

DS114- FastAPI Practise

أكاديمية طويق
Tuwaiq Academy



By: Saleh Alsaeed
eng. Esraa Madhi

Getting Started!

1. Setting Up Your Project

- To begin using FastAPI, the first step is to install the required libraries, namely FastAPI itself and Unicorn. You can achieve this by using the following pip command:

```
pip install fastapi uvicorn pydantic
```

- Start by creating a new folder for your project. Within this folder, create a file named `main.py`. This file will contain the initial FastAPI code.
-

2. Writing Your First FastAPI Code

In the `main.py` file, add the following code:

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def root():
    return "Welcome To Tuwaiq Academy"
```

This simple FastAPI application creates an instance of the FastAPI class and defines a single route ("/") that returns a welcome message.

3. Running Your FastAPI Server Locally

To run your FastAPI server locally, open your command line in the same path of the file `main.py` and enter the following command:

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

The `main` refers to the name of your Python file (`main.py`), and `app` is the name of the FastAPI instance in your code. The `--reload` flag enables automatic reloading of the server when changes are made to the code.

- Optional: `--host` and `--port`:

- Specify the host and port on which your FastAPI server will run. For example:

```
uvicorn main:app --reload --host 0.0.0.0 --port 8000
```

- This command runs the server on all available network interfaces (`0.0.0.0`) and sets the port to `8000`.

- Optional: `--workers`:

- Configure the number of worker processes to handle incoming requests. For example:

```
uvicorn main:app --reload --workers 4
```

- This command starts the server with 4 worker processes. The number can be adjusted based on your server's requirements and available resources.

- Optional: `--log-level`:
 - Set the log level for the server. Options include `critical`, `error`, `warning`, `info`, and `debug`. For example:

```
uvicorn main:app --reload --log-level debug
```

- This command sets the log level to `debug`, providing more detailed logs for debugging purposes.

4. Accessing Your FastAPI Application

Open your web browser and navigate to the following URL:

- `http://127.0.0.1:8000` or `http://localhost:8000`

You should see your FastAPI application's welcome message displayed in the browser. This confirms that your FastAPI server is up and running locally.



```
'pretty-print' ☐
```

```
Welcome To Tuwaiq Academy"
```

5. Interactive Documentation:

FastAPI generates interactive documentation automatically, which can be accessed at `/docs` and `/redoc`:

- **Swagger UI (`/docs`):**

- Visit `http://127.0.0.1:8000/docs` or `http://localhost:8000/docs` in your browser.
- You'll see an interactive UI where you can explore your API, send requests, and view responses.
- The interactive documentation is generated based on the types and comments in your code.

FastAPI 0.1.0 OAS 3.1
/openapi.json

default

GET

/ Root

6. Defining Different HTTP Methods to pass input:

```
from fastapi import FastAPI, HTTPException

app = FastAPI()

# GET request
@app.get("/")
def read_root():
    return {"message": "Welcome to Tuwaiq Academy"}

# post request
@app.post("/items/{item_id}")
def create_item(item_id: item_id):
    return {"item": item_id}
```

Accessing Your FastAPI Application

Open your web browser and navigate to the following URL:

- `http://127.0.0.1:8000/items/3` or `http://localhost:8000/items/3`

FastAPI for Machine Learning Deployment

1. Save your model and any scaler:

```
import joblib

joblib.dump(model, 'knn_model.joblib')
joblib.dump(scaler, 'Models/scaler.joblib')
```

Go to the notebook of knn model or any model you built it before and run those 2 lines at the end of the notebook

2. Load Your Model in the API script:

Start by loading your trained machine learning model into your application in `main.py` file.

```
import joblib

model = joblib.load('knn_model.joblib')
scaler = joblib.load('Models/scaler.joblib')
```

3. Preprocessing input data:

Implement functions to preprocess the incoming data into a format your model expects.

- **Step 1:** First, let's create an input for the API that matches the features used in the model. Add the following three blocks of code to the `main.py` file.

