# 1. Programming & Data Wrangling

## 1) Background / Definition of the Skill

Programming & data wrangling is the end-to-end craft of turning raw, messy, multi-source data into clean, well-typed, documented, and reproducible datasets that are ready for analysis or modeling. It spans acquiring data (files/APIs/DBs), validating schema and quality, parsing and transforming structures, handling missing/outlier values, joining disparate tables, reshaping (wide↔long), and packaging outputs with lineage so others—and future you—can trust and reuse them.

- Acquire → validate → clean → transform → integrate → document → deliver.
- Works across formats (CSV/JSON/Parquet), sources (SQL/NoSQL/APIs), and scales (pandas ↔ Spark/Dask).
- Emphasizes **correctness, reproducibility, and performance** as much as functionality.
- Produces artifacts: tidy tables, feature matrices, data dictionaries, and reusable scripts/pipelines.

## 2) Intelligence Needed to Excel in This Skill

Excelling requires a blend of algorithmic thinking, systems awareness, and ruthless attention to detail—because small data defects silently corrupt downstream insights.

- **Algorithmic & decomposition thinking:** break large messes into tractable steps (parse → type → dedupe → join → validate).
- **Probabilistic/uncertainty awareness:** reason about missingness mechanisms (MCAR/MAR/MNAR) and how imputations affect bias/variance.
- **Debugging mindset:** hypothesize → instrument → test; isolate defects with minimal examples.
- **Pattern recognition:** spot schema drift, mixed data types, unit inconsistencies, encoding issues.
- **Systems thinking:** understand the data's lifecycle (upstream producers, downstream consumers, SLAs).
- **Optimization intuition:** choose vectorized ops, push filters to the DB, minimize shuffles, sample smartly.
- **Communication clarity:** write crisp docstrings, data dictionaries, and commit messages others can follow.

## 3) Impact of This Skill for a Data Scientist

Strong wrangling multiplies your impact: models converge faster, dashboards don't break, and stakeholders trust the numbers—because inputs are reliable and pipelines are reproducible.

- **Speed & iteration:** cleaner inputs → fewer modeling detours; faster EDA and feature engineering.
- **Model quality:** well-typed, de-duplicated, correctly joined data reduces leakage and spurious correlations.
- **Reliability & trust:** documented lineage + validation checks prevent "why do numbers not match?" crises.
- **Collaboration:** standardized datasets/feature stores enable team reuse and consistent metrics.
- **Scalability & cost:** efficient queries and storage (partitioning, columnar formats) cut compute bills.
- **Compliance & risk:** PII handling, versioning, and audit trails reduce regulatory and reputational risk.

**Signals/KPIs of impact**

- Fewer data-quality incidents; faster time-to-first-model; % pipelines with automated tests; SLA adherence; reduction in duplicate transformations; reproducible runs from clean checkout.

## 4) Knowledge Needed to Learn This Skill

You'll need practical fluency with languages, libraries, storage/query systems, and the hygiene that keeps pipelines robust in the wild.

- **Core languages & libraries**
  - Python: pandas, numpy, pyarrow, polars (optional), datetime, regex, requests/httpx.
  - SQL: joins, subqueries, CTEs, window functions, aggregates, indexing, execution plans.
- **Data access & formats**
  - Files: CSV, JSON (nested), Parquet/Feather (columnar), Excel; compression (gzip, zstd).

    - APIs: REST basics, pagination, rate limits, auth (API keys/OAuth), backoff & retries.
    - Datastores: relational (Postgres/MySQL/BigQuery/Snowflake), basics of NoSQL (Mongo/Redis).
- **Cleaning & transformation**
    - Type casting, parsing dates/timezones, text normalization (Unicode, encodings), categorical handling.
    - Missing data strategies, outlier treatment, deduplication, fuzzy matching, schema enforcement.
    - Reshaping: melt/pivot, groupby-agg, joins/merges, rolling windows.
- **Performance & scale**
    - Vectorization vs loops; memory profiling; chunked I/O; pushdown predicates; indexes/partitions.
    - When to move to **Spark/Dask/Polars**; basics of distributed joins and shuffles.
- **Quality & reproducibility**
    - Data validation: pydantic, pandera, Great Expectations; unit tests for transforms.
    - Reproducible envs: venv/conda, requirements.txt/pyproject.toml, data versioning (DVC/LakeFS).
    - Logging & observability: structured logs, row counts, anomaly alerts, lineage (OpenLineage).
- **Ops hygiene**
    - Git basics (branches/PRs), code review, docstrings, notebooks ↔ scripts parity.
    - CLI/Bash fundamentals; scheduling (Airflow/Prefect/Cron) and idempotent jobs.
    - Security & governance: PII masking, access control, GDPR basics, audit trails.
- **Domain & units**
    - Understand business grain (row = what?), primary keys, unit conversions, time granularity, slowly changing dimensions (SCDs).

