Model Deployment and API Testing using Heroku and Postman

Submitted By: Preeti Verma

Submitted To: Data Glacier

LISUM19

The Dataset:

I used a dataset from kaggle.com, with the URL: Dataset

It gives us information regarding a person's brain weight given other features like gender, age and head size (in cm^3).

The Model:

Since this deliverable focussed more on the deployment of the model rather than the performance of the model, I have used a simple Linear Regression Model to predict the weight of the brain given user-entered attributes.

Below is the code to train and serialize the model into a pickle file. Pickle was used as it is quite simple to use and understand.

model.py

```python
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import pickle

data = pd.read_csv(r"C:\Users\Preeti\Desktop\Data Glacier Internship\Deployment on Flask\dataset.csv")
X = np.array(data.iloc[:,0:3])
y = np.array(data.iloc[:,3])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

lm = LinearRegression()
lm.fit(X_train, y_train)

filename = 'model.pkl'
pickle.dump(lm, open(filename, 'wb'))
```

Model Deployment:
Once we have our model stored in the pickle file, we can now go on and deploy the model using Flask. In this step, we de-serialize the model back into a python object to send some unseen data into our model through our webpage and predict an output.

The python script for the deployment of our model is shown below.

app.py > predict

```python
from logging import debug
from flask import Flask, request, render_template, jsonify
import numpy as np
import pickle

app = Flask(__name__)
model = pickle.load(open('model.pkl', 'rb'))

@app.route('/', methods=['GET','POST'])
def index():
    # if request.method == 'GET':
    #     data = 'Hello World!'
    return render_template('index.html')
    # return jsonify({'data': data})

@app.route('/predict/', methods=['GET'])
def predict():
    gender = request.args.get('gender')
    age = int(request.args.get('age'))
    head_size = int(request.args.get('head_size'))

    temp_gender = gender
    temp_age = age

    if gender.strip().lower() == 'male':
        gender = 1
    else:
        gender = 2

    if age <= 18:
        age = 2
    else:
        age = 1

    test_in = np.array([gender,age,head_size]).reshape(1,-1)
    pred_weight = model.predict(test_in)
    output = round(pred_weight[0], 2)
    return render_template('result.html', gender="Gender: {}".format(temp_gender), age="Age: {}".format(temp_age),
     head_size="Head Size: {}".format(head_size), prediction_text="Brain Weight is {} grams".format(output))
    # return jsonify({'Brain Weight': output})

if __name__ == "__main__":
    app.run(debug=True)
```

Here, since our training data has classified ‘Male’ as 1 and ‘Female’ as 2, it isn’t intuitive that the user must enter these values. Instead, the webpage takes in the data as Male/Female and the script transforms it into the required form that our model needs to give an output.
Similarly, for age, all ages > 18 are categorized as 1 while ages <= 18 are categorized as 2.
A similar step was done to prepare the user-inputted data for our model.

HTML Webpage:
Now that our Flask app is ready, we needed an interface to interact with the user and get the data needed to perform predictions. For this, an HTML was used. Below is the HTML code written to generate the page.

Landing Page Source Code:

templates > index.html > html

```html
<html>
    <head>
        <meta charset="UTF-8">
        <title>
            Brain Weight Predictor
        </title>
        <style>
            .container{
                padding: 25px;
                background-color: lightblue;
                text-align: center;
            }
        </style>
    </head>
    <body>
        <center>
            <div class="container">
                <h1>
                    Predicting Weight of Brain using Gender, Age and Head Size
                </h1>
                <form action="{{url_for('predict')}}"method="get">
                    <input type="text" id="gender" name="gender" placeholder="Gender (Male or Female)" required="required" /><br>
                    <input type="text" id="age" name="age" placeholder="Age (in Years)" required="required" /><br>
                    <input type="text" id="head_size" name="head_size" placeholder="Head Size (in cm^3)" required="required" /><br>
                    <input type="submit" class="btn btn-primary btn-block btn-Large" value="Predict" />
                    <br>
                    <br>
                    {{ prediction_text }}
                </form>
            </div>
        </center>
    </body>
</html>
```

