

Model Deployment and API Testing using Heroku and Postman

Submitted By: Udbhav Balaji

Submitted To: Data Glacier

LISUM01

The Dataset:

The dataset used for this deliverable can be found at the URL

<https://www.kaggle.com/anubhabswain/brain-weight-in-humans>.

It gives us information regarding a person's brain weight given other features like gender, age and head size (in cm³).

The Model:

Since this deliverable focussed more on deployment of the model rather than 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 used to train and serialize the model into a pickle file. Pickle was used as it is quite simple to use and understand.

```
model.py
1  import pandas as pd
2  import numpy as np
3  from sklearn.linear_model import LinearRegression
4  from sklearn.model_selection import train_test_split
5  import pickle
6
7  data = pd.read_csv('dataset.csv')
8  X = np.array(data.iloc[:,0:3])
9  y = np.array(data.iloc[:,3])
10
11  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
12
13  lm = LinearRegression()
14  lm.fit(X_train, y_train)
15
16  filename = 'FlaskAPI/model.pkl'
17  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. It is in this step where we de-serialize the model back into a python object so that we can 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.

```
1  from logging import debug
2  from flask import Flask, request, render_template
3  import numpy as np
4  import pickle
5
6  app = Flask(__name__)
7  model = pickle.load(open('model.pkl', 'rb'))
8
9  @app.route('/')
10 def index():
11     return render_template('index.html')
12
13 @app.route('/predict', methods=['POST'])
14 def predict():
15     gender = request.form['gender']
16     age = int(request.form['age'])
17     head_size = int(request.form['head_size'])
18
19     temp_gender = gender
20     temp_age = age
21
22     if gender.strip().lower() == 'male':
23         gender = 1
24     else:
25         gender = 2
26
27     if age <= 18:
28         age = 2
29     else:
30         age = 1
31
32     test_in = np.array([gender, age, head_size]).reshape(1, -1)
33     pred_weight = model.predict(test_in)
34     output = round(pred_weight[0], 2)
35     return render_template('after.html', gender="Gender: {}".format(temp_gender), age="Age: {}".format(temp_age),
36                             head_size="Head Size: {}".format(head_size), prediction_text="Brain Weight is {} grams".format(output))
37
38 if __name__ == "__main__":
39     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:

```
1 <html>
2 <head>
3   <meta charset="UTF-8">
4   <title>
5     Brain Weight Predictor
6   </title>
7   <style>
8     .container{
9       padding: 25px;
10      background-color: lightblue;
11      text-align: center;
12    }
13  </style>
14 </head>
15 <body>
16   <center>
17     <div class="container">
18       <h1>
19         Predicting Weight of Brain using Gender, Age and Head Size
20       </h1>
21       <form action="{{url_for('predict')}}" method="get">
22         <input type="text" id="gender" name="gender" placeholder="Gender (Male or Female)" required="required" /><br>
23         <input type="text" id="age" name="age" placeholder="Age (in Years)" required="required" /><br>
24         <input type="text" id="head_size" name="head_size" placeholder="Head Size (in cm^3)" required="required" /><br>
25         <input type="submit" class="btn btn-primary btn-block btn-Large" value="Predict" />
26         <br>
27         <br>
28         {{ prediction_text }}
29       </form>
30     </div>
31   </center>
32 </body>
33 </html>
```

Output Page Source Code:

```
1 <html>
2 <head>
3   <meta charset="UTF-8">
4   <title>
5     Brain Weight Predictor
6   </title>
7   <style>
8     .container{
9       padding: 25px;
10      background-color: lightgreen;
11      text-align: center;
12    }
13  </style>
14 </head>
15 <body>
16   <center>
17     <div class="container">
18       <h1>
19         Prediction
20       </h1>
21       <form action="{{url_for('index')}}">
22         <br>
23         {{gender}}
24         <br>
25         {{age}}
26         <br>
27         {{head_size}}
28         <br>
29         <br>
30         <input type="submit" class="btn btn-primary btn-block btn-Large" value="Home" />
31         <br>
32         <br>
33         {{ prediction_text }}
34       </form>
35     </div>
36   </center>
37 </body>
38 </html>
```

