

Learning path built into the repo (how you'll level up)

- **RF/IP intuition:** Compare changes in SINR/RSRP to PRB/latency swings; note the cause-effect patterns you can explain in an interview.
 - **Lakehouse discipline:** You'll implement Bronze→Silver→Gold, inspect partitions, and explain predicate/column pruning benefits.
 - **Anomaly framing:** Start with transparent rules; evolve to supervised scoring with proper time-aware validation.
 - **Explainability:** Use feature importances/SHAP to narrate *why* a cell is risky.
 - **Forecasting:** Fit Prophet to latency; discuss capacity planning & expected QoE.
-

Next steps (planned to use and showcase Verizon's tools & techniques)

A) Gold features & labels (Spark) — *Hadoop/Lakehouse depth*

File: `spark/silver_to_gold.py`

- Hourly aggregates per cell: `latency_ms_avg/p95`, `prb_util_pct_avg`, `thrpt_mbps_avg`, RF means (`sinr_db_avg`, `rsrp_dbm_avg`), `jitter_ms_avg`, `pkt_loss_pct_avg`, `drop_rate_pct_avg`, `anomaly_rate`, `n_samples`, plus `ts_hour`.
- **Label:** `y_next_anomaly = (lead(anomaly_rate) > 0)` per cell using a window.
- Partition Gold by `date,region`.

Why this shows knowledge: window functions, percentiles, partitioning for scale, stable contracts from Silver→Gold.

B) Scoring model & anomaly detection — *scikit-learn / XGBoost / NumPy*

File: `ml/train_baseline.py`

- Features: latency p95/avg, PRB avg, throughput avg, jitter/loss/drop, RF averages, `anomaly_rate`, hour of day.
- Models: **XGBoost** (fallback **RandomForest** if no GPU), time-aware split (last 20% test), metrics: **AUC**, precision/recall/F1, calibration plot.
- Artifacts: `model.joblib`, `metrics.json`, `feature_importance.csv` (+ optional SHAP summary).

Why this shows knowledge: end-to-end **scoring solution** (train → evaluate → persist → serve), anomaly detection beyond rules, interpretability hooks.

Operational “solution” angles to discuss in the README/UI or interview:

- If risk↑ due to **PRB saturation** → short-term: scheduler tweaks, load balancing, carrier add; medium-term: capacity augments.
- If risk↑ due to **low SINR** → tilts/power, neighbor optimization, interference mitigation.
- If risk↑ due to **IP QoS** → backhaul checks, QoS policy review, packet loss root cause.

C) Forecasting — *Prophet (formerly Facebook Prophet)*

Notebook: `notebooks/latency_prophet.ipynb`

- Prophet on hourly latency per cell (seasonality + changepoints).
- Backtest with rolling origin; report MAPE/MAE; compare to naïve seasonal baseline.

Why this shows knowledge: capacity/QoE trend forecasting and operational planning.

D) Data warehouse mirrors — *Teradata alignment*

File: `sql/build_duckdb.py` (plus `sql/postgres/*` DDL)

- Materialize `gold_cell_hourly` from parquet and create view `mart_top10_by_region`.
- Provide equivalent DDL for Postgres/Teradata (sketch), including sort/dist keys where relevant.

Why this shows knowledge: view layer and analyst marts; portability between lake and warehouse; how you'd stage into **Teradata**.

E) Graph analytics — *Neo4j / Graph DB*

Files: `neo4j/schema.cypher`, `neo4j/centrality.cypher`

- Nodes: `(:Cell {cell_id, region})`
- Edges: `(:Cell)-[:NEIGHBOR_OF {distance, weight}]->(:Cell)` or `[:HANDOVER_TO {count}]`
- Compute degree/betweenness; join **model risk × centrality** to prioritize fixes.
- UI hook: click a risky cell → neighbor subgraph & centrality values.

Why this shows knowledge: using **graph** to improve triage decisions when multiple cells degrade.

