```
!pip install reportlab pdfplumber pandas --quiet

import os
from datetime import datetime, timedelta
import pandas as pd
from reportlab.lib.pagesizes import A4
from reportlab.pdfgen import canvas
import pdfplumber
import random
import json

# Create folders
os.makedirs("/content/rfps", exist_ok=True)
os.makedirs("/content/data", exist_ok=True)

print("Folders created: /content/rfps  and /content/data")
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.6/43.6 kB 887.2 kB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 67.8/67.8 kB 4.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 28.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.0/60.0 kB 3.3 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.6/5.6 MB 73.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.0/3.0 MB 70.0 MB/s eta 0:00:00
Folders created: /content/rfps  and /content/data
```

Create mock RFP Pdfs (each pdf contains clear metadata fields)

```
def create_mock_rfp_pdf(text_lines, out_path):
    c = canvas.Canvas(out_path, pagesize=A4)
    y = 800
    for line in text_lines:
        c.drawString(40, y, line)
        y -= 16
        if y < 60:
            c.showPage()
            y = 800
    c.showPage()
    c.save()
    return out_path

def make_cable_rfp(rfp_id, client, project, cable_type, conductor, size, insulation, voltage, armour, length_m,
    lines = [
        f"RFP_ID: {rfp_id}",
        f"Client: {client}",
        f"Project: {project}",
        f"Category: Wires & Cables",
        f"Cable_Type: {cable_type}",
        f"Conductor: {conductor}",
        f"Size: {size} sqmm",
        f"Insulation: {insulation}",
        f"Voltage_Rating: {voltage}",
        f"Armour: {armour}",
        f"Length_m: {length_m}",
        f"Standards: {standards}",
        f"Due_Date: {due_date.strftime('%Y-%m-%d')}",
        f"Estimated_Value: {est_value}",
        "",
        "Technical_Specifications:",
        " - Conductor resistance, DC measurements per meter",
        " - Insulation resistance > 1000 MΩ",
        " - Manufacturer to provide test certificates: HV test, IR test, Tensile test",
        "",
        "Delivery_Location: Onsite",
        "Notes: Submit lead time, test plan and commercial terms."
    ]
    fname = os.path.join(path, f"rfp_{rfp_id}.pdf")
    return create_mock_rfp_pdf(lines, fname)

def make_fmeg_rfp(rfp_id, client, project, product, qty, unit, specs, due_date, est_value, path):
    lines = [
        f"RFP_ID: {rfp_id}",
        f"Client: {client}",
        f"Project: {project}",
        f"Category: FMEG",
        f"Product: {product}",
        f"Quantity: {qty} {unit}",
```

```
        f"Specs: {specs}",
        f"Due_Date: {due_date.strftime('%Y-%m-%d')}",
        f"Estimated_Value: {est_value}",
        "",
        "Commercial_Notes: Warranty required, delivery on-site, include installation services where applicable."
        "",
        "Tests_Certification: BIS/CE if applicable."
    ]
    fname = os.path.join(path, f"rfp_{rfp_id}.pdf")
    return create_mock_rfp_pdf(lines, fname)
```

```
rfps_meta = []
today = datetime.today()

# Cable RFP examples (3)
cable_examples = [
    {"cable_type":"3C+E XLPE Power Cable","conductor":"Copper","size":240,"insulation":"XLPE","voltage":"1.1kV",
    {"cable_type":"1C XLPE HT Cable","conductor":"Aluminium","size":400,"insulation":"XLPE","voltage":"11kV","ar
    {"cable_type":"Control Cable 4C PVC","conductor":"Copper","size":4,"insulation":"PVC","voltage":"1.1kV","arm
]

for i, ex in enumerate(cable_examples, start=1):
    rfp_id = f"CAB{i:03d}"
    client = "LargeIndustrialClient"
    project = f"Infrastructure_Project_{100+i}"
    due_date = today + timedelta(days=14 + i*5)
    est_value = int(ex["length_m"] * (200 if ex["size"]<50 else 600))  # rough per-meter estimate
    path = make_cable_rfp(rfp_id, client, project, ex["cable_type"], ex["conductor"], ex["size"], ex["insulation
    rfps_meta.append({
        "rfp_id": rfp_id,
        "client": client,
        "category": "Wires & Cables",
        "project": project,
        "due_date": due_date.strftime("%Y-%m-%d"),
        "est_value": est_value,
        "file_path": path
    })

# FMEG RFP examples (2)
fmeg_examples = [
    {"product":"Industrial LED Floodlight 200W","qty":150,"unit":"nos","specs":"IP66, 200W, 6500K, Lumens>26000"
    {"product":"Distribution Board 3-phase 36 way","qty":25,"unit":"nos","specs":"Metal enclosure, powder-coated
]

for j, ex in enumerate(fmeg_examples, start=1):
    rfp_id = f"FMEG{j:03d}"
    client = "LargeIndustrialClient"
    project = f"Factory_Expansion_{200+j}"
    due_date = today + timedelta(days=10 + j*7)
    est_value = ex["qty"] * (800 if "LED" in ex["product"] else 15000)
    path = make_fmeg_rfp(rfp_id, client, project, ex["product"], ex["qty"], ex["unit"], ex["specs"], due_date, e
    rfps_meta.append({
        "rfp_id": rfp_id,
        "client": client,
        "category": "FMEG",
        "project": project,
        "due_date": due_date.strftime("%Y-%m-%d"),
        "est_value": est_value,
        "file_path": path
    })

print("Created RFP PDFs:")
for r in rfps_meta:
    print("-", r["rfp_id"], r["category"], r["file_path"])
```

```
Created RFP PDFs:
- CAB001 Wires & Cables /content/rfps/rfp_CAB001.pdf
- CAB002 Wires & Cables /content/rfps/rfp_CAB002.pdf
- CAB003 Wires & Cables /content/rfps/rfp_CAB003.pdf
- FMEG001 FMEG /content/rfps/rfp_FMEG001.pdf
- FMEG002 FMEG /content/rfps/rfp_FMEG002.pdf
```

Parse pdfs (pdfplumber) and create rfp_meta.csv with raw_text saved separately

```
def parse_pdf_text(path):
    txt = ""
    with pdfplumber.open(path) as pdf:
```

```
        for p in pdf.pages:
            page_txt = p.extract_text()
            if page_txt:
                txt += page_txt + "\n"
    return txt

parsed = []
for r in rfps_meta:
    raw = parse_pdf_text(r["file_path"])
    entry = r.copy()
    entry["raw_text"] = raw
    parsed.append(entry)

# Save metadata CSV (excluding raw_text) and JSON for raw text
df_meta = pd.DataFrame([{k:v for k,v in e.items() if k!="raw_text"} for e in parsed])
df_meta.to_csv("/content/data/rfp_meta.csv", index=False)

with open("/content/data/rfps_raw.json", "w") as f:
    json.dump({e["rfp_id"]: e["raw_text"] for e in parsed}, f, indent=2)

print("Saved /content/data/rfp_meta.csv and /content/data/rfps_raw.json")
df_meta
```

```
Saved /content/data/rfp_meta.csv and /content/data/rfps_raw.json
```

	rfp_id	client	category	project	due_date	est_value	file_path
0	CAB001	LargeIndustrialClient	Wires & Cables	Infrastructure_Project_101	2026-01-05	1500000	/content/rfps/rfp_CAB001.pdf
1	CAB002	LargeIndustrialClient	Wires & Cables	Infrastructure_Project_102	2026-01-10	720000	/content/rfps/rfp_CAB002.pdf
2	CAB003	LargeIndustrialClient	Wires & Cables	Infrastructure_Project_103	2026-01-15	1600000	/content/rfps/rfp_CAB003.pdf

Next steps: Generate code with df_meta | New interactive sheet

Create product catalog, price_db, stock_db, test_costs, historical_rfps tailored to cables + FMEG

```
products = [
    # Cables (per-meter)
    {"sku_id":"CU_XLPE_3C_240_AR","name":"CU XLPE 3C 240sqmm Armoured","category":"Cable","unit":"m","spec":"Cop
    {"sku_id":"AL_XLPE_1C_400_AR","name":"AL XLPE 1C 400sqmm Armoured","category":"Cable","unit":"m","spec":"Alu
    {"sku_id":"CU_PVC_4C_4_UN","name":"CU PVC Control 4C 4sqmm","category":"Cable","unit":"m","spec":"Copper con
    # FMEG (unit)
    {"sku_id":"LED_FLD_200W","name":"Industrial LED Floodlight 200W","category":"FMEG","unit":"nos","spec":"IP66
    {"sku_id":"DB_3PH_36WAY","name":"Distribution Board 3ph 36way","category":"FMEG","unit":"nos","spec":"Metal
]
pd.DataFrame(products).to_csv("/content/data/products.csv", index=False)

price_db = [
    {"sku_id":"CU_XLPE_3C_240_AR","price_per_unit":650, "unit":"m","margin_band":"mid"},
    {"sku_id":"AL_XLPE_1C_400_AR","price_per_unit":1200, "unit":"m","margin_band":"high"},
    {"sku_id":"CU_PVC_4C_4_UN","price_per_unit":40, "unit":"m","margin_band":"low"},
    {"sku_id":"LED_FLD_200W","price_per_unit":8000, "unit":"nos","margin_band":"mid"},
    {"sku_id":"DB_3PH_36WAY","price_per_unit":15000, "unit":"nos","margin_band":"high"}
]
pd.DataFrame(price_db).to_csv("/content/data/price_db.csv", index=False)

stock_db = [
    {"sku_id":"CU_XLPE_3C_240_AR","current_stock_m":5000,"lead_time_days":21},
    {"sku_id":"AL_XLPE_1C_400_AR","current_stock_m":800,"lead_time_days":45},
    {"sku_id":"CU_PVC_4C_4_UN","current_stock_m":20000,"lead_time_days":5},
    {"sku_id":"LED_FLD_200W","current_stock_units":300,"lead_time_days":7},
    {"sku_id":"DB_3PH_36WAY","current_stock_units":40,"lead_time_days":30}
]
pd.DataFrame(stock_db).to_csv("/content/data/stock_db.csv", index=False)

test_costs = [
    {"test_type":"HV_Test","cost_per_sample":1500},
    {"test_type":"Insulation_Resistance","cost_per_sample":600},
    {"test_type":"Conductor_Resistance","cost_per_sample":400},
    {"test_type":"Tensile_Test","cost_per_sample":800},
    {"test_type":"Lighting_Flm_Test","cost_per_sample":200}
]
pd.DataFrame(test_costs).to_csv("/content/data/test_costs.csv", index=False)

historical = [
```

```
    {"rfp_id":"H_CAB_001","category":"Wires & Cables","value":250000,"status":"win","specmatch_avg":0.85,"price_
    {"rfp_id":"H_FMEG_001","category":"FMEG","value":45000,"status":"loss","specmatch_avg":0.6,"price_aggressive
]
pd.DataFrame(historical).to_csv("/content/data/historical_rfps.csv", index=False)

print("Saved product + price + stock + test + historical CSVs to /content/data")
print(os.listdir("/content/data"))
```

```
Saved product + price + stock + test + historical CSVs to /content/data
['rfp_meta.csv', 'stock_db.csv', 'test_costs.csv', 'price_db.csv', 'rfps_raw.json', 'products.csv', 'historical_r
```

