Heroku Deployment Report

Name: Nonye Nweke

Batch Code: LISUM36

Submission Date: 2023-09-04

Submitted To: Recipient Name

Heroku Deployment Report

Step 1: Loading the Data

The Iris dataset is loaded using the sklearn library. The dataset contains 150 samples with 4 features: sepal length, sepal width, petal length, and petal width.

```
3]: from sklearn.datasets import load_iris
    from sklearn.ensemble import RandomForestClassifier
    import pickle

    # Load the Iris dataset
    iris = load_iris()
    X, y = iris.data, iris.target
```

Heroku Deployment Report

Step 2: Data splitting

The dataset is split into training and testing sets using `train_test_split` from sklearn.

```
# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Train the model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Heroku Deployment Report

Step 3: Training the Model

Logistic Regression model is created using `LogisticRegression` from sklearn and trained on the training data.

```
# Train the model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X, y)
```

Heroku Deployment Report

Step 4: Saving the Model

The trained model is saved using the `pickle` for later use in predictions during deployment.

```
# Save the model to a file
with open('iris_model.pkl', 'wb') as file:
    pickle.dump(model, file)
```

Heroku Deployment Report

Step 5: Creating Flask Application

Flask web application which includes an endpoint `/predict` is created to receive input features and return predictions using the trained model.

```python
from flask import Flask, request, render_template
import pickle

app = Flask(__name__)

# Load the model
with open('iris_model.pkl', 'rb') as file:
    model = pickle.load(file)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    sepal_length = float(request.form['sepal_length'])
    sepal_width = float(request.form['sepal_width'])
    petal_length = float(request.form['petal_length'])
    petal_width = float(request.form['petal_width'])

    prediction = model.predict([[sepal_length, sepal_width, petal_length, petal_width]])

    return render_template('index.html', prediction=prediction[0])

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

Heroku Deployment Report

Step 6: Setting Up a Virtual Environment

A virtual environment is created using the command `python -m venv venv`. The virtual environment is activated using `venv\Scripts\activate` on Windows.

```
home = C:\Users\previ\AppData\Local\Programs\Python\Python312
include-system-site-packages = false
version = 3.12.4
executable = C:\Users\previ\AppData\Local\Programs\Python\Python312\python.exe
command = C:\Users\previ\AppData\Local\Programs\Python\Python312\python.exe -m venv
C:\Users\previ\OneDrive\Desktop\iris-heroku-deployment\venv
```

Step 7: Creating Required Files

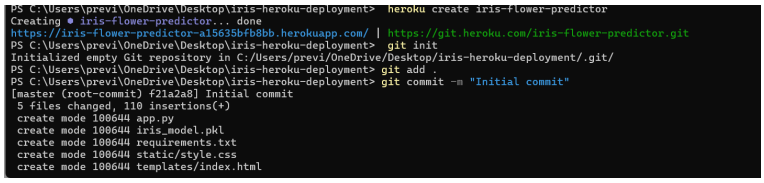
A `requirements.txt` file with necessary packages is created. as well as `Procfile` that specifies the command to run the application using Gunicorn.

```
web: gunicorn app:app
```

Heroku Deployment Report

Step 8: Initializing Git and Commit Files

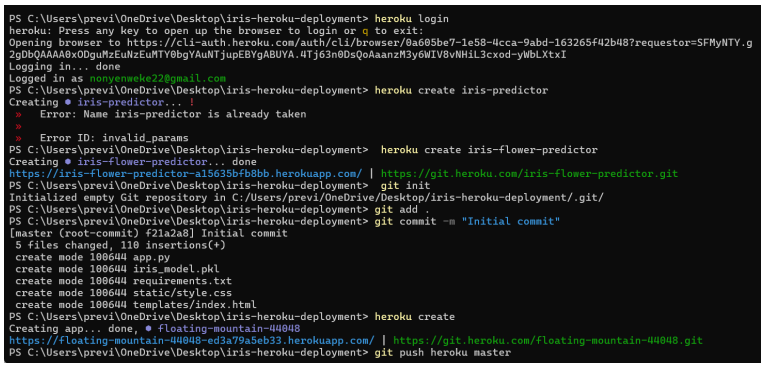
Git repository is initialized using `git init` and commit files using `git add .` and `git commit -m 'Initial commit'`.



Heroku Deployment Report

Step 9: Heroku App creation

a new Heroku app is created using the Heroku CLI command `heroku create your-app-name`.



Heroku Deployment Report

Step 10: Deploying the App on Heroku

The code is pushed to Heroku using `git push heroku master`.

https://floating-mountain-44048-ed3a79a5eb33.herokuapp.com