CDS Assignment 2:
Alternatives to Pandas and Numpy

UID:2309039
Name: Jyothi Mudalu
Rollno.: 030

Aim:

Find out which libraries can be used as an alternative to Pandas and Numpy. Also benchmark the performance of those libraries.

Introduction:

In this report, we look for alternative to Pandas and Numpy and we present comparison the performance of those libraries.

Research for alternative libraries to Pandas:

To find alternative libraries to Pandas, I searched online forums, and the Python Package Index (PyPI) for newer libraries. Some alternatives to consider include:

1) Dask: Dask provides parallel and distributed computing for tasks like data manipulation and analytics. It's designed to scale from a single machine to a cluster.
2) Vaex: Vaex is a DataFrame library that focuses on high-performance and out-of-core processing. It's suitable for working with large datasets.
3) Modin: Modin is designed to speed up Pandas operations by utilizing multiple CPU cores.
4) Polars: Polars is a DataFrame library that aims for high performance and compatibility with Rust's Arrow project.

Installation of the alternative libraries for Pandas:

Once the alternative libraries are identified we install them libraries using pip.

1) Installing Dask:

In [1]:
```
#Installing dask
!pip install dask
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: dask in c:\programdata\anaconda3\lib\site-packages (2023.6.0)
Requirement already satisfied: click>=8.0 in c:\programdata\anaconda3\lib\site-packages (from dask) (8.0.4)
Requirement already satisfied: cloudpickle>=1.5.0 in c:\programdata\anaconda3\lib\site-packages (from dask) (2.2.1)
Requirement already satisfied: fsspec>=2021.09.0 in c:\programdata\anaconda3\lib\site-packages (from dask) (2023.3.0)
Requirement already satisfied: packaging>=20.0 in c:\programdata\anaconda3\lib\site-packages (from dask) (23.0)
Requirement already satisfied: partd>=1.2.0 in c:\programdata\anaconda3\lib\site-packages (from dask) (1.2.0)
Requirement already satisfied: pyyaml>=5.3.1 in c:\programdata\anaconda3\lib\site-packages (from dask) (6.0)
Requirement already satisfied: toolz>=0.10.0 in c:\programdata\anaconda3\lib\site-packages (from dask) (0.12.0)
Requirement already satisfied: importlib-metadata>=4.13.0 in c:\programdata\anaconda3\lib\site-packages (from dask) (6.0.0)
Requirement already satisfied: colorama in c:\programdata\anaconda3\lib\site-packages (from click>=8.0->dask) (0.4.6)
Requirement already satisfied: zipp>=0.5 in c:\programdata\anaconda3\lib\site-packages (from importlib-metadata>=4.13.0->dask)
(3.11.0)
Requirement already satisfied: locket in c:\programdata\anaconda3\lib\site-packages (from partd>=1.2.0->dask) (1.0.0)
```

2) Installing Modin:

In [3]:
```
# Installing Modin
!pip install modin[ray]
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting modin[ray]
  Downloading modin-0.23.1-py3-none-any.whl (1.1 MB)
                                            0.0/1.1 MB ? eta -:--:--
     -----------                            0.3/1.1 MB 9.6 MB/s eta 0:00:01
     -----------------------                0.6/1.1 MB 6.5 MB/s eta 0:00:01
     ----------------------------------     0.9/1.1 MB 8.3 MB/s eta 0:00:01
     ---------------------------------------- 1.1/1.1 MB 6.7 MB/s eta 0:00:00
Collecting pandas<2.1,>=2 (from modin[ray])
  Downloading pandas-2.0.3-cp311-cp311-win_amd64.whl (10.6 MB)
                                            0.0/10.6 MB ? eta -:--:--
     -                                      0.3/10.6 MB 10.6 MB/s eta 0:00:01
     --                                     0.7/10.6 MB 7.9 MB/s eta 0:00:02
     ---                                    1.0/10.6 MB 7.6 MB/s eta 0:00:02
     ----                                   1.2/10.6 MB 6.8 MB/s eta 0:00:02
     ----                                   1.2/10.6 MB 6.5 MB/s eta 0:00:02
     -----                                  1.4/10.6 MB 5.4 MB/s eta 0:00:02
     -----                                  1.5/10.6 MB 5.1 MB/s eta 0:00:02
     ------                                 1.8/10.6 MB 5.1 MB/s eta 0:00:02
     -------                                2.1/10.6 MB 5.3 MB/s eta 0:00:02
```