F) NoSQL option — *Cassandra/DynamoDB*

- Store static cell metadata & neighbor lists keyed by `cell_id`.
 - Spark job reads NoSQL topology and joins with Silver/Gold for context.
 - Notes on consistency and read patterns (eventual vs strong) for ops realism.
-

G) Dataproc/EMR & Jupyter — *orchestration at scale*

Folder: `ops/runbooks/`

- Dataproc submit scripts for Spark jobs and Jupyter initialization.
 - EMR Steps/Livy equivalents.
 - Guidance on cluster sizing, spot usage, IO/cache tuning.
-

H) Data Quality & Reliability — *operational excellence*

Folder: `dq/`

- Checks for nulls, ranges, timestamp precision, and schema drift.
 - Job metrics: rows written, partitions touched, p95 latency per run.
 - Optional: Great Expectations or custom assertions + CI hook.
-

Troubleshooting (Windows)

- **Pandas build errors** → use Python **3.10** (prebuilt wheels).
 - **PySpark refuses wheels-only** → install `pyspark` without `--only-binary`.
 - **Parquet timestamp error** → ensure generator writes **tz-naive μ s** timestamps.
 - **Java not found** → install Temurin JDK 11/17, reopen PowerShell, rerun Spark.
 - **UI KeyError on `region`** → always read parquet as **hive-partitioned dataset** (already fixed in `streamlit_app.py`).
-

Demo flow (now → later)

Now (live today):

1. Generate Bronze → build Silver → open Streamlit; filter by Region/Date; explain top risky cells using RF/IP reasoning.

Soon (incremental showcase):

1. Build **Gold** features (Spark).
2. Train **XGBoost/scikit-learn** model; show risk scores & importances; narrate mitigation options.
3. Create **DuckDB/Postgres/Teradata-style** marts; run an analyst query.
4. Add a **Prophet** notebook for latency forecasting on a cell.
5. Import neighbors to **Neo4j**, compute centrality; show **risk × centrality** prioritization.
6. Submit Spark jobs on **Dataproc/EMR** with the same contracts; optionally run Jupyter on cluster.

Interview one-pager (phrasing you can use)

- “We apply **lakehouse discipline**: Bronze→Silver partitions, Arrow↔Spark precision guardrails, boolean logic correctness.”
 - “KPIs reflect **RF/IP realities**; **p95 latency** & **PRB utilization** are primary congestion signals; RF (SINR/RSRP) modulates risk.”
 - “Scoring with **XGBoost/scikit-learn** and forecasting with **Prophet** gives interpretable decisions for NOC workflows.”
 - “**Scale path**: same jobs on **Dataproc/EMR**; marts mirror **Teradata**; **Neo4j** centrality directs fixes when multiple cells degrade.”
 - “This repo doubles as a **teaching artifact**—each component is documented with why/when/how so new engineers (and interviewers) can trace decisions.”
-

Guiding principle: every commit should strengthen at least one requirement above. If a task doesn't advance **scoring/anomaly**, **lakehouse/warehouse**, **ML/forecasting**, or **4G/5G + IP** understanding, we re-scope it.

Chunk 1 — Scoring Models & Anomaly Detection (primary)

Req tie: “Prior experience in building scoring models, anomaly detection solutions.”

Goal: Move from a transparent rule to an operational scoring solution that predicts “**anomaly next hour?**” per cell.

Build

- `spark/silver_to_gold.py` — hourly features: `latency_ms_avg/p95`, `prb_util_pct_avg`, `thrpt_mbps_avg`, `jitter_ms_avg`, `pkt_loss_pct_avg`, `drop_rate_pct_avg`, `sinr_db_avg`, `rsrp_dbm_avg`, `anomaly_rate`, `n_samples`, `ts_hour`; label `y_next_anomaly = lead(anomaly_rate) > 0`.
- `ml/train_baseline.py` — **XGBoost** (fallback **RandomForest**) with time-aware split; save `model.joblib`, `metrics.json`, `feature_importance.csv` (+ optional SHAP).
- Surface scores in the UI (add “Top risk cells (model)” table + feature-importance snippet).





Proof / Demo

- AUC/PR/ROC chart, confusion matrix, calibrated probabilities.
- Show how rule vs model disagrees on a cell and explain why (e.g., high PRB + low SINR).

You'll learn

- Labeling, leakage control, time-series validation, calibration, and model explainability.

Definition of done


-  Gold parquet exists;  model + metrics saved;  scores visible in Streamlit;  short “model card” in repo.
-

Chunk 2 — Large-Scale Data Engineering & Experimentation

Req tie: “Experience with large data sets to find opportunities... and test effectiveness.”

Goal: Make the pipeline robust and experiment-ready.

Build

- Extend simulator to generate days/weeks of data; optional “load profile” toggles (busy hour, interference).
- Add  checks (nulls, ranges, schema drift) and job metrics (rows written, partitions touched, p95 latency).
- Micro-benchmarks: show partition pruning & column pruning effects.

Proof / Demo

- CLI prints of partitions written; timings before/after pruning; DQ report.

You'll learn

- Practical Spark performance levers, reproducible experiments, and result logging.