Result Page Source Code:

templates > result.html > html

```html
<html>
    <head>
        <meta charset="UTF-8">
        <title>
            Brain Weight Predictor
        </title>
        <style>
            .container{
                padding: 25px;
                background-color: lightgreen;
                text-align: center;
            }
        </style>
    </head>
    <body>
        <center>
            <div class="container">
                <h1>
                    Prediction
                </h1>
                <form action="{{url_for('index')}}">
                    <br>
                    {{gender}}
                    <br>
                    {{age}}
                    <br>
                    {{head_size}}
                    <br>
                    <br>
                    <input type="submit" class="btn btn-primary btn-block btn-Large" value="Home" />
                    <br>
                    <br>
                    {{ prediction_text }}
                </form>
            </div>
        </center>
    </body>
</html>
```

Creating requirements.txt, runtime.txt, and Procfile:

For deployment into Heroku, we need these 3 files: requirements.txt; runtime.txt; and Procfile. The functions of these 3 configuration files are given below.

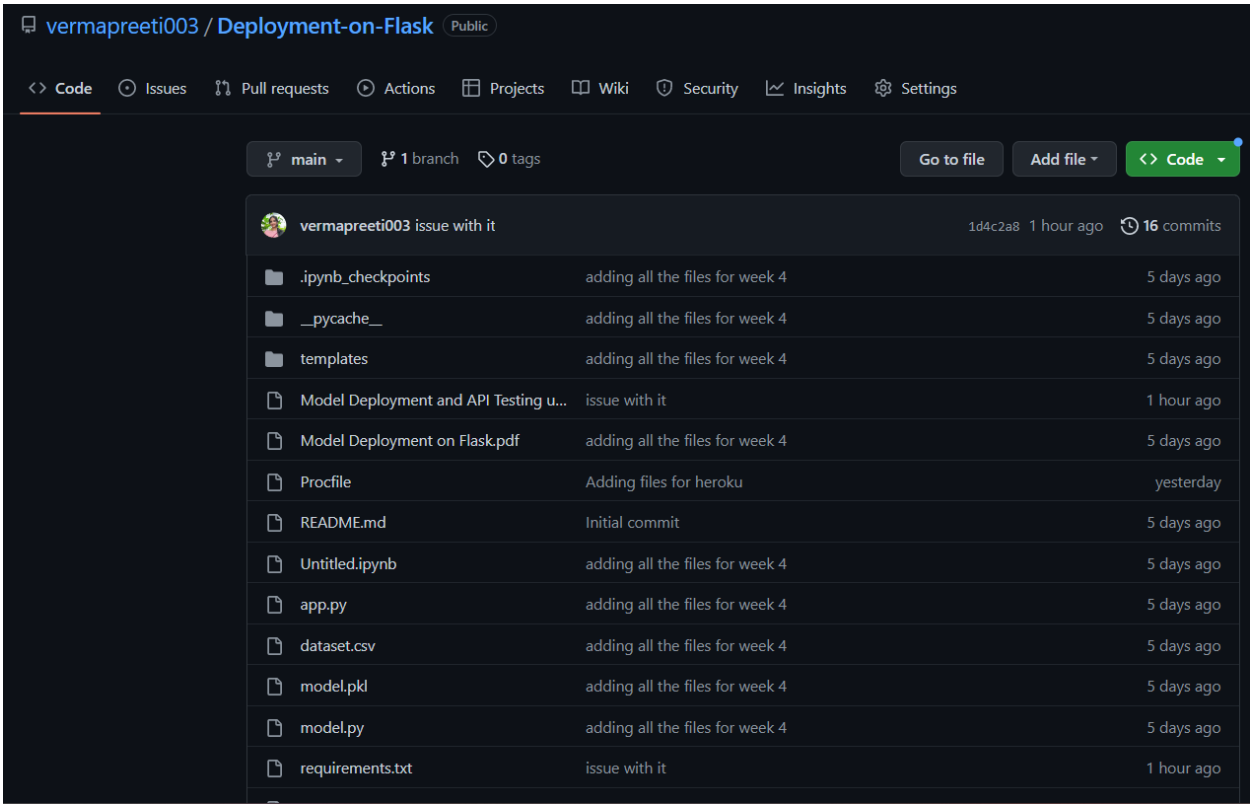
1. requirements.txt: This is a file that contains all the library requirements that our pickled model needs to de-serialize and run to give us the desired output.

2. runtime.txt: This file contains only one line. It specifies the python version that our model and libraries need to run and execute.

