Data Glacier

Your Deep Learning Partner

Week 5: Cloud and API Deployment

Drug Prediction App

Name: Jeeyeon Shon
Batch: LISUM 10
Date: July 4, 2022
Submitted to: https://github.com/shonjeeyeon/DG_Week_4

Agenda

Summary

Problem Statement

Dataset

Preprocessing

Model Development

Model Serialization

Flask: app.py

index.html

Testing on Localhost

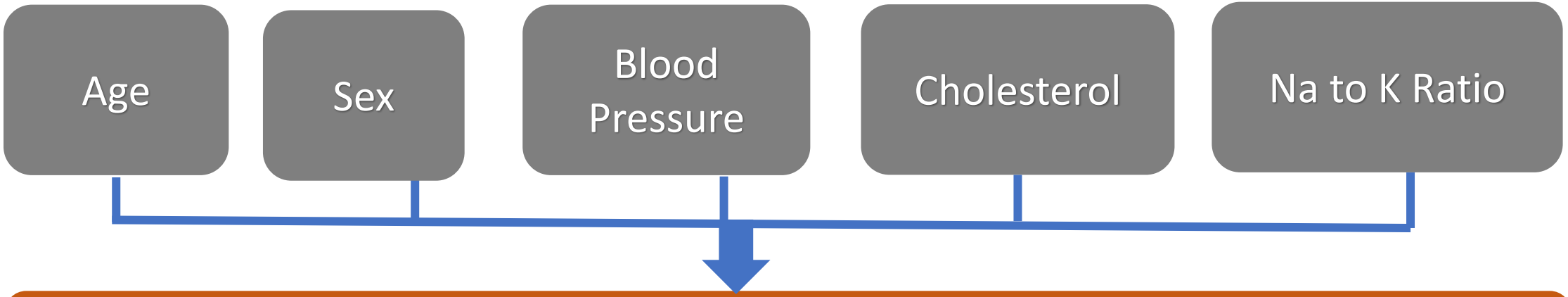
Testing with Postman

Deployment using Heroku

Executive Summary

- The app is developed to:
  - Predict an appropriate drug for a patient, considering:
    - Age
    - Sex
    - Blood Pressure (BP)
    - Cholesterol
    - Sodium to potassium ratio (Na to K ratio) in blood
- Random Forest Classification was used for prediction
  - Accuracy=98.8%
- Flask was used to deploy the model

User accesses the page and
fills out the form (templates/index.html)



Receives input (app.py)

Random Forest Classification (model.pkl)

Prints prediction on templates/index.html (app.py)

Dataset

- The dataset used for training and testing is drug200.csv
  - https://www.kaggle.com/datasets/prathamtripathi/drug-classification
  - Includes 200 observations and 6 features

- Features include:
  - Age, Sex, Blood Pressure, Cholesterol, Na to K ratio, and Drug

- In this assignment, variables will be divided into
  - Target variable (y): Drug
  - Predictive variables (X): The other features

Preview of the Dataset

drug200.csv (6.03 KiB)

Detail Compact Column

About this file

The data set contains various information that effect the predictions like Age, Sex, BP, Cholesterol levels, Na to Potassium Ratio and finally the drug type.

# Age Age of the Patient	A Sex Gender of the patients	A BP Blood Pressure Levels	A Cholesterol Cholesterol Levels	# Na_to_K Sodium to potassium Ration in Blood	A Drug Drug Type
 15 74	M 52% F 48%	HIGH 39% LOW 32% Other (59) 30%	HIGH 52% NORMAL 49%	 6.27 38.2	DrugY 46% drugX 27% Other (55) 28%
23	F	HIGH	HIGH	25.355	DrugY
47	M	LOW	HIGH	13.093	drugC
47	M	LOW	HIGH	10.114	drugC

Captured from: https://www.kaggle.com/datasets/prathamtripathi/drug-classification

Preprocessing

- The dataset was divided into X (predictors) and y (target)

```
#Separate X and y

y=pd.DataFrame(df['Drug'])
print(y.head())

X=df.drop('Drug',axis=1)
print(X.head())
```

- Then each was divided into train and test sets (test size=25%)

```
#Split train vs test data

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                    test_size=0.25,
                                    random_state=0, stratify=y)
```

Model Development

- Random Forest Classifier was used to train and test the data, using parameters recommended by GridSearch CV
- The accuracy of the model was 98.85%

```
#Run RF using the best parameters found from the above search
rf=RandomForestClassifier(n_estimators=250, max_features='auto', max_depth=5, random_state=42, class_weight='balanced', criterion='gini')
rf.fit(X_train, y_train)

cv=RepeatedStratifiedKFold()
accuracy=cross_val_score(rf,X,y,scoring='accuracy',cv=cv,n_jobs=-1)
print('Accuracy:', np.mean(accuracy))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected.
  This is separate from the ipykernel package so we can avoid doing imports until
Accuracy: 0.9885
```

- The complete code can be viewed here

Model Serialization

- After the development, pickle was used to serialize and save the model
- The pickled model was tested again to make sure it is working correctly

```
with open('model.pkl', 'wb') as files:
    pickle.dump(rf, files)
```

```
model = pickle.load(open('model.pkl','rb'))
print(model.predict([[56, 0, 1, 0, 11.349]]))
```

```
['Drug C']
```

