Оптимизация выполнения кода, векторизация, Numba Материалы:

Макрушин С.В. Лекция 3: Оптимизация выполнения кода, векторизация, Numba IPython Cookbook, Second Edition (2018), глава 4
https://numba.pydata.org/numba-doc/latest/user/5minguide.html

In [14]:
```
import pandas as pd
import numpy as np
import string
from collections import Counter
from numba import jit
```

Разминка

1. Сгенерируйте массив A из N=1млн случайных целых чисел на отрезке от 0 до 1000. Пусть B[i] = A[i] + 100. Посчитайте среднее значение массива B.

In [18]:
```
N = 1000000
A = np.random.randint(0, 1000 + 1, size=N)
B = A + 100

%timeit np.average(B)
```

```
303 µs ± 2.31 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

2. Создайте таблицу 2млн строк и с 4 столбцами, заполненными случайными числами. Добавьте столбец key, которые содержит элементы из множества английских букв. Выберите из таблицы подмножество строк, для которых в столбце key указаны первые 5 английских букв.

In [20]:
```
M = 2000000
df = pd.DataFrame(np.random.rand(M, 4), columns=['col1', 'col2', 'col3', 'col4'])
df['key'] = np.random.choice(list(string.ascii_lowercase), size=M)

df.head()
```

Out[20]:

	col1	col2	col3	col4	key
0	0.441157	0.548922	0.607894	0.209414	d
1	0.100888	0.958953	0.964476	0.302508	u
2	0.259978	0.926910	0.176032	0.048809	h
3	0.972437	0.915568	0.758418	0.916820	k
4	0.206797	0.596889	0.110572	0.883218	u

In [23]:
```
%timeit len(df[df['key'].isin(list(string.ascii_lowercase[:5]))])

len(df[df['key'].isin(list(string.ascii_lowercase[:5]))])
```

```
44.7 ms ± 493 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Out[23]:
```
384738
```

Лабораторная работа №8

1. В файлах recipes_sample.csv и reviews_sample.csv находится информация об рецептах блюд и отзывах на эти рецепты соответственно. Загрузите данные из файлов в виде pd.DataFrame с названиями recipes и reviews. Обратите внимание на корректное считывание столбца(ов) с индексами. Приведите столбцы к нужным типам.

In [27]:
```
recipes = pd.read_csv("recipes_sample.csv")
reviews = pd.read_csv("reviews_sample.csv", index_col=0)
```

In [29]:
```
recipes.head()
```

Out[29]:

	name	id	minutes	contributor_id	submitted	n_steps	description	n_ingredients
0	george s at the cove black bean soup	44123	90	35193	2002-10-25	NaN	an original recipe created by chef scott meska...	18.0
1	healthy for them yogurt popsicles	67664	10	91970	2003-07-26	NaN	my children and their friends ask for my homem...	NaN
2	i can t believe it s spinach	38798	30	1533	2002-08-29	NaN	these were so go, it surprised even me.	8.0
3	italian gut busters	35173	45	22724	2002-07-27	NaN	my sister-in-law made these for us at a family...	NaN
4	love is in the air beef fondue sauces	84797	25	4470	2004-02-23	4.0	i think a fondue is a very romantic casual din...	NaN

In [31]:
```
reviews.head()
```

Out[31]:

	user_id	recipe_id	date	rating	review
370476	21752	57993	2003-05-01	5	Last week whole sides of frozen salmon fillet ...
624300	431813	142201	2007-09-16	5	So simple and so tasty! I used a yellow capsi...
187037	400708	252013	2008-01-10	4	Very nice breakfast HH, easy to make and yummy...
706134	2001852463	404716	2017-12-11	5	These are a favorite for the holidays and so e...
312179	95810	129396	2008-03-14	5	Excellent soup! The tomato flavor is just gre...

