



Name: Ian Kihara Wangui

Batch Code: LISUM 19

Submission date: 23/03/2023

Submitted to: Data Glacier

----- Local Deployment -----

This guide illustrates local deployment on your machine. I will consider that you are using Visual Studio Code, in case you are not, no problem at all, the first thing is to create a virtual environment in the folder containing your code.

Install virtualenv:

```
pip install virtualenv
```

In the folder of your project, if not:

```
cd my-project
```

Activate the Virtual Environment

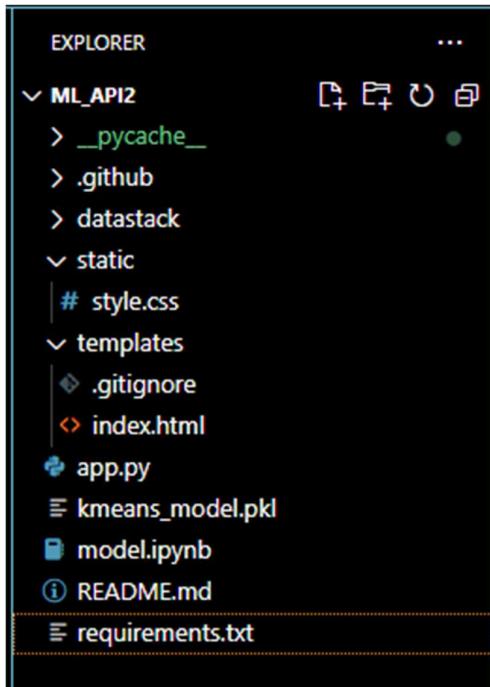
```
virtualenv --python C:\Path\To\Python\python.exe venv
```

Then, activate the Virtual Environment:

```
.\venv\Scripts\activate
```

If you need to install packages from a determined project type: **pip install -r requirements.txt**

All set to start the development, the final folder structure should look like this:



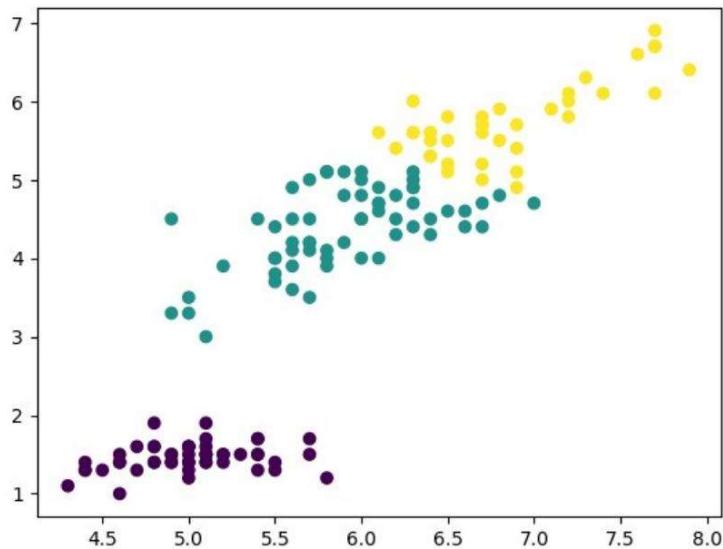
This application uses the iris dataset of scikit-learn.(

[https://archive.ics.uci.edu/ml/datasets/iris\)](https://archive.ics.uci.edu/ml/datasets/iris)

A screenshot of the UCI Machine Learning Repository website. The URL in the browser bar is https://archive.ics.uci.edu/ml/datasets/iris. The page features the UCI logo and the title "Machine Learning Repository" with the subtitle "Center for Machine Learning and Intelligent Systems". A brown banner at the bottom says "Check out the beta version of the new UCI Machine Learning Repository we are currently testing! Contact us if you". Below this, there's a section for the "Iris Data Set" with a "Download" link to "Data Folder, Data Set Description". An abstract states: "Abstract: Famous database; from Fisher, 1936". To the right is a photograph of an Iris flower. A table below provides dataset characteristics.