Data Glacier

Your Deep Learning Partner

Flask: app.py

- Initialized the new Flask instance using (__name__)
- The location of template folder was given
- The pickled model was loaded

```
import numpy as np
import pickle
from flask import Flask, request, render_template

app = Flask(__name__, template_folder='templates')
model = pickle.load(open("model.pkl", 'rb'))
```

Flask: app.py

- @app.route('/') used to route the app to the defined index (Root/index.html)
- Because three of the inputs are categorical values, dictionaries were made to load the categories to index.html

```python
#Consulted https://github.com/memudualimatou/INSURANCE-CHARGES-WEB-APPLICATION/blob/main for categorical inputs

import numpy as np
import pickle
from flask import Flask, request, render_template

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

@app.route('/', methods=['GET', 'POST'])
def index():
    return render_template(
        'index.html',
        # Dictionaries for categorical data
        data1=[{'sex': 'CLICK TO CHOOSE SEX/GENDER'}, {'sex': 'Female'}, {'sex': 'Male'}],
        data2=[{'bp': 'CLICK TO CHOOSE BLOOD PRESSURE LEVEL'}, {'bp': 'High'}, {'bp': 'Low'}, {'bp': 'Normal'}],
        data3=[{'ch': 'CLICK TO CHOOSE CHOLESTEROL LEVEL'}, {'ch': 'High'}, {'ch': 'Normal'}],
    )
```

Flask: app.py

- Predict function are used to GET inputs and POST the prediction
- The categorical inputs are switched into numbers so model can process

```python
@app.route("/predict", methods=['GET', 'POST'])
def predict():
    input_data = list(request.form.values())

    if input_data[1] == 'Female':
        input_data[1] = 0
    elif input_data[1] == 'Male':
        input_data[1] = 1
    else:
        print(ValueError)
```

Flask: app.py

- The inputs from the form will be converted to an array of numeric values so the model use the data

```
    input_values = [x for x in input_data]
    arr_val = [np.array(input_values)]
    prediction = model.predict(arr_val)
```

- The output will be POSTed on the index.html

```
output = prediction[0]

return render_template('index.html', prediction_text=" The predicted drug for the patient is {}".format(output),
                       data1=[{'sex': 'CLICK TO CHOOSE SEX/GENDER'}, {'sex': 'Female'}, {'sex': 'Male'}],
                       data2=[{'bp': 'CLICK TO CHOOSE BLOOD PRESSURE LEVEL'}, {'bp': 'High'}, {'bp': 'Low'}, {'bp': 'Normal'}],
                       data3=[{'ch': 'CLICK TO CHOOSE YOUR CHOLESTEROL LEVEL'}, {'ch': 'High'}, {'ch': 'Normal'}],
                       )
```

Flask: app.py

- The last lines of the file include how the app will run
- By setting the if statement below, the app will run only when it is directly called (that is, not imported)
- The debug mode will be on when the app runs

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

Flask: index.html

- Index.html file codes for the web page the user will encounter.
- Can set title, body text, and input forms

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Drug Prediction Model</title>
</head>

<body>
<b>Drug Prediction Model</b><br>
Please enter patient information below.<br>
<form action="{{url_for('predict')}}" method="post">
          <input class="Input" type="number" name="age" placeholder="Age" required/><br><br>
```

index.html

- Since this is a simple app, css was integrated into the index.html in the head

```
    <style>

body {background-color: #f5f5f5; align='center'; valign='center'; text-align: center;}
p {color: black; font-family: Arial, Helvetica, sans-serif; text-align: center;}
h1 {color: #008080; font-family: Arial, Helvetica, sans-serif; text-align: center;}
h3 {color: black; font-family: Arial, Helvetica, sans-serif; text-align: center;}

input, select {width: 80%; padding: 12px 20px; margin: 8px 0; display: inline-block;
    border: 1px solid #ccc; border-radius: 4px; box-sizing: border-box;}

button {width: 80%; background-color: #008080; color: white; padding: 14px 20px;
  margin: 8px 0; border: none; border-radius: 4px; cursor: pointer;}

</style>
```

index.html

- Below codes for a form where users can type their age.

(This type of form will be also used for entering Na to K ratio)

```
<form action="{{url_for('predict')}}" method="post">
        <input class="Input" type="number" name="age" placeholder="Age" required/><br><br>
```

- Below is for categorical options. ‘data1’ is defined in app.py.

(This type of form will be used for Sex, Blood Pressure, and Cholesterol)

