Name: Naga Pavithra Jajala

Batch Code: LISUM44: 30 March (2025) – 30 June (2025)

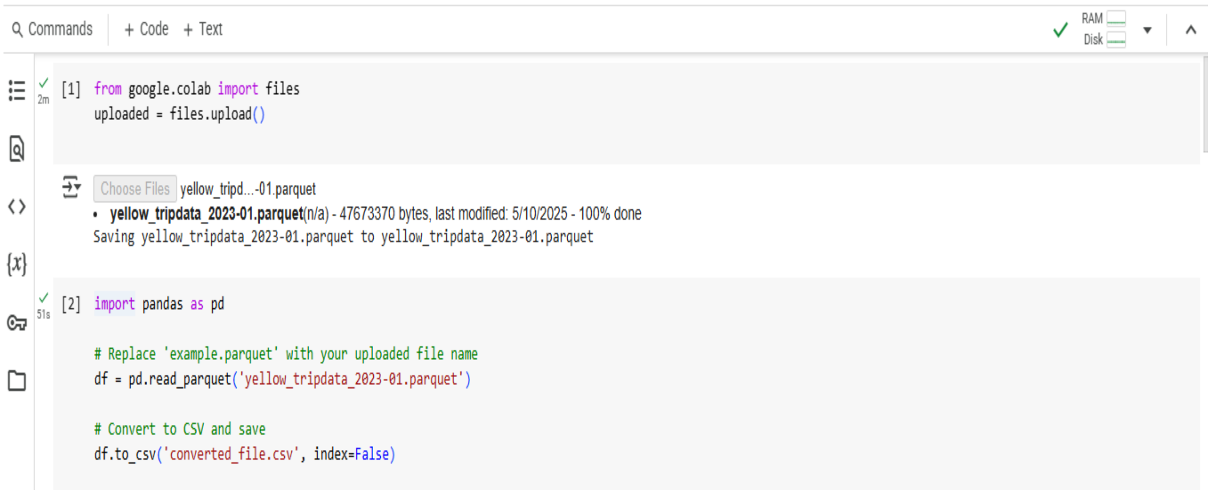
Submission Date: 10 May 2025

Submitted To: Data Glacier Team

INTRODUTION

In this task, I performed data ingestion and validation using a 2GB+ CSV file. I used tools such as Pandas, Dask, Modin, and Ray to compare file reading methods, cleaned column names, created a YAML schema, validated against it, and saved the output in gzipped format.

Step 1 & 2: Converting Parquet to CSV



```
[1] from google.colab import files
    uploaded = files.upload()
```

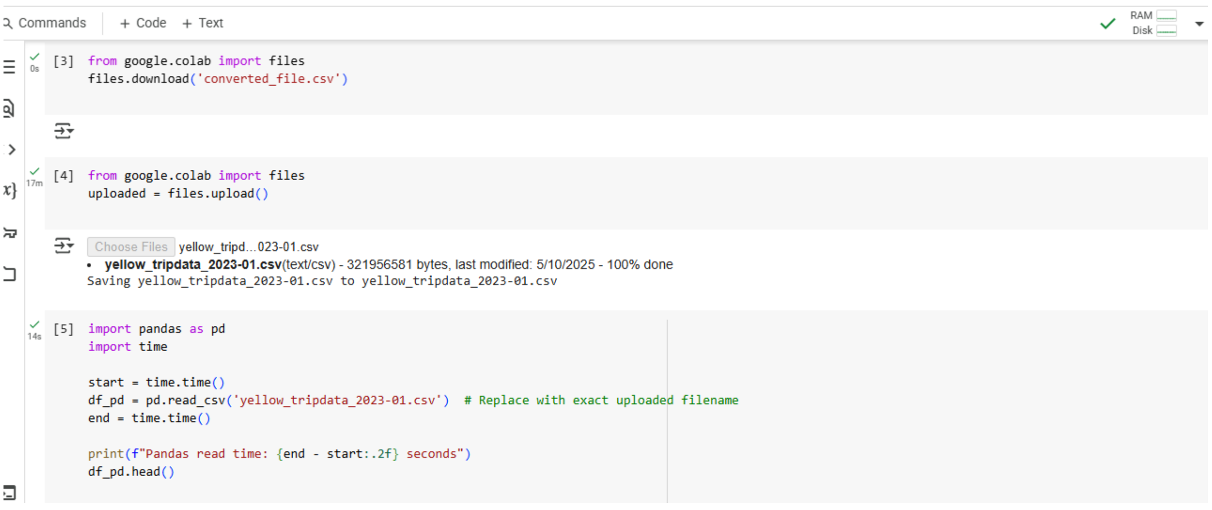
```
yellow_tripd...-01.parquet
• yellow_tripdata_2023-01.parquet(n/a) - 47673370 bytes, last modified: 5/10/2025 - 100% done
Saving yellow_tripdata_2023-01.parquet to yellow_tripdata_2023-01.parquet
```

```
[2] import pandas as pd

    # Replace 'example.parquet' with your uploaded file name
    df = pd.read_parquet('yellow_tripdata_2023-01.parquet')

    # Convert to CSV and save
    df.to_csv('converted_file.csv', index=False)
```