Definition of done

-  DQ script runs in CI-style;  README table with micro-benchmark numbers.
-

Chunk 3 — Data Warehouse & Lake Tech (Teradata, Hadoop, NoSQL, Graph)

Req tie: “Knowledge of data warehouse and data lake technology (Teradata, Hadoop, No SQL, graph DB).”

Goal: Show how the same contracts feed both lake and warehouse—and where NoSQL/graph fit.

Build

- Lakehouse: we already have **Hadoop/Spark** Bronze→Silver; add **Gold** (Chunk 1).
- Warehouse: `sql/build_duckdb.py` to materialize `gold_cell_hourly` and view `mart_top10_by_region`; add `sql/postgres/*` DDL mirroring the view (Teradata-style keys, distribution notes).
- NoSQL: a stub table design (`cell_meta` keyed by `cell_id`) for Cassandra/DynamoDB; sample Spark read join.
- Graph: `neo4j/schema.cypher` + `centrality.cypher` (neighbors/handovers → degree/betweenness); join **risk × centrality**.

Proof / Demo

- Run the mart view; show top-10 risky cells per region from DuckDB/Postgres.
- Neo4j browser screenshot: risky cell subgraph with centrality.

You'll learn

- When to choose Parquet vs warehouse vs NoSQL vs graph—and how to stitch them.

Definition of done

- ☒ DuckDB table + view built; ☒ Postgres DDL checked in; ☒ Neo4j scripts present with sample CSV export.
-

Chunk 4 — AI/ML & Forecasting Stack (NumPy, scikit-learn, XGBoost, Prophet, Jupyter, Dataproc)

Req tie: “Knowledge and experience applying AI/ML... (NumPy, Scikit-Learn, Dataproc, Jupyter, Facebook Prophet, XGBoost...)”

Goal: Demonstrate breadth with a forecasting companion to the classifier.

Build

- [notebooks/latency_prophet.ipynb](#) — Prophet on hourly latency per cell; rolling-origin backtest; MAPE/MAE; compare to naïve baseline.
- Add [ops/dataproc_submit.md](#) showing how to run Spark jobs & Jupyter on Dataproc.

Proof / Demo

- Backtest scores; side-by-side plots; README clip showing commands for Dataproc submission.

You'll learn

- Forecast framing, backtesting rigor, and cloud execution patterns.

Definition of done

- ☒ Notebook with plots & metrics; ☒ Dataproc runbook referencing the same PySpark entry points.

Chunk 5 — Network Expertise: 4G/5G, IP Networking, RF/SP tools & principles

Req tie: “Network expertise... 4G/5G, IP Networking, RF/SP engineering tools/principles.”

Goal: Show real domain understanding, not just data plumbing.

Explain (documented in README + a short [docs/network_notes.md](#))

- **RF:** RSRP (coverage), RSRQ, **SINR** (quality) → affects MCS/throughput and retransmissions.
- **Capacity:** **PRB utilization** indicates RAN congestion.
- **IP QoS:** latency, jitter, packet loss, drop rate reflect user experience.
- **Operational mitigations:**
 - PRB-driven risk → scheduler/load-balancing/carrier add.
 - Low SINR risk → tilt/power/neighbor optimization/interference mitigation.
 - IP-driven risk → backhaul QoS checks, routing/peering, loss remediation.

Hands-on tasks

- Create synthetic **scenarios** (busy hour surge, interference spike) and observe model response.
- Add small “mitigation notes” column to the UI alongside each top-risk cell.

Definition of done

-  Network notes doc;  scenario toggles in simulator;  mitigation hints visible in UI.

Chunk 6 — Performance, Capacity, Reliability, Scalability

Req tie: “Technical knowledge of improving system performance, capacity, reliability and scalability of the network.”

Goal: Make reliability and scale trade-offs explicit.

Build

- Job SLIs/SLOs: write counts, partition counts, error rates, latency percentiles.
- Packaging & reproducibility: requirements lock; env check script; seed regeneration.
- Cloud scale path: EMR/Dataproc step submission; spot/cost notes; storage layout guidance.

Proof / Demo

- A small “run report” printed after each Spark job; README section on cost/perf levers.

Definition of done

-  Run report JSON per job;  scale notes with concrete settings.
-

What's already implemented (you can demo today)

- Spark Bronze→Silver with `anomaly_flag` and hive partitions (`date`, `region`).
- Streamlit UI reading partitions correctly (Region/Date filters, sample table, **Top cells by anomaly rate**).
- **Timestamp & type guardrails** (tz-naive microseconds; boolean OR then cast).