```
<select name="comp_select" class="Input">
      {% for o in data1 %}
      <option value="{{ o.sex }}">{{ o.sex }}</option>
      {% endfor %}
</select><br><br>
```

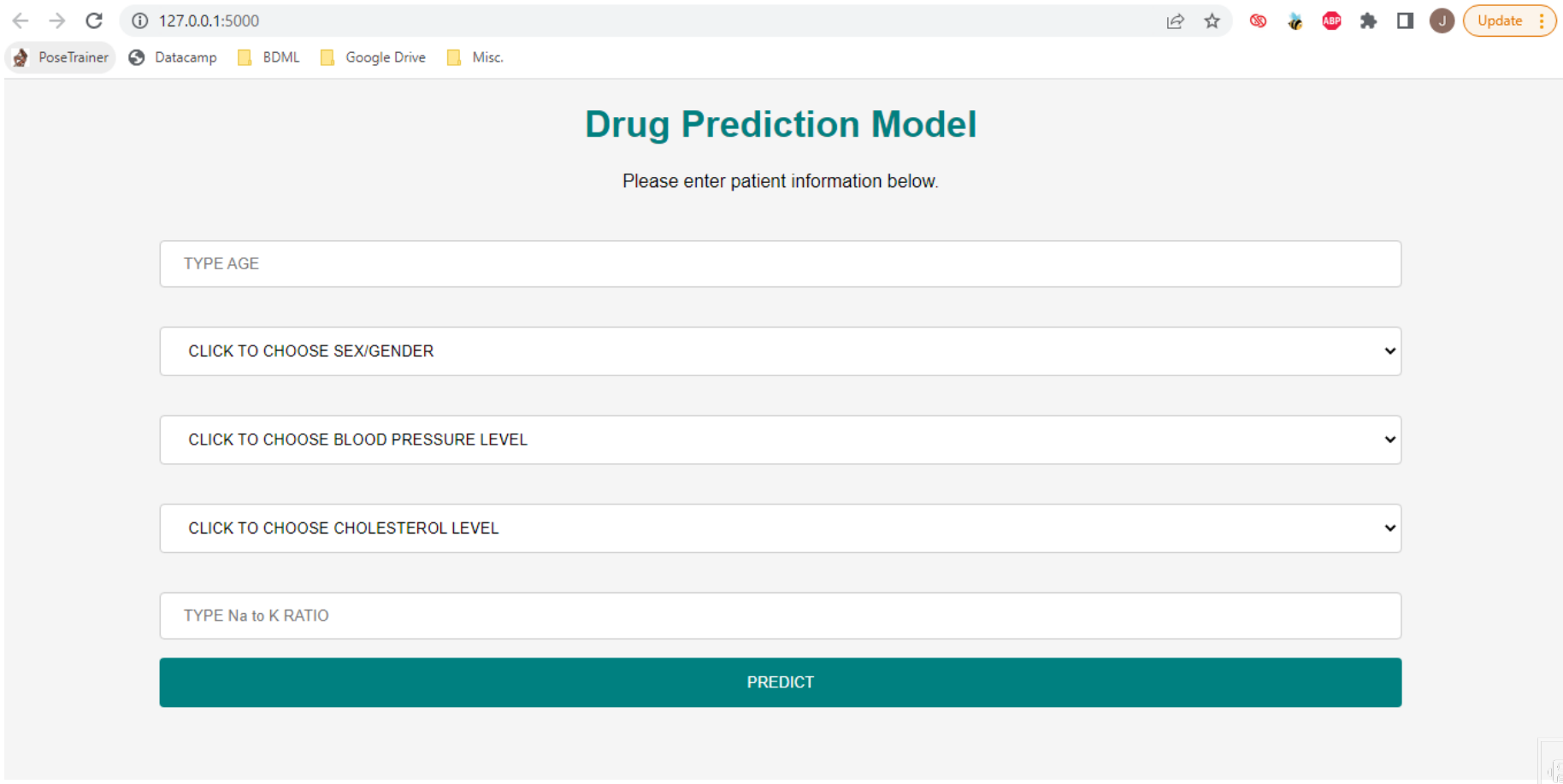
index.html

- After the forms, a button will be placed to have users submit the data.
- The prediction text from app.py will be printed

