Ciclo de vida de los datos de aprendizaje automático en la producción

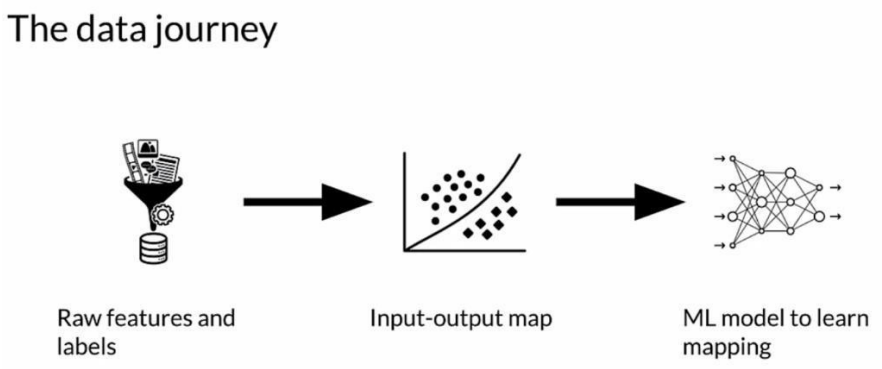
En el segundo curso de Ingeniería de Aprendizaje Automático para la Especialización de Producción, construirá pipelines de datos mediante la recopilación, limpieza y validación de conjuntos de datos y la evaluación de la calidad de los datos; implementará la ingeniería de características, la transformación y la selección con TensorFlow Extended y obtendrá el mayor poder predictivo de sus datos; y establecerá el ciclo de vida de los datos aprovechando las herramientas de metadatos de linaje y procedencia de los datos y seguirá la evolución de los datos con esquemas de datos empresariales.

Entender los conceptos de aprendizaje automático y aprendizaje profundo es esencial, pero si quieres construir una carrera eficaz en el campo de la IA, también necesitas capacidades de ingeniería de producción. La ingeniería de aprendizaje automático para la producción combina los conceptos fundamentales del aprendizaje automático con la experiencia funcional de las funciones modernas de desarrollo de software e ingeniería para ayudarte a desarrollar habilidades listas para la producción.

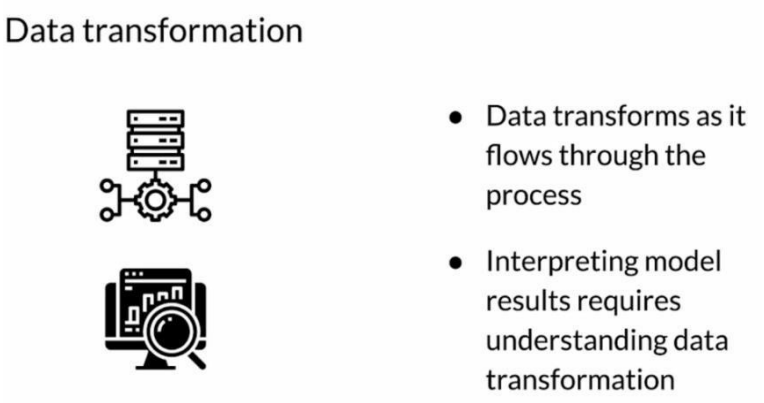
Semana 3: Viaje de datos y almacenamiento de datos

Contenido

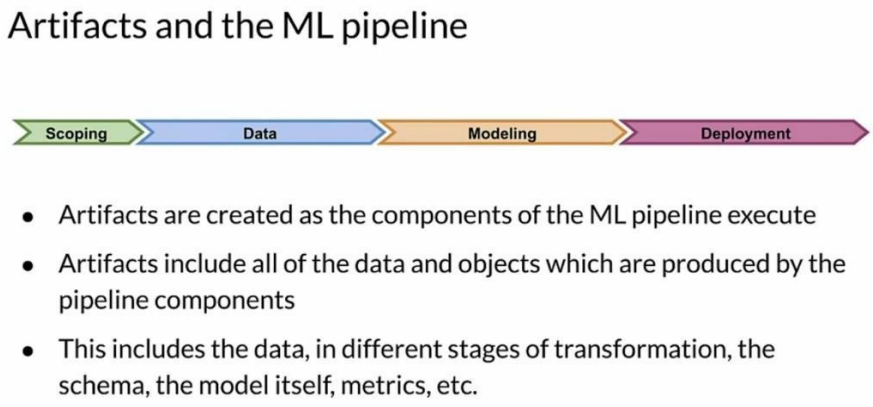
Viaje de datos



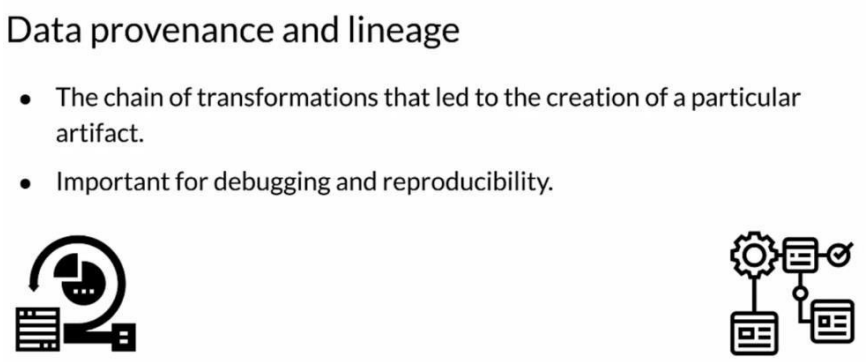
- Para entender la procedencia de los datos es necesario comprender el recorrido de los datos a lo largo del ciclo de vida de la cadena de producción.
- Más concretamente, la contabilización de la evolución de los datos y los modelos a lo largo del proceso.
- Los metadatos de ML son una biblioteca versátil para abordar estos retos, lo que ayuda a la depuración y la reproducibilidad.
- Concretamente, nos permite revisar y seguir los datos y los cambios del modelo a medida que se producen durante el entrenamiento.
- El intento de comprender la procedencia de los datos comienza con el viaje de los datos. El viaje comienza con las características y etiquetas en bruto de cualquier fuente que tengamos.
- Durante el entrenamiento, el modelo aprende el mapeo funcional de la entrada a las etiquetas para ser lo más preciso posible.
- Los datos se transforman y cambian como parte de este proceso de formación.



- A medida que los datos fluyen a través del proceso, se transforman. Algunos ejemplos son el cambio de formato de los datos, la aplicación de ingeniería de características y el entrenamiento del modelo para realizar predicciones.
- Hay una doble conexión entre la comprensión de estas transformaciones de datos y la interpretación de los resultados del modelo.
- Por lo tanto, es importante seguir y documentar de cerca estos cambios.



- Los artefactos de datos se crean a medida que se ejecutan los componentes del pipeline.
- ¿Qué es exactamente un artefacto? Cada vez que un componente produce un resultado, genera un **artefacto**.
- Esto incluye básicamente todo lo que es producido por la tubería, incluyendo los datos en diferentes etapas de transformación, a menudo como resultado de la ingeniería de características y el modelo en sí, y cosas como el esquema, y las métricas y así sucesivamente.
- Básicamente, todo lo que se produce, cada resultado que se produce, es un artefacto.

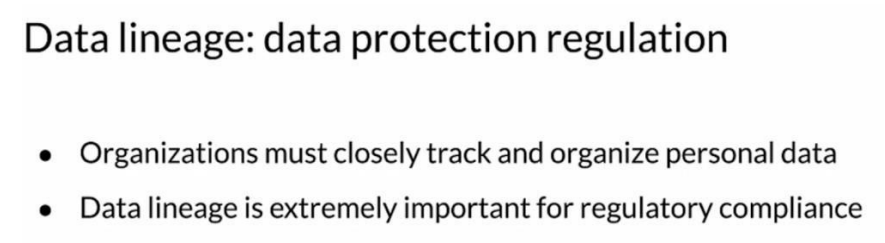


- Los **términos procedencia de los datos y linaje de los datos son básicamente sinónimos** y se utilizan indistintamente.
- La procedencia o el linaje de los datos es una **secuencia de artefactos** que se crean a medida que se avanza en el proceso.
- Esos artefactos están asociados al código y a los componentes que creamos.
- El seguimiento de esas secuencias es fundamental para depurar y comprender el proceso de entrenamiento y comparar diferentes ejecuciones de entrenamiento que pueden ocurrir con meses de diferencia.

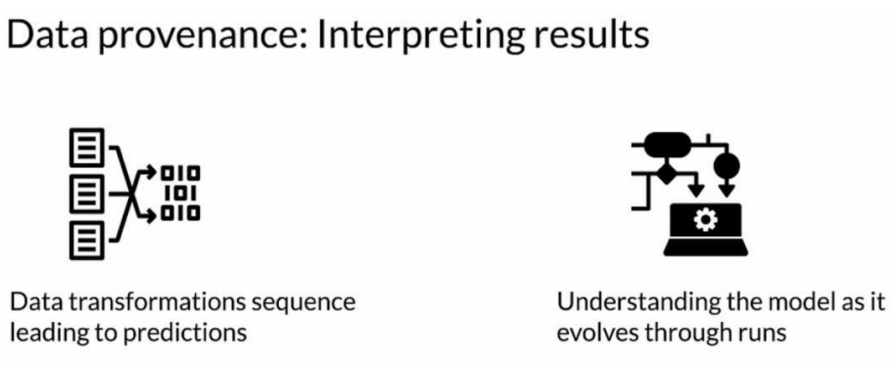
- Los pipelines de aprendizaje automático para la producción han cobrado importancia en varios sectores. Introducen complejidad en el ciclo de vida del ML debido a la gran cantidad de datos, herramientas y flujos de trabajo implicados. Si los datos y

Si los modelos no se rastrean adecuadamente durante el ciclo de vida, resulta inviable volver a crear un modelo de LD desde cero o explicar a las partes interesadas cómo se creó.