## 2. Statistics & Math

### 1) Background / Definition of the Skill

Statistics & math provide the language of uncertainty and the mechanics that make models work. Statistics turns noisy samples into defensible conclusions (estimation, testing, inference), while math—primarily linear algebra, calculus, and optimization—explains how learning algorithms represent data, fit parameters, and generalize.

- **Statistics (inferential):** estimation, confidence/credible intervals, hypothesis testing, effect sizes, power.
- **Probability:** random variables, distributions, expectations, variance, conditional probability, Bayes' rule.
- **Mathematical foundations:** linear algebra (vectors, matrices, eigendecomposition), calculus (gradients), convexity/optimization.
- **Applied areas:** regression/GLMs, time series, experimental design (A/B/n, power), Bayesian methods, resampling (bootstrap), causal inference (DAGs, confounding).
- **Goal:** quantify uncertainty, separate signal from noise, and choose models/assumptions appropriate to the data-generating process.

### 2) Intelligence Needed to Excel in This Skill

Excelling requires comfort with abstraction plus a habit of testing assumptions against reality.

- **Abstraction & formal reasoning:** translate real problems into variables, distributions, and constraints.
- **Modeling judgment:** choose priors/link functions/losses that match the data and stakes.
- **Uncertainty intuition:** reason with intervals, posteriors, and predictive distributions (not just point estimates).
- **Diagnostic mindset:** check residuals, stability, identifiability, collinearity, and misspecification.
- **Trade-off thinking:** bias–variance, variance–cost, precision–recall, Type I vs Type II error.
- **Causal skepticism:** distinguish correlation vs causation; anticipate confounders and selection bias.

- **Numerical sense:** sanity-check magnitudes, units, and condition numbers; detect data leakage.

## 3) Impact of This Skill for a Data Scientist

Strong statistics & math dramatically improve decisions, models, and credibility.

- **Better decisions:** well-powered experiments, interpretable effects, calibrated risk.
- **Stronger models:** appropriate loss functions, regularization, and validation → better generalization.
- **Fewer failures:** early detection of leakage, overfitting, non-stationarity, or spurious correlations.
- **Clear communication:** explain uncertainty and trade-offs to stakeholders; defend choices with evidence.
- **Faster iteration:** principled feature selection, diagnostics, and stopping criteria reduce wasted cycles.
- **Broader toolbox:** confidence to use GLMs, survival/time-to-event, Bayesian hierarchical models, causal estimators when needed.

**Practical impact signals (KPIs):**

- Reduced false alarms/regressions in A/B tests; higher experiment power at lower sample cost.
- Better calibration/Brier score; narrower yet valid intervals; reproducible conclusions across samples.
- Fewer late-stage model reversions due to hidden bias or drift.

## 4) Knowledge Needed to Learn This Skill

A pragmatic stack that balances theory with practice.

