1) Role Definition & Scope — Data Scientist

Mission & value proposition

A Data Scientist turns raw data into reliable decisions, products, and measurable business impact. The mission is to reduce uncertainty with evidence, build models that generalize, and create feedback loops so the organization learns faster than competitors.

- Turn data into decisions: hypotheses → experiments/models → actions → measured impact
- Increase revenue/retention and reduce cost/risk via better targeting, automation, and forecasting
- Shorten decision cycles with reproducible analyses, dashboards, and self-serve tools
- Build durable assets: high-quality datasets, features, models, and documentation
- Institutionalize learning: A/B tests, post-mortems, model monitoring, knowledge bases

Boundaries vs. Data Analyst / ML Engineer / Data Engineer

Data Science overlaps adjacent roles but has distinct ownership over problem framing, modeling, and causal/decision rigor.

- **Data Analyst**
  - Focus: descriptive/diagnostic analytics, dashboards, KPI health, ad-hoc queries
  - Primary artifacts: reports, visualizations, metric definitions
  - DS difference: heavier inference/prediction, experimentation, and model building
- **ML Engineer**
  - Focus: productionizing models, serving/latency, pipelines, infra, scalability
  - Primary artifacts: services, feature stores, CI/CD, monitoring systems
  - DS difference: chooses model/class of solutions, validates assumptions, owns evaluation
- **Data Engineer**
  - Focus: data ingestion, ETL/ELT, warehousing, quality, lineage, governance
  - Primary artifacts: schemas, transformations, data contracts, reliability SLAs
  - DS difference: consumes/defines data requirements, creates features, validates suitability
- **Practical boundary rules**
  - DS frames the question, designs evaluation, builds/validates models; MLE ships them reliably
  - DS specifies data needs and quality gates; DE ensures trustworthy, timely data access
  - DS partners with DA to scale insights and codify metrics into decision tools

Problem types (prediction, inference, optimization, insight)

Data Scientists select the right problem framing before selecting methods—often blending types.

- **Prediction** (What will happen?)
  - Examples: churn, demand, CTR, LTV, fraud probability
  - Methods: supervised learning (trees/boosting, GLMs, DL)
  - Deliverables: calibrated scores, rankers, thresholds, ROC/PR curves
- **Inference/Explanation** (Why/what drives it?)
  - Examples: drivers of conversion, treatment effects, elasticity
  - Methods: causal inference, regression with interpretability, SHAP, experiments
  - Deliverables: effect sizes, confidence intervals, decision guidance
- **Optimization** (What should we do?)
  - Examples: budget allocation, pricing, routing, recommendation policies
  - Methods: bandits, RL, linear/convex programming, stochastic optimization

    - Deliverables: policies, constraints, scenario analyses, payoff curves
- **Insight/Discovery** (What patterns exist?)
    - Examples: segmentation, anomaly detection, topic discovery
    - Methods: clustering, embeddings, dimensionality reduction, unsupervised DL
    - Deliverables: segments/personas, hypotheses, opportunity maps

Contexts (product, growth/marketing, operations, research)

The same skills express differently across business contexts; expectations and KPIs change accordingly.

- **Product**
    - Work: ranking/reco, search quality, personalization, safety
    - KPIs: engagement, retention, revenue per user, latency, fairness
- **Growth/Marketing**
    - Work: targeting/propensity, LTV forecasting, media mix/model lift
    - KPIs: CAC/LTV ratio, incremental lift, ROI, attribution confidence
- **Operations**
    - Work: demand/supply forecasting, workforce planning, quality/defects, logistics
    - KPIs: SLA adherence, cost/unit, utilization, forecast accuracy (MAPE/RMSE)
- **Research/Advanced Methods**
    - Work: novel modeling, causal frameworks, GenAI/LLM evaluation, experimentation science
    - KPIs: published/internal tech notes, method adoption, upstream improvements

Success criteria & ownership (from metric movement to shipped models)

Success is defined by shipped solutions that are correct, reliable, and move the target metric—safely and repeatably.

- **Business impact**: clear north-star (e.g., retention +x%), guardrails (e.g., fairness, latency)
- **Model quality**: offline metrics (AUC/F1/RMSE), calibration, stability across cohorts
- **Causality & experimentation**: valid uplift, power/MDE met, no p-hacking
- **Operationalization**: SLAs (latency/uptime), monitoring (drift, data quality), alerting
- **Adoption**: stakeholders use the output in decisions; documentation lowers onboarding cost
- **Governance**: privacy/compliance, reproducibility, versioning, rollback plans
- **Ownership boundaries**
    - DS owns: problem framing, data & feature requirements, modeling, evaluation plan, experiment design, decision recommendations
    - With MLE: serving pattern choice, deployment pathway, monitoring specs
    - With PM/Leads: success metrics, launch criteria, iteration cadence

Key stakeholders (PM, Eng, Ops, Leadership)

Effective DS work is cross-functional; tailor artifacts and communication to each audience.

- **Product Manager (PM)**
    - Cares about: business outcome, prioritization, trade-offs, timelines
    - How to work: align on problem/metrics, design experiments, define launch gates
- **Engineers (ML/Backend/Data)**
    - Cares about: feasibility, interfaces, reliability, cost, performance
    - How to work: write clear specs (schemas, contracts), tests, and monitoring requirements
- **Operations / GTM**

    - Cares about: workflows, capacity, playbooks, exception handling
    - How to work: deliver actionable outputs (queues, thresholds), SOPs, training
- **Leadership (Functional/Exec)**
    - Cares about: ROI, risk, strategy, resourcing, compliance
    - How to work: concise narratives with alternatives, scenario impact, evidence strength
- **Design/UX & Analytics**
    - Cares about: interpretability, user experience, metric clarity
    - How to work: co-design explainable surfaces, consistent metric definitions
- **Legal/Privacy/Security**
    - Cares about: data use, consent, auditability, bias/safety
    - How to work: early reviews, data minimization plans, model cards, audit trails

2) Job Families & Specializations — Data Scientist

NLP (classification, NER, RAG, LLM apps)

Natural Language Processing turns unstructured text into structured signals and decisions. Modern NLP blends classic supervised tasks (classification/NER) with retrieval-augmented generation (RAG) and LLM-powered apps. **What you do (in depth):** Build datasets from logs/docs/chats, define label taxonomies, choose model families (linear → transformers), evaluate with task-appropriate metrics, and ship APIs or in-product features with monitoring for drift, toxicity, and hallucinations.

