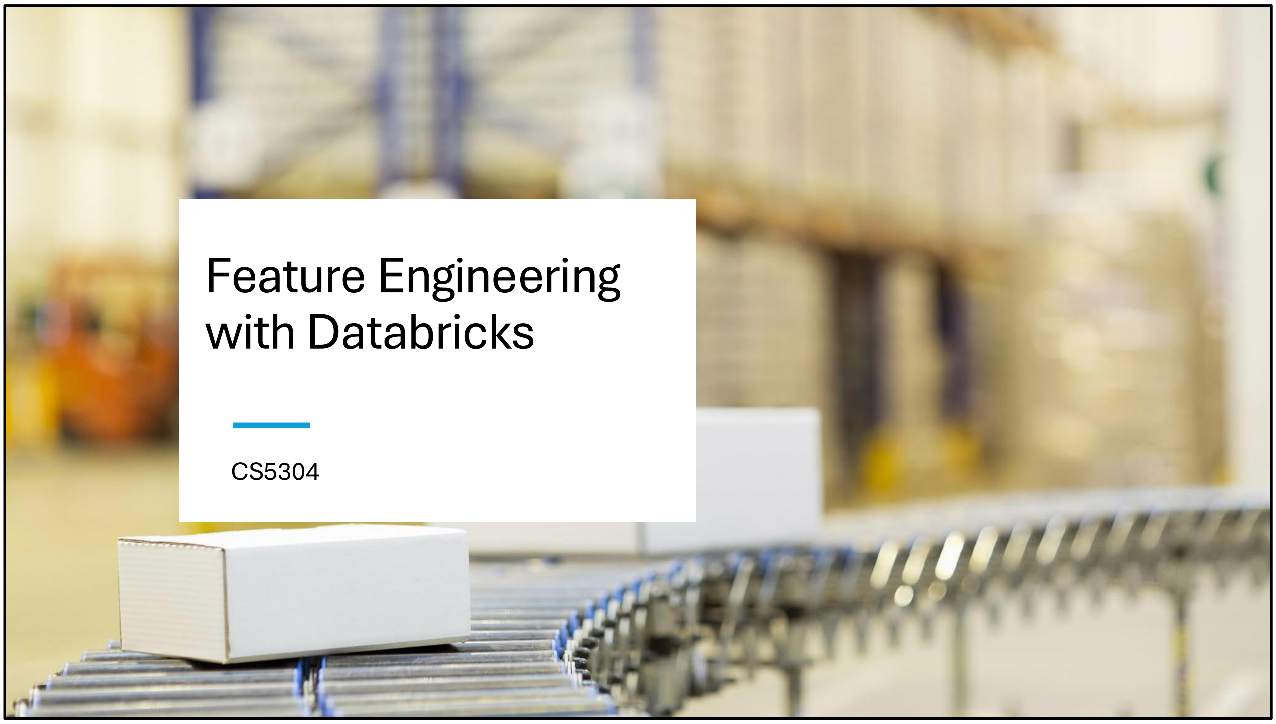
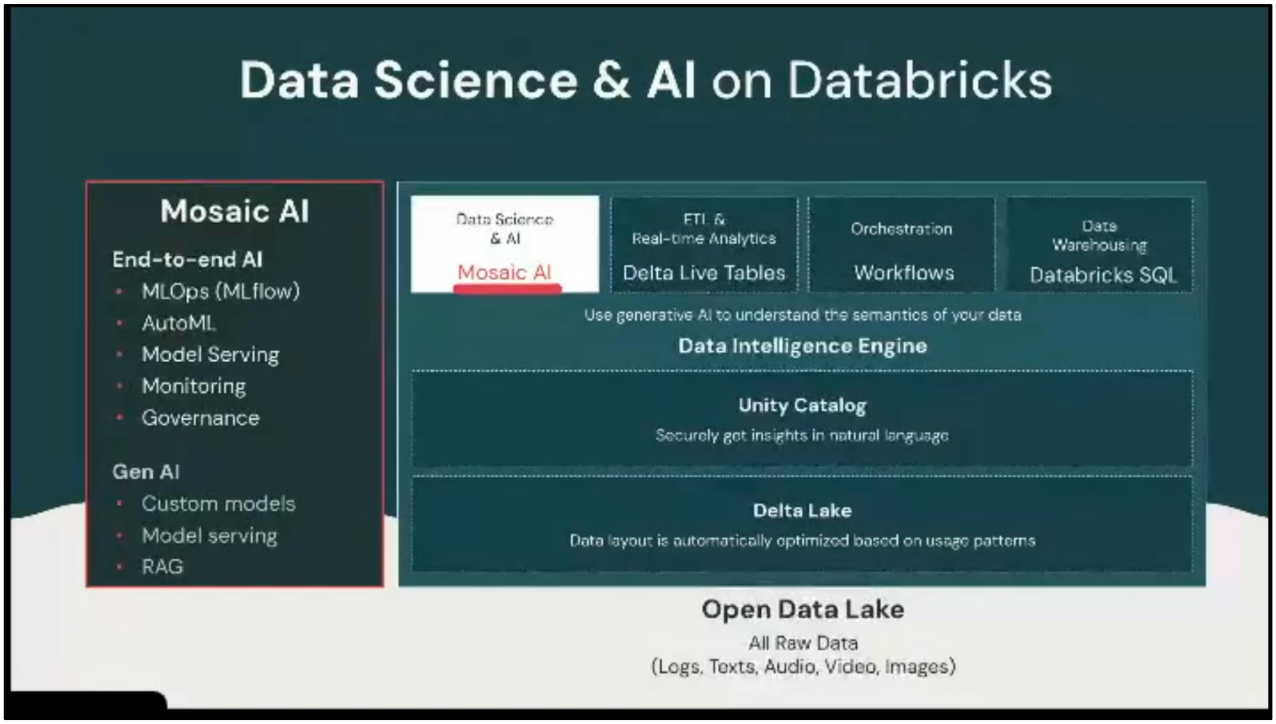
