=== README.md ===

Performance Agent Hackathon - README

This repository is a one-week hackathon project that builds an AI-assisted Performance Management Assist

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_

=== 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

=== src/common/config.py ===
```
# src/common/config.py
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 ===
# 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 ===
# src/common/data_loader.py
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 ===
# src/etl/cleaner.py
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



=== src/etl/merger.py ===
# src/etl/merger.py
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+

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 ===
# 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 ===

# 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):
    """
    Compute KPIs required by the hackathon:
      - velocity: sum of story points completed
      - spillover_pct: spillover / (committed or committed+spillover) per employee
      - defect_density_per_100_sp: defects per 100 story points
      - recognition_rate: total recognitions (count) in period
      - sentiment_mix: ratio of positive/negative/neutral feedback (requires feedback_360.text or sentim
      - copilot_acceptance_pct: average copilot acceptance %
      - rto_compliance_pct: percent of months compliant (if rto data exists)
      - release_quality: defect_ppm and on_time_percent aggregated for releases the engineer is associat
    """
    df = joined.copy()

    # Story points completed (jira)
    sp_cols = [c for c in df.columns if 'story_points_completed' in c or 'story_points_done' in c or ('s
    # fallback heuristics
    if sp_cols:
        df['story_points_completed'] = pd.to_numeric(df[sp_cols[0]], errors='coerce').fillna(0)
    else:
        # try common jira prefixed column
        sp_alt = [c for c in df.columns if 'jira_story_points' in c or 'jira_story_points_completed' in
        df['story_points_completed'] = pd.to_numeric(df[sp_alt[0]], errors='coerce').fillna(0) if sp_alt

    # Committed / spillover
    committed_cols = [c for c in df.columns if 'committed' in c]
    df['story_points_committed'] = pd.to_numeric(df[committed_cols[0]], errors='coerce').fillna(0) if co
    spill_cols = [c for c in df.columns if 'spillover' in c or 'spill' in c]
    df['spillover_points'] = pd.to_numeric(df[spill_cols[0]], errors='coerce').fillna(0) if spill_cols e

    # Defects severity
```

```
defect_cols = [c for c in df.columns if 'defect' in c and ('severity' in c or 'count' in c or 'ppm'
if defect_cols:
    df['defect_count'] = pd.to_numeric(df[defect_cols[0]], errors='coerce').fillna(0)
else:
    # look for defects_defects_severity etc.
    df['defect_count'] = pd.to_numeric(df.get('defects_defect_severity', 0), errors='coerce').fillna

# Recognitions
recog_cols = [c for c in df.columns if c.startswith('recog_') or 'recognition' in c]
# recognition count heuristic: if prefixed columns exist, count non-null messages; else try a simple
if any('recog_message' in c for c in df.columns):
    df['recognitions'] = df.filter(like='recog_message').notna().sum(axis=1)
elif 'recognitions' in df.columns:
    df['recognitions'] = pd.to_numeric(df['recognitions'], errors='coerce').fillna(0)
else:
    df['recognitions'] = 0

# Feedback sentiment: use wd_sentiment_score if present, otherwise 360 feedback text sentiment not c
if 'wd_sentiment_score' in df.columns:
    df['feedback_sentiment'] = pd.to_numeric(df['wd_sentiment_score'], errors='coerce').fillna(0)
elif 'fb_rating' in df.columns:
    df['feedback_sentiment'] = pd.to_numeric(df['fb_rating'], errors='coerce').fillna(0)
else:
    df['feedback_sentiment'] = 0

# github copilot acceptance
if 'gh_copilot_acceptance_pct' in df.columns:
    df['copilot_acceptance_pct'] = pd.to_numeric(df['gh_copilot_acceptance_pct'], errors='coerce').f
else:
    df['copilot_acceptance_pct'] = 0

# RTO compliance - if rto_compliant or rto fields present
if 'rto_compliant' in df.columns:
    df['rto_compliant_flag'] = df['rto_compliant'].astype(bool)
else:
    df['rto_compliant_flag'] = False

# Release-level metrics (if release_metrics info prefixed)
if 'release_defect_ppm' in df.columns:
    df['release_defect_ppm'] = pd.to_numeric(df['release_defect_ppm'], errors='coerce').fillna(0)
else:
    df['release_defect_ppm'] = df.get('release_defect_ppm', 0)

if 'release_on_time_percent' in df.columns:
    df['release_on_time_percent'] = pd.to_numeric(df['release_on_time_percent'], errors='coerce').fi
else:
    df['release_on_time_percent'] = df.get('release_on_time_percent', 0)

# Aggregate per employee
aggs = df.groupby('employee_id').agg({
    'story_points_completed': 'sum',
    'story_points_committed': 'sum',
    'spillover_points': 'sum',
    'defect_count': 'sum',
    'recognitions': 'sum',
    'feedback_sentiment': 'mean',
    'copilot_acceptance_pct': 'mean',
    'rto_compliant_flag': 'mean',
    'release_defect_ppm': 'mean',
    'release_on_time_percent': 'mean'
}).reset_index()

# Derived KPIs
aggs['velocity'] = aggs['story_points_completed']
# spillover pct relative to committed (if no committed, fallback to 0)
aggs['spillover_pct'] = np.where(aggs['story_points_committed']+aggs['spillover_points']>0,
                                 aggs['spillover_points']/(aggs['story_points_committed']+aggs['spil
                                 0)
aggs['defect_density_per_100_sp'] = np.where(aggs['story_points_completed']>0,
