Parita Chavda **paritamavani96@gmail.com** **+1 571-626-1674**

Model Deployment Process

- In this task, I used Flask to create an API. I imported the ML classification model and wrote a function to predict and return a JSON response.
- Also, I used docker to conternerized the API and serve the model easily.
- I wrote the unit tests to test the model for various scenarios like single/multiple prediction, missing valued data etc.

Project structure

```
root
   |- docker-compose.yml
   |- generate_predictions.sh
   |- README.md
   |- .gitignore
   |- predictionapi
      |- __init__.py
      |- app.py
      |- data_preprocessing.py
      |- requirements.txt
      |- Dockerfile
      |- unit_test.py
      |- endpoints
         |- __init__.py
         |- predict.py
      |- models
         |- model.pkl
         |- imputer.pkl
         |- scaler.pkl
      |- test
         |- __init__.py
         |- test_prediction_api.py
```

root

- ‘docker-compose.yml’ is the compose file that documents and configures all of the application's service dependencies.
- generate_predictions.sh’ is the shell script that runs the ‘docker-compose.yml’ file.
- README.md is the project description file

predictionapi

- ‘app.py’ is the main script that runs our app at port 1313.
- data_preprocessing.py’ is the script to preprocess the test data. It uses ‘scaler.pkl’ and ‘imputer.pkl’ to scale and impute the missing data, as well as create dummy variables and returns only those needed by the model.
- ‘requirements.txt’ is the file with all the packages needed to run the docker container.
- ‘Dockerfile’ is the dockerfile to create the docker image.
- ‘unit_test.py’ is the script that runs all tests.

Endpoints

Step 1: Create a prediction endpoint in predict.py

- I initialized the API as a blueprint instead of a Flask app and called it “prediction_api”.

Step 2: Register the endpoint in app.py

- Now that we have the prediction.py file to deal with the endpoint code, we can simply import the blueprint and use it in the Flask app.

Models

I exported model and preprocessing variables as pickle file i.e 'model.pkl', ‘imputer.pkl’ and ‘scaler.pkl’ from the Notebook provided and used in data_preprocessing.py’ and predict.py

Test (unit testing)

Step 1: Write tests(test_prediction_api.py)

- I registered prediction_api blueprint, defined in predict.py
- Later, I stored the app.test_client() in a local variable called “tester”. This will give us access to the API, as if we are hitting it with actual traffic.
- I defined the test functions, i.e. test_predict_single, test_predict_multiple, and test_predict_missing. Here, I send a POST request on the '/prediction' endpoint with the content_type is JSON. The returned response is also in JSON format on which we can write assert statements..
- Here, I am only testing the API and not the model as the model has already been validated and it is assumed the predictions are accurate. Therefore, the assert statements are testing the status code returned(which should be 200) for the input JSON, and the response data is not null.

Step 2: Write a script to run all tests (unit_test.py)

I wrote a script called 'unit_test.py' to run all the tests.