Carmelo R. Casiraro

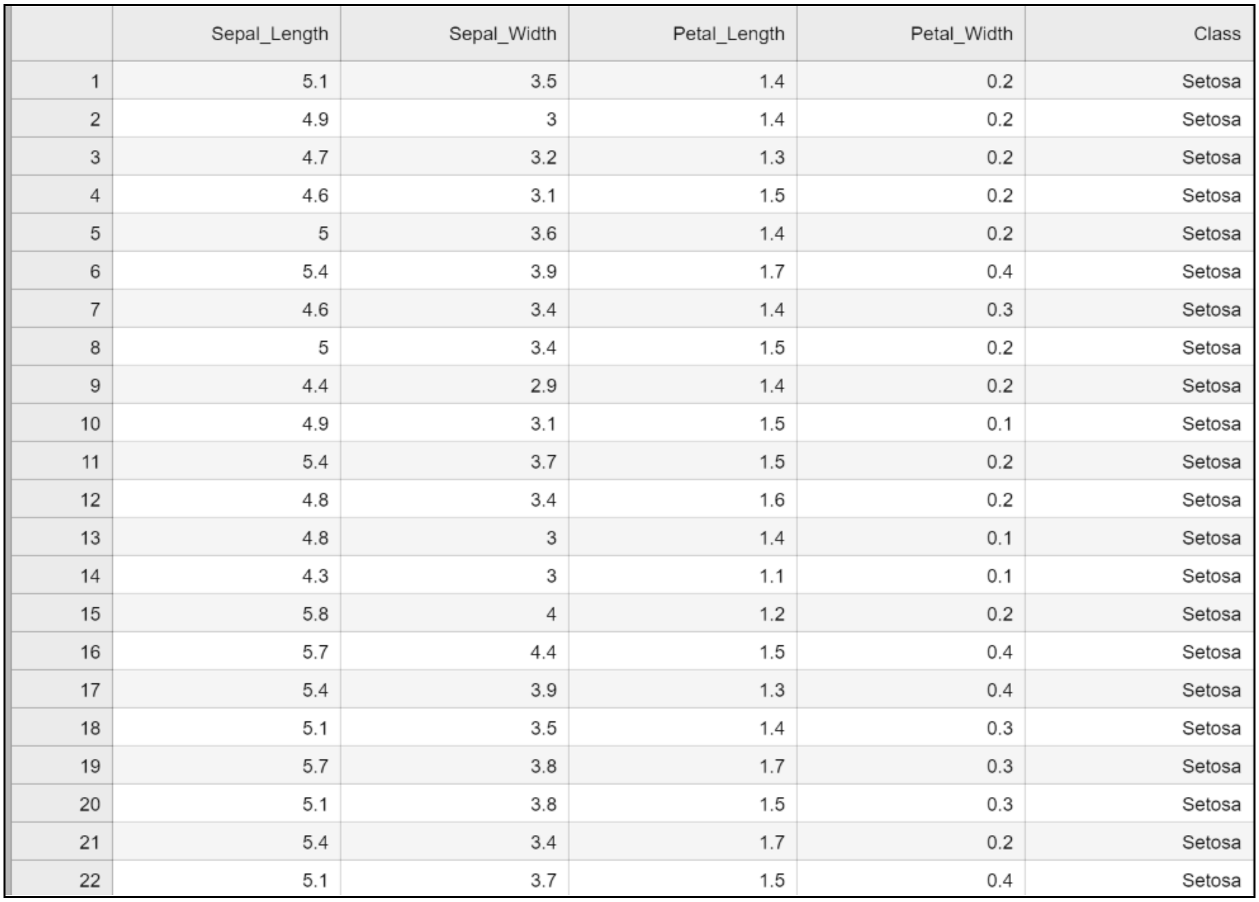
BATCH CODE: LISUM34

SUBMIT DATE: 06/27/24

Model Deployment Using Render.com¶

- Carmelo R. Casiraro, USA- Batch: LISUM34 - Week #5 Assignment- Data Glacier Internship

