wbwqs8y3l

November 12, 2023

1 File ingestion and schema validation

The datasets provided includes the data of male players for the Career Mode from FIFA 15 to FIFA 23. The data allows multiple comparisons for the same players across the last 9 versions of the videogame.

The size of the dataset is 5.64 GB, it has 10 columns and 10003590 rows

######Uploading file directly from kaggle

[1]:
```
! pip install -q kaggle
```

[2]:
```
! mkdir ~/.kaggle
```

[3]:
```
! cp kaggle.json ~/.kaggle/
```

[4]:
```
! chmod 600 ~/.kaggle/kaggle.json
```

[5]:
```
! kaggle datasets download stefanoleone992/fifa-23-complete-player-dataset
```

```
Downloading fifa-23-complete-player-dataset.zip to /content
 99% 1.57G/1.58G [00:18<00:00, 98.8MB/s]
100% 1.58G/1.58G [00:18<00:00, 93.8MB/s]
```

[6]:
```
! unzip fifa-23-complete-player-dataset.zip
```

```
Archive:  fifa-23-complete-player-dataset.zip
  inflating: female_coaches.csv
  inflating: female_players (legacy).csv
  inflating: female_players.csv
  inflating: female_teams.csv
  inflating: male_coaches.csv
  inflating: male_players (legacy).csv
  inflating: male_players.csv
  inflating: male_teams.csv
```

######Importing all necessary documents

[ ]:
```
!pip install modin
```

[ ]:
```
!pip install ray
```

[ ]:
```
!pip install dask
```

[2]:
```
import numpy as np
import pandas as pd
import modin.pandas as mpd
import ray
import dask.dataframe as dd
```

2 Manipulating file

3 Reading file in pandas

The pandas library operates in-memory, so it loads the entire dataset into memory for processing. In the case of a 5.64 GB file it took 8 minutes and said, "Your session crashed after using all available RAM." From this we can see that it would be good for small to medium-sized datasets that fit comfortably in RAM.

[ ]:
```
df = pd.read_csv("male_players.csv")
```

4 Reading file in Ray

Ray is a distributed computing framework for python. Though it is not specifically designed for reading files. It can be used to parallelize processing tasks but not necessarily file reading itself. In the case of a 5.64 GB file it took 4 minutes and said, "Your session crashed after using all available RAM."

[ ]:
```
# Initialize Ray
ray.init()

# Replace 'your_file.csv' with the actual path to your CSV file
file_path = 'male_players.csv'

# Read the CSV file into a pandas DataFrame
df = pd.read_csv(file_path)

# Parallelize processing using Ray
@ray.remote
def process_data(chunk):
    # Your data processing logic goes here
    # This function will be applied in parallel to different chunks of data

    # For example, you might want to perform some computation on each chunk
    processed_data = chunk.apply(lambda x: x * 2)
```

```
    return processed_data

# Split the DataFrame into chunks for parallel processing
num_chunks = 4  # Adjust this based on your available resources
chunks = np.array_split(df, num_chunks)

# Call the remote function in parallel
result_ids = [process_data.remote(chunk) for chunk in chunks]

# Get the results from Ray
results = ray.get(result_ids)

# Combine the results into a single DataFrame
final_result = pd.concat(results)

# Shutdown Ray
ray.shutdown()

# Display the first few rows of the final DataFrame
print(final_result.head())
```

```
2023-11-12 02:02:43,527 INFO worker.py:1673 -- Started a local Ray instance.
```

5 Reading with Modin

Modin is designed to accelerate pandas operations by parallelizing them. It can automatically scale across available resources, such as multiple CPU cores. In the case of a 5.64 GB file it took 9 minutes, 18 seconds, it crushed and said, "The worker died unexpectedly while executing this task. Check python-core-worker-*.log files for more information."

[ ]:
```
ddf = mpd.read_csv('male_players.csv')
```

6 Reading with Dask

Dask is a parallel computing library that integrates with pandas. It allows you to with larger-than-memory datasets by breaking them into smaller, parallelizable tasks. In the case of a 5.64 GB file it completed the run (read the file) in 0s.

[8]:
```
ddf = dd.read_csv('male_players.csv')
```

[14]:
```
# Get the shape (number of rows and columns) as a tuple
num_rows = len(ddf.compute())

print("Number of Rows:", num_rows)
```

```
Number of Rows: 10003590
```

7 Creating Yaml

[10]:
```
import yaml

## Read csv file
ddf = dd.read_csv('male_players.csv', sep=';')

# Get the column names as a list
column_names = ddf.columns.tolist()

# Replace 'new.yaml' with the desired output YAML file path
output_yaml_path = 'new.yaml'

# Create a dictionary with the column names
yaml_data = {'columns': column_names}

# Write the dictionary to a YAML file
with open(output_yaml_path, 'w') as yaml_file:
    yaml.dump(yaml_data, yaml_file, default_flow_style=False)

print(f"Column names written to {output_yaml_path}")
```

```
Column names written to new.yaml
```

8 Validating number of columns in dataframe and column names in YAML

[11]:
```
# Read the YAML file
with open(output_yaml_path, 'r') as yaml_file:
    loaded_yaml = yaml.safe_load(yaml_file)

# Validate number of columns
expected_num_columns = len(column_names)
actual_num_columns = len(loaded_yaml['columns'])
if actual_num_columns != expected_num_columns:
    print(f"Validation Error: Expected {expected_num_columns} columns, but␣
 ↪found {actual_num_columns} columns.")

# Validate column names
expected_column_names = column_names
actual_column_names = loaded_yaml['columns']
if set(actual_column_names) != set(expected_column_names):
    print("Validation Error: Column names mismatch.")

# If no validation errors, print success message
else:
```

```
    print("Validation successful.")
```

```
Validation successful.
```

9 Pipe separated text file (|) in gz format

[ ]:
```
# Replace 'output_file.csv.gz' with the desired output file path
output_file_path = 'output_file.csv.gz'

# Write the Dask DataFrame to a gzipped pipe-separated text file
ddf.to_csv(output_file_path, sep='|', compression='gzip', index=False,␣
 ↪single_file=True)
```