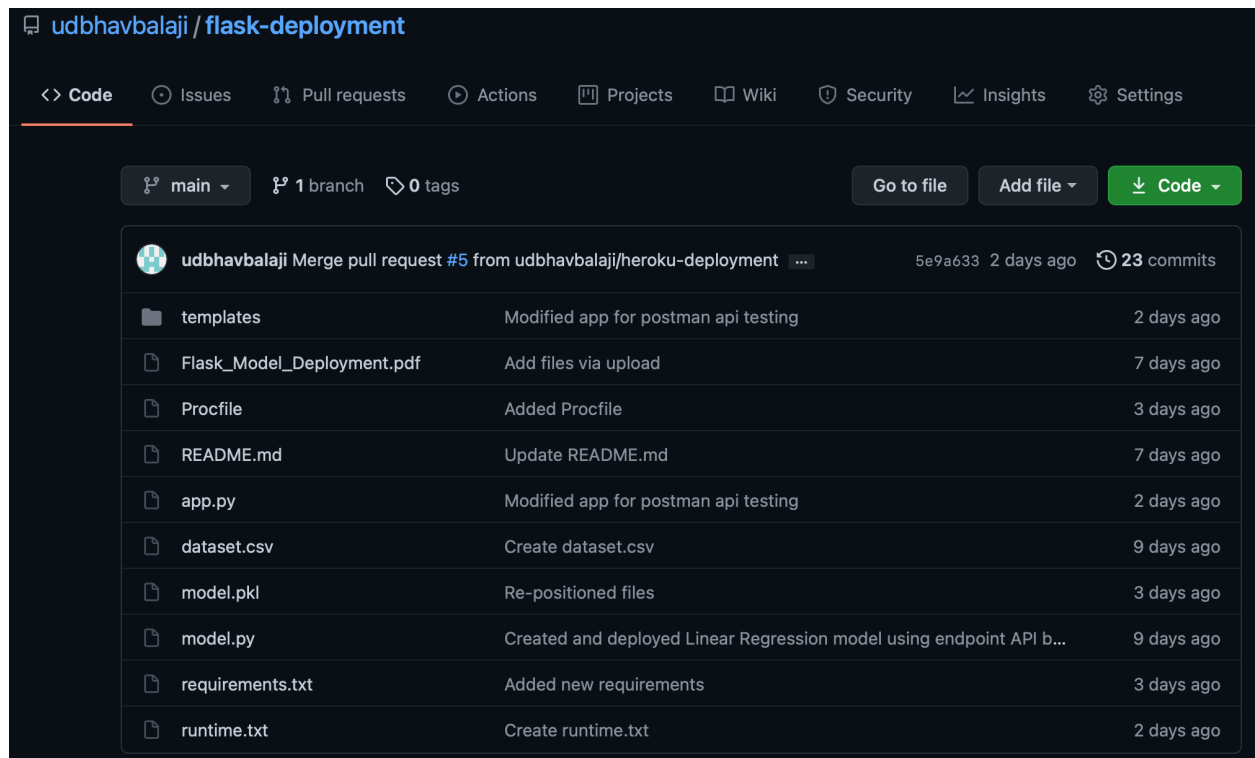
Creating requirements.txt, runtime.txt and Procfile:

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 into our GitHub repository. Once we do this, we can connect the GitHub repo to our Heroku and deploy the model.



