EXP NO: 9	MINI PROJECT - TIME SERIES-BASED STOCK PRICE FORECASTING WITH LINEAR REGRESSION

Aim:

To demonstrate the process of deploying a pre-trained machine learning model as a RESTful API using Flask, and then containerizing this API and model using Docker for easy deployment and portability.

Algorithm:

1. RESTful API Design (Flask)

Representational State Transfer (REST) is an architectural style for networked applications. A RESTful API uses standard HTTP methods (GET, POST, PUT, DELETE) to perform operations on resources.

Key Concepts:

Resources: Any data object that can be identified, named, addressed, or handled in the web. In our case, the ML model's prediction endpoint will be a resource.

Endpoints: Specific URLs that represent the resources.

HTTP Methods:

`POST`: Used to submit data to a specified resource (e.g., send new data for prediction).

`GET`: Used to request data from a specified resource (e.g., check API status).

Statelessness: Each request from a client to a server must contain all the information needed to understand the request. The server should not store any client context between requests.

JSON: JavaScript Object Notation is commonly used for data exchange between the client and the API.

2. Containerization with Docker

Docker is a platform that uses OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries, and configuration files.

Key Concepts:

Dockerfile: A text file that contains all the commands a user could call on the command line to assemble an image. It defines the environment, dependencies, and execution command for your application.

Image: A lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries, and settings.

Container: A runnable instance of an image. You can create, start, stop, move, or delete a container.

Port Mapping: Connecting a port on the host machine to a port inside the Docker container.

CODE:

train.py

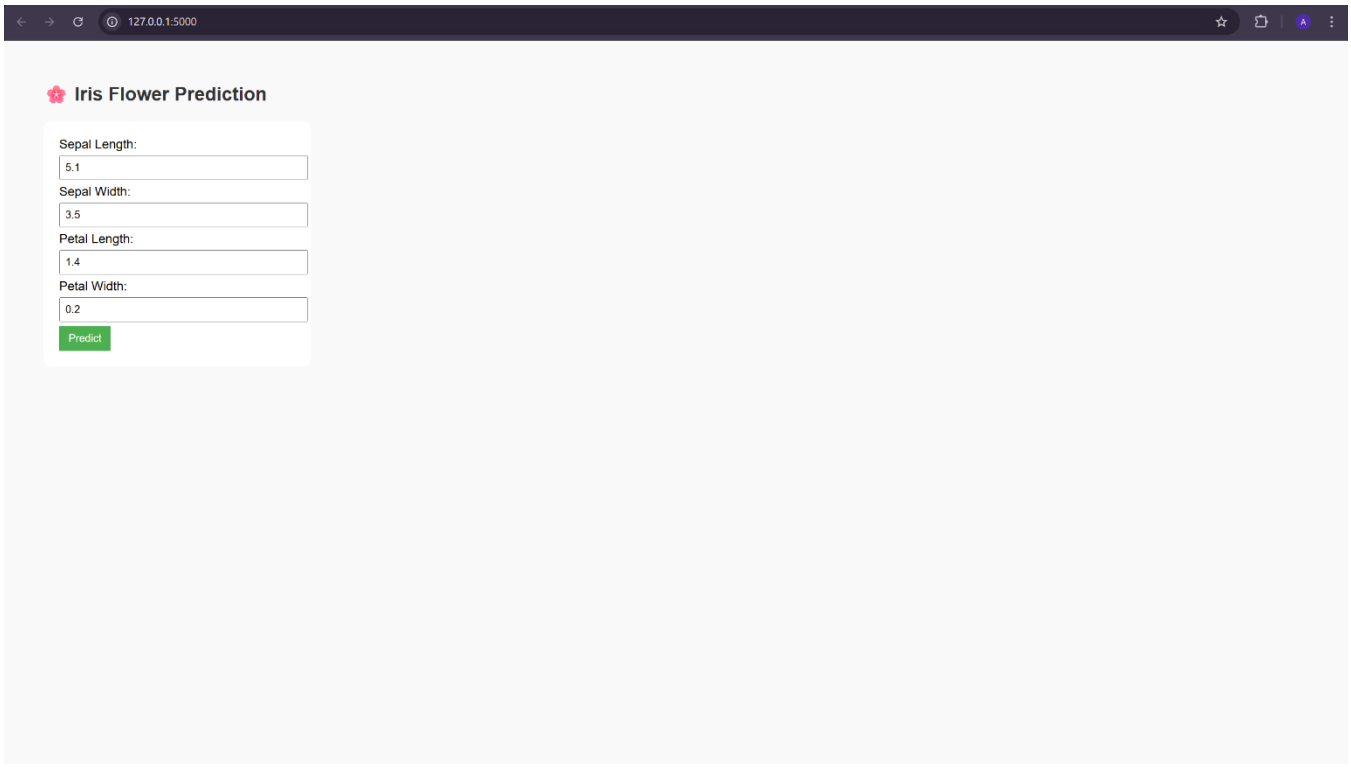
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import joblib
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, test_size=0.2, random_state=42
)
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)
joblib.dump((model, iris.target_names), "iris_model.pkl")
print("Model trained and saved as iris_model.pkl")
```

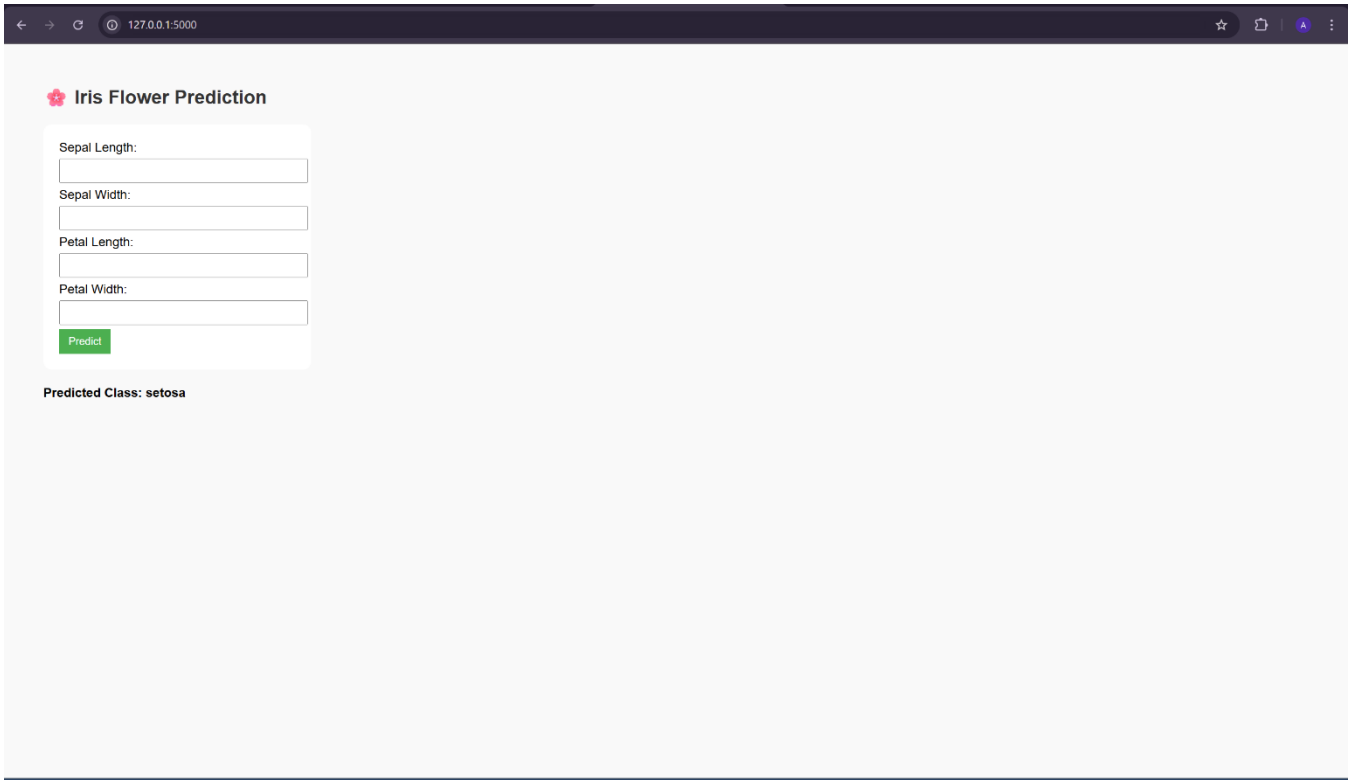
app.py

```
from flask import Flask, render_template_string, request
import joblib
import numpy as np

app = Flask(__name__)
model, target_names = joblib.load("iris_model.pkl")
HTML_TEMPLATE = """
<!DOCTYPE html>
<html>
<head>
    <title>Iris Flower Predictor</title>
</head>
<body>
    <h2>Iris Flower Prediction</h2>
    <form method="POST">
        <label>Sepal Length:</label><input type="number" name="f1" step="any" required><br>
        <label>Sepal Width:</label><input type="number" name="f2" step="any" required><br>
        <label>Petal Length:</label><input type="number" name="f3" step="any" required><br>
        <label>Petal Width:</label><input type="number" name="f4" step="any" required><br>
        <button type="submit">Predict</button>
    </form>
    {% if result %}
        <div class="result">Predicted Class: {{ result }}</div>
```

```
    {% endif %}
</body>
</html>
"""

@app.route("/", methods=["GET", "POST"])
def home():
    result = None
    if request.method == "POST":
        data = [float(request.form['f1']),
                float(request.form['f2']),
                float(request.form['f3']),
                float(request.form['f4'])]
        prediction = model.predict([data])[0]
        result = target_names[prediction]
    return render_template_string(HTML_TEMPLATE, result=result)

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

Output:





RESULT:

The Logistic Regression model was successfully trained and used to classify the iris flower classification. A simple UI that gets the features as input and a simple backend that has endpoints to display the output after prediction.