- **Probability & distributional thinking**
  - Random variables; expectation/variance; LLN/CLT (intuition over proofs).
  - Common families: Gaussian, Bernoulli/Binomial, Poisson, Exponential/Gamma, Beta/Dirichlet, Lognormal.
  - Transformations, mixtures, and tail behavior (heavy-tailed risks).
- **Statistical inference & experimentation**
  - Point/interval estimation; hypothesis tests; multiple testing (FDR).
  - Power analysis, sample-size planning; blocking/stratification; CUPED variance reduction.
  - Non-parametrics and resampling (bootstrap, permutation tests).
- **Regression & generalized linear models**
  - OLS assumptions/diagnostics; regularization (Ridge/Lasso/Elastic Net).
  - GLMs: logistic, Poisson/negative binomial; link functions and interpretation.
  - Multicollinearity, leverage, influence; robust regression (Huber/M-estimators).
- **Time series & dependence**
  - Stationarity, autocorrelation/partial autocorrelation; ARIMA/SARIMA basics.
  - Seasonality, trend, exogenous regressors; cross-validation with temporal splits.
  - Change-points and drift detection.
- **Bayesian fundamentals**
  - Priors/likelihood/posterior; conjugacy; posterior predictive checks.
  - Hierarchical models for partial pooling; MCMC vs variational inference (intuition).
  - When Bayesian beats frequentist (small data, pooling, decision costs).
- **Causal inference (applied)**
  - Potential outcomes, DAGs; exchangeability; back-door/front-door.
  - Estimators: matching, IPW, doubly robust methods; diff-in-diff; instrumental variables.
  - Assumption checks and placebo tests.
- **Math for ML**
  - **Linear algebra:** dot products, matrix factorization/SVD, eigenvalues, conditioning.
  - **Calculus & optimization:** gradients, convexity, line search; SGD variants and regularization.

    - **Numerics:** stability, scaling/normalization, conditioning; when to prefer closed-form vs iterative solvers.
- **Validation & measurement**
    - Proper scoring rules (log loss, Brier), calibration curves, AUC/PR-AUC trade-offs.
    - Uncertainty quantification: delta method, bootstrap CIs, Bayesian credible intervals.
    - Data splitting schemes: k-fold, stratification, group/time-based CV.
- **Tooling**
    - Python/R: statsmodels, scikit-learn, scipy.stats, pymc/cmdstanpy, causalml/DoWhy.
    - Experiment platforms and metric stores; notebooks + reproducible reports.

## 3. Machine Learning Fundamentals

### 1) Background / Definition of the Skill

Machine learning (ML) fundamentals are the core principles for turning data into predictive or descriptive models that generalize to unseen cases. At its heart, ML defines a **representation** of inputs (features), a **target** to learn (labels or structure), an **objective** (loss/score), an **optimizer** to fit parameters, and a **validation scheme** to estimate out-of-sample performance. Fundamentals also include choosing the right problem framing (supervised vs unsupervised vs reinforcement), preventing leakage/overfitting, and translating model outputs into decisions.

- What ML covers:
    - **Problem framings:** supervised (regression/classification), unsupervised (clustering, density, dimensionality reduction), time series/sequence, recommendation, anomaly detection.
    - **Pipeline:** define task → collect/label → split data → features → model → tune → validate → test → monitor.
    - **Generalization:** bias–variance trade-off, capacity/regularization, inductive biases.
    - **Evaluation:** appropriate metrics, uncertainty/calibration, business objectives/constraints.
    - **Ethics & safety:** fairness, privacy, robustness, and responsible deployment.

### 2) Intelligence Needed to Excel in This Skill

Success in ML requires structured problem-solving, statistical intuition, and rigorous experimentation. You must reason about uncertainty, isolate causes, and iterate rapidly without fooling yourself.

- **Problem decomposition:** convert vague goals into clear tasks, data grains, and metrics.
- **Modeling judgment:** when to favor simple linear baselines vs trees/GBMs vs kernels vs neural nets.
- **Statistical intuition:** bias–variance, class imbalance, confidence vs calibration, distribution shift.
- **Experimental rigor:** clean splits (IID vs time-based), ablations, reproducible seeds, proper baselines.
- **Debugging mindset:** diagnose learning failures (underfit/overfit, leakage, non-stationarity).
- **Optimization sense:** recognize when learning is plateauing; reason about learning curves.
- **Communication:** explain trade-offs (precision/recall, latency/accuracy) to non-experts.

### 3) Impact of This Skill for a Data Scientist

Strong ML fundamentals let you deliver models that are both **accurate and trustworthy**, cut iteration time, and make better product decisions. You avoid costly mistakes (leakage, mis-specified metrics), choose simpler solutions where possible, and know when added complexity truly pays off.

- **Product & business impact:** better rankings, conversions, risk screens, alerts → measurable ROI uplift.
- **Reliability:** fewer production regressions via robust validation and monitoring.
- **Speed:** faster convergence to “good enough,” fewer dead-end experiments.

- **Maintainability:** models that are interpretable enough to debug and improve.
- **Scalability:** architectures/feature sets that extend as data and use-cases grow.

