Definition & scope (what DS is / isn't)

Data Science (DS) is the disciplined process of turning raw data into useful knowledge, predictions, and decisions. It blends **statistics**, **computing**, and **domain expertise** to discover patterns, build models, and deliver measurable impact (dashboards, forecasts, automated decisions).

- **What DS is**
  - A problem-solving practice grounded in data, experimentation, and evidence.
  - End-to-end: from collecting/cleaning data → analysis → modeling → communicating results → deploying and monitoring solutions.
  - Output can be **insights** (reports, dashboards) or **systems** (recommendation APIs, fraud detectors).
- **What DS isn't**
  - Not just coding or math in isolation; context and decision impact matter.
  - Not "AI for AI's sake"—the goal is **business/user value**, not fancy algorithms.
  - Not a one-off report; good DS plans for **maintenance and monitoring**.

Short history & key milestones (stats → ML → big data → deep learning → GenAI)

Data Science evolved by layering new computation and data availability on top of classic statistics.

- **Pre-1990s:** Statistical inference (sampling, regression) in academia & industry.
- **1990s–2000s:** Machine Learning (SVMs, ensembles) + data warehousing; web logs at scale.
- **2010s:** "Big Data" (Hadoop/Spark), cloud computing; deep learning breakthroughs (CNNs for vision, RNN/LSTMs for sequence).
- **Late 2010s–early 2020s:** MLOps maturation; model monitoring, data versioning, model registries.
- **2023+ (GenAI era):** Foundation models/transformers (LLMs, multimodal) enable powerful **generation** and **reasoning**; retrieval-augmented systems; governance/regulation ramp up.

Why it matters: each wave lowered friction—cheaper storage/compute, better algorithms—expanding what's feasible for organizations.

Core pillars (data, math/stats, computing, domain)

Great DS sits on four mutually reinforcing pillars.

- **Data**
  - Understanding schemas, quality issues, sampling, bias.
  - Judging if data can actually answer the question.
- **Math/Statistics**
  - Probability, estimation, hypothesis testing, causal thinking.
  - Bias–variance trade-offs; uncertainty quantification.
- **Computing**
  - Programming (Python/SQL), data pipelines, version control, cloud.
  - Performance and reliability (batch vs real-time, memory/latency).
- **Domain Expertise**
  - Vocabulary, constraints, and success metrics of the field (finance, health, ops).
  - Knowing which mistakes are costly and which signals matter.

Rule of thumb: weak in any pillar → brittle outcomes (e.g., great model, wrong problem).

Data types & sources (structured, semi-structured, unstructured; internal vs external)

Data comes in different shapes and from many places; that shape influences tools and methods.

- **Structured:** tables with fixed columns (transactions, customers).
  Fits SQL well; easy for classical ML.
- **Semi-structured:** JSON, logs, event streams.
  Needs parsing/flattening; great for behavioral analytics.
- **Unstructured:** text, images, audio, video.
  Often needs embeddings, deep learning, vector search.

Sources

- **Internal:** product databases, CRM/ERP, sensors/IoT, support tickets, app telemetry.
- **External:** public datasets, vendors, web/APIs, social media, satellite feeds.
- **Acquisition modes:** batch exports, CDC/streams, webhooks, surveys, scraping (with ethics/legal checks).

Quality dimensions to note: completeness, accuracy, timeliness/freshness, consistency, and lineage (where it came from).

Typical lifecycle (problem → data → EDA → modeling → evaluation → deployment → monitoring)

A standard lifecycle reduces risk and creates repeatability.

1. **Problem framing**
   - Define users, decisions, and success metrics (business + technical).
   - Clarify constraints (latency, privacy, budget).
2. **Data access & preparation**
   - Collect, join, and clean data; document assumptions.
   - Handle missing values/outliers; guard against leakage.
3. **EDA (Exploratory Data Analysis)**
   - Summaries, distributions, correlations; spot drift or bias.
   - Form hypotheses; refine features and targets.
4. **Modeling**
   - Baselines first; iterate with suitable algorithms.
   - Cross-validation; hyperparameter tuning; calibration if probabilities matter.
5. **Evaluation**
   - Use task-appropriate metrics (e.g., F1 vs RMSE) and subgroup checks.
   - Stress tests (temporal splits, robustness).
6. **Deployment**
   - Package as batch job, API, or stream processor; document interfaces.
   - Add observability (logging, tracing) and fallback behavior.
7. **Monitoring & improvement**
   - Track data quality, drift, performance, cost, and usage.
   - Retraining/rollback playbooks; periodic reviews with stakeholders.

Deliverables to expect: data spec → EDA brief → experiment log → model card → runbook.

Where it's used (everyday examples across industries)

DS shows up anywhere decisions repeat and data exists.

- **Finance:** credit scoring, fraud detection, customer churn, ATM cash forecasting.
- **Retail/e-commerce:** recommendations, dynamic pricing, demand forecasting, A/B testing.
- **Healthcare:** readmission risk, triage prioritization, medical imaging support, capacity planning.
- **Manufacturing/IoT:** predictive maintenance, quality inspection, supply-chain optimization.

- **Logistics:** route optimization, ETA prediction, inventory positioning.
- **Public sector:** tax anomaly detection, traffic management, disaster response analytics.
- **HR/People analytics:** attrition risk, hiring funnel diagnostics, workforce planning.
- **Customer support:** intent classification, auto-reply, ticket routing, satisfaction prediction.
- **Marketing:** segmentation, uplift modeling, campaign optimization, LTV prediction.