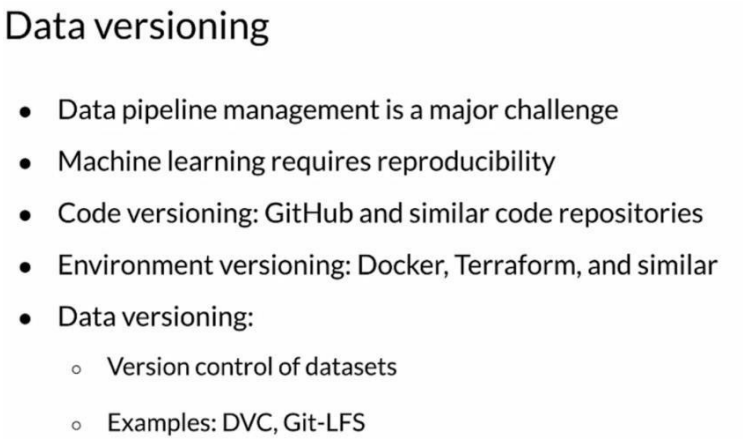
- El establecimiento de mecanismos de seguimiento de la procedencia de los datos y los modelos ayuda a evitar estas deficiencias.
- La procedencia de los datos es muy importante y nos ayuda a entender el proceso y a realizar la depuración.
- La depuración y la comprensión requieren la inspección de esos artefactos en cada punto del proceso de formación, lo que puede ayudarnos a entender cómo se crearon esos artefactos y qué significan realmente los resultados.
- La procedencia también le permitirá hacer un seguimiento de una carrera de entrenamiento desde cualquier punto del proceso.
- Además, la procedencia permite comparar los entrenamientos y entender por qué producen resultados diferentes.



- Según el GDPR o reglamento general de protección de datos, las organizaciones son responsables del origen, los cambios y la ubicación de los datos personales.
- Los datos personales son altamente sensibles, por lo que el seguimiento de los orígenes y los cambios a lo largo de la tubería son clave para el cumplimiento. El linaje de los datos es una excelente manera de que las empresas y organizaciones determinen rápidamente cómo se han utilizado los datos y qué transformaciones se han realizado a medida que los datos se movían a través de la tubería.

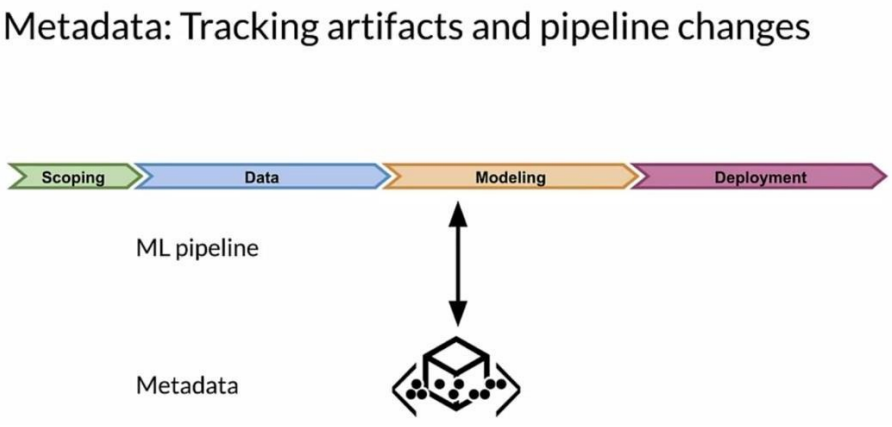


- La procedencia de los datos es clave para interpretar los resultados de los modelos. La comprensión del modelo está relacionada con esto, pero es sólo una parte del cuadro.
- El modelo en sí es una expresión de los datos del conjunto de entrenamiento. En cierto sentido, podemos considerar el modelo como una transformación de los datos.
- La procedencia también nos ayuda a entender cómo evoluciona el modelo a medida que se entrena y, quizás, se optimiza.



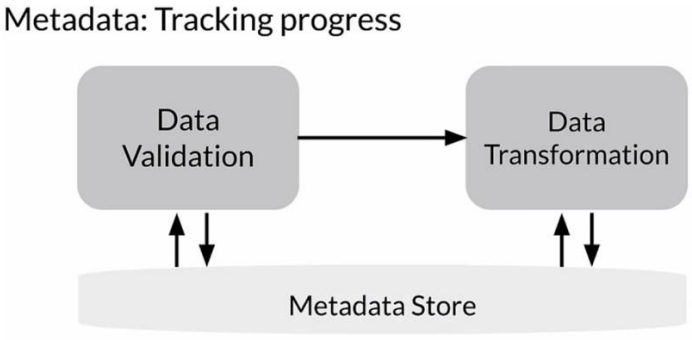
- Añadamos un ingrediente importante aquí, el seguimiento de las diferentes versiones de Datos.
- La gestión de un pipeline de datos es un gran reto, ya que los datos evolucionan a lo largo del ciclo de vida natural de un proyecto, a lo largo de muchos entrenamientos diferentes.
- El aprendizaje automático, cuando se hace correctamente, debería producir resultados que puedan reproducirse de forma bastante consistente. Naturalmente, habrá algunas variaciones, pero los resultados deberían ser parecidos.
- Por lo tanto, los procesos de ML deben incorporar mecanismos resistentes para hacer frente a los datos incoherentes y tener en cuenta las anomalías.
- El control de la versión del código es algo con lo que probablemente esté familiarizado.
- GitHub es uno de los repositorios de código en la nube más populares, y también hay otros.
- El versionado del entorno también es importante. Herramientas como Docker y terraform nos ayudan a crear entornos y configuraciones repetibles.
- Sin embargo, el versionado de datos también es importante, y desempeña un papel significativo para el seguimiento de la procedencia y el linaje de los datos versionados.
- En esencia, se trata de un control de versiones para los archivos de datos, de modo que se pueden rastrear los cambios a lo largo del tiempo y restaurar las versiones anteriores con antelación.
- Pero las herramientas son algo diferentes. Por un lado, por el tamaño de los archivos con los que tratamos, que suelen ser o pueden ser de cualquier manera, mucho más grandes de lo que sería un archivo de código.
- Las herramientas para el versionado de datos están empezando a estar disponibles. Entre ellas se encuentran DVC y Git LFS (almacenamiento de archivos grandes)

Introducción a los metadatos de LD

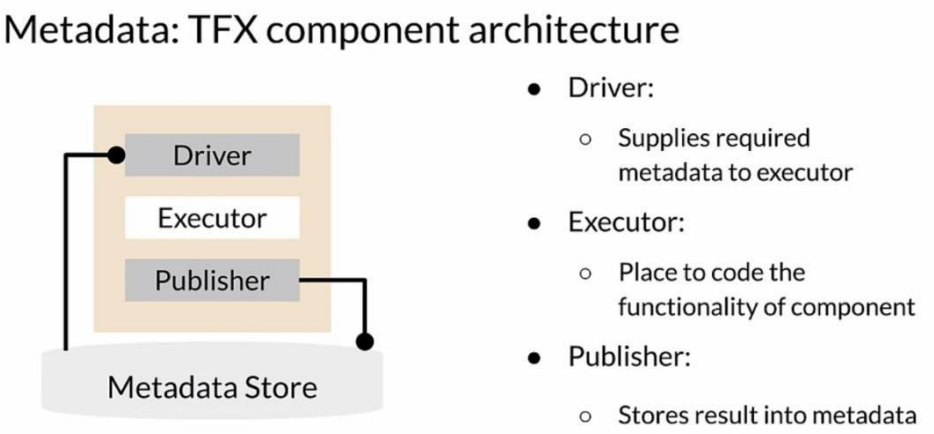


- A medida que el aprendizaje automático se utiliza cada vez más para tomar importantes decisiones empresariales, sanitarias y financieras, la responsabilidad legal se convierte en un factor. Ser capaz de interpretar un modelo y poder rastrear el linaje o la procedencia de los datos que se utilizaron para entrenar el modelo es cada vez más importante para limitar la exposición.

- Ahora vamos a empezar a explorar cómo los metadatos ML (o **MLMD)** pueden ayudarle con el seguimiento de los artefactos y los cambios de la tubería durante un ciclo de vida de producción.
- ML Metadata (MLMD) es una librería para registrar y recuperar metadatos asociados a los pipelines de producción de ML entre otras aplicaciones.
- Cada ejecución de un pipeline ML de producción genera metadatos que contienen información sobre los distintos componentes del pipeline y sus ejecuciones o ejecuciones de entrenamiento y los artefactos resultantes.
- En caso de comportamiento inesperado o de errores en la canalización, estos metadatos pueden aprovecharse para analizar el linaje de los componentes de la canalización y ayudarle a depurar los problemas.
- Piense en estos metadatos como el **equivalente al registro en el desarrollo de software**.
- MLMD le ayuda a comprender y analizar todas las partes interconectadas de su canalización de ML, en lugar de analizarlas de forma aislada.



