Flask Income Classifier API — Code Walkthrough & Explanation

This document explains each part of a typical Flask application used for serving a machine learning model. Examples reference the app we are using (routes: '/', '/health', '/predict') and common patterns like loading a model, preprocessing, parsing JSON, and returning predictions.

# 1) Typical Project Structure

A clean structure helps you separate responsibilities and avoid training/inference drift:

project/  
 ├─ app.py # Flask application entry point  
 ├─ preprocessing.py # Functions to clean/transform inputs at inference time  
 ├─ model/  
 │ └─ income\_model.pkl # Trained model or pipeline persisted with joblib  
 ├─ data/ # (Optional) reference data, encoders, lookups  
 ├─ requirements.txt # Python dependencies  
 ├─ test\_payload.json # Example request body for /predict  
 └─ README.md # Setup and usage notes

# 2) Imports — What They Do

from flask import Flask, request, jsonify  
import pandas as pd  
import joblib  
from pathlib import Path  
from preprocessing import preprocess\_data

• Flask: web framework (App object, HTTP request parsing, JSON responses)  
• request: access request body/headers to read incoming JSON  
• jsonify: produce JSON responses with proper mimetype  
• pandas: convert JSON to DataFrame for model input  
• joblib: load trained model or pipeline from disk  
• pathlib.Path: OS-agnostic filesystem paths  
• preprocess\_data: your custom function to transform raw inputs

# 3) App Initialization

app = Flask(\_\_name\_\_)  
BASE\_DIR = Path(\_\_file\_\_).resolve().parent  
MODEL\_PATH = BASE\_DIR / 'model' / 'income\_model.pkl'

• Flask(\_\_name\_\_): creates the web app instance.  
• BASE\_DIR / MODEL\_PATH: robust path building so the model loads regardless of OS/working directory.

# 4) Loading the Model (or Pipeline)

try:  
 model = joblib.load(MODEL\_PATH)  
except Exception as e:  
 model = None # fail safe; /health can report 'model\_not\_loaded'

• Load once at startup for performance.  
• Prefer saving a single Pipeline (preprocessing + model) so inference matches training exactly.

# 5) Configuration Tips

• To return real Unicode (e.g., ≤), disable ASCII escaping:  
 app.config['JSON\_AS\_ASCII'] = False # Flask ≤ 2.2  
 # For Flask 2.3+: app.json.ensure\_ascii = False  
• Use environment variables for secrets/paths in production.  
• Consider enabling logging for debugging and audit.

# 6) Routes — What Each One Does

A) Home ('/'): Simple landing page / docs

@app.route('/', methods=['GET'])  
def home():  
 return ('<h2>Income Classifier API</h2>'  
 '<p>Use POST /predict with JSON or GET /health.</p>'), 200

Purpose: Confirms the server is up and shows basic usage.

B) Health ('/health'): Liveness check

@app.route('/health', methods=['GET'])  
def health():  
 status = 'ok' if model is not None else 'model\_not\_loaded'  
 return jsonify(status=status), (200 if model else 500)

Purpose: Used by you (and by orchestrators) to verify the app and model are ready.

C) Predict ('/predict'): Core inference endpoint

@app.route('/predict', methods=['POST'])  
def predict():  
 # 1) Parse JSON  
 payload = request.get\_json(force=True)  
 # 2) Normalize to DataFrame (accept dict or list[dict])  
 if isinstance(payload, dict):  
 df = pd.DataFrame([payload])  
 elif isinstance(payload, list):  
 df = pd.DataFrame(payload)  
 else:  
 return jsonify(error='Invalid JSON'), 400  
 # 3) Preprocess (or call pipeline)  
 X = preprocess\_data(df) # if model already includes preprocessing, skip this  
 # 4) Predict and format output  
 preds = model.predict(X)  
 proba = model.predict\_proba(X)[:, 1] if hasattr(model, 'predict\_proba') else None  
 label\_map = {0: 'Income <= 50K', 1: 'Income > 50K'}  
 # 5) Build response (single or batch)  
 results = []  
 for i, p in enumerate(preds):  
 item = {'prediction': label\_map.get(int(p), str(p))}  
 if proba is not None:  
 item['probability\_income\_gt\_50k'] = float(proba[i])  
 conf = proba[i] if p == 1 else 1 - proba[i]  
 item['confidence\_percent'] = round(float(conf) \* 100, 2)  
 results.append(item)  
 return jsonify(results[0] if isinstance(payload, dict) else results), 200

Purpose: Accepts raw ACS-style features, transforms them to model inputs, produces predictions and confidence/probabilities. Returns a single object for one record, or a list for batch input.

# 7) Request Parsing & Validation

• request.get\_json(force=True): Parses request body as JSON.  
• Validate schema: ensure required keys exist; reject unknown types or provide defaults.  
• Normalize to DataFrame to match model expectations.

# 8) Preprocessing — Keeping Training & Inference in Sync

• Preprocess raw inputs exactly as during training (encoding, scaling, imputations).  
• Best practice: persist a single sklearn Pipeline (preprocessor + model) and load it at runtime. Then call pipeline.predict\_proba(df) directly to eliminate drift.  
• Handle sentinel values (e.g., -1) and missing data consistently.

# 9) Prediction & Response Formatting

• model.predict(X): class labels.  
• model.predict\_proba(X)[:, 1]: probability of class 'Income > 50K'.  
• Confidence can be derived from probability: p for positive class, 1-p for negative.  
• Use consistent, user-friendly labels; prefer ASCII (<=) or enable UTF-8 responses.

# 10) Error Handling & Logging

• Return clear messages and appropriate status codes:  
 - 400 for bad input (invalid JSON, missing fields)  
 - 500 for server errors (model not loaded, inference failure)  
• Log transformed shapes/columns during debugging to spot schema mismatches quickly.

# 11) Running the App Locally

Activate your virtual environment, then run:  
 python app.py  
  
You should see: Running on http://127.0.0.1:5000  
Health check:  
 curl http://127.0.0.1:5000/health  
Prediction (Windows PowerShell):  
 curl.exe -X POST "http://127.0.0.1:5000/predict" -H "Content-Type: application/json" --data-binary "@test\_payload.json"  
Prediction (macOS/Linux):  
 curl -X POST http://127.0.0.1:5000/predict -H "Content-Type: application/json" --data @test\_payload.json

# 12) Production Notes (WSGI, Security, Performance)

• Use a production WSGI server (gunicorn/uvicorn) behind a reverse proxy (nginx) instead of Flask dev server.  
• Validate and sanitize inputs; enforce timeouts and request size limits.  
• Version your model artifacts; add health and readiness probes; monitor latency and error rates.  
• Consider batching requests for throughput or caching common results.

# 13) Common Pitfalls & How to Avoid Them

• Training/Inference drift: solve by saving a single Pipeline for both.  
• Wrong working directory: use Path(\_\_file\_\_).resolve().parent to build absolute paths.  
• Unicode escaping: set JSON ensure\_ascii=False or use ASCII labels.  
• curl on Windows using PowerShell alias: use curl.exe to force real cURL.  
• Missing required fields: add schema validation and helpful error messages.