Clase 206 — Testing de datos: Great Expectations, Deequ

Parte: 4 — MLOps · Fuente: Huyen cap. 4 + Great Expectations docs (1.x) + Deequ paper.

Duración estimada: 75 min.

Clase 206 — Testing de datos: Great Expectations, Deequ

> *Parte: 4 — MLOps · Fuente: Huyen cap. 4 + Great Expectations docs (1.x) + Deequ paper. Duración estimada: 75 min.*

Objetivo

Aplicar testing como código a los datos: definir "expectations" (assertions sobre el dataset) y validarlas en cada corrida del pipeline. Detectar schema drift, outliers extremos, nulos inesperados antes de que lleguen al entrenamiento o a la predicción. Bug típico: no es que el modelo se rompe — es que la data está rota y nadie se entera.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Definir un Expectation Suite con Great Expectations 1.x: schema, ranges, uniqueness, null rates, regex.
- Generar un Data Docs HTML que documenta el dataset + resultados de validación (auditoría para regulador).
- Integrar GE en un pipeline DVC/Airflow: si validación falla, abortar el pipeline.
- Usar Deequ (Scala/Python via PyDeequ) para datasets grandes en Spark.
- Diferenciar data tests (sobre el dataset) de unit tests (sobre el código) — son ortogonales.

Temas

#	Tema	Por qué importa
1	Por qué unit tests no alcanzan en data pip	El código está OK; la data llega rota.
2	Expectation Suite: schema + business rules	Catalogar lo que es "data correcta" en est
3	Profiling automático	GE puede inferir un suite inicial del data
4	Data Docs: docs ejecutables	Auditoría + onboarding sin esfuerzo extra.
5	Checkpoints e integración con pipelines	El gate que aborta el flow si la data está
6	Deequ para Spark scale	Cuando el dataset no entra en pandas.

Definiciones y características

- Expectation: assertion sobre los datos. Ej: expect_column_values_to_not_be_null('user_id'), expect_column_values_to_be_between('age', 0, 120), expect_column_distinct_values_to_be_in_set('country', ['AR', 'BR', 'CL']).
- Expectation Suite: colección de expectations versionada (YAML/JSON).
- Validation: ejecutar el suite contra un batch real → result con success: true/false por expectation.
- Checkpoint: configuración que une data + suite + acción (postear a Slack, abortar pipeline, generar docs).
- Data Docs: HTML auto-generado que muestra: schema, ejemplos, resultados de validación, evolución.
- Profiling: GE puede generar un suite inicial inspeccionando el dataset (UserConfigurableProfiler). Punto de partida — siempre revisar manualmente.

- Deequ: librería de Amazon (Scala, también PyDeequ para Python) que hace lo mismo pero sobre Spark. Diseñada para TB.
- Pandera: alternativa Python-first, integrada con pandas/polars DataFrames con sintaxis decorator.

Dataset / recursos

- Dataset: California Housing (sin shift) como reference, después con shift sintético para ver fallas.
- Librerías: great-expectations>=1.0, opcional pandera, pydeequ.

Ejercicios

1. Bootstrap de un suite: gx init para crear el proyecto. gx datasource new apuntando a CSV. Profileá el dataset para suite inicial. Revisalo a mano: descomentá las expectations razonables, borrá las absurdas.
2. Custom expectations: agregá: expect_column_values_to_be_between('MedInc', 0, 20), expect_table_row_count_to_be_between(1000, 100000), expect_column_pair_values_A_to_be_greater_than_B('AveRooms', 'AveBedrms') (más rooms que bedrooms).
3. Checkpoint: configurá un checkpoint que (a) corre el suite, (b) si falla, abre un Slack alert. Corré con gx checkpoint run my_checkpoint.
4. Data Docs: gx docs build. Abrí el HTML. Mostrá a un PM/regulador hipotético: la suite + el último validation result.
5. Pandera alternative: el mismo schema con pandera: class HousingSchema(pa.DataFrameModel): MedInc: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 20}). Compará verbosidad y velocidad.

