Performance Agent — Full Project Code (Hackathon)

This document contains the *complete working code* for the performance-agent hackathon project. It includes every file described in the agreed project structure, implemented to use the CSV headers you provided.

> Files assume CSVs live in `mockdata/` and outputs are written to `output/`.

Project tree

```
performance-agent-hackathon/
├── README.md
├── requirements.txt
├── .env.example
├── mockdata/
│   └── (place your CSVs here)
├── output/
├── artifacts/
├── scripts/
│   └── run_all.sh
├── src/
│   ├── main.py
│   ├── common/
│   │   ├── config.py
│   │   ├── logger.py
│   │   └── data_loader.py
│   ├── etl/
│   │   ├── cleaner.py
│   │   ├── merger.py
│   │   └── etl_pipeline.py
│   ├── features/
│   │   ├── feature_engineering.py
│   │   └── scoring_rules.py
│   ├── llm/
│   │   ├── openai_client.py
│   │   ├── prompts.py
│   │   ├── generator.py
│   │   └── postprocess.py
│   ├── validators/
│   │   ├── red_flag_checker.py
│   │   └── alignment_checker.py
│   └── webapp/
│       └── streamlit_app.py
└── tests/   (omitted for now)
```

All file contents follow. Save each code block to the appropriate path.

`requirements.txt`

```
pandas
numpy
openai
streamlit
python-dotenv
pyyaml
```

`.env.example`

```
OPENAI_API_KEY=your_api_key_here
DATA_DIR=mockdata
OUTPUT_DIR=output
OPENAI_MODEL=gpt-4o-mini
OPENAI_TEMPERATURE=0.2
```

`README.md`

~~~
# Performance Agent Hackathon

This repository is a one-week hackathon project that builds an AI-assisted
Performance Management Assistant using mock CSV datasets and the OpenAI API.

## Quick start
1. Create a virtualenv and install dependencies:
   ```bash
   python -m venv venv
   source venv/bin/activate
   pip install -r requirements.txt
   ```
2. Copy your CSV mock files into `mockdata/`.
3. Set `OPENAI_API_KEY` in your environment or copy `.env.example` to `.env`
and populate it.
4. Run the pipeline:
   ```bash
~~~

~~~
   python src/main.py
   ```
5. Run Streamlit UI:
   ```bash
   streamlit run src/webapp/streamlit_app.py
   ```

Outputs will be written to `output/` including `joined_evals.csv`,
`features.csv`, and `gpt_suggestions_postprocessed.json`.

Notes:
- Update `src/common/config.py` if your CSV filenames or locations differ.
- Tests are intentionally omitted until after the demo.
~~~

`scripts/run_all.sh`

```bash
#!/usr/bin/env bash
set -e
python src/main.py
streamlit run src/webapp/streamlit_app.py
```

Make executable: `chmod +x scripts/run_all.sh`.