Data Set Characteristics:	Multivariate	Number of Instances:	150	Area:	Life
Attribute Characteristics:	Real	Number of Attributes:	4	Date Donated	1988-07-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	5282781

It is time to start working on our machine learning models; here is the snapshot code for **model.ipynb** file:



```
# Sample data for a prediction
new_sample = [[5.7, 4.4, 1.5, 0.4]]
print(new_sample)
[[5.7, 4.4, 1.5, 0.4]]

# Make a prediction
new_label = model.predict(new_sample)
print(new_label)

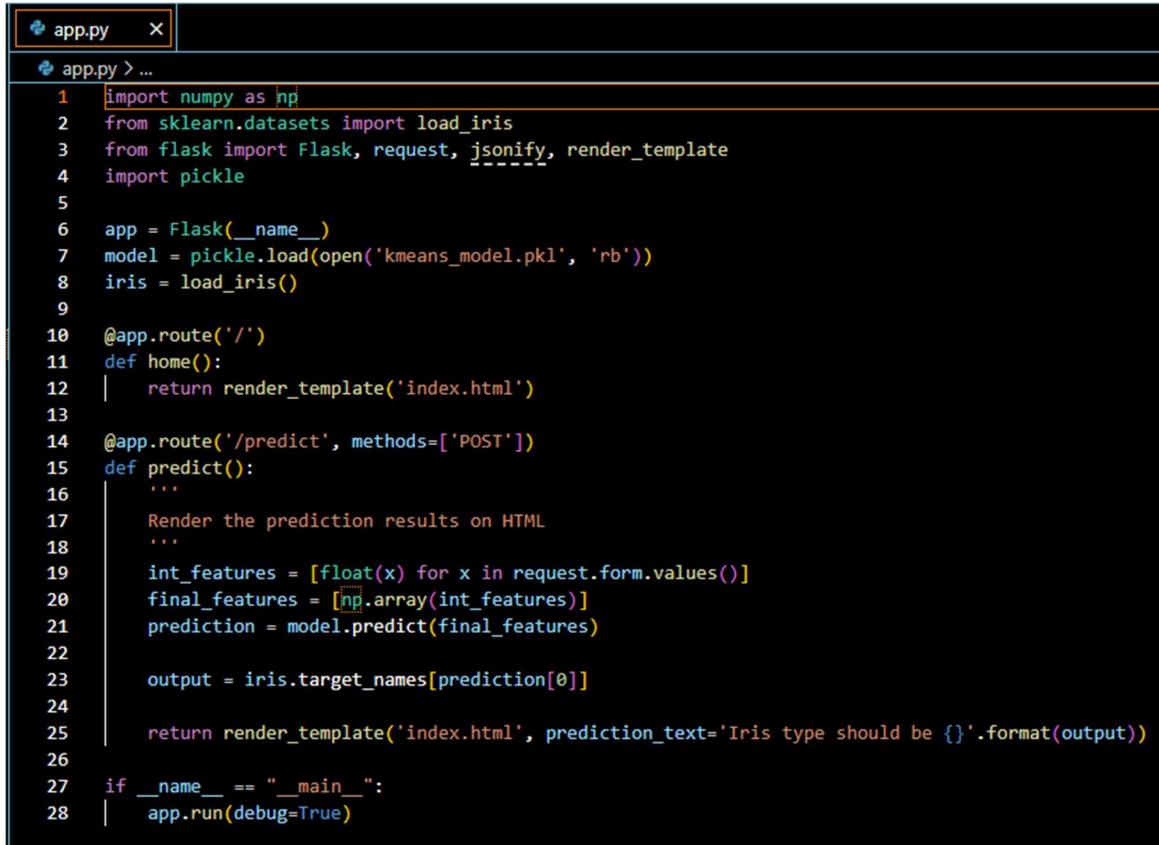
[0]

# Evaluating the clustering with cross tabulation
species = y[:,]
df = pd.DataFrame({'labels': labels, 'species': species})
ct = pd.crosstab(df['labels'], df['species'])
print(ct)

species    0    1    2
labels
0         50   0   0
1         0   48  14
2         0    2   36

# Save model to disk using pickle
with open('kmeans_model.pkl', 'wb') as f:
    pickle.dump(model, f)
```