3. Procfile: This is a configuration file that specifies the app that we want to run, as well as the web we want to use.

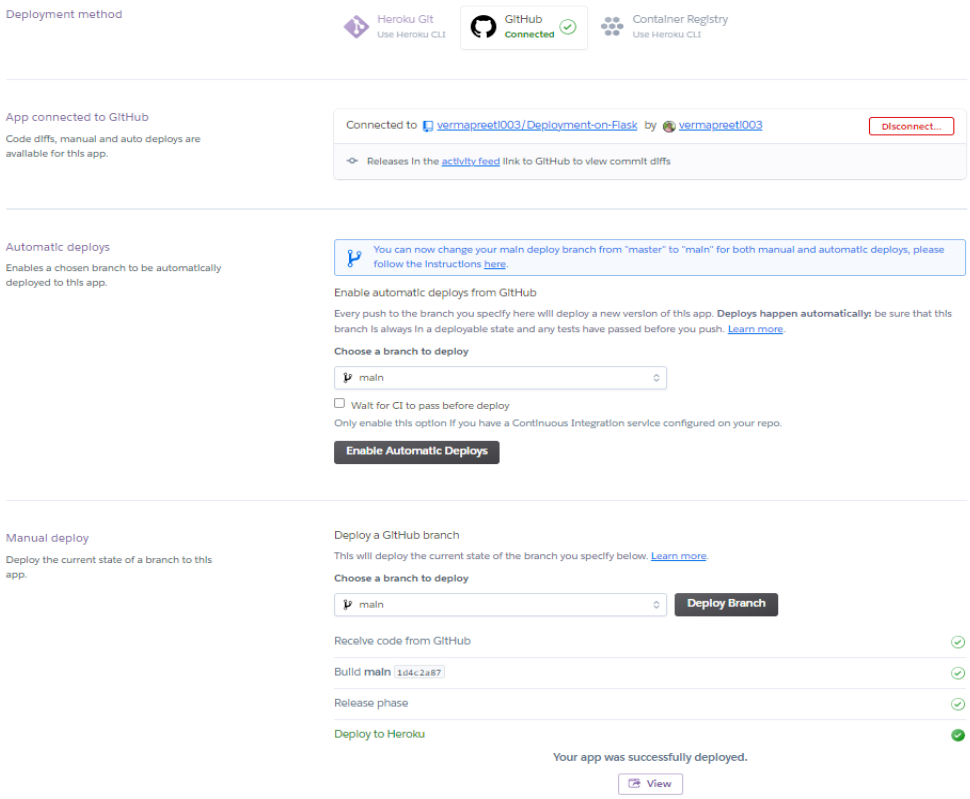
Committing the Code to GitHub:

Once we have deployed the model using Flask and created these configuration files, we will now commit all this code to our GitHub repository. Once we do this, we can connect the GitHub repo to our Heroku and deploy the model.

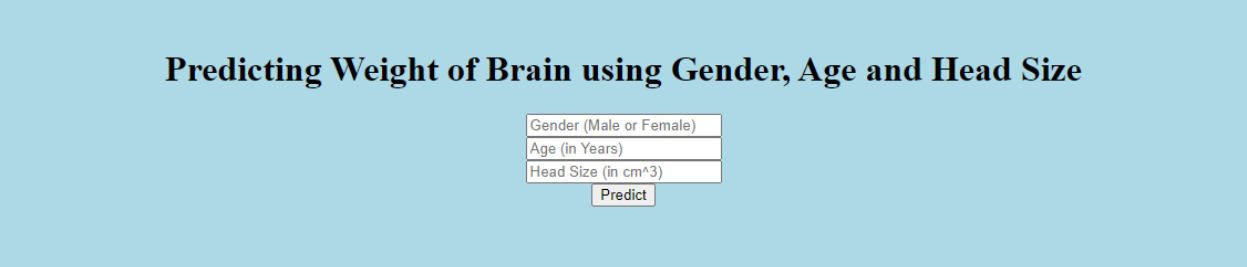


Deploying the Main Branch on Heroku:

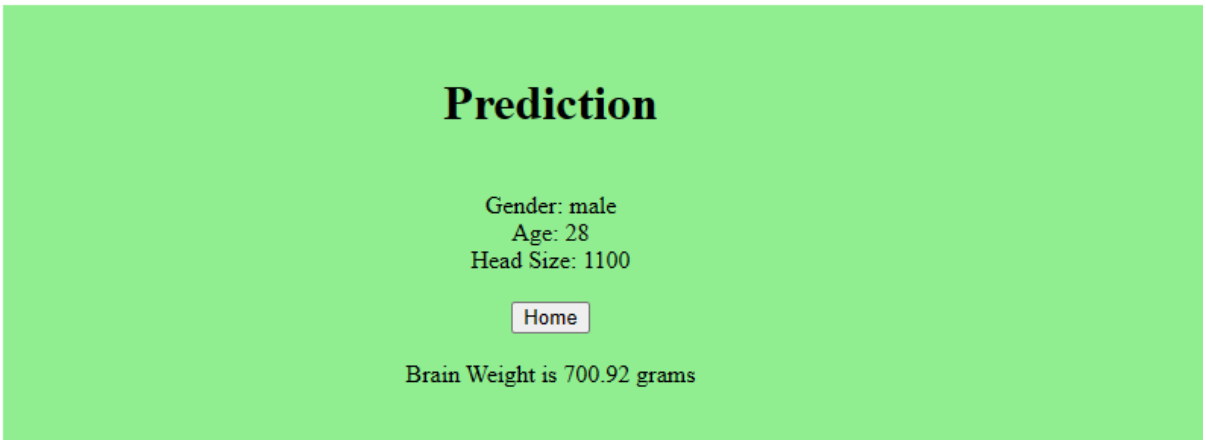
Once our code is committed in GitHub, we can connect it to our Heroku and directly deploy the model. For this deliverable, we will use manual deploy. This means that we will have to redeploy the model with each commit on the repo.



Once our model has been deployed successfully, we can now access the app through the URL: https://heroku-brain-weight-predictor.herokuapp.com/ the landing page we obtained is shown below.



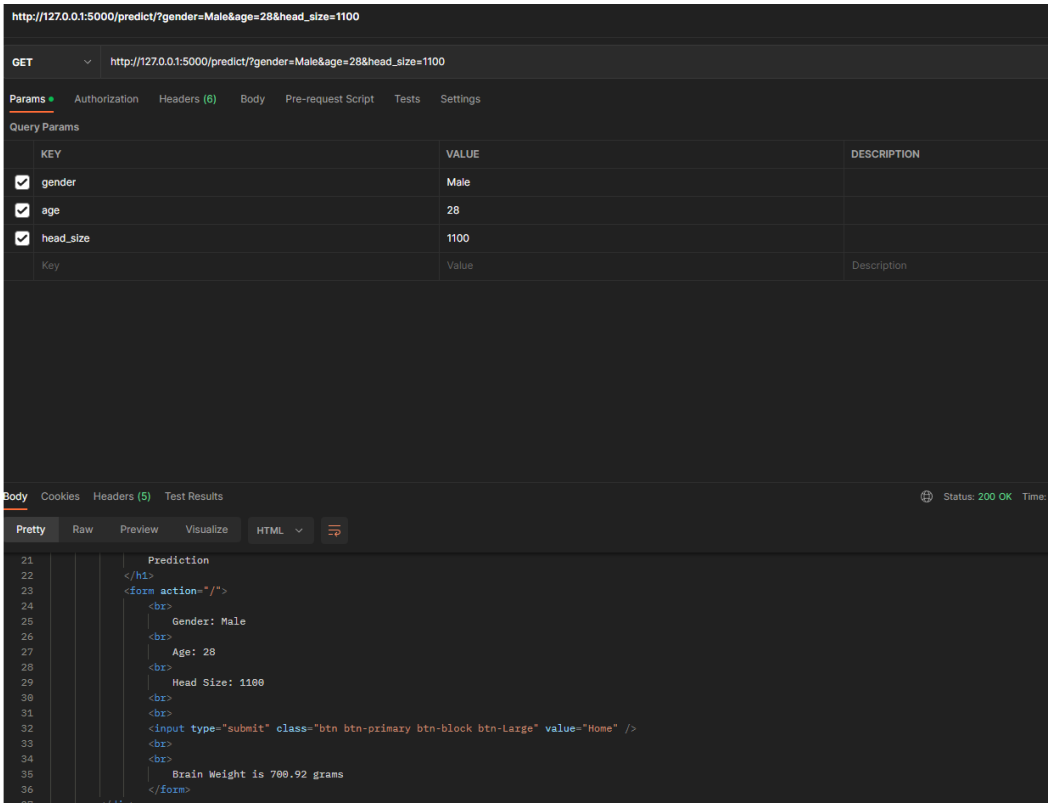
The output page (with example inputs) is shown below.