                                             (aggs['defect_count']/aggs['story_points_completed'])*1
                                             0)
aggs['recognition_rate'] = aggs['recognitions']  # raw count; UI can normalize per quarter
```

```
    aggs['sentiment'] = aggs['feedback_sentiment']
    aggs['copilot_acceptance_pct'] = aggs['copilot_acceptance_pct']
    aggs['rto_compliance_pct'] = aggs['rto_compliant_flag'] * 100
    aggs['release_defect_ppm'] = aggs['release_defect_ppm']
    aggs['release_on_time_percent'] = aggs['release_on_time_percent']

    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 ===
# 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/llm/openai_client.py ===
# 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/postprocess.py ===
# 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 []

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/llm/prompts.py ===

# 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}
- rto_compliance_pct: {rto_compliance_pct}
- release_on_time_percent: {release_on_time_percent}
- release_defect_ppm: {release_defect_ppm}

Produce JSON only, of the form:
{"candidates":[{"label":"A","comment":"...","idp":"...","rationale":["...","..."]}, ...]}

Make 3 candidates:
A: Balanced (1-2 sentences)
B: Strength/praise (1 sentence)
C: Developmental (1-2 sentences, specific improvement)

Additional rules:
- Tone: objective, constructive, non-biased, and grounded in the metrics provided.
- Do not invent facts or percentages not provided above.
- If the manager comment contains possible bias (explicit or heuristic), produce an extra candidate labe
"""



=== src/llm/generator.py ===

# 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, manager_comment=None, bias_info=None):
    """
    Generate comments for a single employee.
```

```
    If bias_info indicates bias, the prompt requests an extra unbiased rewrite candidate labeled D.
    """
    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),
        rto_compliance_pct=metrics.get('rto_compliance_pct', 0),
        release_on_time_percent=metrics.get('release_on_time_percent', 0),
        release_defect_ppm=metrics.get('release_defect_ppm', 0)
    )
    # If bias present, append manager comment and bias notes to user message so model can rewrite
    if bias_info and bias_info.get('is_biased'):
        prompt += f"\n\nManager comment (may be biased): {manager_comment}\nBias details: {json.dumps(bi
        prompt += "Also produce candidate D: an unbiased rewrite of the manager comment, grounded in the

    messages = [
        {"role":"system","content":"You are a professional assistant that writes HR-appropriate, unbiase
        {"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","raw":""}'

    try:
        parsed = json.loads(raw)
    except Exception:
        # model did not return strict JSON, encapsulate raw text
        parsed = {"raw": raw}
    return parsed

def generate_batch(employee_rows, features_df, joined_df=None):
    """
    employee_rows: list of dicts with employee_id and optional role
    features_df: dataframe with features
    joined_df: optional joined dataframe to pass manager comments/bias fields
    """
    suggestions = []
    for emp in employee_rows:
        eid = emp['employee_id']
        role = emp.get('role', 'N/A')
        feats = features_df[features_df['employee_id']==eid]
        metrics = feats.iloc[0].to_dict() if not feats.empty else {}
        manager_comment = None
        bias_info = None
        if joined_df is not None:
            row = joined_df[joined_df['employee_id']==eid]
            if not row.empty:
                row0 = row.iloc[0].to_dict()
                manager_comment = row0.get('comment')
                # detect bias using validator if present (we assume detection ran earlier)
                bias_info = {
                    "bias_type": row0.get('bias_type'),
                    "bias_notes": row0.get('bias_notes'),
                    "explicit_bias": bool(row0.get('bias_type')) or bool(row0.get('bias_notes'))
                }
        sug = generate_for_employee(eid, role, metrics, manager_comment=manager_comment, bias_info=bias_
        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/validators/red_flag_checker.py ===
```

```
# src/validators/red_flag_checker.py
import pandas as pd
from ..common.config import FILES
from ..common.logger import get_logger
from pathlib import Path

log = get_logger("red_flag_checker")

def load_red_flags(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(' ', '_')
        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 ===

# src/validators/alignment_checker.py
from ..common.config import RATING_MAP
from ..features.scoring_rules import has_positive_language

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):
    """
    Detect mismatch where comment tone is positive but rating is low.
    Returns dict with flags.
    """
    num = rating_to_numeric(rating_str)
    pos = has_positive_language(comment)
    mismatch = pos and num is not None and num <= 2
    return {"mismatch": mismatch, "positive_language": pos, "rating_numeric": num}

def detect_bias(evaluation_row):
    """
    Detect bias based on manager_evaluations fields:
      - bias_type (explicit)
      - bias_notes (explicit)
    Also perform heuristic checks: presence of certain biased words in comment.
    Returns a dict with bias flag and details.
    """
    comment = evaluation_row.get('comment', '') or ''