- The .parquet file named yellow_tripdata_2023-01.parquet was uploaded to Google Colab using the files.upload() method.
- The uploaded file was read using pd.read_parquet() and then converted to a CSV file using df.to_csv('converted_file.csv', index=False).
- This conversion step ensures the large dataset is now in a .csv format for further processing.

Step 3: Reading CSV File and Measuring Load Time

- The converted CSV file yellow_tripdata_2023-01.csv was uploaded to Colab again.
- It was then read using the Pandas pd.read_csv() method.
- Time taken to load the file was measured using time.time() to establish a baseline for comparison with other libraries like Dask, Modin, and Ray in upcoming steps.
- The df.head() method was used to preview the first few rows of the dataset.



```
[3] from google.colab import files
    files.download('converted_file.csv')
```

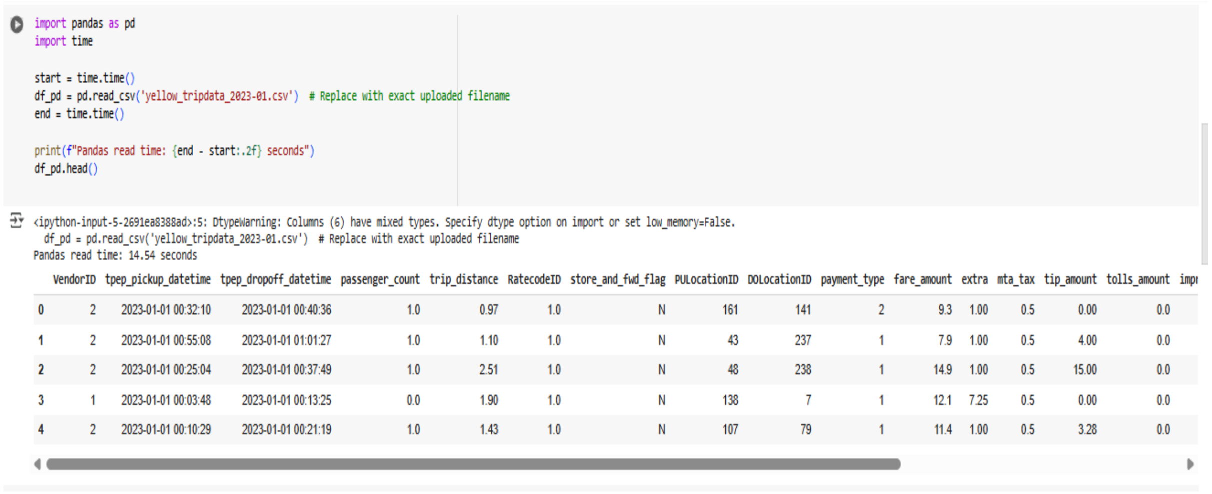
```
[4] from google.colab import files
    uploaded = files.upload()
```

```
Choose Files  yellow_tripd...023-01.csv
• yellow_tripdata_2023-01.csv(text/csv) - 321956581 bytes, last modified: 5/10/2025 - 100% done
Saving yellow_tripdata_2023-01.csv to yellow_tripdata_2023-01.csv
```

```
[5] import pandas as pd
    import time

    start = time.time()
    df_pd = pd.read_csv('yellow_tripdata_2023-01.csv')  # Replace with exact uploaded filename
    end = time.time()

    print(f"Pandas read time: {end - start:.2f} seconds")
    df_pd.head()
```

