Project Name:

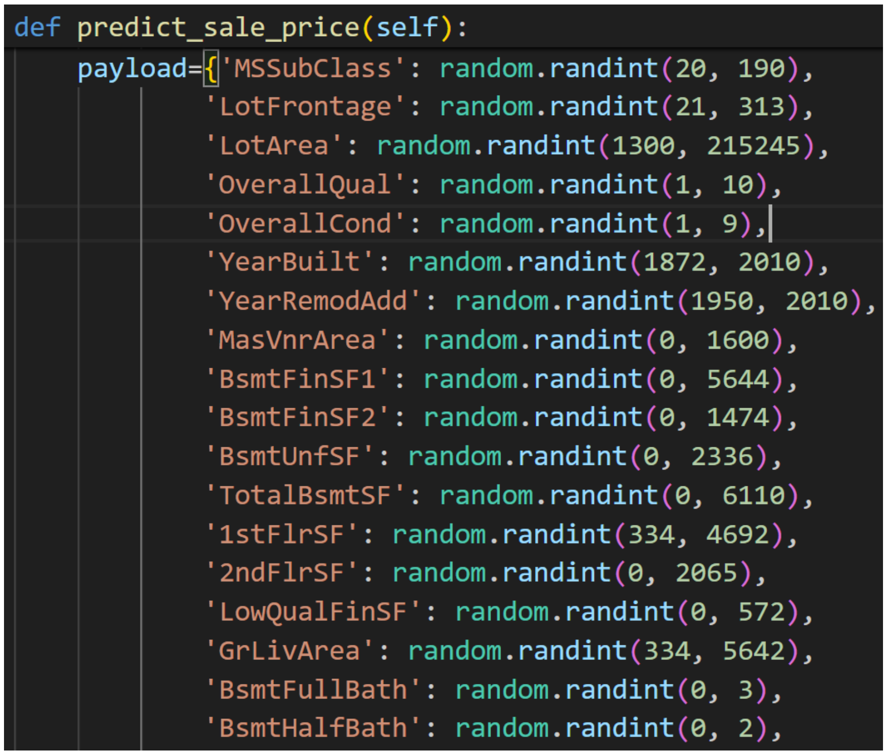
Real Estate Machine Learning Pricing Model

Week 4

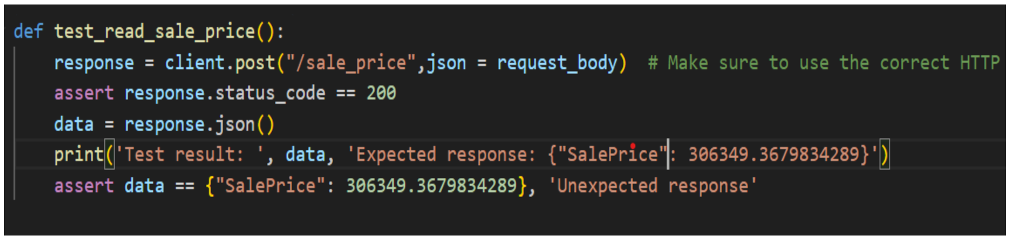
4-10-2024

Test & Load Test

Previous week, our team successfully used FastAPI to make interaction that allows consumer to interact with our model. This week, we continue the work that we have done last week and begin our load test for our model. First of all, our test for the model is as follows:

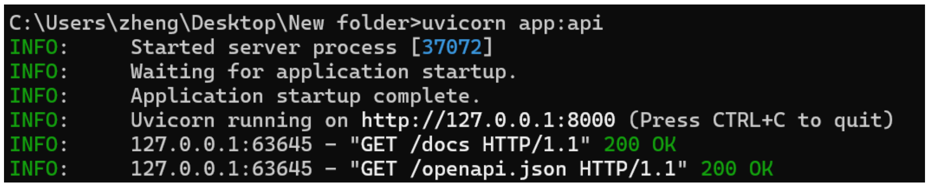
1. Given the random input of with a reasonable range based on the training dataset, all the feature data are set and sort to Jason format for model processing it:
   a. 

```python
def predict_sale_price(self):
    payload={'MSSubClass': random.randint(20, 190),
            'LotFrontage': random.randint(21, 313),
            'LotArea': random.randint(1300, 215245),
            'OverallQual': random.randint(1, 10),
            'OverallCond': random.randint(1, 9),
            'YearBuilt': random.randint(1872, 2010),
            'YearRemodAdd': random.randint(1950, 2010),
            'MasVnrArea': random.randint(0, 1600),
            'BsmtFinSF1': random.randint(0, 5644),
            'BsmtFinSF2': random.randint(0, 1474),
            'BsmtUnfSF': random.randint(0, 2336),
            'TotalBsmtSF': random.randint(0, 6110),
            '1stFlrSF': random.randint(334, 4692),
            '2ndFlrSF': random.randint(0, 2065),
            'LowQualFinSF': random.randint(0, 572),
            'GrLivArea': random.randint(334, 5642),
            'BsmtFullBath': random.randint(0, 3),
            'BsmtHalfBath': random.randint(0, 2),
```

