Wyznaczanie sekwencji akcji
Wybrane rozwiązania programistyczne

Marzena Nowakowska

Wydział Zarządzania i Modelowania Komputerowego
Politechnika Świętokrzyska

Budynek C, p. 3.21

spimn@tu.kielce.pl

Metoda apply

Funkcja apply() aplikuje określoną funkcję do zbioru wartości – kolumny lub wiersza ramki danych (DataFrame) lub do całej serii (Series):

Dla ramki: df.apply(funkc, axis=0 lub 1)

funkc – funkcja, która ma być zastosowana

axis = 0 – gdy funkcja funkc ma pracować dla każdej kolumny ramki

axis = 1 – gdy funkcja funkc ma pracować dla każdego wiersza ramki

Dla serii: s.apply(func)

funkcja funkc zostanie **wykonana dla każdego elementu serii**

Funkcja apply() działa poprzez iterację po elementach zbioru wartości.

```
import pandas as pd
df = pd.DataFrame({
    'A': [1, 5, 3],
    'B': [4, 2, 6] })
result = df.apply(max, axis=1)
print(result)
```

```
import pandas as pd
s = pd.Series([-5, 3, -1, 0, 7])
result = s.apply(abs)
print(result)
```

df - DataFrame

Index	A	B
0	1	4
1	5	2
2	3	6

s - Series

Index	0
0	-5
1	3
2	-1
3	0
4	7

Metoda apply w sekwencjach
Grupowanie wg jednego atrybutu

```
kseqs = df_sorted.groupby("CUSTOMER")["ACTION"].apply(list)
```

df_sorted.groupby("CUSTOMER")
Grupowanie wierszy ramki danych df_sorted według kolumny CUSTOMER; wszystkie wiersze należące do tego samego klienta będą traktowane jako jedna grupa.

[ACTION]
W każdej grupie jest wybierana kolumna ACTION. Efektem jest grupowana seria (SeriesGroupBy), która nie udostępnia bezpośrednio konkretnych danych, ale pozwala wykonywać różne operacje na każdej grupie osobno (np. .sum(), .mean(), .first(), .last(), apply(list) itd.).

.apply(list)
Dla każdej grupy, wszystkie wartości z kolumny "ACTION" są zamieniane na listę. Dzięki temu otrzymuje się listę zawierająca sekwencję akcji dla każdego klienta. Każdy element listy to koszyk.

Wynikiem całej instrukcji jest seria danych zawierająca identyfikatory klientów (kolumna indeksu) oraz listy ich koszyków (kolumna wartości mająca nazwę ACTION).

Metoda apply w sekwencjach
Grupowanie wg dwóch atrybutów

```
df_tst = (df_org.groupby(['CUSTOMER', 'TIME'])['ACTION’]
          .apply(lambda x: ', '.join(sorted(x)))
          .reset_index()  )
```

Dzięki nawiasom Python wie, że całe kilkulinijkowe wyrażenie to jedna instrukcja.

df_org.groupby(['CUSTOMER', 'TIME'])['ACTION’]
Grupowanie wierszy ramki danych df_org według dwóch kolumn CUSTOMER i TIME; wszystkie wiersze należące do tego samego klienta i tego samego momentu czasowego będą traktowane jako jedna grupa.
[ACTION]
W każdej grupie jest wybierana kolumna ACTION. Efektem jest grupowana seria (SeriesGroupBy), której działanie jest podane na poprzednim slajdzie).
.apply(lambda x: ', '.join(sorted(x)))
Dla każdej grupy (rekordy z tym samym klientem i tym samym czasem) działa mała jednorazowa funkcja lambda (anonimowa - nie ma nazwy), której argument jest zastępowany przez elementy grupy. Funkcja najpierw porządkuje te elementy automatycznie – sorted (dlaczego to ważne) a następnie łączy je wszystkie w jeden tekst separując je znakami przecinka i spacji. Stąd, każdy klient w swoich momentach czasu ma przypisane działania (koszyki).
.reset_indeks()
Wynik jest serią danych i ma indeksy wielopoziomowe (CUSTOMER, TIME), a kolumna z połączonymi akcjami nie ma jeszcze nazwy. Operacja .reset_index() zamienia indeksy w kolumny i powstaje ramka danych (reset_index() zawsze zwraca ramkę danych, niezależnie od tego, czy pracuje na serii czy ramce danych).