**Impact signals/KPIs**

- Offline → online metric correlation; improved lift/PR-AUC/AUC/MAE vs baseline; calibrated predictions (Brier/log loss); reduced false positives/negatives at target thresholds; stable performance under drift.

## 4) Knowledge Needed to Learn This Skill

A practical, tool-agnostic base that you can apply across domains.

- **Core concepts**
  - Bias–variance trade-off, capacity/regularization (L1/L2, early stopping), VC/PAC (intuition), inductive bias.
  - Data leakage and prevention; feature leakage vs target leakage.
  - Learning curves; error decomposition; class imbalance strategies (resampling, costs, focal loss).
  - Distribution shift/drift: covariate, label, concept; detection and adaptation basics.
- **Data splitting & validation**
  - Train/validation/test discipline; stratification; group and **time-based** splits.
  - Cross-validation (k-fold, stratified, grouped, nested), rolling/blocked CV for time series.
  - Proper model selection vs final reporting; holdout sanctity.
- **Metrics (pick per task)**
  - Regression: MAE, RMSE, $R^2$, MAPE, pinball loss (quantiles).
  - Classification: Accuracy (with caution), Precision/Recall/F1, ROC-AUC, PR-AUC, log loss, Brier; calibration curves.
  - Ranking/reco: MAP, NDCG, recall@k, coverage, diversity, serendipity.
  - Anomaly: ROC/PR with extreme imbalance, recall at fixed FPR.
  - Operational: latency, throughput, memory, stability.
- **Model families & when to use them**
  - **Linear/GLMs:** linear & logistic regression, regularized (Ridge/Lasso/Elastic Net); fast, interpretable baselines.
  - **Tree-based:** decision trees, Random Forests, Gradient Boosting (XGBoost/LightGBM/CatBoost); tabular SOTA workhorses.
  - **KNN / kernel / SVM:** useful with engineered features or smaller datasets.
  - **Naive Bayes:** high-bias, strong text baseline.
  - **Unsupervised:** k-means, GMM, hierarchical, DBSCAN; **Dimensionality reduction:** PCA, truncated SVD; manifold tools (t-SNE/UMAP) for exploration.
  - **Time series:** ARIMA/SARIMA (baselines), regressors with lags/rolling features, Prophet (quick baselines).
- **Feature engineering & preprocessing**
  - Encoding: one-hot, target encoding (with leakage-safe schemes), embeddings (conceptually).
  - Scaling/normalization; handling missingness; interaction terms and transformations.
  - Text basics: tokenization, TF-IDF, n-grams.
  - Time-aware features: lags, windows, holiday/seasonality encodings.
  - Leakage-safe pipelines (fit on train only, transform on val/test).
- **Hyperparameter tuning**
  - Grid/Random search; **Bayesian optimization** (e.g., Optuna), early stopping, successive halving.
  - Search spaces, seed control, reproducible artifacts; tuning objectives aligned with business metrics.
- **Interpretability & validation beyond accuracy**
  - Global vs local explanations: permutation importance, partial dependence/ICE, SHAP (conceptual usage), counterfactual checks.

    - Stress tests: slice analysis (by segment), adversarial/perturbation tests, fairness metrics (demographic parity, equalized odds).
- **Experiment management & hygiene**
    - Reproducibility: seeds, data snapshots, versioned features.
    - Tracking: MLflow/W&B (params, metrics, artifacts); clear baselines and ablations.
    - Clean code & notebooks: pipelines, tests for data transforms and metrics.

## Deep Learning & GenAI

### 1) Background / Definition of the Skill

Deep learning (DL) uses multi-layer neural networks to learn hierarchical representations from data, enabling state-of-the-art performance on images, text, audio, video, and multi-modal inputs. Generative AI (GenAI) focuses on models that **produce** content—text, images, code, audio—most notably large foundation models (e.g., Transformers) pre-trained on massive corpora and adapted via fine-tuning, prompting, or retrieval. Together, DL + GenAI span the lifecycle from representation learning to controllable generation, with modern practice emphasizing **data/compute scaling**, **transfer learning**, **prompt/program design**, and **safe deployment**.