In [30]:
```
print(f"{reviews.dtypes}\n"), print(f"{recipes.dtypes}\n")
```

```
user_id       int64
recipe_id     int64
date         object
rating        int64
review       object
dtype: object

name              object
id                 int64
minutes            int64
contributor_id     int64
submitted         object
n_steps          float64
description       object
n_ingredients    float64
dtype: object
```

Out[30]: (None, None)

In [89]:
```
%timeit recipes["name"] = recipes['name'].astype("str")
%timeit recipes["description"] = recipes['description'].astype("str")
%timeit recipes["submitted"] = pd.to_datetime(recipes["submitted"])
```

```
256 µs ± 2.8 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
258 µs ± 5.24 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
2.92 ms ± 7.51 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

In [90]:
```
%timeit reviews["review"] = reviews['review'].astype("str")
%timeit reviews["date"] = pd.to_datetime(reviews["date"])
```

```
2.8 ms ± 34.8 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
2.96 ms ± 13.9 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

In [93]:
```
print(f"{reviews.dtypes}\n"), print(f"{recipes.dtypes}\n")
```

```
user_id              int64
recipe_id            int64
date         datetime64[ns]
rating               int64
review              object
dtype: object

name                      object
id                         int64
minutes                    int64
contributor_id             int64
submitted         datetime64[ns]
n_steps                  float64
description               object
n_ingredients            float64
dtype: object
```

Out[93]: (None, None)

2. Реализуйте несколько вариантов функции подсчета среднего значения столбца rating из таблицы reviews для отзывов, оставленных в 2010 году.

С использованием метода DataFrame.iterrows и без использования срезов таблицы С использованием метода DataFrame.iterrows и с использованием срезов таблицы С использованием метода DataFrame.mean Проверьте, что результаты работы всех написанных функций корректны и совпадают. Измерьте время выполнения всех вариантов.

In [39]:
```
def mean_rating_a(reviews):
    total = 0
    count = 0
    for idx, row in reviews.iterrows():
        if row['date'].year == 2010:
            total += row['rating']
            count += 1
    return total / count if count else 0
```

In [41]:
```
def mean_rating_b(reviews):
    reviews_2010 = reviews[reviews['date'].dt.year == 2010]
    total = 0
    count = 0
    for idx, row in reviews_2010.iterrows():
        total += row['rating']
        count += 1
    return total / count if count else 0
```

In [43]:
```
def mean_rating_c(reviews):
    return reviews[reviews['date'].dt.year == 2010]['rating'].mean()
```

In [45]:
```
mean_rating_a(reviews), mean_rating_b(reviews), mean_rating_c(reviews)

%timeit mean_rating_a(reviews)
%timeit mean_rating_b(reviews)
%timeit mean_rating_c(reviews)
```

```
1.38 s ± 7.47 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
133 ms ± 578 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
4.72 ms ± 104 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

3. Какая из созданных функций выполняется медленнее? Что наиболее сильно влияет на скорость выполнения? Для ответа использовать профайлер line_profiler. Сохраните результаты работы профайлера в отдельную текстовую ячейку и прокомментируйте результаты его работы.

(*). Сможете ли вы ускорить работу функции 2B?

In [48]:
```
#pip install line_profiler
```

In [50]:
```
%load_ext line_profiler
```

In [52]:
```
%lprun -f mean_rating_a mean_rating_a(reviews)
```

```
Timer unit: 1e-09 s

Total time: 4.25217 s
File: /var/folders/v1/vr6ths610tx3bk4fxvnmy4wh0000gn/T/ipykernel_17069/3056816033.py
Function: mean_rating_a at line 1

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     1                                           def mean_rating_a(reviews):
     2         1       3000.0   3000.0      0.0      total = 0
     3         1          0.0      0.0      0.0      count = 0
     4    126697 3782979000.0  29858.5     89.0      for idx, row in reviews.iterrows():
     5    126696  431792000.0   3408.1     10.2          if row['date'].year == 2010:
     6     12094   36278000.0   2999.7      0.9              total += row['rating']
     7     12094    1121000.0     92.7      0.0              count += 1
     8         1       1000.0   1000.0      0.0      return total / count if count else 0
```

Анализ: Основное время тратится на итерацию `.iterrows()` и проверку года `row['date'].year == 2010`

In [55]:
```
%lprun -f mean_rating_b mean_rating_b(reviews)
```