```
<button type="submit" class="pred-btn"> Predict </button>
      </form><br>

<h3> {{prediction_text}} </h3>
```

Snapshot of the App on Localhost

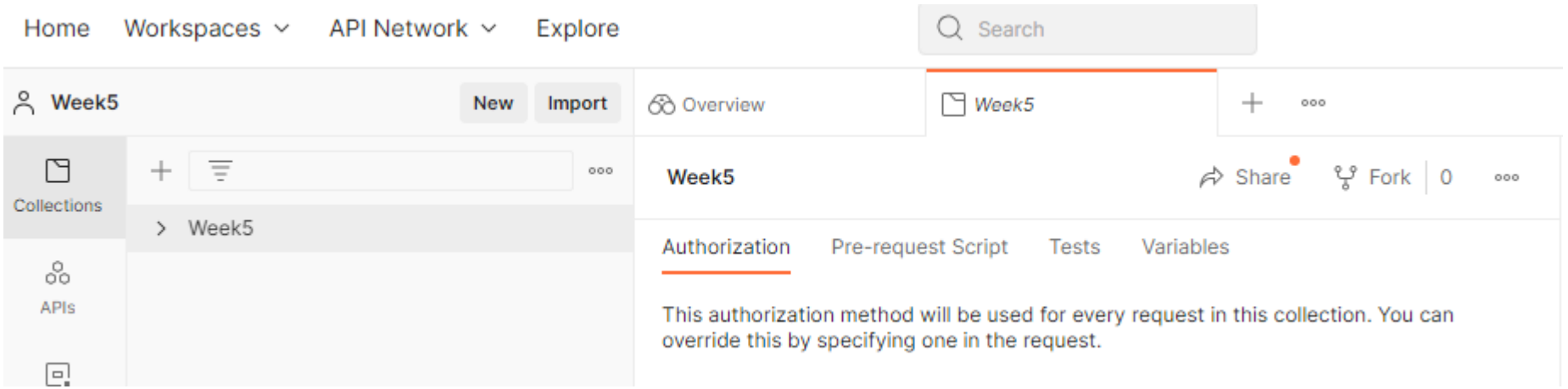


Data Glacier

Your Deep Learning Partner

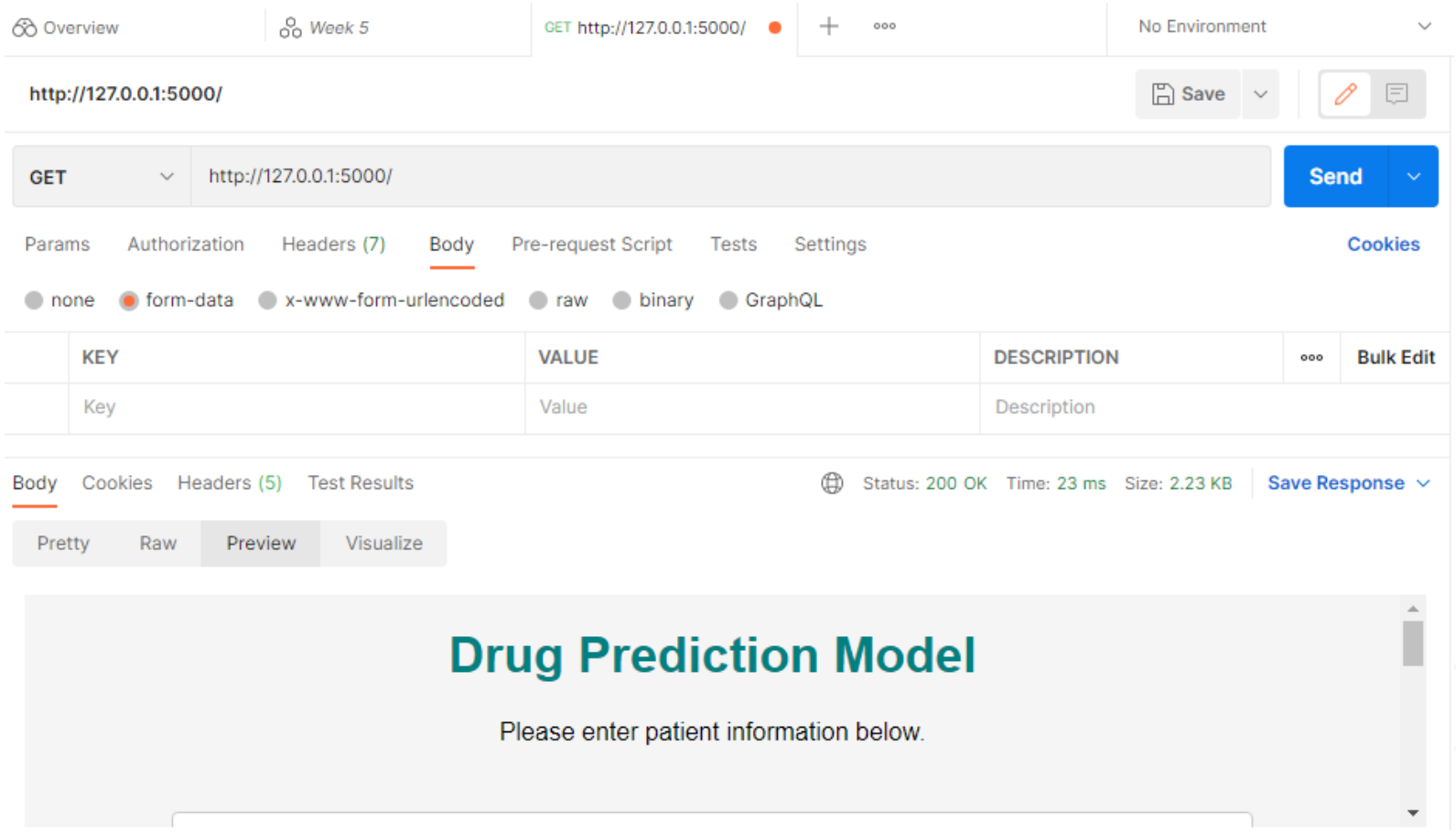
Testing with Postman

- Sign in to Postman and create a new workspace