- Ahora considera las dos etapas de la ingeniería de ML que has visto hasta ahora. Primero has hecho la validación de los datos y luego has pasado los resultados a la transformación de los datos o a la ingeniería de características. Esta es la primera parte de cualquier proceso de entrenamiento de un modelo.
- Pero qué pasaría si tuvieras un repositorio centralizado donde cada vez que ejecutas un componente, almacenas el resultado o la actualización o cualquier otra salida de esa etapa en un repositorio.
- Así, cada vez que se realicen cambios que provoquen un resultado diferente, no tendrás que preocuparte de que se pierdan los progresos realizados hasta el momento.
- Puede examinar sus resultados anteriores para tratar de entender lo que ocurrió y hacer correcciones o aprovechar las mejoras.



- Veámoslo más de cerca. Además del ejecutor donde se ejecuta su código, cada componente incluye dos partes adicionales, el controlador y el editor.
- El ejecutor es donde se realiza el trabajo del componente y eso es lo que hace que los diferentes componentes sean diferentes. Cualquier entrada que se necesite para el ejecutor, es proporcionada por el controlador, que la obtiene del almacén de metadatos.
- Por último, el editor introducirá los resultados de la ejecución del ejecutor en el almacén de metadatos.
- La mayoría de las veces, no necesitarás personalizar el controlador o el editor.
- La creación de componentes personalizados se realiza casi siempre creando un ejecutor personalizado.

ML Metadata library

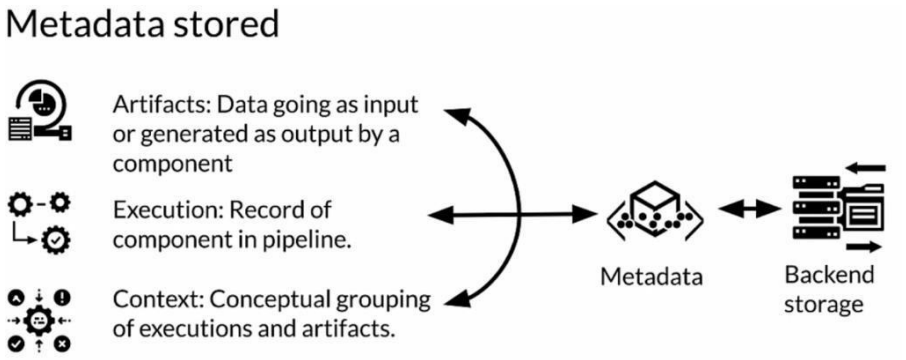
- Tracks metadata flowing between components in pipeline
- Supports multiple storage backends

- Ahora, veamos específicamente los metadatos ML o MLMD.
- **MLMD es una biblioteca para el seguimiento y la recuperación de metadatos** asociados a los flujos de trabajo de los desarrolladores de ML y los científicos de datos.
- El MLMD puede utilizarse como parte integrante de una línea de producción de LD o puede utilizarse de forma independiente.
- Sin embargo, cuando se integra con un pipeline de ML, puede que ni siquiera interactúe explícitamente con MLMD.
- Los objetos que se almacenan en el MLMD se denominan **artefactos**.
- MLMD almacena las propiedades de cada artefacto en una base de datos relacional y almacena los objetos grandes como conjuntos de datos en el disco o en un sistema de archivos o almacén de bloques.

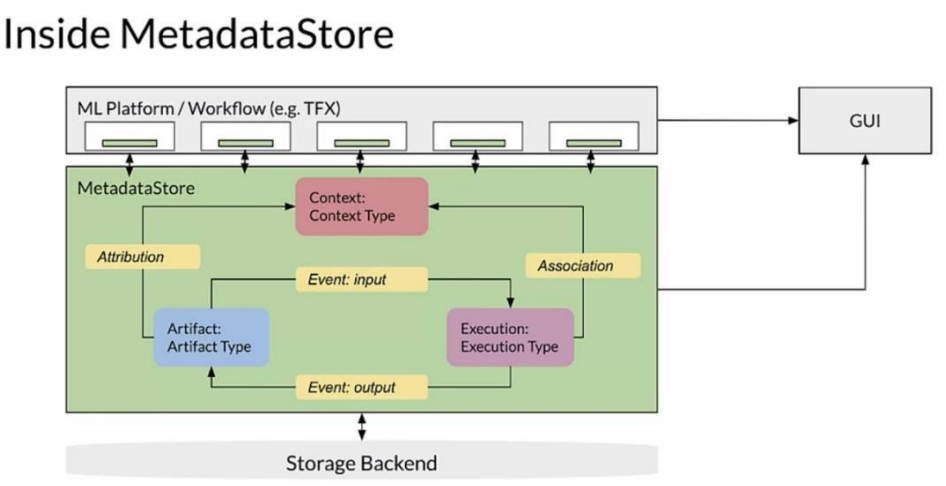
ML Metadata terminology

Units	Types	Relationships
Artifact	ArtifactType	Event
Execution	ExecutionType	Attribution
Context	ContextType	Association

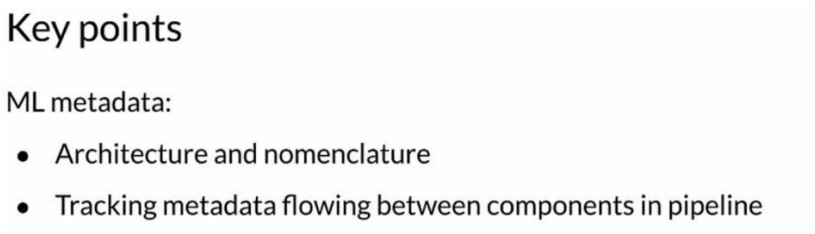
- Cuando se trabaja con metadatos de ML, es necesario saber cómo fluyen los datos entre los diferentes componentes sucesivos.
- Cada paso de este flujo de datos se describe a través de una **entidad** con la que hay que estar familiarizado. En el nivel más alto de MLMD, hay algunas entidades de datos que pueden considerarse como **unidades**.
- En primer lugar, están los artefactos. Un artefacto es una unidad elemental de datos que se introduce en el almacén de metadatos de ML y como los datos se consumen como entrada o se generan como salida de cada componente.
- A continuación están las **ejecuciones**. Cada ejecución es un registro de cualquier componente ejecutado durante el flujo de trabajo de la tubería ML, junto con sus parámetros de ejecución asociados.
- Cualquier artefacto o ejecución se asociará a un solo tipo de componente.
- Los artefactos y las ejecuciones pueden agruparse para cada tipo de componente por separado. Esta agrupación se denomina contexto.
- Un contexto puede contener los metadatos de los proyectos que se ejecutan, los experimentos que se llevan a cabo, los detalles sobre las tuberías, etc.
- Cada una de estas unidades puede contener datos adicionales que la describen con más detalle mediante propiedades.
- A continuación están los tipos. Anteriormente, has visto varios tipos de unidades que se almacenan dentro de los metadatos de ML. Cada tipo incluye las propiedades de ese tipo.
- Por último, tenemos las relaciones. Las relaciones almacenan las distintas unidades que se generan o consumen al interactuar con otras unidades.
- Por ejemplo, **un evento es el registro de una relación entre un artefacto y una ejecución**.
- Una asociación es un registro de la relación entre las ejecuciones y el contexto
- Una atribución es un registro de la relación entre los artefactos y el contexto
- Consulte la documentación de los metadatos de ML para obtener más información



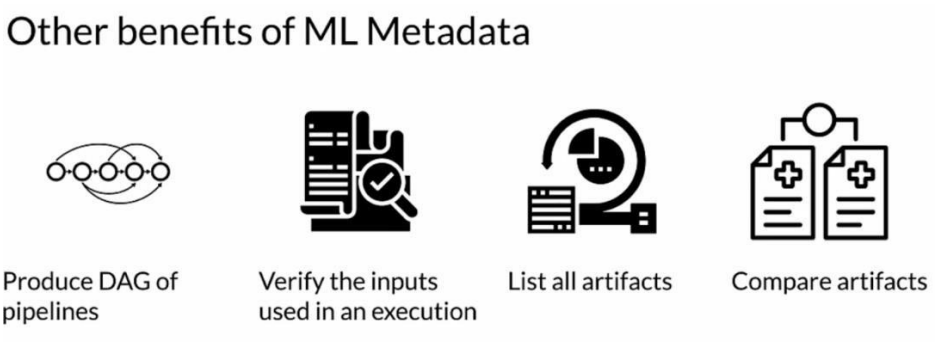
- Así, los metadatos de ML almacenan una amplia gama de información sobre los resultados de los componentes y las ejecuciones de un pipeline.
- Almacena artefactos y almacena las ejecuciones de cada componente en el pipeline.
- También almacena la información del linaje de cada artefacto que se genera.
- Toda esta información se representa en objetos de metadatos y se almacena en una solución de almacenamiento backend.



- Veamos la arquitectura de los metadatos ML o MLMD.
- En la parte superior se encuentran los distintos componentes presentes en cualquier tubería de LD.
- Todos estos componentes están conectados individualmente a un almacén centralizado de metadatos de ML, de modo que cada componente puede acceder de forma independiente a los metadatos en cualquier fase del proceso.
- Un pipeline de ML puede tener opcionalmente una consola GUI que puede acceder a los datos del almacén de metadatos directamente para seguir el progreso de cada componente.
- En el corazón del almacén de metadatos se encuentra el artefacto, que se describe mediante su correspondiente tipo de artefacto.
- Los artefactos se convierten en las entradas de los componentes del pipeline que dependen de ellos. Y el uso correspondiente de los artefactos por parte de los componentes se registra en las ejecuciones.
- La entrada de un artefacto en un componente se describe mediante un evento de entrada y la correspondiente salida de un nuevo artefacto del componente se describe mediante un evento de salida.
- Esta interacción entre artefactos y ejecuciones está representada por el contexto a través de las relaciones de atribución y asociación.
- Por último, todos los datos generados por el almacén de metadatos se almacenan en varios tipos de almacenamiento de fondo como SQLite y MySQL, y los objetos grandes se almacenan en un sistema de archivos o en un almacén de bloques.

