Cloud and API deployment

Brennan Clinch
Data Science Intern at Data Glacier
Batch Code: LISUM12
Date: 5 September 2022

Introduction

For this project, we will be deploying a decision tree classifier model using Python's Flask API. We are wanting to build a model that will predict and classify a type of Iris flower based on certain characteristics.

Dataset Information

- Our dataset is a toy dataset included in Python and R called ‘iris’. This dataset has 150 observations and 5 variables.
- The 5 variables are:
- Sepal length: Sepal length in cm
- Sepal width: Sepal width in cm
- Petal length: Petal length in cm
- Petal width: Petal width in cm
- Class: Type of variety of Iris flower. 3 classes: (“Iris-setosa, Iris-virginica, Iris-versicolor”)

Import Libraries and Dataset

- We first import the appropriate libraries in Python along with the Iris dataset from the UCI Machine Learning Repository.

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

```
# Load Iris Dataset
# load through url
url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
attributes = ["sepal_length", "sepal_width", "petal_length", "petal_width", "class"]
iris = pd.read_csv(url, names = attributes)
iris.columns = attributes
iris.head(10)
```

Data Preprocessing

- We split the dataset into 70% for the training set and 20% for the test set. We also set up the appropriate feature columns and target variables. The feature columns are the attributes of the Iris flower such as sepal width, sepal length, petal width, and petal length. The target (response) variable is the type (class) of the Iris flower. It can be Iris-Setosa, Iris-Virginica, or Iris-Versicolor.

```
# Split up data for test and training before doing EDA on training set
iris_train, iris_test = train_test_split(iris, test_size = 0.3, stratify = iris['class'], random_state = 3)
```

```
X_train = iris_train[['sepal_length','sepal_width','petal_length','petal_width']]
Y_train = iris_train['class']

X_test = iris_test[['sepal_length','sepal_width','petal_length','petal_width']]
Y_test = iris_test['class']
```

Model Build

- After preprocessing the data, we build a machine learning model to classify the Iris flower based on the feature variables. For this model, we will use a Classification Tree since we want to classify the type of Iris flower.
- Accuracy looks pretty good for the training and test sets.

```
# Classification Tree
from sklearn.tree import DecisionTreeClassifier, plot_tree
DT_model = DecisionTreeClassifier(max_depth = 3, random_state = 5)
DT_model = model.fit(X_train,Y_train)
model_predict_train = DT_model.predict(X_train)
model_predict = DT_model.predict(X_test)
print('Accuracy of Test Set:',"{:.3f}".format(metrics.accuracy_score(model_predict,Y_test)))
print('Accuracy of Train Set:',"{:.3f}".format(metrics.accuracy_score(model_predict_train,Y_train)))
```

```
Accuracy of Test Set: 0.911
Accuracy of Train Set: 0.981
```

Save Model

- After building our model, we use the pickle serialization library to save it.

```
pickle.dump(DT_model, open('Treemodel.pickle', 'wb'))
```

Creating a web application

- Next, we will create a web application of our model using Python's Flask API. The way the information and files are stored is shown below. We will discuss the different files next and their purpose for the app.

App.py
templates/
    index.html
static/
    css/
        main.css
model/
    Treemodel.pickle

App.py

- This file contains the main code to be executed that will be ran in the Flask web app. The serialized model was then unserialized here to be used in production in Flask.

App.py

```
#!/usr/bin/env python
# coding: utf-8

# In[7]:


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

app = Flask(__name__,template_folder = 'templates')
model = pickle.load(open('Model/Treemodel.pickle','rb'))

@app.route('/', methods=['GET','POST'])
def home():
    return render_template('index.html')
@app.route('/predict',methods =['POST'])
def predict():
    '''
    For rendering results on html GUI
    '''
    float_features = [float(x) for x in request.form.values()]
    final_features = [np.array(float_features)]
    prediction = model.predict(final_features)
    output = prediction[0]
    return render_template('index.html', prediction_text = 'Class should be {}'.format(output))
if __name__ == "__main__":
    app.run(port = 5000, debug=True)
```

index.html

- This file stores the information necessary to write the application created with Flask, such as text and font along with other features.