Testing with Postman

- Create a new collection and use [GET] to test if the app can be retrieved from localhost

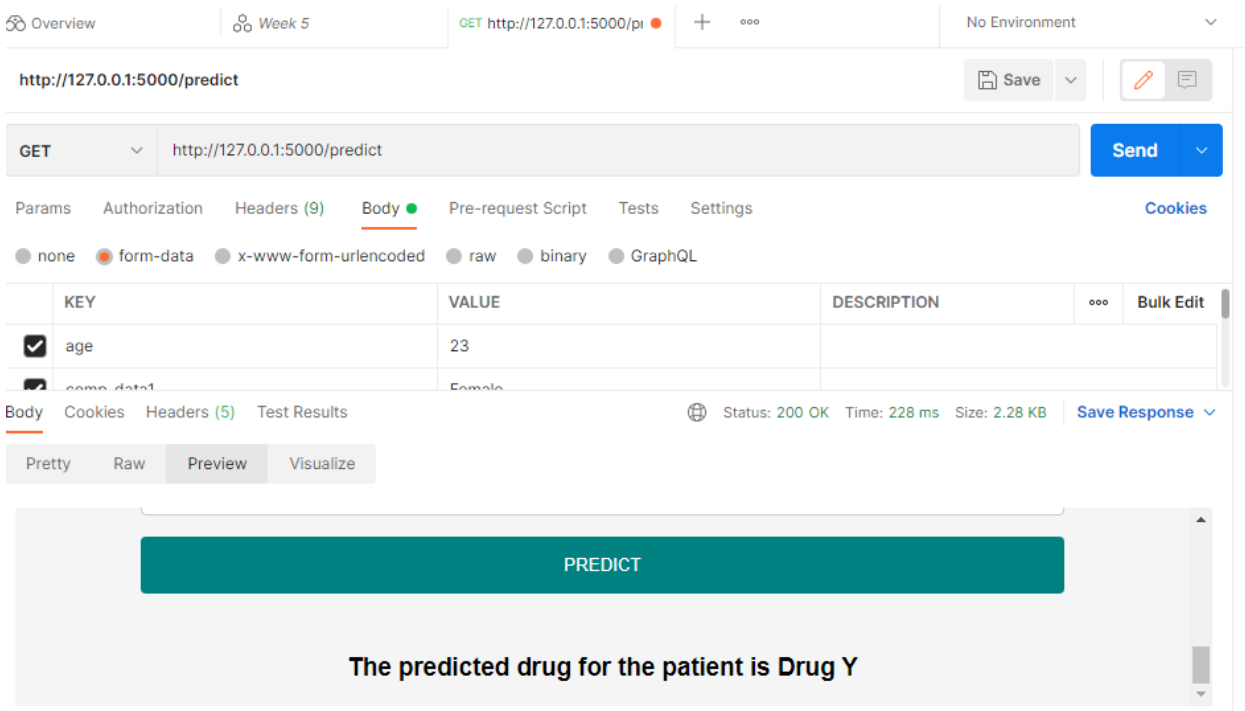


Data Glacier

Your Deep Learning Partner

Testing with Postman

- Use [POST] to test if the user can submit the data via app
- Type keys and values the app requires, and press [Send]