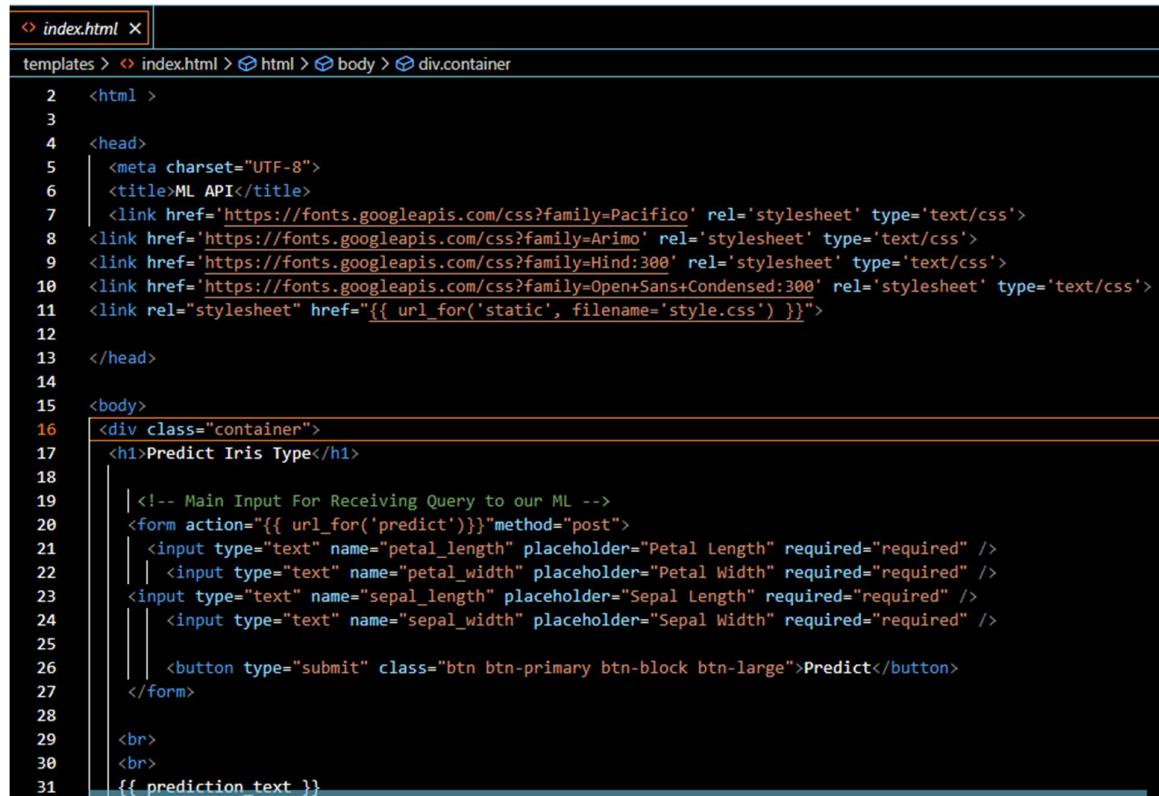
Ok, after the model is trained you should save a pickle file of it, then we will work on the app.py file:



The screenshot shows a code editor window with the title bar "app.py". The code is written in Python and defines a Flask application. It imports numpy, sklearn.datasets, flask, and pickle. It loads a k-means model from a file named "kmeans_model.pkl". The application has two routes: a home route that returns an index.html template, and a predict route that handles POST requests. The predict route performs feature extraction from the request form, makes a prediction using the loaded model, and returns the result in an index.html template. Finally, if the script is run directly, it starts the Flask development server.

```
1 import numpy as np
2 from sklearn.datasets import load_iris
3 from flask import Flask, request, jsonify, render_template
4 import pickle
5
6 app = Flask(__name__)
7 model = pickle.load(open('kmeans_model.pkl', 'rb'))
8 iris = load_iris()
9
10 @app.route('/')
11 def home():
12     return render_template('index.html')
13
14 @app.route('/predict', methods=['POST'])
15 def predict():
16     ...
17     Render the prediction results on HTML
18     ...
19     int_features = [float(x) for x in request.form.values()]
20     final_features = [np.array(int_features)]
21     prediction = model.predict(final_features)
22
23     output = iris.target_names[prediction[0]]
24
25     return render_template('index.html', prediction_text='Iris type should be {}'.format(output))
26
27 if __name__ == "__main__":
28     app.run(debug=True)
```

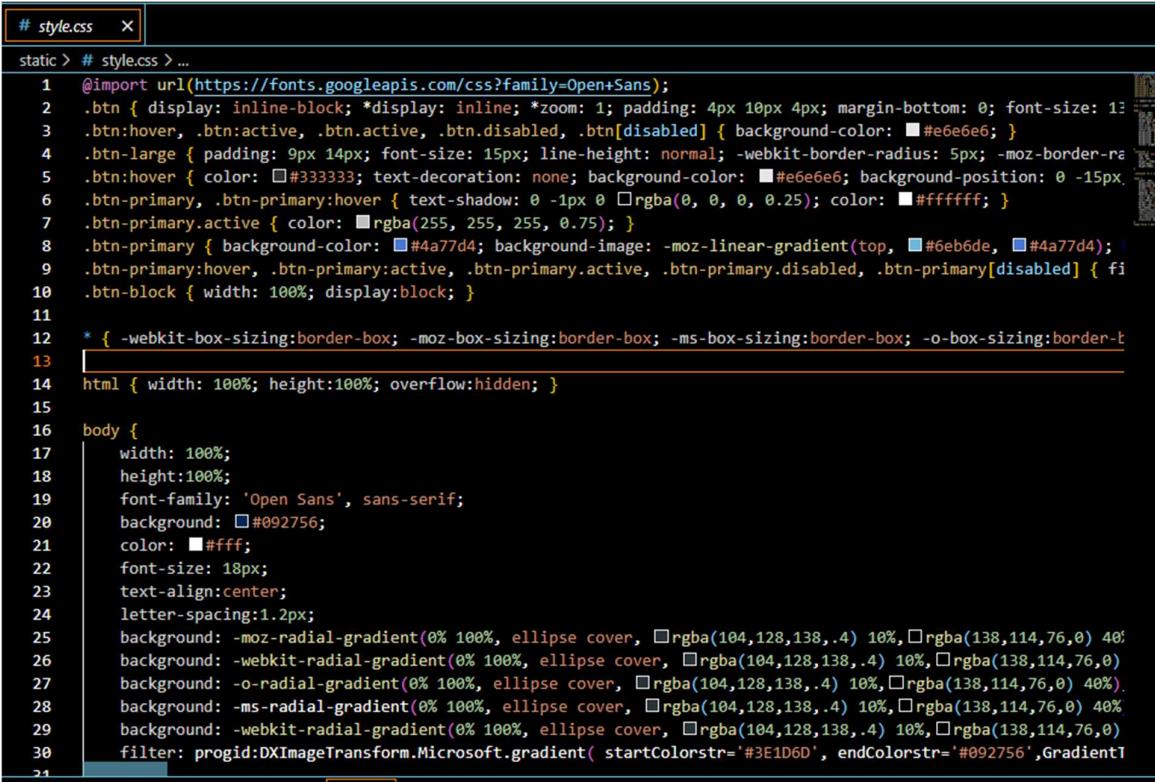
All right, go to the template folder and open a new file named index.html; here is the following code:



