Polars and pandas Comparison

in case of excel data analysis

Introduction

Polars and pandas are simply equivalent. Most of the operations and their syntax is similar. Similarity extends to their logo. These similarities come from the newer one of them being Polars, released in 2021, offers users a similar usage experience to pandas, released in 2008.

What is Polars?

Polars is completely written in Rust and uses Arrow -the native arrow2 Rust implementation- as its foundation. Philosophy and goal of the Polars is to provide a lightning fast DataFrame library that:

- Utilizes all available cores on your machine.
- Optimizes queries to reduce unneeded work/memory allocations.
- Handles datasets much larger than your available RAM.
- Has an API that is consistent and predictable.
- Has a strict schema (data-types should be known before running the query).

Polars is written in Rust which gives it C/C++ performance and allows it to fully control performance critical parts in a query engine. As such Polars goes to great lengths to:

- Reduce redundant copies.
- Traverse memory cache efficiently.
- Minimize contention in parallelism.
- Process data in chunks.
- Reuse memory allocations.

What is pandas?

pandas is a well-known and commonly used DataFrame library for Python. Has abilities such as:

- A fast and efficient DataFrame object for data manipulation with integrated indexing.
- Tools for reading and writing data between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 format.
- Intelligent data alignment and integrated handling of missing data: gain automatic label-based alignment in computations and easily manipulate messy data into an orderly form.
- Flexible reshaping and pivoting of data sets.
- Intelligent label-based slicing, fancy indexing, and subsetting of large data sets.
- Columns can be inserted and deleted from data structures for size mutability.
- Aggregating or transforming data with a powerful group by engine allowing split-apply-combine operations on data sets.
- High performance merging and joining of data sets.
- Hierarchical axis indexing provides an intuitive way of working with high-dimensional data in a lower-dimensional data structure.
- Time series-functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging. Even create domain-specific time offsets and join time series without losing data.
- Highly optimized for performance, with critical code paths written in Cython or C.
- Python with pandas is in use in a wide variety of academic and commercial domains, including Finance, Neuroscience, Economics, Statistics, Advertising, Web Analytics, and more.

pandas aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. pandas aims for A world where data analytics and manipulation software is:

- Accessible to everyone
- Free for users to use and modify
- Flexible
- Powerful
- Easy to use
- Fast

Method of Comparison

We will implement a simple algorithm of searching through two excel files for data resemblance, and after checking the date; printing whether which file is derived from another.

Using pandas:

```
import pandas as pd

import time

import resource
```

```
time_start = time.perf_counter()


def isSame(ind1, ind2): #checks if they are same by comparing a definitely unique data (TC)
    tc1=file1["TC"][ind1]
    tc2=file2["TC"][ind2]
    if tc1==tc2:
        return True
    else:
        return False


def dateChecker(ind1, ind2, col): #compares two given data's dates
    date1=file1["DATE"][ind1]
    date2=file2["DATE"][ind2]
    for i in range(0, 19):
        while(str(date1)[i]!=str(date2)[i]):
            if(int(str(date1)[i])>int(str(date2)[i])):
                print("File1 is derived from File2. (According to " + file1["FIRST NAME"][ind1] + " " + file1["LAST NAME"][ind1] + "'s "+ col +")")
                return
            else:
                print("File2 is derived from File1. (According to " + file1["FIRST NAME"][ind1] + " " + file1["LAST NAME"][ind1] + "'s "+ col +")")
                return
    print("Dates are same.")
    return


def columnTraverser(col1, col2, col): #goes through the same column looking for same data
    x=0
    while x<len(col1):

        y=0

        while y<len(col2):

            if col1[x] == col2[y] and isSame(x, y):

                dateChecker(x, y, col)

            y+=1

        x+=1

    return


def traverser(data1, data2): #goes through the columns looking for a same

    for i in data1.columns:

        for j in data2.columns:

            if i=="DATE" or j=="DATE":

                continue

            if i==j:

                columnTraverser(data1[i],data2[i],i)

    return


#main


print("enter a path or copy the file into the same directory and enter the name")

file1=pd.read_excel(input("File1: "))

file2=pd.read_excel(input("File2: "))


traverser(file1, file2)


time_elapsed = (time.perf_counter() - time_start)

memKb=resource.getrusage(resource.RUSAGE_SELF).ru_maxrss/1024.0

print ("%5.1f secs %5.1f KByte" % (time_elapsed,memKb))
```

Using Polars:

```
import polars as pl
import time
import resource

time_start = time.perf_counter()

def isSame(ind1, ind2): #checks if they are same by comparing a definitely unique data (TC)
    tc1=file1.get_column("TC")[ind1]
    tc2=file2.get_column("TC")[ind2]
    if tc1==tc2:
        return True
    else:
        return False

def dateChecker(ind1, ind2, col): #compares two given data's dates
    date1=file1.get_column("DATE")[ind1]
    date2=file2.get_column("DATE")[ind2]
    for i in range(0, 10):
        while(str(date1)[i]!=str(date2)[i]):
            if(int(date1[i])>int(date2[i])):
                print("File1 is derived from File2. (According to " + file1.get_column("FIRST NAME")[ind1] + " " + file1.get_column("LAST NAME")[ind1] + "'s "+ col +")")
                return
            else:
                print("File2 is derived from File1. (According to " + file1.get_column("FIRST NAME")[ind1] + " " + file1.get_column("LAST NAME")[ind1] + "'s "+ col +")")
                return
    print("Dates are same.")
    return


def columnTraverser(col1, col2, col): #goes through the same column looking for same data
    x=0
    while x<len(col1):
        y=0
        while y<len(col2):
            if col1[x] == col2[y] and isSame(x, y):
                dateChecker(x, y, col)
            y+=1
        x+=1
    return


def traverser(data1, data2): #goes through the columns looking for a same
    for i in data1.columns:
        for j in data2.columns:
            if i=="DATE" or j=="DATE":
                continue
            if i==j:
                columnTraverser(data1.get_column(i),data2.get_column(i),i)
    return


#main


print("enter a path or copy the file into the same directory and enter the name")
file1=pl.read_excel(input("File1: "))
file2=pl.read_excel(input("File2: "))
```

```
traverser(file1, file2)


time_elapsed = (time.perf_counter() - time_start)

memKb=resource.getrusage(resource.RUSAGE_SELF).ru_maxrss/1024.0

print ("%5.1f secs %5.1f KByte" % (time_elapsed,memKb))
```

After executing we get:

Size	**Time (s)**			**Memory (KByte)**		
	pandas	Polars	Improvement (%)	pandas	Polars	Improvement (%)
10	0.1	0.1	0.0	123.1	94.3	30.54
200	1.1	0.5	54.55	124.7	95.1	31.13
1000	17.4	7.5	56.90	125.2	95.5	31.10
9000	1252.8	513.3	59.03	131.2	99.4	31.99

Conclusion

Polars succeeds at its claims as it is faster and uses less memory (than pandas).

For test files

Big File credit: https://www.kaggle.com/datasets/ashishraut64/internet-users

References and For Further Reading:

https://pandas.pydata.org/about/

https://sparkbyexamples.com/pandas/pandas-get-cell-value-from-dataframe/

https://github.com/pola-rs/polars

https://pola-rs.github.io/polars-book/user-guide/