- Preview screen shows the prediction of the model as coded

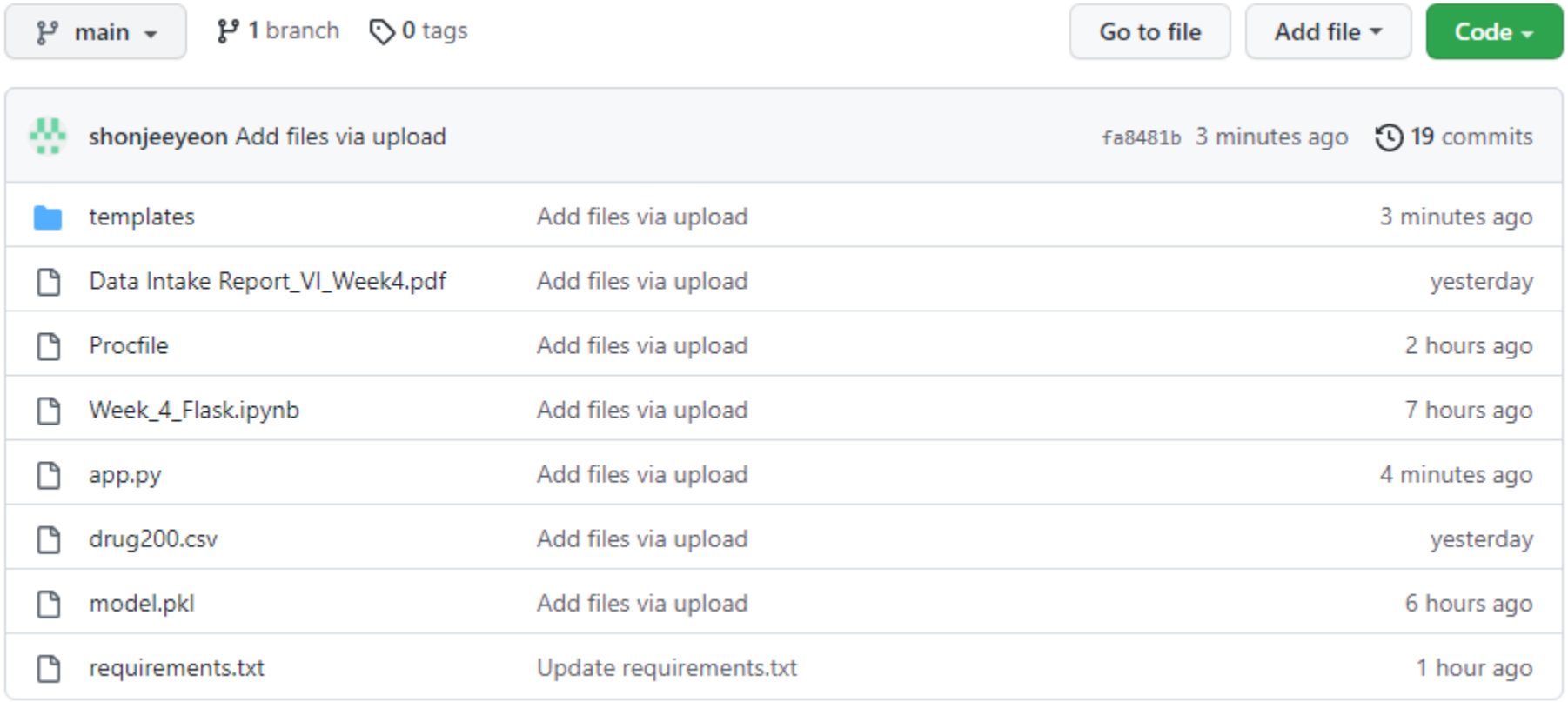


Data Glacier

Your Deep Learning Partner

Deployment Using Heroku

- First build a repository in GitHub
  - requirements.txt file which has a list of necessary modules

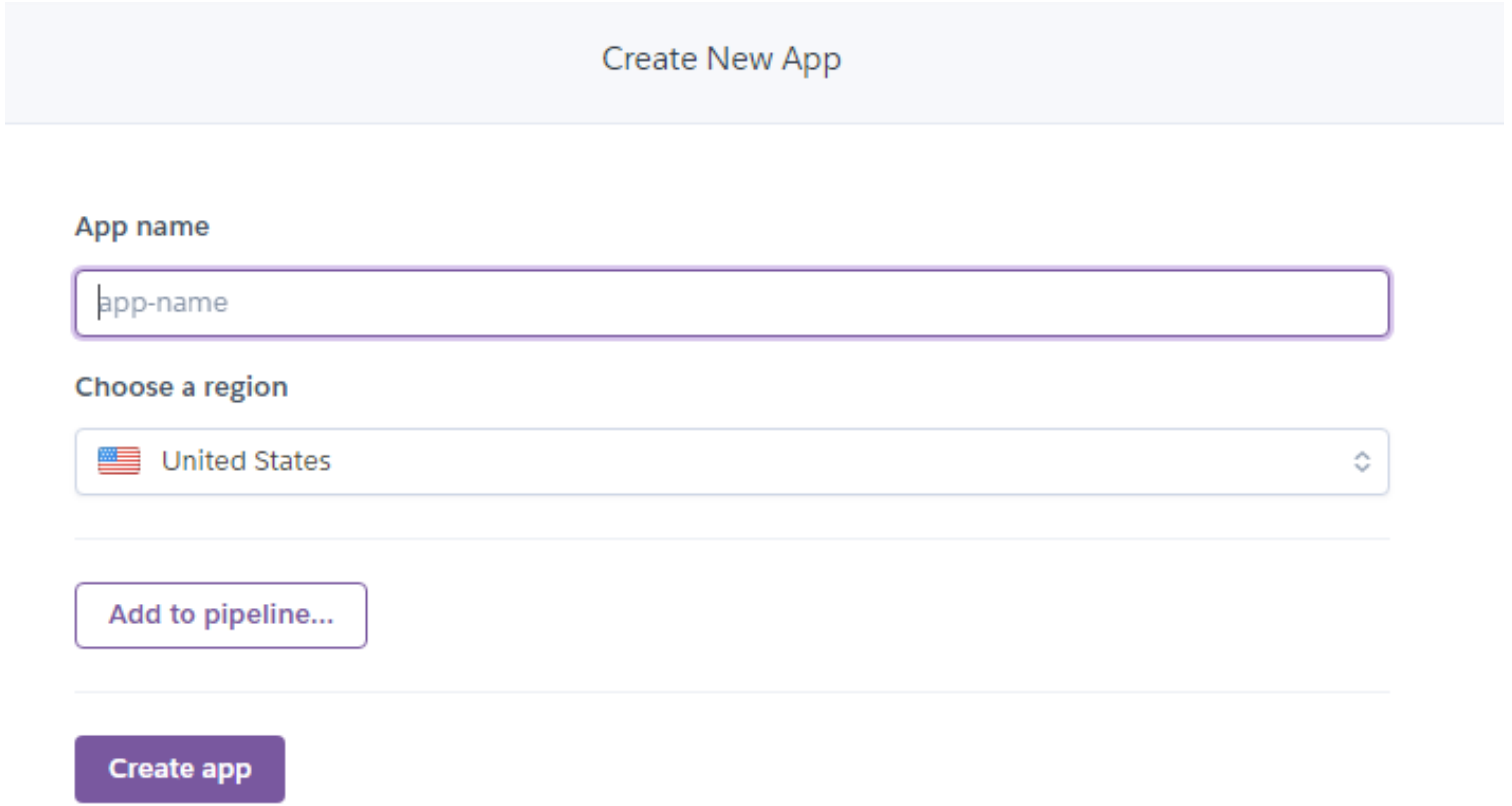