Run

```
py -3.10 -m venv .venv; Set-ExecutionPolicy -Scope Process
-ExecutionPolicy Bypass -Force; .\.venv\Scripts\Activate.ps1
@('pandas==2.2.2','pyarrow==16.1.0','pyspark==3.5.1','streamlit==1.37.
1','duckdb==1.0.0') | Set-Content -Encoding ASCII requirements.txt
python -m pip install -U pip setuptools wheel; python -m pip install
-r requirements.txt
python data\simulate_kpis.py
```

```
$env:JAVA_HOME=(Split-Path -Parent (Split-Path -Parent (Get-Command
java | Select-Object -ExpandProperty Source)));
$env:PATH="$env:JAVA_HOME\bin;$env:PATH"
python spark\bronze_to_silver.py
streamlit run app\streamlit_app.py
```

KPI & anomaly primer (teach-yourself section)

- **Why PRB + SINR drive QoE:** high PRB (congestion) pushes up scheduling delay; low SINR reduces spectral efficiency → more RBs per bit → raises PRB further → latency spikes.
 - **Why p95 latency:** operators triage tail behavior, not just means.
 - **Why partitions:** speed + cost—Spark prunes files by **date|region**, only scanning what's needed.
 - **Why guardrails:** tz-naive μ s avoids Arrow↔Spark parquet incompatibilities.
-

Stretch commands (once chunks are in)

```
# Build Gold features (Chunk 1 & 2)
python spark\silver_to_gold.py

# Train baseline (XGBoost / scikit-learn)
python -m pip install scikit-learn==1.5.1 xgboost==2.1.1 shap==0.45.1
python ml\train_baseline.py

# Build a DuckDB mart (Chunk 3)
python sql\build_duckdb.py
```

Interview one-liners you can use

- “We apply **lakehouse discipline**: Bronze→Silver→Gold with partitioning and schema guardrails; timestamps standardized to μ s so Arrow and Spark agree.”
 - “**Anomaly detection** starts with a transparent rule for ops trust, then a supervised **scoring** model with time-aware validation and calibrated probabilities.”
 - “KPIs encode **RF/IP principles**—**PRB** and **SINR** explain most QoE pain; p95 latency is the triage metric.”
 - “Same contracts run on **Dataproc/EMR**; marts mirror **Teradata** patterns; **Neo4j** centrality multiplies impact when many cells degrade.”
-

Roadmap summary (checkboxes)

- Gold features + label (`silver_to_gold.py`)
- Baseline model (XGBoost/RandomForest) + metrics + UI scores
- DuckDB mart + Postgres/Teradata DDL
- Prophet latency notebook + backtest
- Neo4j schema + centrality + risk×centrality view
- NoSQL cell metadata join
- DQ checks + run reports + cloud runbooks (Dataproc/EMR)

Guiding principle: every commit should advance at least one requirement above *and* one of your roadmaps:

- **AI Expert:** modeling/forecasting/interpretability.
- **Layer-2 Orchestrator:** Spark/partitions/contracts/marts/ops reliability.

You’re building not just a demo—but a **teaching artifact** that proves you can design, explain, and scale intelligent network systems end-to-end.

3) What to look for (talk track)

- **Daily seasonality** — latency peaks often follow user behavior; model captures this with a 24-hour cycle.
- **Confidence band** — uncertainty widens into the future; useful for alert thresholds and ticketing.
- **Cutover** — the dashed line shows where we stop observing and begin forecasting.
- **Ops use** — “If the forecast crosses our SLO (e.g., 60ms), we can pre-emptively shift traffic, throttle per-cell PRB allocations, or schedule backhaul checks.”

Why PRB Utilization Matters for Verizon

- **Direct tie to RF/SP engineering tools & principles:** PRB utilization measures how much of the available radio resources (in LTE/5G scheduling) are being consumed. High PRB utilization often signals congestion in the air interface — a core concern for RAN optimization teams.
- **Predictive of customer experience:** Sustained high PRB utilization directly impacts throughput, latency, and call drop probability. Verizon’s role expects knowledge of how KPIs interrelate in **4G/5G networks**, and PRB is central to that story.
- **Better than latency for this showcase:** Latency forecasts can be noisy at small horizons, but PRB utilization is a steady, capacity-driven metric. Forecasting PRB utilization shows you can anticipate **cell congestion** before it affects users.
- **Relates to system performance & scalability:** Ties directly into Verizon’s requirement for improving **capacity, reliability, and scalability** of the network.

