# pip install pandas

import re

import pandas as pd

from datetime import datetime

from pathlib import Path

from typing import List, Dict, Any, Tuple, Optional

# -------------------------

# Config: anchors & patterns

# -------------------------

CONFIG = {

"anchors": {

# Claim

"claim\_type": [

"claim for", "jenis tuntutan", "tuntutan untuk",

"inpatient", "hospitalization", "outpatient clinic", "outpatient specialist"

],

"date\_of\_event": ["date of event", "tarikh kejadian", "date of submission", "tarikh serahan", "submission date"],

# Provider

"provider\_type\_panel": ["panel provider", "panel hospital", "panel clinic", "penyedia panel"],

"provider\_type\_nonpanel": ["non panel provider", "non-panel", "bukan panel"],

"provider\_name": ["provider", "hospital", "clinic", "klinik", "medical centre", "medical center", "panel provider"],

# Patient

"patient\_name": ["patient name", "nama pesakit", "name of patient"],

"patient\_id": ["mrn", "medical record no", "no rekod perubatan", "patient id", "no pesakit"],

"nric": ["nric", "ic no", "no kad pengenalan", "kp", "passport", "pasport"],

# Diagnosis

"diagnosis": ["diagnosis", "diagnosa"],

# Invoice

"invoice\_no": ["invoice no", "invoice #", "no. invois", "no invois", "reference no", "ref no", "resit no", "receipt no"],

"total\_amount": ["total receipt/invoice amount", "total amount", "amount due", "jumlah besar", "jumlah keseluruhan", "jumlah dibayar", "grand total"],

"currency": ["rm", "myr", "ringgit"]

},

"items\_header\_keywords": [

"description","item","service","procedure","medication","drug","code","cpt","icd",

"qty","quantity","unit","uom","price","charges","amount","total"

],

"category\_keywords": {

"Consultation": ["consultation","consult","professional fee","doctor fee","gp fee","specialist fee"],

"Medication": ["drug","medication","pharmacy","tablet","capsule","syrup","inj","injection"],

"Procedure": ["procedure","surgery","operation","op","theatre","cpt"],

"Lab": ["lab","laboratory","blood","urine","test","panel"],

"Imaging": ["x-ray","xray","ct","mri","ultrasound","imaging","radiology"],

"Room & Board": ["room","ward","bed","icu","hdu","nursing","board"],

"Others": ["misc","others","supplies","consumables","disposable"]

}

}

# -------------------------

# Regexes

# -------------------------

AMOUNT\_RE = re.compile(r'(?i)\b(?:RM|MYR)?\s\*([0-9]{1,3}(?:,[0-9]{3})\*(?:\.\d{2})|[0-9]+(?:\.\d{2})?)\b')

DATE\_RE = re.compile(

r'(?i)\b('

r'\d{1,2}[\/\-]\d{1,2}[\/\-]\d{2,4}' # 16/09/2025 or 16-09-25

r'|\d{4}[\/\-]\d{1,2}[\/\-]\d{1,2}' # 2025-09-16

r'|\d{1,2}\s+[A-Za-z]{3,9}\s+\d{2,4}' # 16 Sep 2025

r')\b'

)

NRIC\_RE = re.compile(r"\b\d{6}-\d{2}-\d{4}\b")

# -------------------------

# Helpers

# -------------------------

def norm\_space(s: str) -> str:

s = s.replace("\u00A0", " ")

s = re.sub(r"[ \t]+", " ", s)

return s.strip()

def norm\_amount(text: Optional[str]) -> Optional[float]:

if not text: return None

t = text.upper().replace("MYR", "").replace("RM", "").replace(",", "").strip()

try: return round(float(t), 2)

except: return None

def parse\_date(s: str) -> Optional[str]:

s = s.strip()

fmts = ['%d/%m/%Y','%d-%m-%Y','%Y-%m-%d','%d %b %Y','%d %B %Y','%d/%m/%y','%d-%m-%y']

for f in fmts:

try: return datetime.strptime(s, f).date().isoformat()

except: pass

# simple dd/mm/yyyy fallback

if re.match(r"^\d{1,2}/\d{1,2}/\d{2,4}$", s):

d, m, y = s.split('/')

if len(y) == 2: y = str(2000 + int(y))

try: return datetime(int(y), int(m), int(d)).isoformat()

except: return None

return None

def score(source: str) -> float:

# simple confidence proxy (no OCR confidence available)

return {

"anchor\_same\_line": 1.0,

"anchor\_next\_lines": 0.8,

"regex\_anywhere": 0.6,

"inferred": 0.4

}.get(source, 0.3)

def find\_after\_anchor(lines: List[str], i: int) -> Optional[str]:

# text after ":" or "-" on same line, else next non-empty line

parts = re.split(r"[:\-]\s\*", lines[i], maxsplit=1)

if len(parts) == 2 and parts[1].strip(): return parts[1].strip()

for j in range(i+1, min(i+3, len(lines))):

if lines[j].strip(): return lines[j].strip()

return None

def first\_line\_contains(lines: List[str], keywords: List[str], start=0, end=None) -> Optional[int]:

if end is None: end = len(lines)

for i in range(start, end):

ln = lines[i].lower()

if any(k in ln for k in keywords): return i

return None

# -------------------------

# Extraction

# -------------------------

def extract\_header\_medical(lines: List[str]) -> Dict[str, Any]:

hdr = {

# Claim

"claim\_type": None, "claim\_type\_score": 0.0,

"date\_of\_event": None, "date\_of\_event\_score": 0.0,

# Provider

"provider\_type": None, "provider\_type\_score": 0.0,

"provider\_name": None, "provider\_name\_score": 0.0,

# Patient

"patient\_name": None, "patient\_name\_score": 0.0,

"patient\_nric": None, "patient\_nric\_score": 0.0,

# Diagnosis

"diagnosis": None, "diagnosis\_score": 0.0,

# Invoice

"invoice\_no": None, "invoice\_no\_score": 0.0,

"total\_amount": None, "total\_amount\_score": 0.0,

"currency": None

}

joined\_upper = "\n".join(lines).upper()

hdr["currency"] = "MYR" if (" MYR" in " "+joined\_upper or " RM" in " "+joined\_upper) else None

# Claim type: look for explicit options on the page

claim\_candidates = []

for i, ln in enumerate(lines):

l = ln.lower()

if any(k in l for k in CONFIG["anchors"]["claim\_type"]):

if "inpatient" in l or "hospitalization" in l:

claim\_candidates.append(("inpatient/hospitalization", score("anchor\_same\_line")))

if "outpatient clinic" in l:

claim\_candidates.append(("outpatient clinic", score("anchor\_same\_line")))

if "outpatient specialist" in l:

claim\_candidates.append(("outpatient specialist", score("anchor\_same\_line")))

if claim\_candidates:

hdr["claim\_type"], hdr["claim\_type\_score"] = max(claim\_candidates, key=lambda x: x[1])

# Date of event/submission

idx = first\_line\_contains(lines, CONFIG["anchors"]["date\_of\_event"])

if idx is not None:

m = DATE\_RE.search(lines[idx]) or DATE\_RE.search(" ".join(lines[idx:idx+3]))

if m:

hdr["date\_of\_event"] = parse\_date(m.group(1))

hdr["date\_of\_event\_score"] = score("anchor\_next\_lines")

if not hdr["date\_of\_event"]:

# first date anywhere (fallback)

for ln in lines:

m = DATE\_RE.search(ln)

if m:

hdr["date\_of\_event"] = parse\_date(m.group(1))

hdr["date\_of\_event\_score"] = score("regex\_anywhere")

break

# Provider type & name

panel\_idx = first\_line\_contains(lines, CONFIG["anchors"]["provider\_type\_panel"])

nonpanel\_idx = first\_line\_contains(lines, CONFIG["anchors"]["provider\_type\_nonpanel"])

if panel\_idx is not None:

hdr["provider\_type"] = "Panel"

hdr["provider\_type\_score"] = score("anchor\_same\_line")

# try to grab name from same/next lines

cand = find\_after\_anchor(lines, panel\_idx)

if cand and len(cand) > 2:

hdr["provider\_name"], hdr["provider\_name\_score"] = cand, score("anchor\_next\_lines")

if not hdr["provider\_type"] and nonpanel\_idx is not None:

hdr["provider\_type"] = "Non Panel"

hdr["provider\_type\_score"] = score("anchor\_same\_line")

cand = find\_after\_anchor(lines, nonpanel\_idx)

if cand and len(cand) > 2:

hdr["provider\_name"], hdr["provider\_name\_score"] = cand, score("anchor\_next\_lines")

if not hdr["provider\_name"]:

# fallback: first line near top that looks like a facility name

for ln in lines[:10]:

if re.search(r"(?i)hospital|clinic|klinik|medical\s+centre|medical\s+center", ln):

hdr["provider\_name"] = ln.strip()

hdr["provider\_name\_score"] = score("regex\_anywhere")

break

# Patient name

idx = first\_line\_contains(lines, CONFIG["anchors"]["patient\_name"])

if idx is not None:

cand = find\_after\_anchor(lines, idx)

if cand:

# strip obvious labels

cand = re.sub(r"(?i)patient\s\*name\s\*[:\-]?", "", cand).strip()

hdr["patient\_name"], hdr["patient\_name\_score"] = cand, score("anchor\_next\_lines")

# NRIC / Passport

for i, ln in enumerate(lines[:60]):

m = NRIC\_RE.search(ln)

if m:

hdr["patient\_nric"], hdr["patient\_nric\_score"] = m.group(0), score("regex\_anywhere")

break

if any(k in ln.lower() for k in CONFIG["anchors"]["nric"]):

# take token after keyword or next line token

cand = find\_after\_anchor(lines, i)

if cand:

tok = re.search(r"[A-Za-z0-9\-]+", cand)

if tok:

hdr["patient\_nric"], hdr["patient\_nric\_score"] = tok.group(0), score("anchor\_next\_lines")

break

# Diagnosis

idx = first\_line\_contains(lines, CONFIG["anchors"]["diagnosis"])

if idx is not None:

cand = find\_after\_anchor(lines, idx)

if cand:

hdr["diagnosis"], hdr["diagnosis\_score"] = cand, score("anchor\_next\_lines")

# Invoice / Receipt No

idx = first\_line\_contains(lines, CONFIG["anchors"]["invoice\_no"])

if idx is not None:

cand = find\_after\_anchor(lines, idx)

if cand:

tok = re.search(r"[A-Za-z0-9][A-Za-z0-9\-\_\/]+", cand)

if tok:

hdr["invoice\_no"], hdr["invoice\_no\_score"] = tok.group(0), score("anchor\_next\_lines")

# Total Amount (scan bottom-up prioritising total labels)

def pick\_amt(ln: str) -> Optional[float]:

m = AMOUNT\_RE.search(ln)

return norm\_amount(m.group(0)) if m else None

for ln in reversed(lines):

if hdr["total\_amount"] is None and any(k in ln.lower() for k in CONFIG["anchors"]["total\_amount"]):

amt = pick\_amt(ln)

if amt is not None:

hdr["total\_amount"], hdr["total\_amount\_score"] = amt, score("anchor\_same\_line")

break

if hdr["total\_amount"] is None:

# fallback: largest amount on page

amts = [norm\_amount(m.group(0)) for m in AMOUNT\_RE.finditer("\n".join(lines))]

if amts:

hdr["total\_amount"], hdr["total\_amount\_score"] = max(amts), score("regex\_anywhere")

return hdr

# -------------------------

# Line items

# -------------------------

def detect\_items\_region(lines: List[str]) -> Tuple[int, int, List[str]]:

header\_idx = -1

for i, ln in enumerate(lines):

hits = sum(k in ln.lower() for k in CONFIG["items\_header\_keywords"])

if hits >= 2:

header\_idx = i

break

if header\_idx == -1: return -1, -1, []

end\_idx = len(lines)

for j in range(header\_idx+1, len(lines)):

if re.search(r'(?i)\b(sub\s\*total|tax|sst|gst|vat|grand\s\*total|total)\b', lines[j]):

end\_idx = j; break

return header\_idx, end\_idx, lines[header\_idx+1:end\_idx]

def parse\_item\_line(ln: str) -> Dict[str, Any]:

raw = ln.strip()

if not raw: return {}

# split on tabs or 2+ spaces

parts = [p.strip() for p in re.split(r"\t+| {2,}", raw) if p.strip()]

desc, qty, unit\_price, line\_total = None, None, None, None

# Identify numeric tokens

amounts = [p for p in parts if AMOUNT\_RE.search(p)]

nums = [p for p in parts if re.fullmatch(r"\d+(?:\.\d+)?", p)]

