Peyton Bailey

August 15th, 2025

D602 – Task 3

This final task of the class was the Deployment phase, where the goal was to create an API that provides departure predictions based on the machine learning model developed during the previous phase. The API should accept inputs for an arrival airport, departure time, and arrival time, then return the average departure delay. After creating this, it needed to be containerized into Docker for deployment. Overall, I completed this task even though I found it to be the most challenging out of all the tasks, thus far. I will walk through the steps I took and the challenges I faced during the process.

The task began with an unfinished API script. The script had already defined a function called "create_airport_encoding" which takes an airport string and a dictionary as arguments and returns a one-hot encoded airport array of the length of the number of arrival airports. My first step was to load the airport encodings JSON file and the pickle file containing the finalized Ridge model from the previous task. The finalized model was assigned to a variable named "model". Then I created the FastAPI instance and called the variable "app". Then I made the two API endpoints. The first was the root, which was simple to create. The second endpoint was the one that was responsible for predicting the delays, and this presented many more challenges.

One of the biggest challenges I faced was figuring out how to properly format the input data so the model could properly read it and return an output. The model needed the arrival and departure times to be converted to the number of seconds past midnight. A user would input the time in a standard format, such as military time, and my program would need to convert that time into seconds before inputting it into the model. I spent hours trying to figure out how to create a

function that did so. Once I finally made one that worked, I erroneously included it within the API endpoint. After much troubleshooting, I realized I could put this function right after the "create_airport_encoding" function, mentioned earlier. For simplicity, I decided that the input times would have to be in military format with no colon between the hours and minutes.
First, I defined the prediction delay endpoint by requesting three string inputs: arrival airport, local departure time, and local arrival time. I specified specific parameters for each input, for example, the arrival airport required three characters. Then the "create_airport_encoding" function was performed on the arrival input string. Secondly, the "time_to_seconds" function I'd previously defined converted the arrival and departure times into seconds.

The input data was now almost ready to be fed to the model; however, I still had much difficulty getting it into a format that it could understand. I kept receiving error messages saying the model could not read the input. Eventually, I realized that it needed to be read the polynomial order, which for my model was always 1. I made the mistake of thinking I needed to perform a poly_fit_transform on the data, leading to hours of confusion and troubleshooting. I finally realized that adding "1" as the first input before the encoded airport solved the problem. I did that and reshaped the data so that it was just a single horizontal array, and the model was satisfied. I called model.predict on the input data and told it to return the delay, which I labeled "average departure delay in minutes". I went back and added my HTTPException errors to the arrival airport and time steps as well, in the case of a user inputting an unknown airport or incorrectly formatted time. That was the completion of the main script, and I could successfully test it in the browser.

Next was the test script (test_main.py), which utilized the TestClient feature. I created three tests: one for all valid inputs and two for invalid inputs. I set the params for the valid input

test as all three inputs were formatted correctly. I asserted that the response status code was 200 and the output data included the average departure time. The two invalid tests checked the responses for invalid airport code and invalid time format, respectively. I made them assert that the error code was 400 and that the response contained "Unknown airport code" and "Invalid time format" for the other. This script took a lot of troubleshooting as well before I was able to run it successfully. Lastly, I containerized everything into Docker, and it was ready for deployment. I ran into a bit of an issue with this initially, but I was able to fix it by specifying the Python version in the YAML file.