Quick sanity-check display

```
import pandas as pd
print("RFP meta:\n")
display(pd.read_csv("/content/data/rfp_meta.csv"))
print("\nProducts sample:\n")
display(pd.read_csv("/content/data/products.csv").head())
```

```
RFP meta:
```

	rfp_id	client	category	project	due_date	est_value	file_path
0	CAB001	LargeIndustrialClient	Wires & Cables	Infrastructure_Project_101	2026-01-05	1500000	/content/rfps/rfp_CAB001.pdf
1	CAB002	LargeIndustrialClient	Wires & Cables	Infrastructure_Project_102	2026-01-10	720000	/content/rfps/rfp_CAB002.pdf
2	CAB003	LargeIndustrialClient	Wires & Cables	Infrastructure_Project_103	2026-01-15	1600000	/content/rfps/rfp_CAB003.pdf
3	FMEG001	LargeIndustrialClient	FMEG	Factory_Expansion_201	2026-01-03	120000	/content/rfps/rfp_FMEG001.pdf
4	FMEG002	LargeIndustrialClient	FMEG	Factory_Expansion_202	2026-01-10	375000	/content/rfps/rfp_FMEG002.pdf

```
Products sample:
```

	sku_id	name	category	unit	spec
0	CU_XLPE_3C_240_AR	CU XLPE 3C 240sqmm Armoured	Cable	m	Copper conductor, XLPE, Armoured, 1.1kV
1	AL_XLPE_1C_400_AR	AL XLPE 1C 400sqmm Armoured	Cable	m	Aluminium conductor, XLPE, 11kV, Armoured

SALES AGENT

```
# Cell 1 — imports & load data
import re
import json
import math
from datetime import datetime
import pandas as pd
import numpy as np
from pathlib import Path

DATA_DIR = Path("/content/data")
RFPS_DIR = Path("/content/rfps")

rfp_meta = pd.read_csv(DATA_DIR / "rfp_meta.csv")
with open(DATA_DIR / "rfps_raw.json", "r") as f:
    rfps_raw = json.load(f)

products = pd.read_csv(DATA_DIR / "products.csv")
price_db = pd.read_csv(DATA_DIR / "price_db.csv")
stock_db = pd.read_csv(DATA_DIR / "stock_db.csv")

print("Loaded:", list(DATA_DIR.iterdir()))
rfp_meta
```

```
Loaded: [PosixPath('/content/data/rfp_meta.csv'), PosixPath('/content/data/stock_db.csv'), PosixPath('/content/da
```

	rfp_id	client	category	project	due_date	est_value	file_path
0	CAB001	LargeIndustrialClient	Wires & Cables	Infrastructure_Project_101	2026-01-05	1500000	/content/rfps/rfp_CAB001.pdf
1	CAB002	LargeIndustrialClient	Wires & Cables	Infrastructure_Project_102	2026-01-10	720000	/content/rfps/rfp_CAB002.pdf
2	CAB003	LargeIndustrialClient	Wires & Cables	Infrastructure_Project_103	2026-01-15	1600000	/content/rfps/rfp_CAB003.pdf
3	FMEG001	LargeIndustrialClient	FMEG	Factory_Expansion_201	2026-01-	120000	/content/rfps/rfp_FMEG001.pdf

Next steps: Generate code with rfp_meta | New interactive sheet

Rule-based extractor for wires/cables + FMEG fields

```
def extract_field(pattern, text, cast=None, first=True, flags=re.IGNORECASE):
    m = re.findall(pattern, text, flags)
    if not m:
        return None
    val = m[0] if first else m
    if isinstance(val, tuple):
        val = " ".join([v for v in val if v])
    if cast and val is not None:
        try:
            return cast(val)
        except:
            return val
    return val

def extract_rfp_metadata(rfp_id, raw_text):
    txt = raw_text.replace("\n", " \n ")
    meta = {"rfp_id": rfp_id, "raw_text": raw_text}
    # Generic fields
    meta["category"] = extract_field(r"Category:\s*([A-Za-z &]+)", txt)
    meta["client"] = extract_field(r"Client:\s*([A-Za-z0-9_ \-]+)", txt)
    meta["project"] = extract_field(r"Project:\s*([A-Za-z0-9_ \-]+)", txt)
    meta["due_date"] = extract_field(r"Due_Date:\s*([0-9]{4}-[0-9]{2}-[0-9]{2})", txt)
    meta["estimated_value"] = extract_field(r"Estimated_Value:\s*([0-9,]+)", txt, lambda s: int(s.replace(",",""
    # Wires & Cables fields
    meta["cable_type"] = extract_field(r"Cable_Type:\s*([A-Za-z0-9\+\- _/]+)", txt)
    meta["conductor"] = extract_field(r"Conductor:\s*([A-Za-z]+)", txt)
    meta["size_sqmm"] = extract_field(r"Size:\s*([0-9]+)\s*sqmm", txt, lambda s: int(s))
    meta["insulation"] = extract_field(r"Insulation:\s*([A-Za-z0-9]+)", txt)
    meta["voltage"] = extract_field(r"Voltage_Rating:\s*([0-9\.kKVv]+)", txt)
    meta["armour"] = extract_field(r"Armour:\s*([A-Za-z]+)", txt)
    meta["length_m"] = extract_field(r"Length_m:\s*([0-9]+)", txt, lambda s: int(s))
    meta["standards"] = extract_field(r"Standards:\s*([A-Za-z0-9 ,/]+)", txt)
    # FMEG fields
    meta["product"] = extract_field(r"Product:\s*([A-Za-z0-9 \-\,\(\)\/]+)", txt)
    qty = extract_field(r"Quantity:\s*([0-9]+)\s*([a-zA-Z]+)?", txt)
    if isinstance(qty, tuple):
        try:
            meta["quantity"] = int(qty[0])
            meta["quantity_unit"] = qty[1] if qty[1] else None
        except:
            meta["quantity"] = None
            meta["quantity_unit"] = None
    else:
        meta["quantity"] = qty
        meta["quantity_unit"] = None
    meta["specs"] = extract_field(r"Specs:\s*(.+?)Due_Date:|Specs:\s*(.+?)Commercial_Notes:|Technical_Specificat
    # Clean specs: join list if returned as list
    if isinstance(meta["specs"], list):
        meta["specs"] = max(meta["specs"], key=len)  # pick longest match
    # Fallbacks: use category as from rfp_meta if missing
    if not meta["category"]:
        # try reading from rfp_meta table
        try:
            cat = rfp_meta.loc[rfp_meta.rfp_id == rfp_id, "category"].values
            if len(cat)>0:
                meta["category"] = cat[0]
        except:
            meta["category"] = None
    # parse due_date to datetime and compute days_to_due
    try:
        meta["due_date_dt"] = datetime.strptime(meta["due_date"], "%Y-%m-%d")
```

```
        meta["days_to_due"] = (meta["due_date_dt"] - datetime.today()).days
    except:
        meta["due_date_dt"] = None
        meta["days_to_due"] = None
    return meta

# Test extraction on all RFPs
extracted = []
for _, row in rfp_meta.iterrows():
    rfp_id = row["rfp_id"]
    raw = rfps_raw.get(rfp_id, "")
    ex = extract_rfp_metadata(rfp_id, raw)
    extracted.append(ex)

df_ex = pd.DataFrame(extracted)
df_ex.head(10)
```

	rfp_id	raw_text	category	client	project	due_date	estimated_value	cable_ty
0	CAB001	RFP_ID: CAB001\nClient: LargeIndustrialClient\...	Wires & Cables	LargeIndustrialClient	Infrastructure_Project_101	2026-01-05	1500000	3C+E XL Power Ca
1	CAB002	RFP_ID: CAB002\nClient: LargeIndustrialClient\...	Wires & Cables	LargeIndustrialClient	Infrastructure_Project_102	2026-01-10	720000	1C XLPE Ca
2	CAB003	RFP_ID: CAB003\nClient: LargeIndustrialClient\...	Wires & Cables	LargeIndustrialClient	Infrastructure_Project_103	2026-01-15	1600000	Cor Cable F
3	FMEG001	RFP_ID: FMEG001\nClient: LargeIndustrialClient...	FMEG	LargeIndustrialClient	Factory_Expansion_201	2026-01-03	120000	N
4	FMEG002	RFP_ID: FMEG002\nClient: LargeIndustrialClient...	FMEG	LargeIndustrialClient	Factory_Expansion_202	2026-01-10	375000	N