3) Installing Polars:

In [5]:
```
# Installing Polars
!pip install polars
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting polars
  Downloading polars-0.19.3-cp38-abi3-win_amd64.whl (20.5 MB)
                                            0.0/20.5 MB ? eta -:--:--
                                            0.2/20.5 MB 6.9 MB/s eta 0:00:03
                                            0.5/20.5 MB 6.4 MB/s eta 0:00:04
     -                                      0.8/20.5 MB 7.4 MB/s eta 0:00:03
     --                                     1.2/20.5 MB 7.4 MB/s eta 0:00:03
     --                                     1.5/20.5 MB 7.3 MB/s eta 0:00:03
     ---                                    1.7/20.5 MB 7.1 MB/s eta 0:00:03
     ---                                    2.0/20.5 MB 7.0 MB/s eta 0:00:03
     ----                                   2.2/20.5 MB 6.8 MB/s eta 0:00:03
     -----                                  2.6/20.5 MB 6.7 MB/s eta 0:00:03
     -----                                  3.0/20.5 MB 7.0 MB/s eta 0:00:03
     ------                                 3.3/20.5 MB 6.9 MB/s eta 0:00:03
     ------                                 3.5/20.5 MB 6.8 MB/s eta 0:00:03
     -------                                3.7/20.5 MB 6.8 MB/s eta 0:00:03
     -------                                3.8/20.5 MB 6.6 MB/s eta 0:00:03
     -------                                3.9/20.5 MB 6.1 MB/s eta 0:00:03
```

Benchmarking using the alternative libraries to Pandas:

Benchmarking different libraries for performance typically involves creating synthetic data or generating test datasets in-memory. Below, I'll provide a simple example of how to benchmark Pandas, Dask, Modin,and Polars using synthetic data and calculate the time it takes to perform a basic operation (e.g., summing a column) on that data:

Code:

```
import pandas as pd
import dask.dataframe as dd
import modin.pandas as mpd
import polars as pl
import numpy as np
import time

# Generate synthetic data
data_size = 10**6  # Adjust the data size as needed
data = {
    'col1': np.random.randint(1, 100, data_size),
    'col2': np.random.rand(data_size),
}

# Pandas Benchmark
start_time = time.time()
df_pandas = pd.DataFrame(data)
result_pandas = df_pandas['col1'].sum()
pandas_time = time.time() - start_time

# Dask Benchmark
start_time = time.time()
ddf = dd.from_pandas(df_pandas, npartitions=4)  # Adjust the number
of partitions as needed
result_dask = ddf['col1'].sum().compute()
dask_time = time.time() - start_time

# Modin Benchmark
start_time = time.time()
df_modin = mpd.DataFrame(data)
result_modin = df_modin['col1'].sum()
modin_time = time.time() - start_time
```

```
# Polars Benchmark
start_time = time.time()
df_polars = pl.DataFrame(data)
result_polars = df_polars['col1'].sum()
polars_time = time.time() - start_time

print(f"Pandas Time: {pandas_time} seconds, Result: {result_pandas}")
print(f"Dask Time: {dask_time} seconds, Result: {result_dask}")
print(f"Modin Time: {modin_time} seconds, Result: {result_modin}")
print(f"Polars Time: {polars_time} seconds, Result: {result_polars}")
```

Output:

```
Pandas Time: 0.0798637866973877 seconds, Result: 49987440
Dask Time: 0.17425799369812012 seconds, Result: 49987440
Modin Time: 0.14992213249206543 seconds, Result: 49987440
Polars Time: 0.02554011344909668 seconds, Result: 49987440
```

Conclusion:

In our performance benchmarking exercise, we compared the execution times of various data manipulation libraries, including Pandas, Dask, Modin, and Polars, on a synthetic dataset. The benchmark aimed to measure the efficiency of each library when performing a simple operation, such as calculating the sum of a column.
Among the libraries tested, Polars emerged as the standout performer, consistently demonstrating significantly shorter execution times.

