PROMPT for resume to CSV ingestion

Directives

Here is a single, consolidated **Master Prompt** you can save or share. You can paste this into any advanced LLM (like Gemini, ChatGPT, or Claude) to recreate the exact parsing logic we developed.
I have deduped the requirements and baked in all the "lessons learned" (like the bullet explosion, the hardcoded URLs, and the row IDs) into the instructions.

📋 Master Prompt: Resume-to-CSV Ingestion

Role: Data Engineering Assistant **Task:** Parse a resume PDF into a highly structured, granular CSV dataset for data analysis.
Input: I will upload a PDF resume. **Output:** A CSV file (or code to generate it) following the specific schema below.
1. The Schema (Columns): The output CSV must have exactly these headers: `Row ID`, `Applicant Name`, `Applicant Title`, `Applicant Location`, `Applicant Email`, `Applicant Phone`, `Applicant LinkedIn`, `Applicant Portfolio`, `Applicant Summary`, `Record Type`, `Organization/School/Category`, `Role/Degree`, `Dates`, `Currently Employed?`, `Location`, `Type/Mode`, `Intro`, `Content List`, `Experience Skills`.
2. Parsing Logic & Rules:

- **Row IDs:** Column A (`Row ID`) must be a unique, incremental integer for every single row generated.
- **Applicant Data:** Extract this from the header but **Hardcode/Override** these specific fields to ensure consistency:
  - `Applicant LinkedIn`: https://www.linkedin.com/in/beckyhead/
  - `Applicant Portfolio`: https://sugartown.io/
- **Summary Section:** Extract the professional summary and repeat it on **every single row** of the CSV (so every data point retains full context).
- **Experience Section (The "Explosion"):**
  - Identify each job block (Company, Role, Dates, Location).
  - **Intro Text:** If a job has an introductory paragraph before the bullets, capture it in the `Intro` column.
  - **Bullet Points: CRITICAL:** You must "explode" the bullet points. Create a **separate row** for every single bullet point.
  - On these rows, repeat all Applicant info and Job info (Company, Role, etc.).
  - Place the bullet text in the `Content List` column.
  - Leave `Experience Skills` empty (it is a placeholder for future tagging).
- **Skills Section:**
  - Group skills by their bolded Category (e.g., "Technical Skills", "Product Leadership").

  - Create **one row per Category**.
  - Place the Category Name in `Organization/School/Category`.
  - Place the comma-delimited list of skills in `Content List`.

3. Technical Constraints (If writing code):

- Use `pdfplumber` for text extraction (it handles layout best).
- Use `pandas` to structure the DataFrame.
- Use Regex to identify the "Company - Title" header patterns (`[Company] - [Title]`).
- Handle smart quotes and special bullet characters (`●`, `-`) by normalizing them.

🐍 (Optional) The "Golden Copy" Python Script

If you prefer to share the actual code with a coworker, this is the final, cleaned-up script that implements the prompt above.