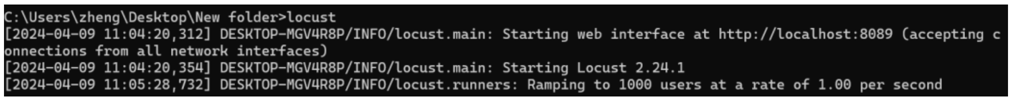
2. The model will run with request from the individual to the API. The output will either be 'Expected response' or 'Unexpected response' in operate panel:
   a. 

```python
def test_read_sale_price():
    response = client.post("/sale_price",json = request_body)  # Make sure to use the correct HTTP
    assert response.status_code == 200
    data = response.json()
    print('Test result: ', data, 'Expected response: {"SalePrice": 306349.3679834289}')
    assert data == {"SalePrice": 306349.3679834289}, 'Unexpected response'
```

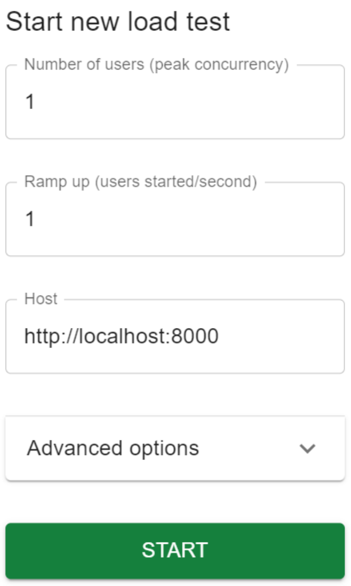
With initial attempts from manually inputting different test data, the model and API work perfectly without any error. We then move onto the load test. Prior to using the load test, following steps must be completed:

1. Unzip the API.zip to a specific folder location of your choice.
    a. All the codes in the API.zip files are already prepared so Locust will successfully run.
2. Open the Command Prompt, and cd your file location.
    a. For example, once the window is prompted, you can enter ‘cd [file location]’
3. Then enter ‘uvicorn app:api’
    a. Note: All the packages and installation must be completed from week
    b. Output as follow:
        i. 

```
C:\Users\zheng\Desktop\New folder>uvicorn app:api
INFO:     Started server process [37072]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     127.0.0.1:63645 - "GET /docs HTTP/1.1" 200 OK
INFO:     127.0.0.1:63645 - "GET /openapi.json HTTP/1.1" 200 OK
```

4. Open another Command Prompt Window
5. Pip install locust
6. Repeat the cd your file location step from step 2
7. Enter code: locust after cd your file
8. Open the link to access the load test, given the output from the code
    a. Example: http://localhost: 8089
        i. 

```
C:\Users\zheng\Desktop\New folder>locust
[2024-04-09 11:04:20,312] DESKTOP-MGV4R8P/INFO/locust.main: Starting web interface at http://localhost:8089 (accepting c
onnections from all network interfaces)
[2024-04-09 11:04:20,354] DESKTOP-MGV4R8P/INFO/locust.main: Starting Locust 2.24.1
[2024-04-09 11:05:28,732] DESKTOP-MGV4R8P/INFO/locust.runners: Ramping to 1000 users at a rate of 1.00 per second
```

