FA25 DSC 204A: Scalable Data Systems
Programming Assignment 1

October 2025

Introduction

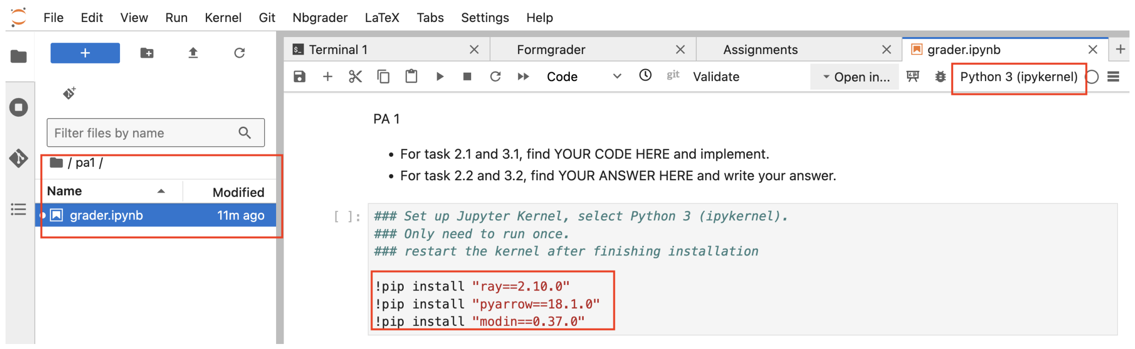
The objective of this programming assignment is to acquaint you with processing large datasets using Ray and Modin. Task 1 guides you in setting up Ray and Modin on Datahub. Task 2 introduces you to computing descriptive statistics on a large dataset and parallelizing fundamental dataframe operations. Task 3 aims to enhance your comprehension of the Ray Core API and parallelizing computations.

Important: All your code and answers will be submitted in `grader.ipynb` on Datahub. For students who have worked on the previous version released on 10/02/2025, this new version won't change your solution. You can still follow Task 1 to set up Datahub and then copy your solutions to the desired places in `grader.ipynb`.

1 Task 1: Set up Datahub

First refer to our seperate Datahub Instruction to fetch `pa1`. You will see there's a `grader.ipynb`, which contains all code and questions for this assignment.

Select Jupyter Kernel as `Python 3 (ipykernel)`, then un-comment and run the `pip install` commands (see the figure below) to install required packages. The installation commands only need to run once.



PA 1

- For task 2.1 and 3.1, find YOUR CODE HERE and implement.
- For task 2.2 and 3.2, find YOUR ANSWER HERE and write your answer.

```
### Set up Jupyter Kernel, select Python 3 (ipykernel).
### Only need to run once.
### restart the kernel after finishing installation

!pip install "ray==2.10.0"
!pip install "pyarrow==18.1.0"
!pip install "modin==0.37.0"
```

2 Task 2: Data Manipulation with Modin (45)

In this question, you will be required to perform some basic data manipulation with Modin (Pandas on Ray). Modin is a library that allows users to perform Pandas workloads at scale. In this assignment, we will focus on parallelizing dataframe operations with Modin.

The dataset is stored in `~/public` folder on Datahub. The dataset has the schema shown in Table 1.

Table 1: The Amazon Reviews Dataset

Column name	**Column description**
`reviewerID`	ID of the reviewer
`vote`	Helpful votes of the review
`overall`	Rating of the product
`unixReviewTime`	Time of the review (unix time)
`reviewTime`	Time of the review (raw)

2.1 Task 2.1

Perform a few data manipulation operations on this dataset and generate a new table with the schema shown in Table 2.

We have provided expected output statistics for you in `~/public/origin_results_PA1_task2_1.json` on Datahub so that you can verify your work. Please use the output function we provided to save the table you generated.