Data Glacier

Your Deep Learning Partner

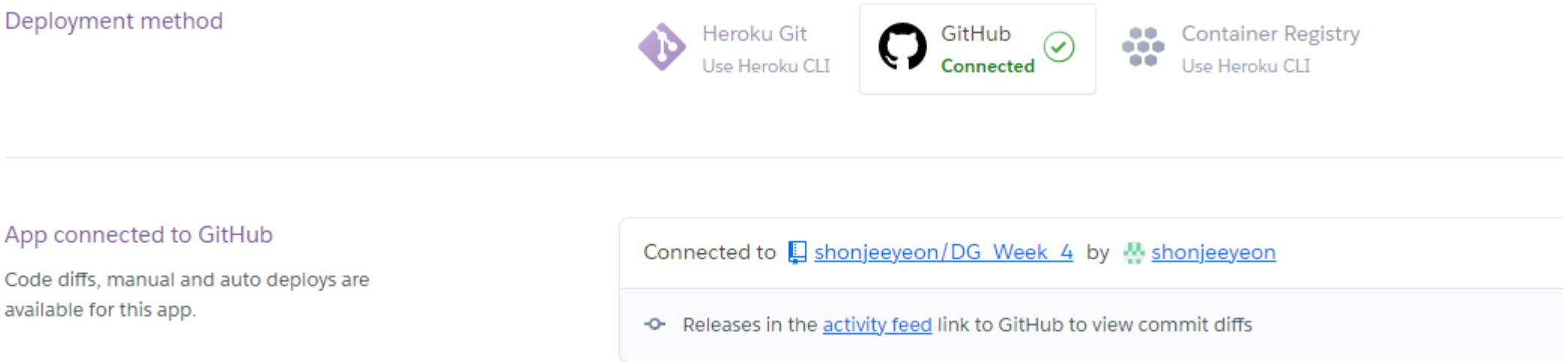
Deployment Using Heroku

- Sign into the Heroku and create an app



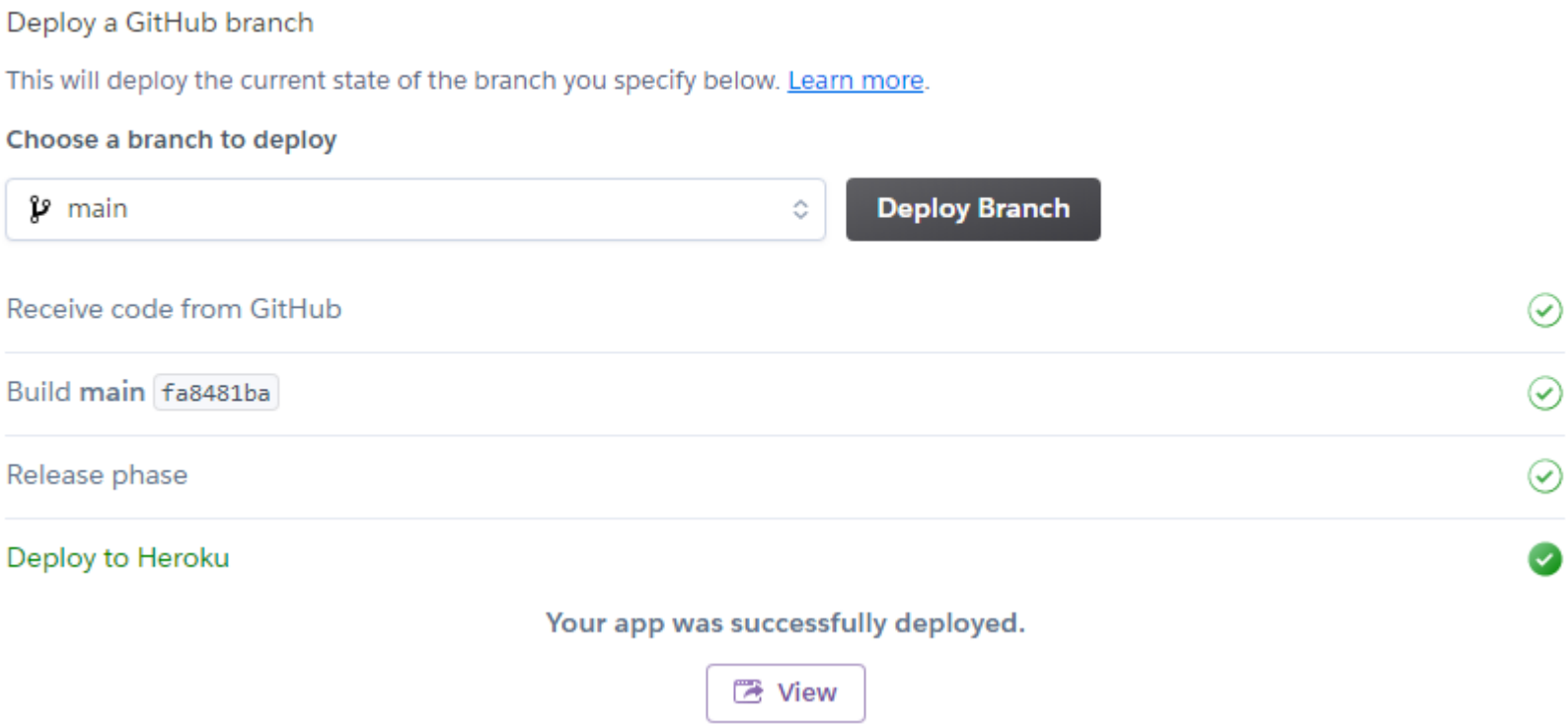
Deployment Using Heroku

- Connect the app to the GitHub repository



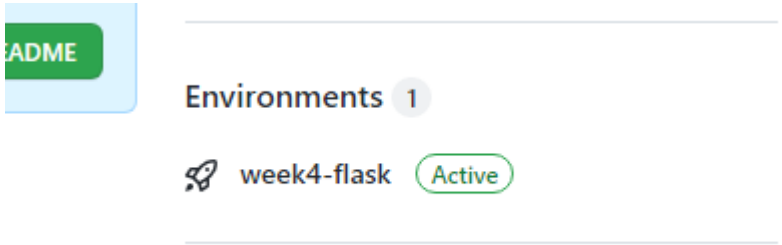
Deployment Using Heroku