- Core tasks
    - Text classification (intent, sentiment, topic) → Accuracy/F1, calibration
    - NER (entities, PII) → Precision/Recall/F1 per class, span F1
    - QA & RAG → Exact Match/F1; retrieval: Recall@k, nDCG; generation: factuality checks
    - Summarization/rewrites → ROUGE/BLEU + human eval (clarity, faithfulness)
- Data & features
    - Label schema design, inter-annotator agreement, active learning loops
    - Tokenization pitfalls (multi-lingual, emoji, code), domain lexicons
- Modeling approaches
    - Baselines (logreg + TF-IDF), small transformers (DistilBERT), domain-tuned LMs
    - RAG: chunking, embeddings, vector stores, reranking; prompt templates
- Production concerns
    - Guardrails: profanity/PII filters, allowed topics, citation requirements
    - Latency vs cost (prompt size, temperature, context window)
    - Hallucination mitigation: retrieval grounding, constrained decoding, answerability detectors
- Anti-patterns
    - Training on test leakage (FAQ overlap), vague labels, no human eval for generative tasks

Computer Vision (detection, segmentation, OCR)

CV converts pixels into objects, masks, and text—used in quality control, document processing, safety, and AR. **What you do (in depth):** Collect labeled images/video, manage class imbalance, pick task-specific models (detection vs segmentation vs OCR), and deploy on cloud or edge with careful latency/throughput trade-offs.

- Core tasks
    - Object detection → mAP, latency (ms), FPS constraints
    - Semantic/instance segmentation → IoU/mIoU, boundary quality
    - OCR & document AI → word/line CER/WER, entity extraction accuracy

- Data & augmentation
    - Multi-angle, lighting/weather, synthetic data, mosaic/cutmix, copy-paste
    - Dataset versioning, per-class metrics, hard-negative mining
- Modeling approaches
    - Detection: YOLO/RetinaNet/DETR families; Segmentation: U-Net/Mask R-CNN/SegFormer
    - OCR pipelines: text detection (DB/EAST) + recognition (CRNN/Transformers)
- Production concerns
    - Edge deployment (TensorRT, ONNX, quantization), batching, NMS tuning
    - Monitoring: drift via color histograms, embedding shift, class frequency changes
- Anti-patterns
    - Training only on "hero" angles, ignoring rare classes, not annotating occlusions/partial objects

Recommender Systems (ranking, retrieval, bandits)

Recsys matches users to items (content, products, ads). It's a two-tower world: fast retrieval then precise ranking, with exploration to avoid stagnation.

What you do (in depth): Design objectives (engagement vs revenue vs diversity), build features (user/item/context), architect retrieval + ranking stacks, and validate online with A/B tests and guardrails.

- Problem framing
    - Retrieval (candidate generation) → Recall@k; ANN/vector search
    - Ranking (score ordering) → nDCG/MAP/AUC; calibration for click-through
    - Policies (bandits) for explore/exploit; slate/diversity constraints
- Features & signals
    - Short-/long-term behavior, content embeddings, popularity priors, recency decay
    - Bias handling (position bias, selection bias), counterfactual logging
- Modeling approaches
    - Two-tower retrieval (user/item embeddings), GBDT/Deep models for ranking (XGBoost/DeepFM/Transformer)
    - Contextual bandits/Thompson sampling for exploration
- Online evaluation
    - Guardrails (session length, dwell time, negative feedback), win-rate by cohort
    - Interference controls (holdouts, ghost experiments, CUPED for variance reduction)
- Anti-patterns
    - Optimizing only CTR (clickbait), ignoring saturation/novelty, neglecting cold-start strategies

Causal Inference & Experimentation (A/B, CUPED, uplift)

Causal DS asks "what caused what?" not just "what correlates." It powers confident decision-making and trustworthy KPI movement.

What you do (in depth): Design experiments, compute power/MDE, run clean randomization, analyze effects with variance reduction (CUPED), and where RCTs aren't possible, use observational methods with sensitivity checks.

- Experiment design
    - Unit of randomization (user, session, geo), power analysis, sample ratio mismatch checks
    - Metric design: north-star vs guardrails, pre-registration to avoid p-hacking
- Analysis & variance reduction
    - CUPED (pre-period covariate regression), stratification, difference-in-differences
    - Sequential testing with alpha-spending; non-parametric CI when needed

- Uplift modeling (heterogeneous treatment effects)
    - Methods: two-model, meta-learners (T-/S-/X-/DR-learner), causal forests
    - Evaluation: Qini/uplift curves, policy value estimation
- Observational causal inference
    - Propensity scores, matching/weighting, IVs, front-door/back-door criteria
    - Sensitivity analyses (partial R^2, Rosenbaum bounds)
- Anti-patterns
    - Peeking, metric shopping, spillover/interference, noncompliance unaddressed

Forecasting & Time-Series (classical vs deep)

Forecasting predicts future demand/signals under seasonality, promotions, and shocks. Choose the simplest model that meets accuracy and operational needs.

What you do (in depth): Build robust pipelines with correct cross-validation, add exogenous regressors, and translate forecasts into inventory/staffing/pricing decisions.