5 rows × 21 columns

```
# REPLACEMENT Cell 3 — robust product-fit matcher & stock feasibility (with debug checks)
import re
import pandas as pd
import math
import traceback

# Make local safe copy of stock_db and products (these should already be loaded)
_stock_db = stock_db.copy() if 'stock_db' in globals() else pd.DataFrame()
_products = products.copy() if 'products' in globals() else pd.DataFrame()
_extracted = extracted if 'extracted' in globals() else []

print("DEBUG: starting Cell 3 checks")
print(" - stock_db columns:", list(_stock_db.columns))
print(" - products columns:", list(_products.columns))
print(" - number of extracted RFPs:", len(_extracted))
if len(_stock_db)>0:
    display(_stock_db.head())
if len(_products)>0:
    display(_products.head())

# Ensure sku_id column exists and is string
if "sku_id" in _stock_db.columns:
    _stock_db["sku_id"] = _stock_db["sku_id"].astype(str)
else:
    # Try to find candidate column
    possible = [c for c in _stock_db.columns if "sku" in c.lower() or "id" in c.lower()]
    if possible:
        _stock_db["sku_id"] = _stock_db[possible[0]].astype(str)
    else:
        _stock_db["sku_id"] = _stock_db.index.astype(str)

# Coerce numeric-like columns to numeric but avoid touching sku_id
for col in _stock_db.columns:
    if col == "sku_id":
```

```
            continue
        if any(k in col.lower() for k in ["stock", "lead_time", "leadtime", "units", "_m", "days", "price", "cost"]):
            _stock_db[col] = pd.to_numeric(_stock_db[col], errors="coerce")

    def _to_number_safe(x):
        if x is None:
            return None
        if isinstance(x, (int, float)):
            if math.isnan(x):
                return None
            return x
        s = str(x).strip()
        if s == "":
            return None
        try:
            return int(s)
        except:
            try:
                return float(s)
            except:
                return None

    def match_product_for_rfp(meta, products_df):
        try:
            text_candidates = []
            cat = str(meta.get("category","")).lower()
            if cat and "cable" in cat:
                parts = [meta.get(k) for k in ["cable_type","conductor","size_sqmm","insulation","armour"] if meta.g
                text_candidates.append(" ".join([str(p) for p in parts]))
            if meta.get("product"):
                text_candidates.append(str(meta.get("product")))
            if meta.get("specs"):
                # specs can be list or string; pick string
                s = meta.get("specs")
                if isinstance(s, list):
                    s = max(s, key=len) if s else ""
                text_candidates.append(str(s))
            candidate_text = " | ".join(text_candidates).lower()
            best = {"sku": None, "score": 0.0}
            # Defensive: ensure products_df has sku_id and name
            if "sku_id" not in products_df.columns:
                products_df = products_df.copy()
                products_df["sku_id"] = products_df.index.astype(str)
            for _, p in products_df.iterrows():
                name_spec = (str(p.get("name","")) + " " + str(p.get("spec",""))).lower()
                tokens_r = set(re.findall(r"[a-z0-9]+", candidate_text))
                tokens_p = set(re.findall(r"[a-z0-9]+", name_spec))
                s = 0.0
                if tokens_r:
                    s = len(tokens_r & tokens_p) / float(max(1, len(tokens_r)))
                if s > best["score"]:
                    best = {"sku": p["sku_id"], "score": float(s)}
            return best["sku"], best["score"]
        except Exception as e:
            print("match_product_for_rfp error:", e)
            traceback.print_exc()
            return None, 0.0

    def check_stock_feasibility(meta, sku, stock_df):
        try:
            if sku is None:
                return False, 0.0
            df = stock_df.copy()
            df["sku_id"] = df["sku_id"].astype(str)
            row = df[df["sku_id"] == str(sku)]
            if row.empty:
                # fallback: contains (safe conversion)
                try:
                    mask = df["sku_id"].str.contains(str(sku), case=False, na=False)
                    row = df[mask]
                except Exception:
                    row = pd.DataFrame()
            if row.empty:
                # no matching sku in stock DB
                return False, 0.0
            row = row.iloc[0].to_dict()
            # Cable length check
            if "current_stock_m" in row and meta.get("length_m") is not None:
                available = _to_number_safe(row.get("current_stock_m"))
                required = _to_number_safe(meta.get("length_m"))
                if available is None or required is None:
                    return True, 0.5
```

```
            feasible = bool(available >= required)
            feas_ratio = min(1.0, float(available) / max(1, float(required))) if required > 0 else 1.0
            return feasible, float(feas_ratio)
        # Unit check
        if "current_stock_units" in row and meta.get("quantity") is not None:
            available = _to_number_safe(row.get("current_stock_units"))
            required = _to_number_safe(meta.get("quantity"))
            if available is None or required is None:
                return True, 0.5
            feasible = bool(available >= required)
            feas_ratio = min(1.0, float(available) / max(1, float(required))) if required > 0 else 1.0
            return feasible, float(feas_ratio)
        # else unknown
        return True, 0.5
    except Exception as e:
        print("check_stock_feasibility error:", e)
        traceback.print_exc()
        return False, 0.0

# Run matching & feasibility and produce df_matches; show debug info
matches = []
for meta in _extracted:
    sku, fit_score = match_product_for_rfp(meta, _products)
    feasible, feas_ratio = check_stock_feasibility(meta, sku, _stock_db)
    matches.append({
        "rfp_id": meta.get("rfp_id"),
        "matched_sku": sku,
        "fit_score": float(fit_score),
        "feasible": bool(feasible),
        "feas_ratio": float(feas_ratio)
    })

df_matches = pd.DataFrame(matches)
print("\n=== df_matches ===")
display(df_matches)

print("\nDEBUG: types in stock_db after coercion:")
print(_stock_db.dtypes.to_dict())

# quick checks for problematic rows
bad = []
for _, r in df_matches.iterrows():
    if r["matched_sku"] is None:
        bad.append((r["rfp_id"], "no_sku_matched"))
    if r["feas_ratio"] == 0.0 and not r["feasible"]:
        bad.append((r["rfp_id"], "not_feasible"))
if bad:
    print("\nWARNING: issues detected for the following RFPs:")
    for b in bad:
        print(" -", b)
else:
    print("\nAll matches produced; no immediate obvious issues.")

# Save matches to variable used downstream
df_matches = df_matches
```

```
DEBUG: starting Cell 3 checks
 - stock_db columns: ['sku_id', 'current_stock_m', 'lead_time_days', 'current_stock_units']
 - products columns: ['sku_id', 'name', 'category', 'unit', 'spec']
 - number of extracted RFPs: 5
```

	sku_id	current_stock_m	lead_time_days	current_stock_units
0	CU_XLPE_3C_240_AR	5000.0	21	NaN
1	AL_XLPE_1C_400_AR	800.0	45	NaN
2	CU_PVC_4C_4_UN	20000.0	5	NaN
3	LED_FLD_200W	NaN	7	300.0
4	DB_3PH_36WAY	NaN	30	40.0

	sku_id	name	category	unit	spec
0	CU_XLPE_3C_240_AR	CU XLPE 3C 240sqmm Armoured	Cable	m	Copper conductor, XLPE, Armoured, 1.1kV
1	AL_XLPE_1C_400_AR	AL XLPE 1C 400sqmm Armoured	Cable	m	Aluminium conductor, XLPE, 11kV, Armoured
2	CU_PVC_4C_4_UN	CU PVC Control 4C 4sqmm	Cable	m	Copper conductor, PVC, Control cable
3	LED_FLD_200W	Industrial LED Floodlight 200W	FMEG	nos	IP66, 200W, 6500K
4	DB_3PH_36WAY	Distribution Board 3ph 36way	FMEG	nos	Metal enclosure with MCBs

```
=== df_matches ===
```

	rfp_id	matched_sku	fit_score	feasible	feas_ratio
0	CAB001	CU_XLPE_3C_240_AR	0.357143	True	1.000000
1	CAB002	AL_XLPE_1C_400_AR	0.384615	False	0.666667
2	CAB003	CU_PVC_4C_4_UN	0.461538	True	1.000000
3	FMEG001	LED_FLD_200W	1.000000	True	0.500000
4	FMEG002	DB_3PH_36WAY	0.333333	True	0.500000

```
DEBUG: types in stock_db after coercion:
{'sku_id': dtype('O'), 'current_stock_m': dtype('float64'), 'lead_time_days': dtype('int64'), 'current_stock_unit

All matches produced; no immediate obvious issues.
```