The screenshot shows the GitHub interface for the repository 'udbhavbalaji / flask-deployment'. The repository has 1 branch (main) and 0 tags. A list of files is displayed, each with its commit message and the time since the last commit.

| File | Commit Message | Time |
|----------------------------|--|------------|
| templates | Modified app for postman api testing | 2 days ago |
| Flask_Model_Deployment.pdf | Add files via upload | 7 days ago |
| Procfile | Added Procfile | 3 days ago |
| README.md | Update README.md | 7 days ago |
| app.py | Modified app for postman api testing | 2 days ago |
| dataset.csv | Create dataset.csv | 9 days ago |
| model.pkl | Re-positioned files | 3 days ago |
| model.py | Created and deployed Linear Regression model using endpoint API b... | 9 days ago |
| requirements.txt | Added new requirements | 3 days ago |
| runtime.txt | Create runtime.txt | 2 days ago |

Deploying the Main Branch on Heroku:

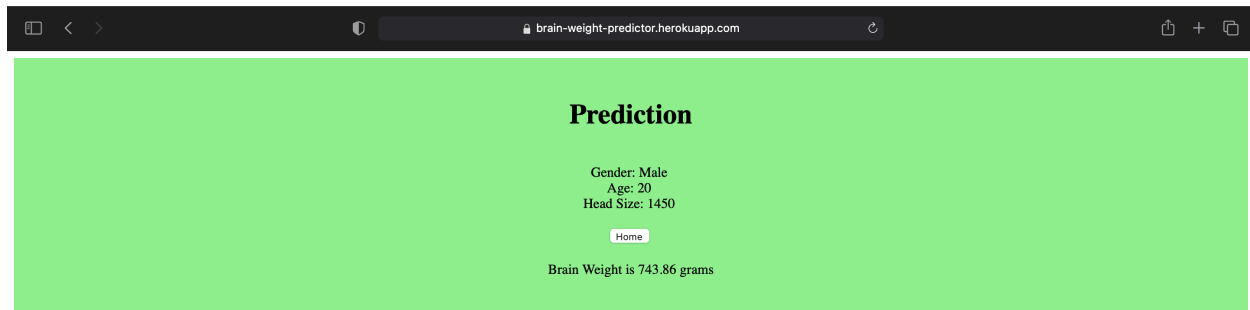
Once our code is committed in GitHub, we can connect it to our Heroku and directly deploy the model. For this deliverable, we will use the manual deploy. This means that we will have to re-deploy the model with each commit on the repo.

The screenshot shows the Heroku dashboard interface. At the top, there's a search bar with the text "Jump to Favorites, Apps, Pipelines, Spaces...". Below this, the "Deployment method" section shows three options: "Heroku Git" (with a purple icon and text "Use Heroku CLI"), "GitHub" (with a black icon, text "Connected", and a green checkmark), and "Container Registry" (with a blue icon and text "Use Heroku CLI"). The "App connected to GitHub" section shows the app is connected to "udbhavbalaji/flask-deployment" by "udbhavbalaji". Below this, the "Manual deploy" section is active, showing a "Deploy a GitHub branch" button. The "Choose a branch to deploy" dropdown is set to "main". The "Deploy Branch" button is visible. Below the dropdown, there are four status indicators with green checkmarks: "Receive code from GitHub", "Build main" (with commit hash 5e9a633d), "Release phase", and "Deploy to Heroku". A message states "Your app was successfully deployed." with a "View" button.

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

The screenshot shows the landing page of the brain-weight-predictor application. The page has a light blue background. At the top, there's a title "Predicting Weight of Brain using Gender, Age and Head Size". Below the title, there are three input fields: "Gender (Male or Female)", "Age (in Years)", and "Head Size (in cm^3)". A "Predict" button is located below the input fields.