```
Timer unit: 1e-09 s

Total time: 0.426339 s
File: /var/folders/v1/vr6ths610tx3bk4fxvnmy4wh0000gn/T/ipykernel_17069/538476784.py
Function: mean_rating_b at line 1

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     1                                           def mean_rating_b(reviews):
     2         1   16839000.0    2e+07      3.9      reviews_2010 = reviews[reviews['date'].dt.year == 2010]
     3         1          0.0      0.0      0.0      total = 0
     4         1          0.0      0.0      0.0      count = 0
     5     12095  364498000.0  30136.3     85.5      for idx, row in reviews_2010.iterrows():
     6     12094   43867000.0   3627.2     10.3          total += row['rating']
     7     12094    1133000.0     93.7      0.3          count += 1
     8         1       2000.0   2000.0      0.0      return total / count if count else 0
```

Анализ: Фильтрация `reviews_2010 = reviews[reviews['date'].dt.year == 2010]` выполняется быстрее предыдущих, так как использует векторизированные операции. Основное время тратится на `.iterrows()` и `total += row['rating']`

In [58]:
```
%lprun -f mean_rating_c mean_rating_c(reviews)
```

```
Timer unit: 1e-09 s

Total time: 0.018152 s
File: /var/folders/v1/vr6ths610tx3bk4fxvnmy4wh0000gn/T/ipykernel_17069/3075400442.py
Function: mean_rating_c at line 1

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     1                                           def mean_rating_c(reviews):
     2         1   18152000.0    2e+07    100.0      return reviews[reviews['date'].dt.year == 2010]['rating'].m
ean()
```

Анализ: Весь функционал производится с помощью векторизированных операций, поэтому нет циклов и `iterrows`, которые замедляют время

Самой медленной функцией оказалась `mean_rating_a`, а самой быстрой `mean_rating_c`

Оптимизация функции B:

Заменяем `iterrows` на `.itertuples()` - работает быстрее, так как возвращает именованные кортежи

In [66]:
```
def mean_rating_b_optimized(reviews):
    reviews_2010 = reviews[reviews['date'].dt.year == 2010]
    total = 0
    count = 0
    for row in reviews_2010.itertuples():
        total += row.rating
        count += 1
    return total / count if count else 0
```

In [68]:
```
%timeit mean_rating_b(reviews)
%timeit mean_rating_b_optimized(reviews)
```

```
136 ms ± 1.14 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
14.7 ms ± 187 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

4. Вам предлагается воспользоваться функцией, которая собирает информацию, сколько отзывов содержат то или иное слово. Измерьте время выполнения этой функции. Сможете ли вы найти узкие места в коде, используя профайлер? Выпишите (словами), что в имеющемся коде реализовано неоптимально. Оптимизируйте функцию и добейтесь значительного прироста в скорости выполнения.

In [96]:
```
def count_words(reviews):
    word_counts = {}
    for idx, row in reviews.iterrows():
        for word in row['review'].split():
            if word not in word_counts:
                word_counts[word] = 0
            word_counts[word] += 1
    return word_counts
```

In [98]:
```
%lprun -f count_words count_words(reviews)
```

```
Timer unit: 1e-09 s

Total time: 6.98474 s
File: /var/folders/v1/vr6ths610tx3bk4fxvnmy4wh0000gn/T/ipykernel_17069/3301249280.py
Function: count_words at line 1

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     1                                           def count_words(reviews):
     2         1       2000.0   2000.0      0.0      word_counts = {}
     3    126697 3886378000.0  30674.6     55.6      for idx, row in reviews.iterrows():
     4   6716583 1288575000.0    191.8     18.4          for word in row['review'].split():
     5   6589887  849259000.0    128.9     12.2              if word not in word_counts:
     6    164272   43296000.0    263.6      0.6                  word_counts[word] = 0
     7   6589887  917224000.0    139.2     13.1              word_counts[word] += 1
     8         1       1000.0   1000.0      0.0      return word_counts
```

