API project

The goal of this project is to build an API that predicts the species of an iris flower based on its **sepal length, sepal width, petal length, and petal width**. To achieve this, you can run the **"Train_&_save_ML_model.ipynb"** notebook to train and save the model. The code is already complete, so no modifications are needed—the focus is on building the API, not training the model.

Once the notebook runs, download the **"model.pkl"** file, which contains the trained model. This model will map the input features to the predicted flower species.

Now, follow the steps in this project to create an API that loads and runs the model. 🚀

Create your env

Create a github repository

1. Go to GitHub and create a new repository.
2. Click to initialize with a **README**.
3. Copy the repository **URL**.

Clone the repository

Run in the terminal:

```
git clone <your_repository_url>
cd <your_repository_name>
```

Copy the model download from the notebook in the repository

Create an env using venv

python -m venv <name_of_your_env>

Activate your env

Windows:

```
venv\Scripts\activate
```

Mac/Linux:

```
source venv/bin/activate
```

Install the packages

```
pip install fastapi uvicorn pydantic scikit-learn pickle-mixin
```

Control your environment

1. Create a file named `.gitignore`.
2. Inside this file, add `<name_of_your_env>`.
   This ensures that your environment is not pushed to your GitHub repository.
3. Instead of including the environment, we'll specify the required packages and their versions to allow for exact environment recreation anywhere.
4. To list installed packages and their versions, run the following command in your terminal:
   ```
   pip list
   ```
5. Create a file named `requirements.txt`.
6. In this file, add the following dependencies, ensuring each is set to its specific installed version:
   ```
   fastapi==X.XX.X
   uvicorn==X.XX.X
   pydantic==X.XX.X
   scikit-learn==X.XX.X
   pickle-mixin==X.XX.X
   ```
   replace X.XX.X by the version given by the command “pip list”

Create a first version of your API

Create main.py

In your project folder, create a file **`app.py`**

Add the following code to the file:

```
from fastapi import FastAPI

app = FastAPI()
```

Create an input payload using pydantic

from pydantic import BaseModel and create an input payload containing the different information needed by the model (sepal_length, sepal_width, petal_length, petal_width) with their type.

Name the class `InputPayload.`

To better understand how to do this part refer to the documentation.
If you still have questions feel free to ask.

Print the information send by the user

Now we're going to create an endpoint that will accept data from the user, and then send it back as a response. This will be a **POST** request, meaning the data will be sent in the body of the request.

Step-by-step explanation:

1. **Define the route**:
    - You need to define an endpoint that responds to a POST request. This will be done by using a specific decorator provided by FastAPI. This decorator will map the URL path to your function.
    - The route URL you choose will determine where the request is sent (e.g., `/echo`). This URL is where the user will send the data.
2. **Specify the function**:
    - After you've defined the route, you'll write a function that gets triggered when someone sends a POST request to that URL.
    - This function will automatically receive the data sent by the user in the body of the request. FastAPI will take care of turning this raw data into the model you defined earlier (e.g., `InputPayload`).
3. **Return the input payload**:
    - You'll need to send the exact same data back to the user. FastAPI makes it easy to return the data automatically in a format that the user can understand (usually JSON).
    - This response will confirm that the server successfully processed the data and is returning it as is.

Test your code

Start the FastAPI server from your terminal run:

```
uvicorn main:app --reload
```

✅ **API Running at:** `http://127.0.0.1:8000`
✅ **Swagger UI for Testing:** `http://127.0.0.1:8000/docs`

1. Open `http://127.0.0.1:8000/docs`
2. Click **"POST /predict"** → **"Try it out"**
3. Enter sample data:

```
{
  "sepal_length": 5.1,
  "sepal_width": 3.5,
  "petal_length": 1.4,
  "petal_width": 0.2
}
```

4. Click **"Execute"**
   ✅ Response Example:

```
{
  “received_data”:{
  "sepal_length": 5.1,
  "sepal_width": 3.5,
  "petal_length": 1.4,
  "petal_width": 0.2
}
}
```

Insert the model in your API

load the model

Load the model “model.pkl” using the pickle package.

Feel free to help you with this documentation.

Create an output payload using pydantic

Follow the same approach you used for the input payload, but this time, define an output payload that holds the predicted species name.

Use this output model to structure the API response and return the predicted species.

For more details on how to implement this, refer to the documentation. If anything is unclear, don't hesitate to ask!

predict using the input payload

1. Create an API Endpoint with FastAPI

- Define a **POST request** route where users can send flower measurements using the input payload.
- Use of pydantic enforce the expected data format.
- Specify the response model to ensure the API always returns data in the expected format.

2. Process the Input Data

- Convert the received values into a **NumPy array** because the machine learning model requires numerical input in a structured format.
- Ensure that the input is shaped correctly (a 2D array, even if it's just one sample).

3. Make a Prediction

- Use the model load previously to predict the species based on the input data.

4. Map the Prediction to a Human-Readable Format

- The model gives a numerical output (e.g., 0, 1, 2), but users need a species name (e.g., "setosa", "versicolor", "virginica").
- Use a dictionary to map the model's numerical prediction to the corresponding species name.

5. Return the Prediction as JSON

- Construct the response in the format expected by the output model.
- Make sure the response includes the predicted species as a string.

Test your API through the swagger

Start the FastAPI server from your terminal run:

```
uvicorn main:app --reload
```

✅ **API Running at:** `http://127.0.0.1:8000`
✅ **Swagger UI for Testing:** `http://127.0.0.1:8000/docs`

4. Open `http://127.0.0.1:8000/docs`
5. Click **"POST /predict"** → **"Try it out"**
6. Enter sample data:

```
{
  "sepal_length": 5.1,
  "sepal_width": 3.5,
  "petal_length": 1.4,
  "petal_width": 0.2
}
```

5. Click **"Execute"**
   ✅ Response Example:

```
{
  "species": "Setosa"
}
```

Deploy your API

add your new file to your branch

run in your terminal:

```
git add .gitignore
git add main.py
git add model.pkl
git add requirements.txt
```

create your commit

Run in your terminal:

```
git commit -m "Initial FastAPI setup with ML model"
```

push your changes

```
git push origin main
```

Deploy your API using Render

1. Go to Render
2. Sign in
3. Click **New Web Service**
4. Connect to your GitHub repo
5. Select **Python Environment**
6. Set the **Start Command**:
   ```
   uvicorn main:app --host 0.0.0.0 --port 10000
   ```
7. Deploy & get your **public API URL** 🎉

Use your deploy API to predict the flowers species

Replace `<your_render_url>` with your API URL and copy paste into your terminal:

```
curl -X 'POST' '<your_render_url>/predict' \
-H 'Content-Type: application/json' \
-d '{
  "sepal_length": 5.9,
  "sepal_width": 3.0,
  "petal_length": 5.1,
  "petal_width": 1.8
}'
```

✅ Expected Response:

```
{
  "species": "Virginica"
}
```