Project: Cloud and API deployment

Name: Gordon Poon
Batch Code: LISUM23
Submission date: 28/07/2023
Submitted to: Data Glacier

Table of contents

1.Introduction

In this project, we will be deploying the machine learning model (logistic regression) using Flask.

Below is a general overview of what the workflow is like:

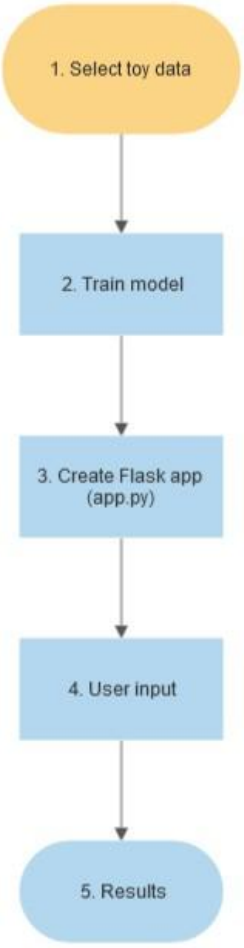


Table 1.1: Flask deployment workflow

To do this, I have selected the iris dataset for data preparation, which will later be used to train the model. In addition to this, the web application setup and website setup will be done for the purpose of flask deployment.

2. Dataset

The dataset used was simply just the iris dataset that I took from the kaggle website. Simply put, the dataset has 4 columns to quantify the attributes of the iris with 3 distinct species of iris. Below is a snapshot of the dataset taken from model.ipynb:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Table 2.1: Iris Dataset

3. Model Building

3.1 Data processing

Firstly, the required libraries were imported.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import joblib
import pickle
```

In this process, 80% percent of the data was used for training the model and 20% was used for testing the model. In addition, a number from 0 to 2 was assigned to each iris species so we can map them later:

```
# Selecting columns from dataset for inputs and outputs
X = data[['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']].values  # Features (input data)
y = data['Species']  # Target variable (output)
```

```
mapping = {'Iris-setosa': 0,'Iris-versicolor': 1,'Iris-virginica':2}
data["Species"] = data["Species"].map(mapping)
```

```
# 80 percent training data, 20 percent testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

3.2 Logistic regression

Afterwards, the training data was fed into the Logistic Regression model, which we implemented from sklearn. In addition, the model score was found to be 0.975, indicating a good fit.

```
# Model initialisation
model = LogisticRegression()
```

```
# Model fit
model.fit(X_train, y_train)
```

```
LogisticRegression()
```

```
model.score(X_train,y_train)
```

```
0.975
```

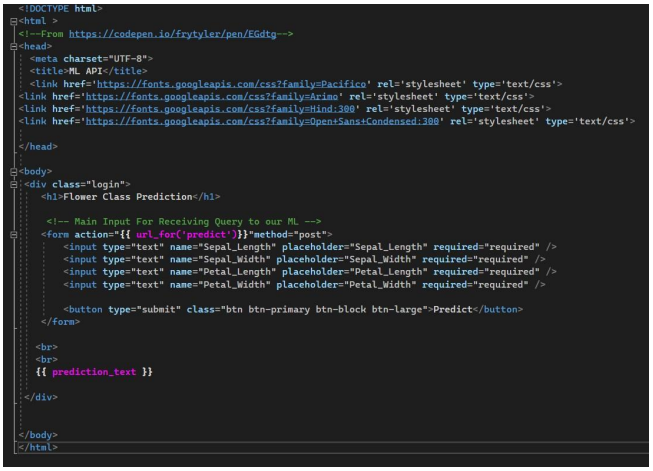
Finally, the model was saved using pickle as a file called ‘model.pkl’

```
# Creating pickle file
with open("model.pkl", "wb") as file:
    pickle.dump(model, file)
```

4. Web application development

4.1. Creating index.html

To create the Flask web application, ‘index.html’ was created as the html template. It serves as the user interface for the user where they can input measurements for the various attributes for each flower as shown from the <form> tags, which would be fed into the model for a prediction using a button as shown from the <button> tag.



At the very bottom, The {{prediction_text}} placeholder will be filled with the prediction result returned by the Flask app. This result will be displayed to users, showing the predicted flower species. Last but not least, it was also saved in the templates directory which allows for the Flask application to find it.

