Machine Learning Project Starter Playbook (2025)

A structured, practical, end-to-end guide for taking an ML project from idea → production with quality, reproducibility, and ethics in mind.

Table of Contents

---

1) Mindset & Lifecycle

Principles - Start simple. Win with the smallest model that meets the goal. - Make it reproducible from day 1. Future-you will thank you. - Separate **science** (experiments) from **engineering** (reliable systems). - Measure what matters; ship measured improvements only. - Ethics and data rights are not optional.

Lifecycle (iterative) 1. **Frame** the problem and define success. 2. **Baseline** with simple, explainable models. 3. **Data**: collect/label, clean, split, validate. 4. **Model**: train → tune → compare. 5. **Evaluate** with the right metrics & uncertainty. 6. **Error analysis** and slice checks. 7. **Deploy** (batch/real-time) with safe guards. 8. **Monitor** performance, drift, and cost. 9. **Document** decisions (data/model cards) and lessons.

---

2) Prerequisites (Math, Coding, Tools)

Math (just enough to be dangerous) - Linear algebra: vectors/matrices, dot product, norms, eigenstuff, SVD. - Calculus: derivatives, chain rule, gradients; (DL: backprop intuition). - Probability & Stats: random variables, distributions, Bayes rule, expectation/variance, confidence intervals, hypothesis testing.

Programming - Python 3.10+; idiomatic NumPy, Pandas, scikit-learn; PyTorch (or JAX) for DL. - Packaging & testing: `pytest`, type hints (mypy/pyright), docstrings. - CLI & configs: `argparse` or Hydra/OmegaConf; `.env` for secrets (never commit!).

Data skills - SQL (SELECT/JOIN/GROUP BY), schema design basics. - Dataframes: merge, pivot, window ops; memory/time profiles.

Ops basics - Git & GitHub/GitLab; branches, PRs, code reviews. - Shell + Linux; Docker fundamentals.

---

3) Environment & Tooling (Reproducibility First)

Environment - Use `pyenv` + `venv`/**Poetry**/**Conda**. Pin versions. Save `requirements.txt` or `poetry.lock`. - Set random **seeds** (NumPy, PyTorch, Python) and control non-determinism when needed.

Hardware - Start on CPU; move to GPU only when it helps. Track cost/time.

Project structure (cookiecutter)

```
mlproject/
  README.md
  pyproject.toml or requirements.txt
  data/               # .gitignore large/raw files
    raw/ processed/ external/
  notebooks/
  src/
    __init__.py
    data/
      make_dataset.py   # ingestion/cleaning
    features/
      build_features.py
    models/
      train.py  predict.py  eval.py
    utils/
      io.py  metrics.py  plots.py  seed.py
  conf/
    config.yaml  params.yaml
  experiments/
    runs.csv  # experiment log
  models/
  reports/
    figures/
  tests/
    test_data.py  test_features.py  test_models.py
  .env.example  .gitignore  Dockerfile
```

Configuration - Keep a single `config.yaml` (data paths, model type, hyperparams, seeds). - Never hard-code file paths or magic numbers in code.

Experiment tracking - Log: config, code commit hash, dataset version, metrics, artifacts, run notes. - Tools: MLflow/W&B/DVC (choose one to avoid sprawl).

Data versioning - Use DVC or LakeFS; at minimum, checksum datasets; store schema.

---

4) Problem Framing & Project Charter

Questions to answer upfront - What decision will this model support? Who uses it? How often? - Task type (classification/regression/ranking/forecasting/NLP/CV/etc.). - Metric that aligns with the real goal (e.g., F1 for imbalance, AUC for ranking, MAE for money). - Constraints: latency, memory, interpretability, fairness, privacy, cost. - Baseline: heuristic or simple model + business as usual. - Success criteria: "Ship if test F1 $\geq$ 0.78 and p95 latency < 80 ms".