Homework verificable

Pipeline ML con:

1. Expectation Suite Great Expectations versionada en git (YAML/JSON).
2. Checkpoint que se corre como step del pipeline DVC o Airflow (Clase 203) antes del training.
3. Data Docs HTML buildeado en CI y publicado a GitHub Pages.
4. Slack alert (con webhook stub si no tenés real) cuando una expectation falla.
5. Demonstración de un dataset alterado (NaN inyectados, outliers, schema cambiado) que el pipeline detecta y aborta, sin que el modelo se entrene con data sucia.

Criterio de aceptación: el pipeline corre clean con data buena. Inyectando NaN en MedInc: el checkpoint falla, el pipeline aborta, el Slack alerta dispara, el modelo NO se re-entrena con data corrupta.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
gx init no encuentra Python venv correcto	GE 1.x quiere su propio gx CLI. Fix: activ
expect_column_values_to_be_in_set falla po	Tu suite quedó vieja. Fix: actualizar el s
Validation pasa pero el modelo igual rinde	Validación de forma ≠ validación de distri
Profiler genera 200 expectations, todas se	El profiler es muy permisivo. Fix: empezar
El Data Docs no muestra el último validati	data_docs no se rebuildeó. Fix: agregar up

Performance lento en 10M filas	GE sobre pandas escanea todo. Fix: para vo

Preguntas frecuentes

¿Great Expectations, Pandera o Deequ?

- GE: más completo (Data Docs, suite versioning, plugin ecosystem), pero verboso.
- Pandera: Python-first, sintaxis decorator, mejor DX para devs. No tiene Data Docs.
- Deequ / PyDeequ: para Spark, datasets grandes.

Para equipos chicos en pandas: Pandera. Para empresas con auditoría: GE. Para Spark: Deequ.

¿Cuándo data testing vs schema validation (Pydantic)?

- Pydantic / dataclasses: validación por fila / record (en API requests). Rápido, in-process.
- GE / Pandera: validación por dataset (batch). Tests sobre agregados (medias, conteos), no solo schema.

Son complementarios: API valida cada request con Pydantic; pipeline batch valida cada slice con GE.

¿Cómo manejo expectations que cambian con el tiempo (estacionalidad)?

(1) Expectations relativas: "row count entre last_week × 0.8 y last_week × 1.2". (2) Múltiples suites por época. (3) Marcar la expectation como warning (no aborta) y revisar manualmente.

¿GE rompe mi pipeline cada deploy?

Sí, si tu suite es muy estricto y la data cambia legítimamente. Fix: separar expectations críticas (abortan: PII no debe ser null, schema no debe cambiar) de alertas (warning, no abortan: distribución shift).

¿Versionar el Expectation Suite en git?

Sí siempre. Cambios al suite van por PR como cambios de código — review, CI, merge. El suite es contrato de datos = parte del contrato del producto.

¿Y datos de producción real-time (streaming)?

GE no es ideal para streaming. Alternativas: Apache Griffin, validations custom en el consumer Kafka, o aceptar que el testing es batch (validar cada 1 h sobre el último window).

Referencias

- Huyen, Chip. Designing Machine Learning Systems (O'Reilly, 2022), cap. 4 — Training Data.
- Great Expectations docs — 1.x es la versión actual (rompió compat con 0.x).
- Pandera docs — schema validation for pandas/polars.
- PyDeequ — Spark-scale data testing.
- Schelter et al. Automating large-scale data quality verification (VLDB 2018) — paper de Deequ.

Siguiente clase

Clase 207 — Testing de modelos: invariance, behavioral tests

Apéndice: notebook (primer bloque)

Definimos un suite de expectations sobre California Housing y validamos con (a) Great Expectations 1.x, (b) Pandera, (c) Polars + checks ad-hoc. Inyectamos un bug a propósito para ver el sistema detectarlo.

```
import pandas as pd, numpy as np
from sklearn.datasets import fetch_california_housing

data = fetch_california_housing(as_frame=True)
df = data.data.copy()
df['target'] = data.target
print(df.shape, '\n', df.describe().round(2).T[['min', 'max', 'mean']])
```

Archivos complementarios

- notebook.ipynb