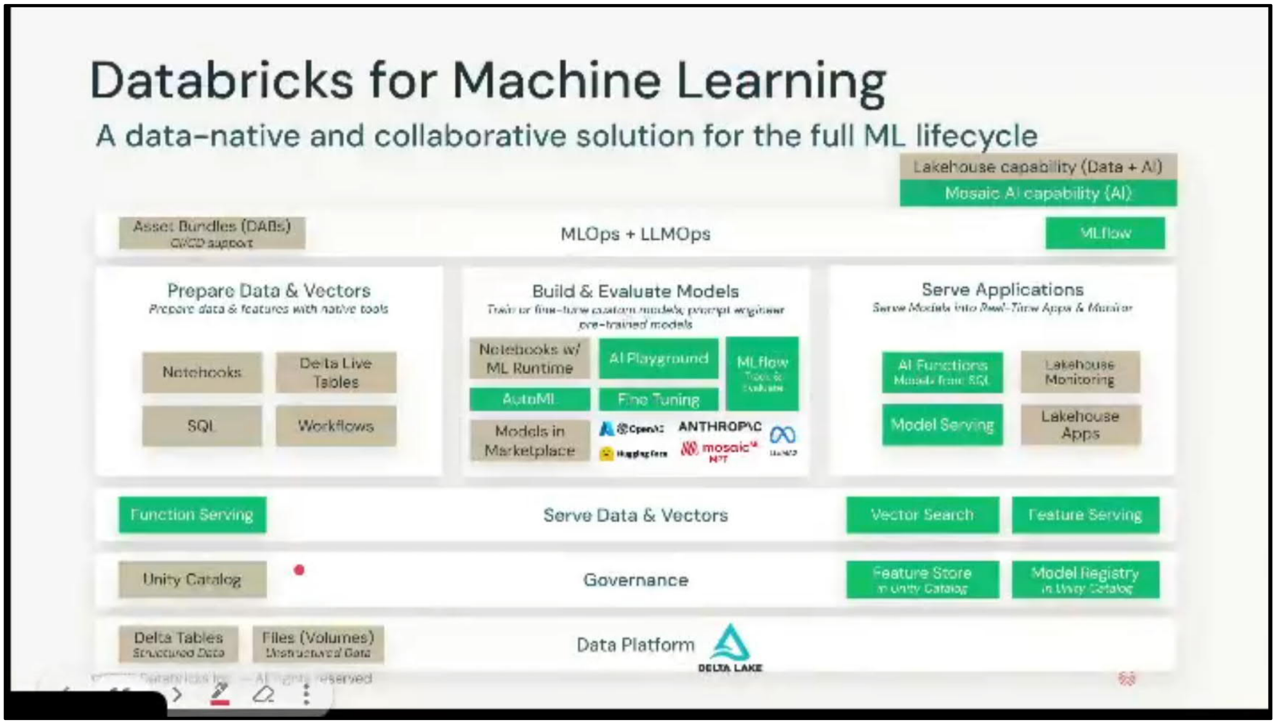