`src/common/config.py`

```python
from pathlib import Path
import os

BASE_DIR = Path(__file__).resolve().parents[2]
DATA_DIR = Path(os.getenv("DATA_DIR", BASE_DIR / "mockdata"))
OUTPUT_DIR = Path(os.getenv("OUTPUT_DIR", BASE_DIR / "output"))
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

# Expected file names
FILES = {
    "employees": "employees.csv",
    "workday_checkins": "workday_checkins.csv",
    "jira_metrics": "jira_metrics.csv",
    "lms_completions": "lms_completions.csv",
    "recognition": "recognition.csv",
    "feedback_360": "feedback_360.csv",
    "github_metrics": "github_metrics.csv",
    "defects": "defects.csv",
    "rto": "rto.csv",
    "release_metrics": "release_metrics.csv",
```

```
    "manager_evaluations": "manager_evaluations.csv",
    "idp_goals": "idp_goals.csv",
    "red_flag_dictionary": "red_flag_dictionary.csv",
}

# Column mapping: canonical -> list of possible headers in CSV
COLUMN_MAP = {
    "employees": {
        "employee_id": ["employee_id","emp_id","id"],
        "name": ["name","employee_name"],
        "email": ["email"],
        "hire_date": ["hire_date","start_date"],
        "role": ["role","job_title"],
        "level": ["level"],
        "org": ["org","organization"],
        "location": ["location"],
        "manager_id": ["manager_id","manager"]
    },
    "workday_checkins": {
        "employee_id": ["employee_id","emp_id"],
        "date": ["date"],
        "self_reflection": ["self_reflection","reflection","note"],
        "manager_note": ["manager_note","manager_comments"],
        "sentiment_score": ["sentiment_score"]
    },
    "jira_metrics": {
        "employee_id": ["employee_id","assignee_id"],
        "sprint": ["sprint"],
        "story_points_committed":
["story_points_committed","story_points_committed"],
        "story_points_completed":
["story_points_completed","story_points_done"],
        "spillover_points": ["spillover_points","spillover"],
        "bugs_introduced": ["bugs_introduced"],
        "bugs_fixed": ["bugs_fixed"]
    },
    "lms_completions": {
        "employee_id": ["employee_id"],
        "course_id": ["course_id"],
        "course_name": ["course_name"],
        "completion_date": ["completion_date"],
        "status": ["status"],
        "score": ["score"]
    },
    "recognition": {
        "employee_id": ["employee_id"],
        "giver_id": ["giver_id"],
        "date": ["date"],
        "recognition_type": ["recognition_type"],
        "message": ["message"]
    },
    "feedback_360": {
        "feedback_id": ["feedback_id"],
        "giver_id": ["giver_id"],
        "receiver_id": ["receiver_id"],
        "date": ["date"],
        "text": ["text","comment"],
        "rating": ["rating"],
        "tags": ["tags"]
    },
    "github_metrics": {
        "employee_id": ["employee_id"],
        "month": ["month"],
        "commits": ["commits"],
        "prs_merged": ["prs_merged","prs"],
        "reviews": ["reviews"],
        "copilot_acceptance_pct":
["copilot_acceptance_pct","copilot_acceptance"]
    },
    "defects": {
        "defect_id": ["defect_id"],
        "assignee_id": ["assignee_id"],
        "reporter_id": ["reporter_id"],
        "severity": ["severity"],
        "created_date": ["created_date"],
        "resolved_date": ["resolved_date"],
        "escape": ["escape"],
        "linked_release": ["linked_release"]
    },
    "rto": {
        "employee_id": ["employee_id"],
        "month": ["month"],
        "days_off": ["days_off"],
        "remote_days": ["remote_days"],
        "in_office_days": ["in_office_days"],
        "compliant": ["compliant"]
    },
    "release_metrics": {
        "release_id": ["release_id"],
        "date": ["date"],
        "release_engineer_id": ["release_engineer_id"],
        "features_delivered": ["features_delivered"],
        "defect_ppm": ["defect_ppm"],
        "on_time_percent": ["on_time_percent"],
        "rollback_count": ["rollback_count"]
    },
    "manager_evaluations": {
        "eval_id": ["eval_id"],
        "employee_id": ["employee_id"],
        "period": ["period","review_period"],
        "rating": ["rating"],
        "comment": ["comment","manager_comment"],
        "bias_type": ["bias_type"],
        "bias_notes": ["bias_notes"],
        "author_id": ["author_id"],
        "timestamp": ["timestamp"]
    },
    "idp_goals": {
        "goal_id": ["goal_id"],
        "employee_id": ["employee_id"],
        "description": ["description","goal_description"],
        "start_date": ["start_date"],
        "due_date": ["due_date"],
        "status": ["status"],
        "owner": ["owner"]
    },
    "red_flag_dictionary": {
        "term": ["term"],
        "severity": ["severity"],
        "category": ["category"],
        "note": ["note"]
    }
}

# Map rating categories to numeric for simple rules
RATING_MAP = {
    "consistently exceeds": 5,
    "exceeds": 4,
    "meets": 3,
    "inconsistently meets": 2,
    "needs improvement": 1
}

# GPT defaults
OPENAI_MODEL = os.getenv("OPENAI_MODEL", "gpt-4o-mini")
OPENAI_TEMPERATURE = float(os.getenv("OPENAI_TEMPERATURE", "0.2"))
```

