

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 cm³).

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
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(r"C:\Users\Preeti\Desktop\Data Glacier Internship\Deployment on Flask\dataset.csv")
8  X = np.array(data.iloc[:,0:3])
9  y = np.array(data.iloc[:,3])
10 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
11
12 lm = LinearRegression()
13 lm.fit(X_train, y_train)
14
15 filename = 'model.pkl'
16 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
1  from logging import debug
2  from flask import Flask, request, render_template, jsonify
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('/', methods=['GET', 'POST'])
10 def index():
11     # if request.method == 'GET':
12     #     data = 'Hello World!'
13     return render_template('index.html')
14     # return jsonify({'data': data})
15
16 @app.route('/predict/', methods=['GET'])
17 def predict():
18     gender = request.args.get('gender')
19     age = int(request.args.get('age'))
20     head_size = int(request.args.get('head_size'))
21
22     temp_gender = gender
23     temp_age = age
24
25     if gender.strip().lower() == 'male':
26         gender = 1
27     else:
28         gender = 2
29
30     if age <= 18:
31         age = 2
32     else:
33         age = 1
34
35     test_in = np.array([gender, age, head_size]).reshape(1, -1)
36     pred_weight = model.predict(test_in)
37     output = round(pred_weight[0], 2)
38     return render_template('result.html', gender="Gender: {}".format(temp_gender), age="Age: {}".format(temp_age),
39                             head_size="Head Size: {}".format(head_size), prediction_text="Brain Weight is {} grams".format(output))
40     # return jsonify({'Brain Weight': output})
41
42 if __name__ == "__main__":
43     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
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>
```

Result Page Source Code:

```
templates > result.html > html
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>
```

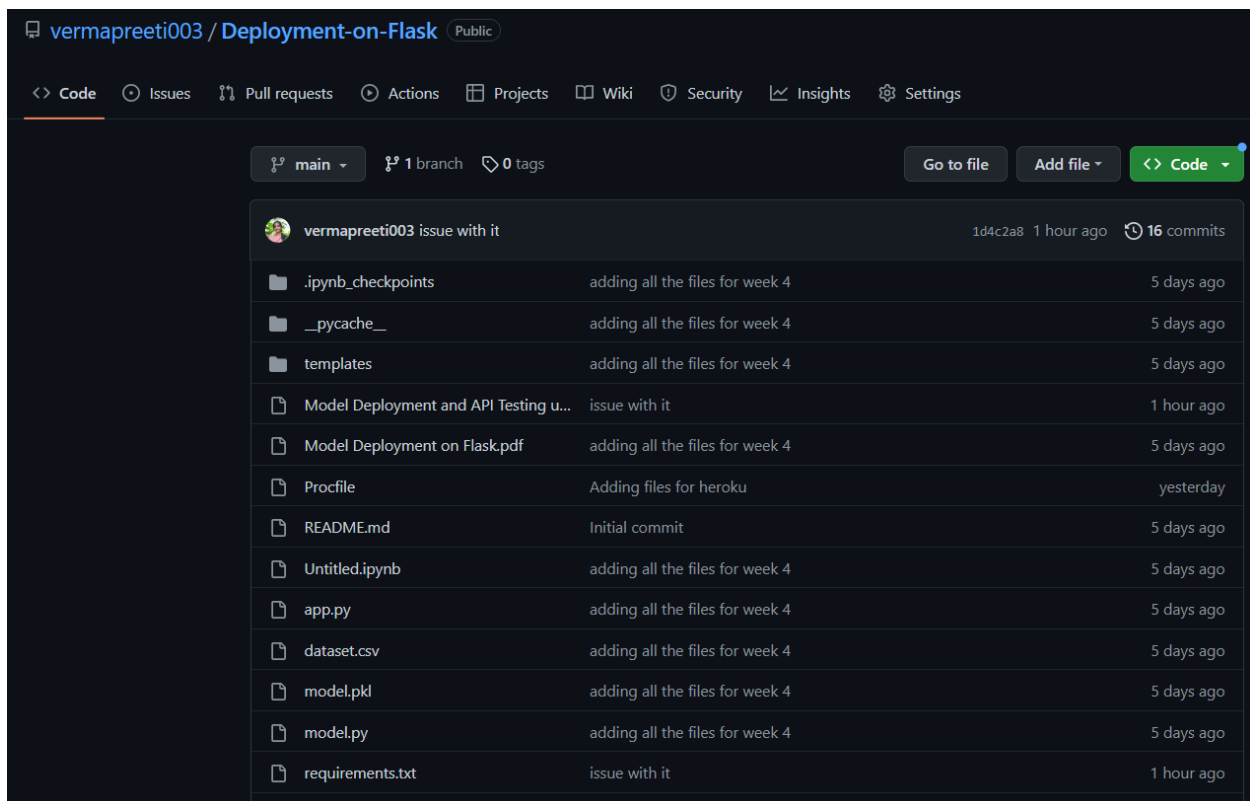
Creating requirements.txt, runtime.txt, and Procfile:

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

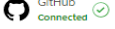



Deploying the Main Branch on Heroku:

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.