# Heuristic: last amount-like → total, previous amount-like → unit; first pure number → qty

if amounts:

line\_total = norm\_amount(amounts[-1])

if len(amounts) >= 2:

unit\_price = norm\_amount(amounts[-2])

if nums:

try: qty = float(nums[0])

except: pass

# Description: everything before first numeric token

first\_num\_idx = next((i for i,p in enumerate(parts) if AMOUNT\_RE.search(p) or re.fullmatch(r"\d+(?:\.\d+)?", p)), None)

if first\_num\_idx is None:

desc = parts[0] if parts else None

else:

desc = " ".join(parts[:first\_num\_idx]).strip() or (parts[0] if parts else None)

if unit\_price is not None and qty is not None and line\_total is None:

line\_total = round(unit\_price \* qty, 2)

# Category (coarse)

category = "Others"

if desc:

d = desc.lower()

best, best\_hits = "Others", 0

for cat, keys in CONFIG["category\_keywords"].items():

hits = sum(1 for k in keys if k in d)

if hits > best\_hits:

best, best\_hits = cat, hits

category = best

return {

"description": desc or None,

"category": category,

"qty": qty,

"unit\_price": unit\_price,

"line\_total": line\_total,

"raw": raw

}

# -------------------------

# Public API

# -------------------------

def extract\_from\_text(ocr\_text: str) -> Tuple[pd.DataFrame, pd.DataFrame, Dict[str, Any]]:

text = "\n".join(norm\_space(l) for l in ocr\_text.splitlines() if l.strip())

lines = text.splitlines()

header = extract\_header\_medical(lines)

hi, ti, item\_lines = detect\_items\_region(lines)

items = []

for ln in item\_lines:

rec = parse\_item\_line(ln)

if any(v is not None for k, v in rec.items() if k != "raw"):

items.append(rec)

# Reconcile totals: if missing, use sum of lines

if items and header.get("total\_amount") is None:

header["total\_amount"] = round(sum([(x.get("line\_total") or 0.0) for x in items]), 2)

header["total\_amount\_score"] = score("inferred")

header\_df = pd.DataFrame([{

"claim\_type": header["claim\_type"],

"date\_of\_event": header["date\_of\_event"],

"provider\_type": header["provider\_type"],

"provider\_name": header["provider\_name"],

"patient\_name": header["patient\_name"],

"patient\_nric": header["patient\_nric"],

"diagnosis": header["diagnosis"],

"invoice\_no": header["invoice\_no"],

"total\_amount": header["total\_amount"],

"currency": header["currency"],

# (optional) scores

"claim\_type\_score": header["claim\_type\_score"],

"date\_of\_event\_score": header["date\_of\_event\_score"],

"provider\_type\_score": header["provider\_type\_score"],

"provider\_name\_score": header["provider\_name\_score"],

"patient\_name\_score": header["patient\_name\_score"],

"patient\_nric\_score": header["patient\_nric\_score"],

"diagnosis\_score": header["diagnosis\_score"],

"invoice\_no\_score": header["invoice\_no\_score"],

"total\_amount\_score": header["total\_amount\_score"],

}])

line\_df = pd.DataFrame(items, columns=["description","category","qty","unit\_price","line\_total","raw"])

if not line\_df.empty:

line\_df.insert(0, "line\_no", range(1, len(line\_df)+1))

debug = {"items\_region": (hi, ti)}

return header\_df, line\_df, debug

def extract\_from\_txt(path: str):

text = Path(path).read\_text(encoding="utf-8", errors="ignore")

return extract\_from\_text(text)

# Single file

header\_df, line\_df, debug = extract\_from\_txt("your\_invoice\_ocr.txt")

print(header\_df)

print(line\_df.head())

# Batch a folder

from pathlib import Path

in\_dir = Path("folder\_of\_txt")

all\_h, all\_l = [], []

for p in sorted(in\_dir.glob("\*.txt")):

h, l, \_ = extract\_from\_txt(str(p))

h.insert(0, "source\_file", p.name)

if not l.empty: l.insert(0, "source\_file", p.name)

all\_h.append(h); all\_l.append(l)

headers = pd.concat(all\_h, ignore\_index=True) if all\_h else pd.DataFrame()

lines = pd.concat(all\_l, ignore\_index=True) if all\_l else pd.DataFrame()

headers.to\_csv("headers.csv", index=False)

lines.to\_csv("line\_items.csv", index=False)