`src/common/logger.py`

```
import logging

def get_logger(name=__name__):
    logging.basicConfig(
        format="%(asctime)s %(levelname)s %(name)s: %(message)s",
        level=logging.INFO
    )
    return logging.getLogger(name)
```

---

src/common/data_loader.py

```python
import pandas as pd
from pathlib import Path
from .config import DATA_DIR, FILES, COLUMN_MAP
from .logger import get_logger

log = get_logger("data_loader")


def _read_csv_if_exists(fname):
    p = Path(DATA_DIR) / fname
    if not p.exists():
        log.warning(f"Missing file: {p}. Returning None.")
        return None
    try:
        df = pd.read_csv(p)
    except Exception as e:
        log.info(f"CSV read failed for {p}, trying Excel: {e}")
        try:
            df = pd.read_excel(p)
        except Exception as e2:
            log.error(f"Failed to read {p}: {e2}")
            return None
    df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')
    return df


def load_all():
    loaded = {}
    for key, fname in FILES.items():
        df = _read_csv_if_exists(fname)
        loaded[key] = df
    return loaded


def find_column(df, candidates):
    if df is None:
        return None
    for c in candidates:
        if c in df.columns:
            return c
    # fallback: substring match
    for col in df.columns:
        for c in candidates:
            if c in col:
                return col
    return None
```

`src/etl/cleaner.py`

```python
import pandas as pd
from ..common.config import COLUMN_MAP
from ..common.data_loader import find_column
from ..common.logger import get_logger

log = get_logger("cleaner")


def normalize_manager_evaluations(df):
    if df is None:
        return None
    df = df.copy()
    cmap = COLUMN_MAP['manager_evaluations']
    eid = find_column(df, cmap['employee_id'])
    if eid:
        df['employee_id'] = df[eid].astype(str).str.strip()
    else:
        log.warning('employee_id not found in manager_evaluations')
        df['employee_id'] = df.index.astype(str)
    # rating
    rcol = find_column(df, cmap['rating'])
    if rcol:
        df['rating_raw'] = df[rcol]
        df['rating'] = df[rcol].astype(str).str.strip().str.lower()
    # comment
    ccol = find_column(df, cmap['comment'])
    if ccol:
        df['comment'] = df[ccol].astype(str)
    # bias fields
    bcol = find_column(df, cmap.get('bias_type', []))
    if bcol:
        df['bias_type'] = df[bcol]
    ncol = find_column(df, cmap.get('bias_notes', []))
    if ncol:
        df['bias_notes'] = df[ncol]
    # period
    pcol = find_column(df, cmap.get('period', []))
    if pcol:
        df['period'] = pd.to_datetime(df[pcol], errors='coerce')
    return df


def normalize_employees(df):
    if df is None:
        return None
    df = df.copy()
    cmap = COLUMN_MAP['employees']
    eid = find_column(df, cmap['employee_id'])
    if eid:
        df['employee_id'] = df[eid].astype(str).str.strip()
    namec = find_column(df, cmap['name'])
    if namec:
        df['name'] = df[namec]
    rolec = find_column(df, cmap['role'])
    if rolec:
        df['role'] = df[rolec]
    return df
```