Next steps: Generate code with df_matches | New interactive sheet

```python
# Cell 4 — scoring function and produce candidate_rfps.csv
# score components (normalized to 0–1):
# - value_score (log scale normalized)
# - urgency_score (days to due; closer = higher)
# - fit_score (from product match)
# - feasibility_score (from stock check)

def normalize_log_value(v, min_v=1000, max_v=1_000_000):
    if v is None or math.isnan(v):
        return 0.0
    v = max(min_v, min(max_v, v))
    return (math.log(v) - math.log(min_v)) / (math.log(max_v) - math.log(min_v))

def urgency_from_days(days, cap_days=90):
    if days is None:
        return 0.0
    d = max(-30, min(cap_days, days))  # clamp
    # closer deadlines => higher score
    return max(0.0, 1.0 - (d / cap_days))

# weights
W_VALUE = 0.40
W_URGENCY = 0.25
W_FIT = 0.25
W_FEAS = 0.10

rows = []
for meta in extracted:
    mid = meta["rfp_id"]
    mmatch = df_matches[df_matches.rfp_id==mid].iloc[0]
    est_val = meta.get("estimated_value") or 0
    val_s = normalize_log_value(est_val)
    urg_s = urgency_from_days(meta.get("days_to_due"))
    fit_s = float(mmatch["fit_score"])
    feas_s = float(mmatch["feas_ratio"])
    final_score = W_VALUE*val_s + W_URGENCY*urg_s + W_FIT*fit_s + W_FEAS*feas_s
    # recommendations shorthand
    rec = "Proceed" if final_score >= 0.45 and mmatch["feasible"] else ("Consider" if final_score>=0.35 else "Sk
```

```
    rows.append({
        "rfp_id": mid,
        "client": meta.get("client"),
        "category": meta.get("category"),
        "est_value": est_val,
        "days_to_due": meta.get("days_to_due"),
        "matched_sku": mmatch["matched_sku"],
        "fit_score": round(fit_s,3),
        "feasible": bool(mmatch["feasible"]),
        "feas_ratio": round(feas_s,3),
        "value_score": round(val_s,3),
        "urgency_score": round(urg_s,3),
        "final_score": round(final_score,3),
        "recommendation": rec
    })

candidates_df = pd.DataFrame(rows).sort_values("final_score", ascending=False)
candidates_df.to_csv(DATA_DIR / "candidate_rfps.csv", index=False)
print("Saved /content/data/candidate_rfps.csv")
candidates_df
```

```
Saved /content/data/candidate_rfps.csv
```

	rfp_id	client	category	est_value	days_to_due	matched_sku	fit_score	feasible	feas_ratio
0	CAB001	LargeIndustrialClient	Wires & Cables	1500000	18	CU_XLPE_3C_240_AR	0.357	True	1.000
2	CAB003	LargeIndustrialClient	Wires & Cables	1600000	28	CU_PVC_4C_4_UN	0.462	True	1.000
3	FMEG001	LargeIndustrialClient	FMEG	120000	16	LED_FLD_200W	1.000	True	0.500
1	CAB002	LargeIndustrialClient	Wires & Cables	720000	23	AL_XLPE_1C_400_AR	0.385	False	0.667
4	FMEG002	LargeIndustrialClient	FMEG	375000	23	DB_3PH_36WAY	0.333	True	0.500

Next steps: Generate code with candidates_df | New interactive sheet

```
# Cell 5 — human-friendly summarizer for each candidate RFP
def summarize_rfp(rfp_id, candidates_df, extracted_list, products_df, price_db_df):
    row = candidates_df[candidates_df.rfp_id==rfp_id]
    if row.empty:
        print("RFP not found:", rfp_id)
        return
    row = row.iloc[0].to_dict()
    meta = next((m for m in extracted_list if m["rfp_id"]==rfp_id), {})
    print("=== RFP SUMMARY:", rfp_id, "===")
    print(f"Client: {row['client']} | Category: {row['category']} | Est. Value: ₹{row['est_value']}")
    print(f"Due in {row['days_to_due']} days | Recommendation: {row['recommendation']} (score {row['final_score'}
    print("\n-- Technical snapshot --")
    if meta.get("category") and "cable" in str(meta.get("category")).lower():
        print("Cable Type:", meta.get("cable_type"))
        print("Conductor / Size / Insulation:", meta.get("conductor"), "/", meta.get("size_sqmm"), "sqmm /", met
        print("Length (m):", meta.get("length_m"), "| Standards:", meta.get("standards"))
    if meta.get("product"):
        print("Product:", meta.get("product"), "| Qty:", meta.get("quantity"), meta.get("quantity_unit"))
        print("Specs:", meta.get("specs") if meta.get("specs") else "-")
    print("\n-- Commercial snapshot --")
    sku = row.get("matched_sku")
    if pd.notna(sku):
        p = price_db_df[price_db_df.sku_id==sku]
        if not p.empty:
            p = p.iloc[0].to_dict()
            print("Matched SKU:", sku, "| Price unit:", p.get("price_per_unit"), p.get("unit"))
    print("Fit score:", row["fit_score"], "| Stock feasibility ratio:", row["feas_ratio"])
    # short suggested next actions
    if row["recommendation"] == "Proceed":
        print("\nNext actions: 1) Prepare technical compliance sheet; 2) Draft commercial (per unit pricing + le
    elif row["recommendation"] == "Consider":
        print("\nNext actions: 1) Check alternate SKUs / subcontracting for shortage; 2) Re-check margin require
    else:
        print("\nNext actions: 1) Skip or low-effort bid (if strategic), 2) Re-evaluate if client requests clari
    print("====================================\n")

# Print summaries for all candidates (ordered)
for r in candidates_df.rfp_id.tolist():
    summarize_rfp(r, candidates_df, extracted, products, price_db)
```

```
Cable Type: Control Cable 4C PVC
Conductor / Size / Insulation: Copper / 4 sqmm / PVC
Length (m): 8000 | Standards: IS 694

-- Commercial snapshot --
Matched SKU: CU_PVC_4C_4_UN | Price unit: 40 m
Fit score: 0.462 | Stock feasibility ratio: 1.0

Next actions: 1) Prepare technical compliance sheet; 2) Draft commercial (per unit pricing + lead time); 3) Bool
====================================

=== RFP SUMMARY: FMEG001 ===
Client: LargeIndustrialClient  | Category: FMEG  | Est. Value: ₹120000
Due in 16 days | Recommendation: Proceed (score 0.783)

-- Technical snapshot --
Product: Industrial LED Floodlight 200W  | Qty: 150 nos None
Specs: -

-- Commercial snapshot --
Matched SKU: LED_FLD_200W | Price unit: 8000 nos
Fit score: 1.0 | Stock feasibility ratio: 0.5

Next actions: 1) Prepare technical compliance sheet; 2) Draft commercial (per unit pricing + lead time); 3) Bool
====================================

=== RFP SUMMARY: CAB002 ===
Client: LargeIndustrialClient  | Category: Wires & Cables  | Est. Value: ₹720000
Due in 23 days | Recommendation: Consider (score 0.73)

-- Technical snapshot --
Cable Type: 1C XLPE HT Cable
Conductor / Size / Insulation: Aluminium / 400 sqmm / XLPE
Length (m): 1200 | Standards: IEC 60502

-- Commercial snapshot --
Matched SKU: AL_XLPE_1C_400_AR | Price unit: 1200 m
Fit score: 0.385 | Stock feasibility ratio: 0.667

Next actions: 1) Check alternate SKUs / subcontracting for shortage; 2) Re-check margin requirements.
====================================

=== RFP SUMMARY: FMEG002 ===
Client: LargeIndustrialClient  | Category: FMEG  | Est. Value: ₹375000
Due in 23 days | Recommendation: Proceed (score 0.663)

-- Technical snapshot --
Product: Distribution Board 3-phase 36 way  | Qty: 25 nos None
Specs: -

-- Commercial snapshot --
Matched SKU: DB_3PH_36WAY | Price unit: 15000 nos
Fit score: 0.333 | Stock feasibility ratio: 0.5

Next actions: 1) Prepare technical compliance sheet; 2) Draft commercial (per unit pricing + lead time); 3) Bool
====================================
```

TUNING SCORING WEIGHTS

```
# Install the embedding model package (sentence-transformers)
!pip install -q sentence-transformers

# Imports
from sentence_transformers import SentenceTransformer, util
import numpy as np
import pandas as pd
import math
from pathlib import Path
from datetime import datetime
import json

DATA_DIR = Path("/content/data")
RFPS_DIR = Path("/content/rfps")

# load previously created artifacts (assumes Step 1 ran)
rfp_meta = pd.read_csv(DATA_DIR / "rfp_meta.csv")
with open(DATA_DIR / "rfps_raw.json", "r") as f:
    rfps_raw = json.load(f)

products = pd.read_csv(DATA_DIR / "products.csv")
price_db = pd.read_csv(DATA_DIR / "price_db.csv")
stock_db = pd.read_csv(DATA_DIR / "stock_db.csv")
# optional suppliers file (if not present, we'll create a small sample)
if (DATA_DIR / "suppliers_stock.csv").exists():
```

```
    suppliers = pd.read_csv(DATA_DIR / "suppliers_stock.csv")
else:
    # create a small sample supplier DB for purchase orders fallback
    suppliers = pd.DataFrame([
        {"supplier_id":"SUP1","sku_id":"CU_XLPE_3C_240_AR","available_m":10000,"lead_time_days":28,"price_per_m"
        {"supplier_id":"SUP2","sku_id":"AL_XLPE_1C_400_AR","available_m":3000,"lead_time_days":45,"price_per_m":
        {"supplier_id":"SUP3","sku_id":"CU_PVC_4C_4_UN","available_m":50000,"lead_time_days":10,"price_per_m":35
        {"supplier_id":"SUP4","sku_id":"LED_FLD_200W","available_units":200,"lead_time_days":14,"price_per_unit"
    ])
    suppliers.to_csv(DATA_DIR / "suppliers_stock.csv", index=False)
    print("Created sample suppliers DB at /content/data/suppliers_stock.csv")

print("Loaded datasets. Products:", len(products), "Stock rows:", len(stock_db), "Suppliers:", len(suppliers))
```

```
WARNING:torchao.kernel.intmm:Warning: Detected no triton, on systems without Triton certain kernels will not work
Created sample suppliers DB at /content/data/suppliers_stock.csv
Loaded datasets. Products: 5 Stock rows: 5 Suppliers: 4
```

```
# TUNEABLES (change these values as per business priority)
W_VALUE = 0.35     # importance of contract value
W_URGENCY = 0.20   # deadline urgency
W_FIT = 0.30       # semantic/spec fit
W_FEAS = 0.15      # stock + lead time feasibility

# Feasibility options
ALLOW_SUBCONTRACT = True   # if True, check suppliers to satisfy shortage
LEADTIME_SENSITIVITY = 1.0 # multiplier controlling how strongly leadtime affects feasibility (higher -> more pe

# Embedding model choice (small & fast — change if you want larger)
EMBEDDING_MODEL = "all-MiniLM-L6-v2"
print("Weights:", W_VALUE, W_URGENCY, W_FIT, W_FEAS)
print("ALLOW_SUBCONTRACT:", ALLOW_SUBCONTRACT, "LEADTIME_SENSITIVITY:", LEADTIME_SENSITIVITY)
```

```
Weights: 0.35 0.2 0.3 0.15
ALLOW_SUBCONTRACT: True LEADTIME_SENSITIVITY: 1.0
```