Research for alternative libraries to NumPy:

1) **CuPy: If you have access to NVIDIA GPUs, CuPy provides GPU-accelerated array operations similar to NumPy.**
2) **PyTorch: PyTorch is a deep learning framework that includes a tensor library with a similar interface to NumPy.It's popular in the deep learning community and is known for its dynamic computation graph.**
3) **Numba: Numba is a just-in-time (JIT) compiler for Python that can accelerate numerical code, including NumPy functions.It compiles Python functions to machine code for performance improvements.**
4) **Blaze: Blaze is a high-level data exploration and manipulation library that supports a wide range of data sources and formats.It provides a unified interface for querying and transforming data from different backends.**

Installation of the alternative libraries for NumPy:

1) Installing Cupy:

```
In [4]: # Installing CuPy (if you have an NVIDIA GPU)
        !pip install cupy
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting cupy
  Downloading cupy-12.2.0.tar.gz (2.0 MB)
                                               0.0/2.0 MB ? eta -:--:--
                                               0.0/2.0 MB ? eta -:--:--
                                               0.0/2.0 MB ? eta -:--:--
                                               0.0/2.0 MB 388.9 kB/s eta 0:00:05
                                               0.0/2.0 MB 388.9 kB/s eta 0:00:05
     --                                        0.1/2.0 MB 726.2 kB/s eta 0:00:03
     --                                        0.1/2.0 MB 514.3 kB/s eta 0:00:04
     ---                                       0.2/2.0 MB 653.6 kB/s eta 0:00:03
     -----                                     0.2/2.0 MB 835.2 kB/s eta 0:00:03
     -----                                     0.2/2.0 MB 835.2 kB/s eta 0:00:03
     -----                                     0.3/2.0 MB 682.7 kB/s eta 0:00:03
     -------                                   0.4/2.0 MB 825.0 kB/s eta 0:00:02
     -------                                   0.4/2.0 MB 865.0 kB/s eta 0:00:02
     ---------                                 0.5/2.0 MB 907.9 kB/s eta 0:00:02
     ---------                                 0.5/2.0 MB 907.9 kB/s eta 0:00:02
     ----------                                0.5/2.0 MB 854.8 kB/s eta 0:00:02
```

2) Installing PyTorch:

```
In [2]: pip install torch
```

```
Collecting torchNote: you may need to restart the kernel to use updated packages.

  Downloading torch-2.0.1-cp311-cp311-win_amd64.whl (172.3 MB)
     ---------------------------------------- 0.0/172.3 MB ? eta -:--:--
     ---------------------------------------- 0.0/172.3 MB ? eta -:--:--
     -------------------------------------- 0.0/172.3 MB 435.7 kB/s eta 0:06:36
     ---------------------------------------- 0.2/172.3 MB 1.5 MB/s eta 0:01:55
     ---------------------------------------- 0.5/172.3 MB 3.3 MB/s eta 0:00:52
     ---------------------------------------- 0.9/172.3 MB 4.3 MB/s eta 0:00:40
     ---------------------------------------- 1.2/172.3 MB 5.3 MB/s eta 0:00:33
     ---------------------------------------- 1.6/172.3 MB 5.5 MB/s eta 0:00:32
     ---------------------------------------- 2.0/172.3 MB 5.7 MB/s eta 0:00:30
      --------------------------------------- 2.3/172.3 MB 5.9 MB/s eta 0:00:29
      --------------------------------------- 2.5/172.3 MB 6.2 MB/s eta 0:00:28
      --------------------------------------- 2.7/172.3 MB 5.6 MB/s eta 0:00:31
      --------------------------------------- 3.0/172.3 MB 5.9 MB/s eta 0:00:29
      --------------------------------------- 3.4/172.3 MB 5.9 MB/s eta 0:00:29
      --------------------------------------- 3.6/172.3 MB 5.7 MB/s eta 0:00:30
      --------------------------------------- 3.8/172.3 MB 5.7 MB/s eta 0:00:30
```