The screenshot shows a code editor window with the title bar "index.html". The code is an HTML template for a web page. It includes a head section with meta charset="UTF-8", title="ML API", and links to various Google Fonts (Pacifico, Arimo, Hind, Open+Sans+Condensed). The body section contains a container div, an h1 header with the text "Predict Iris Type", and a form for receiving user input. The form has four text input fields for petal length, petal width, sepal length, and sepal width, all required. It also contains a submit button labeled "Predict". Below the form, there is a placeholder for the prediction result: "{{ prediction_text }}".

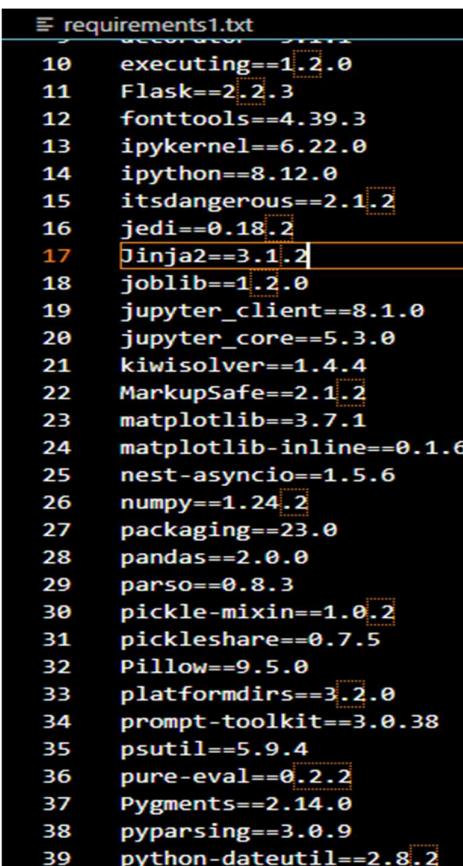
```
1 <html>
2   <head>
3     <meta charset="UTF-8">
4     <title>ML API</title>
5     <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>
6     <link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>
7     <link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>
8     <link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' type='text/css'>
9     <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}>
10
11 </head>
12
13 <body>
14   <div class="container">
15     <h1>Predict Iris Type</h1>
16
17     <!-- Main Input For Receiving Query to our ML -->
18     <form action="{{ url_for('predict') }}" method="post">
19       <input type="text" name="petal_length" placeholder="Petal Length" required="required" />
20       <input type="text" name="petal_width" placeholder="Petal Width" required="required" />
21       <input type="text" name="sepal_length" placeholder="Sepal Length" required="required" />
22       <input type="text" name="sepal_width" placeholder="Sepal Width" required="required" />
23
24       <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
25     </form>
26
27   <br>
28   <br>
29   {{ prediction_text }}
```

