Facsímil 08

La ciencia de los datos

Capítulo 02

Calidad de datos: schema, duplicados, leakage y etiquetas

686f6c61 · 2026

Capítulo 02: Calidad de datos: schema, duplicados, leakage y etiquetas

Entrando en el tema

En datos para IA hay una frase que conviene tatuarse antes de abrir cualquier notebook: **un dataset no falla solo cuando el programa explota**. También falla cuando una etiqueta significa dos cosas distintas, cuando una fila de test se parece demasiado a una de train, cuando una licencia permite evaluar pero no entrenar, cuando una columna nueva entra sin contrato o cuando una cola de revisión crece hasta que nadie sabe qué versión de la verdad está usando el modelo.

La calidad de datos no es una limpieza estética. Es una forma de proteger decisiones. Si el capítulo anterior construía la pregunta “¿de dónde sale este dataset y para qué puedo usarlo?”, este capítulo construye la siguiente: “¿qué pruebas mínimas debe pasar antes de que yo lo use para entrenar, evaluar, recuperar o publicar una conclusión?”.

La respuesta no puede ser una tabla mágica ni una confianza ciega en una herramienta. Tiene que combinar contrato, métricas, revisión humana, evidencia reproducible y criterio de ingeniería. Por eso vamos a hablar de schema, duplicados, leakage y etiquetas como piezas de un mismo sistema: el sistema que evita que una métrica bonita se apoye en datos rotos.

Qué deberías poder hacer al terminar

En el capítulo anterior construimos una idea base: un dataset serio no empieza en el modelo, empieza en el contrato. Ahora damos un paso más. Un contrato escrito no sirve de mucho si no se ejecuta. La calidad de datos aparece justo ahí: en convertir el contrato en comprobaciones que puedan aprobar, revisar o bloquear un dataset antes de que llegue al modelo, al RAG o a una evaluación.

Calidad no significa “datos bonitos”. Significa **datos suficientemente correctos para la decisión que queremos tomar**. Un dataset puede ser aceptable para un prototipo, insuficiente para una evaluación pública, útil para RAG interno y prohibido para entrenamiento. La calidad siempre se lee junto al uso.

Al terminar deberías poder hacer esto:

RESULTADO DE APRENDIZAJE	EVIDENCIA DE QUE LO SABES HACER
Distinguir schema, expectation y contrato.	No confundes “la columna existe” con “el dato es válido”.
Detectar duplicados exactos y cercanos.	Sabes mirar claves repetidas, textos normalizados y similitud.
Explicar leakage con ejemplos de datos.	Puedes detectar cuándo test se contaminó con train o cuándo una feature mira el futuro.
Revisar etiquetas con criterio.	No borras ejemplos difíciles; los llevas a una cola de revisión.
Calcular acuerdo entre anotadores.	Entiendes p_o, p_e y κ.
Diseñar un gate de calidad.	Produces un reporte con `pass`, `review` o `block`.
Conectar calidad con evaluación.	Ves que una métrica puede caer por modelo o por dataset, y sabes separarlo.

La lectura importante es que no basta con detectar fallos: hay que saber qué significan. Un nulo en una columna auxiliar puede abrir revisión; un nulo en `source_id` puede impedir auditar una cita; una etiqueta fuera de catálogo puede romper entrenamiento; un duplicado cruzando train y test puede invalidar una evaluación. El mismo síntoma cambia de gravedad según el uso que vaya a tener el dataset.

La idea central es esta:

> La calidad de datos no se inspecciona al final. Se prueba antes de decidir.

La escena: el modelo parece peor, pero el dataset cambió

Imagina que tenemos una eval de soporte académico. Durante semanas el asistente funciona razonablemente bien. Un día, tras cambiar algunos datos, la métrica baja. El primer impulso es pensar que el modelo ha empeorado o que el prompt ya no sirve.

Pero al mirar el dataset aparecen señales menos espectaculares y más importantes: una etiqueta nueva llamada `resolve` se coló en un catálogo donde solo existían `answer`, `ask_more` y `escalate`; un caso de `train` aparece casi igual en `test`; un campo `source_id` está vacío; una fila de validación lleva licencia de entrenamiento; y dos anotadores no se ponen de acuerdo en varios ejemplos.

Nada de eso requiere cambiar el modelo. Requiere parar la cadena y preguntar: **¿este dataset puede sostener la decisión que estamos tomando?** Si no puede, la métrica deja de ser una medida limpia. Se convierte en una mezcla de comportamiento del modelo, fallos de contrato, errores de etiqueta y contaminación entre splits.

Qué no es calidad de datos

Calidad de datos no es “no tener nulos”. Los nulos importan, pero son solo una parte pequeña. Un dataset puede no tener ningún valor vacío y aun así estar mal: etiquetas inconsistentes, licencia incompatible, duplicados entre splits, texto caducado, clases desbalanceadas o columnas con significado ambiguo.

Tampoco es “pasar un script de limpieza”. Limpiar sin contrato puede empeorar el problema. Si borras todos los casos difíciles porque ensucian la métrica, quizá destruyes justo los ejemplos que el sistema necesita aprender o evaluar. Si deduplicas sin mirar splits, puedes eliminar evidencia legítima o dejar contaminación.

Y no es una puntuación universal. La misma tabla puede estar bien para explorar, mal para entrenar y completamente inaceptable para evaluar. Por eso la pregunta correcta no es “¿tiene calidad?”, sino “¿tiene calidad suficiente para este uso, con este contrato y esta fecha de corte?”.

MALENTENDIDO	LECTURA DE INGENIERÍA
“Si no hay nulos, está limpio”.	Faltan catálogo, licencias, duplicados, leakage, etiquetas y linaje.
“Un duplicado siempre se borra”.	Primero hay que saber si es repetición real, evento legítimo o contaminación.
“La etiqueta registrada es verdad”.	La etiqueta es una decisión humana o automática que puede fallar.
“Si la métrica baja, el modelo empeoró”.	Puede haber cambiado el dataset, el split o la distribución.
“Calidad es cosa de datos, no de ingeniería”.	Un gate de calidad es parte del pipeline de software.

Qué sí es calidad de datos para IA

En este facsímil llamaremos calidad de datos al grado en que un dataset cumple las condiciones necesarias para una decisión de IA: entrenar, evaluar, indexar, recuperar, monitorizar o revisar.

Es una decisión de ingeniería basada en contrato. Para que un dataset avance, el equipo debe poder contestar estas preguntas con evidencias, no con intuiciones:

PIEZA DEL CONTRATO	PREGUNTA QUE DEBE CONTESTAR	EVIDENCIA MÍNIMA
Schema	¿La forma del dataset es la esperada?	Columnas obligatorias, tipos y columnas extra.
Valores	¿Los campos respetan catálogos y rangos?	Etiquetas permitidas, productos válidos, nulos medidos.
Linaje	¿Sabemos de dónde viene cada fila?	`source_id`, fecha, owner, hash y versión.
Licencia	¿El uso previsto está permitido?	Permisos por split: entrenar, evaluar, recuperar o revisar.
Splits	¿La evaluación está separada de aprendizaje y ajuste?	Train, validation y test sin contaminación evidente.
Etiquetas	¿Las referencias tienen criterio estable?	Acuerdo entre anotadores, política de etiqueta y cola de revisión.

La lectura práctica es simple: si una pieza crítica falla, el dataset no debería avanzar como si nada. No hace falta inventar una ecuación para verlo. Hace falta un contrato, checks ejecutables y una decisión trazable.

Ese contrato también protege al equipo de un sesgo muy humano: arreglar lo que se ve fácil y dejar intacto lo que duele. Es cómodo corregir formatos o borrar nulos; es más incómodo revisar etiquetas, licencias, duplicados semánticos o leakage temporal. Pero en IA, muchas métricas falsas nacen precisamente en esas zonas incómodas. Por eso el gate debe guardar evidencia, no solo un estado final.

Dimensiones de calidad: no todo fallo es del mismo tipo

Una forma madura de hablar de calidad de datos es separar dimensiones. Wang y Strong propusieron una idea que sigue siendo útil: la calidad no se define solo desde la tabla, sino desde quien consume el dato y la tarea que necesita resolver.[1] Para IA esto es especialmente importante, porque el mismo campo puede ser correcto para analítica y peligroso para evaluación.

Imagina una columna `created_at`. Si está vacía, falla completitud. Si tiene texto donde esperamos una fecha ISO, falla validez. Si dice que el caso se creó después de su resolución, falla consistencia. Si llega tres días tarde al pipeline, falla actualidad. Y si el campo existe

pero no estaba disponible en el momento de decisión, puede introducir leakage temporal. Es la misma columna, pero no el mismo problema.

DIMENSIÓN	QUÉ PREGUNTA HACE	EJEMPLO EN IA
Completitud	¿Falta algo que el contrato exige?	`source_id` vacío impide auditar una respuesta RAG.
Validez	¿El valor respeta tipo, formato, rango o catálogo?	`label=resolve` no está en el conjunto permitido.
Consistencia	¿Dos piezas de información se contradicen?	`split=test` con licencia solo permitida para entrenamiento.
Unicidad	¿La misma entidad aparece más veces de las permitidas?	`case_id=q004` repetido.
Actualidad	¿El dato está vigente para la fecha de corte?	Documento académico actualizado después de la consulta evaluada.
Representatividad	¿Cubre los casos donde vamos a decidir?	Casi no hay ejemplos de `escalate` en test.
Trazabilidad	¿Podemos reconstruir origen, versión y transformación?	Falta `source_id`, hash o política de anotación.
Adecuación al uso	¿Sirve para entrenar, evaluar, indexar o decidir?	Una muestra exploratoria no debería usarse como benchmark.

Esta tabla evita un error común: decir “calidad baja” y quedarse ahí. En ingeniería necesitamos saber qué dimensión falla, qué uso afecta, si bloquea o abre revisión y qué evidencia queda para repetir la decisión.