Pattern: **predict + prescribe** → make the next action cheaper, faster, or more accurate.

Limits & challenges (data quality, bias, drift, privacy, governance)

Real-world DS must navigate technical, ethical, and operational hurdles.

- **Data/technical**
  - Missing, noisy, or biased data; label quality issues.
  - Non-stationarity & **drift** (data or concept changes over time).
  - Leakage (using future info), small samples, class imbalance.
- **Model/metrics**
  - Overfitting; poorly calibrated probabilities.
  - Metric mismatch (optimizing AUC when cost is asymmetric).
- **Ethics/privacy**
  - PII/PHI handling, consent, retention limits.
  - Fairness: subgroup performance gaps; disparate impact.
  - Transparency & contestability (explainability, human-in-the-loop).
- **Operations/governance**
  - Reproducibility, versioning, and approvals (model risk management).
  - Monitoring blind spots; ownership after go-live.
  - Cost control (compute/storage/egress) and vendor lock-in.

Mitigations

- Data contracts/SLAs, quality checks, and lineage.
- Robust validation schemes, model cards, and review boards.
- Privacy-by-design (minimize data, access controls, audits).
- Clear runbooks for retraining, rollback, and incident response.

Common project archetypes

Most real projects fit one (or a blend) of these patterns. Knowing the archetype clarifies data needs, metrics, and delivery shape.

- **Prediction (regression):** predict a continuous value (price, demand, time-to-fail).
  Data: tabular, time series. *Metrics*: MAE/RMSE/MAPE. *Actions*: price updates, inventory levels.
- **Classification:** assign labels (fraud/not, churn/not).
  Data: tabular, logs, events. *Metrics*: Precision/Recall/F1/AUC; cost-sensitive metrics.
- **Recommendation/Ranking:** order items/users/ads.
  Data: interactions, catalogs, embeddings. *Metrics*: CTR, MAP/NDCG, coverage/diversity, revenue per session.
- **NLP:** classify/extract/summarize/search text.
  Data: tickets, emails, chats, docs. *Metrics*: F1 (NER/cls), ROUGE/BERTScore (gen), MRR/NDCG (search).
- **Vision:** detect/classify/segment images or video.
  Data: images, frames, annotations. *Metrics*: mAP, IoU, recall at fixed FP rate.
- **Anomaly/Quality:** find rare/novel events (fraud, defects).
  Data: sensor logs, transactions. *Metrics*: precision at k, alert yield, time-to-detect.

Domain snapshots

Each domain has “typical” problems, constraints, and KPIs.

- **Finance**
  - *Use cases*: credit scoring, fraud, AML, LTV prediction.
  - *Constraints*: explainability, low latency, strict audit.
  - *KPIs*: default rate, fraud loss prevented, approval lift at constant risk.
- **Healthcare**
  - *Use cases*: readmission risk, imaging assist, triage, capacity.
  - *Constraints*: PHI privacy, safety, clinical validation.
  - *KPIs*: sensitivity at fixed specificity, LOS reduction, throughput.
- **Retail/Marketing**
  - *Use cases*: personalization, demand forecasting, promo uplift.
  - *Constraints*: seasonality, catalog churn.
  - *KPIs*: revenue/visit, conversion, inventory turns, margin.
- **Manufacturing**
  - *Use cases*: predictive maintenance, visual QA, yield optimization.
  - *Constraints*: edge inference, downtime cost.
  - *KPIs*: MTBF↑, scrap↓, OEE↑, false-stop rate↓.
- **Public sector**
  - *Use cases*: risk scoring, resource allocation, mobility.
  - *Constraints*: transparency, fairness/public trust.
  - *KPIs*: service time↓, coverage↑, equitable outcomes.
- **Tech/Startups**
  - *Use cases*: search/ranking, abuse detection, growth analytics.
  - *Constraints*: rapid iteration, scale, cost.
  - *KPIs*: retention, DAU/MAU, abuse rate↓, infra cost per user.

Data characteristics (volume, velocity, variety; batch vs streaming)

Pick processing patterns to match the “shape” of data.

- **Volume:** GB → TB → PB impacts storage (Parquet/Delta), compute (Spark), and sampling strategy.
- **Velocity:** daily batches vs near-real-time streams (Kafka/Kinesis, CDC).
- **Variety:** tabular + logs + text/images ⇒ need schemas, contracts, and feature stores.
- **Processing modes**
  - *Batch*: cheaper, great for nightly refresh (pricing, forecasts).
  - *Streaming/real-time*: low-latency actions (fraud, recommendations).
- **Quality/lineage:** contracts, validation (Great Expectations), lineage tracking, drift monitors.

Programming & SQL (Python basics, pandas/NumPy, SQL joins/window funcs)

You’ll use Python to manipulate data and build models, and SQL to fetch/prepare the data from databases. Mastering both lets you move from raw tables to clean feature sets fast and reproducibly.

- **Learn**
  - Python: variables, control flow, functions, modules, virtual envs; file I/O; error handling.
  - NumPy: arrays, broadcasting, vectorization, basic linear algebra.
  - pandas: Series/DataFrame, indexing, groupby/agg, merge/join, tidy vs wide, time-series basics.
  - SQL: SELECT/WHERE, JOIN types (inner/left/right/full), GROUP BY/HAVING, subqueries, CTEs.