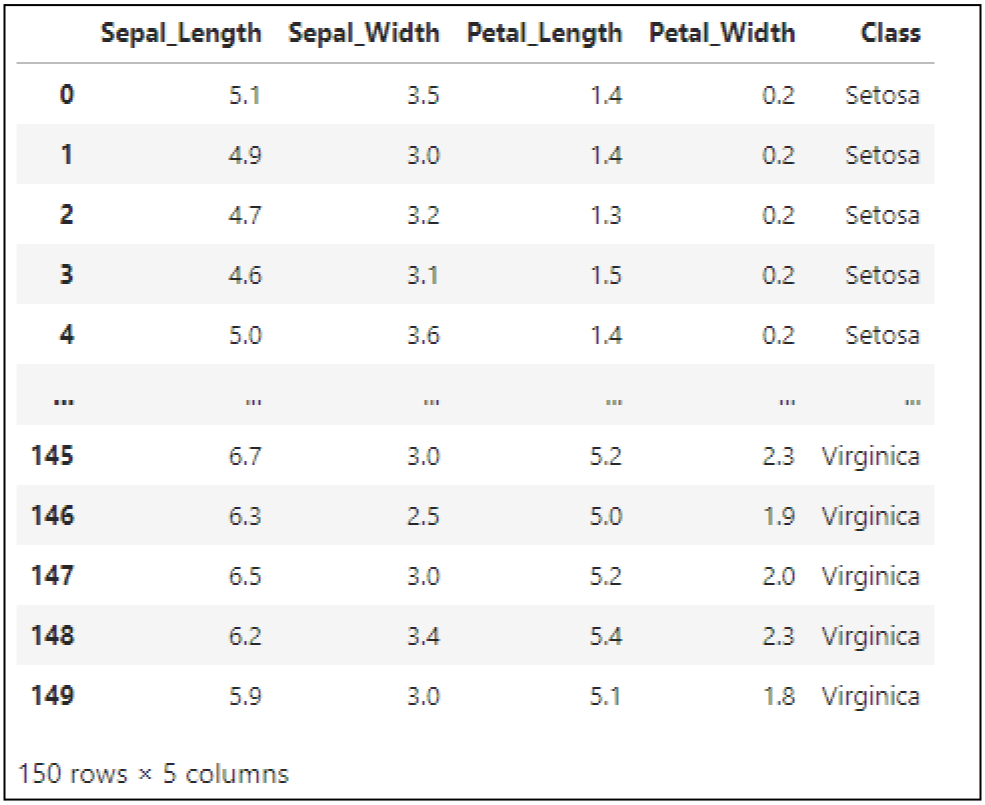
Step 1- Pick Iris Toy Data Set



	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Class
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3	1.4	0.2	Setosa
3	4.7	3.2	1.3	0.2	Setosa
4	4.6	3.1	1.5	0.2	Setosa
5	5	3.6	1.4	0.2	Setosa
6	5.4	3.9	1.7	0.4	Setosa
7	4.6	3.4	1.4	0.3	Setosa
8	5	3.4	1.5	0.2	Setosa
9	4.4	2.9	1.4	0.2	Setosa
10	4.9	3.1	1.5	0.1	Setosa
11	5.4	3.7	1.5	0.2	Setosa
12	4.8	3.4	1.6	0.2	Setosa
13	4.8	3	1.4	0.1	Setosa
14	4.3	3	1.1	0.1	Setosa
15	5.8	4	1.2	0.2	Setosa
16	5.7	4.4	1.5	0.4	Setosa
17	5.4	3.9	1.3	0.4	Setosa
18	5.1	3.5	1.4	0.3	Setosa
19	5.7	3.8	1.7	0.3	Setosa
20	5.1	3.8	1.5	0.3	Setosa
21	5.4	3.4	1.7	0.2	Setosa
22	5.1	3.7	1.5	0.4	Setosa

Step 2- Pre Processing & Modeling Import Libraries

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pickle
from flask import Flask, render_template, request

data = pd.read_csv('iris.csv')
```



	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Class
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Virginica
146	6.3	2.5	5.0	1.9	Virginica
147	6.5	3.0	5.2	2.0	Virginica
148	6.2	3.4	5.4	2.3	Virginica
149	5.9	3.0	5.1	1.8	Virginica

150 rows × 5 columns

Step 3- Split The Data

```
X = data.drop('Class', axis=1)
y = data['Class']
x_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Step 4- Train Model

```
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(x_train, y_train)
RandomForestClassifier(random_state=42)
```

Step 5- Analyze Model

```
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('Model Accuracy:', accuracy)
Model Accuracy: 1.0
```

Step 5.1- if needed, make sure the correct version of scikit-learn is being used

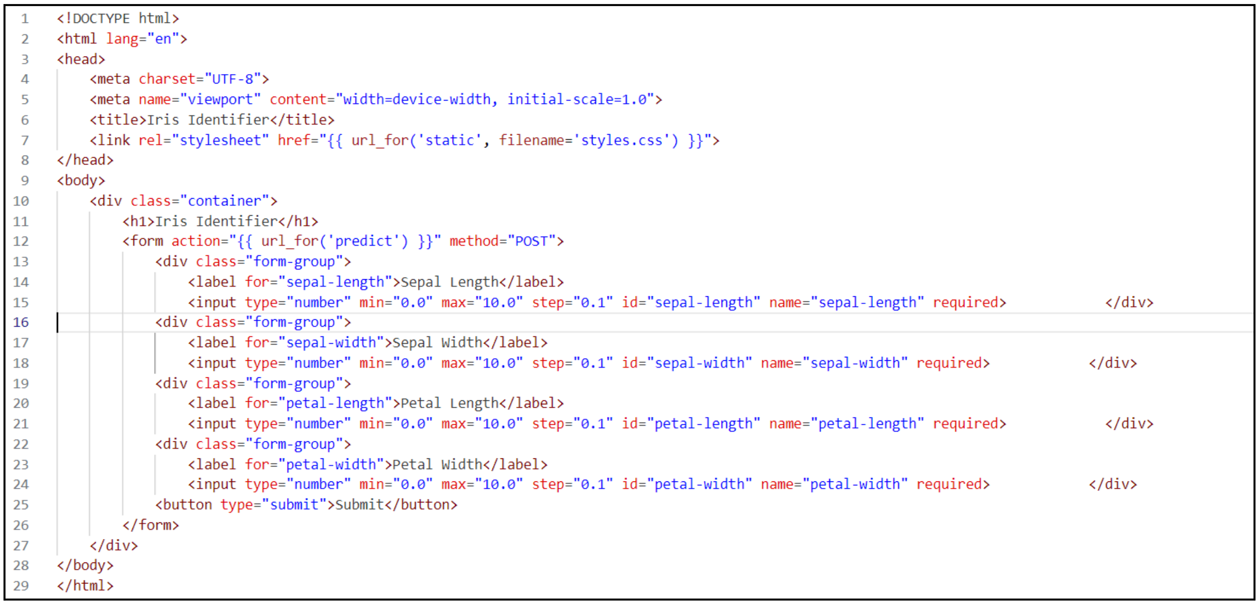
```
import sys
!{sys.executable} -m pip install scikit-learn==1.5.0
import sklearn
print(sklearn.__version__)
1.5.0
```