Fecha de corte del estado del arte

Fecha de corte: 6 de junio de 2026.
Fuentes consultadas: Wang y Strong sobre dimensiones de calidad, TensorFlow Data Validation, Great Expectations, Pandera, Deequ, Soda, Cleanlab, Snorkel, Evidently, scikit-learn, Jaccard, Levenshtein, Broder, Shannon, Manning y otros y literatura clásica sobre acuerdo entre anotadores.

La idea estable es que la validación de datos debe ser declarativa, reproducible y cercana al pipeline. TensorFlow Data Validation perfila datasets, infiere schema y detecta anomalías contra expectativas.[2] Su referencia de anomalías muestra que un sistema de validación no

solo dice “hay error”, sino que clasifica problemas como tipo inválido, dominio inesperado, valor ausente o cambio de distribución.[3]

Great Expectations organiza la calidad como expectations: afirmaciones verificables sobre columnas, rangos, nulos, formatos o relaciones entre datos.[4] Pandera lleva esa misma filosofía al mundo de DataFrames, con schemas programáticos para validar tipos y propiedades de columnas.[5] Deequ propone “unit tests for data” sobre Spark, una frase muy útil porque acerca la calidad de datos a una práctica que cualquier ingeniero de software reconoce.[6]

Para etiquetas, confident learning propone estimar incertidumbre y errores de etiqueta en datasets, y Cleanlab lo popularizó como herramienta práctica.[7] El trabajo sobre errores de etiqueta en test sets mostró algo incómodo: incluso benchmarks muy usados pueden contener errores de etiqueta suficientes para desestabilizar conclusiones.[8]

Snorkel es importante porque enseña otra vía: crear datos de entrenamiento con supervisión débil mediante funciones de etiquetado, y después combinar señales ruidosas de forma sistemática.[9] La lección para este capítulo no es “usa una herramienta concreta”. Es más básica: si una etiqueta puede fallar, la etiqueta también necesita ingeniería.

Las tripas de la calidad: de schema a decisión

La calidad empieza con schema, pero no termina ahí. El schema responde a la forma: qué columnas existen, qué tipos tienen, qué valores son válidos. Después vienen reglas de contenido: nulos, catálogos, rangos, licencias, duplicados, distribución de clases, coherencia entre anotadores y leakage entre splits.

En el cuaderno del facsímil, cada expectation se trata como un check con tres campos: nombre, severidad y evidencia. No estamos demostrando un teorema; estamos construyendo una puerta de release para datos.

CAMPO DEL CHECK	QUÉ GUARDA	EJEMPLO
Nombre	Qué regla se evalúa.	`label_values`.
Severidad	Qué pasa si falla.	`block` o `review`.
Evidencia	Qué casos explican el fallo.	Filas con `label=resolve`.

El gate del cuaderno agrupa esos checks y les asigna una acción. En otro equipo podrían cambiar los nombres o los umbrales, pero la idea operativa es la misma: no todo fallo pesa igual.

La tabla concreta cuándo devuelve cada estado:

RESULTADO	QUÉ SIGNIFICA	QUÉ HARÍA UN EQUIPO
`block`	Hay fallos que impiden usar el dataset.	No entrenar, no evaluar, no indexar ese snapshot.
`review`	No hay bloqueo técnico, pero hay dudas que requieren criterio.	Abrir cola de revisión y documentar decisión.
`pass`	Cumple el contrato actual para el uso declarado.	Usarlo y guardar reporte/versionado.

La parte importante es la palabra “actual”. Un `pass` no significa que el dataset sea perfecto para siempre. Significa que, bajo este contrato y este uso, no hemos encontrado fallos que impidan avanzar.

Esta distinción evita dos extremos igual de malos. El primero es bloquear todo por miedo, hasta que ningún dataset sirve para aprender. El segundo es aprobar todo porque “ya corregiremos después”. Un gate profesional no pretende purificar los datos; pretende tomar una decisión proporcional: bloquear lo que rompe la validez, revisar lo que necesita criterio humano y aprobar lo que cumple para el uso declarado.

Missing y valores fuera de catálogo

El missing se calcula aquí como proporción muestral de indicadores de ausencia sobre las celdas obligatorias. Es una forma directa de resumir datos faltantes; la literatura clásica de missing data insiste en que, además de la tasa, importa el mecanismo que produjo esos huecos.[10]

$$miss(D) = \frac{\sum_{i=1}^{n} \sum_{c \in C} 1[x_{i,c} = \varnothing]}{n \cdot |C|}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
n	Número de filas.	19 casos.
C	Columnas obligatorias.	`case_id`, `source_id`, `label`, `text`, etc.
$x_{i,c}$	Valor de la columna c en la fila i.	`source_id` de `q007`.
1	Indicador: 1 si falta, 0 si no.	Vale 1 si `source_id` está vacío.
$miss(D)$	Tasa total de missing.	0.003096 en el cuaderno.

Un missing bajo puede seguir siendo grave. Si falta una columna secundaria quizá es revisable; si falta `source_id`, perdemos linaje. Por eso la tasa agregada no basta: hay que mirar qué columna falta y para qué uso era necesaria.

Los valores fuera de catálogo son otra familia de errores. Si `label` solo permite `answer`, `ask_more` y `escalate`, una etiqueta `resolve` no es una variación inocente: rompe la semántica del dataset. Quizá representa una nueva clase legítima, pero entonces el contrato debe cambiar y la evaluación debe rehacerse.

Duplicados exactos y duplicados cercanos

Un duplicado exacto puede detectarse normalizando texto:

PASO	QUÉ HACE	EJEMPLO
Normalizar	Minúsculas, quitar signos y compactar espacios.	`Pago duplicado!!!` pasa a `pago duplicado`.
Comparar	Mira si dos textos normalizados coinciden.	`q001` y `q017`.
Decidir	Si cruzan splits, el fallo puede bloquear evaluación.	Duplicado en train y test.

Pero muchos duplicados no son idénticos. Cambian una palabra o añaden un adjetivo. Para enseñar la idea sin dependencias externas, el cuaderno usa Jaccard sobre conjuntos de tokens. La similitud de Jaccard es una medida clásica basada en intersección y unión de conjuntos.[11]

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
A	Tokens del primer texto.	`consulta`, `documentacion`, `beca`.
B	Tokens del segundo texto.	`consulta`, `documentacion`, `beca`, `necesaria`.
$\text{card}(A \cap B)$	Tokens comunes.	Palabras compartidas.
$\text{card}(A \cup B)$	Tokens totales únicos.	Palabras distintas entre ambos textos.
$J(A, B)$	Similitud de Jaccard.	0.833333 en un par del cuaderno.

Jaccard no entiende semántica profunda. No sabe que “matrícula” y “inscripción” pueden estar relacionadas. Pero es suficiente para enseñar una idea crítica: si un caso muy parecido aparece en train y test, la evaluación deja de ser limpia.

Distancia de edición: cuando cambia una palabra, una letra o un formato

Jaccard mira conjuntos de tokens. Eso es útil para frases parecidas, pero no siempre detecta errores de escritura, identificadores casi iguales o pequeñas variaciones de formato. Para ese caso existe una medida clásica: la distancia de edición de Levenshtein, que calcula cuántas operaciones mínimas hacen falta para convertir una cadena en otra.[1][2]

La definición habitual se escribe como programación dinámica:

$$d_{i,j} = \min \begin{cases} d_{i-1,j} + 1 \\ d_{i,j-1} + 1 \\ d_{i-1,j-1} + 1[a_i \neq b_j] \end{cases}$$

SÍMBOLO	SIGNIFICADO	LECTURA PRÁCTICA
a_i	Carácter i de la primera cadena.	Una letra del primer texto normalizado.
b_j	Carácter j de la segunda cadena.	Una letra del segundo texto normalizado.
$d_{i,j}$	Coste mínimo hasta esas posiciones.	Cuántas ediciones llevamos acumuladas.
$+1$	Insertar o borrar.	Falta o sobra un carácter.
$1[a_i \neq b_j]$	Sustituir si los caracteres difieren.	Cambiar una letra por otra.

En un dataset de soporte, Levenshtein ayuda a detectar `doc-mat-001` frente a `doc-mat-OO1`, o “matricula ordinaria” frente a “matrícula ordinaria” tras normalizar acentos. No sustituye a Jaccard ni a embeddings. Es otra lente. La regla de ingeniería es combinar lentes según el daño: claves y códigos pueden necesitar distancia de edición; textos largos pueden necesitar n-gramas, Jaccard o embeddings; documentos completos pueden necesitar hashes, similitud semántica y revisión humana.

Duplicados a escala: blocking, MinHash y candidatos

En un CSV de 19 filas puedes comparar todos los pares. En un dataset real, no. El número de pares posibles crece como:

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

Con $n = 1\,000\,000$, eso son casi quinientos mil millones de comparaciones. Aunque cada comparación fuera barata, el planteamiento ya nació mal. Por eso los sistemas de calidad de datos no suelen buscar duplicados cercanos comparando todo contra todo. Primero reducen candidatos.

La estrategia clásica tiene tres capas:

CAPA	QUÉ HACE	EJEMPLO
Blocking	Agrupa por una clave barata.	Mismo producto, misma fuente, mismo prefijo de documento.
Fingerprinting	Resume el texto o documento en una firma compacta.	Hashes de shingles o tokens normalizados.
Scoring	Calcula similitud solo entre candidatos.	Jaccard, Levenshtein, embeddings o revisión humana.