- Neural networks learn features automatically (vs. manual feature engineering).
- Architectures: CNNs (vision), RNNs/Seq2Seq (legacy sequence), **Transformers** (current SOTA for text/vision/audio/multimodal).
- Foundation models: pre-trained on broad data; adapted by **fine-tuning**, **parameter-efficient tuning** (LoRA/adapters), **RAG** (retrieval-augmented generation), or **prompting**.
- Objectives: predictive (cross-entropy, MSE) and generative (language modeling, diffusion).
- Emphasis on **evaluation, safety, and guardrails** due to open-ended generation.

### 2) Intelligence Needed to Excel in This Skill

Excelling requires a blend of mathematical intuition, empirical rigor, and systems thinking—plus the craft of controlling large models.

- **Representation & abstraction:** reason about embeddings, attention, inductive biases.
- **Empirical discipline:** design fair ablations, read learning curves, separate data/compute/batch effects.
- **Optimization intuition:** diagnose under/over-fit, vanishing/exploding grads, LR schedules, regularization.
- **Systems mindset:** manage GPUs, mixed precision, memory/throughput trade-offs, data pipelines.
- **Product sense:** convert capabilities into user value under latency/cost constraints.
- **Prompt & interface design:** structure inputs, constraints, and tools to steer generative behavior.
- **Risk awareness:** anticipate failure modes (hallucination, bias, jailbreaks, toxicity) and mitigation tactics.

### 3) Impact of This Skill for a Data Scientist

Deep learning and GenAI unlock unstructured data at scale and supercharge productivity—from automating annotation to building intelligent copilots—while enabling new product surfaces (chatbots, summarizers, vision classifiers, speech interfaces).

- **Broader problem coverage:** text, code, image, audio, video, multimodal fusion.
- **Higher accuracy:** surpasses classic ML on complex perceptual/sequence tasks.
- **Leverage via transfer:** strong results with limited labeled data (few-shot, PEFT, RAG).
- **Workflow acceleration:** automated EDA summaries, feature ideation, test generation, documentation.

- **New experiences:** conversational search, personalized recommendations, creative generation.
- **Measurable business impact:** improved conversion/retention, reduced support cost, faster ops.
- **Defensibility:** proprietary data + safe deployment pipelines become competitive moats.

**Impact signals/KPIs**

- Uplift on task metrics (e.g., top-1/top-k, Rouge/BERTScore for text, mAP for vision).
- Reduced time-to-value: faster protos, fewer labeled examples needed.
- Production metrics: latency, cost per 1k tokens/images, hallucination rate, guardrail catch rate, safety incident frequency.

## 4) Knowledge Needed to Learn This Skill

A practitioner stack that covers theory, training, adaptation, evaluation, and deployment—with special focus on reliability and cost.

- **Foundations**
  - Math: linear algebra (matrix ops, SVD), calculus (gradients), probability (distributions), optimization (SGD/Adam, momentum, weight decay).
  - NN basics: MLPs, activations (ReLU/GELU), initialization, normalization (Batch/LayerNorm), dropout, residual connections.
- **Architectures & objectives**
  - **Transformers:** self-attention, positional encodings/rotary, encoder/decoder, causal vs bidirectional.
  - **Vision:** CNNs, Vision Transformers, detection/segmentation heads.
  - **Generative models:**
    - Language modeling (auto-regressive).
    - Diffusion (images/audio/video) and VAEs; guidance, schedulers, CFG.
  - **Sequence & time series:** temporal convolutions, attention over time, forecasting heads.
  - **Recommendation basics:** embeddings, two-tower, sequence models.
- **Training & efficiency**
  - Data pipelines: tokenization, augmentation (MixUp/CutMix), deduplication, curriculum.
  - Schedules: warmup, cosine decay, early stopping; gradient clipping; mixed precision (fp16/bf16).
  - Distributed training: data/model/pipeline parallelism; ZeRO, checkpointing.
  - **Parameter-efficient fine-tuning (PEFT):** LoRA/adapters, prefix tuning, IA3.
  - **Continual learning & domain adaptation:** avoid catastrophic forgetting.
- **Adaptation & control**
  - **Prompting patterns:** instruction following, few-shot, tool-use, function calling, grounding with schemas.
  - **RAG:** indexing (vector DBs), chunking, retrieval strategies (BM25 vs dense), re-ranking, caching.
  - **Finetuning choices:** SFT vs DPO/RLHF (concepts), supervised vs preference data, eval of preference models.
  - **Constraint techniques:** system prompts, output schemas, guards, refusal policies.
- **Evaluation & safety**
  - Task metrics:
    - Text: exact match/F1, Rouge, BLEU, BERTScore; factuality/hallucination checks.
    - Vision: accuracy, mAP, IoU, FID for generation.
    - Code: pass@k, unit-test pass rate.
  - **Robustness:** adversarial/perturbation tests, slice/stress evals, OOD detection.
  - **Safety & ethics:** bias/toxicity testing, red-teaming, jailbreak defenses, privacy (PII filtering), copyright.
  - **Human evaluation:** rubric design, inter-rater reliability, cost/quality balance.
- **Deployment & operations**