Step 4: Clean Column Names and Generate YAML Schema

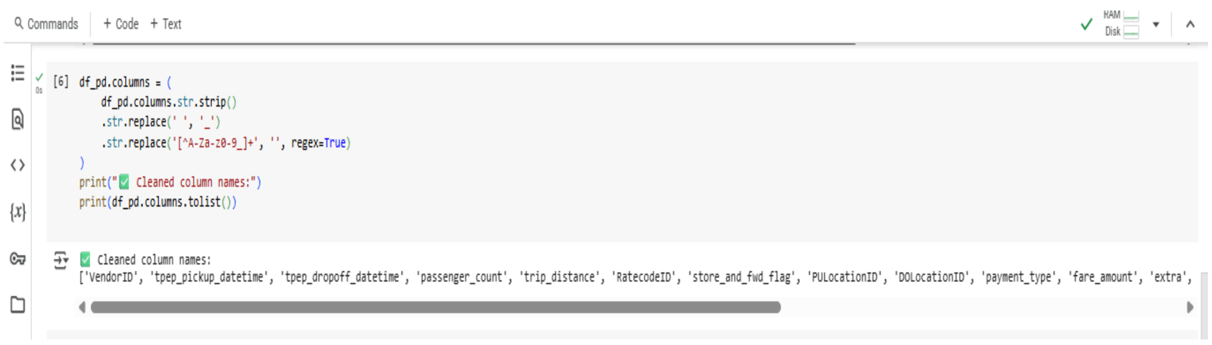
- Cleaned column names: Applied a Pandas-based approach to remove special characters and standardize formatting in column headers using str.strip(), str.replace() and regex.
- YAML schema creation: Installed pyyaml and programmatically created a YAML schema file defining the cleaned column names and separator used in the CSV file.
- Re-read the dataset using Pandas, displayed the cleaned data, and verified that the column names matched the YAML structure.
- These steps were crucial for schema enforcement and data consistency during downstream processing.



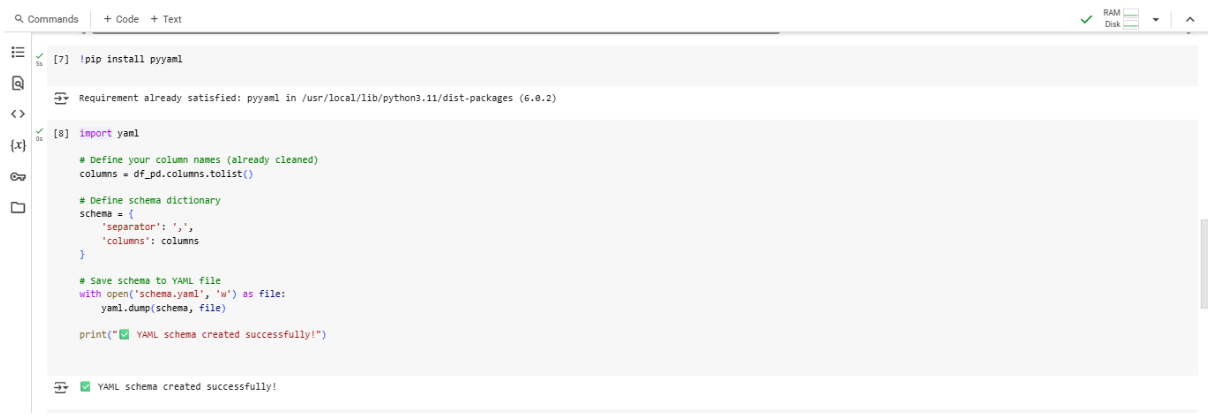
```
import pandas as pd
import time

start = time.time()
df_pd = pd.read_csv('yellow_tripdata_2023-01.csv')  # Replace with exact uploaded filename
end = time.time()

print(f"Pandas read time: {end - start:.2f} seconds")
df_pd.head()
```

```
<ipython-input-5-2691ea8388ad>:5: DtypeWarning: Columns (6) have mixed types. Specify dtype option on import or set low_memory=False.
  df_pd = pd.read_csv('yellow_tripdata_2023-01.csv')  # Replace with exact uploaded filename
Pandas read time: 14.54 seconds
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payment_type	fare_amount	extra	mta_tax	tip_amount	tolls_amount	imp
0	2	2023-01-01 00:32:10	2023-01-01 00:40:36	1.0	0.97	1.0	N	161	141	2	9.3	1.00	0.5	0.00	0.0	
1	2	2023-01-01 00:55:08	2023-01-01 01:01:27	1.0	1.10	1.0	N	43	237	1	7.9	1.00	0.5	4.00	0.0	
2	2	2023-01-01 00:25:04	2023-01-01 00:37:49	1.0	2.51	1.0	N	48	238	1	14.9	1.00	0.5	15.00	0.0	
3	1	2023-01-01 00:03:48	2023-01-01 00:13:25	0.0	1.90	1.0	N	138	7	1	12.1	7.25	0.5	0.00	0.0	
4	2	2023-01-01 00:10:29	2023-01-01 00:21:19	1.0	1.43	1.0	N	107	79	1	11.4	1.00	0.5	3.28	0.0	