Mini Project Charter (fill this before coding) - **Problem statement:** - **Impact & users:** - **Primary metric + target:** - **Secondary metrics:** - **Data sources & rights:** - **Risks & harms:** - **Timeline & milestones:** - **Definition of Done:**

---

5) Data: Sourcing → Quality → Splits → Privacy

Sourcing & labeling - Verify **licenses/terms**; record provenance. - Labeling plan: guidelines, inter-annotator agreement (κ), spot checks.

Quality checks (automate) - Schema (types, ranges), missing values, duplicates, outliers. - Leakage scan: future info, target proxies, duplicated entities across splits. - Class balance & rare slices.

Splitting strategies - IID: `train/val/test` (e.g., 70/15/15) with **stratification** for classification. - **Group** split to keep entities (user/item/patient) in one split. - **Time-based** split for forecasting/temporal leakage avoidance. - Nested CV when data is scarce.

Feature engineering - Categorical: one-hot, target/statistical encoders (watch leakage!). - Numerical: scaling, winsorizing/clipping, log/Box-Cox for skew. - Text: TF-IDF, subword embeddings; Images: resize/ normalize/augment.

Class imbalance - Metric choice (PR-AUC/F1), threshold tuning. - Resampling (SMOTE), class weights, focal loss.

Privacy & security - Minimize PII; pseudonymize/aggregate where possible. - Access control, audit logs; secure secrets and keys.

---

6) Modeling: Baselines → Classical → Deep Learning

Start with baselines - Heuristic or majority/median; linear/logistic; decision tree; dummy predictor.

Classical ML - Go-to models: Logistic/Linear, Ridge/Lasso/ElasticNet, RandomForest, XGBoost/LightGBM, SVM. - Pros: strong tabular performance, interpretable, fast to iterate.

Deep Learning (when justified) - Use transfer learning for CV/NLP first. - Architectures: MLP (tabular), CNN/ViT (images), RNN/Transformers (sequence/NLP), TCN for time series. - Tricks: early stopping, dropout, weight decay, batch norm, mixed precision.

Hyperparameter optimization (HPO) - Random search > grid for high-dim spaces; consider Bayesian/TPE (Optuna). - Budgeting: fix max trials/time; save top-k checkpoints.

Regularization & robustness - L1/L2, dropout, data augmentation, label smoothing, early stopping. - Ensembling for a final squeeze (bagging/stacking) if latency allows.

---

7) Training & Experimentation

Pipelines - Deterministic `DataLoader` → model → loss → optimizer → scheduler. - Keep **data transforms** in one place; log all params.

Reproducibility - Fix seeds; log library versions; snapshot config+code with each run.

Checkpoints - Save best-val model; also periodic snapshots for recovery.

HPO workflow 1. Define search space. 2. Random/TPE search with early stopping. 3. Promote top candidates to longer training.

Compute efficiency - Mixed precision (AMP), gradient accumulation, smaller batch with accumulation, profiling to find bottlenecks.

---

8) Evaluation & Error Analysis

Pick the right metrics - **Classification:** Accuracy (only for balanced), Precision/Recall/F1, PR-AUC, ROC-AUC, log loss. - **Regression:** MAE (robust, money), RMSE (penalizes large errors), R^2. - **Ranking/RecSys:** MAP, NDCG@k, Hit@k, MRR, Coverage, Diversity. - **Forecasting:** sMAPE, MASE; backtest with rolling windows. - **NLP:** BLEU/ROUGE/METEOR; token-level F1 for NER; perplexity for LMs. - **CV:** mAP (detection), IoU/Dice (segmentation), Top-k accuracy.

Uncertainty & confidence - Report 95% CIs via bootstrap; show variance across seeds/folds.

Calibration - Reliability plots; temperature scaling or isotonic regression if needed.

Fairness & slices - Evaluate by sensitive/important slices (region/device/tenure). Watch trade-offs.

Error analysis loop 1. Build a confusion matrix and per-slice metrics. 2. Inspect high-loss examples; tag error types. 3. Hypothesize data/model fixes; iterate.

---

9) Deployment & Monitoring

Serving patterns - **Batch**: score offline (ETL/cron). Simple and cheap. - **Online**: FastAPI/gRPC microservice with autoscaling. Cache features. - **Streaming**: Kafka/Flink for low-latency updates.