Step 6- Create PKL file

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

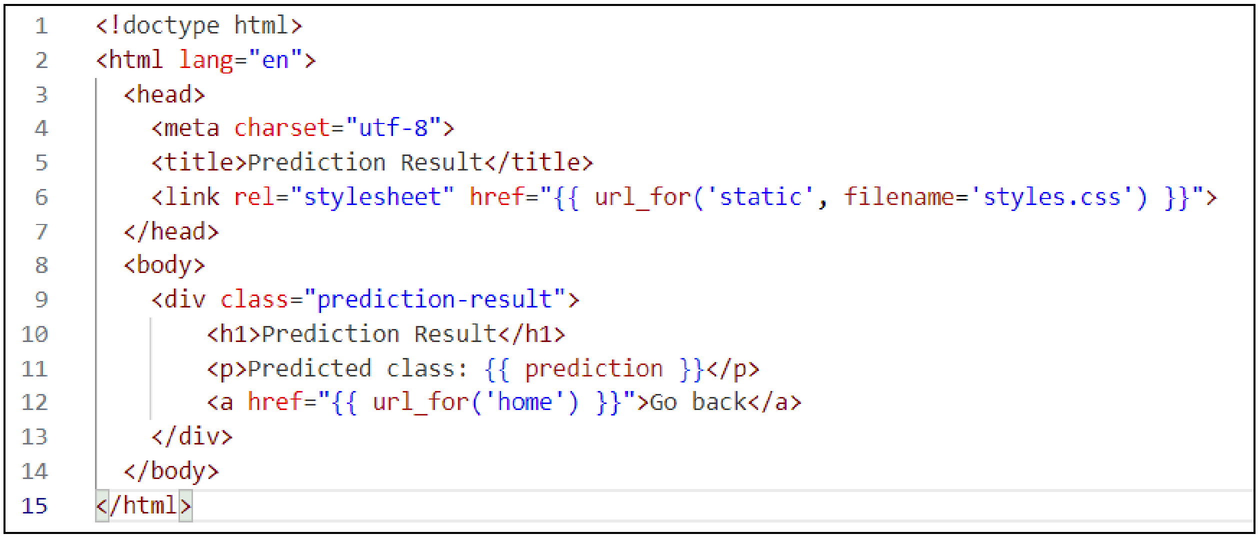
Step 7- Created Html and CSS files for Web Page Deployment

