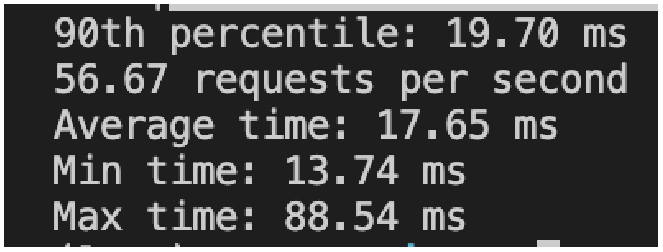
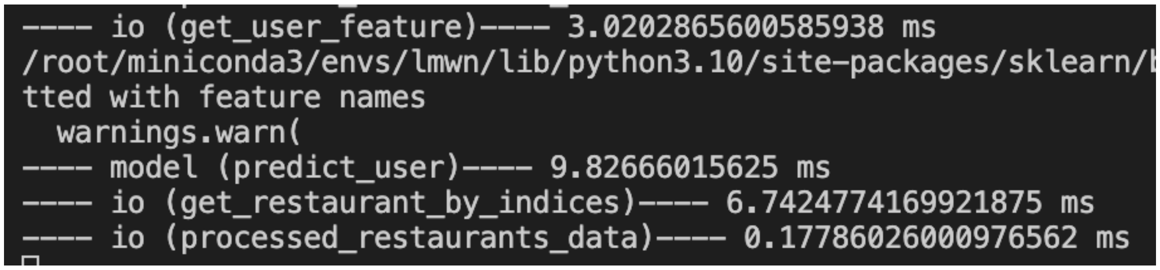
Final Result



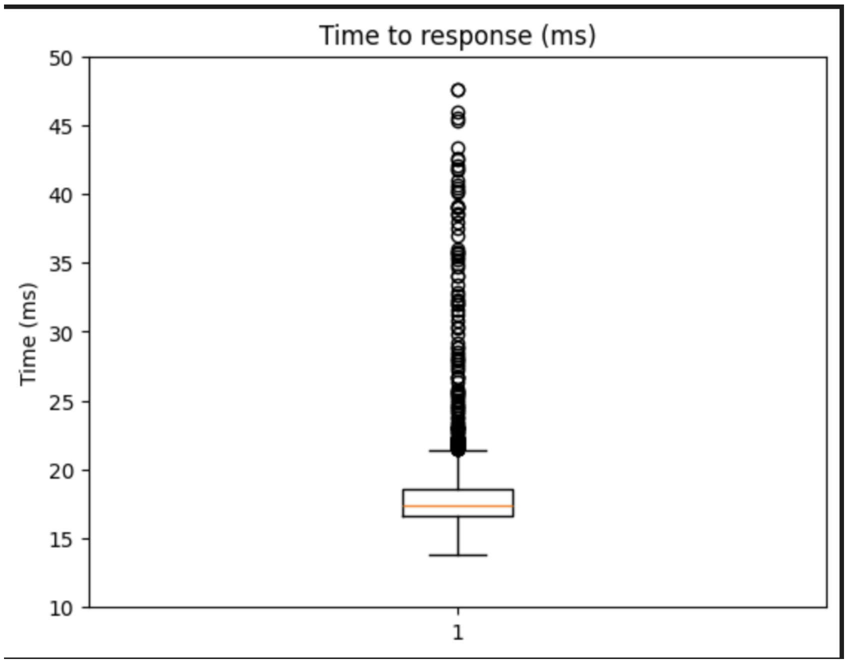
```
90th percentile: 19.70 ms
56.67 requests per second
Average time: 17.65 ms
Min time: 13.74 ms
Max time: 88.54 ms
```

Performance results from running `test_perf.py` on 2,000 requests.



```
---- io (get_user_feature)---- 3.0202865600585938 ms
/root/miniconda3/envs/lmwn/lib/python3.10/site-packages/sklearn/b
tted with feature names
  warnings.warn(
---- model (predict_user)---- 9.82666015625 ms
---- io (get_restaurant_by_indices)---- 6.7424774169921875 ms
---- io (processed_restaurants_data)---- 0.17786026000976562 ms
```

Performance on each operation.



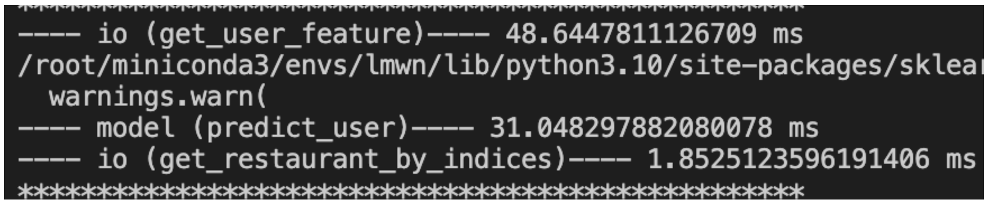
Box plots show distribution of response time for each request.

Past Result

This was the result before I did the optimization.

- Reqs/S = 24.2588
- 90th Percentile = 59.8761 ms

So I decided to explore which step of the request (IO/Model) is slow.



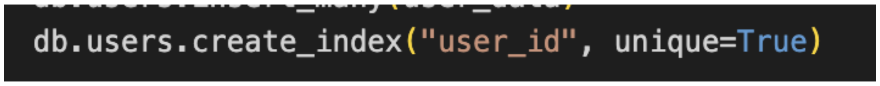
```
************************************************
---- io (get_user_feature)---- 48.6447811126709 ms
/root/miniconda3/envs/lmwn/lib/python3.10/site-packages/sklea
  warnings.warn(
---- model (predict_user)---- 31.048297882080078 ms
---- io (get_restaurant_by_indices)---- 1.8525123596191406 ms
************************************************
```

Performance on each operation.

get_user_feature: Query to get features by user_id.
predict_user: Model prediction
get_restaurant_by_indices: Query restaurant data by indices from the model (At that time this function didn't include geodesic computation so it was faster than now)

I thought that get_user_feature could be faster than this so I decided to search on the internet how to make this query part faster.

And then I found out that creating an index on the user_id field can make the query faster so I implemented it with just one line of code.



```
db.users.create_index("user_id", unique=True)
```

Create index on user_id field (create-database.py)

Only this line of code can improve overall performance by almost 95%.