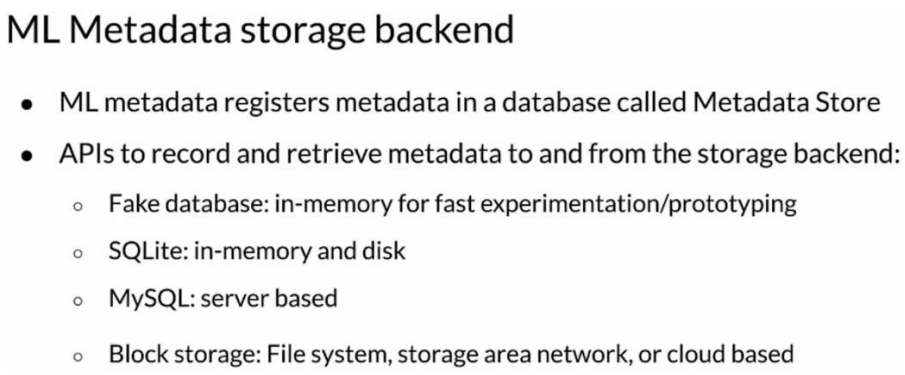


Metadatos ML en acción



- Ahora, echemos un vistazo al uso de los Metadatos ML con un ejemplo de codificación.
- Además del seguimiento del linaje y la procedencia de los datos, se obtienen otras ventajas gracias a los metadatos de ML.
- Esto incluye la capacidad de construir un gráfico acíclico dirigido o DAG, de las ejecuciones de componentes que ocurren en una tubería, que puede ser útil para fines de depuración.
- A través de esto, se puede verificar qué entradas se han utilizado en una ejecución.
- También puede resumir todos los artefactos pertenecientes a un tipo específico generados tras una serie de experimentos.
- Por ejemplo, puede hacer una lista de todos los modelos que se han entrenado. A continuación, puede compararlos para evaluar sus distintas ejecuciones de entrenamiento.

- Ahora, vamos a empezar a escribir el código para ML Metadata. Es posible que tenga que instalar ML Metadata, lo que puede hacer usando PIP como se muestra aquí.
- Luego hay dos importaciones de ML Metadata que se utilizan con frecuencia, el propio almacén de ML Metadata y el almacén de ML Metadata PB2, que es un buffer de protocolo o protobuf.



- Comience por configurar el backend de almacenamiento. El almacén de metadatos de ML es la base de datos donde ML Metadata registra todos los metadatos asociados a su proyecto.
- ML Metadata proporciona APIs para conectar con una base de datos falsa para la creación rápida de prototipos, SQLite y MySQL.
- También necesitamos un almacén de bloques o un sistema de archivos en el que los metadatos de ML almacenen objetos grandes como conjuntos de datos.

Fake database

```
connection_config = metadata_store_pb2.ConnectionConfig()

# Set an empty fake database proto
connection_config.fake_database.SetInParent()


store = metadata_store.MetadataStore(connection_config)
```

- Exploremos rápidamente las tres primeras opciones. Para cualquier backend de almacenamiento, necesitarás crear un objeto connection_config usando el protobuf del almacén de metadatos PB2.
- A continuación, en función de la elección del backend de almacenamiento, hay que configurar este objeto de conexión.
- Aquí estás señalando que tu base de datos falsa está en el padre, que es la memoria primaria del sistema en el que se está ejecutando.
- Por último, se crea el objeto almacén pasando la configuración de la conexión. Independientemente del almacenamiento o la base de datos que utilices, el objeto store es una parte clave de cómo interactúas con ML Metadata.
- **El objeto MetaDataStore recibe una configuración de conexión que corresponde al backend de almacenamiento utilizado.**

SQLite

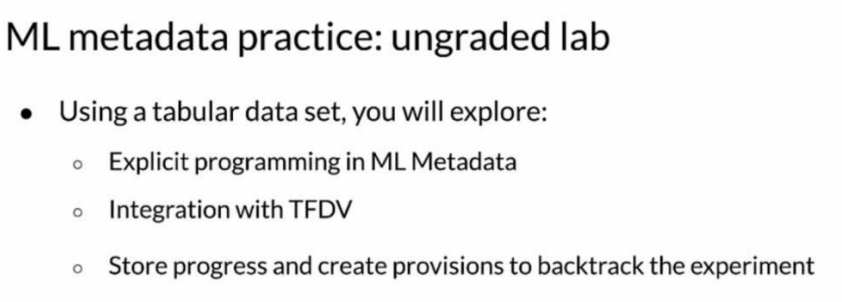
```
connection_config = metadata_store_pb2.ConnectionConfig()

connection_config.sqlite.filename_uri = '...'
connection_config.sqlite.connection_mode = 3 # READWRITE_OPENCREATE

store = metadata_store.MetadataStore(connection_config)
```

- Para utilizar SQLite, se empieza por crear de nuevo una conexión config y luego se configura el objeto de conexión config con la ubicación de su archivo SQLite.
- Asegúrese de proporcionarle los permisos de lectura y escritura pertinentes en función de su aplicación.
- Por último, se crea el objeto tienda pasando la configuración de la conexión.

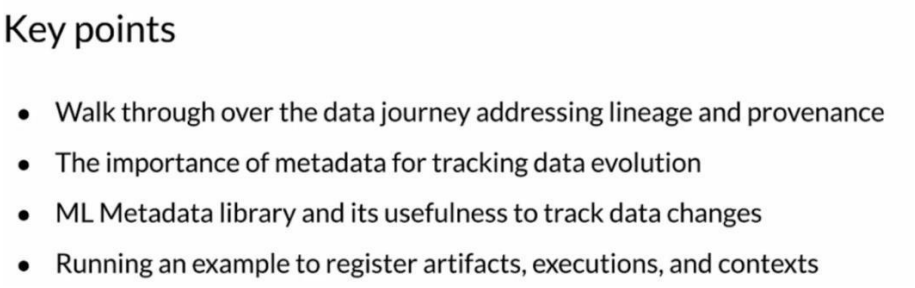
MySQL

```
connection_config = metadata_store_pb2.ConnectionConfig()

connection_config.mysql.host = '...'
connection_config.mysql.port = '...'
connection_config.mysql.database = '...'
connection_config.mysql.user = '...'
connection_config.mysql.password = '...'

store = metadata_store.MetadataStore(connection_config)
```

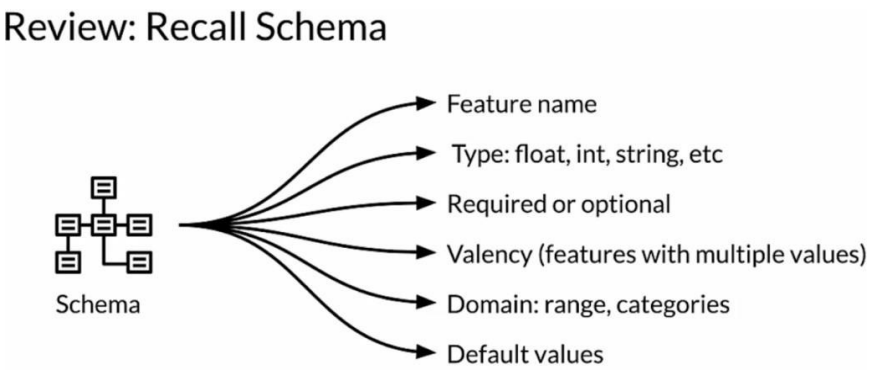
- Como se puede imaginar, el uso de MySQL es muy similar, se empieza por crear una configuración de conexión.
- Tu objeto de configuración de la conexión debe estar configurado con el nombre del host correspondiente, el número de puerto, la ubicación de la base de datos, el nombre de usuario y la contraseña de tu usuario de la base de datos MySQL, y eso se muestra aquí.
- Por último, se crea el objeto tienda, pasando la configuración de la conexión.



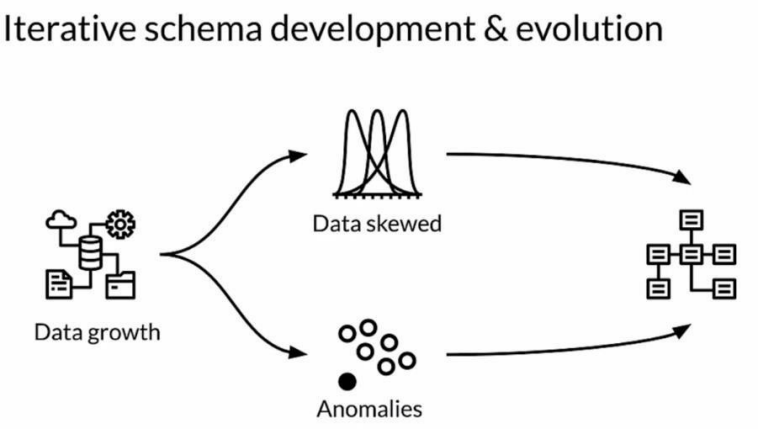
- Ahora, que el backend de almacenamiento está configurado, es el momento de utilizar ML Metadata para resolver un problema. Vamos a repasar un flujo de trabajo usando **TFDV**. Vamos a utilizarlo junto con ML Metadata.
- En este caso se trata de un conjunto de datos tabulares que contiene muchas características. En el laboratorio, los metadatos de ML se programan explícitamente porque en una tubería de ML completa, los metadatos de ML son intrínsecamente capaces de entender el flujo de datos entre varios componentes y realizar sus tareas necesarias.
- Para ayudarle a comprender mejor los metadatos ML, los utilizará fuera de una canalización.
- El laboratorio también muestra la integración de metadatos ML con TensorFlow Data Validation o TFDV.
- Al final del laboratorio, debería tener alguna intuición sobre cómo los metadatos de ML pueden hacer un seguimiento del progreso y cómo puede utilizar los metadatos de ML para hacer un seguimiento de su proceso de formación y su canalización.