index.html

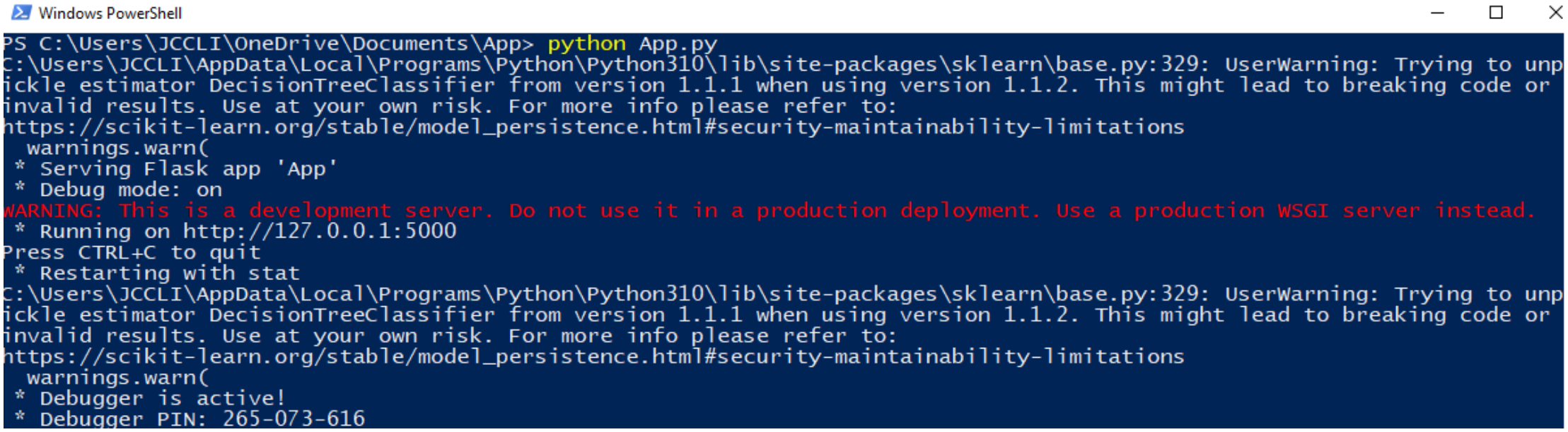
```
<!DOCTYPE html>
<html>
  <head>

    <link rel="stylesheet" type = "text/css" href="{{ url_for('static', filename='css/main.css') }}">
  </head>
    <body>
        <div class="container">
            <h1>Iris Classification</h1>
            <p>This model uses information from the iris dataset to classify the class of iris plant based on several feature characteristics.
            </p>
            <div class='Login'>
              <form action="{{ url_for('predict') }}"method="post">
                  <input type="text" class='form-control' name="sepal_length" placeholder="Sepal Length (cm)" required="required" /><br>
                  <input type="text" class='form-control' name="sepal_width" placeholder="Sepal Width (cm)" required="required" /><br>
                  <input type="text" class='form-control' name="petal_length" placeholder="Petal Length (cm)" required="required" /><br>
                  <input type="text" class='form-control' name="petal_width" placeholder="Petal Width (cm)" required="required" /><br>


                  <button type="submit" class='btn btn-primary btn-block btn-Large'>Predict</button>
              </form>
            </div>
            <br>
            <br>
            <h1>{{ prediction_text }}</h1>
        </div>
    </body>
</html
```

main.css

- This file is used to create the styling of the web app (color, theming, etc).