3) Installing Numba:

```
In [3]: pip install numba
```

```
Note: you may need to restart the kernel to use updated packages.Collecting numba
  Obtaining dependency information for numba from https://files.pythonhosted.org/packages/e8/1c/5d65ac922a4f9a6f90a10207b818e22
e4d48a782af6574a6e7a50fae074d/numba-0.58.0-cp311-cp311-win_amd64.whl.metadata
  Using cached numba-0.58.0-cp311-cp311-win_amd64.whl.metadata (2.8 kB)
Collecting llvmlite<0.42,>=0.41.0dev0 (from numba)
  Obtaining dependency information for llvmlite<0.42,>=0.41.0dev0 from https://files.pythonhosted.org/packages/14/3b/f9665a4648
6f70a7cbb6237308e49e18ed42e4763f4e92e92cd37ea67ead/llvmlite-0.41.0-cp311-cp311-win_amd64.whl.metadata
  Using cached llvmlite-0.41.0-cp311-cp311-win_amd64.whl.metadata (5.0 kB)
Collecting numpy<1.26,>=1.21 (from numba)
  Obtaining dependency information for numpy<1.26,>=1.21 from https://files.pythonhosted.org/packages/72/b2/02770e60c4e2f7e158d
923ab0dea4e9f146a2dbf267fec6d8dc61d475689/numpy-1.25.2-cp311-cp311-win_amd64.whl.metadata
  Using cached numpy-1.25.2-cp311-cp311-win_amd64.whl.metadata (5.7 kB)
Using cached numba-0.58.0-cp311-cp311-win_amd64.whl (2.6 MB)
Using cached llvmlite-0.41.0-cp311-cp311-win_amd64.whl (28.1 MB)
Using cached numpy-1.25.2-cp311-cp311-win_amd64.whl (15.5 MB)
Installing collected packages: numpy, llvmlite, numba
  Attempting uninstall: numpy
    Found existing installation: numpy 1.26.0
    Uninstalling numpy-1.26.0:
      Successfully uninstalled numpy-1.26.0
```

Benchmarking using the alternative libraries NumPy:

Below is a simple example of how you can benchmark the time it takes to perform a matrix multiplication operation using NumPy, Numba, and PyTorch. This code uses the timeit module to measure execution time. This code generates random matrices and measures the time it takes to perform matrix multiplication using NumPy, Numba, and PyTorch.

Code:

```
import numpy as np
import numba
import torch
import timeit

# Define matrix size
matrix_size = 1000

# NumPy benchmark
def numpy_benchmark():
    a = np.random.rand(matrix_size, matrix_size)
    b = np.random.rand(matrix_size, matrix_size)
    result = np.dot(a, b)

# Numba benchmark
@numba.jit
def numba_benchmark():
    a = np.random.rand(matrix_size, matrix_size)
    b = np.random.rand(matrix_size, matrix_size)
```

```
    result = np.dot(a, b)

# PyTorch benchmark
def pytorch_benchmark():
    a = torch.rand(matrix_size, matrix_size)
    b = torch.rand(matrix_size, matrix_size)
    result = torch.mm(a, b)

# Measure execution time
num_runs = 100  # Number of runs for each benchmark

numpy_time = timeit.timeit(numpy_benchmark, number=num_runs)
numba_time = timeit.timeit(numba_benchmark, number=num_runs)
pytorch_time = timeit.timeit(pytorch_benchmark, number=num_runs)

# Print results
print(f"NumPy Time: {numpy_time:.4f} seconds")
print(f"Numba Time: {numba_time:.4f} seconds")
print(f"PyTorch Time: {pytorch_time:.4f} seconds")
```

Output:

```
NumPy Time: 6.1094 seconds
Numba Time: 7.4629 seconds
PyTorch Time: 3.4996 seconds
```

Conclusion:

In our performance benchmarking exercise, we compared the execution times of various data manipulation libraries, includingNumPy,Numba and PyTorch, on a synthetic dataset. The benchmark aimed to measure the efficiency of each library when performing a simple operation, such as matrix multiplication. Among the libraries tested, PyTorch emerged as the standout performer, consistently demonstrating significantly shorter execution times.