

Integrating a Machine Learning Model with FastAPI – Step-by-Step Guide

1. High-Level Steps

1. Train and validate your ML model using Jupyter Notebook or a Python script.
2. Serialize (save) the trained model using pickle or joblib.
3. Create a FastAPI app that loads the saved model, validates user input, and returns predictions.
4. Add input validation, logging, and error handling.
5. Test the API locally using curl, Postman, or FastAPI docs.
6. Package the app (requirements.txt, Dockerfile) and deploy.
7. Maintain model versioning and monitor performance.

2. Theoretical Explanation

Training and validating ensures your model learns patterns correctly. Serialization avoids retraining each time.

FastAPI allows serving predictions over HTTP APIs. Pydantic validation ensures user inputs are safe.

Pipelines help combine preprocessing and model training together, ensuring consistency.

3. Example: Training and Saving Model

```
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
import joblib

# Load Data
df = pd.read_csv("Data Sheet/house_data.csv")
X = df[["Area_sqft", "Bedrooms", "Bathrooms", "Floors", "Year_Built"]]
y = df["Price"]

# Create Pipeline
pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("lr", LinearRegression())
])
```

```

# Train Model
pipeline.fit(X, y)

# Save Model
joblib.dump(pipeline, "model/house_price_pipeline.joblib")

```

4. Example: FastAPI Application

```

from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, Field
import joblib
import pandas as pd
from fastapi.responses import JSONResponse

model = joblib.load("model/house_price_pipeline.joblib")
app = FastAPI(title="House Price Prediction API")

class UserInput(BaseModel):
    Area_sqft: float = Field(..., gt=0, example=1200.0)
    Bedrooms: int = Field(..., ge=0, example=3)
    Bathrooms: float = Field(..., ge=0, example=2.0)
    Floors: float = Field(..., ge=0, example=1)
    Year_Built: int = Field(..., ge=1800, le=2100, example=2018)

@app.get("/")
def root():
    return {"message": "Welcome to the House Price Prediction API!"}

@app.post("/predict")
def predict_price(data: UserInput):
    try:
        df = pd.DataFrame([data.dict()])
        pred = model.predict(df)
        predicted_price = float(pred[0])
        return JSONResponse(status_code=200, content={
            "predicted_price": round(predicted_price, 2),
            "message": f"The estimated price of the house is ₹{round(predicted_price, 2)}"
        })
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

```

5. How to Run

1. Install dependencies: `pip install fastapi unicorn scikit-learn joblib pandas`
2. Run API: `uvicorn app:app --reload`
3. Open browser: <http://127.0.0.1:8000/docs>
4. Test POST /predict with JSON body like:

```
{  
    "Area_sqft": 1200,  
    "Bedrooms": 3,  
    "Bathrooms": 2,  
    "Floors": 1,  
    "Year_Built": 2019  
}
```

6. Production Tips

- Always save preprocessing + model together in a pipeline.
- Keep model versions (v1, v2, ...).
- Avoid unpickling untrusted files.
- Add try/except for safe error handling.
- Deploy behind Gunicorn/Uvicorn workers for better performance.
- Use logging and monitoring for drift detection.