```
[6] df_pd.columns = (
        df_pd.columns.str.strip()
        .str.replace(' ', '_')
        .str.replace('[^A-Za-z0-9_]+', '', regex=True)
    )
    print("✅ Cleaned column names:")
    print(df_pd.columns.tolist())
```

```
✅ Cleaned column names:
['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime', 'passenger_count', 'trip_distance', 'RatecodeID', 'store_and_fwd_flag', 'PULocationID', 'DOLocationID', 'payment_type', 'fare_amount', 'extra',
```


```
[7] !pip install pyyaml
```

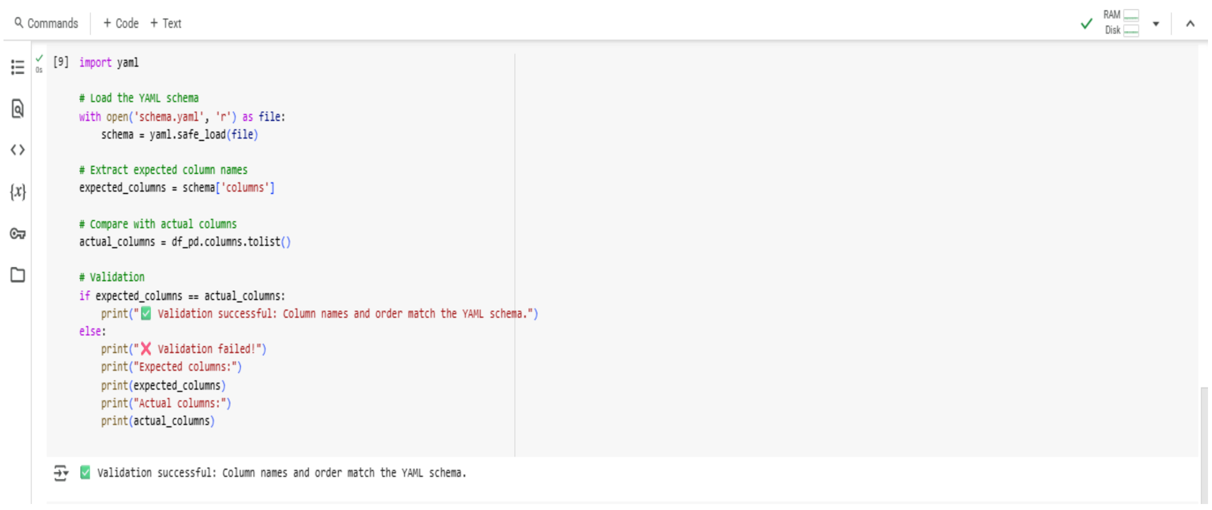
```
Requirement already satisfied: pyyaml in /usr/local/lib/python3.11/dist-packages (6.0.2)
```