Feature Engineering
with Databricks
CS5304


Data Science & AI on Databricks
Mosaic AI
End-to-end AI
MLOps (MLflow)
AutoML
Model Serving
Monitoring
Governance
Gen AI
Custom models
Model serving
RAG
Data Science & AI
Mosaic AI
ETL & Real-time Analytics
Delta Live Tables
Orchestration
Workflows
Data Warehousing
Databricks SQL
Use generative AI to understand the semantics of your data
Data Intelligence Engine
Unity Catalog
Securely get insights in natural language
Delta Lake
Data layout is automatically optimized based on usage patterns
Open Data Lake
All Raw Data
(Logs, Texts, Audio, Video, Images)

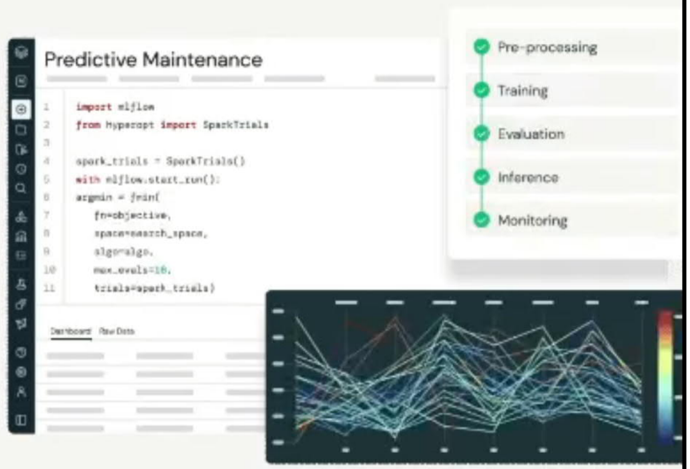


Databricks for Machine Learning
A data-native and collaborative solution for the full ML lifecycle
Lakehouse capability (Data + AI)
Mosaic AI capability (AI)
Asset Bundles (DABs)
CI/CD support
MLOps + LLMOps
MLflow
Prepare Data & Vectors
Prepare data & features with native tools
Notebooks
Delta Live Tables
SQL
Workflows
Build & Evaluate Models
Notebooks w/ ML Runtime
AI Playground
MLflow
AutoML
Fine Tuning
Models in Marketplace
ANTHROPIC
mosaic
Serve Applications
AI Functions
Lakehouse Monitoring
Model Serving
Lakehouse Apps
Function Serving
Serve Data & Vectors
Vector Search
Feature Serving
Unity Catalog
Governance
Feature Store
Model Registry
Delta Tables
Structured Data
Files (Volumes)
Data Platform
DELTA LAKE