To style the website I used Bootstrap to speed up my development.



```
# style.css X
static > # style.css > ...
1 @import url(https://fonts.googleapis.com/css?family=Open+Sans);
2 .btn { display: inline-block; *display: inline; *zoom: 1; padding: 4px 10px 4px; margin-bottom: 0; font-size: 13px; }
3 .btn:hover, .btn:active, .btn.active, .btn.disabled, .btn[disabled] { background-color: #e6e6e6; }
4 .btn-large { padding: 9px 14px; font-size: 15px; line-height: normal; -webkit-border-radius: 5px; -moz-border-radius: 5px; border-radius: 5px; }
5 .btn:hover { color: #333333; text-decoration: none; background-color: #e6e6e6; background-position: 0 -15px; }
6 .btn-primary, .btn-primary:hover { text-shadow: 0 -1px 0 #rgba(0, 0, 0, 0.25); color: #ffffff; }
7 .btn-primary.active { color: #rgba(255, 255, 255, 0.75); }
8 .btn-primary { background-color: #4a77d4; background-image: -moz-linear-gradient(top, #6eb6de, #4a77d4); }
9 .btn-primary:hover, .btn-primary:active, .btn-primary.active, .btn-primary.disabled, .btn-primary[disabled] { background-color: #4a77d4; }
10 .btn-block { width: 100%; display: block; }
11
12 * { -webkit-box-sizing:border-box; -moz-box-sizing:border-box; -ms-box-sizing:border-box; -o-box-sizing:border-box; border: 1px solid #e6e6e6; }
13
14 html { width: 100%; height:100%; overflow:hidden; }
15
16 body {
17     width: 100%; height:100%; font-family: 'Open Sans', sans-serif; background: #092756; color: #fff; font-size: 18px; text-align:center; letter-spacing:1.2px; background: -moz-radial-gradient(0% 100%, ellipse cover, #rgba(104,128,138,.4) 10%, #rgba(138,114,76,0) 40%); background: -webkit-radial-gradient(0% 100%, ellipse cover, #rgba(104,128,138,.4) 10%, #rgba(138,114,76,0) 40%); background: -o-radial-gradient(0% 100%, ellipse cover, #rgba(104,128,138,.4) 10%, #rgba(138,114,76,0) 40%); background: -ms-radial-gradient(0% 100%, ellipse cover, #rgba(104,128,138,.4) 10%, #rgba(138,114,76,0) 40%); background: -webkit-radial-gradient(0% 100%, ellipse cover, #rgba(104,128,138,.4) 10%, #rgba(138,114,76,0) 40%); filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#3E1D6D', endColorstr='#092756', GradientType=1 ); }
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
```

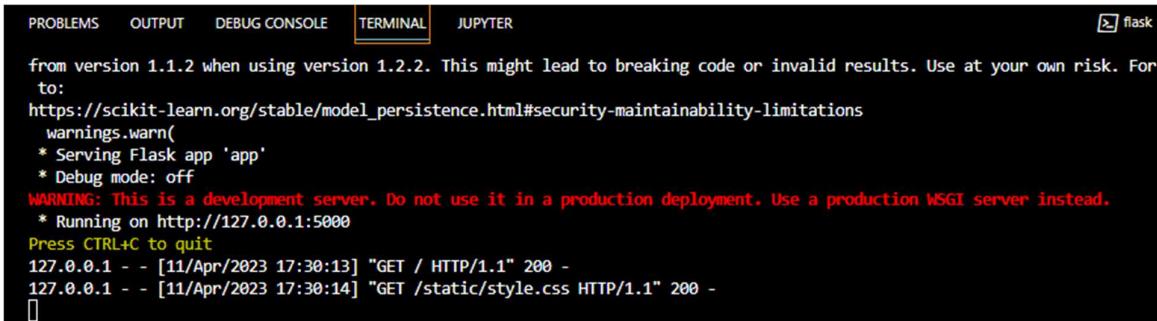
Type the following commands to generate a list of packages to install when you finish the deployment in the terminal: **pip freeze > requirements.txt**



```
requirements1.txt
10 executing==1.2.0
11 Flask==2.2.3
12 fonttools==4.39.3
13 ipykernel==6.22.0
14 ipython==8.12.0
15 itsdangerous==2.1.2
16 jedi==0.18.2
17 Jinja2==3.1.2
18 joblib==1.2.0
19 jupyter_client==8.1.0
20 jupyter_core==5.3.0
21 kiwisolver==1.4.4
22 MarkupSafe==2.1.2
23 matplotlib==3.7.1
24 matplotlib-inline==0.1.6
25 nest-asyncio==1.5.6
26 numpy==1.24.2
27 packaging==23.0
28 pandas==2.0.0
29 parso==0.8.3
30 pickle-mixin==1.0.2
31 pickleshare==0.7.5
32 Pillow==9.5.0
33 platformdirs==3.2.0
34 prompt-toolkit==3.0.38
35 psutil==5.9.4
36 pure-eval==0.2.2
37 Pygments==2.14.0
38 pyparsing==3.0.9
39 python-dateutil==2.8.2
```