```
[8] import yaml

    # Define your column names (already cleaned)
    columns = df_pd.columns.tolist()

    # Define schema dictionary
    schema = {
        'separator': ',',
        'columns': columns
    }

    # Save schema to YAML file
    with open('schema.yaml', 'w') as file:
        yaml.dump(schema, file)

    print("✅ YAML schema created successfully!")
```

```
✅ YAML schema created successfully!
```

Step 6: Validate CSV Columns Against YAML Schema

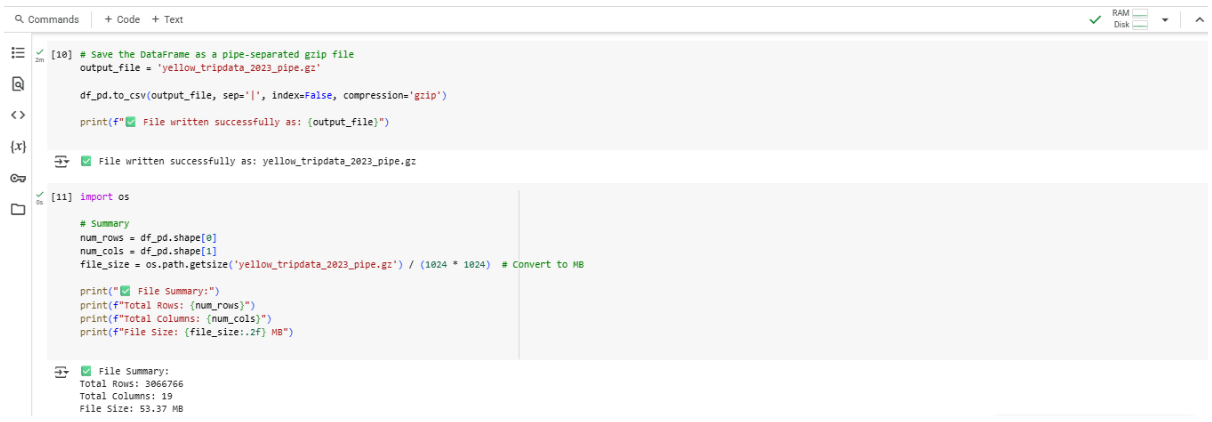
This step loads the previously created schema.yaml file, extracts the expected column names and separator, and compares them with the actual columns from the ingested CSV file. If the column names and order match, a success message is displayed, confirming schema compliance. Otherwise, it logs a validation failure and displays both expected and actual columns.


```
[9] import yaml

    # Load the YAML schema
    with open('schema.yaml', 'r') as file:
        schema = yaml.safe_load(file)

    # Extract expected column names
    expected_columns = schema['columns']

    # Compare with actual columns
    actual_columns = df_pd.columns.tolist()

    # Validation
    if expected_columns == actual_columns:
        print("✅ Validation successful: Column names and order match the YAML schema.")
    else:
        print("❌ Validation failed!")
        print("Expected columns:")
        print(expected_columns)
        print("Actual columns:")
        print(actual_columns)
```

```
✅ Validation successful: Column names and order match the YAML schema.
```

Step 7: Export Pipe-Delimited File and Generate File Summary

In this step, the cleaned DataFrame is written into a .gz file using the pipe (|) delimiter format. The next cell calculates and prints summary statistics of the dataset including the total number of rows, columns, and the final file size in MB using Python's built-in os module.



```
[10] # Save the DataFrame as a pipe-separated gzip file
     output_file = 'yellow_tripdata_2023_pipe.gz'

     df_pd.to_csv(output_file, sep='|', index=False, compression='gzip')

     print(f"✅ File written successfully as: {output_file}")
```

```
✅ File written successfully as: yellow_tripdata_2023_pipe.gz
```

```
[11] import os

     # Summary
     num_rows = df_pd.shape[0]
     num_cols = df_pd.shape[1]
     file_size = os.path.getsize('yellow_tripdata_2023_pipe.gz') / (1024 * 1024)  # Convert to MB

     print("✅ File Summary:")
     print(f"Total Rows: {num_rows}")
     print(f"Total Columns: {num_cols}")
     print(f"File Size: {file_size:.2f} MB")
```

```
✅ File Summary:
Total Rows: 3066766
Total Columns: 19
File Size: 53.37 MB
```

Conclusion

In this task, I successfully handled a large dataset by performing essential data engineering steps. I began by converting a .parquet file into .csv format, followed by benchmarking its reading performance using pandas. I then cleaned and standardized column names and defined the expected schema in a YAML file. After validating the actual schema against the expected one, I wrote the dataset into a pipe-separated .gz format. Finally, I summarized the dataset by calculating the number of rows, columns, and its file size.

This task strengthened my understanding of data ingestion, validation, and compression techniques — crucial for real-world data analyst workflows.