Universidad San Francisco de Quito

Data Mining

Proyecto 04

---

1) Resumen

Levantarás **Docker Compose** con **Jupyter+Spark**, **Postgres** y un **servicio "obt-builder"**. Ingerirás **todos los Parquet 2015–2025 (Yellow & Green)** del dataset **NYC TLC** hacia **Postgres** en el esquema **`raw`**, con estas tablas **obligatorias**:

- `raw.yellow_taxi_trip`
- `raw.green_taxi_trip`
- `raw.taxi_zone_lookup`

Luego, ejecutarás el servicio **`obt-builder`** para correr tu **script CLI** que crea/actualiza la **One Big Table**: **`analytics.obt_trips`** (1 fila = 1 viaje).
Finalmente, entrenarás y evaluarás **regresión** para predecir **`total_amount`** con **implementaciones propias (NumPy)** de **SGD, Lasso, Ridge, Elastic Net** (con *Polynomial Features*) y las **compararás** contra sus equivalentes en **scikit-learn**.

---

2) Objetivos de aprendizaje

1. Operar **Spark en Jupyter** para **ingesta masiva** (2015–2025) a **Postgres**.
2. Construir una **OBT** reproducible en **`analytics.obt_trips`** desde `raw.*`.
3. Implementar **desde cero** modelos lineales regularizados y **compararlos** con **scikit-learn**.
4. Diseñar un **pipeline reproducible** (variables de ambiente, seeds, CLI) evaluable por el profesor con **un solo comando**.
5. Seleccionar el **mejor modelo** con métricas sólidas y diagnóstico de errores.

---

3) Alcance y restricciones

- **Fuente**: NYC TLC en **Parquet** (Yellow/Green) 2015–2025.
- **Destino**: **Postgres** con esquemas:
  - **`raw`**: `raw.yellow_taxi_trip`, `raw.green_taxi_trip`, `raw.taxi_zone_lookup` (con metadatos de ingesta).
  - **`analytics`**: **`analytics.obt_trips`** (OBT, 1 fila = 1 viaje).
- **Procesamiento**: Spark (ingesta/backfill) → Postgres; **obt-builder** (script CLI) para construir OBT.
- **Seguridad**: **variables de ambiente** para **todas** las credenciales y parámetros.
- **ML obligatorio**:
  - **From-scratch (NumPy)**: **SGD**, **Ridge**, **Lasso**, **Elastic Net** (+ *Polynomial Features*).
  - **scikit-learn:** los mismos 4 modelos (+ Polynomial Features) con el mismo preprocesamiento y el mismo split.

---

4) Arquitectura esperada (alto nivel)

- **spark-notebook**: Notebooks de ingesta (Parquet→`raw.*`) y exploración.
- **postgres**: almacena `raw.*` y `analytics.obt_trips`.
- **obt-builder**: contenedor "worker" que corre **tu script CLI** para crear/actualizar `analytics.obt_trips` **leyendo** `raw.*` (2015–2025).
- *(Opcional)* **pgAdmin**: UI web para validar tablas/esquemas.

---

5) Seguridad y variables de ambiente (obligatorio)

- Entregar **`.env.example`** (sin secretos) y usar **`.env`** local.
- Variables:
  - **Postgres**: `PG_HOST` (nombre del servicio en Compose), `PG_PORT`, `PG_DB`, `PG_USER`, `PG_PASSWORD`, `PG_SCHEMA_RAW=raw`, `PG_SCHEMA_ANALYTICS=analytics`.
  - **Ingesta**: rutas/URL de Parquet, `YEARS`, `SERVICES`, `RUN_ID`.
  - **obt-builder**: parámetros de ejecución (modo, rango de años/meses).
- **Prohibido** hardcodear credenciales en notebooks o scripts.

---

6) Dataset y OBT (recordatorio)

- **Yellow**: `tpep_pickup_datetime`, `tpep_dropoff_datetime`, `PULocationID`, `DOLocationID`, `passenger_count`, `trip_distance`, tarifas itemizadas, `payment_type`, `RatecodeID`, `VendorID`.
- **Green**: análogo (`lpep_*_datetime`) + **`trip_type`** (1 street-hail, 2 dispatch).
- **Taxi Zone Lookup**: mapea `LocationID` → `zone` / `borough`.
- **`service_type`**: derivado del origen (yellow/green).
- **`analytics.obt_trips`** (mínimos):
    - **Tiempo**: `pickup_datetime`, `dropoff_datetime`, `pickup_hour`, `pickup_dow`, `month`, `year`.
    - **Ubicación**: `pu_location_id`, `pu_zone`, `pu_borough`, `do_location_id`, `do_zone`, `do_borough`.
    - **Servicio/Códigos**: `service_type`, `vendor_id`/`vendor_name`, `rate_code_id`/`rate_code_desc`, `payment_type`/`payment_type_desc`, `trip_type` (green).
    - **Viaje/Montos**: `passenger_count`, `trip_distance`, `fare_amount`, `extra`, `mta_tax`, `tip_amount`, `tolls_amount`, `improvement_surcharge`, `congestion_surcharge`, `airport_fee`, `total_amount`, `store_and_fwd_flag`.
    - **Derivadas**: `trip_duration_min`, `avg_speed_mph`, `tip_pct`.
    - **Metadatos**: `run_id`, `source_year`, `source_month`, `ingested_at_utc`.