Deployment method

 Heroku Git
Use Heroku CLI

 GitHub
Connected

 Container Registry
Use Heroku CLI

App connected to GitHub

Code diffs, manual and auto deploys are available for this app.

Connected to [vermapreet003/Deployment-on-Flask](#) by [vermapreet003](#) [Disconnect...](#)

Releases in the [activity feed](#) link to GitHub to view commit diffs

Automatic deploys

Enables a chosen branch to be automatically deployed to this app.

You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please follow the instructions [here](#).

Enable automatic deploys from GitHub

Every push to the branch you specify here will deploy a new version of this app. **Deploys happen automatically:** be sure that this branch is always in a deployable state and any tests have passed before you push. [Learn more](#)

Choose a branch to deploy

main

☐ Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

Enable Automatic Deploys

Manual deploy

Deploy the current state of a branch to this app.

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. [Learn more](#)

Choose a branch to deploy

main

Deploy Branch

Receive code from GitHub

Build `main` `1d4c2a87`

Release phase

Deploy to Heroku

✓

✓

✓

✓

Your app was successfully deployed.

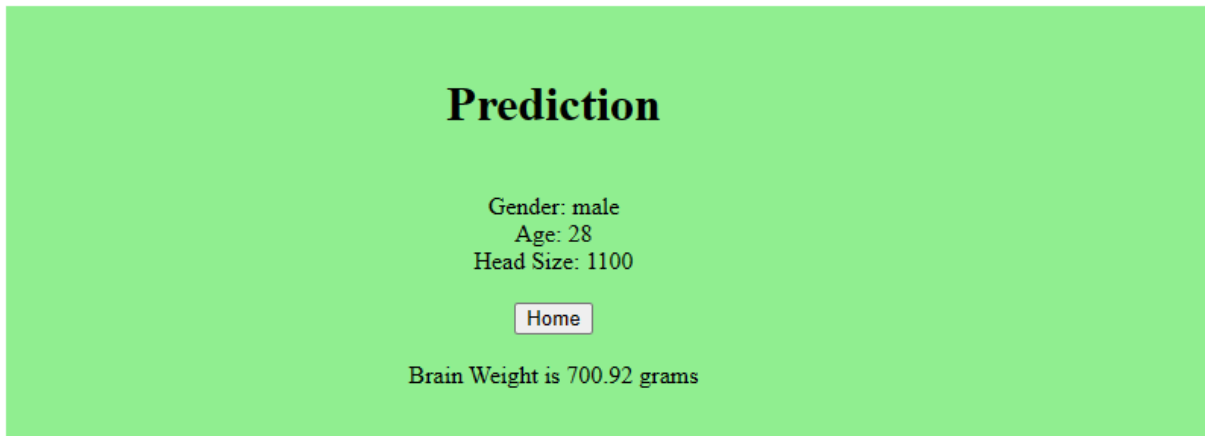
View

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.