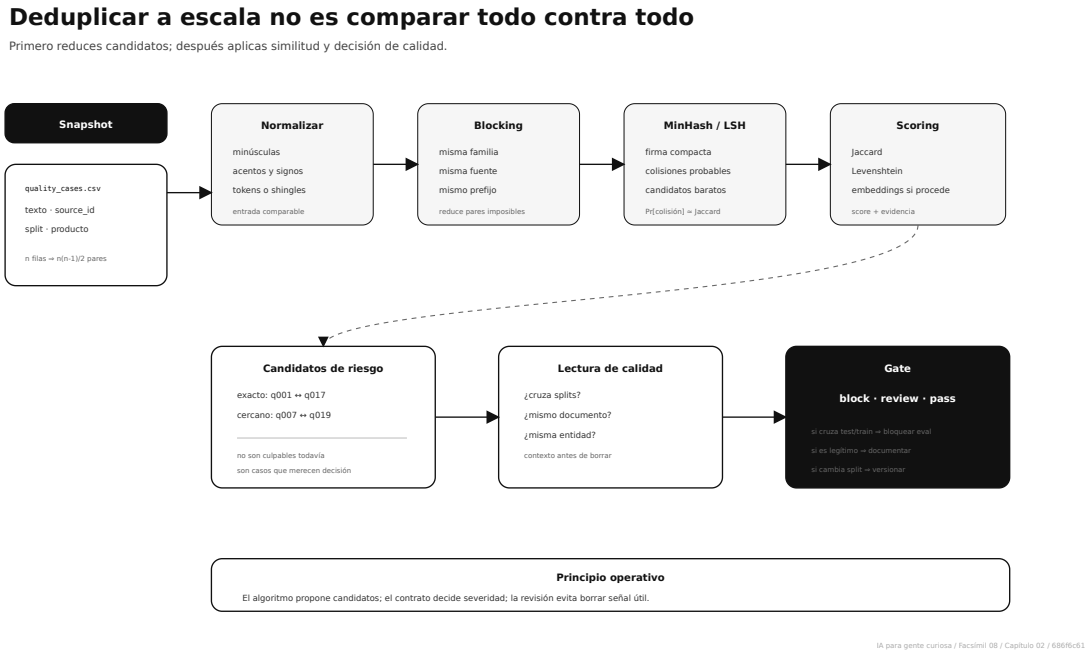
Broder formalizó una idea que sigue siendo central en deduplicación documental: estimar la semejanza de conjuntos mediante muestreo/fingerprints en lugar de comparar documentos completos.[13] En MinHash, si aplicamos una permutación aleatoria π y nos quedamos con el elemento mínimo de cada conjunto, se cumple:

$$\Pr[\min(\pi(A)) = \min(\pi(B))] = J(A, B)$$

SÍMBOLO	SIGNIFICADO	LECTURA DE INGENIERÍA
A, B	Conjuntos de tokens, shingles o fingerprints.	Dos textos o documentos normalizados.
π	Permutación aleatoria o función hash que la aproxima.	Una forma reproducible de reordenar elementos.
$\min(\pi(A))$	Valor mínimo de la firma para A.	Un componente de la firma MinHash.
$\Pr[\cdot]$	Probabilidad de colisión entre firmas.	Si colisionan mucho, probablemente se parecen.
$J(A, B)$	Similitud de Jaccard real.	La cantidad que queremos estimar sin comparar todo.

Esto no significa que MinHash “pruebe” que dos ejemplos son duplicados. Significa que ayuda a proponer candidatos con coste manejable. Después el gate debe mirar severidad: si el candidato cruza train y test, puede bloquear evaluación; si está dentro del mismo split, quizá

se revisa; si es un documento casi contenido en otro, puede afectar a RAG o a entrenamiento, no necesariamente a lo mismo.



> Para datasets grandes, deduplicar exige reducir candidatos antes de calcular similitudes caras. MinHash ayuda a escalar la búsqueda; el gate decide qué hacer con cada candidato.

Leakage entre splits

Leakage no es solo duplicado textual. Es cualquier información que permite al sistema obtener ventaja en evaluación porque ya vio directa o indirectamente la respuesta.

En el cuaderno del facsímil usamos un procedimiento auditable: comparar ejemplos de test contra train con coincidencia exacta y Jaccard, revisar los candidatos y bloquear si cruzan splits por encima del umbral escrito en el contrato.

EVIDENCIA DE LEAKAGE	CÓMO SE DETECTA	QUÉ SIGNIFICA
Texto exacto cruzado	Coincidencia después de normalizar.	Test contiene algo ya visto por train.
Texto muy parecido	Jaccard alto entre tokens.	El caso puede estar midiendo memoria de plantilla.
Misma entidad	`student_id`, `document_id` o `source_id` cruzan splits.	El sistema puede reconocer contexto repetido.
Tiempo mal ordenado	Train contiene información posterior a test.	El modelo aprende futuro.

En el próximo capítulo entraremos más a fondo en splits y leakage. Aquí basta con una regla operativa: si algo de test está demasiado cerca de train, no uses ese test para decidir si el sistema generaliza.

La palabra “cerca” es deliberadamente incómoda. A veces la cercanía es exacta: misma fila, mismo texto, mismo `case_id`. Otras veces es semántica: dos consultas redactadas distinto pero con la misma solución, dos chunks del mismo documento, dos tickets de la misma persona o una feature que resume el futuro. Un buen gate no se queda en igualdad de strings. Produce candidatos, muestra evidencia y obliga a una revisión cuando el riesgo de contaminación afecta a la evaluación.

En LLMs aparece una variante especialmente incómoda: contaminación de benchmarks. Un modelo puede haber visto durante entrenamiento parte de un benchmark, sus soluciones o textos muy próximos. Sainz y otros insisten en que la contaminación debe medirse por benchmark y no suponerse resuelta de forma genérica.[14] Para un ingeniero de datos, la traducción es concreta: guarda hashes, snapshots, fechas de corte, fuentes de benchmark y reglas de exclusión. Si no puedes explicar qué datos no pudieron entrar en entrenamiento, tu evaluación tiene un agujero.

Distribución de etiquetas

Una etiqueta no es solo un texto. Es una decisión que influye en el aprendizaje o en la evaluación. Por eso conviene mirar proporciones. Lo que escribimos abajo es la frecuencia relativa muestral de una clase: una estimación empírica de la distribución de etiquetas.[15]

$$p(y = k) = \frac{n_k}{n}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
y	Etiqueta.	`answer`, `ask_more`, `escalate`.
k	Clase concreta.	`ask_more`.
n_k	Número de ejemplos de esa clase.	4 casos.
n	Número total de ejemplos.	19 casos.

Si casi todo es `answer`, el sistema puede aprender a contestar siempre. Si casi no hay `escalate`, puede parecer correcto hasta que aparece un caso crítico. En clasificación y evaluación, mirar la distribución por split es obligatorio.

En un proyecto real, esta revisión debería hacerse antes de mirar el modelo. Si una clase crítica aparece dos veces en test, cualquier porcentaje será frágil. Si una clase domina train, un baseline perezoso puede parecer competente. Y si la distribución cambia entre train, validation y test, quizá el problema no sea el algoritmo sino la muestra. La distribución de etiquetas no decide sola, pero sí te dice si la evaluación tiene suelo suficiente para sostener una conclusión.

Otra forma académica de medir concentración de etiquetas es la entropía de Shannon.[16] Si Y es la variable etiqueta:

$$H(Y) = -\sum_{k=1}^{K} p_k \log p_k$$

SÍMBOLO	SIGNIFICADO	LECTURA PRÁCTICA
K	Número de clases posibles.	`answer`, `ask_more`, `escalate`.
p_k	Proporción de la clase k.	Frecuencia relativa de `escalate`.
$H(Y)$	Entropía de la distribución de etiquetas.	Baja si casi todo cae en una clase.

La entropía no dice si las etiquetas son correctas. Solo mide concentración. Si $H(Y)$ cae mucho entre versiones, puede haber cambio real de demanda, sesgo de muestreo, error de pipeline o limpieza que borró casos difíciles. En ingeniería de datos no se usa como veredicto; se usa como señal para preguntar mejor.

Para comparar la distribución global con la de cada split podemos usar distancia total variation, una medida estándar de distancia entre distribuciones de probabilidad.[17]

$$D_{TV}(P, Q) = \frac{1}{2} \sum_i |P_i - Q_i|$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
P_i	Proporción global de la clase i.	Proporción global de `answer`.
Q_i	Proporción de la clase i en un split.	Proporción de `answer` en test.
D_{TV}	Distancia entre distribuciones.	0.172932 para test en el cuaderno.

scikit-learn permite hacer splits estratificados con `train_test_split(..., stratify=y)` cuando esa estrategia encaja con el problema.[18] Estratificar no arregla etiquetas malas, pero ayuda a que cada partición conserve una mezcla parecida de clases.

Calidad de etiquetas y acuerdo entre anotadores

Las etiquetas son datos, no dogmas. En el cuaderno aparecen tres señales de revisión: desacuerdo entre anotadores, diferencia con una referencia didáctica y baja confianza de un modelo auxiliar. En un proyecto real no siempre tendrás `expected_label`, pero sí puedes tener doble anotación, reglas de revisión, muestras auditadas o modelos que prioricen casos dudosos.

El acuerdo observado se calcula así:

$$p_o = \frac{\sum_{i=1}^{n} 1[a_i = b_i]}{n}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
a_i	Etiqueta del anotador A para el ejemplo i.	`answer`.
b_i	Etiqueta del anotador B para el ejemplo i.	`ask_more`.
p_o	Proporción de acuerdo observado.	0.789474 en el cuaderno.

Pero parte del acuerdo puede ocurrir por azar, especialmente si una clase domina. Kappa de Cohen corrige ese efecto:[19]

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
p_o	Acuerdo observado.	0.789474.
p_e	Acuerdo esperado por azar según distribuciones marginales.	0.518006.
κ	Acuerdo corregido por azar.	0.563218.

En el cuaderno, κ queda por debajo del umbral `0.6`, así que no bloquea por sí solo, pero exige revisión. Esta distinción importa: una etiqueta dudosa no siempre se borra; se revisa con política.

Más de dos anotadores: Fleiss y Krippendorff

Cohen encaja cuando hay dos anotadores. En proyectos de datos para IA no siempre es así. Puedes tener tres revisores por caso, un adjudicador, revisiones parciales o anotadores distintos según el producto. En ese contexto, usar la fórmula de dos anotadores por comodidad puede dar una falsa sensación de rigor.