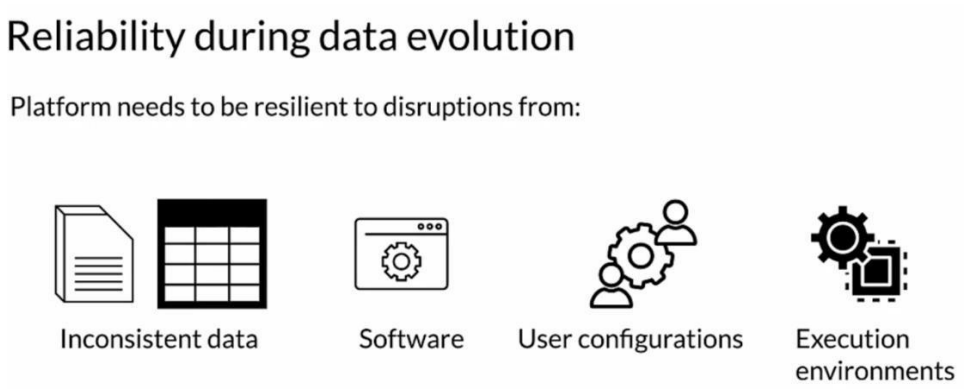
Desarrollo de esquemas



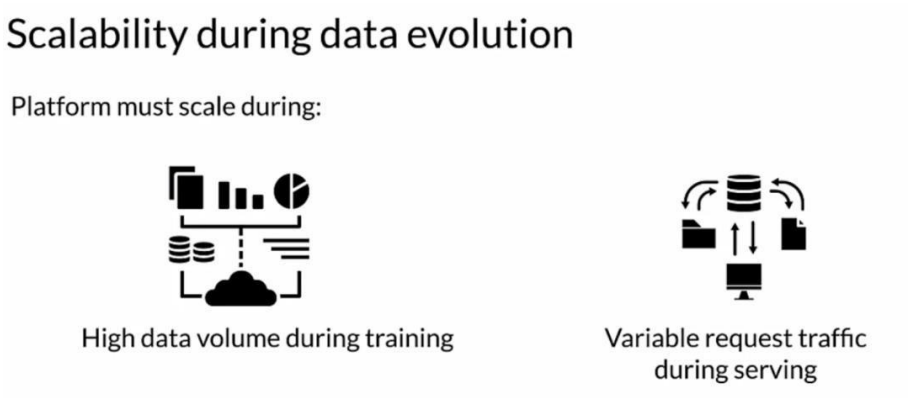
- Ahora vamos a hablar del desarrollo de esquemas. En esta lección sobre la evolución de los datos, aprenderá a desarrollar entornos de esquemas empresariales y a crear y mantener iterativamente esquemas de datos empresariales.
- Utilizarás mucho los esquemas. Repasemos rápidamente lo que es un esquema y su utilidad en el contexto del ML de producción.
- **Los esquemas son objetos relacionales que resumen las características de un determinado conjunto de datos o proyecto.** Incluyen el nombre de la característica, la característica de tipo variable.
- Por ejemplo, una variable entera, flotante, de cadena o categórica. Si la característica es necesaria o no. La valencia de la característica, que se aplica a las características con múltiples valores, como las listas o las características de matriz, y expresa el número mínimo y máximo de valores.
- Información sobre la gama y las categorías y los valores por defecto de las características.



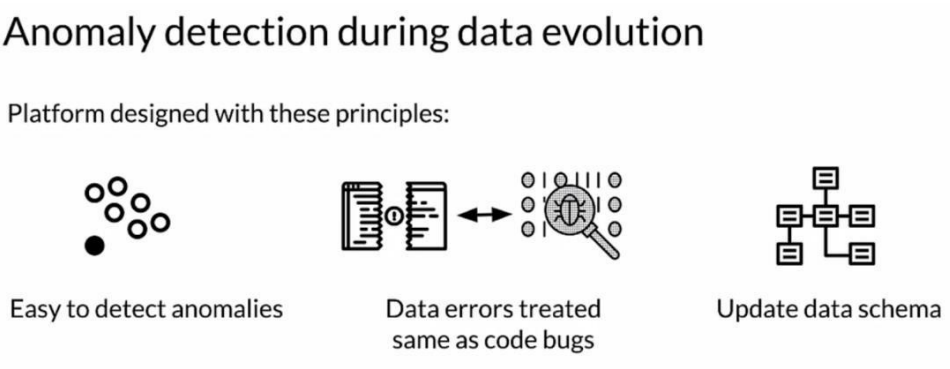
- Los esquemas son importantes porque los datos y el conjunto de características evolucionan con el tiempo.
- Por su experiencia, sabe que los datos no dejan de cambiar y que este cambio suele dar lugar a distribuciones de cambio. Vamos a centrarnos en cómo observar los datos que han cambiado mientras siguen evolucionando.
- **La modificación de los datos suele dar lugar a la generación de un nuevo esquema.**
- Sin embargo, existen algunos casos de uso especiales. Imagínese que, incluso antes de evaluar el conjunto de datos, tiene una idea o información sobre el rango de valores esperado para sus características.
- El conjunto de datos inicial que has recibido sólo cubre un pequeño rango de esos valores.
- En ese caso, tiene sentido ajustar o crear su esquema para reflejar el rango de valores esperado para sus características.
- Una situación similar puede darse para los valores esperados de las características categóricas.
- Además, su esquema puede ayudarle a encontrar problemas o anomalías en su conjunto de datos, como la falta de valores obligatorios o valores de tipo incorrecto.
- Todos estos factores deben tenerse en cuenta a la hora de diseñar el esquema de su pipeline de ML.



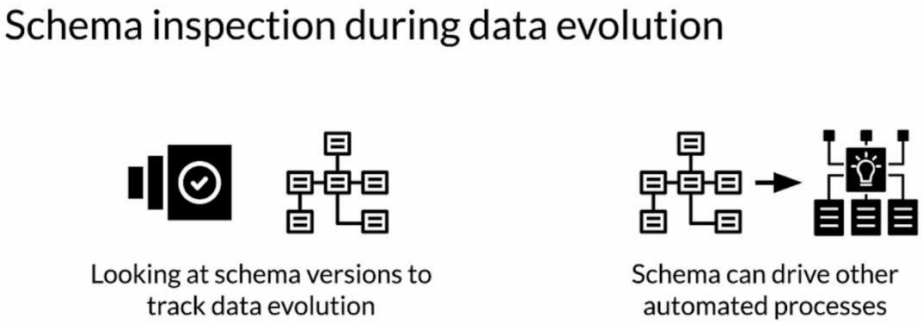
- Como los datos siguen evolucionando, hay algunos requisitos que deben cumplirse en las implantaciones de producción. Consideremos algunos importantes.
- El primer factor es la **fiabilidad**. La plataforma de ML que elijas debe ser **resistente a las interrupciones** derivadas de la inconsistencia de los datos.
- No hay garantía de que recibas datos limpios y consistentes cada vez. De hecho, casi puedo garantizarte que no lo harás.
- Su sistema debe estar diseñado para manejarlo con eficacia.
- Además, su software puede generar errores inesperados en tiempo de ejecución y su canalización necesita manejar eso también con gracia.
- Los problemas de desconfiguración deben ser detectados y manejados con elegancia.
- Por encima de todo, la orquestación de la ejecución entre los diferentes componentes de la cadena de producción debe producirse de forma fluida y eficiente.



- Otro factor a tener en cuenta en su plataforma de canalización de ML es la **escalabilidad** durante la evolución de los datos.
- Durante el entrenamiento, su canalización puede necesitar manejar bien una gran cantidad de datos de entrenamiento, incluyendo mantener ocupados los costosos aceleradores como las GPUs y TPUs.
- Cuando sirves tu modelo, especialmente en escenarios en línea como el que se ejecuta en un servidor, casi siempre tendrás diferentes niveles de tráfico de solicitudes.
- Su infraestructura tiene que **escalar hacia arriba y hacia abajo para cumplir con esos requisitos de latencia** al mismo tiempo que minimiza el coste.