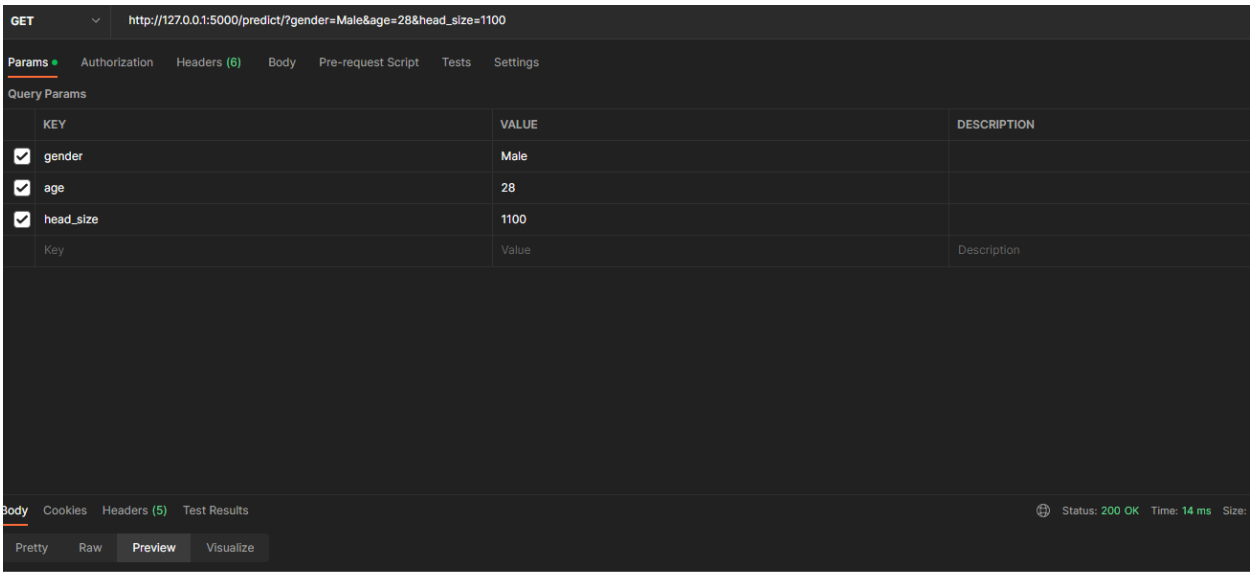
With this, we can see that our app is successfully deployed and working well.

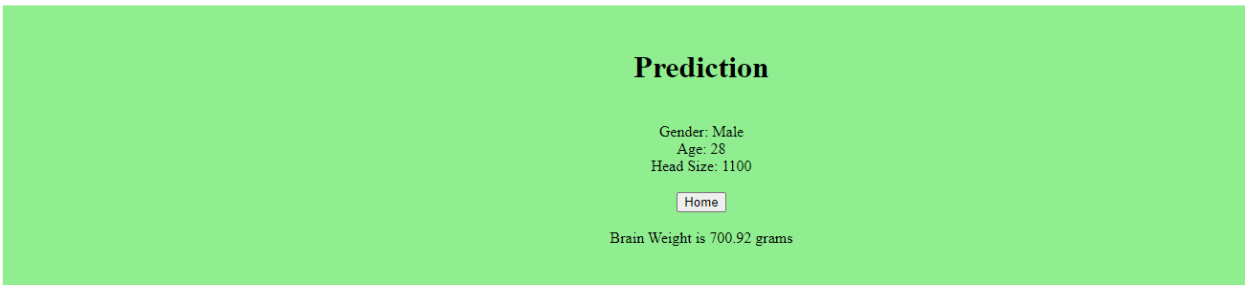
API Testing with Postman:

Now that we have created this API, it is now time to test its correct output with Postman. In order to first check that we can use this to get the desired output, we first tested it in the Flask deployed app that was built as part of the Week 4 Deliverable. Also, to make sure that we were getting the required predicted value, we returned a JSON with just the predicted value. The results of that are shown below.

