг статьи

★★★★★


TAGS:

pandas data processing

pandas преобразование данных


python pandas

What's your reaction?




Love

1




Sad

0




Happy

0




Sleepy

0




Angry

0



Dead

0



Wink

0

SHARE ON

f









Руководство по написанию чистого кода
в соответствии с PEP 8 на языке Python

Генераторы и итераторы в Python.
Способы применения

 Leave a Reply

 Подписаться ▼



Оставьте первый комментарий!

B *I* U    “ ”   {} [+]



0 КОММЕНТАРИЕВ



You Might Also Enjoy

Python. Статьи по языку программирования Пай

Что такое Cython и CPython? И как они связаны с Python? В чем разница?



infozone.group | 5 Min Read

Python. Статьи по языку программирования Пай

Создание календаря с помощью Python. Как хранить календарь. Примеры использования



infozone.group | 7 Min Read

Python. Статьи по языку программирования Пай

Паттерны разработки на Python: TDD, DDD и событийно-ориентированная архитектура



infozone.group | 6 Min Read

INFOZONE.PRO

Издательство Infozone.Pro