---

src/etl/merger.py

```python
import pandas as pd
from ..common.logger import get_logger

log = get_logger("merger")


def left_merge_on_employee(base_df, other_df, other_prefix):
    if other_df is None:
        return base_df
    odf = other_df.copy()
    # ensure employee id column present
    if 'employee_id' not in odf.columns:
        # try common fallbacks
        if 'assignee_id' in odf.columns:
            odf = odf.rename(columns={'assignee_id':'employee_id'})
        elif 'receiver_id' in odf.columns:
            odf = odf.rename(columns={'receiver_id':'employee_id'})
        elif 'assignee_id' in odf.columns:
            odf = odf.rename(columns={'assignee_id':'employee_id'})
    return base_df.merge(odf.add_prefix(other_prefix+'_'),
left_on='employee_id', right_on=other_prefix+'_employee_id', how='left')


def build_joined(loaded):
    mgr = loaded.get('manager_evaluations')
    emp = loaded.get('employees')
    jira = loaded.get('jira_metrics')
    release = loaded.get('release_metrics')
    defects = loaded.get('defects')
    feedback = loaded.get('feedback_360')
    idp = loaded.get('idp_goals')
    rto = loaded.get('rto')
    recognition = loaded.get('recognition')
    github = loaded.get('github_metrics')
    workday = loaded.get('workday_checkins')
    lms = loaded.get('lms_completions')
    base = mgr.copy() if mgr is not None else (emp.copy() if emp is not None
else pd.DataFrame())
    # ensure employee_id exists on base
    if 'employee_id' not in base.columns and emp is not None and
'employee_id' in emp.columns:
        base = base.merge(emp[['employee_id']], left_index=True,
right_index=False, how='left')
    # sequential merges
    base = left_merge_on_employee(base, emp, 'emp')
    base = left_merge_on_employee(base, jira, 'jira')
    base = left_merge_on_employee(base, release, 'release')
    base = left_merge_on_employee(base, defects, 'defects')
    base = left_merge_on_employee(base, feedback, 'fb')
    base = left_merge_on_employee(base, idp, 'idp')
    base = left_merge_on_employee(base, rto, 'rto')
    base = left_merge_on_employee(base, recognition, 'recog')
    base = left_merge_on_employee(base, github, 'gh')
    base = left_merge_on_employee(base, workday, 'wd')
    base = left_merge_on_employee(base, lms, 'lms')
    return base
```

---

src/etl/etl_pipeline.py

```
from ..common.data_loader import load_all
from .cleaner import normalize_manager_evaluations, normalize_employees
from .merger import build_joined
from ..common.config import OUTPUT_DIR
from ..common.logger import get_logger

log = get_logger("etl_pipeline")


def run_etl():
    loaded = load_all()
    # normalize
    loaded['manager_evaluations'] =
normalize_manager_evaluations(loaded.get('manager_evaluations'))
    loaded['employees'] = normalize_employees(loaded.get('employees'))
    joined = build_joined(loaded)
    out_path = OUTPUT_DIR / 'joined_evals.csv'
    joined.to_csv(out_path, index=False)
    log.info(f'Wrote joined_evals to {out_path}')
    return joined
```