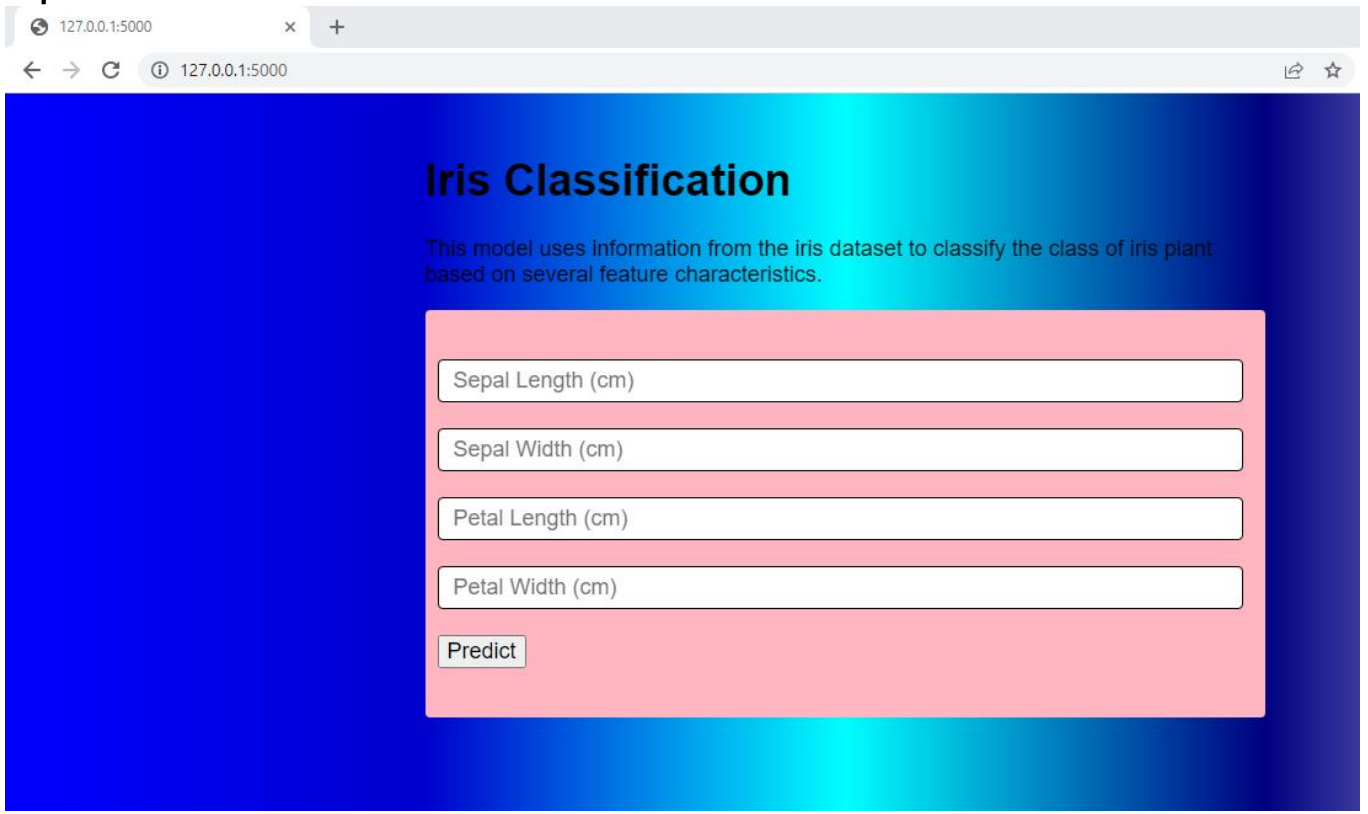
Running the web application

- We are now ready to run the app. To do this, we open a command prompt (I used PowerShell) or run the app by double clicking on the python file App.py. To get to our app, we must go to the url specified in the command prompt (http://127.0.0.1:5000).