9. Lastly, you will be able to begin load test by your choice of load size on the local window.
    a. 

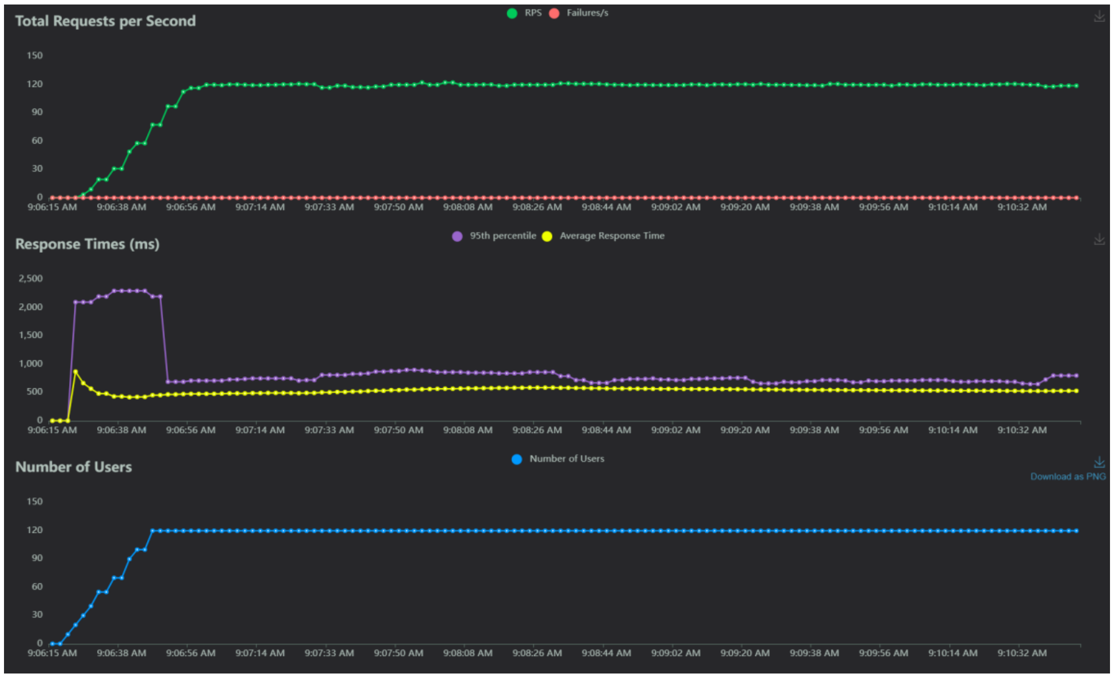
Start new load test

Number of users (peak concurrency): 1

Ramp up (users started/second): 1

Host: http://localhost:8000

Model's Load Test Performance



There are three graphs in the above exhibit:

The first graph shows a steady increase in request per second (RPS) over time, plateauing at around 150 RPS. There are no failures reported, as the failure line is flat at zero. (Show as the red line)

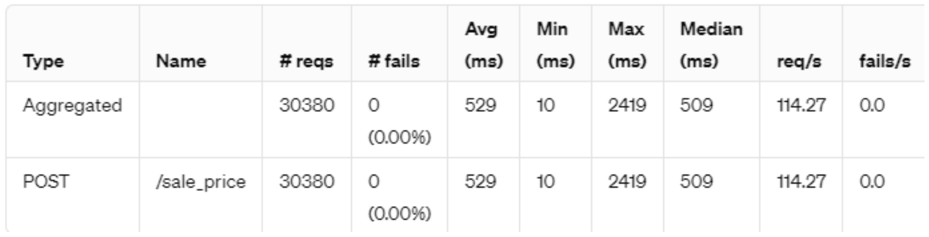
The second graph displays response times with two metrics:

1. The 95 percentile
2. The average response rate

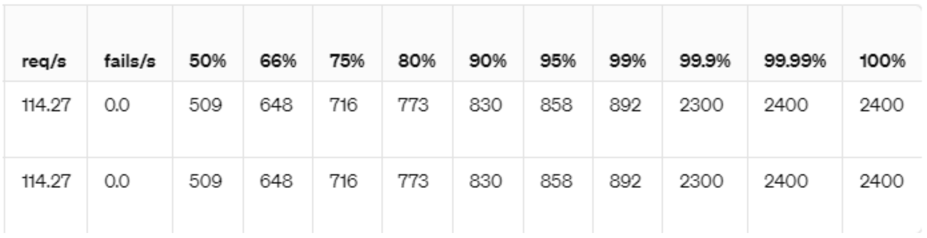
Initially, the response time spikes significantly, with the 95th percentile reaching just above 2,000 milliseconds (2 seconds), while the average response time climbs to just under 500 milliseconds. After the spike, both line drops back down to a stable level of 100 milliseconds for the 95th and 200 milliseconds for average.

The third graph indicates the number of users over the same time frame. There is a consistent increase in number of users, mirroring the pattern seen in the request per second. It intuitively makes sense why those two patterns are similar and it stabilizes around 150 users.

Overall, for our initial expectation on smoothness of the system, we consider the overall health of the service to be good especially it also shows the absence of failures. We also put some of the system performance output into follow table for better view:



Type	Name	# reqs	# fails	Avg (ms)	Min (ms)	Max (ms)	Median (ms)	req/s	fails/s
Aggregated		30380	0 (0.00%)	529	10	2419	509	114.27	0.0
POST	/sale_price	30380	0 (0.00%)	529	10	2419	509	114.27	0.0



req/s	fails/s	50%	66%	75%	80%	90%	95%	99%	99.9%	99.99%	100%
114.27	0.0	509	648	716	773	830	858	892	2300	2400	2400
114.27	0.0	509	648	716	773	830	858	892	2300	2400	2400