    - **Window functions**: ROW_NUMBER, RANK, LAG/LEAD, moving averages.
- **Practice**
    - Recreate Excel tasks in SQL (joins instead of VLOOKUP; window funcs for running totals).
    - Build an end-to-end Python script: read CSV → clean → join → save Parquet.
- **Pitfalls**
    - Chained pandas operations without .copy() → SettingWithCopy bugs.
    - Cartesian products from careless joins; forgetting primary keys.
- **Artifacts**
    - Reusable data-cleaning script; SQL snippets library; requirements.txt/conda env.

Statistics & Probability (distributions, sampling, testing, intervals)

Stats helps you quantify uncertainty and avoid wrong conclusions. You'll use it to compare groups, design experiments, and reason about noisy data.

- **Learn**
    - Random variables & common distributions (Bernoulli, Binomial, Normal, Poisson).
    - Law of large numbers & Central Limit Theorem (why means stabilize).
    - Estimation: confidence intervals (what 95% really means).
    - Hypothesis testing: null/alternative, p-values, type I/II errors, power & effect size.
    - Sampling: simple vs stratified; bias and variance; bootstrap.
- **Practice**
    - Compute a 95% CI for a mean and a proportion; explain it in plain language.
    - A/B test readout: difference in conversion, CI, p-value, and business impact.
- **Pitfalls**
    - "$p < 0.05 \Rightarrow$ truth" fallacy; multiple comparisons without correction.
    - Correlation $\neq$ causation; Simpson's paradox.
- **Artifacts**
    - Stats cheat sheet (tests by scenario); experiment readout template.

Machine Learning (supervised/unsupervised, model selection, tuning)

ML turns patterns into predictions and decisions. Start simple, then iterate with validation and tuning to avoid overfitting.

- **Learn**
    - Task types: regression vs classification; clustering & anomaly detection.
    - Pipelines: split → baseline → features → model → tune → evaluate.
    - Algorithms: Linear/Logistic, Trees/Random Forest, Gradient Boosting (XGBoost/LightGBM), k-means, PCA.
    - **Model selection**: cross-validation; compare on the right metrics.
    - **Tuning**: grid/random search; early stopping; regularization; class weights.
- **Practice**
    - Build a baseline (mean/majority) then beat it with 2 models; log results.
    - Tune max_depth/learning_rate on a GBM; plot validation curves.
- **Pitfalls**
    - Data leakage (using future/target info in features); improper CV for time series.
    - Chasing tiny metric gains with very complex models.
- **Artifacts**
    - Experiment log; model card (what/why/how/limits); saved pipeline.

EDA & Visualization (descriptives, correlations, dashboards)

EDA is where you learn the dataset's shape, spot issues, and form hypotheses. Good visuals make insights obvious to others.

- **Learn**
  - Univariate: histograms/boxplots; central tendency & spread; skew/kurtosis.
  - Bivariate: scatter/violin/heatmaps; correlation (Pearson/Spearman).
  - Target analysis: class balance; partial plots for key features.
  - Dashboard basics: filters, drilldowns, KPIs; tidy layout and labeling.
- **Practice**
  - Produce a one-page EDA summary: 5 key charts + 5 bullet insights.
  - Compare log vs raw scale for skewed variables; justify transformations.
- **Pitfalls**
  - Misleading axes/scales; ignoring missingness patterns; overtrusting correlations.
- **Artifacts**
  - EDA notebook + PNGs; lightweight narrative with "so-what" insights.

Feature Engineering (encoding, scaling, transformations, leakage avoidance)

Most performance comes from good features. Engineer them systematically while preventing leakage and overfitting.

- **Learn**
  - Missing/outliers: imputation strategies; winsorization vs clipping.
  - Encodings: one-hot vs ordinal; target/mean encoding (with CV).
  - Scaling: standard vs min-max; when models require it (SVM, KNN).
  - Transformations: log, binning, interactions, ratios; date/time features; rolling windows for TS.
  - **Leakage guards**: fit imputers/encoders inside CV folds or pipeline.
- **Practice**
  - Build a sklearn Pipeline/ColumnTransformer handling numeric/categorical features end-to-end.
  - Create rolling-mean features only from past windows on TS.
- **Pitfalls**
  - Fitting scalers on full data; using target statistics without proper CV.
- **Artifacts**
  - Reusable feature pipeline; data dictionary of engineered features.

Model Evaluation (classification/regression metrics, validation schemes)

Pick metrics that match decisions and costs. Validate the right way for your data's structure.

- **Learn**
  - Classification: accuracy, precision/recall/F1, ROC-AUC vs PR-AUC; confusion matrix.
  - Thresholding: cost-sensitive decisions; precision-at-k; lift/gain.
  - Regression: MAE vs RMSE vs R^2; MAPE/WMAPE; quantile loss if predicting ranges.
  - Calibration: reliability curves; Brier score (probability quality).
  - Validation: holdout, k-fold/stratified, **time-series CV** (rolling windows).
- **Practice**
  - Plot ROC and PR curves; pick a threshold based on a cost matrix.
  - Run rolling-origin evaluation on a TS model; report WAPE by horizon.
- **Pitfalls**
  - Reporting AUC when you deploy a fixed-capacity top-k list; random splits on temporal data.

- **Artifacts**
    - Evaluation report with metric table, curves, and threshold rationale.

MLOps Basics (packaging, APIs, monitoring, data/model versioning)

MLOps makes models reliable in production. Package, serve, and monitor with versioned data and models.