Para etiquetas nominales con varios anotadores por ítem, una referencia clásica es kappa de Fleiss.[20] Su forma agregada mantiene la misma intuición que Cohen: compara acuerdo observado contra acuerdo esperado por azar.

$$\kappa_F = \frac{\bar{P} - \bar{P}_e}{1 - \bar{P}_e}$$

Donde, si cada ejemplo recibe m anotaciones y n_{ij} es cuántos anotadores asignaron la clase j al ejemplo i:

$$P_i = \frac{1}{m(m-1)} \sum_{j=1}^{k} n_{ij}(n_{ij} - 1)$$

$$\bar{P} = \frac{1}{N} \sum_{i=1}^{N} P_i$$

$$\bar{P}_e = \sum_{j=1}^{k} p_j^2$$

SÍMBOLO	SIGNIFICADO
N	Número de ejemplos anotados.
m	Número de anotaciones por ejemplo.
k	Número de clases posibles.
n_{ij}	Anotadores que eligieron la clase j para el ejemplo i.
P_i	Acuerdo entre anotadores en el ejemplo i.
$\bar{P}$	Acuerdo medio observado.
$\bar{P}_e$	Acuerdo esperado por azar según prevalencia de clases.

Krippendorff propuso otra familia de coeficientes especialmente útil cuando hay anotaciones incompletas, distintos niveles de medida o datos de contenido.[21] Su intuición se resume así:

$$\alpha = 1 - \frac{D_o}{D_e}$$

SÍMBOLO	SIGNIFICADO
D_o	Desacuerdo observado.
D_e	Desacuerdo esperado por azar.
α	Fiabilidad corregida por azar.

En este capítulo no vamos a calcular Fleiss ni Krippendorff en el cuaderno, porque el ejercicio trae dos anotadores para que Cohen sea transparente. Pero sí conviene que un ingeniero de datos lo tenga en la cabeza: si tu proceso real tiene más de dos personas, anotaciones incompletas o escalas ordinales, la métrica de acuerdo también forma parte del contrato. No vale decir “hay consenso” sin definir cómo se midió.

Medir la cola de revisión: precisión y exhaustividad

Cuando una herramienta marca posibles errores de etiqueta, no conviene creerla como si fuera una autoridad. Conviene evaluarla como evaluaríamos un detector. La pregunta no es solo “¿ha encontrado casos raros?”, sino “¿cuántos de los casos que marcó eran errores reales?” y “¿cuántos errores reales se le escaparon?”. En recuperación de información y clasificación, estas preguntas se expresan con precisión y exhaustividad, normalmente llamada *recall* en documentación técnica.[22]

$$precision = \frac{TP}{TP + FP}$$

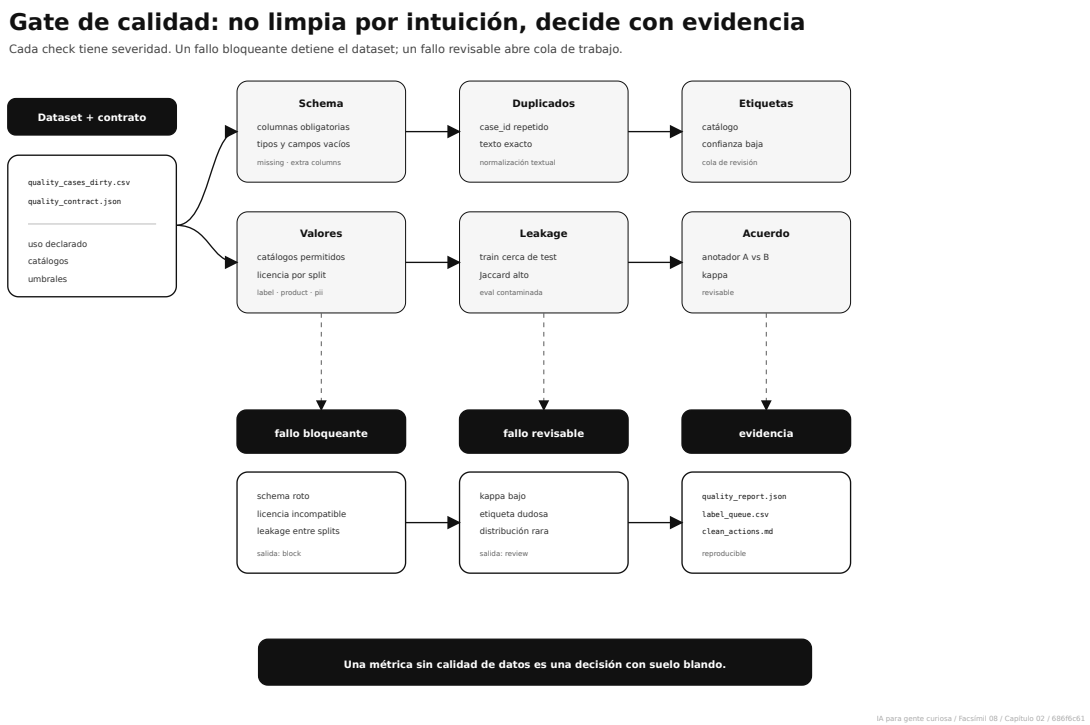
$$recall = \frac{TP}{TP + FN}$$

SÍMBOLO	SIGNIFICADO EN REVISIÓN DE ETIQUETAS
TP	Casos marcados como dudosos que, tras revisión humana, eran errores reales.
FP	Casos marcados como dudosos que estaban bien etiquetados.
FN	Casos que eran errores reales, pero el detector no propuso revisar.

Esto baja la herramienta a tierra. Si la cola tiene mucha precisión pero poco recall, revisas pocos casos y casi todos aportan valor, pero se te escapan errores. Si tiene mucho recall y poca precisión, atrapas casi todo, pero saturas al equipo con falsos positivos. En un dataset pequeño de evaluación, quizá prefieres recall alto porque cada etiqueta pesa mucho. En entrenamiento masivo, quizá priorizas precisión para no convertir la revisión en una fábrica imposible.

El cuaderno no pretende resolver automáticamente este equilibrio. Lo enseña de forma controlada: `label_review_queue.csv` no es una lista de culpables, es una bandeja de entrada para aplicar la política de anotación, registrar decisiones y medir si la heurística de revisión está ayudando o solo generando ruido.

Anatomía de un gate de calidad



Un gate de calidad separa fallos bloqueantes, revisiones necesarias y evidencia generada. No sustituye el criterio humano: lo organiza.

Severidad: cuándo bloquear, cuándo revisar y cuándo aceptar deuda

Un gate útil no trata todos los fallos igual. Si una etiqueta dudosa y una licencia incompatible aparecen en la misma lista sin severidad, el equipo pierde criterio. La severidad existe para responder a una pregunta muy concreta: **¿qué daño produce avanzar con este fallo?**

En ingeniería de IA conviene separar tres niveles. Un fallo bloqueante invalida el uso automático del dataset. Un fallo revisable no permite publicar una decisión todavía, pero sí permite abrir trabajo humano. Una deuda aceptada es un problema conocido que no afecta al uso actual o que tiene un plan explícito de corrección.

NIVEL	PREGUNTA DE DECISIÓN	EJEMPLO	ACCIÓN
`block`	¿Puede invalidar entrenamiento, evaluación, permiso o seguridad del uso?	Leakage entre `train` y `test`; licencia incompatible; etiqueta fuera de catálogo.	Detener uso automatizado.
`review`	¿Necesita criterio humano antes de confiar en la decisión?	Kappa bajo; baja confianza de etiqueta; distribución rara.	Abrir cola de revisión.
`accepted_debt`	¿Se puede vivir con esto durante una ventana concreta?	Missing en un campo no usado por este modelo.	Documentar owner, fecha y plan.

Esta tabla evita dos extremos. El primero es bloquearlo todo y convertir calidad en un cuello de botella imposible. El segundo es dejar pasar todo porque “ya lo arreglaremos”. La severidad obliga a escribir el riesgo y la acción. Si no puedes explicar por qué algo es `block` o `review`, quizá la regla todavía no está bien formulada.

La matriz también debe depender del uso. Un `source_id` vacío puede ser revisable para exploración local, pero bloqueante para evaluación trazable. Una etiqueta dudosa puede ser tolerable en entrenamiento con millones de ejemplos, pero bloqueante en un test pequeño que decide entre dos modelos. La severidad no vive aislada: vive con el contrato.

SLI, SLO y presupuesto de error de datos

En operación ya vimos que un SLI es un indicador medible, un SLO es un objetivo y el presupuesto de error dice cuánto margen queda antes de actuar. Esa idea encaja muy bien con calidad de datos.

Un **SLI de datos** mide una propiedad concreta del dataset. Un **SLO de datos** fija el umbral aceptable. Un **presupuesto de error de datos** define qué ocurre cuando nos acercamos al límite o lo superamos. No es una frase de marketing: es una herramienta para que el equipo sepa cuándo parar.

SLI DE DATOS	QUÉ MIDE	SLO POSIBLE	ACCIÓN SI FALLA
`critical_missing_rate`	Missing en columnas críticas.	`<= 0.5%`	Bloquear snapshot si afecta linaje o etiqueta.
`cross_split_duplicates`	Duplicados entre splits.	`0`	Bloquear evaluación.
`invalid_catalog_values`	Valores fuera de catálogo.	`0`	Bloquear hasta migrar contrato o datos.
`license_mismatches`	Uso incompatible con licencia.	`0`	Bloquear uso automatizado.
`annotator_kappa`	Acuerdo corregido por azar.	`>= 0.6`	Abrir revisión de política de anotación.
`label_review_queue_size`	Casos pendientes de revisar.	`<= 2`	No cerrar release si supera el umbral.

En el cuaderno del facsímil, estos objetivos aparecen en `contracts/quality_slos.json`. Es importante verlo así: no es una ruta para buscar dentro del repositorio, sino un archivo que puedes abrir, modificar y versionar como harías en un proyecto real.