- Data characteristics
    - Granularity (SKU-store-day), intermittency, hierarchy (global → regional → store), cold-start
    - Feature engineering: holiday calendars, lags/rolling stats, price/promos, weather
- Modeling approaches
    - Classical: ARIMA/SARIMA, ETS, TBATS, Prophet; hierarchical reconciliation
    - Machine learning: gradient boosting with lag features, quantile regression
    - Deep: LSTM/TCN/Temporal Fusion Transformer/Informer for long context
- Evaluation & validation
    - Rolling-origin CV; metrics: MAPE/sMAPE/WAPE, RMSE; quantile loss & coverage for PIs
    - Business metrics: stock-outs, overstock cost, service levels (fill rate)
- Production concerns
    - Retrain cadence, anomaly handling, blackout periods, backfill logic
    - Scenario planning (best/base/worst), forecast explainability for planners
- Anti-patterns
    - Random CV splits, metric misuse on intermittent series, ignoring hierarchy leakage

GenAI / LLM Applications (prompting, fine-tuning, safety)

GenAI apps synthesize, plan, and converse. Success depends on grounding, safety, and cost/latency control—not just model choice.

What you do (in depth): Design tasks, choose between prompting, tool-use, RAG, or fine-tuning, define human+automatic evals, and implement safety/abuse controls.

- Build choices
    - Prompt engineering & tool-use (function calling, retrieval)
    - Lightweight adaptation (LoRA, adapters) vs full fine-tune; when to choose which
    - RAG stacks (chunk, embed, retrieve, rerank, generate) with caching
- Evaluation
    - Automated: exact match, BLEU/ROUGE for structured tasks; rubric-based LLM judges
    - Human: helpfulness, faithfulness, harmlessness; rubric sampling per release
- Safety & compliance
    - Red-teaming, jailbreak resistance, toxicity/PII filters, content policies

    - Data governance: consented corpora, opt-out, copyright handling
- Ops & economics
    - Latency budgets, context window management, response caching, cost per 1k tokens
    - Observability: prompt/version registry, drift and hallucination rates, incident playbooks
- Anti-patterns
    - No ground truth eval, over-long contexts, letting hallucinations reach users unflagged

Applied Research vs Product DS (horizons, deliverables)

Two complementary tracks: **Applied Research** pushes methods and prototypes; **Product DS** ships impact to customers and metrics.

What you do (in depth): Choose the track (or blend) that matches the problem horizon, then manage hand-offs so research becomes real product value.

- Scope & time horizon
    - Applied Research: 6–24-month horizon, novel methods, internal/external publications
    - Product DS: 2–12-week iterations, shipping features/models to users
- Deliverables
    - Research: tech notes, benchmarks, reference implementations, datasets/simulators
    - Product: KPI movement, A/B reads, dashboards, model services with SLAs
- Collaboration patterns
    - Research ↔ Product: gated tech transfer, pilot experiments, de-risking studies
    - With MLE/Platform: productionization plans, latency/cost modeling, monitoring specs
- Success metrics
    - Research: SOTA/near-SOTA on internal benchmarks, adoption by product teams
    - Product: sustained metric lift, reliability, maintainability, stakeholder adoption
- Anti-patterns
    - Research with no path to integration; product work ignoring methodological risks

3) Core Competencies — Data Scientist

Math & Stats (probability, inference, causality basics)

A data scientist's rigor comes from statistical thinking: modeling uncertainty, identifying signal vs noise, and making defensible claims about cause and effect. You should be able to choose appropriate assumptions, quantify uncertainty, and explain trade-offs (bias/variance, power, sample size).

- Probability & distributions: conditional probability, Bayes rule; common families (Bernoulli/Binomial, Poisson, Gaussian, Log-normal); when heavy tails matter.
- Estimation: MLE/MAP, bias–variance trade-off, bootstrapping/jackknife for CIs, calibration of predicted probabilities.
- Hypothesis testing: null/alt, p-values, power/MDE, multiple testing control (BH/FDR), non-parametric tests (MWU/KS).
- Uncertainty: confidence vs prediction intervals; propagation of error; robust stats (medians, Huber loss).
- Causality basics: potential outcomes, ATE/ATE by cohort (HTE), confounding, DAGs/back-door, randomization as gold standard.
- Observational methods: matching/weighting, propensity scores, DiD, IVs; sensitivity analysis to hidden bias.

- Common pitfalls: p-hacking/metric shopping, Simpson's paradox, leakage, non-IID data (clusters, time dependence).

ML Algorithms (trees/boosting, linear models, clustering, embeddings)

Know when a simple model wins (interpretability, speed) and when complex ones pay off. Be fluent in failure modes, key hyperparameters, and evaluation suited to the problem.

- Linear/Generalized Linear Models
  - When: fast baselines, linear/semi-linear relations, explainability.
  - Tools: regularization (L1/L2/elastic net), link functions (logit/poisson).
  - Watch-outs: multicollinearity, feature scaling, outliers, nonlinearity.
- Trees & Gradient Boosting (RF, XGBoost/LightGBM, CatBoost)
  - Strengths: nonlinearities, interactions, strong tabular performance.
  - Keys: depth, learning_rate, n_estimators, min_child_weight; early stopping; class weights.
  - Watch-outs: overfitting without early stopping; leakage in target encoding; imbalanced data handling.
- Clustering & Unsupervised
  - K-means/K-medoids, DBSCAN/HDBSCAN, Gaussian Mixtures; dimensionality reduction (PCA, UMAP).
  - Evaluate: silhouette, Davies–Bouldin, stability; always validate clusters against business utility.
  - Watch-outs: scale sensitivity, arbitrary K, spurious clusters from noise.
- Embeddings
  - Tabular (entity embeddings), text (word/transformer embeddings), images (CNN/ViT features).
  - Uses: retrieval, similarity, cold-start, feature learning; metric learning/triplet losses.
  - Watch-outs: drift, alignment with downstream objective, privacy concerns with vector search.