- **Learn**
    - Packaging: requirements.txt/conda; Docker basics; reproducible seeds.
    - Serving: batch jobs vs REST APIs (FastAPI) vs streaming; auth & rate limits.
    - Monitoring: data quality (missing/drift), performance, latency/cost; alerts & dashboards.
    - Versioning: MLflow/DVC (params, artifacts, models); registries; CI/CD.
    - Runbooks: rollback, retraining cadence, ownership.
- **Practice**
    - Serve a trained model via FastAPI; log requests/responses; save a model to a registry.
    - Add a simple drift check (population stability index or KS test) on input features.
- **Pitfalls**
    - "Works on my machine"; no rollback plan; silent model decay.
- **Artifacts**
    - Dockerfile + API spec; monitoring dashboard; runbook; model registry record.

Big Data & Distributed (Spark, data lakes/warehouses, Parquet)

Use distributed tools when data is too large/fast for a single machine. Prefer columnar storage and efficient queries.

- **Learn**
    - Storage: Parquet/Delta (columnar, compressed), partitioning, file sizing.
    - Compute: Spark DataFrame ops; joins, windowing, UDF pitfalls.
    - Platforms: data lake vs warehouse (ELT), query engines (Hive/Presto/Trino).
    - When to stream (Kafka/PubSub) vs batch; CDC patterns.
- **Practice**
    - Convert CSV → Parquet; benchmark query times; add partitioning (by date).
    - Re-implement a pandas pipeline in PySpark and validate parity.
- **Pitfalls**
    - Too many small files; skewed joins; using Python UDFs unnecessarily.
- **Artifacts**
    - Spark notebook; lake/warehouse schema + partition plan; cost notes.

NLP/CV/TS (intros to text, images, time series, embeddings)

Specialized modalities need tailored preprocessing and models. Start with transfer learning and simple baselines.

- **NLP (text)**
    - *Learn*: tokenization, TF-IDF vs embeddings, classification/NER, RAG basics.
    - *Practice*: classify support tickets; build a keyword extractor; evaluate F1.
    - *Pitfalls*: data leakage via document splits; domain shift; hallucinations in gen models.
- **CV (images)**
    - *Learn*: augmentation, CNNs, transfer learning, detection/segmentation metrics (mAP, IoU).
    - *Practice*: fine-tune a pre-trained model on a small image dataset; evaluate confusion matrix & IoU.
    - *Pitfalls*: train/val leakage from near-duplicate images; class imbalance.
- **TS (time series)**

    - *Learn*: trend/seasonality, exogenous vars, rolling CV, forecasting (ARIMA/Prophet/GBM).
    - *Practice*: forecast demand with quantile loss; compare naive vs model WAPE.
    - *Pitfalls*: using future info in features; not respecting temporal splits.
- **Embeddings (all modalities)**
    - *Learn*: vector representations; cosine similarity; vector DB basics.
    - *Practice*: build a mini semantic search over documents/images.
    - *Artifacts*: modality-specific evaluation scripts; embedding search demo.

Ethics & Governance (fairness, privacy, security)

Responsible AI prevents harm and keeps you compliant. Bake it in from day one—cheaper than fixing later.

- **Learn**
    - Privacy: PII/PHI inventory, minimization, consent, retention, access control.
    - Fairness: subgroup metrics, disparate impact checks, remediation.
    - Explainability: global/local (feature importance, SHAP); model cards.
    - Security: secrets management, encryption at rest/in transit, abuse/poisoning risks.
    - Governance: approvals, audit trails, Model Risk Management (MRM).
- **Practice**
    - Add subgroup performance tables to every evaluation; write a model card.
    - Run a privacy review: identify sensitive fields; propose minimization.
- **Pitfalls**
    - One-time "ethics check"; ignoring high-risk slices (e.g., low-prevalence groups).
- **Artifacts**
    - Model card; risk assessment; access logs; DPA/consent documentation.

Communication & Product Thinking (stakeholders, decision impact)

Great work fails without adoption. Tie models to decisions, tell a clear story, and measure real impact.

- **Learn**
    - Problem framing: "Who will do what differently because of this model?"
    - Stakeholders: roles (owner, approver, user, maintainer); update cadence.
    - Decision economics: cost–benefit, thresholds, capacity constraints, SLAs.
    - Storytelling: structure (context → insight → action → impact), visuals, FAQs.
- **Practice**
    - Write a one-page problem charter and a post-pilot readout with KPIs.
    - Define threshold rules tied to business costs; simulate outcomes.
- **Pitfalls**
    - Optimizing vanity metrics; ignoring adoption & change management.
- **Artifacts**
    - Problem charter; executive one-pager; dashboard link; rollout plan.

Data Analyst — BI & reporting

Data Analysts turn raw tables into decisions people can act on daily. They specialize in cleaning data, building reports, and answering "what happened/why" with clear visuals and narratives.

- **Core responsibilities**
    - Pull, clean, and join data from multiple sources.
    - Build reusable dashboards and ad-hoc analyses.

    - Define/maintain business definitions (metrics, dimensions).
    - Partner with business owners to translate questions into queries.
- **Typical outputs**
    - KPI dashboards, cohort/funnel reports, weekly business reviews.
    - Deep-dives: variance analysis, seasonality, anomaly investigations.
- **Tools & skills**
    - SQL (joins, window functions), Excel/Sheets, BI tools (Power BI/Tableau/Looker).
    - Basic stats (averages, CIs), data modeling (star/snowflake).
- **How success is measured**
    - Report accuracy and freshness, stakeholder adoption, reduced time-to-insight.
    - Clear documentation and consistent metric definitions.