El presupuesto de error evita discusiones vagas. Por ejemplo: si el SLO exige cero duplicados entre train y test, un solo duplicado consume todo el presupuesto y bloquea. Si el SLO permite hasta dos casos en cola de revisión, un tercero no significa que el dataset sea inútil, pero sí que la release no debería cerrarse sin mirar esos casos.

Ciclo de vida de una incidencia de datos

Detectar un fallo es el principio, no el final. Una incidencia de datos debería tener un ciclo de vida tan claro como una incidencia de software:

1. Detectar: el gate marca un check fallido.
2. Clasificar: se decide severidad, owner y uso afectado.
3. Diagnosticar: se busca causa raíz.
4. Corregir: se cambia dato, contrato, pipeline o política.
5. Reejecutar: se vuelve a correr el gate.
6. Versionar: se guarda snapshot, reporte y decisión.
7. Aprender: se añade un check para que no vuelva a pasar igual.

La causa raíz importa porque no todos los fallos se arreglan tocando el CSV. Si aparece una etiqueta `resolve`, puede ser un error de anotación, pero también una señal de que el dominio necesita una nueva clase. Si aparece `admisiones` en `product`, puede ser un campo mal escrito, o puede ser que el producto haya crecido y el contrato esté viejo. Si baja kappa, quizá los anotadores fallaron, pero quizá la política era ambigua.

SÍNTOMA	CAUSA RAÍZ POSIBLE	ARREGLO POBRE	ARREGLO DE INGENIERÍA
Etiqueta nueva	Catálogo real cambió.	Reemplazarla por la clase más parecida.	Versionar contrato y migrar evaluación.
Duplicado entre splits	Split creado después de deduplicar mal.	Borrar una fila sin registro.	Rehacer split desde snapshot limpio.
Kappa bajo	Política de anotación ambigua.	Pedir “más cuidado”.	Añadir casos frontera y regla de desempate.
Licencia incompatible	Export mezcló permisos.	Ignorar en prototipo.	Separar usos permitidos por split.
Missing en linaje	Pipeline perdió metadatos.	Rellenar a mano.	Corregir ingestión y reejecutar.

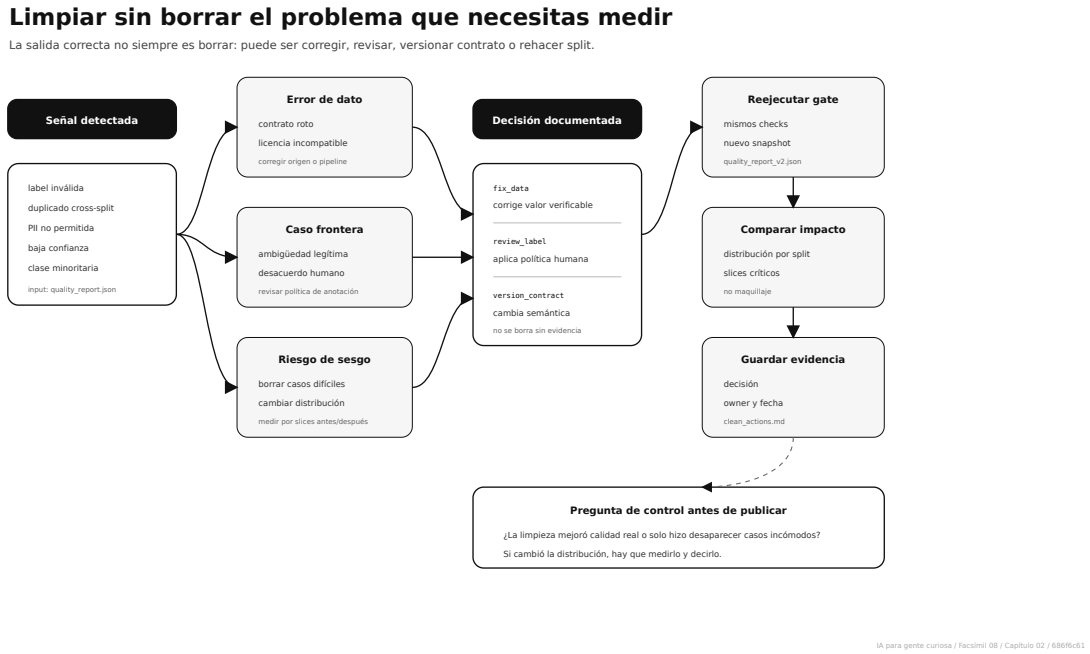
Esta forma de pensar es muy de ingeniería: no arreglar solo el síntoma, sino el mecanismo que permitió que apareciera.

Limpiar sin destruir señal

Una limpieza mala suele tener buena intención. Alguien mira el dataset, ve casos difíciles y piensa: “si quito esto, la métrica sube”. A veces sube. El problema es que quizá ya no mide el problema real. En IA, limpiar no puede significar borrar todo lo que molesta. Tiene que significar separar errores de datos, casos frontera legítimos y ejemplos difíciles que el sistema necesita aprender o evaluar.

La diferencia es importante. Una fila con `label=resolve` puede ser un error de catálogo. Una pregunta ambigua sobre becas puede ser un caso legítimo de `ask_more`. Un pago duplicado puede ser una incidencia real que debe escalarse, no un outlier que conviene eliminar. Si el equipo borra sin política, convierte la calidad en maquillaje.

CASO	QUÉ NO HACER	QUÉ HACER
Etiqueta inválida	Mapearla a mano a la clase más frecuente.	Abrir incidencia: ¿error de anotación o nueva clase?
Duplicado entre splits	Borrar el que “moleste menos”.	Rehacer split desde snapshot deduplicado y guardar manifiesto.
Baja confianza	Eliminar todos los casos de baja confianza.	Revisar muestra, medir precisión/recall de la cola y ajustar política.
Clase minoritaria	Balancear borrando clases frecuentes sin criterio.	Diseñar muestreo, pesos o evaluación por slices.
PII alta	Enmascarar sin registrar transformación.	Aplicar política de privacidad, dejar linaje y revalidar.



Limpiar bien no consiste en borrar filas molestas. Consiste en clasificar el problema, aplicar una acción trazable, reejecutar el gate y comprobar que no hemos sesgado el dataset.

Schema evolution: cuando el dataset cambia sin pedir permiso

Los datasets vivos cambian. Aparecen columnas, se renombran campos, se añaden etiquetas, se deprecan productos y cambian tipos. Ese cambio no es malo. Lo peligroso es que ocurra sin versión, sin migración y sin compatibilidad.

Schema evolution significa gestionar cambios de forma controlada. En un dataset de IA, no basta con que el código lea la nueva columna. Hay que saber si el significado cambió. Una columna `label` puede mantener el mismo nombre y cambiar por completo si el equipo añade una política nueva. Un valor `resolve` puede ser un error hoy y una clase legítima mañana.

CAMBIO	RIESGO	ESTRATEGIA
Nueva columna opcional	Los consumidores antiguos la ignoran.	Añadirla como `optional` y documentar fecha.
Columna obligatoria nueva	Pipelines antiguos fallan.	Crear versión `v2` del contrato y periodo de convivencia.
Nuevo valor de catálogo	Métricas antiguas no comparan igual.	Versionar label set y rehacer splits/evals.
Cambio de tipo	Validación y transformación pueden romper.	Migración explícita con tests de datos.
Cambio de significado	El peor caso: parece compatible, pero no lo es.	Nota de contrato, ejemplos y revisión humana.

Una regla útil: si un cambio altera una decisión, no es un detalle técnico. Es una nueva versión del contrato.

Política de anotación: la etiqueta también necesita contrato

Una etiqueta como `answer` parece sencilla hasta que aparecen casos frontera. ¿Qué hacemos si falta un dato menor? ¿Y si el usuario mezcla beca y matrícula? ¿Y si el asistente podría responder, pero con riesgo de orientar mal? Sin política, cada anotador inventa su criterio.

El cuaderno del facsímil incluye una política mínima en `contracts/annotation_policy.md`. Ese archivo es deliberadamente pequeño, pero tiene una función seria: convertir desacuerdos humanos en reglas revisables, no en conversaciones de pasillo.

Una política de anotación profesional debería incluir:

PIEZA	POR QUÉ IMPORTA
Definición de cada etiqueta	Evita que dos personas usen palabras iguales con criterios distintos.
Ejemplos positivos	Muestran cuándo sí aplica la etiqueta.
Ejemplos negativos	Muestran cuándo no aplica aunque se parezca.
Casos frontera	Reducen desacuerdo donde el dominio es ambiguo.
Regla de desempate	Permite cerrar conflictos sin improvisar.
Versión de política	Hace trazable qué criterio produjo cada etiqueta.

Esto conecta directamente con kappa. Si el acuerdo es bajo, no siempre significa que los anotadores sean malos. Puede significar que la política no explica los casos difíciles. En ese caso, el siguiente paso no es reetiquetar a ciegas: es mejorar la política y después revisar.

Calidad por segmentos

Una media global puede esconder un problema local. El dataset puede tener buen missing global, pero fallar en `becas`; buen kappa global, pero mucho desacuerdo en `chat`; buen reparto de etiquetas, pero ningún caso crítico en `test`.

Por eso la calidad debería mirarse por segmentos:

SEGMENTO	PREGUNTA
`product`	¿Todos los temas tienen cobertura y etiquetas coherentes?
`channel`	¿Chat, email y portal tienen estilos comparables?
`split`	¿Train, validation y test mantienen distribuciones razonables?
`pii_risk`	¿Los casos sensibles tienen controles suficientes?
`language`	¿La calidad se mantiene si hay varios idiomas?
`criticality`	¿Los casos de alto impacto están representados y revisados?

Esta idea preparará el capítulo de slices. La intuición es sencilla: un dataset no se rompe siempre “en general”. Muchas veces se rompe en un rincón. La ingeniería consiste en encontrar ese rincón antes de que lo encuentre producción.

Leakage temporal

Hasta ahora hemos hablado de leakage entre splits. Hay otro tipo especialmente traicionero: leakage temporal. Ocurre cuando el dataset usa información que no existía en el momento en que el sistema habría tenido que decidir.