Funkcja Counter

seq_counts = Counter(tuple(seq) for seq in kseqs)

Funkcja zwraca obiekt typu Counter, działający jak słownik, w którym kluczem jest sekwencja akcji, a wartością krotność wystąpienia tej sekwencji

tuple(seq) for seq in kseqs

Wyrażenie generatora, które tworzy obiekt typu *generator*. Generator nie przechowuje danych, tylko tworzy je w trakcie - nie da się powiedzieć, że jego zawartość jest mutowalna lub niemutowalna. To „strumień”, nie struktura danych.
Każdy element, który ww. generator zwraca, jest krotką (tuple) — a krotka w Pythonie jest niemutowalna. Może więc być wykorzystana jako klucz w kolekcji liczników.

Counter(tuple(seq) for seq in kseqs)

Counter z modułu collections działa jak słownik (dict), w którym kluczem jest sekwencja, a wartością liczba jego wystąpień. W Pythonie taki klucz musi być hashowalny.

Pętla działająca jak Counter

```
# Utworzenie pustego słownika do zliczania krotności sekwencji (słownik liczników)
seq_counts = {}

# Zamiana listy zawierającej sekencje na krotkę (tuple)
# Lista nie może być kluczem, bo nie jest hashowalna
# W petli:
# Sprawdzenie, czy bieżąca krotka już jest w słowniku
# Jeżeli tak, to zwiększenie licznika o 1
# Jeśli nie, to dodanie krotki do słownika jako klucza z wartością 1
for seq in kseqs:
        seq_tuple = tuple(seq)
        if seq_tuple in seq_counts:
            seq_counts[seq_tuple] += 1
       else:
seq_counts[seq_tuple] = 1

# Wyświetlenie wartości słownika
print(seq_counts)
```

Funkcja zip

zip(iterable1, iterable2, ...)

Funkcja zip() w Pythonie grupuje elementy z wielu iterowalnych obiektów (jak listy, krotki) w pary, trójki, itp. Działa jak "zamek błyskawiczny", (stąd nazwa *zip*), łącząc pierwszy element z pierwszego obiektu z pierwszym elementem z drugiego obiektu itd., a następnie tworzy krotki z odpowiadających elementów z każdego obiektu.

Funkcja łączy elementy o tych samych pozycjach z obu obiektów.

```
names = ['Ala', 'Ola', 'Jan']
ages = [10, 12, 9]
pairs = zip(names, ages)
print(list(pairs))
```

Wynik:
[('Ala', 10), ('Ola', 12), ('Jan', 9)]

Metoda join

patterns = [' → '.join(p[0]) if len(p[0]) > 1 else str(p[0][0]) for p in wzorce]

for p in wzorce	iteruje po elementach kolekcji wzorce
p[0]	pierwszy element każdego p (np. lista znaków)
len(p[0]) > 1	sprawdza, czy lista ma więcej niż 1 element
' → '.join(p[0])	łączy elementy listy w jeden tekst, oddzielone strzałką
else str(p[0][0])	jeśli jest tylko jeden element, zamienia go na string

Złączenie (comprehension) tworzy listę napisów, w której każdy element to połączone elementy poszczególnych wierszy ze wzorca p[0] , oddzielone " → ". Jeśli p[0] ma tylko jeden element, zostaje on przekształcony do tekstu bez strzałek.

```
elementy = ['Ala', 'Ola', 'Tomek']
zdanie1 = ' → '.join(elementy)
```

Ala → Ola → Tomek

```
wyrazy = ["tomek", "lubi", "lody", "pistacjowe"]
zdanie2 = " ".join([el.capitalize() if el == wyrazy[0] else el for el in wyrazy])
```

Tomek lubi lody pistacjowe