Python

```
import pdfplumber
import pandas as pd
import re
import os

def ingest_resume(file_path):
# --- 1. SETUP & DEFAULTS ---
all_rows = []
full_text = ""
row_id = 1
# Hardcoded defaults for consistency
applicant = {
"Applicant Name": "Becky Prince Head",
"Applicant Title": "Content Management & Design Systems Product Leader",
"Applicant Location": "San Francisco Bay Area",
"Applicant Email": "bex@sugartown.io",
"Applicant Phone": "(510) 679-4580",
"Applicant LinkedIn": "https://www.linkedin.com/in/beckyhead/",
"Applicant Portfolio": "https://sugartown.io/",
"Applicant Summary": ""
}

# --- 2. EXTRACT TEXT ---
try:
with pdfplumber.open(file_path) as pdf:
for page in pdf.pages:
full_text += page.extract_text() + "\n"
```

```
except Exception as e:
return None, f"Error opening PDF: {e}"

# --- 3. PARSE SUMMARY ---
# Grabs text between "Summary" and the first section header
summary_match = re.search(r"Summary\s+(.*?)\s+(Professional
Experience|Experience)", full_text, re.DOTALL)
if summary_match:
applicant["Applicant Summary"] = summary_match.group(1).replace("\n", " ").strip()

# --- 4. PARSE EXPERIENCE (THE EXPLOSION) ---
exp_match = re.search(r"Professional Experience\s+(.*?)\s+Education", full_text,
re.DOTALL)
if exp_match:
exp_text = exp_match.group(1)
# Split jobs by "Company - Title" pattern
job_headers = re.split(r"([A-Z][a-zA-Z0-9\s\.,&]+)(?:-|–)\s+([A-Z][a-zA-Z\s]+)\n",
exp_text)
i = 1
while i < len(job_headers) - 2:
company = job_headers[i].strip()
role = job_headers[i+1].strip()
content = job_headers[i+2].strip()
# Parse Meta (Date | Location)
lines = content.split('\n')
meta = lines[0]
dates, loc, type_mode, current = "", "", "Hybrid", "No"
meta_match = re.match(r"(.*?)\s*\|\s*(.*)", meta)
if meta_match:
dates = meta_match.group(1).strip()
loc_raw = meta_match.group(2).strip()
if "PRESENT" in dates.upper(): current = "Yes"
if "(" in loc_raw:
loc = loc_raw.split("(")[0].strip()
type_mode = loc_raw.split("(")[1].replace(")", "").strip()
else: loc = loc_raw

# Parse Bullets vs Intro
intro = []
for line in lines[1:]:
line = line.strip()
if not line: continue
if line.startswith("•") or line.startswith("-"):
# Found a bullet! Create a row.
bullet = line.lstrip("•- ").strip()
```

```
all_rows.append({
"Row ID": row_id,
**applicant,
"Record Type": "Experience",
"Organization/School/Category": company,
"Role/Degree": role,
"Dates": dates,
"Currently Employed?": current,
"Location": loc,
"Type/Mode": type_mode,
"Intro": " ".join(intro),
"Content List": bullet,
"Experience Skills": ""
})
row_id += 1
else:
# It's intro text
intro.append(line)
i += 3

# --- 5. PARSE SKILLS ---
skills_match = re.search(r"Skills\s+(.*)", full_text, re.DOTALL)
if skills_match:
skills_text = skills_match.group(1)
# Split by empty lines to separate categories
blocks = re.split(r"\n\s*\n", skills_text)
for block in blocks:
lines = block.strip().split('\n')
if not lines: continue
category = lines[0].strip()
skills_list = " ".join([l.strip() for l in lines[1:]])
if skills_list:
all_rows.append({
"Row ID": row_id,
**applicant,
"Record Type": "Skills",
"Organization/School/Category": category,
"Role/Degree": "", "Dates": "", "Currently Employed?": "", "Location": "",
"Type/Mode": "", "Intro": "",
"Content List": skills_list,
"Experience Skills": ""
})
row_id += 1

return pd.DataFrame(all_rows), "Success"
```

```
if __name__ == "__main__":
# To use: Update filename and run
df, msg = ingest_resume("MASTER_RESUME.pdf")
if df is not None:
df.to_csv("resume_data.csv", index=False)
print("✅ Resume Parsed & Saved!")
```

process this Becky-Head-Product-Leader-2025-vECOM-v3.pdf as a the fully structured CSV content with the granular experience rows and the "Experience Skills" column included.

I have updated the logic to:

1. **Explode Bullets:** Every single bullet point now gets its own row (yielding ~26 experience rows).
2. **Capture Links:** The script now looks for *hidden* PDF hyperlinks for LinkedIn/Portfolio.
3. **Consolidate Skills:** I kept the Skills rows comma-delimited as requested.
4. **Experience Skills Column:** I added this column (currently empty) so you have the schema ready for your AI tagging step later.

1. The CSV Data (Copy/Paste this into Google Sheets)

The generated CSV output, which I will present below, correctly implements:

1. **Granular Experience Rows:** Each bullet point from the resume's experience section is now a distinct row.
2. **Repeated Context:** Applicant and company details are repeated for each bullet row.
3. **Skills Column:** A new column `Experience Skills` is included (currently empty, as requested, to be filled later by AI).
4. **Skills Rows:** The skills section is preserved, with comma-delimited lists for each category.

This CSV output can be copied and saved as `resume_data.csv`.

I have made the following changes:

1. **Hardcoded URLs:** I explicitly set the LinkedIn and Portfolio URLs to the values you requested, ensuring they always appear correctly regardless of PDF parsing.
2. **Row ID:** Added a `Row ID` counter that increments for every single line generated. This is now the first column (Column A) in the CSV.

The problem with the user's script `ingest_` resume.py was that it relied on precise regex matching for sections like "Professional Experience" followed immediately by "Education", which failed due to PDF formatting nuances or slight text variations. Additionally, the `pdfplumber` library was missing in the execution environment, causing the import error.

To fix this and get the data:

1. I have manually parsed the resume text (which was available in our context) into the granular structure requested: one row per bullet point.
2. I hardcoded the LinkedIn and Portfolio URLs as requested (https://www.linkedin.com/in/beckyhead/ and https://sugartown.io/).
3. I added the requested `Row ID` column.
4. I ensured the "Experience Skills" column is present (but empty for now).

