**DataFrame中查找4W个目标的性能分析**

这次我们来研究一个有意思的话题，在DataFrame进行数据查找的性能分析。

**一、需求**

在日常的数据分析项目需求中，很多时候我们需要在很多的用户中寻找一批用户，现在假设我们有4千万的用户，目标客户是4万个，她们都在两个Dataframe中进行存储，我们用代码生成：

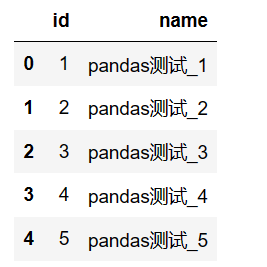
import pandas as pd

data = {'id': range(1, 40000001),

'name': ['pandas测试\_' + str(each) for each in range(1, 40000001)]}

df = pd.DataFrame(data)

df.head()



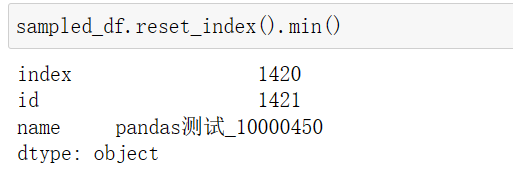
我们这里使用抽样方法获取4w的查询样本：

sampled\_df = df.sample(n=40000)

sampled\_df.head()



我们看一下此次抽样数据的情况（因为随机抽样），大家抽样的结果与我的不同哦：



可以看出，id的最小值为1421。

现在我们的需求就是，怎么从全量的df中查询抽样的sampled\_df中的4W个样本的时间。为此我们先写一个计算时间的函数：

import time

def timer\_decorator(func):

def wrapper(\*args, \*\*kwargs):

start\_time = time.time()

result = func(\*args, \*\*kwargs)

end\_time = time.time()

execution\_time = end\_time - start\_time

print(f"函数 {func.\_\_name\_\_} 的运行时间为 {execution\_time} 秒")

return result

return wrapper

接下来我们就使用几种方法来实现这个需求并统计时间。

1. **query()方法**我们首先使用query()来实现一下这个搜索，query()方法的相关用法这里大家可以从官网的文档进行学习：

@timer\_decorator

def query\_use(df,sampled\_df):

sampled\_ids = sampled\_df['id'].tolist()

query\_string = 'id in @sampled\_ids'

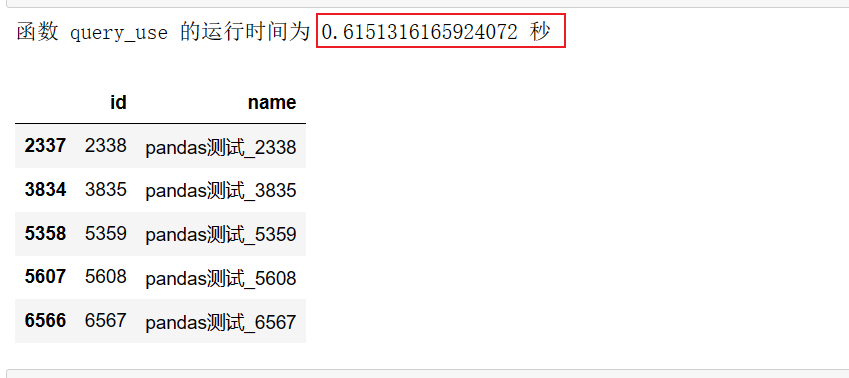
matched\_df = df.query(query\_string)

return matched\_df

query\_use = query\_use(df,sampled\_df)

query\_use.head()

输出结果如下：



函数的运行时间大约是0.6S。单独使用isin方法如何，代码如下：

@timer\_decorator

def isin\_use(df,sampled\_df):

matched\_df = df[df['id'].isin(sampled\_df['id'])]

return matched\_df

isin\_use = isin\_use(df,sampled\_df)

isin\_use.head()



0.48的时间还是不错的。

1. **where方法**

where方法也差不多，代码如下：

@timer\_decorator

def where\_use(df,sampled\_df):

sampled\_ids = sampled\_df['id'].tolist()

matched\_df = df['id'].where(df['id'].isin(sampled\_ids))

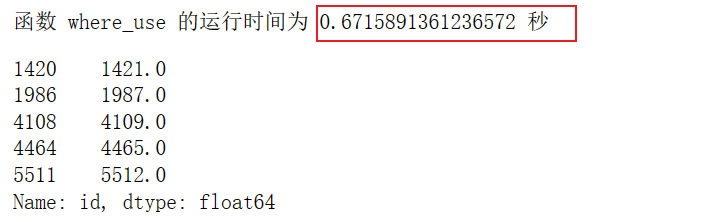
matched\_df = matched\_df.dropna()

return matched\_df

where\_use = where\_use(df,sampled\_df)

where\_use.head()

时间结果如下：



1. **loc方法**

loc[]在一定程度上是dataframe的一个属性，其可以根据行列的索引来选择数据。我们来看一下：

@timer\_decorator

def loc\_use(df,sampled\_df):

sampled\_ids = sampled\_df['id'].tolist()

matched\_df = df.loc[df['id'].isin(sampled\_ids)]

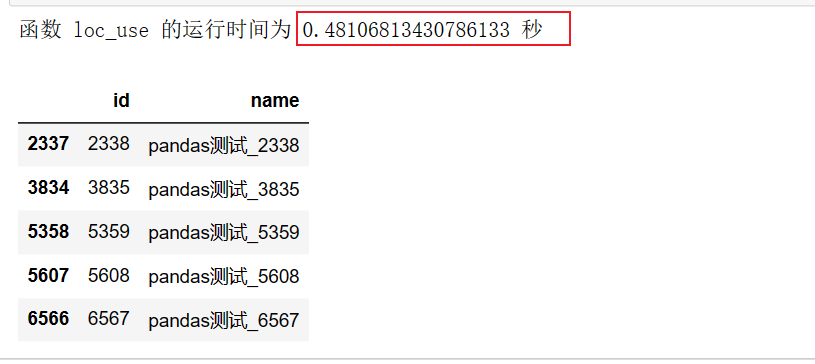
# 显示匹配的数据

return matched\_df

loc\_use = loc\_use(df,sampled\_df)

loc\_use.head()

输出结果如下：



时间也在0.5一下，还是很不错的。

1. **重置索引方法**

我们先重置df的索引，然后使用loc来进行查找：

@timer\_decorator

def reset\_index\_use(df,sampled\_df):

sampled\_ids = sampled\_df['id'].tolist()

# 重置df的索引为'id'列

df.set\_index('id', inplace=True)

# 使用loc来查询df

matched\_df = df.loc[sampled\_ids]

matched\_df.reset\_index(inplace=True)

return matched\_df

reset\_index\_use = reset\_index\_use(df,sampled\_df)

reset\_index\_use.head()

输出如下：



时间并不是很少，可能是该方法在此场景下不是很实用。

可以看出单独使用isin和loc还是比较快的，下次可以优先考虑。上述实验在不同的计算机上实现具有差异，大家可以自己试试。

本次的文章就是这些，下次再见。

最后使用join实验一下看看？