Step 7.1- index.html file- input the values to predict



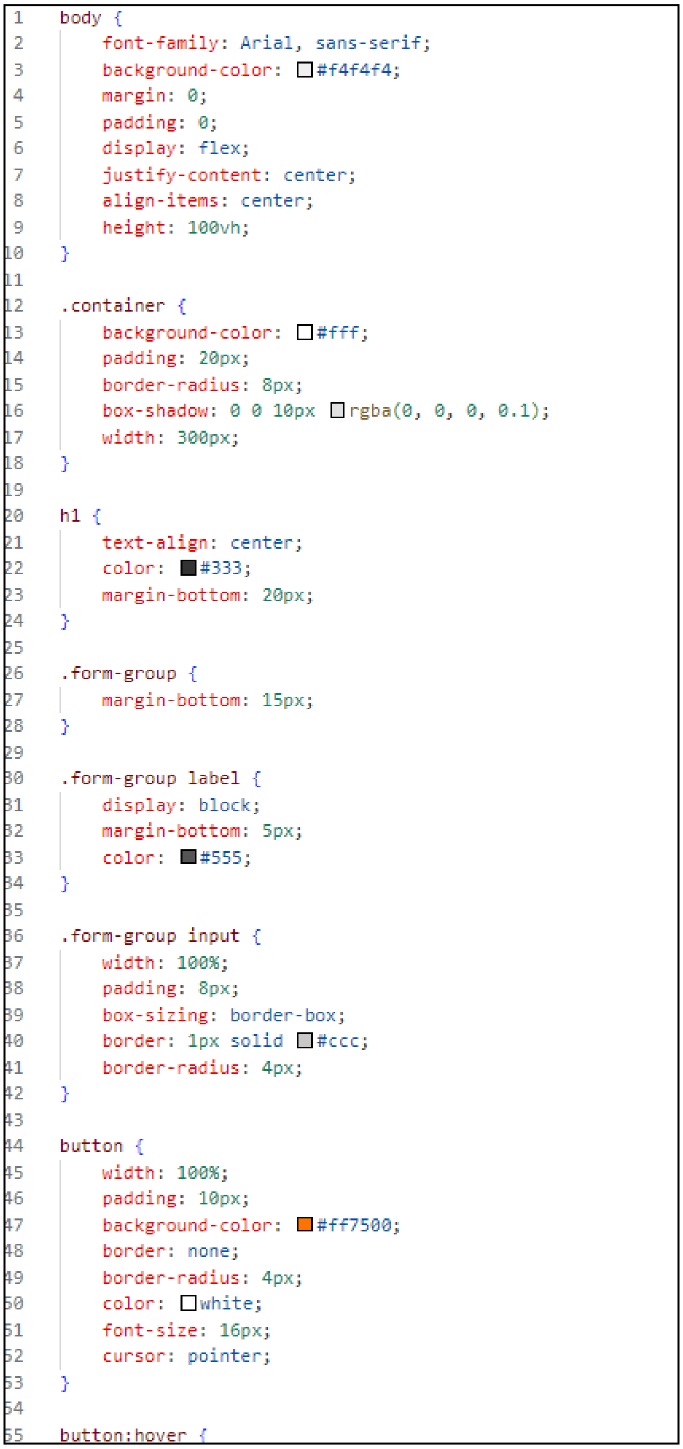
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Iris Identifier</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <div class="container">
        <h1>Iris Identifier</h1>
        <form action="{{ url_for('predict') }}" method="POST">
            <div class="form-group">
                <label for="sepal-length">Sepal Length</label>
                <input type="number" min="0.0" max="10.0" step="0.1" id="sepal-length" name="sepal-length" required>            </div>
            <div class="form-group">
                <label for="sepal-width">Sepal Width</label>
                <input type="number" min="0.0" max="10.0" step="0.1" id="sepal-width" name="sepal-width" required>            </div>
            <div class="form-group">
                <label for="petal-length">Petal Length</label>
                <input type="number" min="0.0" max="10.0" step="0.1" id="petal-length" name="petal-length" required>            </div>
            <div class="form-group">
                <label for="petal-width">Petal Width</label>