In [102…
```
def count_words_optimized(reviews):
    all_words = []
    for review in reviews['review']:
        all_words.extend(review.split())
    return Counter(all_words)
```

In [104…
```
%lprun -f count_words_optimized count_words_optimized(reviews)
```

```
Timer unit: 1e-09 s

Total time: 0.695688 s
File: /var/folders/v1/vr6ths610tx3bk4fxvnmy4wh0000gn/T/ipykernel_17069/2534330936.py
Function: count_words_optimized at line 1

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     1                                           def count_words_optimized(reviews):
     2         1       4000.0   4000.0      0.0      all_words = []
     3    126697   18644000.0    147.2      2.7      for review in reviews['review']:
     4    126696  252064000.0   1989.5     36.2          all_words.extend(review.split())
     5         1  424976000.0    4e+08     61.1      return Counter(all_words)
```

In [106…
```
%timeit count_words(reviews)
%timeit count_words_optimized(reviews)
```

```
2.18 s ± 15.6 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
656 ms ± 1.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

В исходной функции:

- `iterrows()` возвращает каждую строку как `pandas.Series`, что очень медленно
- Внутри цикла происходит многократный доступ к `row['review']`, что также замедляет выполнение
- Двойной цикл, который увеличивает время выполнения функции

В оптимизированной функции:

- `' '.join(...).split()` вместо вложенных циклов
- `collections.Counter` работает быстрее чем `dict()` в `python`

5. Напишите несколько версий функции MAPE (см. MAPE) для расчета среднего процентного отклонения значения рейтинга для отзыва от среднего значения рейтинга для этого отзыва.

Без использования массивов numpy и numba Без использования массивов numpy, но с использованием numba С использованием массивов numpy, но без использования numba С использованием массивов numpy и numba Измерьте время выполнения каждой из реализаций.

Замечание: удалите из выборки отзывы с нулевым рейтингом.

$$MAPE = \frac{1}{n}\sum_{t=1}^{n} | \frac{A_t - F_t}{A_t} | * 100$$

A_t - actual value

F_t - forecast value

In [33]:
```
filter_reviews = reviews[reviews['rating'] != 0]
mean_rtg = filter_reviews.rating.mean()
ratings = filter_reviews['rating'].values
```

In [35]:
```
def mape_1(reviews, mean_rating):
    errors = []
    for idx, row in reviews.iterrows():
```

```
        errors.append(abs(row['rating'] - mean_rating) / row['rating'])
    return sum(errors) / len(errors) * 100
```

In [37]:
```
@jit(nopython=True)
def mape_2(ratings, mean_rating):
    errors = np.empty(len(ratings))
    for i in range(len(ratings)):
        errors[i] = abs(ratings[i] - mean_rating) / ratings[i]
    return np.mean(errors) * 100
```

In [39]:
```
def mape_3(ratings, mean_rating):
    return np.mean(np.abs(ratings - mean_rating) / ratings) * 100
```

In [41]:
```
@jit(nopython=True)
def mape_4(ratings, mean_rating):
    return np.mean(np.abs(ratings - mean_rating) / ratings) * 100
```

In [57]:
```
%timeit mape_1(filter_reviews, mean_rtg)
%timeit mape_2(ratings, mean_rtg)
%timeit mape_3(ratings, mean_rtg)
%timeit mape_4(ratings, mean_rtg)
```

```
1.4 s ± 9.85 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
165 µs ± 1.25 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
134 µs ± 3.04 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
131 µs ± 163 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

In [59]:
```
mape_1(filter_reviews, mean_rtg), mape_2(ratings, mean_rtg), mape_3(ratings, mean_rtg), mape_4(ratings, mean_rt
```

Out[59]:
```
(16.26666333876162, 16.26666333876162, 16.266663338799717, 16.26666333876162)
```