    - Serving: Torch/TensorRT, TF-Serving, vLLM/text-generation inference; batch vs streaming.
    - **Optimization:** quantization (INT8/FP8), pruning, distillation, speculative decoding, KV-cache.
    - **Systems:** GPU/CPU autoscaling, token/image budgeting, caching layers.
    - Monitoring: drift/factuality/safety monitors; feedback loops; Canary/A-B tests.
    - Cost management: prompt budgets, retrieval hit-rates, model routing (small → large), offline vs realtime mix.
- **Tooling**
    - Frameworks: **PyTorch/TensorFlow**, HuggingFace (Transformers/Datasets/PEFT), Lightning, Accelerate.
    - Retrieval stack: FAISS/ScaNN, vector DBs (e.g., Milvus, Pinecone), re-rankers.
    - Experiment tracking: MLflow/W&B; dataset/version control (DVC/LakeFS).
    - Evaluation harnesses: prompt test suites, golden sets, unit tests for prompts/chains.

## 5. Data Engineering Basics

### 1) Background / Definition of the Skill

Data engineering is the discipline of **designing, building, and operating** the systems that move, transform, store, and serve data—reliably, securely, and at scale. It turns scattered operational data (OLTP) into analytics-ready, governed datasets (OLAP) via pipelines and storage layers with clear **lineage, quality checks, and SLAs**. For a data scientist, it's the substrate that makes analysis and ML possible—and repeatable.

- Build ingestion → transformation → storage → serving layers.
- Orchestrate batch/stream pipelines with observability and recovery.
- Enforce governance: schemas, access control, PII handling, retention.
- Deliver "contracted" data products (tables/feature-sets) to downstream users.

### 2) Intelligence Needed to Excel in This Skill

You need systems thinking, pragmatism, and an obsession with correctness under failure.

- **Systems thinking:** reason about throughput, latency, backpressure, idempotency, eventual consistency.
- **Reliability mindset:** design for retries, exactly-once/at-least-once semantics, schema evolution, disaster recovery.
- **Optimization sense:** choose partitioning, file sizes, indexes; minimize shuffles and I/O.
- **Contract thinking:** define interfaces (schemas/SLAs) between producers and consumers; manage breaking changes.
- **Security & compliance awareness:** least-privilege access, data masking/tokenization, auditability.
- **Cost discipline:** understand storage/compute egress; engineer for cost/performance balance.
- **Operational literacy:** alerting, runbooks, incident response, postmortems.

### 3) Impact of This Skill for a Data Scientist

Good data engineering multiplies a DS team's output and credibility; bad pipelines create silent errors and endless rework.

- **Faster research & modeling:** stable, well-documented datasets reduce time-to-first-model and iteration loops.
- **Higher model quality:** consistent features, correct joins, and de-duplicated records prevent leakage and bias.
- **Reliability & trust:** reproducible jobs with data tests stop "numbers-don't-match" escalations.

- **Scalability:** pipelines that handle growth (volume/velocity/variety) without rewrites.
- **Compliance & risk reduction:** governed access to sensitive data; clear lineage for audits.

**Impact signals / KPIs**

- % pipelines with automated data tests; SLA adherence; recovery time (MTTR); data freshness lag; cost per TB processed; defect rate (bad records caught upstream); consumer satisfaction (fewer ad-hoc extracts).

## 4) Knowledge Needed to Learn This Skill

A pragmatic stack spanning ingestion, transformation, storage, orchestration, quality, and governance.

