MGMT 467 — Step■by■Step Lab Handout

Kaggle ➜ Google Cloud Storage ➜ BigQuery ➜ Data Quality (DQ)

Overview

In this lab you will: (1) fetch a Netflix-style dataset from Kaggle, (2) stage the raw CSVs in Google Cloud Storage (GCS), (3) load them into BigQuery (BQ), and (4) run core Data Quality (DQ) profiling: missingness, duplicates, outliers & anomaly flags. You will generate code incrementally using LLM prompts (Gemini/Vertex AI), verify after each step, and record brief reflections explaining the what & why.

Prerequisites (Before You Start)

- Active Google Cloud project with billing enabled.
- Permissions for GCS & BigQuery.
- Kaggle account and API token (kaggle.json).
- Consistent region choice (e.g., us-central1) for bucket & compute.

Learning Objectives

- Explain why we stage data in GCS and then load to BigQuery.
- Build an idempotent, auditable pipeline with clear verification points.
- Diagnose missingness (MCAR/MAR/MNAR), duplicates (exact vs near), and outliers (IQR/winsorize).
- Connect data quality choices to business and ML impact.

Step 0: Environment Setup (Auth + Project/Region)

Why this matters

- Gives Colab permission to use gcloud, GCS, and BigQuery on your behalf.
- Sets consistent PROJECT_ID and REGION to control cost and avoid cross■region surprises.

What you will do

1 Authenticate to Google Cloud in Colab.

2 Set PROJECT_ID via input; set REGION (e.g., us-central1); export GOOGLE_CLOUD_PROJECT.

3 Run gcloud config set project and print active project/region.

Verify before moving on

- gcloud config get-value project returns your PROJECT_ID.
- REGION variable echoes as expected (e.g., us-central1).

Reflection (write 2–4 sentences in your notebook)

- Why define PROJECT_ID and REGION up front? What breaks if you forget?

Step 1: Kaggle API Setup

Why this matters

- Enables reproducible, audit■friendly downloads without manual clicks.
- Storing kaggle.json correctly protects secrets.

What you will do

1. Upload kaggle.json and save to ~/.kaggle/kaggle.json with 0600 permissions.
2. Confirm Kaggle CLI is available (`kaggle --version`).

Verify before moving on

- `kaggle --help` prints usage; no permission errors are shown.

Reflection (write 2–4 sentences in your notebook)

- Why require 0600 file permissions for API tokens? What risks are mitigated?

Tips

- If upload fails, regenerate a new token from Kaggle > Account.

Step 2: Download & Unzip Dataset

Why this matters

- Keeping raw files under a predictable path (/content/data/raw) simplifies automation and auditing.
- Listing file sizes helps catch partial downloads or mismatches early.

What you will do

1. Create /content/data/raw.
2. Download dataset sayeeduddin/netflix-2025user-behavior-dataset-210k-records to /content/data.
3. Unzip into /content/data/raw and list CSV inventory (names & sizes).

Verify before moving on

- Exactly six CSV files are present and readable.

Reflection (write 2–4 sentences in your notebook)

- Why maintain a clean file inventory? How does it help downstream in BQ and Git?

Step 3: Create GCS Bucket & Upload CSVs

Why this matters

- Staging in GCS gives a stable, versionable source for BigQuery loads.

- A single prefix (gs:///netflix/) keeps artifacts organized.

What you will do

1. Create a globally unique bucket in your REGION (use a random suffix).
2. Upload all CSVs to gs:///netflix/.
3. Record the bucket name for later steps.

Verify before moving on

- gcloud storage ls gs:///netflix/ shows all objects with sizes.

Reflection (write 2–4 sentences in your notebook)

- List two benefits of staging in GCS vs. loading directly from local Colab.

Tips

- Use lowercase, hyphenated bucket names; avoid personally identifying info in names.

Watch■outs

- Buckets are global—names that worked before may be taken later.

Step 4: Create BigQuery Dataset & Load Tables

Why this matters

- Centralizes data for SQL analysis & ML; supports lineage and access control.
- Autodetect speeds up labs; explicit schemas can be applied later for stricter governance.

What you will do

1. Create dataset netflix (idempotent; handle 'already exists').
2. Load six CSVs (users, movies, watch_history, recommendation_logs, search_logs, reviews) from GCS with autodetect and header skip.
3. Run row■count queries to confirm loads.

Verify before moving on

- A single query returns table_name, row_count for all six tables in .netflix.

Reflection (write 2–4 sentences in your notebook)

- When is autodetect acceptable? When should you enforce explicit schemas and why?

Watch■outs

- Ensure dataset location (US) aligns with your bucket/region policies to avoid multi■region egress.

Step 5: Data Quality Profiling (Missingness, Duplicates, Outliers)

Why this matters

- Bad data → bad models; profiling quantifies risk and guides cleaning policies.
- MAR (missing at random) patterns can bias models if ignored.
- Duplicates inflate engagement metrics and corrupt labels.
- Outliers can destabilize linear/logistic models; winsorizing provides robustness.

What you will do

1. Missingness (users): compute % missing for region, plan_tier, age_band; then check MAR by region.
2. Duplicates (watch_history): identify exact duplicates on (user_id, movie_id, event_ts, device_type); create watch_history_dedup keeping one 'best' record (progress_ratio, then minutes_watched).
3. Outliers: compute IQR bounds & % outliers; create watch_history_robust with minutes_watched capped to P01/P99; report quantiles before/after.
4. Anomaly flags: flag_binge (session > 8h), flag_age_extreme (<10 or >100 if parseable), flag_duration_anomaly (duration <15m or >8h if available).

Verify before moving on

- Tables/views compile and return expected counts/percentages.
- Before/after comparisons show reduced duplicates and stabilized quantiles.

Reflection (write 2–4 sentences in your notebook)

- Which DQ issue was most pronounced? How might it affect churn/engagement models?
- Which flags/indicators would you keep as features and why?

Step 6: Save & Submit

Why this matters

- Reproducibility enables peer review, grading, and future reuse.
- Documentation connects decisions to business/ML impact.

What you will do

1. Save the executed notebook to your team Drive.
2. Export your DQ SQL into a single dq_cleaning.sql file.
3. Push notebook + SQL to your team GitHub repo with a descriptive commit message.
4. Add a README noting PROJECT_ID, REGION, bucket name, dataset, and today’s row counts.

Verify before moving on

- Repo shows notebook (.ipynb), dq_cleaning.sql, and README with metadata.

Reflection (write 2–4 sentences in your notebook)

- In 5–7 sentences, summarize key DQ findings, your cleaning policies, and expected ML impact.

Troubleshooting (Fast Fixes)

- Kaggle token error: regenerate token in Kaggle > Account; ensure permissions are 0600.
- Bucket creation fails: pick a different random suffix; bucket names are global.
- BQ load errors: confirm CSV headers and that you used --skip_leading_rows=1 and --autodetect.
- Permissions: ensure your IAM role allows GCS/BQ operations (e.g., Storage Object Admin, BigQuery Data Editor).

Grading Rubric (Quick)

- Profiling completeness (30)
- Cleaning policy correctness & reproducibility (40)
- Reflection/insight (20)
- Hygiene (naming, verification, idempotence) (10)

Tip: Keep your prompts and verification outputs in the notebook for auditability.