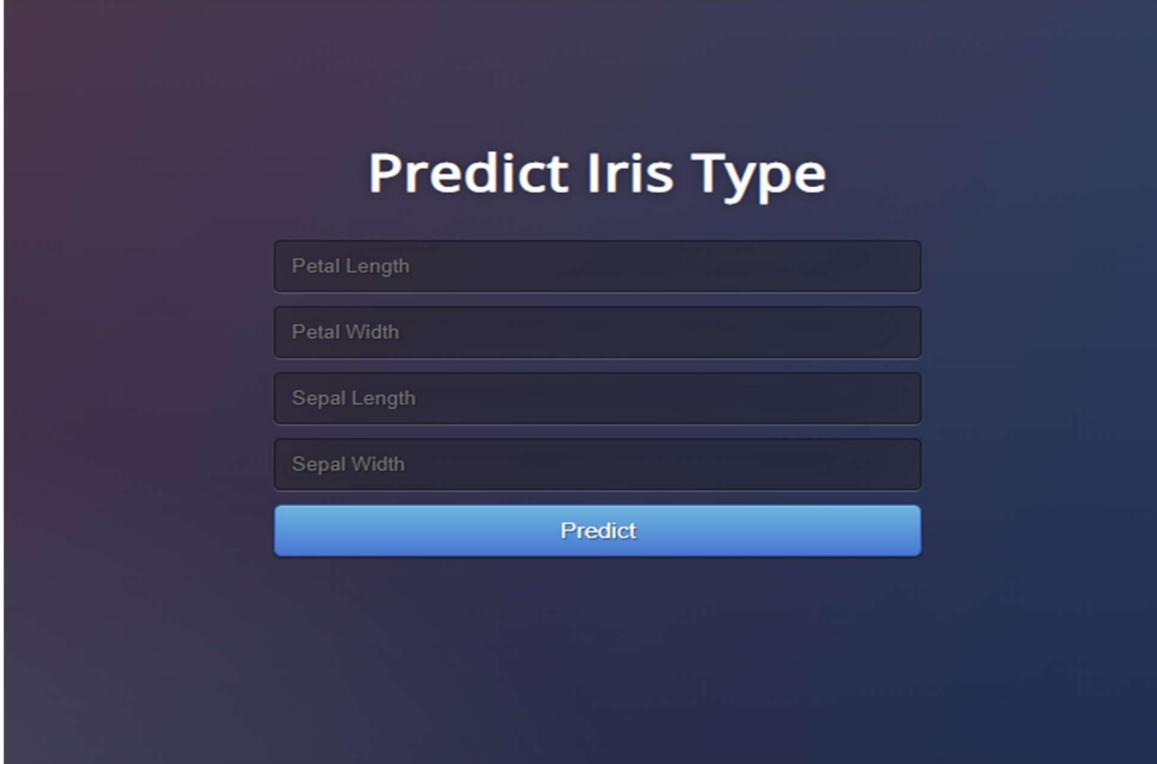
At this point, we can run the following command to run the flask application:

```
flask -app app run
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER flask
from version 1.1.2 when using version 1.2.2. This might lead to breaking code or invalid results. Use at your own risk. For
to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
    * Serving Flask app 'app'
    * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
  * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [11/Apr/2023 17:30:13] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [11/Apr/2023 17:30:14] "GET /static/style.css HTTP/1.1" 200 -
```

Go to <http://127.0.0.1:5000> in your browser, you see the website, once you do some request, the terminal should change to show the **API request**, you can notice the methods **POST** and **GET** and the **http response 200** for successful anything different, it points an error.



Finally, after you run all codes, you need to deactivate the virtual environment; here is the command:

```
deactivate
```