Sprint 1 Kickoff: ML-based Recommender System

Learning Outcomes

- Apply Agile best practices to initialize a Product Backlog and run Sprint Planning.
- Translate *Epics* ➜ *User Stories (US)* ➜ *Tasks (issues)* with Acceptance Criteria.
- Configure two GitHub Projects:
  - **Product Backlog** (Epics & US as *draft issues*).
  - **Sprint 1 Kanban** (Tasks as *issues*).
- Assign every task to an individual and estimate hours.
- Prepare CI/CD hooks for data and model pipelines (DVC, MLflow).
- Maintain daily progress in the shared Excel (one row per student-day).

Agenda (timeboxed)

1. **(10')** Opening & goals recap.
2. **(20')** Create team repository & labels, templates, Projects.
3. **(25')** Backlog Refinement: import Epics → derive US with Acceptance Criteria.
4. **(25')** Sprint Planning: select Sprint 1 US, estimate story points.
5. **(35')** Task Breakdown: create issues, assign owners, estimate hours.
6. **(10')** Definition of Ready/Done, Excel tracking, next steps.

Roles

- **Product Owner (Fixed)**: validates scope & acceptance criteria, prioritizes backlog, ensures value delivery, starts sprint reviews.
- **Scrum Master (Fixed or rotating)**: flow, timeboxing, blockers, manage daily meetings, delivers Excel to instructor.
- **Developers / MLOps / Data Engineers / Data Analysts**: deliver work and keep boards & Excel updated.

Sprint 1 Kickoff: Student Guide

You will start the **ML-based Recommendation System** following Agile + MLOps best practices.

1) Create / prepare your team repository

- In your Organization, create a new repo: `team-<name>-recsys`.
- Add a minimal structure:

```
.github/
  workflows/
  ISSUE_TEMPLATE/   # epic.yml, user_story.yml, task.yml
data/                  # raw, interim, processed (use DVC)
notebooks/             # exploration
src/                   # packages for pipelines and API
```

```
  data/
  models/
  serving/
app/                    # streamlit frontend
api/                    # FastAPI endpoints
mlruns/                 # (MLflow local) or remote tracking URI
README.md
```

Add Issue Templates (Issue Forms)

To use the ready-made **Epic / User Story / Task** forms in GitHub:

- Put the YAML files in your repo at: `.github/ISSUE_TEMPLATE/`
  (files: `epic.yml`, `user_story.yml`, `task.yml`, `config.yml`)
- Commit them to the **default branch** (usually `main`).
  Alternative (org-wide): create a repo named `.github` and place the same folder there.
- Web UI option: **Settings → Issues → Set up templates** (or **Issues → New issue → Configure templates**) and **Upload files**.
- Verify: go to **Issues → New issue** and choose **Epic / User Story / Task**.

2) Create Labels

Go to your repo `Settings → Labels`.

Create labels with name, color, and description based on the `labels.csv`.

- Type: `epic`, `story`, `task`
- Domain: `data`, `pipeline`, `modeling`, `serving`, `frontend`, `infra`
- Status: `blocked`, `ready`, `needs-info`
- Priority: `P0`, `P1`, `P2`

3) Create *two* GitHub Projects

- **Product Backlog** → add *draft issues* for *Epics* and *User Stories*.
- **Sprint 1 Kanban** → add *issues* for **Tasks** only.

Recommended fields for both Projects:

- `Story Points` (number), `Priority` (enum P0/P1/P2), `Status` (Backlog/Selected/In Progress/Review/Done), `Assignee` (person).
- *(Optional)* **Projects v2 auto-add**: in **Product Backlog**, add a rule to auto-add items with label `story`; in **Sprint Kanban**, auto-add label `task`.

4) Seed the Backlog

Start with these **Epics** and derive **User Stories (US)** with acceptance criteria.

Project Goals

Align on key objectives for the project. Ensure all team members understand the desired outcomes.