                <input type="number" min="0.0" max="10.0" step="0.1" id="petal-width" name="petal-width" required>            </div>
            <button type="submit">Submit</button>
        </form>
    </div>
</body>
</html>
```

Step 7.2- result.html file- this shows the predicted results



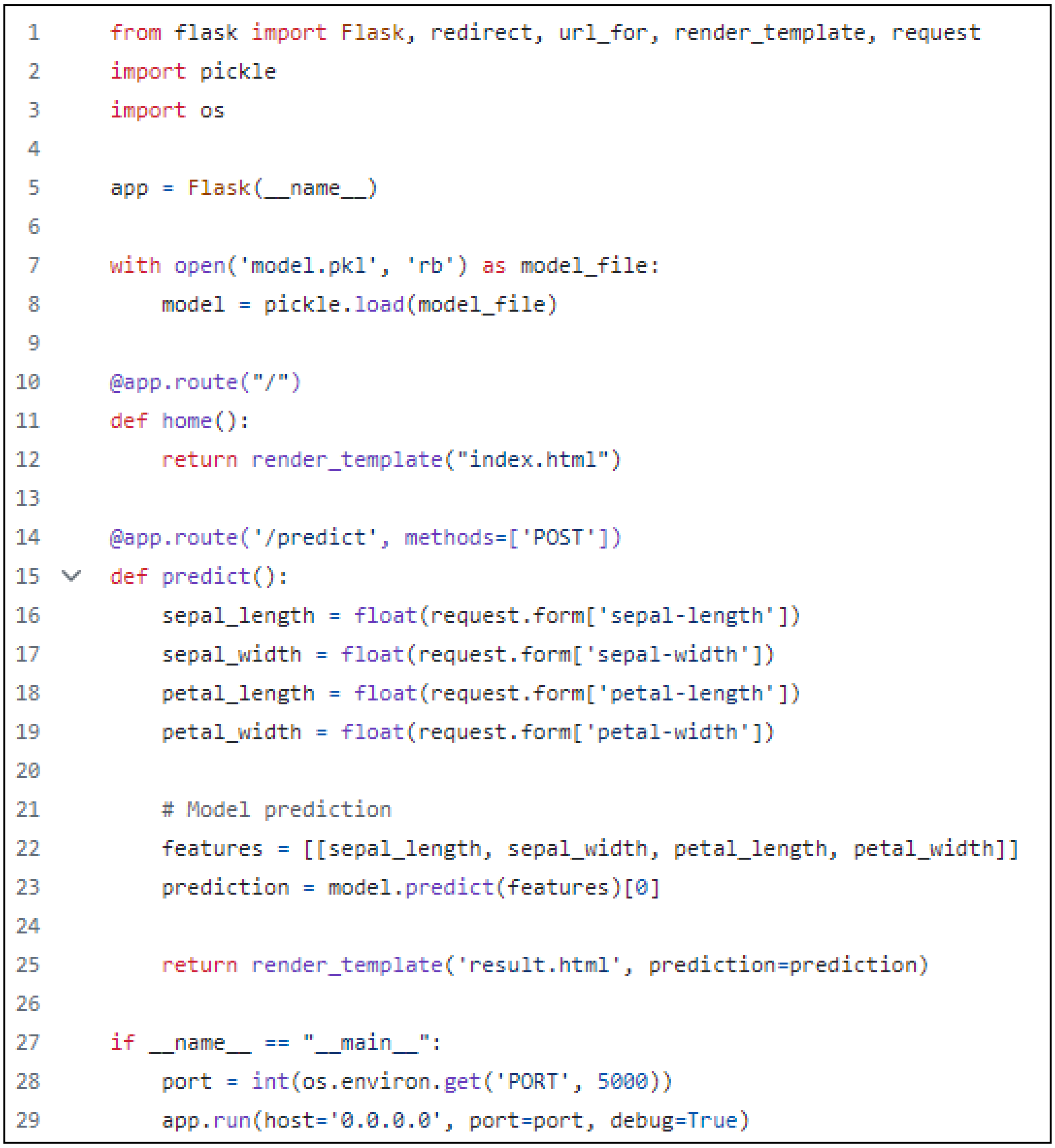
```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Prediction Result</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
  </head>
  <body>
    <div class="prediction-result">
        <h1>Prediction Result</h1>
        <p>Predicted class: {{ prediction }}</p>
        <a href="{{ url_for('home') }}">Go back</a>
    </div>
  </body>
</html>
```

Step 7.3- styles.css file- this formats the web page



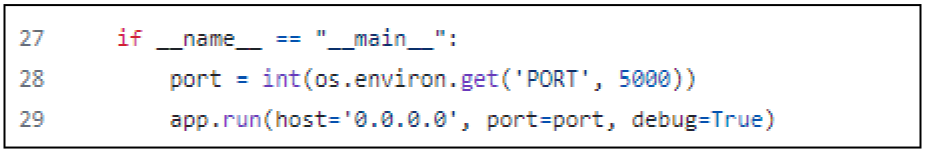
```css
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}

.container {
    background-color: #fff;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    width: 300px;
}

h1 {
    text-align: center;
    color: #333;
    margin-bottom: 20px;
}

.form-group {
    margin-bottom: 15px;
}

.form-group label {
    display: block;
    margin-bottom: 5px;
    color: #555;
}