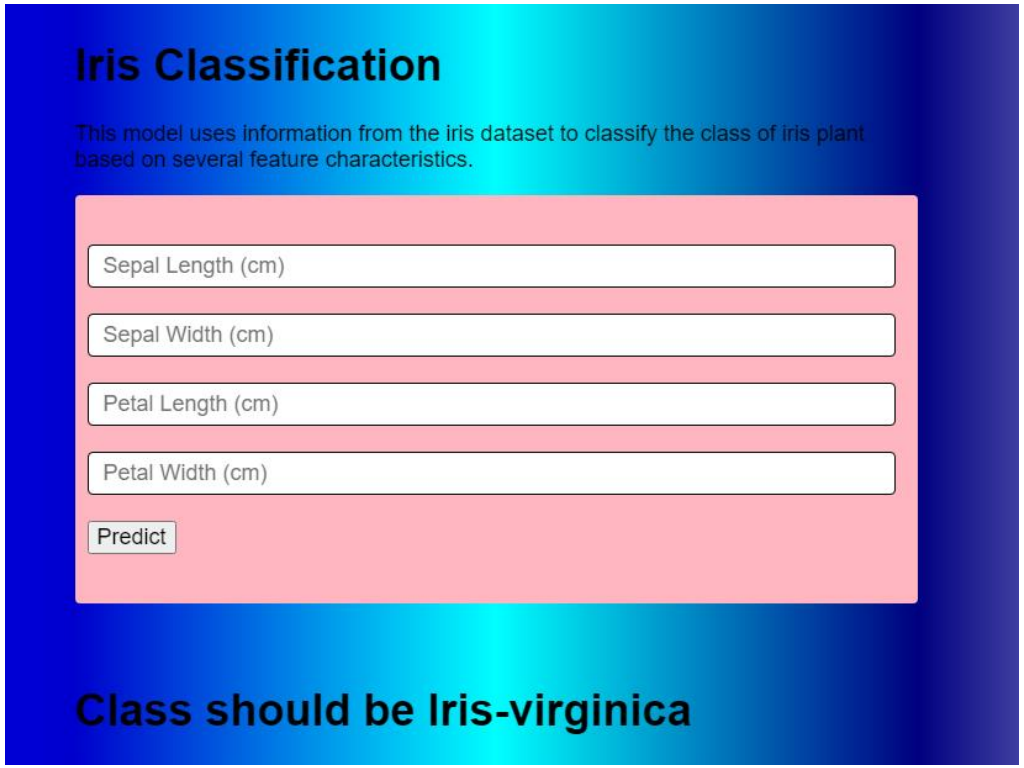
Running the web application

- After going to the url specified, our app page should look like this. We can input numeric values (decimals too) into the 4 boxes and then click on the predict button to classify the type of Iris based on our inputs.



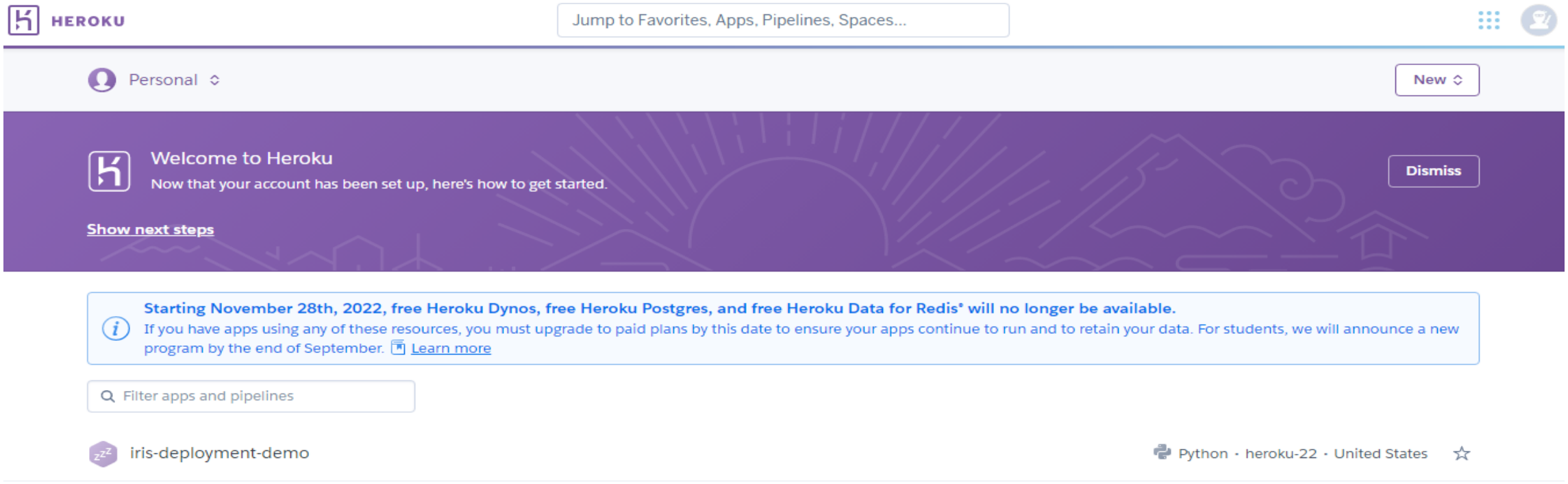
Running the web application

- Let's do a sample prediction. Let Sepal Length = 0.4, Sepal Width = 1.5, Petal Length = 3.3, and Petal Width = 1.7. After pushing the predict button, our classified Iris flower is Iris-virginica. We tested other values and the model works successfully by classifying other Iris flowers based on what we input.