Options for the next chunk (pick one tomorrow)

1. Bring PRB forecasts into Streamlit

- Add a small “**Forecast**” section under the model table.
- Show a sparkline + last forecasted value for the selected region/cell.
- Button: “Open full forecast plot” (link to saved PNG).
- *Roadmaps:* user-facing decision support; orchestration/UI polish.

2. Neighbor risk (Neo4j) MVP

- Load cells/handovers into Neo4j, compute degree/centrality.
- “Risk propagation” = risky cell + high-centrality neighbors → alert.
- *Roadmaps*: graph analytics + network engineering insight.

3. Model quality & governance

- Add simple **data contracts** (PRB in 0–100, hourly monotonicity, min coverage).
- Add **train/serve parity checks** and metrics history (AUC/PR, coverage).
- *Roadmaps*: production-minded ML, auditability.

4. Postgres mirror (later)

- We paused here; once Docker/Postgres is set, mirror **gold** + **scores** for BI.

Suggested plan for tomorrow (if you want my pick)

Option 1: Wire forecasts into the app

1. Read `artifacts/forecast_prb_latest.csv` in `streamlit_app.py` (guarded by region).
2. Show **last observed PRB** vs **next 6–12h forecast** with a tiny line chart.
3. Add a rule: flag cells where forecasted PRB > **80%** for ≥N hours → “capacity risk”.

This will make the demo feel end-to-end and very “Verizon ops” ready.

If you’re happy with this, we can start that chunk next. Great progress today—nice work staying skeptical of the first model and documenting the fix.

✓ What we covered (mapped to chunks)

- **Chunk 1 – Scoring & anomaly detection**
 - Built Spark **Silver** → **Gold** hourly features.
 - Trained **XGBoost** baseline (`ml/train_baseline.py`), saved **model + metrics + feature_importance**.
 - Streamlit: **Top cells (model) + Alerts table** with quick reason codes.
 - README updated with results and model card notes.
- **Chunk 3 – Warehouse/Lake tech (partial)**
 - DuckDB materialization script (`sql/build_duckdb.py`) created; optional Postgres mirror script stubbed (we deferred the actual connection).
- **Chunk 4 – AI/ML & forecasting (partial)**
 - Forecasting flow stood up.
 - We switched to the **PRB utilization** KPI (most relevant to Verizon), produced `artifacts/forecast_prb_latest.csv`, and added plots.
 - Diagnosed an unrealistic forecast (“cliff to 0%”), documented the issue, and **fixed** it by increasing history and min-points; README snippet added.
- **Chunk 5 – Network expertise (seeded)**
 - README now ties KPIs (PRB, SINR/RSRP/RSRQ, latency/jitter/packet loss) to **RF/IP principles** and operational mitigations.

What remains in those chunks

- Chunk 1: (nice-to-have) add ROC/PR plots + calibration curve to artifacts/README.
- Chunk 3: finish **mart view** and **graph/neighbor centrality** join; Postgres mirror is optional and can wait.
- Chunk 4: optional notebook/backtest; add Streamlit panel for forecast preview.

What we start tomorrow

Chunk 3 — Data Warehouse & Lake Tech (Graph + Mart)

Goals

- Materialize a **mart** (top-risk cells per region) in DuckDB.
- Add **neighbor/graph signal** (degree/betweenness centrality) and join with model risk.
- Surface a simple “**neighbor risk**” table in Streamlit.

Plan

1. DuckDB mart

- `sql/build_duckdb.py`: add a view/table `mart_top10_by_region` from Gold + scores.
- Output: `artifacts/mart_top10_by_region.csv` for quick inspection.

2. Neighbor graph (no external installs required)

- Create `data/sim/neighbors.csv` (synthetic handovers/adjacency across our 12 cells).
- `graph/build_neighbor_centrality.py`: compute **degree** and **betweenness** (NetworkX), save `artifacts/neighbor_centrality.csv`.

3. Join & expose

- `ml/join_risk_neighbor.py`: join **risk_score** × **centrality** → `artifacts/neighbor_risk.csv`.
- Streamlit: add a “**Neighbor risk**” section (top cells with high risk & high centrality) + one-click filter by region/cell.

4. (Optional later) Neo4j path and Postgres mirror (we’ll keep the scripts ready but stay pure-Python first to avoid new installs).

What you’ll see at the end

- `artifacts/mart_top10_by_region.csv`

- `artifacts/neighbor_centrality.csv`
- `artifacts/neighbor_risk.csv`
- Streamlit section “**Neighbor risk (risk × centrality)**”

So yes—everything we did is inside the chunk framework, and **tomorrow we start with Chunk 3** (Graph + Mart), keeping Postgres for later.