# Understanding the AI Invoice Assistant - Code Flow & Explanation

## 1. What This Script Does

This Flask-based script powers an AI Invoice Assistant web application. It enables users to upload invoice images, extract structured data using OCR and a language model, render the data as a table on a webpage, and interact with the data using a chatbot. It integrates modern AI tools to parse real-world invoices intelligently, even when the layout or content is noisy.

## 2. OCR Engine Used and Rationale

The script uses \*\*Doctr (Torch-based)\*\* as the OCR engine, which is preferred over traditional tools like EasyOCR because:  
- Doctr is optimized for deep learning and layout-aware document parsing.  
- It handles multi-column, tabular, and structured invoice formats more effectively.  
- It uses Transformer-based models with strong accuracy.  
  
Other tools considered but not chosen:  
- \*\*EasyOCR\*\*: Simpler but fails with complex document layouts.  
- \*\*Tesseract\*\*: Good for simple text but not layout aware.  
  
Better results can be obtained using paid, enterprise-grade OCR tools like:  
- \*\*Google Cloud Vision API\*\*: Offers intelligent document understanding.  
- \*\*Microsoft Azure LayoutLM\*\*: Built specifically for layout-rich documents.  
These services are not open source but offer superior accuracy and robustness for commercial applications.

## 3. Image Preprocessing: When and Why

Image preprocessing is important for improving OCR accuracy. It becomes essential when dealing with noisy, low-resolution, or skewed documents.  
Common preprocessing steps include grayscale conversion, thresholding, resizing, noise removal, and deskewing.

Sample preprocessing script:

from PIL import Image, ImageFilter, ImageOps  
  
def preprocess\_image(image\_path):  
 img = Image.open(image\_path).convert('L') # Convert to grayscale  
 img = ImageOps.invert(img) # Optional: invert for better contrast  
 img = img.filter(ImageFilter.MedianFilter()) # Denoise  
 img = img.point(lambda x: 0 if x < 140 else 255, '1') # Binarize  
 return img

## 4. Full Working of Code - End-to-End Flow

Here's how the code functions from start to finish:

1. \*\*Image Upload\*\*:  
 - User uploads an invoice image through the UI.  
 - The image is saved locally under `static/uploads/`.  
  
2. \*\*OCR + LLM Parsing\*\*:  
 - The Doctr OCR model reads the text from the uploaded image.  
 - The OCR text is passed to an LLM (via Groq) using a carefully engineered prompt to extract structured JSON data.  
  
3. \*\*JSON Cleaning & Storage\*\*:  
 - The raw LLM response is cleaned and validated to ensure correct JSON.  
 - The JSON is saved to `static/invoice\_data.json` for persistence.  
  
4. \*\*Frontend Rendering\*\*:  
 - The JSON is passed to the HTML template and rendered dynamically into an invoice table.  
 - It supports nested and iterable data like `items`, `buyer`, `shipping\_address`, etc.  
  
5. \*\*Chatbot Functionality\*\*:  
 - The chatbot UI at the bottom-left allows user queries.  
 - On submission, it sends the question and the stored JSON to Groq LLM.  
 - The LLM generates an intelligent answer based on the structured invoice content.  
  
6. \*\*Clear & Reset\*\*:  
 - Users can clear previous data/images to restart the process.

## 5. Additional Suggestions

- You can optionally add logging to debug and trace user queries and OCR outputs.  
- Use a database instead of JSON files for multi-user or production deployment.  
- Add error handling if JSON extraction fails or LLM output is malformed.  
- Compress/resize uploaded images to optimize performance.