---

7) ¿Qué es OBT? (síntesis)

One Big Table = tabla **desnormalizada** que minimiza JOINs para acelerar análisis.
Ventajas: simplicidad, rapidez; limitaciones: duplicación y *rebuild* por partición (mitigar con proceso **by-partition** y documentación de supuestos).

---

8) Infra con Docker Compose y servicio obt-builder

8.1 Servicios requeridos:

- **spark-notebook**: Jupyter+Spark (puertos expuestos, volumen para notebooks).
- **postgres**: base de datos con esquemas `raw` y `analytics`.

- **obt-builder**: ejecuta **tu script CLI** para construir **`analytics.obt_trips`** desde `raw.*`.
    - **Entrada**: credenciales Postgres por variables; **args** (modo, años, meses, RUN_ID, overwrite).
    - **Salida**: OBT creada/actualizada; **logs** con conteos por partición y tiempos.

  Requisito de evaluación: el profesor ejecutará **un solo comando** (p. ej., `docker compose run obt-builder --full-rebuild`) y tu **OBT** debe **construirse** end-to-end desde `raw.*` (2015–2025).

8.2 Especificación del script CLI (sin código)

- **Nombre sugerido**: `build_obt.py`
- **Argumentos**:
    - `--mode {full, by-partition}`
    - `--year-start 2015 --year-end 2025` (+ `--months` opcional)
    - `--services yellow,green`
    - `--run-id <string>`
    - `--overwrite {true,false}`
- **Comportamiento**:
    - **FULL**: (re)crea `analytics.obt_trips` completo (2015–2025).
    - **BY-PARTITION**: procesa solo particiones faltantes/seleccionadas.
    - **Idempotencia**: no duplicar (merge por clave natural o por partición).
    - **Evidencias**: imprimir conteos por año/mes, duración y *summary* final.

(Opcional) **pgAdmin** como tercer servicio para validar `raw` y `analytics`.

---

9) Notebooks obligatorios

9.1 Ingesta (Spark → Postgres RAW)

`01_ingesta_parquet_raw.ipynb`

- Backfill **2015–2025** (Yellow/Green) **mes a mes**.
- Estandariza timestamps y tipos mínimos.
- Escribe en **`raw.yellow_taxi_trip`**, **`raw.green_taxi_trip`**, **`raw.taxi_zone_lookup`** con metadatos y conteos por lote.

9.2 Construcción OBT (ejecutada por obt-builder)

Script CLI (Sección 8.2) crea/actualiza **`analytics.obt_trips`** (unificación + join a zonas + derivadas + metadatos).

9.3 ML Notebook — *from-scratch vs scikit-learn*

Archivo: `ml_total_amount_regression.ipynb`
Meta: predecir **`total_amount`** en pickup (sin *leakage*).

Estructura requerida (sin código):

1. **Problema y target**; decisiones que habilita; evitar leakage (no usar `dropoff_*` ni derivadas post-viaje).
2. **Carga de datos** desde Postgres (o *export* previo).
3. **EDA breve**: distribución del target, cardinalidad categórica, nulos/outliers.
4. **Features** (solo disponibles en pickup):
    - Numéricas: `trip_distance`, `passenger_count`, `pickup_hour`, `pickup_dow`, `month`, `year`, *flags* (hora pico/fin de semana).
    - Categóricas (Top-K + "Other"): `service_type`, `vendor`, `rate_code_desc`, `pu_borough`, `pu_zone` (controlar cardinalidad).
5. **Split temporal**: Train (años viejos), **Validación** (intermedio), **Test** (reciente) — justificar.
6. **Preprocesamiento común**: imputación en ausentes, **escalado** (obligatorio para L1/L2/SGD), OHE, **PolynomialFeatures**.
7. **Modelos FROM-SCRATCH (NumPy)**:
    - **SGD** (MSE; `alpha`, `learning_rate`, `max_iter`).
    - **Ridge** (L2).
    - **Lasso** (L1).
    - **Elastic Net** (L1+L2).
    - **Tuning**: GridSearch: registro de hiperparámetros y tiempos.

8. **Modelos scikit-learn equivalentes**:
    - **SGDRegressor**, **Ridge**, **Lasso**, **ElasticNet**, con **el mismo** preprocesamiento y el **mismo** split.
    - **Grid/Random Search** comparable al from-scratch.
9. **Comparación** (obligatoria):
    - Tabla **RMSE/MAE/R²** en **validación** y **test** para **los 8 pipelines** (4 propios + 4 sklearn).
    - **Tiempos** de entrenamiento y recuento de coeficientes.
    - Discusión: estabilidad, sensibilidad a `alpha`/`l1_ratio`/`eta`, sesgo-varianza.
10. **Selección final**:
    - Elegir **1 ganador** (menor RMSE en validación + simplicidad), reentrenar en Train+Val y evaluar en Test.