Ejemplos:

CASO	LEAKAGE TEMPORAL
Predicción de urgencia	Usar `days_to_resolution`, que solo se conoce después de cerrar el caso.
RAG académico	Evaluar con un documento actualizado después de la consulta original.
Feature de pagos	Usar un estado de pago corregido días después del evento.
Etiqueta de soporte	Etiquetar con información de resolución posterior y usarla como entrada.

La regla de oro es: para cada feature o documento, pregunta "¿esto estaba disponible en ese momento?". Si la respuesta es no, quizá sirve para análisis histórico, pero no para entrenar o evaluar una decisión que pretende simular producción.

En el día a día

En un equipo de IA, la calidad de datos debería estar cerca de CI. Igual que un cambio de código no debería desplegarse si rompe tests, un snapshot de datos no debería llegar a entrenamiento o evaluación si rompe checks críticos.

La práctica madura tiene cuatro capas:

CAPA	PREGUNTA	ARTEFACTO
Contrato	¿Qué prometen los datos?	JSON, YAML, ODCS, schema programático.
Validación	¿Cumplen lo prometido?	Reporte de checks y anomalías.
Revisión	¿Qué casos requieren criterio?	Cola de etiquetas, duplicados, incidencias.
Decisión	¿Qué hacemos con el snapshot?	`pass`, `review`, `block` y plan de limpieza.

La parte difícil no es escribir `if row["label"] not in allowed_labels`. Eso lo puede hacer cualquiera. La parte difícil es decidir qué fallos bloquean, cuáles abren revisión y cuáles se aceptan temporalmente con una nota. Ahí aparece la ingeniería: convertir incertidumbre en reglas operativas que un equipo pueda sostener.

En CI/CD, el patrón mínimo sería:

```
python3 ops/data_quality_gate.py --write
python3 ops/ci_assert_gate.py
```

Si `release_gate.json` contiene `block`, el segundo comando termina con error y el pipeline se detiene. Si el equipo permite releases en `review` para entornos manuales, podría ejecutarlo con:

```
python3 ops/ci_assert_gate.py --allow-review
```

La clave es que el pipeline no decide por intuición. Lee un artefacto. Ese artefacto queda guardado y puede revisarse después.

Herramientas por capa

Las herramientas importan, pero no por el logo. Importan porque obligan a convertir dudas en artefactos: expectations, schemas, reportes, anomalías, colas de revisión o alertas. Si una herramienta no deja evidencia revisable, en este capítulo no nos sirve demasiado.

CAPA	HERRAMIENTAS	QUÉ APORTAN	QUÉ NO DELEGARÍA
Validación declarativa	Great Expectations, Pandera, Deequ, Soda Core	Checks sobre columnas, nulos, rangos, catálogos, reglas de tabla y contratos.	Decidir severidad de negocio o si un fallo invalida una evaluación.
Validación ML pipeline	TensorFlow Data Validation	Schema, estadísticas, anomalías y comparación entre datasets dentro de pipelines TFX.	Entender por sí sola si una etiqueta es pedagógicamente correcta.
Datos y etiquetas	Cleanlab, Snorkel	Detección/priorización de errores de etiqueta y creación de señales de supervisión débil.	Convertir una señal probabilística en verdad sin revisión.
Monitorización	Evidently, Soda, observabilidad de datos	Drift, estadísticas, calidad recurrente y comparación contra referencia.	Sustituir un contrato de datos escrito.
CI/CD de datos	Scripts propios, dbt tests, Airflow/Dagster assets	Ejecutar gates, guardar outputs y bloquear snapshots.	Arreglar automáticamente un fallo sin política.

Great Expectations trabaja con expectations sobre datos; Pandera permite expresar schemas de DataFrames en código; Deequ acerca la idea de tests de datos a Spark; Soda Core permite definir checks y contratos; TFDV perfila datasets y detecta anomalías; Cleanlab ayuda a encontrar posibles problemas de etiqueta; Snorkel estructura supervisión débil; Evidently se usa para evaluar y monitorizar cambios de datos y modelos.[23242526272829]

La decisión práctica sería esta: usa una herramienta declarativa para reglas obvias, una herramienta de perfilado para cambios de distribución, una herramienta de etiqueta para priorizar revisión y un gate propio para convertir todo eso en `pass`, `review` o `block`. El gate propio no tiene que ser grande. Tiene que ser explícito, versionado y entendible por el equipo.

Herramientas externas de mercado: cuándo tienen sentido

Fecha de corte de esta sección: 21 de junio de 2026. El mercado de data observability cambia deprisa, así que lo importante no es memorizar proveedores. Lo importante es entender qué problema operativo resuelve cada familia y qué evidencia debería producir.

Una herramienta externa tiene sentido cuando el problema ya no cabe en un script local: muchas tablas, muchos owners, varios warehouses, alertas recurrentes, incidentes con impacto, linaje que cruza equipos, necesidad de priorizar por negocio o auditoría de cambios entre entornos. Si solo tienes un CSV y un contrato pequeño, el cuaderno del facsímil es mejor profesor que una plataforma. Si tienes cien pipelines críticos, necesitas otra cosa.

HERRAMIENTA	DÓNDE ENCAJA	QUÉ DEBERÍAS PEDIRLE COMO EVIDENCIA	RIESGO SI LA USAS MAL
Monte Carlo	Observabilidad de datos y AI/data pipelines: monitores, alertas, impacto y resolución.	Incidente, tabla afectada, patrón anómalo, lineage/impacto y owner notificado.	Creer que una alerta sustituye al contrato semántico del dataset.
Bigeye	Observabilidad empresarial con reglas, anomalías, lineage, reconciliación e incidentes.	Métrica monitorizada, umbral o anomalía, histórico, impacto y flujo de resolución.	Comprar visibilidad sin definir qué bloquea una release de IA.
Anomalo	Monitorización automática y detección de anomalías en warehouses grandes.	Dato tardío, faltante o anómalo, explicación de root cause y prioridad.	Confundir anomalía estadística con error de negocio.
Datafold	Data diff: comparación de tablas, columnas, filas o migraciones.	Diferencias exactas entre fuente y destino, filas afectadas y columnas cambiadas.	Usarlo solo para migraciones y olvidar que un cambio semántico puede no verse como diff obvio.
Soda	Checks, contratos, observabilidad y workflow de calidad.	Check fallido, contrato afectado, owner y evolución del problema.	Llenar el sistema de checks sin severidad ni decisión.
Great Expectations Cloud/Core	Expectations declarativas y validación reproducible.	Suite de expectations, resultado por expectation y Data Docs/reporte.	Tener expectativas técnicas sin conectar con uso de IA.
Cleanlab	Priorización de problemas de etiqueta o datos.	Ranking de casos sospechosos y señales de incertidumbre.	Tratar un score como verdad sin revisión humana.
Evidently	Evals, drift, calidad y monitorización de modelos/datos.	Reporte comparando referencia contra producción o batch actual.	Mirar drift global y no revisar slices críticos.

Monte Carlo se presenta como una plataforma de observabilidad de datos y AI/data pipelines que aprende patrones y alerta sobre incidencias en warehouses, lakes, ETL y BI.[30] Bigeye describe una plataforma de data observability con lineage, anomalías, reglas, reconciliación e incident management.[31] Anomalo enfatiza monitorización automática con aprendizaje no supervisado para detectar datos tardíos, faltantes o anómalos a escala.[32] Datafold documenta `Data Diff` como comparación a nivel de valores entre tablas para detectar cambios críticos.[33]

La regla de compra para un ingeniero de datos sería sobria: antes de pagar una plataforma, escribe tres incidentes que quieres detectar, tres decisiones que quieres bloquear y tres evidencias que una persona revisora necesita leer. Si la herramienta no puede producir esas evidencias, no está resolviendo este capítulo; solo está poniendo una interfaz encima.

Por qué debería importarte

Si no revisas calidad, puedes entrenar con etiquetas rotas, evaluar con leakage, elegir un modelo por una métrica falsa o indexar documentos que no debían usarse. Todo eso produce sistemas que parecen funcionar hasta que llegan a producción o hasta que alguien intenta explicar una decisión.

Además, la calidad de datos es una de las formas más baratas de mejorar IA. Cambiar de modelo puede costar dinero, latencia y complejidad. Corregir una etiqueta mal puesta, separar bien splits o bloquear una licencia incompatible puede mejorar la decisión sin tocar arquitectura.