Programming (Python/R), packaging, testing

Production-oriented coding turns notebooks into reliable assets. Aim for readable, testable, reproducible code with clear environments and CI.

- Environments & packaging: pyproject.toml/Poetry or pip + requirements; pinned versions; reproducible seeds.
- Structure: notebook → module pattern; CLI scripts; config via .env/YAML; logging (logging), typing (typing, pydantic).
- Testing: unit tests for feature functions and metrics (pytest); data contract tests (Great Expectations); property-based tests (hypothesis).
- Reproducibility: seeds, data snapshots, deterministic ops; model/feature registries (MLflow).
- Performance: vectorization (NumPy/Pandas), profiling (cProfile, line_profiler), parallelism (joblib), JIT (Numba) when needed.
- Quality: lint/format (ruff/black), pre-commit hooks, small pure functions, docstrings.
- Anti-patterns: hidden state in notebooks, mixing EDA & training code paths, unpinned deps.

SQL & Data Wrangling (joins, window funcs, performance)

Strong SQL prevents subtle errors and slow dashboards. You should write correct, explainable queries that scale and respect data contracts.

- Joins: inner/left/right/full; null semantics; de-duplication with surrogate keys; many-to-many explode hazards.

- Window functions: ROW_NUMBER, LAG/LEAD, cumulative sums, sessionization; partitioning by entity and ordering by time.
- Time & grains: defining canonical event time, late-arriving data, slowly changing dimensions (SCD), timezone consistency.
- Performance: predicate pushdown, clustering/partitioning, avoiding SELECT *, approximate functions for big scans.
- Reliability: CTEs for readability; idempotent incremental loads (MERGE/UPSERT); data quality checks (row counts, uniqueness, referential integrity).
- Wrangling: tidy data shapes, wide↔long pivots, categorical encoding, missing-data strategies.
- Anti-patterns: windowing without partitions, cartesian joins, computing KPIs at mixed grains.

Visualization & Storytelling (clear narratives, visuals)

Visuals are for decisions, not decoration. Great storytelling frames the question, shows comparisons, and annotates the "so what".

- Narrative arc: question → method (brief) → result → implication/recommendation → risk/next step.
- Chart choices:
    - Distribution (hist/box/violin), comparison (bar/line), relationship (scatter), part-of-whole (stacked bars), uncertainty (error bands/intervals).
- Principles: preattentive cues (position first), declutter (zero baselines where needed), consistent scales/colors, highlight deltas.
- Uncertainty: CIs, prediction intervals, fan charts, sample sizes; avoid hiding variance.
- Dashboards vs memos: operational monitoring (freshness & guardrails) vs one-time decision docs (annotated insights).
- Accessibility: colorblind-safe palettes, readable labels, mobile-first layouts when needed.
- Anti-patterns: dual y-axes misuse, 3D charts, truncated axes to exaggerate effects, KPI soup without context.

Business Problem Framing (hypotheses, constraints, ROI)

Technical excellence only matters if it changes outcomes. Frame problems as decisions under constraints with explicit payoffs and risks.

- Define the decision: action set, frequency, who acts; map to a north-star metric and guardrails.
- Hypotheses: causal story of how your lever moves the metric; measurable assumptions.
- Constraints: latency, budget, data availability/quality, regulatory/privacy limits, fairness thresholds.
- ROI model: back-of-envelope value, cost to build/maintain, sensitivity to adoption and error rates.
- Data readiness: sources, coverage, leakage risks, label latency, feedback loops.
- Evaluation plan: offline metrics ↔ business KPI, decision thresholds (ROC→profit curve), A/B or quasi-experimental design.
- Risk & fallback: failure modes, rollback, shadow launches, human-in-the-loop.
- Anti-patterns: starting from an algorithm, optimizing proxy metrics, ignoring operational adoption.

Communication & Stakeholder Management

Influence comes from clarity, cadence, and trust. Tailor depth to the audience and keep a shared source of truth.

- Audience mapping: execs (outcomes/risks), PMs (trade-offs/roadmap), Eng (interfaces/SLAs), Ops (playbooks), Legal (compliance).
- Artifacts: 1-pager PRD/DRD, experiment pre-reg, weekly updates, launch checklist, post-mortems; living metric definitions.

- Cadence: kickoffs, design reviews, decision meetings with options, read-outs with “recommend / next steps / owner”.
- Expectation setting: scope, assumptions, MDE/power limits, timelines; say “no” with alternatives.
- Collaboration: write clear specs (schemas, APIs), agree on data contracts, co-own monitoring.
- Conflict handling: surface trade-offs, show scenario tables, log decisions; escalate with crisp evidence.
- Anti-patterns: over-promising, burying the lede, jargon dumps, undocumented decisions.

4) Tools & Tech Stack — Data Scientist

Data Platforms (Snowflake / BigQuery / Redshift; Lakehouse)

Modern analytics lives on cloud warehouses and lakehouses. Warehouses (Snowflake, BigQuery, Redshift) prioritize ANSI SQL, elastic compute, and governance; lakehouses (Databricks + Delta, Apache Iceberg/Hudi on S3/GCS/ADLS) unify data lakes with ACID tables and open formats for ML at scale.

- What they’re best at
  - **Warehouses:** fast SQL, concurrency, fine-grained RBAC, simple ops, reliable BI.
  - **Lakehouse:** cheap storage, open formats (Parquet + Delta/Iceberg), ML-friendly files, streaming + batch unification.
- Selection criteria
  - Data size & variety (semi-structured JSON/Avro), **latency** needs, **cost model** (on-demand vs slots/warehouses), **governance** (RBAC, data masking), **interoperability** with ML tools.
- Performance & cost hygiene
  - Column pruning, clustering/partitioning, materialized views, result caching, **avoid SELECT ***, compressed columnar formats, storage lifecycle policies.
- Reliability & quality
  - **Data contracts**, schema evolution, SCD handling, time-travel/versioned tables, CDC ingestion (Debezium/Fivetran).
- Pitfalls
  - Mixed grains in KPIs, unbounded costs from ad-hoc scans, unmanaged external tables without ACID.