```
from pydantic import BaseModel

# Define a Pydantic model for input data validation
# Note: Feel free to change the paramters in this class according to your model input
```

```

class InputFeatures(BaseModel):
    Year: int
    Engine_Size: float
    Mileage: float
    Type: str
    Make: str
    Options: str

```

```

def preprocessing(input_features: InputFeatures):
    """function that applies the same preprocessing steps (used on the training data) to a new test row, ensuring consistency with the training data preprocessing pipeline."""
    dict_f = {
        'Year': input_features.Year,
        'Engine_Size': input_features.Engine_Size,
        'Mileage': input_features.Mileage,
        'Type_Accent': input_features.Type == 'Accent',
        'Type_Land Cruiser': input_features.Type == 'Land Cruiser',
        'Make_Hyundai': input_features.Make == 'Hyundai',
        'Make_Mercedes': input_features.Make == 'Mercedes',
        'Options_Full': input_features.Options == 'Full',
        'Options_Standard': input_features.Options == 'Standard'
    }
    return dict_f

```

```

@app.post("/predict")
def predict(input_features: InputFeatures):

```

```
return preprocessing(input_features)
```

In your terminal, run the following request:

```
curl -X POST "http://localhost:8000/predict" \  
  -H "Content-Type: application/json" \  
  -d '{  
    "Year": 2020,  
    "Engine_Size": 2.5,  
    "Mileage": 15000,  
    "Type": "Accent",  
    "Make": "Hyundai",  
    "Options": "Full"  
  }'
```

- **Step 2:** Update the preprocessing function to include scaling and complete all necessary preparation steps for the input data:

```
def preprocessing(input_features: InputFeatures):  
    dict_f = {  
        'Year': input_features.Year,  
        'Engine_Size': input_features.Engine_Size,  
        'Mileage': input_features.Mileage,  
        'Type_Accent': input_features.Type == 'Accent',  
        'Type_Land Cruiser': input_features.Type == 'Land  
Cruiser',  
        'Make_Hyundai': input_features.Make == 'Hyundai',  
        'Make_Mercedes': input_features.Make == 'Mercede  
s',  
        'Options_Full': input_features.Options == 'Full',  
        'Options_Standard': input_features.Options == 'Sta  
ndard'
```

```

    }

    # Convert dictionary values to a list in the correct order
    features_list = [dict_f[key] for key in sorted(dict_f)]

    # Scale the input features
    scaled_features = scaler.transform([list(dict_f.values
    ())[0]])

    return scaled_features

```

4. **Update Prediction Endpoint:**

Modify the API endpoint to accept input data, process it, and return the predictions generated by your model.

```

@app.post("/predict")
async def predict(input_features: InputFeatures):
    data = preprocessing(input_features)
    y_pred = model.predict(data)
    return {"pred": y_pred.tolist()[0]}

```

In your terminal, run the following request:

```

curl -X POST "http://localhost:8000/predict" \
  -H "Content-Type: application/json" \
  -d '{
    "Year": 2020,
    "Engine_Size": 2.5,
    "Mileage": 15000,
    "Type": "Accent",
    "Make": "Hyundai",
    "Options": "Full"
  }'

```


5. Follow steps in “DS114- Host Your ML Application”

Extra Resources for Deployment into Docker Container:

- <https://towardsdatascience.com/step-by-step-approach-to-build-your-machine-learning-api-using-fast-api-21bd32f2bbdb>
 - https://dev.to/code_jedi/machine-learning-model-deployment-with-fastapi-and-docker-llo
 - <https://engineering.rappi.com/using-fastapi-to-deploy-machine-learning-models-cd5ed7219ea>
-

Resources:

- <https://fastapi.tiangolo.com/tutorial/path-params/>
- https://www.tutorialspoint.com/fastapi/fastapi_rest_architecture.htm
- <https://medium.com/@reza.shokrzad/fastapi-the-modern-toolkit-for-machine-learning-deployment-af31d72b6589>
- <https://www.datacamp.com/tutorial/introduction-fastapi-tutorial>
- <https://dorian599.medium.com/ml-deploy-machine-learning-models-using-fastapi-6ab6aef7e777>