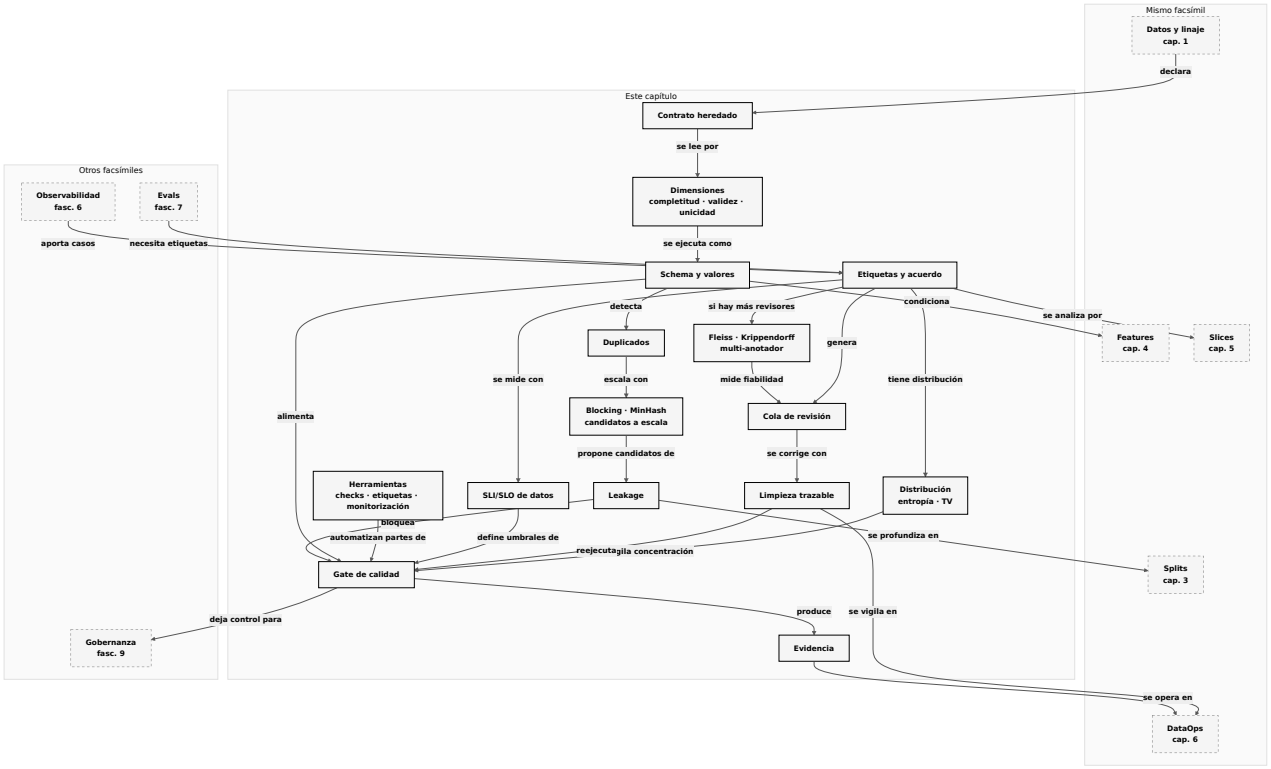
La calidad no compite con el modelado. Lo hace posible.

Dónde solía tropezar yo

TROPIEZO	POR QUÉ OCURRE	ANTÍDOTO
Mirar solo nulos	Es el check más fácil de programar.	Separar schema, catálogo, licencias, duplicados, leakage y etiquetas.
Borrar duplicados sin mirar splits	Parece una limpieza obvia.	Revisar si el duplicado contamina evaluación o representa un evento legítimo.
Tratar una etiqueta como verdad	La columna `label` da falsa seguridad.	Medir acuerdo, revisar baja confianza y documentar política de anotación.
Mezclar fallos bloqueantes y revisables	Todo se mete en una lista única.	Asignar severidad: `block` para lo que invalida uso, `review` para lo que pide criterio.
Celebrar un `pass` sin contexto	Suena a certificado universal.	Leerlo siempre junto a contrato, uso, fecha y versión del dataset.
Cambiar schema sin versión	El código sigue funcionando y parece compatible.	Versionar contrato si cambia una decisión o significado.
Mirar calidad solo global	Las medias esconden problemas locales.	Revisar por producto, canal, split, sensibilidad e idioma.
Borrar casos difíciles para subir la métrica	La limpieza se confunde con maquillaje.	Separar error, caso frontera y señal útil antes de eliminar.
Creer a la cola de revisión sin medirla	La herramienta suena objetiva.	Medir precisión/recall con una muestra revisada por personas.
Elegir herramienta antes de contrato	Es más cómodo instalar que pensar.	Definir uso, severidad y evidencia antes de automatizar.
Comparar todo contra todo	Funciona en una demo pequeña y se rompe a escala.	Usar blocking, fingerprints y candidatos antes del scoring caro.
Usar Cohen con cualquier proceso de anotación	Es la métrica que más suena.	Usar Cohen solo con dos anotadores; mirar Fleiss o Krippendorff cuando el proceso cambia.
Ignorar contaminación de benchmarks	El benchmark parece externo y limpio.	Guardar fechas de corte, hashes y reglas de exclusión.
Comprar observabilidad sin gate	La plataforma promete visibilidad.	Exigir evidencia, severidad y acción antes de integrarla en releases de IA.

Cómo encaja todo

El capítulo anterior nos dio contrato y linaje. Este capítulo ejecuta ese contrato. El siguiente capítulo se centrará en splits y leakage con más profundidad. Más adelante veremos features, slices, drift y análisis aplicado. La calidad de datos es el puente entre “tengo datos” y “puedo tomar una decisión con ellos”.



Vocabulario aprendido

TÉRMINO	DEFINICIÓN BREVE
Calidad de datos	Grado en que un dataset cumple el contrato necesario para una decisión concreta.
Schema	Forma esperada de los datos: columnas, tipos, catálogos y reglas.
Expectation	Regla verificable sobre el dataset.
Gate	Decisión automatizada de `pass`, `review` o `block`.
Duplicado exacto	Registro que repite clave o texto normalizado.
Duplicado cercano	Registro muy parecido según una medida de similitud.
Jaccard	Similitud entre conjuntos basada en intersección y unión.
Leakage	Información que contamina entrenamiento, evaluación o tiempo de decisión.
Etiqueta	Clase o salida esperada asociada a un ejemplo.
Ruido de etiqueta	Sospecha de que la etiqueta registrada no sigue la política.
Kappa	Acuerdo entre dos anotadores corregido por azar.
Cola de revisión	Lista priorizada de casos que necesitan criterio humano.
SLI de datos	Indicador medible de salud del dataset.
SLO de datos	Objetivo mínimo aceptable para un indicador de datos.
Presupuesto de error de datos	Margen de fallos tolerables antes de bloquear o revisar.
Schema evolution	Gestión versionada de cambios en columnas, tipos y catálogos.
Leakage temporal	Uso de información que no existía en el momento de decisión.
Dimensión de calidad	Aspecto evaluable de la calidad: completitud, validez, consistencia, unicidad, actualidad o trazabilidad.
Validez	Cumplimiento de tipos, formatos, rangos y catálogos permitidos.
Consistencia	Ausencia de contradicciones entre campos, tablas, fuentes o permisos.

TÉRMINO	DEFINICIÓN BREVE
Unicidad	Garantía de que una entidad o ejemplo no aparece repetido cuando el contrato lo prohíbe.
Distancia de edición	Medida de cuántos cambios hacen falta para convertir una cadena en otra.
Precisión de revisión	Qué proporción de casos marcados como sospechosos eran problemas reales.
Recall de revisión	Qué proporción de problemas reales logró detectar la cola de revisión.
Observabilidad de datos	Capacidad de detectar, explicar y trazar cambios de calidad, schema, volumen o distribución.
Data diff	Comparación de datos entre versiones, tablas o sistemas para detectar cambios de filas, columnas y valores.
Data observability	Monitorización operativa de frescura, volumen, schema, distribución, lineage, anomalías e impacto.
Blocking key	Clave barata que reduce candidatos antes de calcular similitudes más caras.
MinHash	Técnica probabilística para estimar Jaccard con firmas compactas.
LSH	Familia de técnicas para encontrar candidatos similares sin comparar todo contra todo.
Entropía de etiquetas	Medida de concentración o diversidad de la distribución de clases.
Kappa de Fleiss	Acuerdo nominal corregido por azar para más de dos anotadores.
Alfa de Krippendorff	Fiabilidad corregida por azar útil con distintos tipos de anotación y datos incompletos.

Antes de pasar página

Antes de avanzar, deberías poder responder:

1. ¿Por qué calidad de datos no significa solo “sin nulos”?
2. ¿Qué diferencia hay entre schema y expectation?
3. ¿Por qué un valor fuera de catálogo puede bloquear un dataset?
4. ¿Qué diferencia hay entre duplicado exacto y duplicado cercano?
5. ¿Cómo se calcula Jaccard y qué limitación tiene?
6. ¿Por qué un duplicado entre train y test puede contaminar una evaluación?

7. ¿Qué significa que una etiqueta vaya a cola de revisión?
8. ¿Por qué kappa corrige el acuerdo observado?
9. ¿Qué diferencia hay entre `block`, `review` y `pass`?
10. ¿Qué artefacto usarías para justificar una decisión de calidad ante otra persona?
11. ¿Qué SLI y SLO definirías para duplicados entre splits?
12. ¿Cuándo un cambio de schema exige nueva versión del contrato?
13. ¿Qué problema resuelve una política de anotación?
14. ¿Qué es leakage temporal y por qué no lo detecta siempre un deduplicador?
15. ¿Cómo conectarías el gate de datos a CI/CD?
16. ¿Qué diferencia hay entre completitud, validez, consistencia y unicidad?
17. ¿Cuándo usarías Jaccard y cuándo distancia de edición?
18. ¿Por qué una cola de revisión necesita precisión y recall?
19. ¿Cómo limpiarías un dataset sin destruir clases minoritarias o casos frontera?
20. ¿Qué herramienta usarías para checks declarativos y cuál para priorizar errores de etiqueta?
21. ¿Por qué comparar todos los pares de un dataset grande no escala?
22. ¿Qué relación hay entre MinHash y Jaccard?
23. ¿Cuándo usarías Fleiss o Krippendorff en lugar de Cohen?
24. ¿Qué evidencia guardarías para defender que una evaluación no está contaminada?
25. ¿Qué parte de tu proceso de calidad traducirías a una plataforma externa y cuál mantendrías como contrato propio?

En resumen

IDEA	QUÉ TE LLEVAS
Calidad es calidad para un uso.	No existe un dataset bueno en abstracto.
El schema es el principio.	La forma importa, pero no basta.
Los duplicados pueden contaminar.	Especialmente si cruzan splits.
Leakage rompe la evaluación.	El test deja de medir generalización.
Las etiquetas son datos revisables.	Necesitan política, acuerdo y cola de revisión.
Un gate organiza la decisión.	`block`, `review` y `pass` separan tipos de acción.
La práctica debe dejar evidencia.	Reporte, cola, duplicados y plan de limpieza.
Los SLOs hacen operativa la calidad.	Un indicador sin objetivo no guía una decisión.
El schema evoluciona.	Los cambios de significado necesitan versión y migración.
La calidad se mira por segmentos.	El fallo puede vivir en un producto, canal o split concreto.
CI/CD también sirve para datos.	El pipeline puede bloquear snapshots con evidencia reproducible.
Limpiar también puede sesgar.	Borrar casos difíciles puede mejorar una métrica y empeorar el sistema.
Las herramientas producen señales, no verdades.	Great Expectations, TFDV, Cleanlab o Evidently ayudan, pero la severidad sigue siendo una decisión de ingeniería.
La revisión se mide.	Precisión y recall permiten saber si una cola ayuda o solo genera ruido.
Deduplicar exige escala.	Blocking, MinHash y scoring separan candidatos baratos de decisiones caras.
El acuerdo depende del proceso.	Dos anotadores, muchos anotadores y anotaciones incompletas no se miden igual.
La contaminación moderna es documental.	En LLMs hay que controlar snapshots, benchmarks, hashes y fechas de corte.
Las herramientas externas no sustituyen criterio.	Monte Carlo, Bigeye, Anomalo, Datafold o Soda ayudan si producen evidencia accionable.