- Deploy the GitHub branch



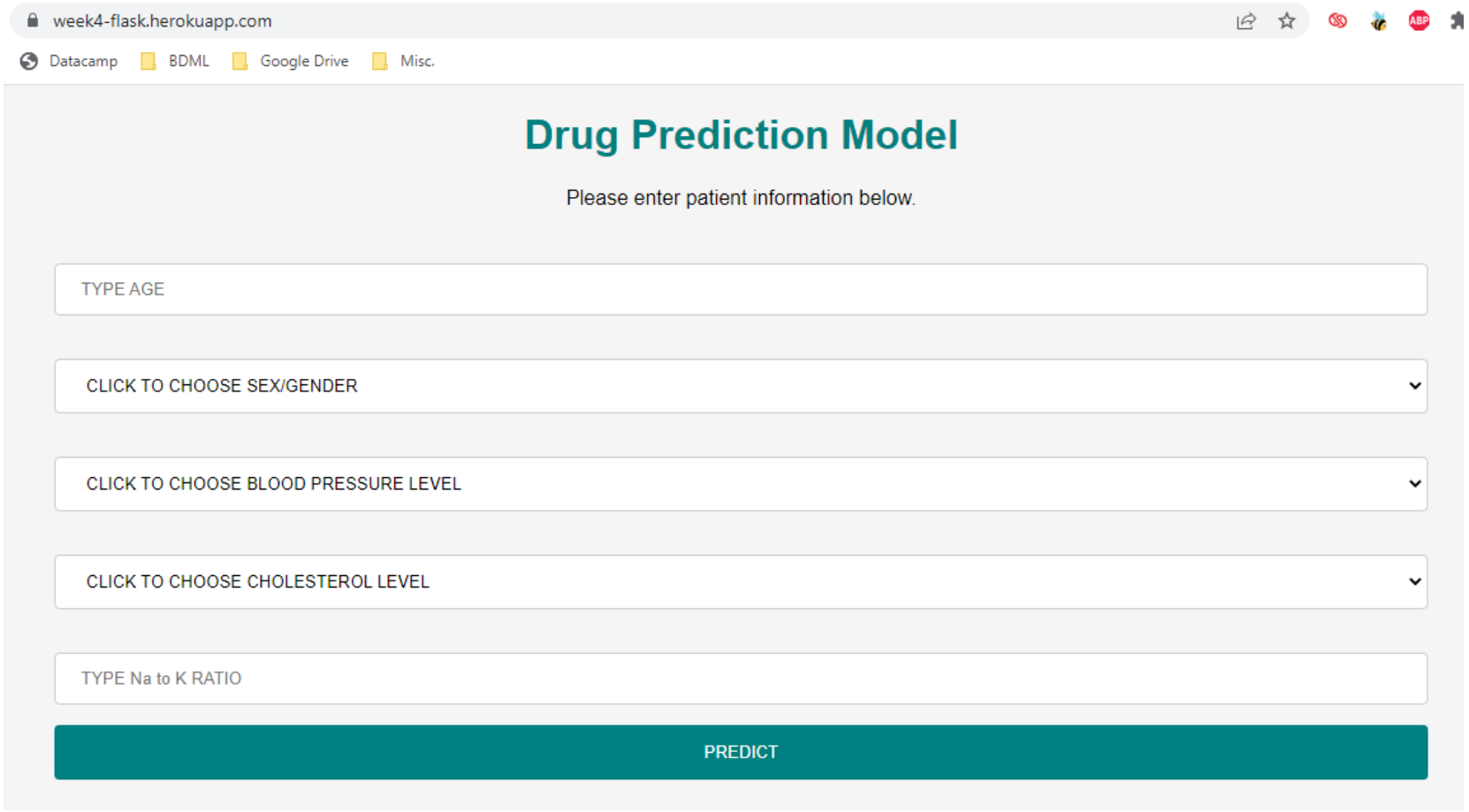
Deployment Using Heroku

- The app can be accessed by clicking the name of the app, which is located bottom right of the GitHub repository page



- **Direct link to the app: https://week4-flask.herokuapp.com/**

Snapshot of the Deployed App



Thank You