1 You have a Python script that processes millions of records in a single thread. How would you optimize it to leverage multiple cores and reduce the execution time? Provide a sample code snippet

Necessitated by the fact that multiple cores of the CPU is required, I would naturally implemented a solution that utilized multiprocessing. Here, several processes, each running their own threads, are run across multiple CPU cores which would allow for a reduction in execution time. Below is an example of how this can be achieved.

Listing 1: Multiprocessing

```python

import pandas as pd
import numpy as np
import datetime
import time
import multiprocessing as mp

x = datetime.datetime(1993, 12, 9, 0, 0)

simulated_dates = []
# Create 2 million data points of dates starting from 1993 with 1 hour increments
for i in range(2000000):
    x += datetime.timedelta(minutes=60)
    simulated_dates.append(x)

df = pd.DataFrame(simulated_dates, columns=['Dates'])

print(df.shape)

# Function to add an hour to a given chunk of data
def add_hour(dates):
    return dates + pd.Timedelta(hours=1)

# Without multiprocessing
start_time = time.time()
df['Dates'] = df['Dates'].apply(add_hour)
end_time = time.time()

print("Without multiprocessing:", end_time - start_time, "seconds")


```

```python
# With multiprocessing
def process_chunk(chunk):
    return add_hour(chunk)

start_time = time.time()
num_cores = mp.cpu_count()
chunk_size = len(df) // num_cores
chunks = [df['Dates'][i:i+chunk_size] for i in range(0, len(df), chunk_size)]

with mp.Pool(processes=num_cores) as pool:
    df['Dates'] = pd.concat(pool.map(process_chunk, chunks))
end_time = time.time()

print("With multiprocessing:", end_time - start_time, "seconds")
```

The script can be found on my GitHub here in notebook format. In the notebook you will see that the results are as such:

Without multiprocessing: 16.4 seconds
With multiprocessing: 0.24 seconds

2 During a data pipeline run in Azure Data Factory, a step failed due to an invalid data format. Describe how you would debug this issue and prevent it from happening in the future

I have not worked in ADF before but from what I can gather from ChatGPT a reasonable sounding course of action would be first identify the part of the pipeline that failed. This can be do by inspecting the ADF pipeline run logs where one can see which error message has been generated. Depending on how well the ADF pipeline has been configured in regards to error handling, this error message may pinpoint exactly what the invalid data format means (null or some specific data type that was not expected.

Then I would check the source data to ensure it matches expected schemas and formats. This would include reviewing sample data in the source system, checking for recent changes in the source data structure and verifying any data extraction or ingestion processes.

If the source of the error has not yet been discovered I would then review data transformations in order to ascertain whether the data is valid at the source, examine any transformations applied in the pipeline that might be causing the format issue.

Lastly, I would check destination requirements in order to ensure the data format aligns with the requirements of the destination system or storage.

Of course this is all hypothetical musings of how I would solve a generic ADF invalid data format error. In practice I would have more hands-on knowledge of how the pipeline is structured, know which parts are more likely to break down and expected data format of source and sink systems and so on. This would lend to more realistic troubleshooting process.

3 Write a Python script using the Azure SDK that uploads a file to an Azure Blob Storage container. Ensure the script checks if the container exists and creates it if it does not

The script can be found on my GitHub here and is listed below as well:

Listing 2: Azure Upload

```

from azure.storage.blob import BlobServiceClient
from azure.core.exceptions import ResourceExistsError
from azure.core.exceptions import ResourceExistsError
from dotenv import load_dotenv
import os

load_dotenv('.env')

# Azure Storage account details
connection_string = os.getenv("CONNECTION_STRING")
container_name = "container"
local_file_path = "/home/saim/dtu_test_case/test.txt"
blob_name = "test_blob.txt"


blob_service_client = BlobServiceClient.from_connection_string(connection_string)
container_client = blob_service_client.get_container_client(container_name)

try:
    container_client.create_container()
    print(f"Container '{container_name}' created successfully.")
except ResourceExistsError:
    print(f"Container '{container_name}' already exists.")

blob_client = blob_service_client.get_blob_client(container=container_name, blob=
    blob_name)

```

```
with open(local_file_path, "rb") as data:
    blob_client.upload_blob(data, overwrite=True)

print(f"File '{local_file_path}' uploaded to Azure Blob Storage as '{blob_name}' in
   container '{container_name}'.")
```

By inserting your own connection string in the connection string variable and giving a different value to the local file path you should be able to run the script (given you have the necessary dependencies installed as well).

4 Write a Python script to download logs from Azure (e.g. events from a specific resource)

I have created a simple Flask app and deployed as an Azure Web App via a Docker container. It can be found here (spoiler alert: it is not working due to a file path error I am too lazy to fix, you can see it in the logs!). Then I download the logs by using the Azure CLI in Python like so:

Listing 3: Azure Upload

```
import zipfile
import subprocess

# Azure subscription and resource details
resource_group_name = "Zoomcamp"
app_name = "my-app"

# Download the logs using Azure CLI
command = f"az webapp log download --resource-group {resource_group_name} --name {
   app_name} --log-file azure_webapp_logs.zip"

subprocess.run(command, shell=True, check=True)

with zipfile.ZipFile('azure_webapp_logs.zip', 'r') as zip_ref:
    zip_ref.extractall('logs')

print("Logs have been downloaded and extracted.")
```

The logs can be viewed on my GitHub in the 'logs' directory here.