Complete step-by-step guide on creating a **house price prediction AI model** using **Scikit-learn**, training it on CSV data, and then serving it with **FastAPI** so you can pass input and get predictions.

We'll break it down into **three main steps**:

- data preparation & model training,
- saving the model, and
- building a FastAPI API.

Let's set up a **ready-to-run FastAPI project** for your house price prediction. The structure will include create a project folder fastapi and save all files in it :

```
fastapi/
│
├─ house_prices.csv       # Sample training data
├─ train_model.py         # Script to train and save the model
├─ main.py                # FastAPI app for predictions
└─ requirements.txt       # Python dependencies
```

Make sure the CSV exists

1. Create a file named **`house_prices.csv`** in the same folder as `train_model.py` (in your case, `D:\ai\fastapi\`).

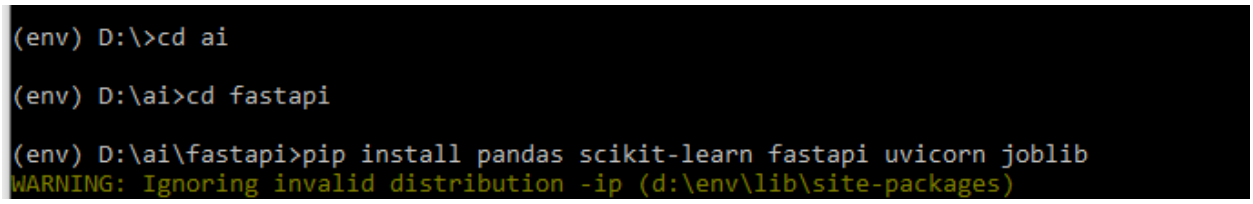
2. Paste the sample data in house_prices.csv :

```
bedrooms,bathrooms,sqft,price
2,1,850,150000
3,2,1200,200000
4,3,2000,350000
3,2,1500,250000
5,4,3000,500000
2,1,900,160000
3,2,1300,220000
4,3,2100,360000
3,2,1400,240000
5,3,2800,480000
```

Step 1: Train the House Price Prediction Model

1. **Install required packages**:

```
pip install pandas scikit-learn fastapi uvicorn joblib
```



2. **Train the model on CSV data** (fastapi/`train_model.py`):

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import joblib

# Step 1: Load your CSV data
data = pd.read_csv("house_prices.csv")  # Make sure your CSV has features &
target

# Step 2: Select features and target
# Assume CSV has columns: ['bedrooms', 'bathrooms', 'sqft', 'location',
'price']
# For simplicity, let's use numeric features only
X = data[['bedrooms', 'bathrooms', 'sqft']]
y = data['price']

# Step 3: Split data
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 = LinearRegression()
model.fit(X_train, y_train)
```

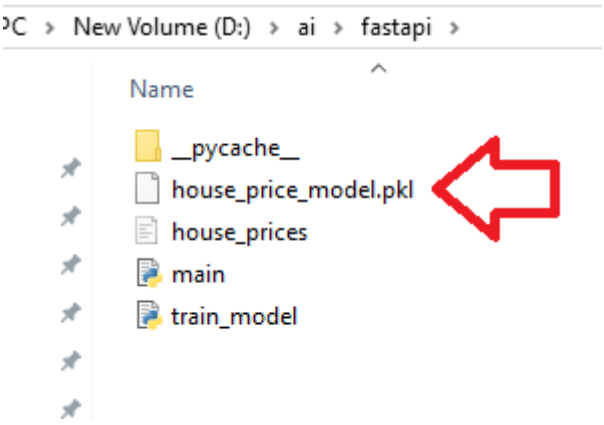
```
# Step 5: Evaluate model
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Step 6: Save model
joblib.dump(model, "house_price_model.pkl")
print("Model saved as house_price_model.pkl")
```

Run file:- python train_model.py

```
(env) D:\ai\fastapi>python train_model.py
Mean Squared Error: 80472418.78653811
Model saved as house_price_model.pkl

(env) D:\ai\fastapi>
```



Step 2: Create FastAPI App create folder fastapi and inside create main.py file :-

1. **Create fastapi/`main.py`:**

```
from fastapi import FastAPI
from pydantic import BaseModel
import joblib

model = joblib.load("house_price_model.pkl")
app = FastAPI(title="House Price Prediction API")

class HouseFeatures(BaseModel):
    bedrooms: int
    bathrooms: float
```

```
    sqft: int

# POST JSON
@app.post("/predict")
def predict_post(features: HouseFeatures):
    input_data = [[features.bedrooms, features.bathrooms, features.sqft]]
    prediction = model.predict(input_data)
    return {"predicted_price": prediction[0]}

# GET query parameters
@app.get("/predict")
def predict_get(bedrooms: int, bathrooms: float, sqft: int):
    input_data = [[bedrooms, bathrooms, sqft]]
    prediction = model.predict(input_data)
    return {"predicted_price": prediction[0]}
```

Step 3: Run the FastAPI server

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

Now your API is running at `http://127.0.0.1:8000`.

You can test it with a POST request using **curl** or **Postman**:

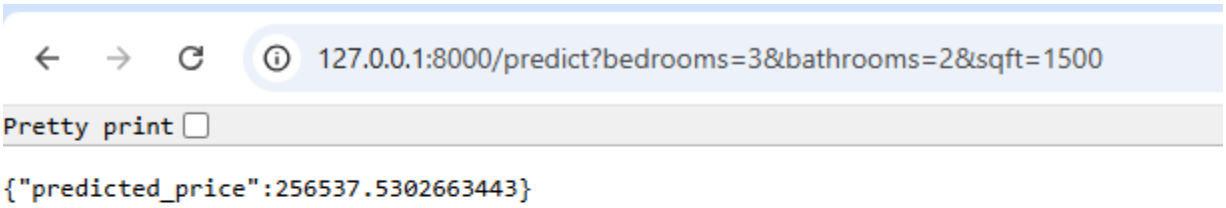
```
curl -X POST "http://127.0.0.1:8000/predict" \
-H "Content-Type: application/json" \
-d '{"bedrooms": 3, "bathrooms": 2, "sqft": 1500}'
```

Expected output:-

```
{
  "predicted_price": 350000.0
}
```

You can test it with Get request using url:-

http://127.0.0.1:8000/predict?bedrooms=3&bathrooms=2&sqft=1500



And you can also use fastapi docs for testing both get andpost request :-

http://127.0.0.1:8000/docs#/