Features of Databricks for Machine Learning

Non exhaustive list of features that will be used throughout this module

- **Collaborative notebooks**
- **ML Runtime**
- **Governance of Data & Models *(via Unity Catalog)***
- **Feature Store**
- Managed MLflow
- Model Serving
- AutoML



Quick Exploratory Data Analysis

Native tools for visualizing and understanding data in ML workflow

Create **interactive charts** to visualize data in the Notebook with only two clicks

Summarize a data set's essential properties and statistics in a **data profile** with the push of a button

Data Preparation for ML projects

Goal: Optimize input quality for accurate model predictions

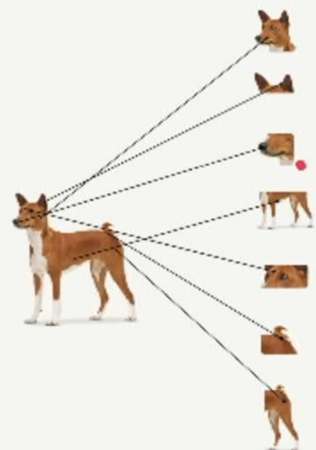
Data preparation includes the following tasks:

- **Cleaning and formatting data:** This includes tasks such as **handling missing values** or outliers, ensuring data is in the correct format, and removing unneeded columns.
- **Feature Engineering:** This includes tasks like numerical **transformations**, **aggregating data**, encoding text or image data, and **creating new features**.

Feature Extraction

Transforming raw data into a set of features that better represent the underlying patterns

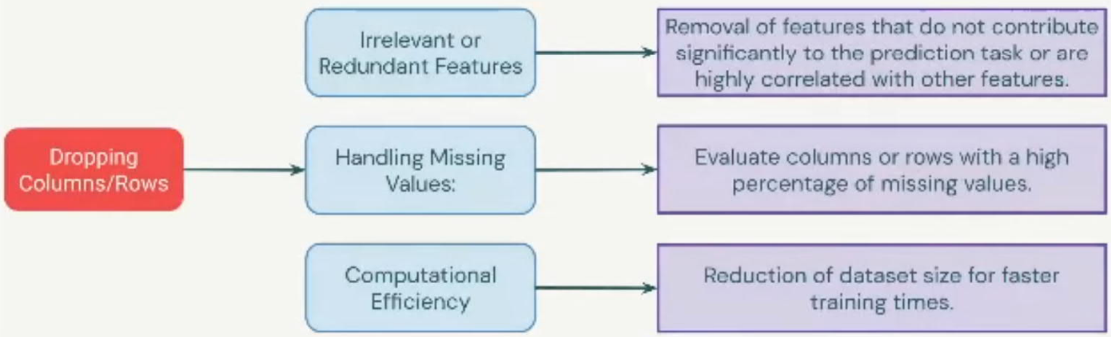
- Transforming Raw Data for Enhanced Modeling
- Dimensionality Reduction for Improved Performance
- Simplifying Feature Engineering