Para saber más

AWS Labs. (2026). Deequ: Unit Tests for Data. https://github.com/awslabs/deequ

Broder, A. Z. (1997). On the Resemblance and Containment of Documents. *Proceedings. Compression and Complexity of Sequences 1997*, 21-29. https://doi.org/10.1109/SEQUEN.1997.666900

Cleanlab. (2026). Cleanlab Documentation. https://docs.cleanlab.ai/

Cohen, J. (1960). A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1), 37-46. https://doi.org/10.1177/001316446002000104

Cover, T. M. y Thomas, J. A. (2006). *Elements of Information Theory* (2.ª ed.). Wiley.

Evidently AI. (2026). Evidently Documentation. https://docs.evidentlyai.com/docs/library/overview

Fleiss, J. L. (1971). Measuring Nominal Scale Agreement among Many Raters. *Psychological Bulletin*, 76(5), 378-382. https://doi.org/10.1037/h0031619

Great Expectations. (2026). Expectations Overview. https://docs.greatexpectations.io/docs/cloud/expectations/expectations_overview/

Jaccard, P. (1901). Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37, 547-579.

Levenshtein, V. I. (1966). Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. *Soviet Physics Doklady*, 10(8), 707-710.

Little, R. J. A. y Rubin, D. B. (2019). *Statistical Analysis with Missing Data* (3.ª ed.). Wiley. https://doi.org/10.1002/9781119482260

Krippendorff, K. (2004). Reliability in Content Analysis: Some Common Misconceptions and Recommendations. *Human Communication Research*, 30(3), 411-433. https://doi.org/10.1111/j.1468-2958.2004.tb00738.x

Manning, C. D., Raghavan, P. y Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Northcutt, C. G., Athalye, A. y Mueller, J. (2021). Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks. *NeurIPS Datasets and Benchmarks*. https://arxiv.org/abs/2103.14749

Northcutt, C. G., Jiang, L. y Chuang, I. L. (2021). Confident Learning: Estimating Uncertainty in Dataset Labels. *Journal of Artificial Intelligence Research*, 70, 1373-1411. https://doi.org/10.1613/jair.1.12125

Pandera. (2026). Pandera Documentation. https://pandera.readthedocs.io/en/stable/

Ratner, A., Bach, S. H., Ehrenberg, H., Fries, J., Wu, S. y Ré, C. (2017). Snorkel: Rapid Training Data Creation with Weak Supervision. *PVLDB*, 11(3), 269-282. https://doi.org/10.14778/3157794.3157797

scikit-learn. (2026). train_test_split. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

Shannon, C. E. (1948). A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(3), 379-423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

Sainz, O., Campos, J. A., García-Ferrero, I., Etxaniz, J., de Lacalle, O. L. y Agirre, E. (2023). NLP Evaluation in Trouble: On the Need to Measure LLM Data Contamination for each Benchmark. *Findings of ACL: EMNLP 2023*. https://doi.org/10.18653/v1/2023.findings-emnlp.722

Soda. (2026). What is Soda? https://docs.soda.io/

TensorFlow. (2026). TensorFlow Data Validation. https://www.tensorflow.org/tfx/data_validation/get_started/

TensorFlow. (2026). TensorFlow Data Validation Anomalies Reference. https://www.tensorflow.org/tfx/data_validation/anomalies

Wang, R. Y. y Strong, D. M. (1996). Beyond Accuracy: What Data Quality Means to Data Consumers. *Journal of Management Information Systems*, 12(4), 5-33. https://doi.org/10.1080/07421222.1996.11518099

Notas

1. Wang, R. Y. y Strong, D. M. (1996). Beyond Accuracy: What Data Quality Means to Data Consumers. *Journal of Management Information Systems*, 12(4), 5-33. https://doi.org/10.1080/07421222.1996.11518099 ↩

2. TensorFlow. (2026). *TensorFlow Data Validation*. https://www.tensorflow.org/tfx/data_validation/get_started/. Consultado el 6 de junio de 2026. ↩

3. TensorFlow. (2026). *TensorFlow Data Validation Anomalies Reference*. https://www.tensorflow.org/tfx/data_validation/anomalies. Consultado el 6 de junio de 2026. ↩

4. Great Expectations. (2026). *Expectations Overview*. https://docs.greatexpectations.io/docs/cloud/expectations/expectations_overview/. Consultado el 6 de junio de 2026. ↩

5. Pandera. (2026). *Pandera Documentation*. https://pandera.readthedocs.io/en/stable/. Consultado el 6 de junio de 2026. ↩

6. AWS Labs. (2026). *Deequ: Unit Tests for Data*. https://github.com/awslabs/deequ. Consultado el 6 de junio de 2026. ↩

7. Northcutt, C. G., Jiang, L. y Chuang, I. L. (2021). Confident Learning: Estimating Uncertainty in Dataset Labels. *Journal of Artificial Intelligence Research*, 70, 1373-1411. https://doi.org/10.1613/jair.1.12125 ↩

8. Northcutt, C. G., Athalye, A. y Mueller, J. (2021). Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks. *NeurIPS Datasets and Benchmarks*. https://arxiv.org/abs/2103.14749 ↩

9. Ratner, A., Bach, S. H., Ehrenberg, H., Fries, J., Wu, S. y Ré, C. (2017). Snorkel: Rapid Training Data Creation with Weak Supervision. *PVLDB*, 11(3), 269-282. https://doi.org/10.14778/3157794.3157797 ↩

.0. Little, R. J. A. y Rubin, D. B. (2019). *Statistical Analysis with Missing Data* (3.ª ed.). Wiley. ↩

.1. Jaccard, P. (1901). Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37, 547-579. ↩

.2. Levenshtein, V. I. (1966). Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. *Soviet Physics Doklady*, 10(8), 707-710. ↩

.3. Broder, A. Z. (1997). On the Resemblance and Containment of Documents. *Proceedings. Compression and Complexity of Sequences 1997*, 21-29. https://doi.org/10.1109/SEQUEN.1997.666900 ↩

.4. Sainz, O., Campos, J. A., García-Ferrero, I., Etxaniz, J., de Lacalle, O. L. y Agirre, E. (2023). NLP Evaluation in Trouble: On the Need to Measure LLM Data Contamination for each Benchmark. *Findings of ACL: EMNLP 2023*. https://doi.org/10.18653/v1/2023.findings-emnlp.722 ↩

.5. Cover, T. M. y Thomas, J. A. (2006). *Elements of Information Theory* (2.ª ed.). Wiley. ↩

.6. Shannon, C. E. (1948). A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(3), 379-423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x ↩

.7. Cover, T. M. y Thomas, J. A. (2006). *Elements of Information Theory* (2.ª ed.). Wiley. ↩

.8. scikit-learn. (2026). *train_test_split*. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html. Consultado el 6 de junio de 2026. ↩

.9. Cohen, J. (1960). A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1), 37-46. https://doi.org/10.1177/001316446002000104 ↩

20. Fleiss, J. L. (1971). Measuring Nominal Scale Agreement among Many Raters. *Psychological Bulletin*, 76(5), 378-382. https://doi.org/10.1037/h0031619 ↩

21. Krippendorff, K. (2004). Reliability in Content Analysis: Some Common Misconceptions and Recommendations. *Human Communication Research*, 30(3), 411-433. https://doi.org/10.1111/j.1468-2958.2004.tb00738.x ↩

22. Manning, C. D., Raghavan, P. y Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press. ↩

23. Great Expectations. (2026). *Expectations Overview*. https://docs.greatexpectations.io/docs/cloud/expectations/expectations_overview/. Consultado el 6 de junio de 2026. ↩

24. Pandera. (2026). *Pandera Documentation*. https://pandera.readthedocs.io/en/stable/. Consultado el 6 de junio de 2026. ↩

25. AWS Labs. (2026). *Deequ: Unit Tests for Data*. https://github.com/awslabs/deequ. Consultado el 6 de junio de 2026. ↩

26. Soda. (2026). *What is Soda?* https://docs.soda.io/. Consultado el 21 de junio de 2026. ↩

27. TensorFlow. (2026). *TensorFlow Data Validation*. https://www.tensorflow.org/tfx/data_validation/get_started/. Consultado el 6 de junio de 2026. ↩

28. Cleanlab. (2026). *Cleanlab Documentation*. https://docs.cleanlab.ai/. Consultado el 21 de junio de 2026. ↩

29. Evidently AI. (2026). *Evidently Documentation*. https://docs.evidentlyai.com/docs/library/overview. Consultado el 7 de junio de 2026. ↩

30. Monte Carlo. (2026). *Monte Carlo at a Glance*. https://docs.getmontecarlo.com/docs/architecture. Consultado el 21 de junio de 2026. ↩

31. Bigeye. (2026). *What is Bigeye?* https://docs.bigeye.com/docs/what-is-bigeye. Consultado el 21 de junio de 2026. ↩

32. Anomalo. (2026). *Product Overview*. https://www.anomalo.com/product-overview/. Consultado el 21 de junio de 2026. ↩

33. Datafold. (2026). *What's a Data Diff?* https://docs.datafold.com/data-diff/what-is-data-diff. Consultado el 21 de junio de 2026. ↩