Orchestration & Transformation (Airflow, dbt)

Orchestration schedules reliable pipelines; transformation frameworks make SQL changes testable and modular.

- Roles
  - **Airflow (or Prefect/Dagster):** DAG orchestration, retries, SLAs, backfills, dependencies.
  - **dbt:** SQL-first transformations with **tests**, **documentation**, **macros**, and **environments**.
- Best practices
  - Separate **extract/load** from **transform**; small idempotent tasks; explicit data dependencies; parameterize by date.
  - In dbt: source freshness tests, unique/not-null tests, staging → marts layers, CI on pull requests, docs site.
- Ops & observability
  - Task-level retries, alerting on SLAs/failures, lineage graphs, data quality gates pre-publish.
- Pitfalls
  - Spaghetti DAGs, hidden side effects, long monolithic SQL models, lack of backfills/version pins.

Notebooks / IDEs (Jupyter, VS Code)

Notebooks excel at EDA and narration; IDEs shine for production code, tests, and refactors. Use both deliberately.

- Working pattern
  - Start in **Jupyter/JupyterLab** for exploration; migrate stable code into **VS Code** modules with tests.
- Reproducibility
  - Pin environments, **nbstripout** outputs, parameterize with **papermill/jupytext**, set random seeds, cache dataset snapshots.
- Productivity
  - VS Code: remote dev containers, debugger, notebooks editor, type checking, test runner; Jupyter: rich widgets/plots.
- Pitfalls
  - Hidden state, out-of-order execution, large data in cells, notebooks as the only artifact.

ML Libraries (scikit-learn, XGBoost/LightGBM, PyTorch/TensorFlow)

Pick the simplest library that meets accuracy/latency needs; prefer tabular SOTA (boosting) before deep nets unless representation learning is required.

- Tabular & classical
  - **scikit-learn:** pipelines, metrics, model selection; baselines (logreg, RF).
  - **XGBoost/LightGBM/CatBoost:** strong tabular performance; handle nonlinearity & interactions.
- Deep learning
  - **PyTorch/TensorFlow/Keras:** images/text/sequence models, custom architectures, GPU acceleration.
  - Ecosystem: Hugging Face (transformers/datasets/eval), timm, torchvision.
- Training & tuning
  - Cross-validation, early stopping, class weights, Optuna/Bayes optimization, calibration (Platt/isotonic).
- Serving
  - Batch scoring vs real-time (ONNX/TensorRT), quantization/pruning, AOT compilers (TorchScript, XLA).
- Pitfalls
  - Leakage in feature engineering, chasing deep models on small tabular data, ignoring calibration/uncertainty.

BI & Experiment Tools (Tableau / Power BI; experiment platforms)

BI surfaces "what is happening"; experiment platforms validate "what works." Both need trustworthy metrics and governance.

- BI stacks
  - **Tableau/Power BI/Looker/Superset** for dashboards, semantic layers, row-level security, scheduled extracts.
  - Practices: certified dashboards, single source of truth for metric definitions, usage monitoring.
- Experimentation
  - **Optimizely/Statsig/LaunchDarkly/GrowthBook** or in-house: randomization, CUPED, sequential tests, guardrails.
  - Checks: power/MDE, sample ratio mismatch (SRM), interference/spillover, pre-registration of hypotheses.
- Communication

    - Scorecards: treatment effects, CIs, cohort cuts, decision recommendations, rollback criteria.
- Pitfalls
    - Metric shopping, unmanaged KPI proliferation, running too many concurrent tests on same population.

Cloud & Infra (AWS / GCP / Azure; containers)

Cloud gives elastic compute/storage; containers/Kubernetes provide portability and repeatable deployment.

- Core building blocks
    - **Compute:** EC2/GCE/VMSS, serverless (Lambda/Cloud Functions), batch; **GPU** (A100/H100) for DL.
    - **Storage:** S3/GCS/ADLS (data lake), EBS/PD (block), EFS/Filestore (shared).
    - **Networking & security:** VPCs, private subnets, security groups, IAM least privilege, KMS/CMK.
    - **Containers:** Docker + **Kubernetes** (EKS/GKE/AKS) for scaling model APIs and jobs.
- Ops & cost
    - Autoscaling, spot/preemptible for training, node pools by workload, cost tags/budgets, artifact registries.
- MLOps services
    - Managed notebooks, model endpoints (SageMaker/Vertex/AML), pipelines, feature stores, monitoring.
- Pitfalls
    - Secrets in code, no egress controls, over-provisioned GPUs, missing resource limits/requests on K8s.

Tracking & Registry (MLflow; model/feature stores)

Without tracking, you can't reproduce or govern models. Use experiment tracking + registries + feature stores for consistency.

- Experiment tracking
    - **MLflow/W&B/Vertex Experiments:** log params, metrics, artifacts, datasets, code versions; compare runs.
- Model registry
    - Lifecycle states (Staging/Production/Archived), approval workflows, model versioning, deployable flavors (pyfunc, ONNX).
- Feature stores
    - **Feast/Tecton/SageMaker/Vertex:** define features once, serve online/offline consistently, backfills, point-in-time correctness.
- Monitoring & lineage
    - Data drift, concept drift, label latency, feature freshness; lineage from raw sources → features → models → endpoints.
- Pitfalls
    - Training/serving skew, ad-hoc features in notebooks, no audit trail for promoted models.

Version Control & Environments (Git, venv/conda)

Versioning code, data, and environments makes work reviewable and recoverable.