4.2. Creating app.py

‘app.py’ essentially is the python script responsible for the creation of the Flask application itself. Its purpose is primarily to define routes and load the model for the application to be used for the user. To describe how the file was created, the following provides a brief rundown:

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

# Create flask app
flask_app = Flask(__name__)
model = pickle.load(open("model.pkl", "rb"))

@flask_app.route("/")
def Home():
    return render_template("index.html")

@flask_app.route("/predict", methods = ["POST"])
def predict():
    float_features = [float(x) for x in request.form.values()]
    features = [np.array(float_features)]
    prediction = model.predict(features)
    return render_template("index.html", prediction_text = "The flower species is {}".format(prediction))

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

1. First flask_app was created by using Flask which can be stored as a ‘variable’
2. Then the pickle file ‘mode.pkl’ was loaded
3. Subsequently, two routes were created

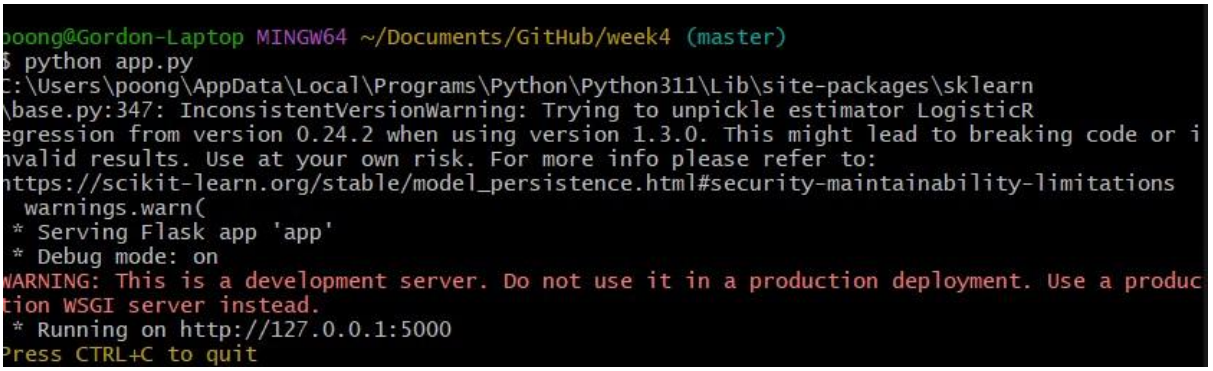
a. The ‘/’ routes maps to the ‘Home’ function which leads to the homepage of ‘index.html’
b. The ‘/predict’ maps to the ‘predict’ function which leads to the predictions

4. Finally, ‘flask_app.run(debug=True)’ renders the app to run in debug mode, which will allow for the developer to spot errors.

5. Flask deployment

5.1 Full rundown

To run the API, we execute ‘app.py’ on terminal as shown:



This then provides a link http://127.0.0.1:5000 .

Upon clicking on the link, we are directed to a website as shown below:

Furthermore, the user is then able to input measurements for predictions:

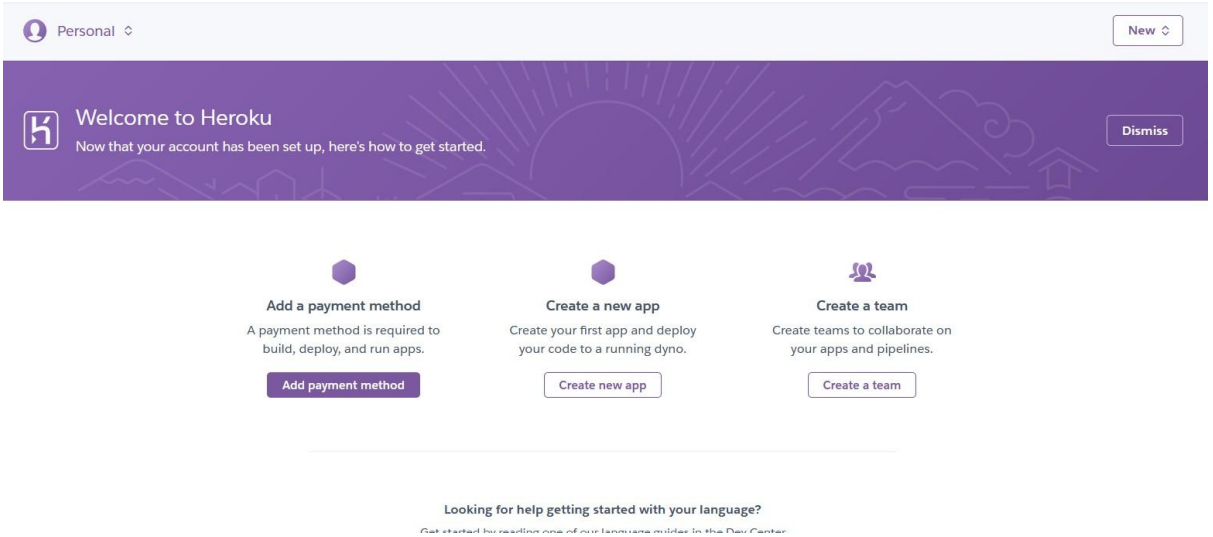
By clicking on the predict button, the user is then directed to a page with the predicted species of flower:

6. Heroku

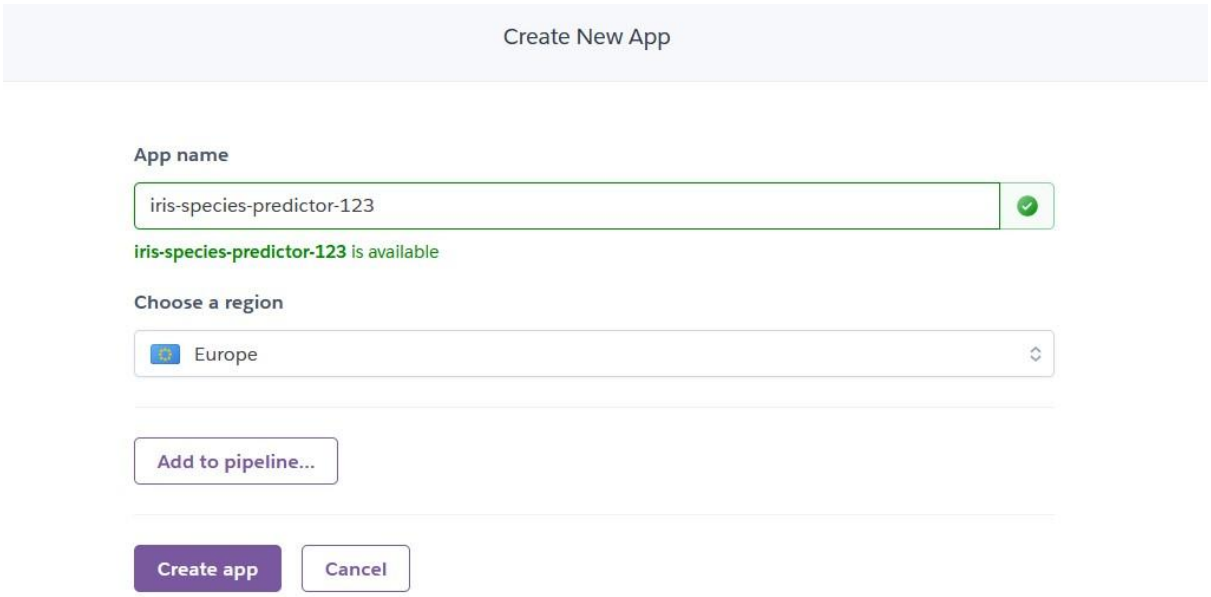
6.1 Heroku deployment

Before deployment on Heroku, connect it to your github repository. Also, create ‘requirements.txt’, listing the necessary packages and the version within.

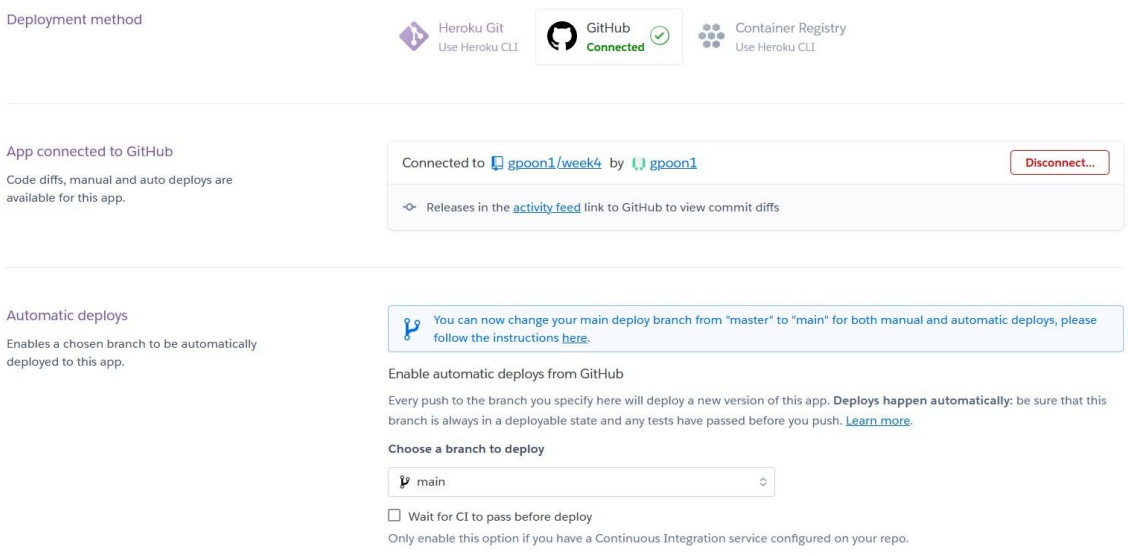
Step 1. Create a new app on the Heroku dashboard



Step 2. Give app a name



Step 3. Connect app to github repository



Step 4. Successful deploy