Build embeddings for product catalog

```
# Load embedding model
model = SentenceTransformer(EMBEDDING_MODEL)

# Build product text (name + spec)
def product_text_row(row):
    pieces = [str(row.get("name","")), str(row.get("spec","")), str(row.get("category",""))]
    return " | ".join([p for p in pieces if p and p.lower() != "nan"])

products["product_text"] = products.apply(product_text_row, axis=1)

# Compute embeddings (cache to disk to avoid re-computation if re-running)
emb_path = DATA_DIR / f"product_embeddings_{EMBEDDING_MODEL}.npz"
meta_path = DATA_DIR / f"product_ids_{EMBEDDING_MODEL}.csv"

if emb_path.exists() and meta_path.exists():
    # load
    emb = np.load(emb_path)["arr"]
    product_ids = pd.read_csv(meta_path)
    if list(product_ids.sku_id) == list(products.sku_id):
        product_embeddings = emb
        print("Loaded cached product embeddings.")
    else:
        print("Product list changed; recomputing embeddings.")
        product_embeddings = model.encode(products["product_text"].tolist(), convert_to_numpy=True, show_progres
        np.savez_compressed(emb_path, product_embeddings)
        products.to_csv(meta_path, index=False)
else:
    product_embeddings = model.encode(products["product_text"].tolist(), convert_to_numpy=True, show_progress_ba
    np.savez_compressed(emb_path, product_embeddings)
    products.to_csv(meta_path, index=False)
print("Product embeddings shape:", product_embeddings.shape)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/t
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
modules.json: 100%                                                  349/349 [00:00<00:00, 6.25kB/s]
config_sentence_transformers.json: 100%                                  116/116 [00:00<00:00, 9.58kB/s]
README.md:       10.5k/? [00:00<00:00, 528kB/s]
sentence_bert_config.json: 100%                                  53.0/53.0 [00:00<00:00, 6.14kB/s]
config.json: 100%                                                612/612 [00:00<00:00, 68.6kB/s]
model.safetensors: 100%                                          90.9M/90.9M [00:04<00:00, 23.1MB/s]
tokenizer_config.json: 100%                                      350/350 [00:00<00:00, 32.2kB/s]
vocab.txt:       232k/? [00:00<00:00, 5.37MB/s]
tokenizer.json:       466k/? [00:00<00:00, 17.7MB/s]
special_tokens_map.json: 100%                                    112/112 [00:00<00:00, 9.84kB/s]
config.json: 100%                                                190/190 [00:00<00:00, 17.0kB/s]
Batches: 100%                                                    1/1 [00:00<00:00,  3.71it/s]
Product embeddings shape: (5, 384)
```

Semantic matcher + enhanced feasibility

```
from math import log

# Helper: build candidate text from RFP meta (same approach as earlier but combine fields)
def build_rfp_candidate_text(meta):
    parts = []
    if meta.get("category"): parts.append(str(meta.get("category")))
    if meta.get("cable_type"): parts.append(str(meta.get("cable_type")))
    if meta.get("conductor"): parts.append(str(meta.get("conductor")))
    if meta.get("size_sqmm"): parts.append(f"{meta.get('size_sqmm')}sqmm")
    if meta.get("product"): parts.append(str(meta.get("product")))
    if meta.get("specs"):
        s = meta.get("specs")
        if isinstance(s, list): s = max(s, key=len) if s else ""
        parts.append(str(s))
    # include short raw_text if nothing else
    if not parts and meta.get("raw_text"):
        parts.append(meta.get("raw_text")[:512])
    return " | ".join(parts)

# Preprocess stock & suppliers to numeric safe forms
stock_db_local = stock_db.copy()
if "sku_id" in stock_db_local.columns:
    stock_db_local["sku_id"] = stock_db_local["sku_id"].astype(str)
# ensure numeric columns
for c in stock_db_local.columns:
    if c != "sku_id":
        try:
            stock_db_local[c] = pd.to_numeric(stock_db_local[c], errors="coerce")
        except:
            pass

suppliers_local = suppliers.copy()
if "sku_id" in suppliers_local.columns:
    suppliers_local["sku_id"] = suppliers_local["sku_id"].astype(str)
# numeric coercion for supplier numeric columns
for c in suppliers_local.columns:
    if c not in ["sku_id","supplier_id"]:
        try:
            suppliers_local[c] = pd.to_numeric(suppliers_local[c], errors="coerce")
        except:
            pass

# Enhanced match function: semantic similarity + token overlap fallback
def semantic_match(meta, products_df, product_embs, model, top_k=3):
    candidate_text = build_rfp_candidate_text(meta)
    if not candidate_text.strip():
        return None, 0.0, []
    # compute embedding for candidate
    q_emb = model.encode(candidate_text, convert_to_numpy=True)
```

```
    # cosine similarities with product_embs (numpy)
    cos_sims = util.cos_sim(q_emb, product_embs)[0]  # returns tensor; but util.cos_sim gives torch/tensor, cast
    if hasattr(cos_sims, "cpu"):
        cos_arr = cos_sims.cpu().numpy()
    else:
        cos_arr = np.array(cos_sims)
    # top match
    top_idx = int(np.argmax(cos_arr))
    top_s = float(cos_arr[top_idx])
    top_sku = products_df.iloc[top_idx]["sku_id"]
    # also return top_k candidates for fallback & explainability
    topk_idx = np.argsort(-cos_arr)[:top_k]
    topk = [(products_df.iloc[int(i)]["sku_id"], float(cos_arr[int(i)])) for i in topk_idx]
    return top_sku, top_s, topk

# Enhanced feasibility: stock + lead time + suppliers fallback
def enhanced_feasibility(meta, sku, stock_df, suppliers_df, days_to_due, allow_subcontract=True, leadtime_sensit
    # default unknown
    feas_score = 0.0
    feasible = False
    reason = []
    required_units = None
    # Determine required amount
    if meta.get("category") and "cable" in meta.get("category").lower():
        required_units = int(meta.get("length_m")) if meta.get("length_m") is not None else None
        stock_col = "current_stock_m"
    else:
        required_units = int(meta.get("quantity")) if meta.get("quantity") is not None else None
        stock_col = "current_stock_units"

    # find stock row
    stock_row = None
    if sku is not None:
        stock_row_df = stock_df[stock_df["sku_id"] == str(sku)]
        if not stock_row_df.empty:
            stock_row = stock_row_df.iloc[0].to_dict()

    # if we have stock info and required_units
    if stock_row and required_units is not None:
        available = stock_row.get(stock_col) if stock_col in stock_row else None
        lead_time_days = stock_row.get("lead_time_days") or stock_row.get("leadtime_days") or stock_row.get("lea
        # numeric safety
        try:
            available_n = float(available) if available is not None else None
        except:
            available_n = None
        try:
            lt_n = float(lead_time_days) if lead_time_days is not None else None
        except:
            lt_n = None

        if available_n is None:
            reason.append("stock_unknown")
            feas_score = 0.5  # moderate
            feasible = True  # conservative allow
        else:
            ratio = min(1.0, available_n / max(1.0, float(required_units)))
            # base feas by availability ratio
            feas_score = ratio
            # adjust for lead time vs days_to_due
            if days_to_due is not None and lt_n is not None:
                # if lead time > days to due -> penalty
                if lt_n > days_to_due:
                    # heavy penalty proportional to difference and sensitivity
                    penalty = min(1.0, (lt_n - days_to_due) / max(1.0, days_to_due) * leadtime_sensitivity)
                    feas_score = feas_score * (1 - penalty)
                    reason.append(f"leadtime_penalty_{penalty:.2f}")
            feasible = (available_n >= required_units) and (lt_n is None or (days_to_due is None or lt_n <= days
    else:
        # No direct stock or required amount — try suppliers if allowed
        reason.append("no_direct_stock")
        if allow_subcontract and sku is not None:
            sup_df = suppliers_df[suppliers_df["sku_id"] == str(sku)]
            if not sup_df.empty:
                # pick supplier with earliest lead time that can satisfy amount
                sup_df_sorted = sup_df.sort_values("lead_time_days")
                chosen = None
                for _, s in sup_df_sorted.iterrows():
                    avail = s.get("available_m") if "available_m" in s else s.get("available_units")
                    lt = s.get("lead_time_days")
                    if required_units is None:
                        chosen = s
```

```
                        break
                    try:
                        avail_n = float(avail) if avail is not None else 0
                    except:
                        avail_n = 0
                    if avail_n >= required_units:
                        chosen = s
                        break
            if chosen is not None:
                # feasibility based on supplier lead time vs due
                lt = chosen.get("lead_time_days")
                if lt is not None and days_to_due is not None and lt > days_to_due:
                    feas_score = 0.2  # possible but risky (late)
                    reason.append("supplier_too_slow")
                    feasible = False
                else:
                    feas_score = 0.8  # fairly feasible via supplier
                    feasible = True
                    reason.append("supplier_ok")
            else:
                # partial supplier availability -> compute partial ratio
                total_avail = sup_df_sorted.get("available_m", pd.Series(dtype=float)).fillna(0).sum() if "a
                if required_units:
                    ratio = min(1.0, float(total_avail) / float(required_units)) if required_units>0 else 0.
                    feas_score = 0.3 * ratio
                    feasible = False
                    reason.append("supplier_partial")
                else:
                    feas_score = 0.5
                    feasible = False
                    reason.append("supplier_unknown_qty")
        else:
            feas_score = 0.1
            feasible = False
            reason.append("no_supplier")
    else:
        feas_score = 0.05
        feasible = False
        reason.append("no_subcontract_allowed")

    # clamp
    feas_score = max(0.0, min(1.0, float(feas_score)))
    return feasible, feas_score, reason

# Test semantic match & feasibility on one sample RFP (debug)
sample_meta = None
# pick first extracted meta (if extracted list exists)
try:
    sample_meta = extracted[0]
except:
    pass
if sample_meta:
    sku, ssim, topk = semantic_match(sample_meta, products, product_embeddings, model)
    feasible, feas_score, reasons = enhanced_feasibility(sample_meta, sku, stock_db_local, suppliers_local, samp
    print("Sample RFP:", sample_meta["rfp_id"], "=> matched sku:", sku, "sim:", ssim)
    print("Topk:", topk)
    print("Feas:", feasible, feas_score, reasons)
else:
    print("No sample meta found to test.")
```

```
Sample RFP: CAB001 => matched sku: CU_XLPE_3C_240_AR sim: 0.7032895088195801
Topk: [('CU_XLPE_3C_240_AR', 0.7032895088195801), ('CU_PVC_4C_4_UN', 0.5783826112747192), ('AL_XLPE_1C_400_AR', 0
Feas: False 0.8333333333333334 ['leadtime_penalty_0.17']
```