---

src/features/feature_engineering.py

```
import pandas as pd
from ..common.logger import get_logger
from ..common.config import OUTPUT_DIR
import numpy as np

log = get_logger("feature_engineering")


def compute_kpis(joined):
    df = joined.copy()
    # detect story points
    sp_cols = [c for c in df.columns if 'story_points' in c]
    if sp_cols:
        df['story_points_completed'] = pd.to_numeric(df[sp_cols[0]],
errors='coerce').fillna(0)
    else:
        df['story_points_completed'] = 0
    # spillover
    spill_cols = [c for c in df.columns if 'spill' in c]
    if spill_cols:
        df['spillover_points'] = pd.to_numeric(df[spill_cols[0]],
errors='coerce').fillna(0)
    else:
        df['spillover_points'] = 0
    # defects
    defect_cols = [c for c in df.columns if 'defect' in c and ('severity' in
c or 'count' in c)]
    if defect_cols:
        df['defect_severity'] = pd.to_numeric(df[defect_cols[0]],
errors='coerce').fillna(0)
    else:
        # fallback: use defects_defects_severity if present
        df['defect_severity'] = pd.to_numeric(df.get('defects_severity', 0),
errors='coerce').fillna(0)
    # recognition rate
    df['recognitions'] = df.filter(like='recog_').shape[1]
    # feedback sentiment (simple heuristic): use sentiment_score if present
    if 'wd_sentiment_score' in df.columns:
        df['feedback_sentiment'] = df['wd_sentiment_score']
    else:
        df['feedback_sentiment'] = 0
    # copilot acceptance
    if 'gh_copilot_acceptance_pct' in df.columns:
        df['copilot_acceptance_pct'] =
pd.to_numeric(df['gh_copilot_acceptance_pct'], errors='coerce').fillna(0)
    else:
        df['copilot_acceptance_pct'] = 0
    # RTO compliance from rto_compliant
    if 'rto_compliant' in df.columns:
        df['rto_compliant'] = df['rto_compliant']
    else:
        df['rto_compliant'] = df.get('rto_compliant', False)

    # aggregate per employee
    aggs = df.groupby('employee_id').agg({
        'story_points_completed':'sum',
        'spillover_points':'sum',
        'defect_severity':'sum',
        'recognitions':'sum',
        'feedback_sentiment':'mean',
        'copilot_acceptance_pct':'mean',
        'rto_compliant':'max'
    }).reset_index()

    # KPIs derived
    aggs['velocity'] = aggs['story_points_completed']
    aggs['spillover_pct'] = np.where((aggs['story_points_completed']
+aggs['spillover_points'])>0, aggs['spillover_points']/
(aggs['story_points_completed']+aggs['spillover_points']), 0)
    aggs['defect_density_per_100_sp'] =
np.where(aggs['story_points_completed']>0, (aggs['defect_severity']/
aggs['story_points_completed'])*100, 0)
    aggs['recognition_rate'] = aggs['recognitions']
    aggs['sentiment'] = aggs['feedback_sentiment']
    aggs['copilot_acceptance_pct'] = aggs['copilot_acceptance_pct']
    aggs['rto_compliant'] = aggs['rto_compliant']

    out = OUTPUT_DIR / 'features.csv'
    aggs.to_csv(out, index=False)
    log.info(f'Wrote features to {out}')
    return aggs
```

---

src/features/scoring_rules.py

```
from ..common.logger import get_logger
log = get_logger("scoring_rules")

POSITIVE_TERMS =
["excellent","outstanding","strong","exceptional","great","well
done","consistent"]


def has_positive_language(text):
    if not text or not isinstance(text, str):
        return False
    t = text.lower()
    return any(p in t for p in POSITIVE_TERMS)
```

---

src/validators/red_flag_checker.py

```
import pandas as pd
from ..common.config import FILES
from ..common.logger import get_logger

log = get_logger("red_flag_checker")


def load_red_flags(data_dir):
    p = Path(data_dir) / FILES['red_flag_dictionary']
    try:
        df = pd.read_csv(p)
        df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')
        terms = df['term'].astype(str).str.lower().tolist()
        return terms
    except Exception as e:
        log.warning(f"Could not load red flag dictionary: {e}")
        return []


def find_red_flags_in_text(text, red_terms):
    if not text or not isinstance(text, str):
        return []
    t = text.lower()
    return [r for r in red_terms if r in t]
```

---

src/validators/alignment_checker.py

```
from ..common.config import RATING_MAP


def rating_to_numeric(rating_str):
    if not rating_str:
        return None
    rs = str(rating_str).strip().lower()
    return RATING_MAP.get(rs)


def detect_mismatch(comment, rating_str):
    # simplistic: if comment contains positive terms but rating low
    from ..features.scoring_rules import has_positive_language
    num = rating_to_numeric(rating_str)
    pos = has_positive_language(comment)
    if pos and num is not None and num <= 2:
        return True
    return False
```

`src/llm/openai_client.py`

```
import os
from openai import OpenAI
from ..common.config import OPENAI_MODEL, OPENAI_TEMPERATURE
from ..common.logger import get_logger

log = get_logger("openai_client")


def get_client():
    api_key = os.getenv("OPENAI_API_KEY")
    if not api_key:
        raise EnvironmentError("OPENAI_API_KEY not set in environment")
    client = OpenAI(api_key=api_key)
    return client


def chat_generate(client, messages, max_tokens=400, temperature=None,
model=None):
    model = model or OPENAI_MODEL
    temperature = OPENAI_TEMPERATURE if temperature is None else temperature
    resp = client.chat.completions.create(
        model=model,
        messages=messages,
        max_tokens=max_tokens,
        temperature=temperature
    )
    return resp.choices[0].message["content"]
```