Artifacts & registries - Store model (and preproc) together; version with a registry.

Monitoring - **Data**: schema drift, feature distribution shift, missing rates. - **Performance**: live metrics vs offline expectation; capture labels when possible for delayed ground truth. - **Ops**: latency, throughput, error rates, cost.

Release safety - Shadow deploy → canary (1–5%) → full rollout with auto-rollback.

---

10) MLOps Minimal Stack (Solo/Small Team Picks)

- **Lang/runtime:** Python 3.11
- **Core libs:** NumPy, Pandas, scikit-learn; PyTorch (DL)
- **Config:** Hydra or OmegaConf (YAML)
- **Tracking:** MLflow (runs, params, artifacts)
- **Data versioning:** DVC (S3/GDrive/remote)
- **Serving:** FastAPI + Uvicorn (Dockerized)
- **Orchestration (optional):** Prefect
- **HPO:** Optuna
- **CI/CD:** GitHub Actions
- **Quality:** black/ruff, mypy, pytest

Pick one in each category to avoid tool sprawl.

---

11) Templates

A) Config (YAML)

```
project:
  name: churn_model
  seed: 42
paths:
  data: data/processed/train.parquet
```

```
  artifacts: models/
model:
  type: xgboost
  params:
    n_estimators: 500
    max_depth: 6
    learning_rate: 0.05
train:
  cv_folds: 5
  early_stopping_rounds: 50
  metric: auc
```

B) Model Card (fill before release)

- **Model name & version**
- **Intended use**
- **Training data summary & rights**
- **Evaluation datasets & metrics (with CIs)**
- **Slices evaluated & fairness notes**
- **Limitations & known failure modes**
- **Safety mitigations & monitoring plan**
- **Owner & contact**

C) Data Card

- **Source & licensing**
- **Collection timeframe**
- **Schema & units**
- **Preprocessing & filters**
- **Biases/coverage gaps**
- **Quality checks & QA results**

D) Experiment Log (CSV columns)

`timestamp, git_sha, data_version, model_type, config_path, metric_primary, metric_secondary, train_time_min, notes`

E) README structure

- Problem & goals
- Data description
- Repro steps (`make` targets or commands)
- Results & how to reproduce tables/plots
- Deployment & monitoring notes

---

12) A 30-Day Plan (Milestones & Checklists)

Week 1 — Frame & Baseline - Write the **project charter**; define metrics & success. - Acquire sample data; run EDA; set up project skeleton. - Implement 2 baselines (heuristic + logistic/linear).

Week 2 — Data & Features - Finalize split strategy; lock schema & checks. - First feature set; track data version; start experiment logging.

Week 3 — Modeling & HPO - Train classical models; run small HPO; pick top 2. - Error analysis; targeted data fixes/augmentations.

Week 4 — Hardening & Deploy - Robust evaluation with CIs; threshold tuning. - Package a FastAPI batch/online service; add monitoring hooks. - Write Model/Data Cards; prepare launch notes.

---

13) Common Pitfalls & Anti-patterns

- Optimizing the wrong metric; ignoring business cost asymmetry.
- Data leakage via target encoding/time leakage.
- Comparing models on **val** but reporting **test** only after peeking.
- Training on mixed distribution; deploying to shifted reality.
- Over-engineering: 10 tools when 3 suffice.
- Skipping seeds and experiment logs.
- No rollback plan.

---

14) Glossary (Plain English)

- **Baseline:** The simplest method to beat (e.g., predict average).
- **Drift:** When live data looks different from training data.
- **Ensemble:** Combine multiple models to improve accuracy.
- **Feature store:** Central place to compute/serve features consistently.
- **Label leakage:** Using information at train time that won't exist at predict time.
- **Regularization:** Techniques that prevent overfitting by penalizing complexity.
- **Stratified split:** Keeps class proportions similar across splits.

Final Advice

- Bias for action: get a baseline in week 1.
- Log everything; write 1–2 sentences per experiment.
- Keep the simplest tool that works. Scale tools only when pain is real.

Bookmark this playbook and clone its structure for every new project.