- Git workflow
    - Branching strategy (feature → PR → main), small PRs, **code owners**, protected branches, semantic commits and tags.
    - CI on PRs: tests, lint, **dbt build/test**, data contract checks; release notes and changelogs.

- Environments
    - **venv/conda/Poetry** with pinned versions and lockfiles; separate **runtime** (serving) from **training** envs.
    - Reproducible containers (Dockerfiles) with minimal bases; pre-commit hooks (black/ruff/isort).
- Data & artifact versioning
    - DVC/lakehouse time-travel for datasets; artifact registries for models.
- Pitfalls
    - "Works on my machine," unpinned dependencies, committing notebooks with large outputs, no lock to model code version.

5) Responsibilities & Deliverables — Data Scientist

EDA & Data Quality reports

A good EDA clarifies *what the data can and cannot support* before modeling. It surfaces leakage, label latency, missingness patterns, and cohort biases; it also states whether the dataset is decision-grade.

- What to include
    - **Context:** purpose, target decision/KPI, unit of analysis, time window.
    - **Schema profile:** fields, types, cardinality, allowed ranges, PII flags.
    - **Target audit:** base rates, drift over time, label latency, leakage risks.
    - **Missingness & anomalies:** MCAR/MAR/MNAR hypotheses, outliers, duplicates.
    - **Cohorts & bias:** performance-critical slices (geo, product line, device), prevalence by slice.
    - **Data lineage & freshness:** source systems, ingestion lag, backfills, known quirks.
    - **Decision note:** "Fit for purpose" verdict + gaps and mitigation plan.
- Acceptance criteria
    - Reproducible notebook/script; summary one-pager with charts and a **go/no-go** statement.
- Common pitfalls
    - Mixed grains, silent de-duplication, using post-outcome fields (leakage), ignoring seasonality.

Feature engineering & dataset contracts

Feature work turns raw tables into stable, error-tolerant signals. A **dataset/feature contract** encodes definitions so upstream changes don't silently break models.

- What to include
    - **Feature spec:** name, business meaning, SQL/py pseudocode, allowed nulls, range, timezone, unit.
    - **Point-in-time correctness:** time joins, label windows, leakage checks.
    - **Training/serving parity:** same logic (dbt/Feast/feature store), backfill strategy.
    - **Quality tests:** uniqueness, referential integrity, freshness SLAs, anomaly thresholds.
    - **Versioning:** semantic version + change log, deprecation policy.
- Deliverables
    - dbt models (or feature store defs), validation tests, and a rendered contract page.
- Common pitfalls
    - Target/mean encoding without leakage guards; recomputing features differently online vs offline.

Modeling & Evaluation write-ups

This document explains *why this model is the right choice*, how it was validated, and when it should be retrained or rolled back.

- What to include
  - **Problem framing:** prediction target, horizon, actionability, constraints (latency, fairness).
  - **Baselines & contenders:** simple baseline → advanced models; ablations; cost/latency table.
  - **Validation design:** temporal CV/rolling origin; stratification; data leakage controls.
  - **Metrics:** offline (AUC/F1/PR-AUC/RMSE), calibration (ECE/Brier), cohort breakdowns.
  - **Interpretability:** SHAP/global vs local, limitations, stability checks.
  - **Risk register:** failure modes, guardrails, rollback triggers; monitoring plan.
  - **Lifecycle:** retrain cadence, data drift thresholds, owner/rotation.
- Deliverables
  - Readable memo + MLflow run links + model card.
- Common pitfalls
  - Reporting a single metric; tuning on test; ignoring calibration and cost-sensitive thresholds.

Experiment design, analysis, and readouts

Experiments convert model quality into causal business impact. The artifacts prevent p-hacking and make decisions auditable.

- Pre-registration (before launch)
  - **Hypothesis & success criteria**; north-star & guardrails; **MDE/power**; randomization unit; segmentation plan; SRM checks; duration cap; stopping rules.
- Analysis plan
  - CUPED/stratification, non-parametric backups, treatment compliance handling, interference mitigation.
- Readout (after)
  - Effect sizes + CIs; cohort cuts; diagnostics (SRM, pre-trend); *recommendation with rollout plan/rollback*.
- Deliverables
  - One-pager scorecard + appendix notebook; link to flags/config used.
- Common pitfalls
  - Peeking, metric shopping, overlapping experiments, ignoring seasonality/holidays.

Dashboards & Metric definitions

Dashboards inform *operational decisions*; metric definitions ensure everyone calculates KPIs the same way.

- Metric catalog
  - **Name, formula, grain, filters**, inclusion/exclusion rules, time zone, SLO/SLA, owner.
  - Guardrails (e.g., complaint rate, latency) alongside north-star metrics.
- Dashboard design
  - Executive view (few KPIs + trends + annotations) → drill-downs by cohort; freshness badges and data quality warnings.
  - Comparisons (WoW/MoM), cause-and-effect panels for experiments/launches.
- Deliverables
  - Certified dashboard with usage monitoring + living metric doc.
- Common pitfalls
  - KPI soup, mixed grains, no annotations for breaks/backfills, stale tiles.

Decision/Strategy docs (trade-offs, risks)

A short, persuasive memo that ties analysis to an explicit decision, surfaces alternatives, and quantifies risk/ROI.

- Structure (4–6 pages or less)
    - **Context & goal**, options considered, evidence summary, cost/benefit table, risks & mitigations, **recommendation** and next steps/owners.
    - Scenario analysis (best/base/worst), fairness & compliance impact, kill/rollback criteria.
- Deliverables
    - Decision Review Doc (DRD) with appendix links (EDA, model, experiment).
- Common pitfalls
    - Presenting results without recommending an action; hiding uncertainty; no owner or timeline.

Prototypes/APIs & handoffs to engineering

Bridge from notebook to production. Provide a minimal, well-specified service or batch job plus clear ownership for ongoing support.