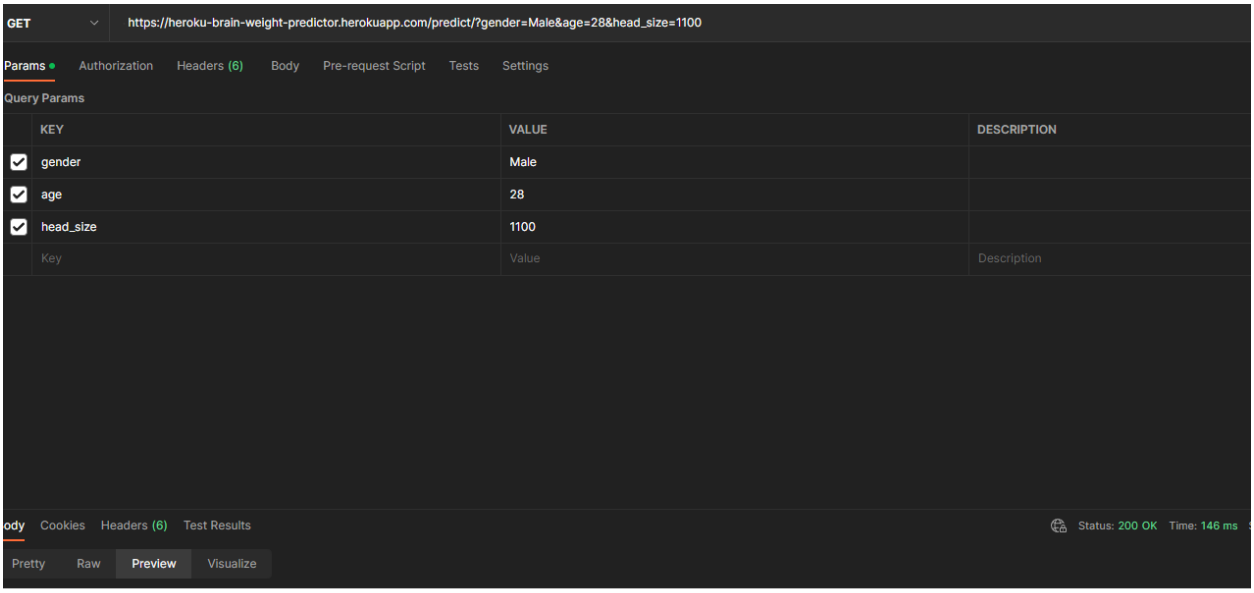


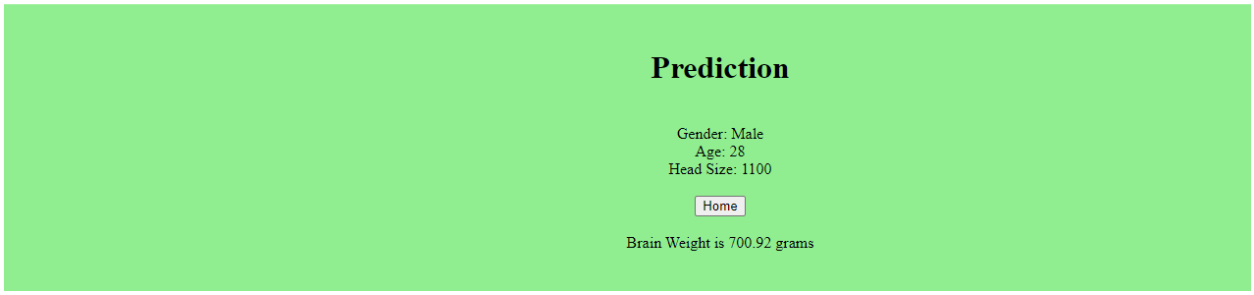
Now that we know that we are getting the right answer, we move on and test it with the original code, which returns an HTML webpage as output. The output for this request is shown below.





We now know that the API hosted on the local server is working perfectly. The last thing we need to test is the model deployed on Heroku. We have used the same inputs in all these tests so that we can make sure we are getting uniform results. The output of the request made to the Heroku app is shown below.





With this step, we have seen that our model deployed on Heroku is working well and can now be accessed globally. Thus, we have deployed our Machine Learning Model using Flask on Heroku and tested it using Postman.