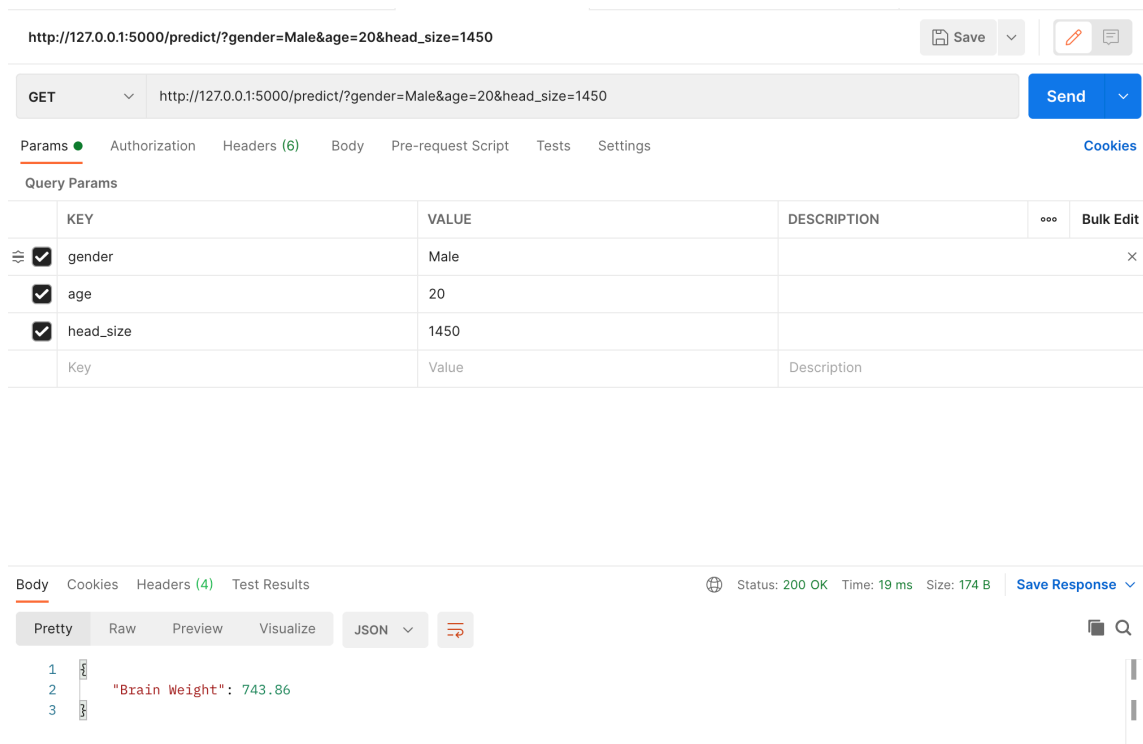
The output page (with example inputs) is shown below.



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 a html webpage as output. The output for this request is shown below.

http://127.0.0.1:5000/predict/?gender=Male&age=20&head_size=1450

GET http://127.0.0.1:5000/predict/?gender=Male&age=20&head_size=1450

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

| KEY | VALUE | DESCRIPTION | ... | Bulk Edit |
|---|-------|-------------|-----|-----------|
| <input checked="" type="checkbox"/> gender | Male | | | |
| <input checked="" type="checkbox"/> age | 20 | | | |
| <input checked="" type="checkbox"/> head_size | 1450 | | | |
| Key | Value | Description | | |

Body Cookies Headers (4) Test Results Status: 200 OK Time: 50 ms Size: 1.11 KB Save Response

Pretty Raw Preview Visualize

Prediction

Gender: Male
Age: 20
Head Size: 1450

[Home](#)

Brain Weight is 743.86 grams

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.

https://brain-weight-predictor.herokuapp.com/predict/?gender=Male&age=20&head_size=1450

Save

GET https://brain-weight-predictor.herokuapp.com/predict/?gender=Male&age=20&head_size=1450 Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

| | KEY | VALUE | DESCRIPTION | ... | Bulk Edit |
|-------------------------------------|-----------|-------|-------------|-----|-----------|
| <input checked="" type="checkbox"/> | gender | Male | | | |
| <input checked="" type="checkbox"/> | age | 20 | | | |
| <input checked="" type="checkbox"/> | head_size | 1450 | | | |
| | Key | Value | Description | | |

Body Cookies Headers (6) Test Results Status: 200 OK Time: 992 ms Size: 1.13 KB Save Response

Pretty Raw Preview Visualize

Prediction

Gender: Male
Age: 20
Head Size: 1450

Home

Brain Weight is 743.86 grams

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.