Data Scientist — modeling & experimentation

Data Scientists build and validate predictive/causal models, moving from EDA to prototypes that inform or automate decisions.

- **Core responsibilities**
    - Frame problems, run EDA, engineer features, train/evaluate models.
    - Design/interpret experiments (A/B tests), communicate findings.
    - Collaborate with engineers to hand off models or decision rules.
- **Typical outputs**
    - Notebooks, model artifacts/pipelines, experiment readouts, model cards.
- **Tools & skills**
    - Python, pandas/NumPy, scikit-learn; stats (hypothesis testing), ML basics.
    - Versioning/experimentation (Git, MLflow/W&B).
- **How success is measured**
    - Offline/online metric lift (F1/AUC/MAE, business KPIs), clarity of insights, reproducibility.

ML Engineer — productionizing models

ML Engineers make models dependable in production. They package, serve, scale, and monitor ML systems.

- **Core responsibilities**
    - Convert research code to robust services/pipelines (batch/real-time).
    - Optimize latency/cost; add monitoring, logging, and alerting.
    - Manage model registries, CI/CD, and rollout/rollback strategies.
- **Typical outputs**
    - APIs (FastAPI/gRPC), batch jobs, feature and inference pipelines, infra-as-code.
- **Tools & skills**
    - Python, Docker, CI/CD, cloud (AWS/GCP/Azure), orchestration (Airflow), feature stores.
    - Observability (Prometheus/Grafana), testing (unit/integration).
- **How success is measured**
    - SLOs (p95 latency, uptime), data/metric drift detection, time-to-deploy, reliability.

Data Engineer — data pipelines & platforms

Data Engineers build the plumbing: reliable, scalable data flows and storage so others can analyze and model.

- **Core responsibilities**
    - Design schemas; build ETL/ELT pipelines (batch/streaming).

    - Ensure quality (validation, SLAs) and lineage.
    - Optimize performance and costs across lake/warehouse.
- **Typical outputs**
    - Curated tables/models, ingestion jobs, documentation, data contracts.
- **Tools & skills**
    - SQL, PySpark/Spark, dbt, Kafka/PubSub, cloud data stacks (BigQuery/Redshift/Snowflake).
    - Parquet/Delta, partitioning, orchestration (Airflow/Prefect).
- **How success is measured**
    - Data freshness/availability, query performance, reliability, cost efficiency.

Analytics Engineer — transforming for BI

Analytics Engineers sit between DE and BI. They model business logic in SQL/dbt so metrics are consistent and dashboards stay fast.

- **Core responsibilities**
    - Translate business rules into tested SQL models.
    - Maintain semantic layers/metric definitions; reduce duplication.
    - Enable self-serve analytics via clean, documented datasets.
- **Typical outputs**
    - dbt projects, semantic layers/metrics, BI-ready tables and documentation.
- **Tools & skills**
    - SQL, dbt, Git, BI tools (Looker/Power BI/Tableau), data testing (Great Expectations/dbt tests).
- **How success is measured**
    - Data model clarity, test coverage, dashboard performance, stakeholder trust.

MLOps Engineer — deployment & monitoring at scale

MLOps Engineers create the tooling and processes that keep ML systems healthy from training to serving.

- **Core responsibilities**
    - Build/maintain model registries, pipelines, and automated retraining.
    - Implement monitoring for data quality, performance, and drift.
    - Govern versions, approvals, and compliance for model changes.
- **Typical outputs**
    - Training/serving pipelines, monitoring dashboards, incident runbooks.
- **Tools & skills**
    - MLflow/SageMaker/Vertex, Docker/Kubernetes, CI/CD, feature stores, terraform/IaC.
- **How success is measured**
    - Deployment frequency/lead time, rollback safety, drift detection speed, audit readiness.

Applied/Research Scientist — advanced methods/GenAI

Applied/Research Scientists push capability frontiers (e.g., GenAI, CV/NLP), then adapt them to products.

- **Core responsibilities**
    - Explore new architectures, pretrain/fine-tune models, run benchmarks.
    - Publish/internal tech notes; collaborate on productization with MLEs.
- **Typical outputs**
    - SOTA prototypes, evaluation suites, papers/tech reports, reusable model libs.
- **Tools & skills**

    - PyTorch/JAX/TF, CUDA, distributed training, evaluation design, literature review.
- **How success is measured**
    - Research quality (benchmarks, novelty), transfer to product impact, reusable components.

Problem framing & business sense (turn questions into measurable objectives)

Great DS starts with a clear decision to improve. You translate vague asks ("reduce churn") into a concrete objective, target user, success metric, constraints, and a plan that can be tested and owned after launch.

- **Learn**
    - Define the decision, the user, and the action that will change because of your model.
    - Specify target variable, unit of analysis, population, horizon, and constraints (latency, budget, privacy).
    - Map success: business KPI ↔ ML metric; cost of errors; capacity limits (e.g., top-k outreach).
- **Practice**
    - Rewrite ambiguous prompts into a one-page problem charter with SMART goals and acceptance criteria.
    - Build a **cost matrix** for false positives/negatives and a rule of engagement (who acts, when).
- **Pitfalls**
    - Optimizing the wrong metric; skipping baselines; "boil-the-ocean" scope; starting with data before the decision.
- **Artifacts**
    - Problem charter, metric/guardrail table, baseline definition, decision playbook.