`src/llm/prompts.py`

```
GEN_PROMPT = """
You are an assistant that drafts concise, professional performance evaluation
comments tied to data.
Employee: {employee_id}
Role: {role}
Metrics summary:
- velocity: {velocity}
- spillover_pct: {spillover_pct}
- defect_density_per_100_sp: {defect_density_per_100_sp}
- recognition_rate: {recognition_rate}
- sentiment: {sentiment}
- copilot_acceptance_pct: {copilot_acceptance_pct}

Produce JSON only, of the form:
{"candidates":[{"label":"A","comment":"...","idp":"...","rationale":
["...","..."]}, ...]}
Make 3 candidates:
A: Balanced
B: Strength/praise
C: Developmental
Tone: objective, non-biased, no invented facts.
"""
```

---

src/llm/generator.py

```
import json
from .openai_client import get_client, chat_generate
from .prompts import GEN_PROMPT
from ..common.logger import get_logger
from ..common.config import OUTPUT_DIR

log = get_logger("llm_generator")


def generate_for_employee(eid, role, metrics):
    prompt = GEN_PROMPT.format(
        employee_id=eid,
        role=role or 'N/A',
        velocity=metrics.get('velocity', 0),
        spillover_pct=metrics.get('spillover_pct', 0),
        defect_density_per_100_sp=metrics.get('defect_density_per_100_sp',
0),
        recognition_rate=metrics.get('recognition_rate', 0),
        sentiment=metrics.get('sentiment', 0),
        copilot_acceptance_pct=metrics.get('copilot_acceptance_pct', 0)
    )
    messages = [
        {"role":"system","content":"You are a professional assistant."},
        {"role":"user","content":prompt}
    ]
    client = get_client()
    try:
        raw = chat_generate(client, messages)
    except Exception as e:
        log.error("OpenAI call failed: %s", e)
        raw = '{"error":"openai_failed"}'
    try:
        parsed = json.loads(raw)
    except Exception:
        parsed = {"raw": raw}
    return parsed


def generate_batch(employees, features_df):
    suggestions = []
    for emp in employees:
        eid = emp['employee_id']
        role = emp.get('role', 'N/A')
        feats = features_df[features_df['employee_id']==eid]
        if not feats.empty:
            metrics = feats.iloc[0].to_dict()
        else:
            metrics = {}
        sug = generate_for_employee(eid, role, metrics)
        suggestions.append({"employee_id":eid, "suggestions":sug})
    out = OUTPUT_DIR / 'gpt_suggestions.json'
    with open(out, 'w') as f:
        json.dump(suggestions, f, indent=2)
    log.info(f'Wrote suggestions to {out}')
    return suggestions
```

---

src/llm/postprocess.py

```
import json
from ..common.config import OUTPUT_DIR, FILES
from ..common.logger import get_logger
from ..validators.red_flag_checker import find_red_flags_in_text
import pandas as pd
from pathlib import Path

log = get_logger('postprocess')


def load_red_terms(data_dir=Path(__file__).resolve().parents[3] /
'mockdata'):
    p = Path(data_dir) / FILES['red_flag_dictionary']
    try:
        df = pd.read_csv(p)
        df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')
        return df['term'].astype(str).str.lower().tolist()
    except Exception as e:
        log.warning(f'Could not load red flag dictionary: {e}')
        return []
```

```python
def postprocess_and_flag(suggestions):
    red_terms = load_red_terms()
    for s in suggestions:
        sug = s.get('suggestions')
        if isinstance(sug, dict) and 'candidates' in sug:
            for c in sug['candidates']:
                txt = c.get('comment','')
                flags = find_red_flags_in_text(txt, red_terms)
                c['red_flags'] = flags
    out = OUTPUT_DIR / 'gpt_suggestions_postprocessed.json'
    with open(out, 'w') as f:
        json.dump(suggestions, f, indent=2)
    log.info(f'Wrote postprocessed suggestions to {out}')
    return suggestions
```