- G1: *Data dashboard / understanding*
- G2: *Data storage (DVC)*
- G3: *Data cleaning pipeline (pandas/polars + GitHub Actions)*
- G4: *Model training pipeline (MLflow)*
- G5: *Model evaluation pipeline (MLflow)*
- G6: *Model registry & monitoring (MLflow)*
- G7: *Model serving (MLflow + FastAPI)*
- G8: *Model consumer app (Streamlit)*

EPICS (given personas)

- **E1 (Analyst)**: analyze & monitor data to enable a recommender.
- **E2 (Recommender Dev)**: batch recommendations available for recommender.
- **E3 (Data Scientist)**: reproducible experiments & results.
- **E4 (DevOps)**: model versioning registered & controlled.
- **E5 (Data Scientist)**: monitor performance & retrain easily.
- **E6 (User)**: get movie recommendations for tonight.
- **E7 (User)**: use a web app to search, store prefs & ratings.

User Stories (US)

For each **Epic**, define one or more **User Stories (US)** using the template:

- **Template:** *As a* `<persona>`*, I want* `<capability>` *so that* `<value>`.

Guidelines

- A US must be **small enough to fit in one sprint** and must include **Acceptance Criteria**.
- **Epics never have Acceptance Criteria**; US do.
- Tasks will be created later by **splitting selected US** during Sprint Planning (≤ 8h per task).

Suggested US per Epic

E1 (Analyst) → US1: As an Analyst, I want a data-profiling dashboard so that the team can understand and monitor data quality.

E2 (Recommender Dev) → US2: As a Recommender Developer, I want a batch job that produces top-N recommendations per user so that downstream services can consume them.

E3 (Data Scientist) → US3: As a Data Scientist, I want data, code, and parameters versioned so that experiments are fully reproducible.

E4 (DevOps) → US4: As a DevOps engineer, I want trained models automatically registered with version tags so that model history is controlled.

E5 (Data Scientist) → US5: As a Data Scientist, I want a monitoring process that tracks recommendation quality over time so that I can trigger retraining when performance drops.

E6 (User) → US6: As a User, I want quick movie recommendations for tonight based on simple inputs so that I can choose what to watch.

E7 (User) → US7: As a User, I want a web app to search, save preferences, and rate items so that future recommendations improve.

Start with these **Epics** and derive **User Stories (US)** with acceptance criteria.

> Create a **draft issue** per Epic and per US in the **Product Backlog Project**.

5) Acceptance Criteria examples

- **AC1**: "Given a data profile report is generated (e.g., pandas-profiling), when I open it in the dashboard, then I can see schema, missing values, distributions, and drift alerts."
- **AC2**: "Given DVC and a fixed params.yaml, when I re-run the pipeline on main, then metrics and artifacts match previous MLflow run within tolerance."
- **AC3**: "When a new model is trained, then it is registered to MLflow Model Registry with version tag and changelog."
- **AC4**: "Given a user genre + mood, when I request recommendations, then I receive ≥5 items ranked by predicted score in <1s on local."

Definition of Ready (DoR) for US

- Clear *persona* and *value*.
- Acceptance Criteria (testable).
- Dependencies/risks identified.
- Small enough (fits in a sprint) and estimated (SP).

Definition of Done (DoD) for Tasks

- Code & config committed, passing CI.
- Unit/test artifacts provided.
- Docs updated (README/US body).
- Demoable (when applicable).

6) Estimation

- **Stories** → use **Story Points** (Fibonacci: 1,2,3,5,8).
- **Tasks** → estimate **hours/person** (<= 8h per task for Sprint 1).

7) Task Breakdown and Kanban

- For each selected US, create tasks named: `US<id> – <concise task>`
  Examples: `US1 – Create data schema & profiling notebook`, `US4 – MLflow tracking URI setup`.
- Assign exactly **one owner**; add labels; move to **Sprint 1 Kanban**.
- Keep columns: `Backlog` → `Selected` → `In Progress` → `Review` → `Done`.

8) Daily Excel Tracking

- Check the Excel Template guide for Agile Managment for detailed instructions.

- Update all the sheets as per the guide:

    - **Release & Sprint Plan** → Sprints overview
    - **Product Backlog** → Epics & US
    - **PB Burndown** → overall progress
    - **Sprint Backlog** → daily task tracking
    - **Task Slips** → per-task mini-cards

9) Working Agreements

- Keep issues updated; move cards daily; respect WIP ≤ 2/person.
- Demo at the end of Sprint 1: minimal pipeline running, plus basic UI stub.

Assessment Signals

- Boards kept current (no unassigned tasks; WIP limited).
- Excel tracking consistent with assignments & estimates.
- PRs reviewed with actionable comments.
- Reproducibility: DVC/MLflow pipelines runnable end-to-end.