Python & SQL fundamentals (clean, readable code; efficient queries)

Python is how you transform data and build models; SQL is how you fetch and shape data at the source. Clean code and efficient queries save hours weekly and reduce bugs.

- **Learn**
    - Python: functions, modules, exceptions, typing, pathlib, packaging, virtual envs; pandas/NumPy idioms.
    - SQL: JOINs, GROUP BY/HAVING, subqueries/CTEs, window functions (LAG/LEAD/RANK), indexing basics.
    - Style/discipline: PEP8, docstrings, small pure functions, profiling.
- **Practice**
    - Build an idempotent Python script: read → clean → join → validate → write Parquet.
    - Solve 5 JOIN problems (one-to-many, many-to-many) and 5 window-function tasks (rolling sums, de-dupe).
- **Pitfalls**
    - pandas SettingWithCopy, accidental cartesian joins, SELECT * in prod queries, unbounded window specs.
- **Artifacts**
    - utils.py helpers, requirements.txt, SQL "cookbook" of reusable snippets.

Statistics intuition (uncertainty, tests, bias/variance)

Stats is how you reason under uncertainty: what's signal vs noise, and how confident you are. It powers A/B tests, intervals, and understanding model limits.

- **Learn**
    - Distributions (Bernoulli/Binomial/Normal/Poisson), CLT, sampling distributions.

    - Estimation: confidence intervals, effect size, power & sample size.
    - Hypothesis testing: null/alternative, p-values, Type I/II errors; multiple-testing corrections.
    - Bias vs variance; correlation vs causation; confounding and Simpson's paradox.
- **Practice**
    - Write an A/B readout: point estimate + 95% CI + p-value + plain-English implication.
    - Bootstrap a CI for a metric; simulate CLT to see means stabilize.
- **Pitfalls**
    - "p < 0.05 ⇒ truth" fallacy; p-hacking; ignoring confounders; reporting only averages.
- **Artifacts**
    - Stats cheat sheet, experiment template, quick sample-size calculator.

Data wrangling & EDA (cleaning, joining, spotting issues)

You'll spend most time here. Wrangling and EDA expose quality problems, suggest features, and prevent bad surprises later.

- **Learn**
    - Cleaning: types/units, parsing dates, deduping, missingness patterns, outlier handling (winsorize/clip).
    - Joining: keys/grain, surrogate keys, time-aware joins, anti/semijoins for QA.
    - EDA: distributions, pairplots, imbalance checks, leakage/drift detection, target–feature relationships.
- **Practice**
    - Produce a one-page EDA: 5 key charts + 5 "so-what" insights + a data dictionary.
    - Add assertive checks (row counts, null rates, unique keys) that fail fast.
- **Pitfalls**
    - Joining at the wrong grain, silent parse errors, using future information, ignoring drift over time.
- **Artifacts**
    - EDA notebook & images, data-quality report, feature/column dictionary.

Model basics (when to use what; evaluate properly)

Pick the simplest model that solves the problem, beat a baseline, validate correctly, and document trade-offs.

- **Learn**
    - Task ↔ method: linear/logistic for simple & interpretable; trees/GBM for nonlinear tabular; k-means/PCA for structure.
    - Workflow: split → baseline → features → model → tune → evaluate → calibrate (if probabilities matter).
    - Key knobs: regularization, class weights, early stopping; calibration curves.
- **Practice**
    - Build a naive baseline then beat it with 2 models; log metrics and rationale.
    - Tune GBM (learning_rate, max_depth, n_estimators) and plot validation curves.
- **Pitfalls**
    - Leakage, time-series evaluated with random splits, using accuracy on imbalanced data, overfitting for tiny gains.
- **Artifacts**
    - Experiment log, saved pipeline, model card (purpose, data, metrics, limits).

Communication (clear narratives, simple visuals, stakeholder updates)

If others can't understand or act on your work, it doesn't matter. Communicate decisions, not just numbers.

- **Learn**
  - Story shape: **context** → **insight** → **action** → **impact**, with an executive summary up top.
  - Visuals: pick the right chart, label clearly, show uncertainty, avoid clutter and dual axes.
  - Updates: regular cadence, risks/assumptions, asks/decisions needed.
- **Practice**
  - Write a 1-page executive readout and a 3-slide deck for the same analysis.
  - Turn a complex chart into a simple one with annotations and a headline.
- **Pitfalls**
  - Jargon, burying the lead, showing metrics without a recommendation, hiding caveats.
- **Artifacts**
  - Readout template, chart style guide, stakeholder FAQ.

Reproducibility (Git, environments, notebooks vs scripts discipline)

Reproducibility is the "science" in data science. Anyone should be able to rerun your work and get the same result.

- **Learn**
  - Git flow (branches, PRs, reviews), semantic commits, tagging releases.
  - Environments: pinned deps, lockfiles, seeds; data & model versioning (DVC/MLflow).
  - Structure: src/ packages, notebooks/ for exploration, config/logging/testing basics.
- **Practice**
  - Convert a notebook into a Python package + CLI entrypoint; reproduce a run from scratch.
  - Track experiments (params/metrics/artifacts) with MLflow or W&B.
- **Pitfalls**
  - Hidden notebook state, "works on my machine", magic numbers, untracked data changes.
- **Artifacts**
  - Repo skeleton, CI pipeline, model registry record, runbook for reruns.