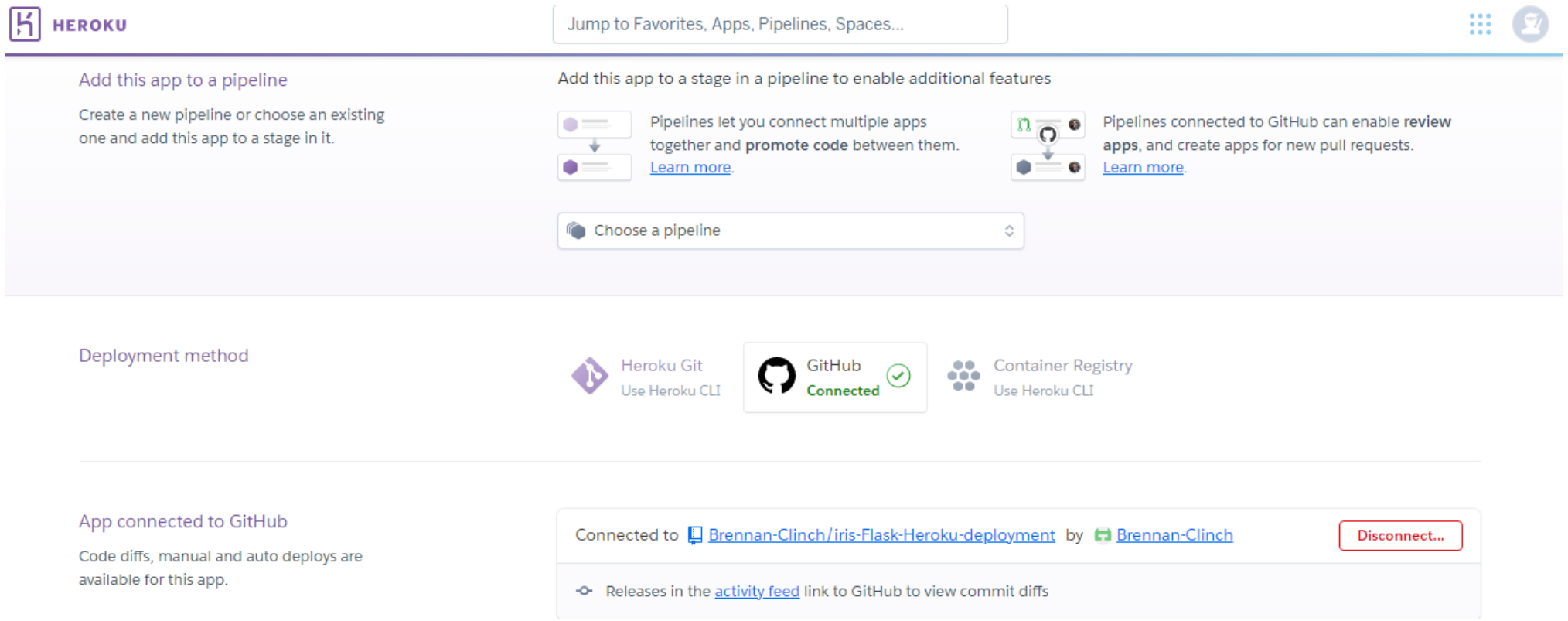
Deploying the app on Heroku (cloud)

- We are now ready to deploy our app to the cloud. We will be using the free open source cloud platform Heroku
- Before we can deploy our model, we must have the following files installed in github.
  - **Requirements.txt**: This is a text file listing all the files and packages needed to run our application.
  - **Procfile:** This file tells Heroku commands to be ran when we start the app. The Gunicorn command connects the Flask app to the HTTP server.
  - We name our app Iris-deployment-demo.



Deploying the app on Heroku (cloud)

We now configure our app by connecting to github.



Deploying the app on Heroku (cloud)

- After connecting to github, go to the option named “Manual Deploy” and click “Deploy Branch”. If the github contains all the necessary files (requirements.txt, and Procfile) then the app should run successfully.