- **Data modeling & storage**
  - OLTP vs OLAP; **star/snowflake** schemas; normalization vs denormalization; slowly changing dimensions (SCD).
  - **File formats:** row (CSV/JSON) vs columnar (**Parquet/ORC**) and when to use each.
  - **Lake/Lakehouse/Warehouse:** object storage (S3/GCS/ADLS) + table formats (Delta/Iceberg/Hudi) vs BigQuery/Snowflake/Redshift.
  - Partitioning & clustering strategies; Z-ordering/sorting; small-files problem & compaction.
- **Ingestion & integration**
  - **Batch:** scheduled pulls, CDC snapshots.
  - **Streaming/CDC:** Kafka/Kinesis/Pub/Sub; Debezium for DB change capture; exactly-once vs at-least-once trade-offs.
  - API ingestion: auth, pagination, rate limits, retries/backoff; idempotent upserts/merges.
- **Transformation engines**
  - **SQL** (warehouse-native), **Spark** (PySpark), **Dask/Polars** for scaling beyond pandas.
  - ELT with **dbt** (modularity, tests, docs); window functions, UDFs, incremental models.
  - Performance tuning: pushdown predicates, broadcast joins, skew handling, caching.
- **Orchestration & scheduling**
  - **Airflow/Prefect/Dagster**: DAGs, retries, SLAs, backfills, parametrized runs, secrets management.
  - Idempotency & re-runs; event-driven vs time-based scheduling; data-aware triggers.
- **Data quality & observability**
  - **Testing/validation:** Great Expectations, dbt tests, **pandera**; contract tests at interfaces.
  - **Monitoring:** freshness, volume, schema, distribution drift, row-level anomalies; lineage (OpenLineage/Marquez).
  - **Incident response:** alert routing, SLOs, dashboards, runbooks.
- **Security, privacy, and governance**
  - Access control (RBAC/ABAC), secrets/key management (KMS), encryption at rest/in transit.
  - PII detection/classification; masking, tokenization, differential privacy (concepts).
  - Data catalogs (Amundsen/DataHub), metadata management, retention & deletion policies.
- **Feature delivery for ML**
  - **Feature stores** (Feast/Tecton/Databricks FS): offline/online consistency, point-in-time correctness, backfills.
  - Sliding windows, late-arriving data, training-serving skew prevention.
- **Cost & FinOps**
  - Storage lifecycle policies, compression, file sizing; warehouse slot management; cost attribution (tags/labels).
  - Caching, result reuse, and pruning to reduce scan costs.
- **Ops hygiene**
  - Git workflows, code reviews, CI/CD for pipelines (tests + lint + deploy).
  - Infra-as-code (Terraform) for reproducible environments; containerization (Docker) for portability.

# MLOps / Productionization

## 1) Background / Definition of the Skill

MLOps is the discipline of taking models from notebooks to **reliable, monitored, and cost-aware production services**. It blends software engineering, data engineering, and DevOps to manage the **full ML lifecycle**: data/version control → training & tuning → packaging → CI/CD → deployment (batch/online) → monitoring (data, model, system) → governance & rollback. The goal is **repeatability and safety**: anyone on the team can reproduce a model, ship it behind clear contracts, observe it in the wild, and improve it without breaking downstream users.

- End-to-end lifecycle: *data → train → evaluate → register → deploy → monitor → iterate*.
- Treats models as artifacts with versions, metadata, and approvals.
- Emphasizes automation (CI/CD), observability (metrics/logs/traces), and governance (review, audit).
- Supports multiple delivery patterns: batch scoring, real-time APIs, streaming, on-device.

## 2) Intelligence Needed to Excel in This Skill

You need systems thinking, risk management, and a bias for automation—plus the judgment to keep solutions simple.

- **Systems & reliability mindset:** design for failure, timeouts, retries, graceful degradation, and rollbacks.
- **Experiment discipline:** track lineage (data/code/params) so results are reproducible and auditable.
- **Automation instinct:** codify manual steps (tests, builds, validations) into pipelines.
- **Operational judgment:** balance latency, accuracy, and cost; right-size infra and models.
- **Monitoring intuition:** decide what to measure (data drift, quality, performance, business KPIs) and thresholds for action.
- **Security & governance awareness:** isolate secrets, enforce least privilege, manage PII & compliance.
- **Communication & change management:** align stakeholders on SLAs, release cadence, and rollback criteria.

