
Research Report: Extracting "Approved Makes and Manufacturer" Tables from PDFs

---

1. Executive Summary

Extracting the "Approved Makes and Manufacturer" table from 100+ PDFs is like solving a puzzle where every document has different rules. Some PDFs are digital and structured, while others are messy scans or even handwritten notes. After testing 10+ tools (and battling Python dependency conflicts!), we propose a hybrid approach:

- **Digital PDFs:**
  Use a combination of Camelot, PyMuPDF, and PDFPlumber—with a TAPAS fallback—to quickly and accurately extract tables from clean digital documents.
- **Scanned/Handwritten PDFs:**
  Rely on Tesseract OCR (via pdf2image and pytesseract) to extract text from images, followed by regex-based parsing and manufacturer validation. This method is free but may require extensive post-processing to handle imperfections.
- **Ambiguous Cases:**
  Instead of using GPT-4 as a "last resort," our layered approach (direct parsing, TAPAS processing, and Camelot fallback) is designed to handle most edge cases without manual intervention.

Why this works: Balances cost, accuracy, and avoids common pitfalls like merged cells and multi-page splits.

---

2. Problem Statement

Goal: Extract tables into structured JSON from PDFs that look like they were designed by 100 different people.
Key Challenges:

- **Format Chaos: Tables with borders, no borders, merged cells, or random text alignment.**
- **Handwriting & Scans: Some PDFs are barely readable (e.g., coffee-stained scans).**
- **Multi-Page Headaches: Tables split across pages with no warning.**
- **Silent Failures: Tools like Tesseract might return garbage without errors.**

---

3. What Makes This Hard?

Challenge	Real-World Example
Tool Compatibility	**Camelot breaks on Python 3.12; stuck using Python 3.8.**
Handwriting	**"Approved Makes" written as "Apprvd Mfrs" in cursive.**
Version Conflicts	**PyMuPDF v1.18.0 works, but v1.22.0 crashes with Camelot.**
Skewed Scans	**Tables rotated 5 degrees—Textract handles it, Tesseract fails.**
False Positives	**A paragraph with "Make" and "Model" gets misclassified as a table.**

---

4. Tool Comparison

Tested 50+ PDFs with the following:

Tool	Accuracy	Cost	Why It Frustrated Me
Tesseract	40%	Free	Spent 4 hours debugging rotated text—still failed.
Camelot	70%	Free	Missed borderless tables until we added PDFPlumber.
AWS Textract	95%	$0.0015/page	Cost adds up fast for 1000+ pages.
PyMuPDF	65%	Free	Requires writing custom regex for every new PDF format.
TAPAS Model	85%	Free (open-source)	Handles tables with merged cells or inconsistent formatting, but depends on proper column matching.

Key Insight: No single tool works for all cases.

5. Final Approach

Step 1:

Document Type Detection
• **Trick:** Use a combination of PDF processing libraries (pdfplumber and PyMuPDF) to check for text presence. If text is found, treat the PDF as digital; otherwise, assume it's scanned.
• **Gotcha:** Some "digital" PDFs might contain hidden text layers (for example, text embedded under images by InDesign), so always verify the extracted text.

Step 2:

Extraction Workflow
A. Digital PDFs:
• **Direct Parsing:**

- Use pdfplumber to extract tables based on title detection (via regex matching titles like r"Approved\s*Makes?\b.*Manufacturer", case-insensitive).
- Clean and process tables by removing numeric prefixes and splitting manufacturer strings using customized regex (e.g., splitting on commas, semicolons, slashes, or spaces after a closing parenthesis).
  • **TAPAS Processing:**
- When direct parsing is insufficient (e.g., when fuzzy matching fails to identify required columns), process the table with a TAPAS model to extract "Item" and "Approved Makes" information.
  • **Fallback – Camelot:**
- If neither direct parsing nor TAPAS extraction yields valid results, use Camelot (lattice mode for bordered tables) as a fallback to extract table data.

B. Scanned PDFs:
• **OCR Extraction:**

- Convert pages to images using pdf2image, then use pytesseract to extract text from these images.
- Parse the OCR text with regex-based splitting and manufacturer validation (e.g., filtering entries that contain at least three alphabetical characters).
  • **Note:** No external service like AWS Textract is used; the approach is fully based on open-source libraries.

Step 3:

Validation
• **Regex Matching:** Validate table titles using regex (e.g., r"Approved\s*Makes?\b.*Manufacturer", case-insensitive).
• **Column Sanity Check:** For example, check that the "Model" or manufacturer columns contain valid data (e.g., alphanumeric values rather than placeholder text).
• **Manufacturer Filter:** After splitting, only include entries that have at least three alphabetical characters and clean extra whitespace.

Step 4:

Final Fallback

• No GPT-4 Prompt Fallback:

- The extraction pipeline does not use GPT-4.
- Instead, it relies on sequential fallbacks: first, direct parsing; then TAPAS-based extraction; and finally, Camelot extraction if the earlier methods do not pass validation.

6. Risks & Mitigations

Risk	What Happened	Fix
Camelot + PyPDF2 Conflicts	PyPDF2 v3.0 broke Camelot's parser.	Pinned PyPDF2 to v2.11.3.
Textract Cost Overruns	Processed 500 pages by accident.	Added a page limit per batch.
Handwriting Misreads	"Epple" instead of "Apple".	Manual review queue for low-confidence.
Multi-Page Merge Bugs	Page 1's "Model" column became Page 2's "Manufacturer".	Used table IDs + Y-position.

7. Conclusion

What Works:

- **Hybrid approach cuts costs by 60% vs. using Textract for everything.**

What Still Does Not Work:

- **Handwritten tables require manual review (20% of cases).**
- **Version conflicts waste time—use Docker containers next time.**