Recompute scores and save enhanced candidate list

```
# REPLACEMENT cell: build enhanced candidates list with robust numeric parsing

import re
import math
import pandas as pd

def extract_first_number(x):
    """Return first integer found in x (string/number). Return None if none found."""
    if x is None:
        return None
    if isinstance(x, (int, float)):
        if math.isnan(x):
            return None
```

```
            return int(x)
        s = str(x)
        # find first group of digits (allow commas)
        m = re.search(r"([0-9][0-9,]*)", s.replace(" ", ""))
        if not m:
            return None
        num_str = m.group(1).replace(",", "")
        try:
            return int(num_str)
        except:
            try:
                return int(float(num_str))
            except:
                return None

    # Updated enhanced_feasibility that uses extract_first_number for quantities/lengths
    def enhanced_feasibility(meta, sku, stock_df, suppliers_df, days_to_due, allow_subcontract=True, leadtime_sensit
        # default unknown
        feas_score = 0.0
        feasible = False
        reason = []
        required_units = None
        # Determine required amount (robustly)
        try:
            if meta.get("category") and "cable" in str(meta.get("category")).lower():
                required_units = extract_first_number(meta.get("length_m") or meta.get("Length_m") or meta.get("leng
                stock_col = "current_stock_m"
            else:
                required_units = extract_first_number(meta.get("quantity") or meta.get("Quantity"))
                stock_col = "current_stock_units"
        except Exception:
            required_units = None
            stock_col = "current_stock_units"

        # find stock row
        stock_row = None
        if sku is not None and "sku_id" in stock_df.columns:
            stock_row_df = stock_df[stock_df["sku_id"] == str(sku)]
            if not stock_row_df.empty:
                stock_row = stock_row_df.iloc[0].to_dict()

        # if we have stock info and required_units
        if stock_row and required_units is not None:
            available = stock_row.get(stock_col) if stock_col in stock_row else None
            lead_time_days = stock_row.get("lead_time_days") or stock_row.get("leadtime_days") or stock_row.get("lea
            # numeric safety
            try:
                available_n = float(available) if available is not None else None
            except:
                available_n = None
            try:
                lt_n = float(lead_time_days) if lead_time_days is not None else None
            except:
                lt_n = None

            if available_n is None:
                reason.append("stock_unknown")
                feas_score = 0.5  # moderate
                feasible = True  # conservative allow
            else:
                ratio = min(1.0, available_n / max(1.0, float(required_units)))
                feas_score = ratio
                if days_to_due is not None and lt_n is not None:
                    if lt_n > days_to_due:
                        penalty = min(1.0, (lt_n - days_to_due) / max(1.0, days_to_due) * leadtime_sensitivity)
                        feas_score = feas_score * (1 - penalty)
                        reason.append(f"leadtime_penalty_{penalty:.2f}")
                feasible = (available_n >= required_units) and (lt_n is None or (days_to_due is None or lt_n <= days
        else:
            # No direct stock or required amount — try suppliers if allowed
            reason.append("no_direct_stock")
            if allow_subcontract and sku is not None and "sku_id" in suppliers_df.columns:
                sup_df = suppliers_df[suppliers_df["sku_id"] == str(sku)]
                if not sup_df.empty:
                    sup_df_sorted = sup_df.sort_values("lead_time_days", na_position="last")
                    chosen = None
                    for _, s in sup_df_sorted.iterrows():
                        # supplier may have 'available_m' or 'available_units'
                        avail = s.get("available_m") if "available_m" in s else s.get("available_units") if "availab
                        lt = s.get("lead_time_days") or s.get("leadtime_days") or s.get("lead_time")
                        avail_n = None
                        try:
```

```
                        avail_n = float(avail) if avail is not None else 0.0
                    except:
                        avail_n = 0.0
                    # if required_units unknown, pick first supplier
                    if required_units is None:
                        chosen = s
                        break
                    if avail_n >= required_units:
                        chosen = s
                        break
                if chosen is not None:
                    lt = chosen.get("lead_time_days") or chosen.get("leadtime_days") or chosen.get("lead_time")
                    try:
                        lt_n = float(lt) if lt is not None else None
                    except:
                        lt_n = None
                    if lt_n is not None and days_to_due is not None and lt_n > days_to_due:
                        feas_score = 0.2
                        reason.append("supplier_too_slow")
                        feasible = False
                    else:
                        feas_score = 0.8
                        feasible = True
                        reason.append("supplier_ok")
                else:
                    # partial supplier availability -> compute partial ratio
                    total_avail = 0.0
                    if "available_m" in sup_df.columns:
                        total_avail = sup_df["available_m"].fillna(0).astype(float).sum()
                    elif "available_units" in sup_df.columns:
                        total_avail = sup_df["available_units"].fillna(0).astype(float).sum()
                    if required_units and total_avail:
                        ratio = min(1.0, float(total_avail) / float(required_units)) if required_units>0 else 0.
                        feas_score = 0.3 * ratio
                        feasible = False
                        reason.append("supplier_partial")
                    else:
                        feas_score = 0.5
                        feasible = False
                        reason.append("supplier_unknown_qty")
            else:
                feas_score = 0.1
                feasible = False
                reason.append("no_supplier")
        else:
            feas_score = 0.05
            feasible = False
            reason.append("no_subcontract_allowed")

    feas_score = max(0.0, min(1.0, float(feas_score)))
    return feasible, feas_score, reason

# Now build the candidate rows (uses your semantic_match, W_* and other vars)
rows = []
for meta in extracted:
    try:
        sku, sim_score, topk = semantic_match(meta, products, product_embeddings, model)
    except Exception as e:
        sku, sim_score, topk = None, 0.0, []
    feasible, feas_score, reasons = enhanced_feasibility(meta, sku, stock_db_local, suppliers_local, meta.get("d

    # safe normalization helpers (same as before)
    def normalize_log_value(v, min_v=1000, max_v=1_000_000):
        try:
            v = float(str(v).replace(",",""))
        except:
            return 0.0
        v = max(min_v, min(max_v, v))
        return (math.log(v) - math.log(min_v)) / (math.log(max_v) - math.log(min_v))
    def urgency_from_days(days, cap_days=90):
        if days is None:
            return 0.0
        d = max(-30, min(cap_days, days))
        return max(0.0, 1.0 - (d / cap_days))

    val_s = normalize_log_value(meta.get("estimated_value") or 0)
    urg_s = urgency_from_days(meta.get("days_to_due"))
    fit_s = float(sim_score) if sim_score is not None else 0.0
    feas_s = float(feas_score)

    final_score = W_VALUE*val_s + W_URGENCY*urg_s + W_FIT*fit_s + W_FEAS*feas_s
```

```
    if final_score >= 0.55 and feasible:
        rec = "Proceed"
    elif final_score >= 0.40:
        rec = "Consider"
    else:
        rec = "Skip"

    rows.append({
        "rfp_id": meta.get("rfp_id"),
        "client": meta.get("client"),
        "category": meta.get("category"),
        "est_value": meta.get("estimated_value"),
        "days_to_due": meta.get("days_to_due"),
        "matched_sku": sku,
        "semantic_fit": round(fit_s,4),
        "feasible": bool(feasible),
        "feas_score": round(feas_s,4),
        "value_score": round(val_s,4),
        "urgency_score": round(urg_s,4),
        "final_score": round(final_score,4),
        "recommendation": rec,
        "feas_reasons": ";".join(reasons) if isinstance(reasons, list) else str(reasons),
        "topk_matches": str(topk)
    })

candidates_enh = pd.DataFrame(rows).sort_values("final_score", ascending=False)
candidates_enh.to_csv(DATA_DIR / "candidate_rfps_enhanced.csv", index=False)
print("Saved /content/data/candidate_rfps_enhanced.csv")
candidates_enh
```

```
Saved /content/data/candidate_rfps_enhanced.csv
```

	rfp_id	client	category	est_value	days_to_due	matched_sku	semantic_fit	feasible	feas_sco
2	CAB003	LargeIndustrialClient	Wires & Cables	1600000	28	CU_PVC_4C_4_UN	0.7888	True	1.00
0	CAB001	LargeIndustrialClient	Wires & Cables	1500000	18	CU_XLPE_3C_240_AR	0.7033	False	0.83
3	FMEG001	LargeIndustrialClient	FMEG	120000	16	LED_FLD_200W	0.8241	True	1.00
4	FMEG002	LargeIndustrialClient	FMEG	375000	23	DB_3PH_36WAY	0.6348	False	0.69
1	CAB002	LargeIndustrialClient	Wires & Cables	720000	23	AL_XLPE_1C_400_AR	0.6857	False	0.02