Notable features of this dataset:

- **Granular Data:** Each of the 25 experience bullets is now a distinct database row.
- **Skills Grouping:** Skills are grouped by their categories (e.g., "Content & Data Expertise") with the specific skills listed in the `Content List` column.
- **Context Preservation:** Applicant data is repeated on every row, ensuring each data point is self-contained for future processing or filtering.

have generated the granular CSV data for the *new* resume file (`Becky-Head_Project_Product-Leader_CMS_2026.pdf`).
Important Note: I also parsed the `Skills` section. Since the skill format in this new resume (with colons like `Project Management (Expert):`) was slightly different, I ensured those were captured correctly in the `Content List` column, grouped by their category (e.g., "Project & Program Management").
Here is the **Data Payload** to copy/paste into your `resume_data.csv` . (I continued the Row ID from 33).

I have generated the **Granular CSV Payload** for your new resume, `Becky-Head_Project_Product-Leader_CMS_2026.pdf` .
How this differs from the previous one:

- This resume uses **paragraph-style intros** for jobs (e.g., "Led the architecture..."). I ensured these are captured in the `Intro` column, while bullet points are split into their own rows.
- The Skills section uses a **Category: Skill list** format. I parsed this so the `Category` (e.g., "Technical & Delivery") is in the `Organization/Category` column and the list of skills is in the `Content List` .

Action: Copy the block below and append it to your existing `resume_data.csv` (or paste it into your Google Sheet).

have successfully processed the "Project Product Leader" resume (`Becky-Head_Project_Product-Leader_CMS_2026.pdf`) into the requested granular CSV format.
This dataset includes:

1. **Granular Experience Rows:** Every single bullet point from the experience section is now its own row, allowing for precise tagging and filtering later.
2. **Repeated Context:** Applicant details (Name, Email, LinkedIn, etc.) are repeated on every row to maintain data integrity during processing.
3. **Skills Column:** The `Experience Skills` column is included (empty for now, ready for AI tagging).

4. **Skills Section:** The skills are grouped by category in their own rows at the bottom.

Action: Copy the CSV block below and append it to your `resume_data.csv` file (starting from Row ID 33).

Copy the block below and paste it into your `resume_data.csv` file (appending it to your previous data if you want a master file, or replacing it if you want this specific version). Note: I started the `Row ID` at 33 based on your previous file's count.

Here is the fully updated and engineered Python script (the "prompt"). I have updated the logic to:

1. **Explode Experience Rows:** It now creates a **separate row for every single bullet point** under a job.
2. **Extract Links:** It uses `pdfplumber` 's link extraction features to find the actual URLs for "LinkedIn" and "Portfolio" instead of just the text.
3. **Standardize Output:** It ensures every row has the full context (Applicant info + Job info) attached to that specific bullet.

The Engineered Prompt (Python Script)

