Process Pool Executors: Takeaways

Syntax

- Breaking a DataFrame into chunks:

```
def make_chunks(df, num_chunks):
    chunk_size = math.ceil(df.shape[0] / num_chunks)
    return [df[i:i+chunk_size] for i in range(0, df.shape[0], chunk_size)]
```

- Using a process pool executor:

```
import concurrent.futures
def mul_string(string, times):
    return string * times
strings = ["a", "b", "c", "d"]
times = 3
with concurrent.futures.ProcessPoolExecutor() as executor:
    futures = [executor.submit(mul_string, string, times) for string in strings] # Create
and run the processes
results = [future.result() for future in futures] # Gather the results
```

- Merging a list of dictionaries into a single one:

```
merged_dict = {}
for dictionary in dictionary_list:
    merged_dict.update(dictionary)
```

Concepts

- The `concurrent.futures` module provides a way to execute functions inside processes and get the result.
- Parallel processing can greatly improve the performance of a data processing task.
- If the data fits in memory, we are better of splitting it after we load it rather than reading it in chunks. However, more often than not, the data doesn't fit into memory. In this case, we can combine what we've learned in the previous course with this lesson to both reduce the memory footprint and accelerate the calculations.

Resources

- Processing Pool Executor
- MapReduce.