- What to include
    - **Interface spec:** request/response schema, units, error codes, timeouts, SLOs; idempotency for batch.
    - **Artifacts:** container/Dockerfile, sample requests, load test results, dependency lockfile.
    - **Operational docs:** monitoring signals, on-call runbook, rate limits, data retention, PII handling.
    - **Acceptance:** shadow test or limited rollout with agreement on success gates.
- Deliverables
    - OpenAPI spec + reference client + canary plan; or scheduled batch with checksum reports.
- Common pitfalls
    - Training/serving skew, no backpressure strategy, secrets in code, undefined retry semantics.

Reproducibility (code, seeds, data snapshots)

If you can't reproduce it, you can't trust it. Reproducibility enables audits, handovers, and safe iteration.

- Practices
    - **Code:** modular repo; pinned deps (requirements.txt/poetry.lock), CI tests, linting, type hints.
    - **Runs:** MLflow/W&B run IDs; log data checksums, feature versions, random seeds, commit SHA.
    - **Data:** time-stamped snapshots or lakehouse time-travel; point-in-time joins; sample artifacts for reviews.
    - **Envs:** Docker image tags; training vs serving env parity; deterministic ops where feasible.
    - **Docs:** README with exact commands; makefile/CLI; changelog for models/features.
- Deliverables
    - "Reproduce in 1 command" script; audit bundle (code, config, data slice, model).
- Common pitfalls
    - Hidden notebook state, unpinned versions, overwriting training data, missing seeds.

6) Hiring, Qualifications & Interviews — Data Scientist

Minimum vs preferred qualifications

A solid data-science hire blends statistical rigor, product sense, and production-minded coding. "Minimum" ensures they can contribute safely; "preferred" signals they can lead end-to-end impact.

What "minimum" usually means (must-have):

- **Foundations:** probability/inference basics; supervised learning (linear/logistic, trees/boosting), validation, and metrics.
- **Core tooling:** SQL (joins, window functions), Python (pandas/numpy/sklearn), notebooks → scripts.
- **Data literacy:** tidy data, leakage awareness, time-based splits, cohorting, missing-data strategies.
- **Communication:** can explain a model and its *business* purpose in clear, non-jargon language.
- **Evidence:** at least one reproducible project or internship showing EDA → model → evaluation.

What "preferred" looks like (nice-to-have / bar-raising):

- **Advanced methods:** causal inference/experimentation, recsys, NLP/CV, or forecasting depth; calibration and uncertainty.
- **MLOps awareness:** CI tests, experiment tracking, model registry, basic deployment patterns (batch/real-time).
- **Product thinking:** can translate metrics into ROI; proposes guardrails; designs A/Bs.
- **Scale & reliability:** dbt or similar, Airflow/Prefect, cloud (AWS/GCP/Azure), containers.
- **Leadership signals:** mentoring, scoping ambiguous problems, cross-functional decision memos.

'

Portfolio expectations (repos, case studies, notebooks)

Portfolios should show problems that matter, decisions driven, and code that others can run. Curate 3–4 flagship pieces rather than many half-finished ones.

What great portfolios include:

- **Readable README:** problem statement, data source, decision/KPI, how to run (make run or one command), and results.
- **Reproducibility:** requirements.txt / pyproject.toml, seed control, small sample dataset, MLflow/W&B links or logs.
- **End-to-end story:** EDA → features (with a brief *contract*) → model baselines → validation design → calibration → error analysis.
- **Decision framing:** a short "so what" section: suggested threshold/policy, projected impact, risks/rollbacks.
- **Live artifact:** Streamlit/Gradio demo or a REST stub + OpenAPI spec.

Case-study template (1–2 pages each):

- Context & hypothesis → Data & caveats → Methods & baselines → Results (with CIs) → Business recommendation → Limitations/next steps.

Common misses:

- Hidden notebook state; no baseline; single metric; no leakage checks; unreproducible env; no explanation of *business* value.

Interview stages (screen → SQL/Python → ML/case → system design → behavioral)

Expect a funnel testing signals from breadth to depth to collaboration. Each stage should be passable on its own.

1) Recruiter/phone screen (15–30 min):

- 2-minute project pitch, clarity of role fit, compensation/logistics, high-level signals of impact.

2) SQL/Python practical (45–60 min):

- **SQL:** multi-table joins, window functions, deduping, cohorting, sessionization, correctness at the right grain.

- **Python:** data wrangling, writing a small function/class, tests, time/space reasoning (not Big-O perfection).

3) ML/case interview (60 min):

- Frame a real product problem (e.g., churn prediction): define target/horizon, leakage risks, metrics, baselines, validation, calibration, and how to select a threshold or policy.
- Expect trade-offs (AUC vs PR-AUC, cost weighting), error analysis, and a quick *experiment plan.*

4) ML system design (45–60 min):

- Design training → registry → deployment (batch/real-time) → monitoring (data/Concept drift, latency, cost) → retrain cadence.
- Call out feature stores, training/serving skew, canary/rollback, privacy and access controls.

5) Behavioral (45 min):

- STAR format on conflict, influencing without authority, moving a metric, handling failures/ethics/privacy.
- Look for ownership, clarity, reflection, and stakeholder empathy.

Take-home vs live exercises (rubrics, timelines)

Both formats can be fair if scoped tightly and graded with a rubric.

Take-home (typ. 4–8 hrs, 3–7 days window):

- **Brief:** clearly stated goal, dataset, constraints, deliverables; require a *decision memo* + runnable code.
- **Rubric:** correctness, validation design, clarity of reasoning, reproducibility, business recommendation, and code hygiene.
- **Good signs:** measured scope (one model + baseline), explicit evaluation, simple demo.

Live exercises:

- **Pairing style:** work on SQL or a modeling vignette together; interviewer probes thought process.
- **Rubric:** communication under time pressure, ability to ask clarifying questions, incremental problem solving.