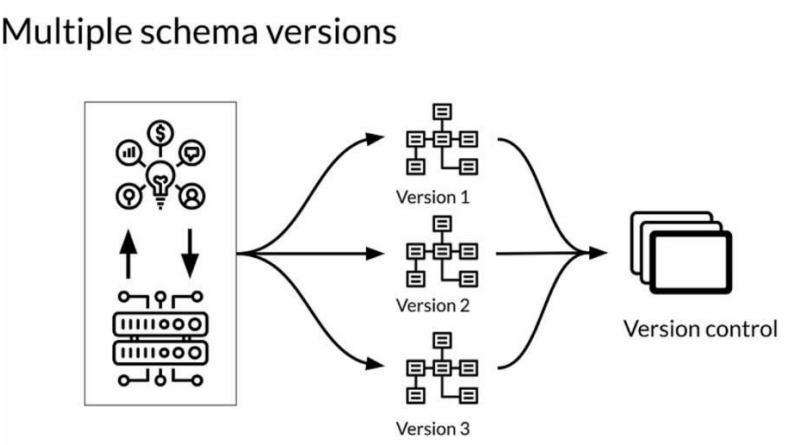


- Si su sistema no está diseñado para manejar la evolución de los datos, se encontrará rápidamente con problemas.
- Entre ellas, la introducción de anomalías en su conjunto de datos. ¿Su sistema detectará esas anomalías? Su sistema y su proceso de desarrollo deben estar diseñados para **tratar los errores de los datos como ciudadanos de primera clase, del** mismo modo que se tratan los fallos en el código.
- En algunos casos, esas anomalías le están alertando de que necesita actualizar el esquema y acomodar los cambios válidos en sus datos.

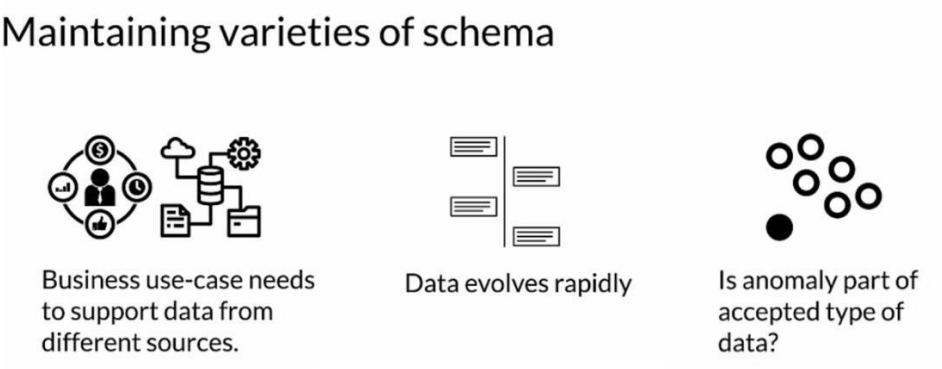


- La evolución de sus esquemas puede ser una herramienta útil para comprender y seguir la evolución de sus datos.
- Además, los esquemas pueden utilizarse como entradas clave en los procesos automatizados que trabajan con sus datos, como la ingeniería automática de características.

Entornos del esquema



- Hablemos de los entornos de esquemas. Su negocio y sus datos evolucionarán a lo largo de la vida de su canal de producción. A menudo ocurre que, a medida que sus datos evolucionan, su esquema también lo hace.
- Mientras desarrolla su código para manejar los cambios en su esquema, puede tener múltiples versiones de su esquema, todas activas al mismo tiempo.
- Es posible que tenga un esquema que se utilice para el desarrollo, otro en pruebas y otro en producción.
- Disponer de un control de versiones para los esquemas, al igual que se hace con el código, ayuda a que esto sea manejable.



- En algunos casos, puede ser necesario tener diferentes esquemas para soportar múltiples escenarios de formación y despliegue para diferentes entornos de datos.
- Por ejemplo, es posible que desee utilizar el mismo modelo en un servidor y en una aplicación móvil, pero imagine que una característica concreta es diferente en esos dos entornos.
- Quizá en un caso sea un entero y en el otro un flotador.
- Es necesario tener un **esquema diferente para cada uno para reflejar la diferencia en los datos**.
- Junto con eso, sus datos están evolucionando. Potencialmente, en todos tus diferentes entornos de datos a la vez.
- Al mismo tiempo, había que comprobar si los datos presentaban problemas o anomalías, y **los esquemas son una parte fundamental para comprobar las anomalías.**

Inspect anomalies in serving dataset

```
stats_options = tfdv.StatsOptions(schema=schema,
                                  infer_type_from_schema=True)

eval_stats = tfdv.generate_statistics_from_csv(
    data_location=SERVING_DATASET,
    stats_options=stats_options
)

serving_anomalies = tfdv.validate_statistics(eval_stats, schema)
tfdv.display_anomalies(serving_anomalies)
```

- Veamos un ejemplo de cómo un esquema puede ayudarle a detectar errores en sus datos de solicitud de servicio y por qué son importantes las múltiples versiones de su esquema.
- Empezaremos por inferir el esquema de servicio y para ello utilizaremos TensorFlow Data Validation o TFDV.
- A continuación, vamos a generar estadísticas para el conjunto de datos que sirve. Luego usaremos TFDV para encontrar si hay algún problema con estos datos y visualizaremos el resultado en un cuaderno.

Anomaly: No labels in serving dataset

Feature name	Anomaly short description	Anomaly long description
'Cover_Type'	Out-of-range values	Unexpectedly small value: 0.

- Mira esto. TFDV informa que hay anomalías en los datos de servicio.
- Dado que se trata de un conjunto de datos que contiene solicitudes de predicción, esto no es sorprendente.
- Falta la **etiqueta** que es el tipo de cobertura, pero el esquema le dice a TFDV que la característica del tipo de cobertura es necesaria. Por lo tanto, está marcando esto como una anomalía. ¿Cómo podemos solucionar este problema?

Schema environments

- Customize the schema for each environment
- Ex: Add or remove label in schema based on type of dataset

- En los escenarios en los que se necesita mantener varios tipos de los mismos esquemas, a menudo es necesario mantener el
entorno del esquema.
- Esto es lo más común en cuanto a la diferencia entre los datos de entrenamiento y los de servicio.
- Puede optar por personalizar su esquema en función de la situación que vaya a manejar.
- Por ejemplo, en este caso, la configuración es mantener dos esquemas. Uno para los datos de entrenamiento, donde se requiere la etiqueta y el otro para servir donde sabemos que no tendremos la etiqueta.

Create environments for each schema

```
schema.default_environment.append('TRAINING')
schema.default_environment.append('SERVING')

tfdv.get_feature(schema, 'Cover_Type')
    .not_in_environment.append('SERVING')
```

- El código para los entornos de esquemas múltiples es bastante sencillo. En nuestro entorno actual, ya tenemos un esquema para la formación.
- A continuación, creamos dos entornos con nombre, denominados formación y servicio.
- Modificamos nuestro entorno de servir eliminando la característica de tipo de cubierta. Como sabemos que al servir, no tendremos eso en nuestro conjunto de características.

Inspect anomalies in serving dataset

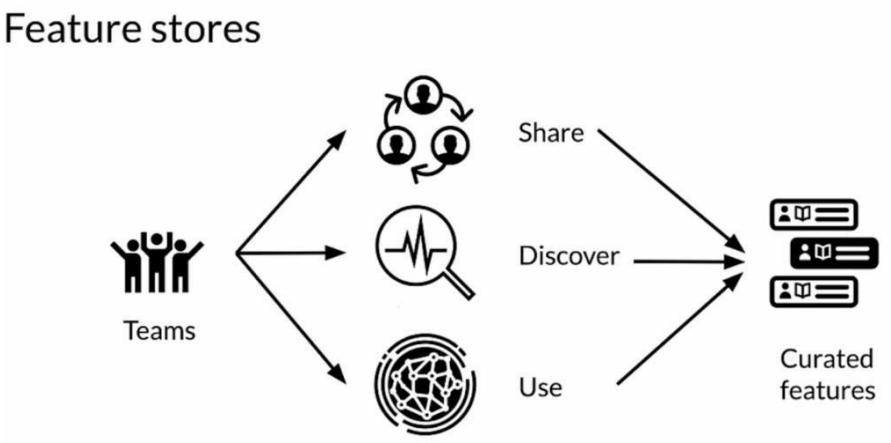
```
serving_anomalies = tfdv.validate_statistics(eval_stats,
                                             schema,
                                             environment='SERVING')

tfdv.display_anomalies(serving_anomalies)
# No anomalies found
```

- Por último, el código configura el entorno de servicio y lo utiliza para validar los datos de servicio.
- Ahora, no se encuentra ninguna anomalía ya que estamos utilizando el esquema correcto para nuestros datos.