## 3) Impact of This Skill for a Data Scientist

MLOps turns one-off experiments into **durable product capabilities**, reducing firefighting and accelerating learning cycles.

- **Faster iteration & safer releases:** CI/CD gates catch issues before prod; blue-green/canary reduce blast radius.
- **Stable performance:** monitoring detects drift and regressions early; automated retraining keeps models fresh.
- **Lower total cost:** right-sized serving, caching, and batching cut infra spend.
- **Trust & compliance:** versioned artifacts, approvals, and audit trails satisfy internal and regulatory checks.
- **Team scalability:** shared pipelines and feature/model registries enable reuse and consistent practices.

**Impact signals / KPIs**

- Lead time to production (idea → deploy).
- Change failure rate & mean time to recovery (MTTR).
- Serving SLOs: p95 latency, availability, cost per 1k predictions.
- Data/model quality: drift alerts resolved, calibration stability, business KPI lift maintained post-deploy.
- % of pipelines covered by tests and automated checks.

## 4) Knowledge Needed to Learn This Skill

A pragmatic stack covering packaging, deployment patterns, monitoring, and governance—tool-agnostic but production-grade.

- **Experiment tracking & artifact management**
  - Run tracking (params, metrics, plots), dataset snapshots, model registry (staging/production), lineage.
  - Reproducibility: fixed seeds, environment locks (requirements.txt/poetry.lock/containers).
- **Packaging & interfaces**
  - Model packaging (Python package, Docker image, serialized artifacts).
  - Serving contracts: REST/gRPC schemas, prediction & health endpoints, input validation.
  - Batch interfaces: file/Table I/O, feature store reads, idempotent writes.
- **Deployment patterns**
  - **Batch scoring:** scheduled jobs on tables/files; great for non-latency-sensitive cases.
  - **Online inference:** microservices with autoscaling, request batching, caching, and circuit breakers.
  - **Streaming:** consume events (Kafka/Pub/Sub), maintain online features, emit predictions.
  - **Edge/on-device:** quantization, model distillation, local caching, update channels.
- **CI/CD for ML**
  - Tests: unit (data transforms), integration (end-to-end), regression (golden datasets), performance tests.
  - Build steps: lint/format, type checks, security scans, image build, schema validation.
  - Promotion: dev → staging (shadow/canary) → prod with approval gates.
- **Data & feature management**
  - Feature stores: **offline/online** consistency, point-in-time correctness, backfills, TTLs.
  - Data contracts with producers; schema evolution strategy; PII handling and access control.
- **Monitoring & observability**
  - **Inputs:** schema/freshness/volume checks; covariate drift (PSI, KL, KS).
  - **Outputs:** calibration, residuals, class balance, uncertainty; concept drift.
  - **System:** latency, error rate, saturation; logs/traces; SLO dashboards & alert routing.
  - **Business:** conversion, risk loss, revenue, SLA compliance; A/B and holdout monitors.
  - Feedback loops: capture labels/outcomes for retraining and post-deploy evaluation.
- **Retraining & lifecycle automation**
  - Triggers: time-based, data-driven (drift), business events.
  - Pipelines: feature build → train → evaluate against golden set → bias/fairness checks → register → roll forward or rollback.
  - Model cards & documentation for each release.
- **Performance & cost engineering**
  - Quantization, pruning, distillation; vectorized inference; GPU/CPU routing; dynamic batching; caching embeddings/scores.
  - Capacity planning, autoscaling rules; cost attribution and budgets.
- **Security & compliance**
  - Secret management (vault/KMS), network policies, image signing, SBOMs, supply-chain security.
  - Data minimization, encryption at rest/in transit, audit logs, retention policies.
  - Responsible AI: bias tests, explainability reports, human-in-the-loop where needed.
- **Tooling (examples, not mandates)**
  - Tracking/registry: MLflow, Weights & Biases.
  - Orchestration: Airflow, Prefect, Dagster; CI (GitHub Actions/GitLab CI).
  - Serving: FastAPI, BentoML, KFServing/Seldon, Ray Serve, NVIDIA Triton, vLLM/TGI (for LLMs).
  - Monitoring: Prometheus/Grafana, OpenTelemetry, Evidently, WhyLabs, Arize.
  - Feature stores: Feast, Tecton, Databricks FS.