Anti-patterns (for both):

- Vague objectives, unrealistic data sizes/time limits, evaluating on style over substance, penalizing candidates for not using the interviewer's favorite library.

Evaluation criteria (correctness, clarity, impact thinking)

Define the bar *before* interviewing and stick to it. A sample scoring split (adjust as needed):

- **Technical correctness (30%)**
  - Sound stats/ML, proper validation (temporal CV where needed), leakage avoided, calibrated outputs when decisions require.
- **Problem framing & impact (25%)**
  - Clear target/KPI, thoughtful thresholds, experiment plan, cost/benefit reasoning, risks & guardrails.
- **Data & SQL (15%)**
  - Correct joins, windowing, grain control, performance awareness, tidy outputs.
- **Code quality & reproducibility (15%)**
  - Structure, tests, env pinning, docs, deterministic seeds, small clean PRs.
- **Communication & collaboration (15%)**
  - Clear narrative, stakeholder alignment, trade-off articulation, receptive to feedback.

Bonus signals:

- Sensitivity analyses, thoughtful error slicing, fairness considerations, measuring uncertainty.

9) Career Levels & Compensation — Data Scientist

Level expectations (Junior → Senior → Staff/Principal)

Progression reflects increasing **independence, scope, and risk ownership**. Titles vary, but the capability curve is consistent: from *executing clearly scoped tasks* to *setting technical direction and de-risking ambiguous problems for others*.

- **Junior / Associate (entry–~2 years)**
  - Executes well-defined tasks: clean datasets, build baselines, run evaluations.
  - Needs guidance to frame problems and avoid pitfalls (leakage, invalid CV).
  - Learning focus: strong SQL/EDA, validation patterns, writing crisp analysis notes.
- **Mid-Level (solid individual contributor)**
  - Owns a feature/model end-to-end under a PM/tech lead's umbrella.
  - Anticipates edge cases, writes reproducible code, proposes reasonable baselines & metrics.
  - Collaborates smoothly with Eng/PM; can run a small A/B.
- **Senior**
  - Frames problems with stakeholders; chooses approach, metrics, experiment design.
  - Multimodel comparisons, calibration, monitoring plans; mentors juniors.
  - Predictably ships business impact; drafts decision docs and rollout/rollback plans.
- **Staff** (org-level problem solver)
  - Tackles **high-ambiguity** problems across teams; creates standards (e.g., feature contracts, KPI catalogs).
  - De-risks complex launches; drives technical roadmaps; influences headcount and platform direction.
  - Mentors seniors; coordinates multi-team initiatives.
- **Principal / Distinguished**
  - Sets **long-horizon** strategy (6–24 months): methodology, platforms, cross-org investments.
  - Creates widely adopted patterns; represents the company externally; shapes hiring bar.
  - Solves "no known playbook" problems, often with applied research depth.

Scope & autonomy by level (project → product area → org)

As you rise, scope widens from a **project** to a **portfolio/product area** to **organizational** concerns; autonomy tracks your ability to manage risk and trade-offs.

- **Project scope (Junior/Mid):** a single model or dashboard with clear acceptance criteria; relies on lead for framing.
- **Product-area scope (Senior):** multiple related models/metrics; balances impact vs complexity; defines success metrics & guardrails.
- **Org scope (Staff+):** platforms, standards, or multi-team efforts; resolves cross-team dependencies; sets roadmaps.
- **Autonomy signals:** reduces manager load; proactively raises risks; quantifies trade-offs; aligns peers without escalation.

IC vs Manager tracks (people leadership vs deep IC)

Two *equally senior* ladders. ICs lead via **technical direction and execution**; managers lead through **people, outcomes, and process**.

- **IC (Senior/Staff/Principal)**
  - Deep technical ownership, design reviews, de-risking plans, cross-team technical influence.
  - Mentors and unblocks others; no formal performance management.

    - Ideal if you enjoy building, teaching through code/docs, and setting technical standards.
- **Manager (DS Manager → Sr. Manager → Director/Head)**
    - Owns hiring, growth, performance, prioritization, stakeholder alignment, and delivery.
    - Builds teams & processes (on-call, QA, experiment review), manages budgets.
    - Ideal if you like coaching, portfolio management, and cross-functional negotiation.
- **Switching considerations**
    - IC→Mgr: proven mentorship, reliable project delivery, appetite for hiring/perf.
    - Mgr→IC: desire for depth, complex design work, less interest in performance cycles.

Comp components (base, bonus, equity) & drivers (region, stage, sector)

Comp is a bundle: **base salary** (cash), **bonus** (performance), **equity** (ownership), sometimes **sign-on** and **benefits**. Market levels move with macro cycles.

- **Components**
    - **Base:** predictable monthly cash; banded by level/location.
    - **Bonus:** % of base; tied to company + individual performance.
    - **Equity:** options (strike price, 4-yr vest, 1-yr cliff) or RSUs (vest on schedule/liquidity).
    - **Extras:** sign-on, relocation, education, WFH stipend, on-call pay (less common for DS).
- **Key drivers**
    - **Region:** US/EU metros pay > emerging markets; fully-remote often aligns to geo bands.
    - **Company stage:** late-stage/FAANG-ish = higher cash + RSUs; early-stage = lower cash + higher options (higher variance).
    - **Sector:** finance/quant, ads, and AI platform infra pay premiums; non-profit/public sector lower but stable.
    - **Skill scarcity:** recsys, causal inference at scale, LLM safety/eval, or strong MLOps can command premiums.
- **Equity nuances**
    - Options need an exit to realize value; RSUs are more liquid at public companies.
    - Refreshers typically start at Senior+; ask about refresher cadence and promotion equity.

(Avoid quoting hard numbers in docs you'll reuse; bands fluctuate by quarter and geography.)