Table 2: Schema for the processed dataframe

Column name	**Column description**
`rating_bucket`	Overall rating rounded to the nearest integer rating bucket.
`n_reviews`	Number of reviews in this rating bucket.
`n_unique_reviewers`	Distinct reviewers in this bucket.
`avg_helpful_votes_per_review`	Average helpful votes per review in this bucket.
`earliest_review_year`	Earliest review year observed in this bucket.

Hints.

- `rating_bucket`: round `overall`; store as a nullable integer(int64).
- `vote`: normalize formatting and treat missing values as 0.0 before aggregation.
- `reviewYear`: convert from `unixReviewTime`

2.2 Task 2.2

Change the number of CPUs used by Modin or the Ray backend (documentation here) on your instance and run your data manipulation code. Document the execution times you see with 1, 2, 3, and 4 CPUs. Is it a linear speedup? If not, why?

2.3 Grading Rubric

For Task 2.1, there are 4 columns (apart from the `rating_bucket`) in the output table. If all descriptive stats (mean, std dev, min, and max) with 1% error margin match the ground truth, we award 10 points per column. Task 2.2 is worth 5 points.

3 Task 3: The Ray Core API (55)

This question will focus on becoming familiar with the Ray Core API, with the three key abstractions: Ray tasks, Ray actors, and Ray objects. Strictly speaking, you only need to know about parallelizing computations with Ray tasks to solve this question.

3.1 Task 3.1

We will implement a distributed merge sort algorithm in Python. The standard merge sort algorithm uses a divide and conquer strategy to partition a given sequence into two halves, and then recursively sorts these two halves. The crucial phase is the merge step, where two sorted halves are merged to get a final sorted list. We will focus on a slightly modified version of the algorithm:

- In the first stage, the input sequence is partitioned into 4 subsequences.
- Next, each subsequence is sorted through a call to the standard merge sort algorithm.
- Finally, the 4 sorted subparts are merged using a heap-based merge algorithm.

A plain implementation of this algorithm is provided to you. Your task is to use Ray to parallelize computations to utilize 4 CPUs in your local machine.

Parts of the code that can and cannot be modified are also highlighted in the comments in `grader.ipynb`. Your goal should be to get a speedup (measured as the ratio of time taken for plain sorting / time taken with Ray) above 1.50.

P.S. You're not supposed to make algorithmic changes here (i.e., change the heap-based algorithm to something else, etc.). Focus on what tasks can and can't be parallelized and use Ray.

3.2 Task 3.2

The ideal speedup you can get for a task with 4 workers is, of course, 4. Can you estimate the theoretical maximum speedup you can get for the above merge sort algorithm (in terms of the time for sorting sublists, and the time for merging)? How do you account for the difference between the theoretical result and the observed speedup with Ray?

3.3 Grading Rubric

Task 3.2 is worth 5 points. For Task 3.1, your merge sort code will first be checked for accuracy: we expect the code to accurately sort the given list. Incorrect code will not receive any points. For runtime, we will run your scripts thrice and get the average runtime. We will use the rubric in Table 3.

Table 3: Runtime scoring rubric for Task 3.1

Speedup S	**Points**
$S \geq 1.40$	50
$1.20 \leq S < 1.40$	40
$1.00 \leq S < 1.20$	20
$S < 1.00$	0

4 Deliverables

All solutions (2.1, 2.2, 3.1, 3.2) will be submitted in Datahub. Refer to our seperate Datahub Instruction on how to submit.

5 Tips

- The assignment will require a substantial amount of RAM. We suggest that you do not have any other on the Datahub server while your code is running, to avoid Out-of-Memory errors.
- While testing your code, it might be helpful to operate on a smaller version of the given data (for example, a smaller list in Task 3) in order to catch errors and iterate faster.
- Use separate columns for raw and processed data in Task 2 to prevent accidental data corruption.

Additional references:

- Modin on Ray: `https://docs.ray.io/en/latest/ray-more-libs/modin/index.html`
- Configuring Modin resources: `https://modin.readthedocs.io/en/stable/getting_started/using_modin/using_modin_locally.html#advanced-configuring-the-resources-modin-uses`