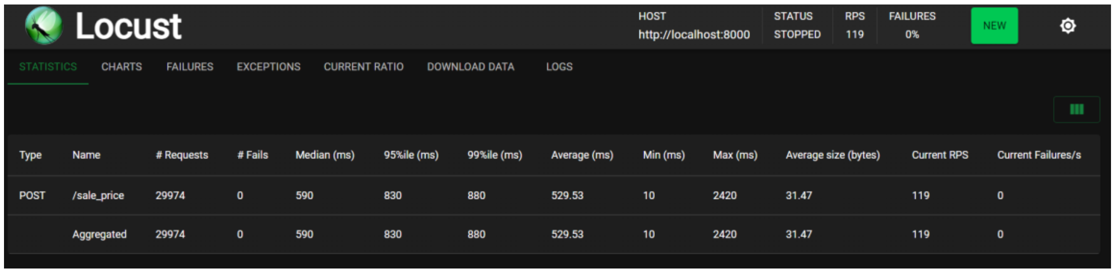
Note: The 50% to 100% represent various percentiles of response time. The output shows the same conclusion explained above. We also attached additional screenshots at the end of the memo.

Future Adjustment Considerations

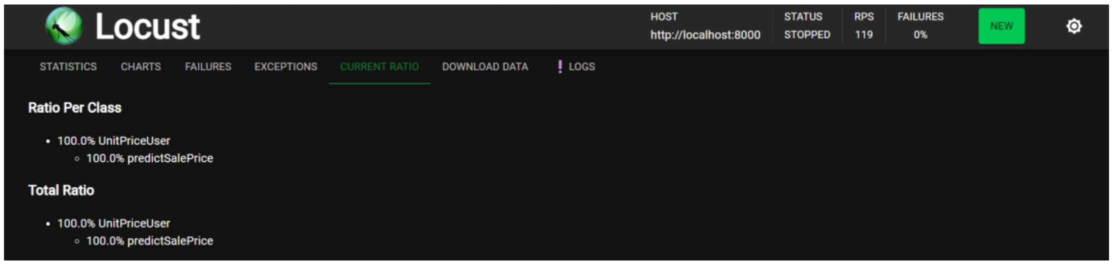
After successful completion of all the steps from previous weeks, our model is now successfully run and can produce a reliable/expected output. One thing that we will consider when we start building the UIUX for our product is reasonability.

It's not practical for user to enter a Jason format input for all ~70 features. Our team now is considering an approach where we will select 15 ~ 20 features that the model deems as significant. (Regression Model) We will also consider some field specific knowledge when picking the features. Ideally, the users will be able to just enter 15 ~ 20 info regarding their house, the model will be able to predict a price or price range (we haven't decided yet given the potential loss of model accuracy if it's a specific price) for them. For the rest of the features (~50), we will randomly select the values for them so that the front-end and back-end will be able to connect and run smoothly.

Additional Screenshots



Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
POST	/sale_price	29974	0	590	830	880	529.53	10	2420	31.47	119	0
	Aggregated	29974	0	590	830	880	529.53	10	2420	31.47	119	0

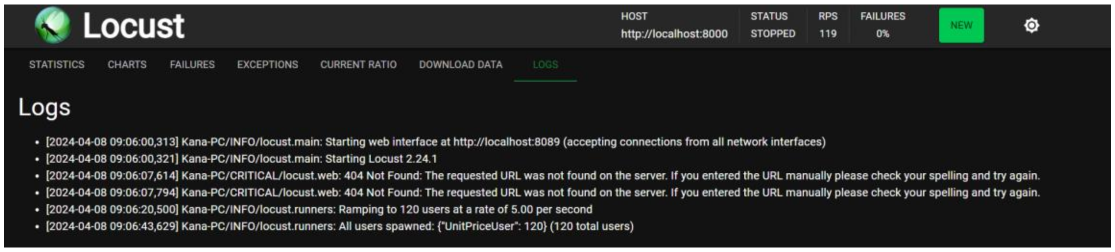


Ratio Per Class

- 100.0% UnitPriceUser
  - 100.0% predictSalePrice

Total Ratio

- 100.0% UnitPriceUser
  - 100.0% predictSalePrice



Logs

```
• [2024-04-08 09:06:00,313] Kana-PC/INFO/locust.main: Starting web interface at http://localhost:8089 (accepting connections from all network interfaces)
• [2024-04-08 09:06:00,321] Kana-PC/INFO/locust.main: Starting Locust 2.24.1
• [2024-04-08 09:06:07,614] Kana-PC/CRITICAL/locust.web: 404 Not Found: The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.
• [2024-04-08 09:06:07,794] Kana-PC/CRITICAL/locust.web: 404 Not Found: The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.
• [2024-04-08 09:06:20,500] Kana-PC/INFO/locust.runners: Ramping to 120 users at a rate of 5.00 per second
• [2024-04-08 09:06:43,629] Kana-PC/INFO/locust.runners: All users spawned: {"UnitPriceUser": 120} (120 total users)
```