    bias_type = evaluation_row.get('bias_type') if evaluation_row.get('bias_type') and str(evaluation_ro
    bias_notes = evaluation_row.get('bias_notes') if evaluation_row.get('bias_notes') and str(evaluation

    # simple heuristic: if bias_type provided, mark biased
    explicit_bias = bool(bias_type)
    # heuristic: gendered or ageist terms - very small list for demo
    biased_terms = ['male','female','young','old','kids','mom','dad','boy','girl']
    comment_lower = comment.lower()
    heuristic_flags = [t for t in biased_terms if t in comment_lower]
    heuristic_bias = len(heuristic_flags) > 0

    is_biased = explicit_bias or heuristic_bias or bool(bias_notes)
    details = {
        "is_biased": is_biased,
```

```
        "explicit_bias": explicit_bias,
        "bias_type": bias_type,
        "bias_notes": bias_notes,
        "heuristic_flags": heuristic_flags
    }
    return details



=== src/llm/postprocess.py ===
# 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 []

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 ===

# src/webapp/streamlit_app.py
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:
    left, right = st.columns([2,3])
    with left:
        st.header('Team')
        # build display table with name & role if available
        display = joined[['employee_id']]
        if 'emp_name' in joined.columns:
            display['name'] = joined['emp_name']
        if 'emp_role' in joined.columns:
            display['role'] = joined['emp_role']
        display = display.drop_duplicates().reset_index(drop=True)
        sel = st.selectbox('Select employee', display['employee_id'].tolist())
        st.dataframe(display)
    with right:
        st.header('Employee 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)'))
        # Bias panel
        st.subheader('Bias & Fairness')
        bias_type = row.get('bias_type')
        bias_notes = row.get('bias_notes')
        if bias_type or bias_notes:
            st.error(f"Bias detected: {bias_type if bias_type else ''} {': ' + bias_notes if bias_notes
        else:
            st.success('No explicit bias fields found.')
        # features
        f = features[features['employee_id']==sel]
        if not f.empty:
            st.subheader('KPIs (selected)')
            # order columns for display according to hackathon priority
            show_cols = ['velocity','spillover_pct','defect_density_per_100_sp','recognition_rate','sent
            available = [c for c in show_cols if c in f.columns]
            st.table(f[available].T)
        # suggestions
        emp_sug = next((x for x in sugs if x.get('employee_id')==sel), None)
        if emp_sug:
            st.subheader('AI suggestions')
            # if bias exists, look for candidate D in suggestions
            candidates = emp_sug['suggestions'].get('candidates') if isinstance(emp_sug['suggestions'],
            if candidates:
                for c in candidates:
                    label = c.get('label','?')
                    comment = c.get('comment','')
                    st.markdown(f"**{label}** — {comment}")
                    st.write('IDP:', c.get('idp','(none)'))
                    st.write('Rationale:', c.get('rationale',[]))
                    if c.get('red_flags'):
                        st.warning('Red flags found in this generated comment: ' + ', '.join(c.get('red_
                    cols = st.columns([1,1,2])
                    if cols[0].button(f'Accept {label}', key=f'accept_{sel}_{label}'):
                        st.success(f'Accepted suggestion {label}')
                        # append to audit log
                        audit_path = OUTPUT_DIR / 'audit_log.csv'
                        import csv, datetime
                        rec = {'timestamp': datetime.datetime.utcnow().isoformat(), 'employee_id': sel,
                        write_header = not audit_path.exists()
                        with open(audit_path, 'a', newline='') as af:
                            writer = csv.DictWriter(af, fieldnames=list(rec.keys()))
                            if write_header:
                                writer.writeheader()
                            writer.writerow(rec)
                    if cols[1].button(f'Edit {label}', key=f'edit_{sel}_{label}'):
                        st.session_state[f'editing_{sel}_{label}'] = True
                    if st.session_state.get(f'editing_{sel}_{label}', False):
                        new_txt = st.text_area(f'Edit {label} comment', value=comment, key=f'ta_{sel}_{l
                        if st.button(f'Save edited {label}', key=f'save_{sel}_{label}'):
                            # save to audit
                            audit_path = OUTPUT_DIR / 'audit_log.csv'
                            import csv, datetime
                            rec = {'timestamp': datetime.datetime.utcnow().isoformat(), 'employee_id': s
                            write_header = not audit_path.exists()
```

```
                    with open(audit_path, 'a', newline='') as af:
                        writer = csv.DictWriter(af, fieldnames=list(rec.keys()))
                        if write_header:
                            writer.writeheader()
                        writer.writerow(rec)
                    st.success('Saved edited comment to audit log')
    else:
        st.info('No structured candidates found; raw model output may be present.')
else:
    st.info('No suggestions available. Run generator.')
```