Generative AI & foundation models (text, image, multimodal assistants)

Foundation models shift DS from "predict" to "generate & reason," enabling assistants that read, write, summarize, plan, and create images/audio/video. Multimodal models combine text, images, tables, and sensor data so one system can parse documents, inspect photos, and chat about dashboards.

- **Where it helps:** document processing (RAG), code/data copilots, customer support, creative assets, analytics Q&A.
- **Key enablers:** prompt engineering, retrieval-augmented generation, fine-tuning/LoRA, vector databases, guardrails.
- **KPIs to track:** task success rate, time-to-answer, hallucination rate, top-k retrieval precision/recall, cost per request.
- **Risks:** hallucinations, data leakage via prompts, IP/privacy exposure, unpredictable latency/cost spikes.
- **Practical actions:** start with RAG (not full fine-tune), add deterministic fallbacks, log prompts/answers, red-team prompts, maintain a prompt/model registry.

Real-time & edge inference (low-latency decisions, streaming features)

Businesses want decisions "in the moment"—fraud checks during checkout, recommendations as a user scrolls, safety checks on a factory line. Edge deployment keeps inference close to the event for reliability and privacy.

- **Use cases:** fraud/abuse detection, dynamic pricing, on-device quality inspection, driver/robot assistance, IoT anomaly alerts.
- **Tech patterns:** streaming features (Kafka/PubSub), low-latency feature stores, on-device/edge runtimes (ONNX, TensorRT), async queues, canary rollouts.
- **SLOs/KPIs:** p95/p99 latency, throughput (req/s), freshness (feature age), offline→online feature parity, alert precision@K.
- **Risks:** feature skew vs training data, cold-start, backpressure under spikes, on-device model drift.
- **Practical actions:** define online/offline contract for features, add shadow mode before going live, measure end-to-end latency (network+model), budget for circuit breakers & fallbacks.

Privacy-enhancing tech (federated learning, differential privacy)

PETs let you learn from sensitive data without centralizing raw records or exposing individuals. They're becoming table stakes where regulation or trust is critical.

- **Approaches:** federated learning (train where data lives), differential privacy (noise to protect individuals), secure enclaves/TEEs, secure aggregation, synthetic data.
- **When to use:** healthcare/finance/education, cross-org collaborations, mobile/edge training, jurisdictions with strict residency.
- **KPIs:** privacy budget (ε), utility loss vs baseline, participation rate of clients, secure aggregation coverage, audit findings.
- **Risks:** degraded accuracy if privacy budget is too tight, complex ops, false sense of security if governance is weak.
- **Practical actions:** classify data/PII, select PET per risk, start with DP on analytics queries, pilot FL with a narrow model, document ε and trade-offs in model cards.

Automation & copilots (AutoML, code/data assistants)

Copilots lift productivity: they scaffold pipelines, generate SQL, write tests, and review notebooks. AutoML covers baseline modeling/tuning so scientists focus on framing, evaluation, and deployment.

- **Targets:** SQL generation/validation, data quality rule suggestions, feature discovery, pipeline boilerplate, docstrings/tests, unit data checks.
- **Benefits:** faster iteration, fewer trivial bugs, better onboarding, standardized scaffolds.
- **KPIs:** cycle time (idea→first result), review defects caught, % code/tests suggested by copilot adopted, analyst hours saved.
- **Risks:** cargo-culted code, hidden data assumptions, dependency on vendor models.
- **Practical actions:** define "copilot-ready" repos with templates, require human review, keep a pattern library of approved prompts, restrict copilot access to non-sensitive code/data.

Regulation & model governance (AI acts, audit trails, documentation)

Scrutiny is rising: organizations must prove models are safe, fair, documented, and controllable. Governance turns ad-hoc ML into audited, repeatable practice.

- **Core components:** risk tiering (use-case risk levels), approvals, model cards & data sheets, change management, incident playbooks, independent validation, lineage & access logs.

- **What to document:** purpose, data sources, consent, metrics (overall + subgroups), limitations, human oversight, rollback criteria, retraining cadence.
- **KPIs:** audit pass rate, time to approve changes, % models with complete documentation, incidents MTTR, subgroup gap thresholds.
- **Risks:** "paper compliance" without monitoring, untracked prompt/model drift, unclear ownership.
- **Practical actions:** set model inventory & registry, standardize review checklists, automate lineage/metrics capture, schedule periodic fairness & drift reviews.

Demand drivers (digital transformation, data abundance, automation ROI)

Organizations fund DS because digital processes throw off data, AI can turn that data into decisions, and automation improves margins at scale. Boards see DS as a lever for growth, risk control, and cost reduction.

- **What's pushing demand**
  - Cloud + SaaS adoption → easier data access and experimentation.
  - Data exhaust from apps, sensors, and logs → new signals to exploit.
  - Labor scarcity & cost pressure → automation and decision support.
  - Competitive dynamics → personalization, faster cycle times, better UX.
- **Typical executive goals**
  - Revenue lift, churn reduction, fraud/risk control, working-capital efficiency.
  - SLA/latency improvements, "do more with less," regulatory assurance.
- **Buying signals**
  - Data lake/warehouse already in place, analytics backlog, manual repetitive decisions, rising service costs.
- **KPIs sponsors care about**
  - $/savings or lift per model, payback period, adoption/coverage, p95 latency, compliance pass rate.