11. **Diagnóstico**: errores en la predicción.
12. **Conclusiones**: operatividad, reentrenamiento.

---

10) Métricas y evaluación

- **Primarias**: **RMSE** y **MAE** (validación y test).
- **Secundaria**: **R^2**.
- **Baseline**: media/mediana o lineal simple (sin regularización).
- **Reglas**: mismas *features*, *scaler*, *poly*, *split* y **seed** entre from-scratch y sklearn.

---

11) Entregables (GitHub)

1. **`docker-compose.yml`** con **spark-notebook**, **postgres** y **obt-builder** (todas las credenciales desde `.env`).
2. **Script CLI** `build_obt.py` (o equivalente) que crea/actualiza **`analytics.obt_trips`** **leyendo** `raw.*` (2015–2025).
3. **Notebooks**:
   - `01_ingesta_parquet_raw.ipynb`
   - `ml_total_amount_regression.ipynb` (from-scratch vs sklearn)
4. **README**:
   - Cómo levantar Compose, **ingestar RAW**, **construir OBT** (comando exacto que yo ejecutaré) y correr el notebook.
   - Variables de ambiente requeridas.
   - Modo **full** y **by-partition** de `build_obt.py`.
5. **Evidencias**:
   - Logs/capturas de **obt-builder** (conteos por año/mes, tiempos).
   - Tabla comparativa de **todos** los modelos (propios y sklearn) con métricas.
   - Gráficos de residuales/errores por *bucket*.

---

12) Rúbrica (100 pts) — basada en error + cumplimiento

A. Pipeline de datos en Postgres (25 pts)

- (10) **RAW** 2015–2025 (`raw.*`) cargado con metadatos.
- (10) **OBT** creada por **obt-builder** (comando reproducible, logs, conteos).
- (5) **Idempotencia** / ejecución **by-partition** sin duplicados en el OBT script.

B. Modelado from-scratch (25 pts)

- (10) Implementaciones propias de **SGD, Ridge, Lasso, Elastic Net** operativas.
- (5) *Polynomial Features* + escalado aplicados correctamente.
- (10) **Tuning** y tabla de resultados en **validación** (RMSE/MAE/R^2).

C. Comparación con scikit-learn (15 pts)

- (10) Pipelines sklearn **equivalentes** al from-scratch.
- (5) Comparativa clara (métricas + tiempos + discusión).

D. Selección, test y diagnóstico (10 pts)

- (5) Elección del **mejor** por validación y evaluación en **test**.
- (5) Análisis de residuales y **errores por buckets**.

E. Reproducibilidad & documentación (10 pts)

- (5) `.env`, comandos claros, seeds fijas; **obt-builder** ejecutable por el profesor.
- (5) README y evidencias completas.

F. Puntaje por RMSE (15 pts) — **ranking entre estudiantes**

Se mide sobre el **Test** del **modelo ganador** de cada estudiante (target `total_amount`).

- **15 pts** si `RMSE_test ≤ P50` de la clase.
- **10 pts** si `P50 < RMSE_test ≤ P90`.
- **6 pts** si `P90 < RMSE_test < RMSE_baseline`.
- **0 pts** si `RMSE_test ≥ RMSE_baseline` o hay **leakage**.

Penalizaciones

- *Leakage* (uso de `dropoff_*` u otras señales post-viaje): **−25 pts** y anulación del puntaje por RMSE.
- Falta de comparativa **from-scratch o sklearn**: **−35 pts**.
- `build_obt.py` **no** crea la OBT end-to-end: **−50 pts**.

---

13) Checklist de aceptación

- **RAW** en Postgres: `raw.yellow_taxi_trip`, `raw.green_taxi_trip`, `raw.taxi_zone_lookup` (2015–2025).
- **OBT** `analytics.obt_trips` creada por **obt-builder** (comando reproducible, logs).

- **ML**: 4 modelos **from-scratch** + 4 **sklearn** (mismo preprocesamiento y split).
- **Comparativa**: tabla RMSE/MAE/R^2 (validación y test) + tiempos.
- **Diagnóstico**: residuales y errores por buckets.
- **README**: comandos de ingesta, **creación OBT** (comando que yo ejecutaré), ejecución notebook, variables `.env`.
- Seeds fijas; resultados reproducibles.

---

14) Recomendaciones finales

- Mantén **paridad estricta** entre from-scratch y sklearn (mismas *features*, *scaler*, *poly*, *split*, **seed**).
- Limita **cardinalidad** (Top-K + “Other”) en categorías; *Polynomial* solo en 2–3 numéricas clave.
- Explora **alpha** y **l1_ratio** de forma pragmática (grillas pequeñas, comparables).
- Considera **log-transform** del target y reporta RMSE en escala original.