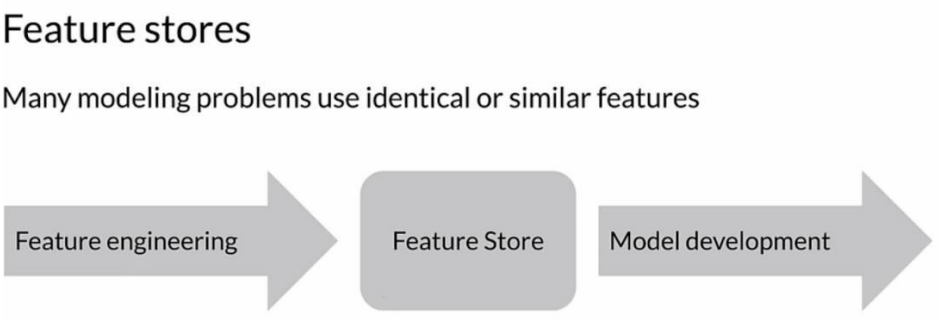
Key points

- Iteratively update and fine-tune schema to adapt to evolving data
- How to deal with scalability and anomalies
- Set schema environments to detect anomalies in serving requests

Tiendas de artículos

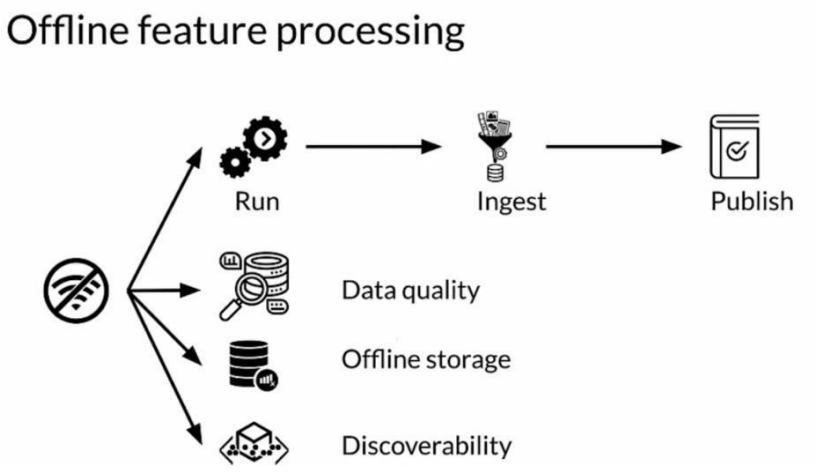


- Ahora pasemos a la cuestión de dónde debemos almacenar nuestros datos. Empezaremos con una discusión sobre los almacenes de características.
- Un **almacén de características es un repositorio central para almacenar características documentadas, curadas y de acceso controlado.**
- El uso de un almacén de funciones permite a los equipos compartir, descubrir y utilizar funciones altamente seleccionadas.
- Un almacén de funciones facilita el descubrimiento y el consumo de esas funciones, y eso puede ser tanto en línea como fuera de línea, tanto para el servicio como para la formación.
- *Nota propia - Buen artículo sobre lo que son las tiendas de características: https://www.phdata.io/blog/what-is-a-feature-store/*



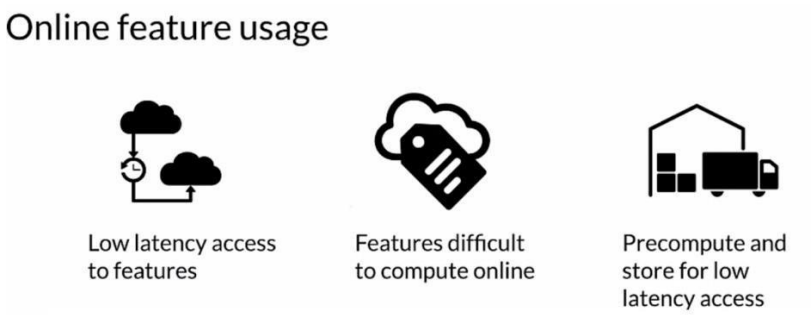
- La gente suele descubrir que muchos problemas de modelización utilizan características idénticas o similares. Así que a menudo los mismos datos se utilizan en múltiples escenarios de modelado.
- En muchos casos, un almacén de características puede considerarse la interfaz entre la ingeniería de características y el desarrollo de modelos.
- Los almacenes de características son **valiosos repositorios centralizados de características que reducen el trabajo redundante**. También son valiosos porque permiten a los equipos compartir datos y descubrir datos que ya están disponibles.
- Puede haber diferentes equipos en una organización con diferentes problemas de negocio que están tratando de resolver. Pero están utilizando datos idénticos o muy similares.
- Por estas razones, **los almacenes de características se están convirtiendo en la opción predominante para el almacenamiento de datos empresariales**, para el aprendizaje automático, para grandes proyectos y organizaciones.

- Los almacenes de características suelen permitir la transformación de los datos para evitar la duplicación de ese procesamiento en diferentes pipelines individuales.
- El acceso a los datos en los almacenes de características puede ser controlado en base a permisos basados en roles.
- Los datos de los almacenes de características pueden agregarse para formar nuevas características.
- También puede tener un tamaño anónimo e incluso purgarse para cosas como el borrado para el cumplimiento del GDPR

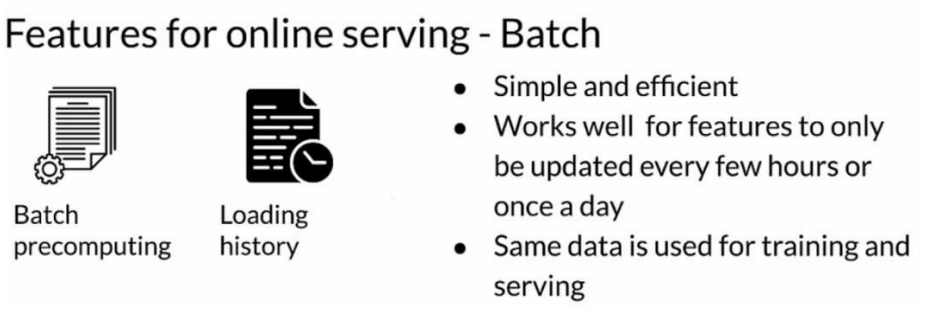


- Los almacenes de características suelen permitir el procesamiento de características **fuera de línea** que se puede hacer de forma regular, tal vez en un trabajo CRON, por ejemplo.

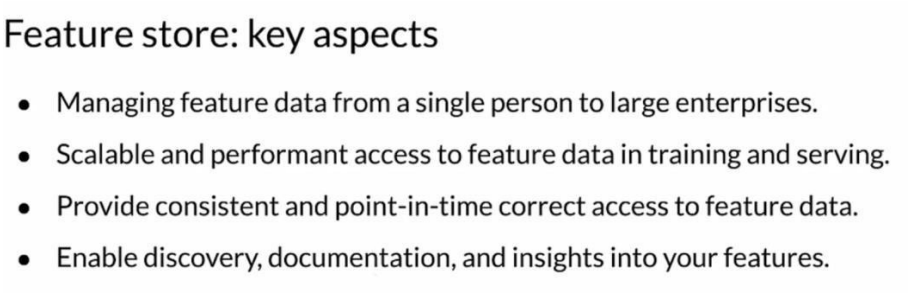
- Imagina que vas a ejecutar un trabajo de ingesta de datos. Y luego tal vez hacer algo de ingeniería de características en él y producir características adicionales de la misma. Tal vez para los cruces de características, por ejemplo
- Estas nuevas características también se publicarán en la tienda de características. Puedes integrarlo con las herramientas de monitorización
- Mientras se procesan y ajustan los datos, se podría estar ejecutando la supervisión fuera de línea.
- Estas características procesadas se almacenan para su uso fuera de línea.
- También pueden formar parte de una solicitud de predicción haciendo una unión con los datos proporcionados en la solicitud de predicción para obtener información adicional.
- Los metadatos de las características le permiten descubrir las características que necesita. Los metadatos que describen los datos que guardas son una herramienta (y a menudo la principal) para intentar descubrir los datos que buscas.



- Para el uso de funciones en línea, donde las predicciones deben ser **devueltas en tiempo real**, los requisitos de latencia suelen ser bastante estrictos.
- Tendrás que asegurarte de que tienes un acceso rápido a esos datos.
- Si vas a hacer una unión, por ejemplo, tal vez con la información de la cuenta del usuario junto con las solicitudes individuales, esa unión tiene que ocurrir rápidamente. Eso es bueno, pero a menudo es difícil calcular algunas de esas características de manera eficiente en línea.
- Por lo tanto, tener características precalculadas suele ser una buena idea. Si se precalculan y almacenan esas características, se pueden utilizar más tarde. Y normalmente eso es con una latencia bastante baja.

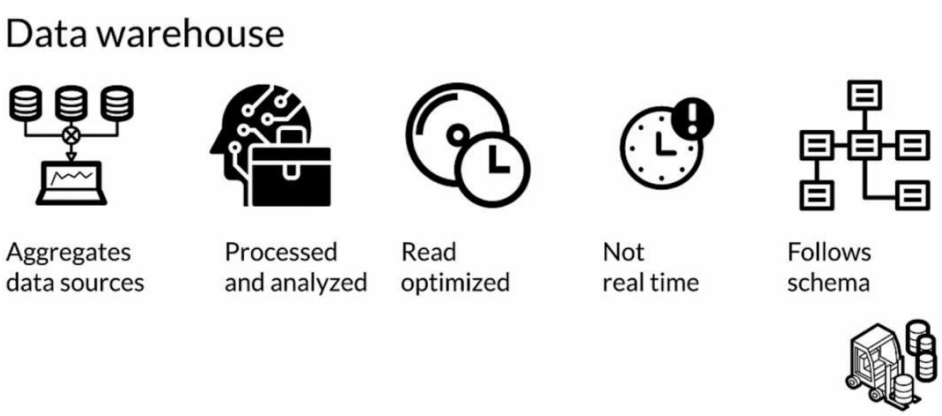


- También puedes hacerlo en un entorno por lotes. Una vez más, no querrás que la latencia sea demasiado larga, pero probablemente no sea tan estricta como una solicitud en línea.
- Para la precomputación y la carga, especialmente en el caso de las características históricas, suele ser bastante sencillo. Para las características históricas en un entorno por lotes, también suele ser sencillo.
- Sin embargo, a la hora de entrenar el modelo**,** hay que **asegurarse de que sólo se incluyan los datos que estarán disponibles en el momento en que se realice una solicitud de servicio**.
- Incluir datos que sólo están disponibles en algún momento después de una solicitud de servicio se denomina **viaje en el tiempo**, y muchos almacenes de funciones incluyen salvaguardias para evitarlo.
- Puedes hacer precomputación en un reloj cada pocas horas o una vez al día. Por supuesto, vas a utilizar los mismos datos para el entrenamiento y para el servicio, con el fin de evitar el sesgo de entrenamiento-servicio.