Predicting Weight of Brain using Gender, Age and Head Size

Gender (Male or Female)
Age (in Years)
Head Size (in cm^3)
Predict

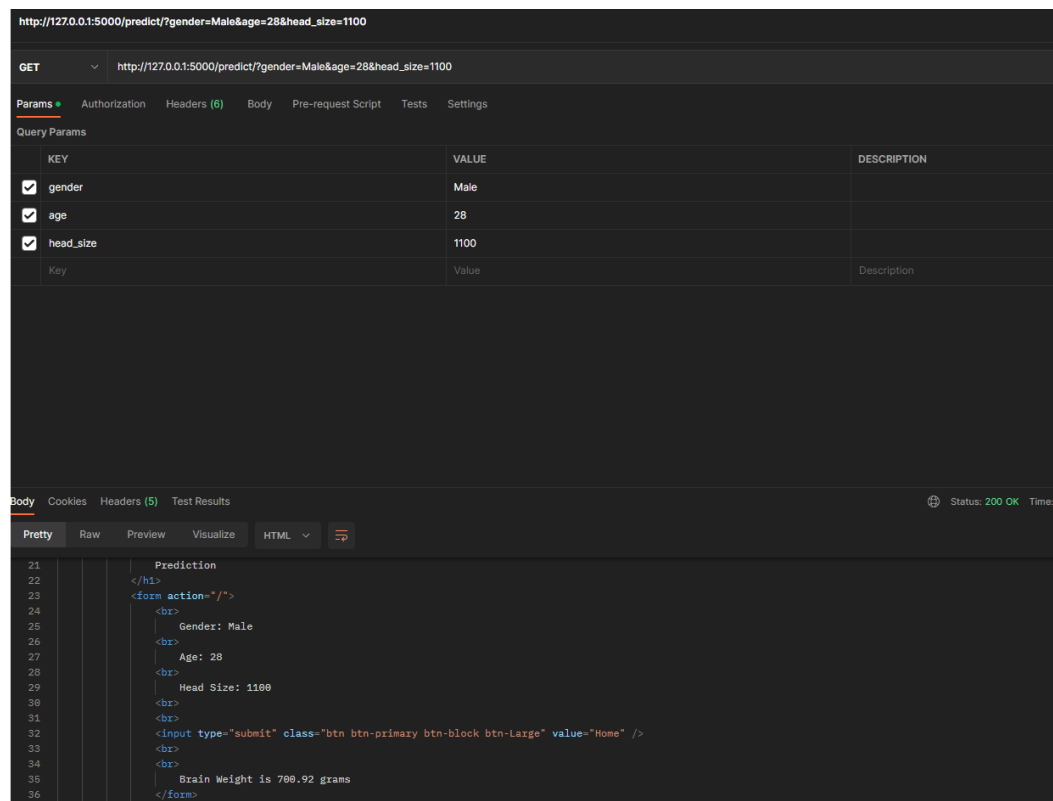
The output page (with example inputs) is shown below.



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.

The screenshot displays a REST client interface at the top, showing a GET request to `http://127.0.0.1:5000/predict/?gender=Male&age=28&head_size=1100`. The 'Params' tab is active, showing query parameters: `gender=Male`, `age=28`, and `head_size=1100`. The status is 200 OK, and the response time is 14 ms. Below the interface, the 'Preview' tab shows an HTML page with a green background. The page features the title 'Prediction' in bold, followed by the input details: 'Gender: Male', 'Age: 28', and 'Head Size: 1100'. A 'Home' button is centered below this text. At the bottom of the page, it states 'Brain Weight is 700.92 grams'.

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> gender	Male	
<input checked="" type="checkbox"/> age	28	
<input checked="" type="checkbox"/> head_size	1100	
Key	Value	Description

Body Cookies Headers (5) Test Results Status: 200 OK Time: 14 ms Size:

Pretty Raw **Preview** Visualize

Prediction

Gender: Male
Age: 28
Head Size: 1100

[Home](#)

Brain Weight is 700.92 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.

GET

https://heroku-brain-weight-predictor.herokuapp.com/predict?gender=Male&age=28&head_size=1100

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Query Params

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	gender	Male	
<input checked="" type="checkbox"/>	age	28	
<input checked="" type="checkbox"/>	head_size	1100	
	Key	Value	Description

Body

Cookies

Headers (6)

Test Results

Status: 200 OK Time: 146 ms

Pretty

Raw

Preview

Visualize

Prediction

Gender: Male
Age: 28
Head Size: 1100

[Home](#)

Brain Weight is 700.92 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.