Next steps: Generate code with candidates_enh | New interactive sheet

TECHNICAL AGENT

```
import pandas as pd
import numpy as np
from pathlib import Path

DATA_DIR = Path("/content/data")

candidates = pd.read_csv(DATA_DIR / "candidate_rfps_enhanced.csv")
products = pd.read_csv(DATA_DIR / "products.csv")
price_db = pd.read_csv(DATA_DIR / "price_db.csv")
stock_db = pd.read_csv(DATA_DIR / "stock_db.csv")
test_costs = pd.read_csv(DATA_DIR / "test_costs.csv")
suppliers = pd.read_csv(DATA_DIR / "suppliers_stock.csv") if (DATA_DIR/"suppliers_stock.csv").exists() else pd.D

# normalize sku_id columns to strings
for df in [products, price_db, stock_db, suppliers]:
    if "sku_id" in df.columns:
        df["sku_id"] = df["sku_id"].astype(str)

print("Loaded candidates:", len(candidates), "products:", len(products))
```

```
Loaded candidates: 5 products: 5
```

```
import pandas as pd
import numpy as np
import re
import ast
from typing import Dict, Any, List, Tuple, Optional

# ------------------------------
# Helpers
# ------------------------------
def detect_price_col(price_df: pd.DataFrame) -> Optional[str]:
    if price_df is None or len(price_df) == 0:
        return None
    common = ["unit_price","price","Price","rate","Rate","price_per_unit","price_per_m","unit_rate","mrp","selli
    for c in common:
        if c in price_df.columns:
            return c
    for c in price_df.columns:
        cl = c.lower()
        if ("price" in cl) or ("rate" in cl):
            return c
    return None

def safe_parse_topk(x) -> List[Tuple[str, float]]:
    """
    topk_matches looks like:
      "[('CU_PVC_4C_4_UN', 0.7887), ('CU_...', 0.70), ...]"
    """
    if x is None or (isinstance(x, float) and np.isnan(x)):
        return []
    if isinstance(x, list):
        return [(str(a), float(b)) for a, b in x]
    s = str(x).strip()
    if not s:
        return []
    try:
        obj = ast.literal_eval(s)
        if isinstance(obj, list):
            out = []
            for t in obj:
                if isinstance(t, (tuple, list)) and len(t) >= 2:
                    out.append((str(t[0]), float(t[1])))
            return out
    except:
        return []
    return []


# ------------------------------
# TechnicalAgent (topk-driven)
# ------------------------------
class TechnicalAgent:
    def __init__(self, products_df: pd.DataFrame, price_df: pd.DataFrame, stock_df: pd.DataFrame):
        self.products = products_df.copy()
        self.price_db = price_df.copy()
        self.stock_db = stock_df.copy()

        # normalize sku_id
        for df in [self.products, self.price_db, self.stock_db]:
            if "sku_id" in df.columns:
                df["sku_id"] = df["sku_id"].astype(str)

        inv = self.products.copy()

        # price merge
        price_col = detect_price_col(self.price_db)
        if price_col:
            inv = inv.merge(
                self.price_db[["sku_id", price_col]].rename(columns={price_col: "unit_price"}),
                on="sku_id", how="left"
            )
        else:
            inv["unit_price"] = np.nan

        # stock merge
        stock_cols = [c for c in ["current_stock_qty", "unit", "lead_time_days", "avg_cost"] if c in self.stock_
        if stock_cols:
            inv = inv.merge(self.stock_db[["sku_id"] + stock_cols], on="sku_id", how="left")
        else:
            inv["current_stock_qty"] = np.nan
            inv["lead_time_days"] = np.nan
            inv["unit"] = None
            inv["avg_cost"] = np.nan
```

```
        self.inventory = inv


    def recommend_top3_from_candidate(self, cand_row: pd.Series) -> pd.DataFrame:
        """
        Uses Sales Agent output:
          - matched_sku
          - semantic_fit
          - topk_matches (list of (sku, score))
        Returns Top-3 SKUs with SpecMatch% (proxy)
        """
        topk = safe_parse_topk(cand_row.get("topk_matches"))
        if not topk:
            # fallback: use matched_sku only
            ms = cand_row.get("matched_sku")
            topk = [(str(ms), float(cand_row.get("semantic_fit", 0.0)))] if ms is not None else []

        # Take top 3
        topk = topk[:3]

        rows = []
        for sku, score in topk:
            rows.append({"sku_id": str(sku), "Spec_Match_Score": float(score) * 100.0})
        top_df = pd.DataFrame(rows)

        # Join with inventory to attach specs + price + stock
        out = top_df.merge(self.inventory, on="sku_id", how="left")
        return out


    def generate_comparison_table(self, cand_row: pd.Series, top_matches: pd.DataFrame) -> pd.DataFrame:
        """
        Comparison table for demo: shows RFP id + category + est_value + days_to_due
        plus specs/prices for Top1-3.
        """
        # RFP-side fields that exist in your CSV
        rfp_fields = {
            "rfp_id": cand_row.get("rfp_id", ""),
            "category": cand_row.get("category", ""),
            "est_value": cand_row.get("est_value", ""),
            "days_to_due": cand_row.get("days_to_due", ""),
            "matched_sku": cand_row.get("matched_sku", ""),
            "semantic_fit": cand_row.get("semantic_fit", ""),
            "feasible": cand_row.get("feasible", ""),
            "final_score": cand_row.get("final_score", ""),
            "recommendation": cand_row.get("recommendation", ""),
        }

        params = [
            "sku_id","Spec_Match_Score",
            "Voltage_Rating","No_of_Cores","Cross_Section_Area","Insulation","Armouring","Conductor_Material",
            "unit_price","current_stock_qty","lead_time_days","avg_cost"
        ]

        top = top_matches.reset_index(drop=True).copy()
        while top.shape[0] < 3:
            top = pd.concat([top, pd.DataFrame([{}])], ignore_index=True)

        comp = {"Param": [], "RFP": [], "Top1": [], "Top2": [], "Top3": []}

        # Put RFP fields first (as rows)
        for k, v in rfp_fields.items():
            comp["Param"].append(k)
            comp["RFP"].append(v)
            comp["Top1"].append("")
            comp["Top2"].append("")
            comp["Top3"].append("")

        # Then product comparison rows
        for p in params:
            comp["Param"].append(p)
            comp["RFP"].append("")
            for i in range(3):
                val = ""
                if p in top.columns and pd.notna(top.loc[i].get(p, np.nan)):
                    val = top.loc[i, p]
                comp[f"Top{i+1}"].append(val)

        return pd.DataFrame(comp).set_index("Param")
```

```
    def build_selected_product_summary(self, cand_row: pd.Series, selected_row: pd.Series) -> Dict[str, Any]:
        sku = selected_row.get("sku_id")
        score = float(selected_row.get("Spec_Match_Score", 0.0))

        # concise, slide-friendly summary
        summary_text = (
            f"Selected SKU: {sku} | SpecMatch: {score:.1f}% | "
            f"Price: {selected_row.get('unit_price', 'NA')} | "
            f"Stock: {selected_row.get('current_stock_qty', 'NA')} | "
            f"Lead time (days): {selected_row.get('lead_time_days', 'NA')} | "
            f"RFP: {cand_row.get('rfp_id')} ({cand_row.get('category')}), "
            f"Due in {cand_row.get('days_to_due')} days, Recommendation: {cand_row.get('recommendation')}"
        )

        return {
            "rfp_id": cand_row.get("rfp_id"),
            "category": cand_row.get("category"),
            "days_to_due": cand_row.get("days_to_due"),
            "est_value": cand_row.get("est_value"),
            "selected_sku_id": sku,
            "spec_match_percent": score,
            "selected_sku_specs": {
                "Voltage_Rating": selected_row.get("Voltage_Rating"),
                "No_of_Cores": selected_row.get("No_of_Cores"),
                "Cross_Section_Area": selected_row.get("Cross_Section_Area"),
                "Insulation": selected_row.get("Insulation"),
                "Armouring": selected_row.get("Armouring"),
                "Conductor_Material": selected_row.get("Conductor_Material"),
            },
            "commercials": {
                "unit_price": selected_row.get("unit_price"),
                "current_stock_qty": selected_row.get("current_stock_qty"),
                "lead_time_days": selected_row.get("lead_time_days"),
                "avg_cost": selected_row.get("avg_cost"),
            },
            "summary_text": summary_text
        }


# ------------------------------
# RUN (end-to-end for 1 RFP)
# ------------------------------
tech = TechnicalAgent(products_df=products, price_df=price_db, stock_df=stock_db)

# pick best RFP according to sales-agent final_score
cand = candidates.sort_values("final_score", ascending=False).iloc[0]

top_matches = tech.recommend_top3_from_candidate(cand)
comparison_table = tech.generate_comparison_table(cand, top_matches)

print("Selected RFP:", cand.get("rfp_id"), "| category:", cand.get("category"),
      "| days_to_due:", cand.get("days_to_due"), "| recommendation:", cand.get("recommendation"))

print("\n--- TOP 3 SKU RECOMMENDATIONS ---")
cols = [c for c in ["sku_id","Spec_Match_Score","unit_price","current_stock_qty","lead_time_days"] if c in top_m
print(top_matches[cols].to_string(index=False))

print("\n--- COMPARISON TABLE ---")
print(comparison_table.to_string())

selected_summary = tech.build_selected_product_summary(cand, top_matches.iloc[0])
print("\n--- FINAL PRODUCT SUMMARY (to send forward) ---")
print(selected_summary["summary_text"])
```

```
Selected RFP: CAB003 | category: Wires & Cables  | days_to_due: 28 | recommendation: Proceed

--- TOP 3 SKU RECOMMENDATIONS ---
           sku_id  Spec_Match_Score  unit_price  lead_time_days
   CU_PVC_4C_4_UN         78.877521          40               5
CU_XLPE_3C_240_AR         53.861612         650              21
AL_XLPE_1C_400_AR         45.554173        1200              45

--- COMPARISON TABLE ---
                               RFP            Top1              Top2              Top3
Param
rfp_id                      CAB003
category           Wires & Cables
est_value                 1600000
days_to_due                    28
matched_sku          CU_PVC_4C_4_UN
semantic_fit                0.7888
feasible                      True
final_score                 0.8744
```

```
recommendation              Proceed
sku_id                                CU_PVC_4C_4_UN  CU_XLPE_3C_240_AR  AL_XLPE_1C_400_AR
Spec_Match_Score                           78.877521          53.861612          45.554173
Voltage_Rating
No_of_Cores
Cross_Section_Area
Insulation
Armouring
Conductor_Material
unit_price                                        40                650               1200
current_stock_qty
lead_time_days                                     5                 21                 45
avg_cost

--- FINAL PRODUCT SUMMARY (to send forward) ---
Selected SKU: CU_PVC_4C_4_UN | SpecMatch: 78.9% | Price: 40 | Stock: NA | Lead time (days): 5 | RFP: CAB003 (Wire
```