.form-group input {
    width: 100%;
    padding: 8px;
    box-sizing: border-box;
    border: 1px solid #ccc;
    border-radius: 4px;
}

button {
    width: 100%;
    padding: 10px;
    background-color: #ff7500;
    border: none;
    border-radius: 4px;
    color: white;
    font-size: 16px;
    cursor: pointer;
}

button:hover {
```

Step 8- Create a Flask Application



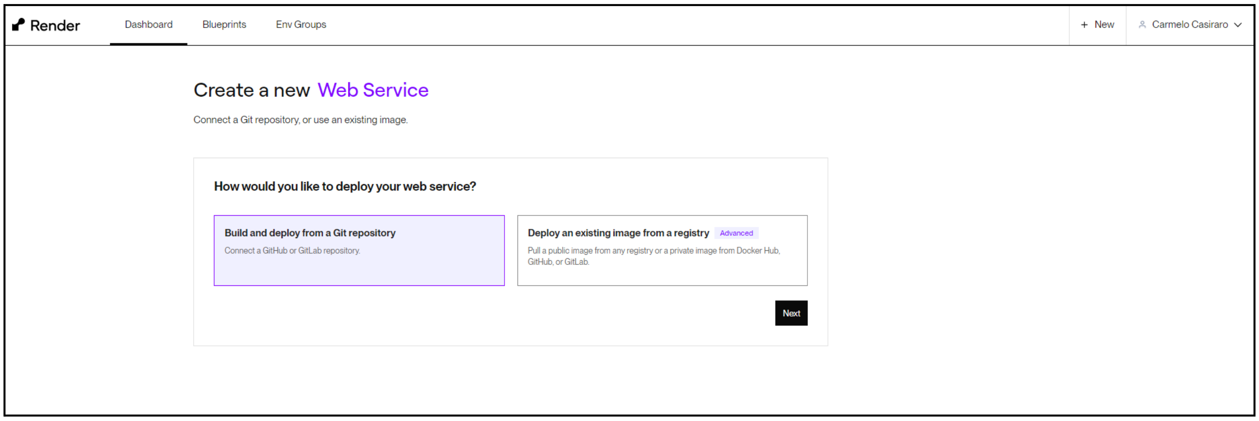
```python
from flask import Flask, redirect, url_for, render_template, request
import pickle
import os

app = Flask(__name__)

with open('model.pkl', 'rb') as model_file:
    model = pickle.load(model_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'])

    # Model prediction
    features = [[sepal_length, sepal_width, petal_length, petal_width]]
    prediction = model.predict(features)[0]

    return render_template('result.html', prediction=prediction)

if __name__ == "__main__":
    port = int(os.environ.get('PORT', 5000))
    app.run(host='0.0.0.0', port=port, debug=True)
```

Step 8.1- Added port through which the Render app can run.



```python
if __name__ == "__main__":
    port = int(os.environ.get('PORT', 5000))
    app.run(host='0.0.0.0', port=port, debug=True)
```

Step 8.2- Create a new Web Service on Render.com



Create a new Web Service

Connect a Git repository, or use an existing image.

How would you like to deploy your web service?

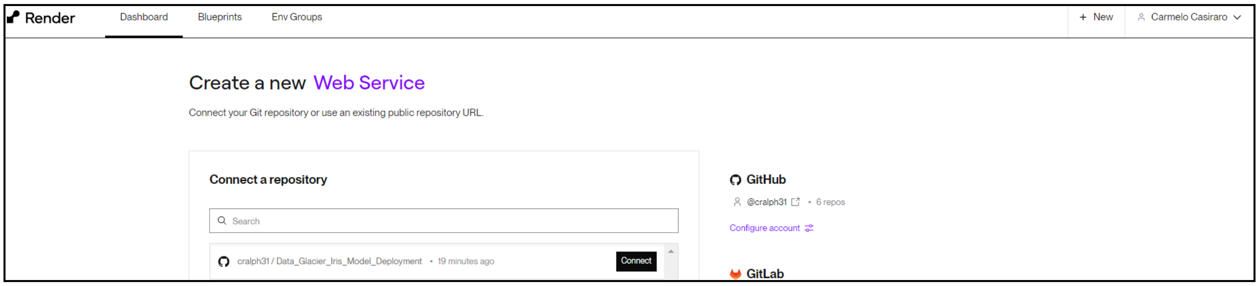
Build and deploy from a Git repository

Connect a GitHub or GitLab repository.

Deploy an existing image from a registry Advanced

Pull a public image from any registry or a private image from Docker Hub, GitHub, or GitLab.

Step 8.3- Connecting to the Github repository through Render.com



Create a new Web Service

Connect your Git repository or use an existing public repository URL.

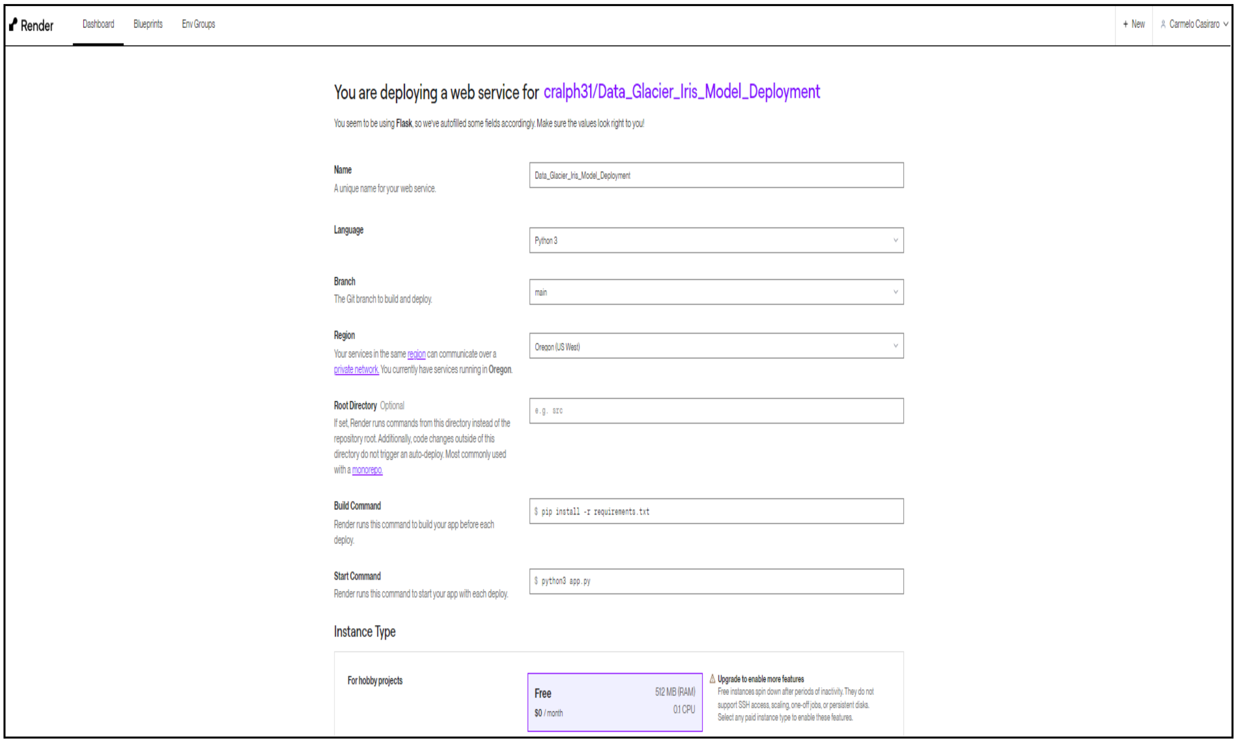
Connect a repository

cralph31 / Data_Glacier_Iris_Model_Deployment • 19 minutes ago

GitHub

@cralph31 • 6 repos

Step 8.4- Creating a web service for Iris Model Deployment repository thru Render.com



You are deploying a web service for cralph31/Data_Glacier_Iris_Model_Deployment

You seem to be using **Flask**, so we've autofilled some fields accordingly. Make sure the values look right to you!

Name
A unique name for your web service.
Data_Glacier_Iris_Model_Deployment

Language
Python 3

Branch
The Git branch to build and deploy.
main

Region
Your services in the same region can communicate over a private network. You currently have services running in **Oregon**.
Oregon (US West)

Root Directory Optional
If set, Render runs commands from this directory instead of the repository root. Additionally, code changes outside of this directory do not trigger an auto-deploy. Most commonly used with a monorepo.
e.g. src

Build Command
Render runs this command to build your app before each deploy.