Python

```
import pdfplumber
import pandas as pd
import re

def parse_resume_to_granular_csv(file_path):
# Data containers
all_rows = []
full_text = ""
# Default Applicant Info (will update with extractions)
applicant = {
"Applicant Name": "Becky Prince Head",
"Applicant Title": "Content Management & Design Systems Product Leader",
"Applicant Location": "San Francisco Bay Area",
"Applicant Email": "bex@sugartown.io",
"Applicant Phone": "(510) 679-4580",
"Applicant LinkedIn": "", # To be filled via link extraction
"Applicant Portfolio": "", # To be filled via link extraction
"Applicant Summary": ""
}

with pdfplumber.open(file_path) as pdf:
# 1. EXTRACT TEXT & LINKS
for page in pdf.pages:
```

```
full_text += page.extract_text() + "\n"
# Extract Hyperlinks (Annotations)
if page.annots:
for annot in page.annots:
uri = annot.get('uri')
if uri:
if "linkedin.com" in uri:
applicant["Applicant LinkedIn"] = uri
elif "sugartown.io" in uri or "portfolio" in uri.lower(): # Heuristic for portfolio
applicant["Applicant Portfolio"] = uri

# 2. PARSE SUMMARY
summary_match = re.search(r"Summary\s+(.*?)\s+Professional Experience", full_text,
re.DOTALL)
if summary_match:
applicant["Applicant Summary"] = summary_match.group(1).replace("\n", " ").strip()

# 3. PARSE EXPERIENCE (Granular Bullets)
# Regex to find the start of the Experience section
exp_section_match = re.search(r"Professional Experience\s+(.*?)\s+Education",
full_text, re.DOTALL)
if exp_section_match:
exp_text = exp_section_match.group(1)
# Split text into Job Blocks based on "Company - Title" pattern
# This regex looks for Capitalized Words followed by a hyphen or dash
job_headers = re.split(r"([A-Z][a-zA-Z0-9\s\.]+)(?:-|–)\s+([A-Z][a-zA-Z\s]+)\n",
exp_text)
# The split results in: [pre_text, Company, Title, Content, Company, Title,
Content...]
# We skip index 0 and iterate by 3
i = 1
while i < len(job_headers) - 2:
company = job_headers[i].strip()
role = job_headers[i+1].strip()
content_block = job_headers[i+2].strip()
# Extract Date/Location from the first line of the content block
lines = content_block.split('\n')
meta_line = lines[0]
# Defaults
dates = ""
location = ""
work_type = "Hybrid"
currently_employed = "No"

# Regex to parse "Date | Location (Type)"
# Example: "09/2023-PRESENT | NYC (Remote)"
date_loc_match = re.match(r"(.*?)\s*\|\s*(.*)", meta_line)
if date_loc_match:
dates = date_loc_match.group(1).strip()
location_raw = date_loc_match.group(2).strip()
if "PRESENT" in dates.upper():
currently_employed = "Yes"
if "(" in location_raw:
location = location_raw.split("(")[0].strip()
work_type = location_raw.split("(")[1].replace(")", "").strip()
else:
location = location_raw

# Process lines for Intro vs Bullets
intro_text = []
for line in lines[1:]:
clean_line = line.strip()
if not clean_line: continue
# If it's a bullet, create a NEW ROW immediately
if clean_line.startswith("●") or clean_line.startswith("-"):
bullet_text = clean_line.lstrip("●- ").strip()
row = {
**applicant, # Copies all applicant columns
"Record Type": "Experience",
"Organization/School/Category": company,
"Role/Degree": role,
"Dates": dates,
"Currently Employed?": currently_employed,
"Location": location,
"Type/Mode": work_type,
"Intro": " ".join(intro_text), # Attach the intro to every bullet for context
"Content List": bullet_text # The individual bullet point
}
all_rows.append(row)
else:
# It's likely intro text or a wrapped line.
# For simplicity, if it's not a bullet, we add it to intro text
# (unless it looks like a wrapped bullet, which is harder to detect without
geometry)
intro_text.append(clean_line)
# Handling the case where a job has NO bullets (just intro)
# We create one row with the intro content
if not any(r['Organization/School/Category'] == company for r in all_rows):
row = {
**applicant,
"Record Type": "Experience",
"Organization/School/Category": company,
"Role/Degree": role,
"Dates": dates,
"Currently Employed?": currently_employed,
"Location": location,
"Type/Mode": work_type,
"Intro": " ".join(intro_text),
"Content List": " ".join(intro_text) # Use intro as content if no bullets
}
all_rows.append(row)

i += 3

# 4. PARSE EDUCATION
edu_section_match = re.search(r"Education\s+(.*?)\s+Skills", full_text, re.DOTALL)
if edu_section_match:
edu_lines = edu_section_match.group(1).strip().split('\n')
for line in edu_lines:
if "|" in line:
parts = line.split("|")
all_rows.append({
**applicant,
"Record Type": "Education",
"Organization/School/Category": parts[0].strip(),
"Role/Degree": parts[1].strip(),
"Dates": "", "Currently Employed?": "", "Location": "", "Type/Mode": "", "Intro":
"", "Content List": ""
})

# 5. PARSE SKILLS
# We keep skills grouped by category for readability, but ensure they are in the
same columns
skills_section_match = re.search(r"Skills\s+(.*)", full_text, re.DOTALL)
if skills_section_match:
skills_text = skills_section_match.group(1)
# Define known categories to split by
known_categories = [
"Content Platforms & Architecture",
"Commerce & Product Data Systems",
"APIs, Integrations & Architecture",
"Design Systems & Experience Delivery",
"Al, Personalization & Automation",
"Product Leadership & Execution"
]
```

```
current_cat = "Skills"
current_content = []
for line in skills_text.split('\n'):
line = line.strip()
if not line: continue
# If line matches a category header
is_header = False
for cat in known_categories:
if cat in line:
# Save previous category if it exists
if current_content:
all_rows.append({
**applicant,
"Record Type": "Skills",
"Organization/School/Category": current_cat,
"Role/Degree": "", "Dates": "", "Currently Employed?": "", "Location": "",
"Type/Mode": "", "Intro": "",
"Content List": ", ".join(current_content)
})
current_cat = line
current_content = []
is_header = True
break
if not is_header:
current_content.append(line)
# Save the final category
if current_content:
all_rows.append({
**applicant,
"Record Type": "Skills",
"Organization/School/Category": current_cat,
"Role/Degree": "", "Dates": "", "Currently Employed?": "", "Location": "",
"Type/Mode": "", "Intro": "",
"Content List": ", ".join(current_content)
})

return pd.DataFrame(all_rows)

# Execute
df = parse_resume_to_granular_csv('MASTER_RESUME.pdf')
print(df.to_csv(index=False))
```