PRICING AGENT

```
import pandas as pd
import numpy as np
import math
import ast
from pathlib import Path
from datetime import datetime

DATA_DIR = Path("/content/data")
OUT_DIR = Path("/content/output")
OUT_DIR.mkdir(parents=True, exist_ok=True)

# Load
candidates_enh = pd.read_csv(DATA_DIR / "candidate_rfps_enhanced.csv")
products = pd.read_csv(DATA_DIR / "products.csv")
price_db = pd.read_csv(DATA_DIR / "price_db.csv")
stock_db = pd.read_csv(DATA_DIR / "stock_db.csv")
test_costs = pd.read_csv(DATA_DIR / "test_costs.csv")

# ---------- helpers ----------
def detect_price_col(df):
    for c in ["price_per_unit","price_per_m","unit_price","price","base_price"]:
        if c in df.columns: return c
    for c in df.columns:
        if "price" in c.lower() or "rate" in c.lower():
            return c
    return None

PRICE_COL = detect_price_col(price_db)
if PRICE_COL is None:
    raise RuntimeError("price_db has no price column detected.")

price_db["sku_id"] = price_db["sku_id"].astype(str)
products["sku_id"] = products["sku_id"].astype(str)
stock_db["sku_id"] = stock_db["sku_id"].astype(str)

# ---- SAFE maps ----
price_map = dict(zip(
    price_db["sku_id"].astype(str),
    pd.to_numeric(price_db[PRICE_COL], errors="coerce")
))

if "avg_cost" in stock_db.columns:
    cost_map = dict(zip(
        stock_db["sku_id"].astype(str),
        pd.to_numeric(stock_db["avg_cost"], errors="coerce")
    ))
else:
    cost_map = {}  # fallback when avg_cost missing
    print("⚠️ stock_db has no 'avg_cost' column. Falling back to 0.75*price heuristic.")



def safe_parse_topk(x):
    if pd.isna(x): return []
    if isinstance(x, list): return x
    try:
        obj = ast.literal_eval(str(x))
        return obj if isinstance(obj, list) else []
    except:
        return []
```

```
def infer_unit_from_category(cat: str):
    cat = str(cat).lower()
    if "cable" in cat or "wire" in cat:
        return "m"
    return "nos"

def make_fallback_bom_row(cand_row):
    """
    Builds a fake-but-plausible BOM line from Sales Agent output:
    - sku_id = matched_sku or topk[0]
    - required_qty derived from est_value / price
    - unit_cost_used from avg_cost (fallback to 0.75*price if missing)
    """
    sku = cand_row.get("matched_sku")
    if pd.isna(sku) or sku is None:
        topk = safe_parse_topk(cand_row.get("topk_matches"))
        if topk:
            sku = topk[0][0]
    sku = str(sku)

    est_value = float(cand_row.get("est_value", 0) or 0)

    unit_price = price_map.get(sku, np.nan)
    if unit_price is None or (isinstance(unit_price,float) and np.isnan(unit_price)) or unit_price <= 0:
        unit_price = 1.0  # avoid divide-by-zero for demo

    # estimate qty: assume est_value ≈ (unit_price * qty)
    qty = max(1.0, est_value / unit_price) if est_value > 0 else 100.0

    # cost used
    avg_cost = cost_map.get(sku, np.nan)
    if avg_cost is None or (isinstance(avg_cost,float) and np.isnan(avg_cost)):
        avg_cost = 0.75 * unit_price  # heuristic

    unit = infer_unit_from_category(cand_row.get("category",""))

    return pd.DataFrame([{
        "sku_id": sku,
        "required_qty": round(qty, 2),
        "unit": unit,
        "unit_cost_used": round(float(avg_cost), 2),
        "inhouse_price": unit_price
    }])

# ---------- Pricing logic (simple + stable) ----------
GST_RATE = 0.18
OVERHEAD_FACTOR = 1.09

def compute_price(unit_cost, margin=0.20):
    cost_with_overhead = unit_cost * OVERHEAD_FACTOR
    return cost_with_overhead / (1 - margin)

def margin_for_category(cat):
    cat = str(cat).lower()
    if "cable" in cat or "wire" in cat:
        return 0.18
    if "fmeg" in cat:
        return 0.25
    return 0.20

# ---------- Run pricing across candidates (no BOM required) ----------
summary_rows = []
per_rfp_lines = {}

for _, cand in candidates_enh.iterrows():
    rfp_id = cand["rfp_id"]
    bom_path = DATA_DIR / f"bom_{rfp_id}.csv"

    if bom_path.exists():
        bom = pd.read_csv(bom_path)
    else:
        bom = make_fallback_bom_row(cand)

    bom["required_qty"] = pd.to_numeric(bom["required_qty"], errors="coerce").fillna(0)

    lines = []
    for _, ln in bom.iterrows():
        sku = str(ln["sku_id"])
        qty = float(ln["required_qty"])
        unit_cost = float(ln.get("unit_cost_used", 0) or 0)
        unit = ln.get("unit", "nos")

        m = margin_for_category(cand.get("category",""))
```

```
            suggested = compute_price(unit_cost, m)

            # For demo: if we have an inhouse price, keep the suggested close to it (avoid absurd outputs)
            ih = ln.get("inhouse_price", np.nan)
            if pd.notna(ih):
                suggested = 0.5*suggested + 0.5*float(ih)

            line_total = suggested * qty
            gst = line_total * GST_RATE
            inc = line_total + gst

            lines.append({
                "rfp_id": rfp_id,
                "category": cand.get("category"),
                "sku_id": sku,
                "unit": unit,
                "required_qty": round(qty, 2),
                "unit_cost_used": round(unit_cost, 2),
                "suggested_unit_price": round(suggested, 2),
                "line_total_ex_gst": round(line_total, 2),
                "gst_amount": round(gst, 2),
                "line_total_inc_gst": round(inc, 2),
            })

        df_lines = pd.DataFrame(lines)
        per_rfp_lines[rfp_id] = df_lines

        total_ex = df_lines["line_total_ex_gst"].sum()
        total_gst = df_lines["gst_amount"].sum()
        total_inc = df_lines["line_total_inc_gst"].sum()

        summary_rows.append({
            "rfp_id": rfp_id,
            "category": cand.get("category"),
            "recommended_total_ex_gst": round(total_ex, 2),
            "recommended_gst": round(total_gst, 2),
            "recommended_total_inc_gst": round(total_inc, 2),
            "matched_sku": cand.get("matched_sku"),
            "semantic_fit": cand.get("semantic_fit"),
            "final_score": cand.get("final_score"),
            "recommendation": cand.get("recommendation"),
        })

    # ---------- Write workbook ----------
    excel_path = OUT_DIR / f"commercial_proposals_FALLBACKBOM_{datetime.now().strftime('%Y%m%d_%H%M%S')}.xlsx"
    with pd.ExcelWriter(excel_path, engine="openpyxl") as writer:
        for rfp_id, df in per_rfp_lines.items():
            df.to_excel(writer, sheet_name=str(rfp_id)[:31], index=False)
```