High-investment sectors (finance, healthcare, e-commerce, logistics, SaaS)

Spend concentrates where data is rich, decisions repeat, and regulation or competition is intense. Each sector has a typical "first 3" use cases.

- **Finance**
  - *Use cases*: credit scoring, fraud/AML, next-best-action.
  - *Buyer*: Risk/Collections/Head of Analytics. *Constraints*: explainability, audit, latency.
- **Healthcare**
  - *Use cases*: readmission risk, imaging triage, capacity planning.
  - *Buyer*: Hospital ops, payers, digital health. *Constraints*: PHI, clinical validation.
- **E-commerce/Retail**
  - *Use cases*: recommendations, pricing, demand forecasting.
  - *Buyer*: Growth/Category/Supply chain. *Constraints*: seasonality, catalog churn.
- **Logistics/Manufacturing**
  - *Use cases*: ETA/pick-path optimization, predictive maintenance, quality inspection.
  - *Buyer*: Ops/Plant managers. *Constraints*: edge latency, downtime cost.
- **SaaS/Tech**
  - *Use cases*: abuse detection, search/ranking, product analytics.
  - *Buyer*: Product/Platform. *Constraints*: scale, cost per inference.

Role trends (ML/GenAI engineers, analytics engineers, MLOps)

Hiring patterns show where opportunity is moving: from one-off analyses → reliable ML systems → GenAI-enabled apps.

- **Growing roles**
  - **ML/GenAI Engineer:** ship LLM/RAG features, latency/cost tuning, safety/guardrails.
  - **Analytics Engineer:** dbt/semantic layers so metrics are consistent and BI scales.
  - **MLOps Engineer:** monitoring, registries, CI/CD, retraining automation.
- **Stable roles**
  - **Data Scientist / Product DS:** experiment design, model development, KPI strategy.
  - **Data Engineer:** ingestion, quality, lineage, batch/streaming platforms.
- **Signals of demand**
  - Job posts asking for vector DBs, prompt/retrieval pipelines, dbt + BI ownership, model governance experience.
- **Implication**
  - Opportunities favor people who bridge **models ↔ production** and **data modeling ↔ business metrics**.

Product spaces (recommendation, forecasting, fraud/risk, copilots, vector search)

These are repeatable problem types with clear ROI and maturing patterns. They're good bets for products, services, or internal platforms.

- **Recommendation/Ranking**
  - *Value*: lift conversion and AOV with personalization.
  - *Moat*: first-party interaction data, inference latency. *Metrics*: CTR, revenue/session, diversity.
- **Forecasting & Optimization**
  - *Value*: reduce stockouts/overstocks, improve staffing and cash cycles.
  - *Moat*: high-quality exogenous signals. *Metrics*: WAPE/MAPE, service level, markdowns.
- **Fraud/Risk/Trust & Safety**
  - *Value*: prevent loss with minimal friction.
  - *Moat*: labels + real-time graph features. *Metrics*: precision@K, false-positive rate, manual review load.
- **Copilots/Assistants (GenAI)**
  - *Value*: compress analyst/agent time; self-serve analytics/search.
  - *Moat*: retrieval quality + guardrails + domain prompts. *Metrics*: task success, time-to-answer, hallucination rate.
- **Vector Search & RAG platforms**
  - *Value*: find semantically similar content across text/images/code.
  - *Moat*: embeddings quality + chunking + metadata pipeline. *Metrics*: recall@K, MRR/NDCG, cost/query.

Build vs buy economics (TCO, lock-in, talent availability)

Choosing to build or buy hinges on total cost, speed, and how "differentiating" the problem is for your org.

- **Cost components (TCO)**
  - Data work (ingestion/cleaning/labels), infra (compute/storage/egress), platform tooling, talent, security/compliance, ongoing monitoring.
- **When to buy**

    - Commodity capability (OCR, transcription, generic chat), need speed to market, limited DS/MLE bandwidth, strong vendor benchmarks & SLAs.
- **When to build**
    - Proprietary data is your moat, tight integration/latency constraints, scale makes vendor usage expensive, IP/regulatory needs.
- **Lock-in checks**
    - Export paths for data/models, BYO-key encryption, transparent pricing, offline/batch fallbacks.
- **Simple rule**
    - *Pilot with buy; if usage > threshold and data advantage emerges, re-platform to build.*

Geographic considerations (remote work, regional hubs, compliance regimes)

Talent and compliance shape where and how you execute. Remote expands the pool; regulations constrain data flow.

- **Talent hubs**
    - Mature DS/MLE communities (US, EU, India, SEA) + emerging hubs near universities.
    - Nearshore/offshore models for 24/7 ops and cost leverage.
- **Compliance & data residency**
    - Sector rules (banking/health), cross-border transfer limits, localization laws.
- **Operating model**
    - Hybrid teams: product & data owners near users; platform & model ops distributed.
- **Practical tips**
    - Keep sensitive training data region-local; move models not data; document data contracts.

Barriers & risks (talent gap, data access, change management)

Most failed DS programs fall on organizational—not algorithmic—issues. Plan for them early.

- **Common blockers**
    - Unclear problem owners; missing labels or data contracts; privacy/security reviews slow access.
    - Shadow metrics (different teams compute KPIs differently); low adoption of outputs.
- **Risk patterns**
    - Model built without deployment path; drift/monitoring missing; ROI not measured.
- **Mitigations**
    - Problem charters with owners/KPIs, metric definitions in a semantic layer, data access playbooks, model registry + monitoring SLOs, change-management plan (training, incentives).