```
$ pip install -r requirements.txt
```

Start Command
Render runs this command to start your app with each deploy.

```
$ python3 app.py
```

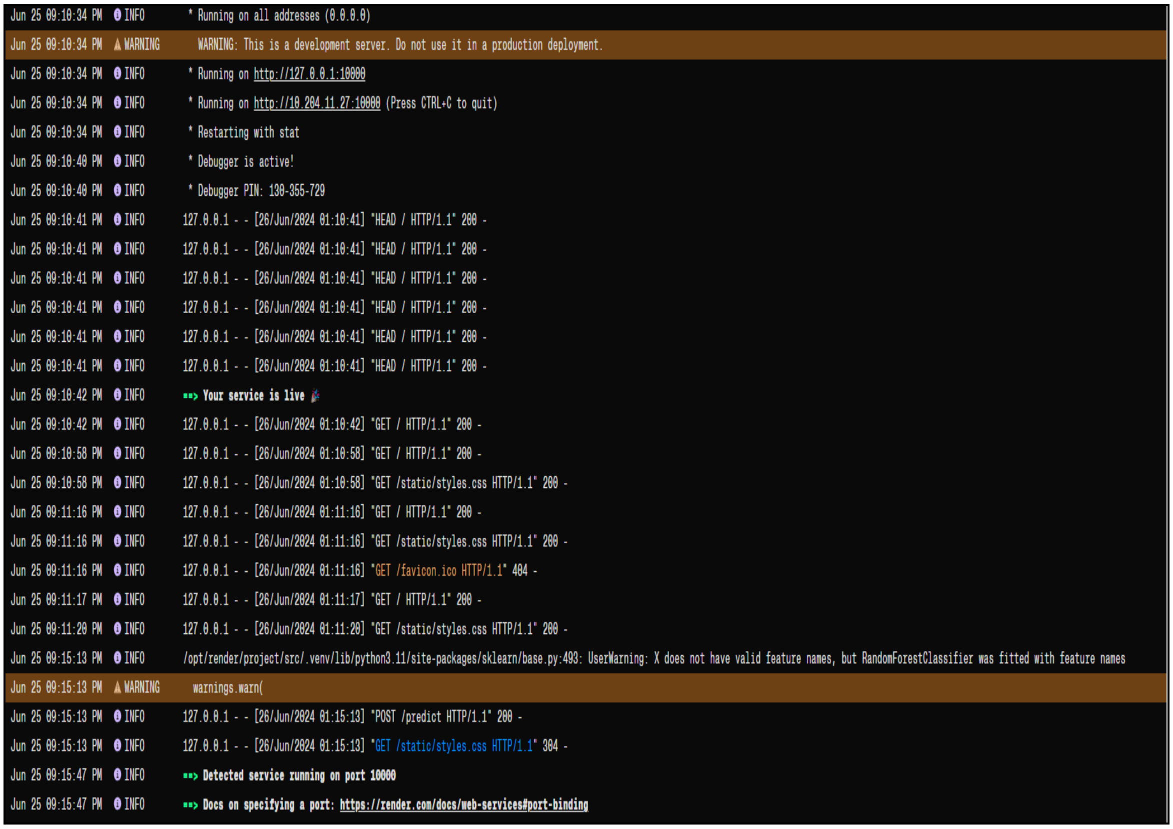
Instance Type

For hobby projects

Free 512 MB (RAM)
$0 / month 0.1 CPU

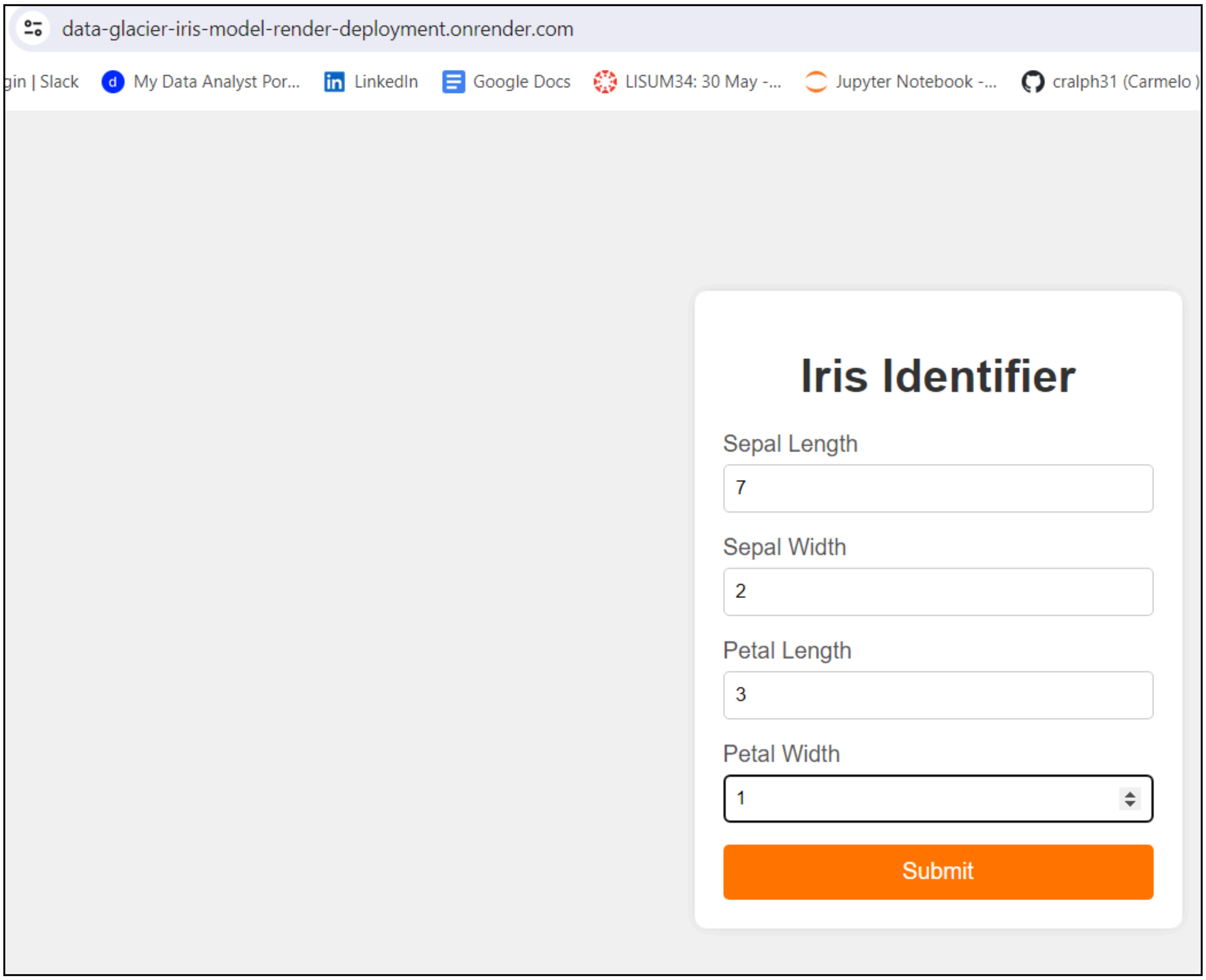
⚠ **Upgrade to enable more features**
Free instances spin down after periods of inactivity. They do not support SSH access, scaling, one-off jobs, or persistent disks. Select any paid instance type to enable these features.

Step 9- Deploy Application Using Command Prompt