---

src/webapp/streamlit_app.py

```python
import streamlit as st
import pandas as pd
import json
from pathlib import Path
from ..common.config import OUTPUT_DIR
from ..common.logger import get_logger

log = get_logger('streamlit')
st.set_page_config(page_title='AI Performance Assistant', layout='wide')

joined_p = OUTPUT_DIR / 'joined_evals.csv'
features_p = OUTPUT_DIR / 'features.csv'
sug_p = OUTPUT_DIR / 'gpt_suggestions_postprocessed.json'

joined = pd.read_csv(joined_p) if joined_p.exists() else pd.DataFrame()
features = pd.read_csv(features_p) if features_p.exists() else pd.DataFrame()
sugs = json.load(open(sug_p)) if sug_p.exists() else []

st.title('AI Performance Assistant — Demo')
if joined.empty:

st.warning('No joined_evals.csv found. Run the ETL pipeline first (python
src/main.py).')
else:
    cols = st.columns([2,3])
    with cols[0]:
        st.header('Team')
        table = joined[['employee_id','emp_name','emp_role']] if 'emp_name'
in joined.columns else joined[['employee_id']].drop_duplicates()
```

```
        # unify display
        if 'emp_name' in joined.columns:
            display =
table.rename(columns={'employee_id':'id','emp_name':'name','emp_role':'role'})
        else:
            display = table.rename(columns={'employee_id':'id'})
        sel = st.selectbox('Select employee', display['id'].tolist())
        st.dataframe(display)
    with cols[1]:
        st.header('Details')
        row = joined[joined['employee_id']==sel].iloc[0]
        st.subheader(f"{row.get('emp_name', sel)} — {row.get('emp_role','N/
A')}")
        st.write('Manager comment:', row.get('comment','(none)'))
        f = features[features['employee_id']==sel]
        if not f.empty:
            st.subheader('KPIs')
            st.table(f.T)
        emp_sug = next((x for x in sugs if x.get('employee_id')==sel), None)
        if emp_sug:
            st.subheader('AI suggestions')
            for c in emp_sug['suggestions'].get('candidates', []):
                st.markdown(f"**{c.get('label','?')}** —
{c.get('comment','')}")
                st.write('IDP:', c.get('idp',''))
                st.write('Rationale:', c.get('rationale',[]))
                if st.button(f"Accept {c.get('label')}", key=f"accept_{sel}
_{c.get('label')}"):
                    st.success('Accepted (demo)')
        else:
            st.info('No suggestions available. Run generator.')
```

src/main.py

```
from src.etl.etl_pipeline import run_etl
from src.features.feature_engineering import compute_kpis
from src.llm.generator import generate_batch
from src.llm.postprocess import postprocess_and_flag
from src.common.logger import get_logger

log = get_logger('main')


def main():
    log.info('Running ETL...')
    joined = run_etl()
    log.info('Computing features...')
    feats = compute_kpis(joined)
    emp_rows = joined[['employee_id']].drop_duplicates().to_dict('records')
    # attempt to include role if present
    if 'emp_role' in joined.columns:
        emp_rows =
joined[['employee_id','emp_role']].drop_duplicates().to_dict('records')
    else:
        emp_rows =
joined[['employee_id']].drop_duplicates().to_dict('records')
    log.info('Generating GPT suggestions...')
    suggestions = generate_batch(emp_rows, feats)
    log.info('Postprocessing suggestions...')
    postprocess_and_flag(suggestions)
    log.info('Done. Outputs are in output/')

if __name__ == '__main__':
    main()
```

How I used your header images

I used the CSV header text you provided (from the images) and implemented column mapping in `src/common/config.py`.

If any header was mis-read, edit `COLUMN_MAP` in `src/common/config.py` and re-run the pipeline.

Next steps

- Run `python src/main.py` after placing CSVs in `mockdata/` and setting `OPENAI_API_KEY`.
- Run `streamlit run src/webapp/streamlit_app.py` to view the UI.

If you want, I can now: - produce a ZIP of these files for download, or - tweak the OpenAI client to match a different SDK, or - update the Streamlit UI layout or add graphs.

End of generated project.