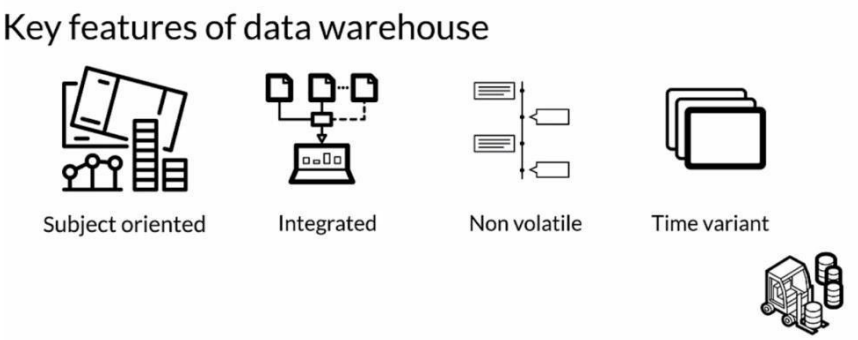


- Los objetivos de la mayoría de los almacenes de características son proporcionar un medio unificado de gestión de datos de características que pueda escalar desde una sola persona hasta grandes empresas.
- Tiene que tener un buen rendimiento y hay que intentar utilizar los mismos datos, tanto cuando se entrenan como cuando se sirven los modelos.
- Usted quiere consistencia y acceso correcto en un momento dado a los datos de las características. Quiere evitar hacer una predicción, por ejemplo, utilizando datos que sólo estarán disponibles en el futuro al servir su modelo.
- En otras palabras, si estás tratando de predecir algo que va a suceder mañana, debes asegurarte de que no estás incluyendo datos de mañana. Sólo deben ser datos de antes de mañana.
- La mayoría de los almacenes de características proporcionan herramientas para permitir el descubrimiento y para permitirle documentar y proporcionar información sobre sus características.

Almacén de datos



- Ahora vamos a hablar de otra solución de almacenamiento de datos empresarial muy popular: los almacenes de datos.
- Los almacenes de datos se desarrollaron originalmente para aplicaciones de big data y de inteligencia empresarial, pero también son herramientas valiosas para el ML de producción.
- Un almacén de datos es una tecnología que **agrega datos de una o varias fuentes** para poder procesarlos y analizarlos.
- Un almacén de datos suele estar pensado para trabajos por lotes de larga duración y su almacenamiento está optimizado para operaciones de lectura.
- Los datos que se introducen en el almacén pueden no estar en tiempo real.
- Cuando se almacenan datos en un almacén de datos, éstos deben **seguir un esquema coherente**.



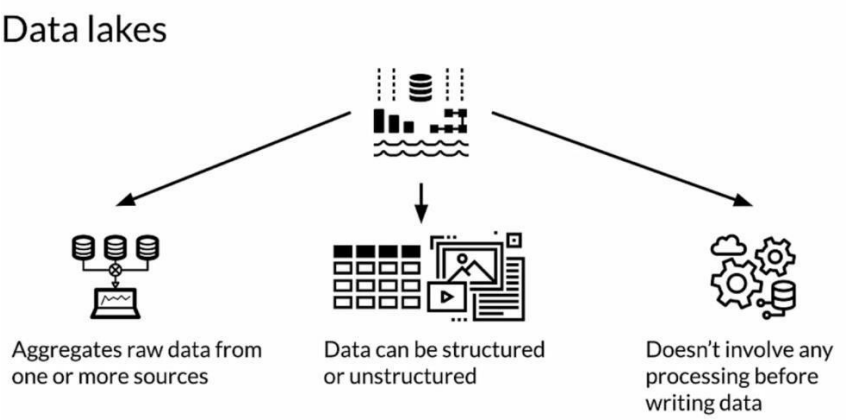
- Un almacén de datos está **orientado a un** tema, y la información que se almacena en él gira en torno a un tema.
- Por ejemplo, los datos almacenados en un almacén de datos pueden centrarse en los clientes o proveedores de la organización, etc.
- Los datos del almacén de datos pueden proceder de varios tipos de fuentes, como bases de datos relacionales o archivos, etc.
- Los datos recogidos en un almacén de datos suelen llevar un sello de tiempo para mantener el contexto de cuándo se generaron.
- Los almacenes de datos son **no volátiles**, lo que significa que las versiones anteriores de los datos **no** se borran cuando se añaden nuevos datos. Eso significa que se puede acceder a los datos almacenados en un almacén de datos en función del tiempo y comprender cómo han evolucionado esos datos.

- Los almacenes de datos ofrecen una mayor capacidad para analizar sus datos. Un almacén de datos puede ayudar a mantener los contextos gracias a la marca de tiempo de los datos.
- Cuando se almacenan los datos en un almacén de datos, éstos siguen un esquema coherente y eso ayuda a mejorar la calidad y la coherencia de los datos.
- Los estudios han demostrado que el **retorno de la inversión en los almacenes de datos tiende a ser bastante alto para muchos casos de uso.**
- Por último, la eficacia de lectura y consulta de los almacenes de datos suele ser alta, lo que permite **un acceso rápido** a los datos.

Comparison with databases

Data warehouse	Database
Online analytical processing (OLAP)	Online transactional processing (OLTP)
Data is refreshed from source systems	Data is available real-time
Stores historical and current data	Stores only current data
Data size can scale to >= terabytes	Data size can scale to gigabytes
Queries are complex, used for analysis	Queries are simple, used for transactions
Queries are long running jobs	Queries executed almost in real-time
Tables need not be normalized	Tables normalized for efficiency

- **Una pregunta natural es, ¿cuál es la diferencia entre un almacén de datos y una base de datos?**
- **Los almacenes de datos están pensados para analizar los datos, mientras que las bases de datos suelen utilizarse para realizar transacciones.**
- En un almacén de datos, puede haber un retraso entre el almacenamiento de los datos y su reflejo en el sistema. Pero en una base de datos, **los datos suelen estar disponibles inmediatamente después de ser almacenados**.
- Los almacenes de datos almacenan los datos en función del tiempo, por lo que también se dispone de datos históricos.
- Los almacenes de datos suelen ser capaces de almacenar una mayor cantidad de datos en comparación con las bases de datos.
- Las consultas en los almacenes de datos son complejas por naturaleza y tienden a ejecutarse durante mucho tiempo, mientras que las consultas en las bases de datos son simples y tienden a ejecutarse en tiempo real.
- La **normalización** no es necesaria para los almacenes de datos, pero debería utilizarse con las bases de datos.



- Un lago de datos es un sistema o repositorio de datos almacenados en su **formato natural y crudo**, que suele ser en forma de **blobs** o **archivos**.
- Un lago de datos, al igual que un almacén de datos, agrega datos de varias fuentes de datos empresariales.
- Un lago de datos puede incluir datos estructurados, como bases de datos relacionales, o semiestructurados, como archivos CSV, o no estructurados, como una colección de imágenes o documentos, etc.
- Como los lagos de datos almacenan los datos en su formato bruto, **no realizan ningún tipo de procesamiento** y **no** suelen **seguir un esquema.**

Comparison with data warehouse

	Data warehouses	Data lakes
Data Structure	Processed	Raw
Purpose of data	Currently in use	Not yet determined
Users	Business professionals	Data scientists
Accessibility	More complicated and costly to make changes	Highly accessible and quick to update

- ¿En qué se diferencia un lago de datos de un almacén de datos? Comparemos los dos.
- La principal diferencia entre ellos es que en un almacén de datos, los datos se almacenan en un formato coherente que sigue un esquema, mientras que en los lagos de datos, los datos suelen estar en su formato bruto.
- En los lagos de datos, la razón por la que se almacenan los datos no suele estar determinada de antemano. No suele ser el caso de un almacén de datos, en el que se suele almacenar con un propósito concreto.
- Los almacenes de datos suelen ser utilizados también por los profesionales de la empresa, mientras que **los lagos de datos suelen ser utilizados únicamente por los profesionales de los datos, como los científicos de datos**.
- Dado que los datos de los almacenes de datos se almacenan en un formato consistente, los cambios en los datos pueden ser complejos y costosos. Los lagos de datos, sin embargo, son más flexibles y facilitan la realización de cambios en los datos.

Key points

- **Feature store**: central repository for storing documented, curated, and access-controlled features, specifically for ML.
- **Data warehouse**: subject-oriented repository of structured data optimized for fast read.
- **Data lakes**: repository of data stored in its natural and raw format.

Referencias

Versionado de datos:
https://dvc.org/
https://git-lfs.github.com/

Metadatos ML:
https://www.tensorflow.org/tfx/guide/mlmd#data_model
https://www.tensorflow.org/tfx/guide/understanding_custom_components

Conjunto de datos sobre viajes en taxi en Chicago:
https://data.cityofchicago.org/Transportation/Taxi-Trips/wrvz-psew/data
https://archive.ics.uci.edu/ml/datasets/covertype

Fiesta:
https://cloud.google.com/blog/products/ai-machine-learning/introducing-feast-an-open-source-feature-store-for-machine-learning
https://github.com/feast-dev/feast
https://www.gojek.io/blog/feast-bridging-ml-models-and-data