```
Jun 25 09:10:34 PM  ⓘ INFO       * Running on all addresses (0.0.0.0)
Jun 25 09:10:34 PM  ⚠ WARNING     WARNING: This is a development server. Do not use it in a production deployment.
Jun 25 09:10:34 PM  ⓘ INFO       * Running on http://127.0.0.1:10000
Jun 25 09:10:34 PM  ⓘ INFO       * Running on http://10.204.11.27:10000 (Press CTRL+C to quit)
Jun 25 09:10:34 PM  ⓘ INFO       * Restarting with stat
Jun 25 09:10:40 PM  ⓘ INFO       * Debugger is active!
Jun 25 09:10:40 PM  ⓘ INFO       * Debugger PIN: 130-355-729
Jun 25 09:10:41 PM  ⓘ INFO      127.0.0.1 - - [26/Jun/2024 01:10:41] "HEAD / HTTP/1.1" 200 -
Jun 25 09:10:41 PM  ⓘ INFO      127.0.0.1 - - [26/Jun/2024 01:10:41] "HEAD / HTTP/1.1" 200 -
Jun 25 09:10:41 PM  ⓘ INFO      127.0.0.1 - - [26/Jun/2024 01:10:41] "HEAD / HTTP/1.1" 200 -
Jun 25 09:10:41 PM  ⓘ INFO      127.0.0.1 - - [26/Jun/2024 01:10:41] "HEAD / HTTP/1.1" 200 -
Jun 25 09:10:41 PM  ⓘ INFO      127.0.0.1 - - [26/Jun/2024 01:10:41] "HEAD / HTTP/1.1" 200 -
Jun 25 09:10:41 PM  ⓘ INFO      127.0.0.1 - - [26/Jun/2024 01:10:41] "HEAD / HTTP/1.1" 200 -
Jun 25 09:10:42 PM  ⓘ INFO      ==> Your service is live 🎉
Jun 25 09:10:42 PM  ⓘ INFO      127.0.0.1 - - [26/Jun/2024 01:10:42] "GET / HTTP/1.1" 200 -
Jun 25 09:10:58 PM  ⓘ INFO      127.0.0.1 - - [26/Jun/2024 01:10:58] "GET / HTTP/1.1" 200 -
Jun 25 09:10:58 PM  ⓘ INFO      127.0.0.1 - - [26/Jun/2024 01:10:58] "GET /static/styles.css HTTP/1.1" 200 -
Jun 25 09:11:16 PM  ⓘ INFO      127.0.0.1 - - [26/Jun/2024 01:11:16] "GET / HTTP/1.1" 200 -
Jun 25 09:11:16 PM  ⓘ INFO      127.0.0.1 - - [26/Jun/2024 01:11:16] "GET /static/styles.css HTTP/1.1" 200 -
Jun 25 09:11:16 PM  ⓘ INFO      127.0.0.1 - - [26/Jun/2024 01:11:16] "GET /favicon.ico HTTP/1.1" 404 -
Jun 25 09:11:17 PM  ⓘ INFO      127.0.0.1 - - [26/Jun/2024 01:11:17] "GET / HTTP/1.1" 200 -
Jun 25 09:11:20 PM  ⓘ INFO      127.0.0.1 - - [26/Jun/2024 01:11:20] "GET /static/styles.css HTTP/1.1" 200 -
Jun 25 09:15:13 PM  ⓘ INFO      /opt/render/project/src/.venv/lib/python3.11/site-packages/sklearn/base.py:493: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names
Jun 25 09:15:13 PM  ⚠ WARNING     warnings.warn(
Jun 25 09:15:13 PM  ⓘ INFO      127.0.0.1 - - [26/Jun/2024 01:15:13] "POST /predict HTTP/1.1" 200 -
Jun 25 09:15:13 PM  ⓘ INFO      127.0.0.1 - - [26/Jun/2024 01:15:13] "GET /static/styles.css HTTP/1.1" 304 -
Jun 25 09:15:47 PM  ⓘ INFO      ==> Detected service running on port 10000
Jun 25 09:15:47 PM  ⓘ INFO      ==> Docs on specifying a port: https://render.com/docs/web-services#port-binding
```

Step 10- Web Page Test Using Render



data-glacier-iris-model-render-deployment.onrender.com

Iris Identifier

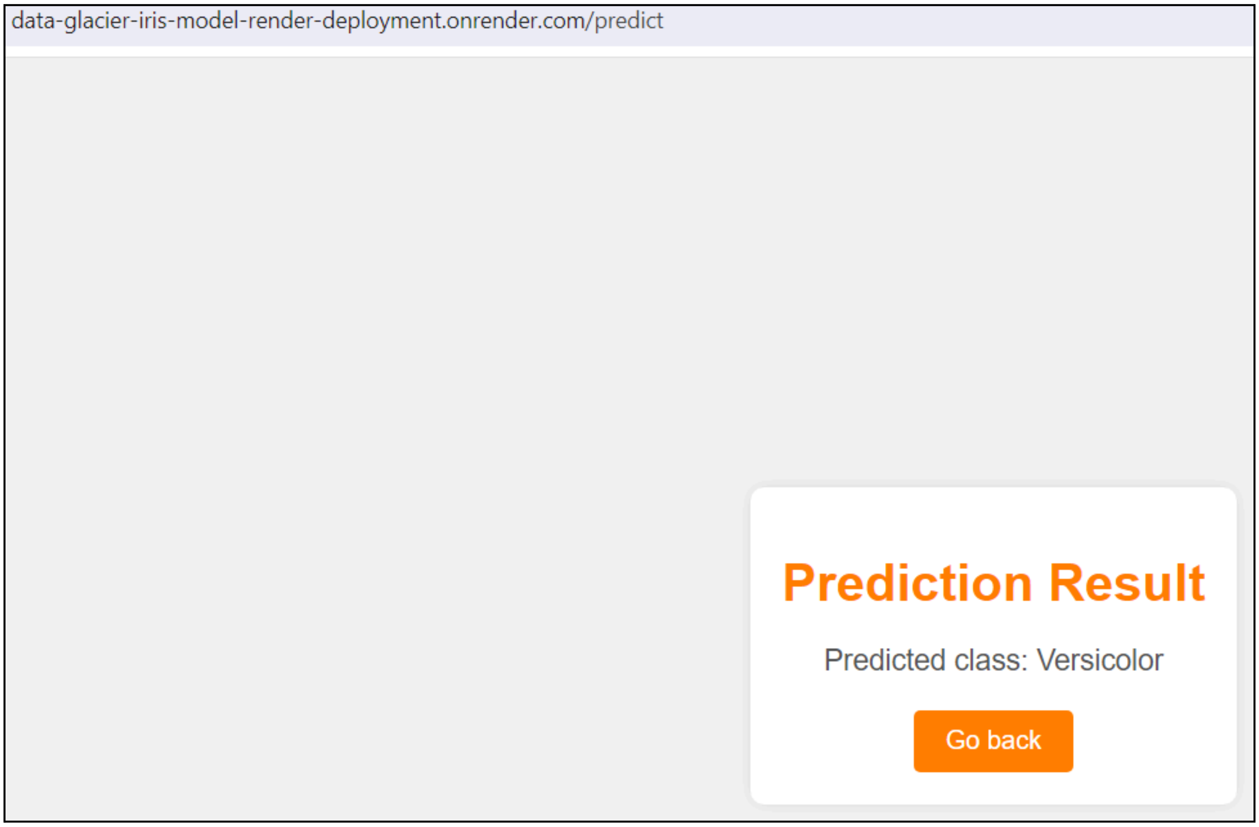
Sepal Length: 7

Sepal Width: 2

Petal Length: 3

Petal Width: 1

Step 11- Predicted Result



data-glacier-iris-model-render-deployment.onrender.com/predict

Prediction Result

Predicted class: Versicolor

Summary of Project

- This project involved creating a machine learning model to predict the species of a Iris flower based on sepal and petal dimensions. The Iris data set was used to train and test the model. The model was built using Python sklearn library. A flask application was created allowing users to input different numbers for dimensions to predict the type of flower. Then, html files were created. First file was index.html file to inputting values, the second file was a results.html file to predict the results. The application was then deployed using Visual studio code using command prompt. After deployment, user inputted values for dimensions for flower then clicked Submit. The result of the submission, showed the flower predicted based on the model. In conclusion, this project shows how to create a machine learning model & web application, then, finally deploying it all using a Flask application on Render.