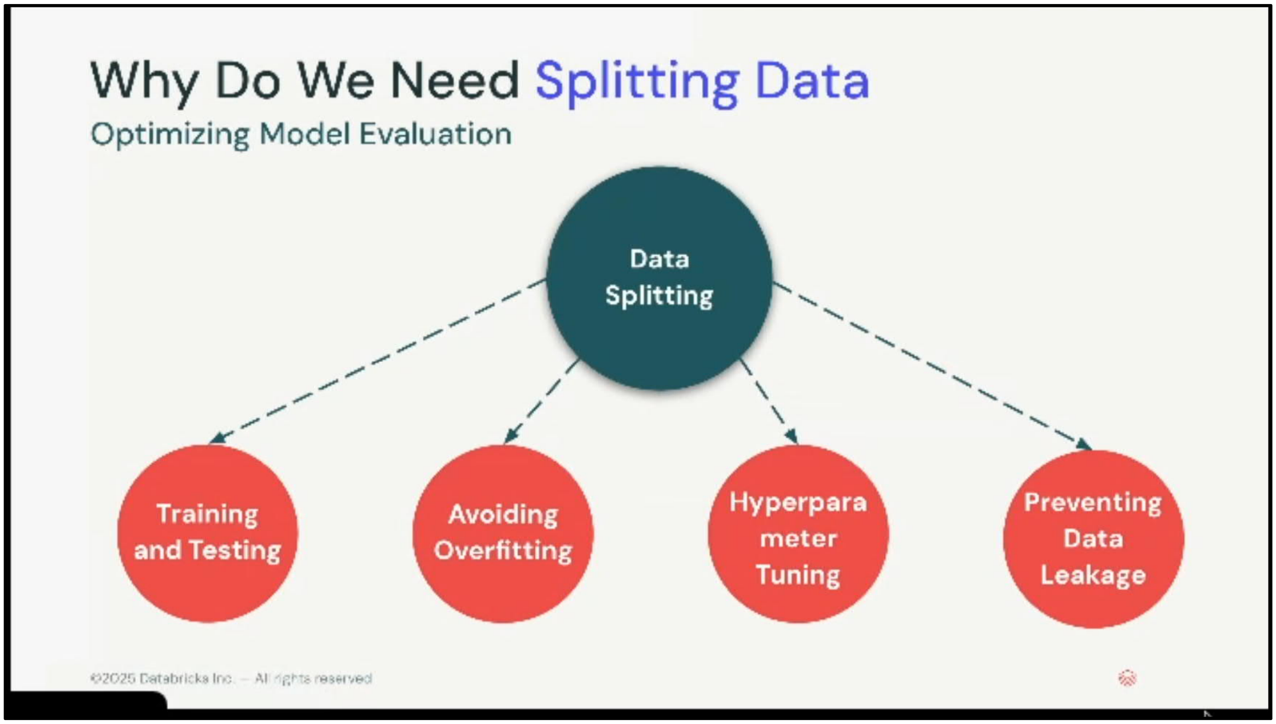


Feature Selection: Dropping Columns/Rows

Enhancing Model Efficiency

Streamlining the dataset by selectively removing columns (features) or rows (instances) to improve model efficiency and effectiveness.

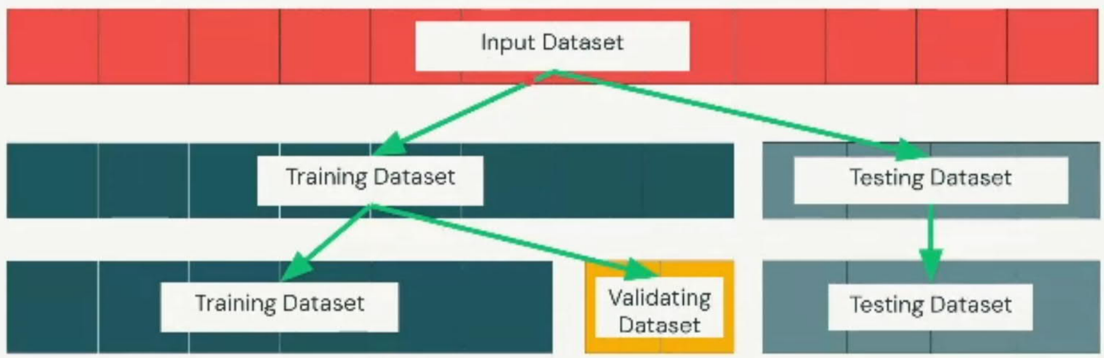



Why Do We Need Splitting Data
Optimizing Model Evaluation
Data Splitting
Training and Testing
Avoiding Overfitting
Hyperparameter Tuning
Preventing Data Leakage
©2025 Databricks Inc. — All rights reserved

Splitting Data into Multiple Sets

Optimizing Model Training, Validation, and Testing

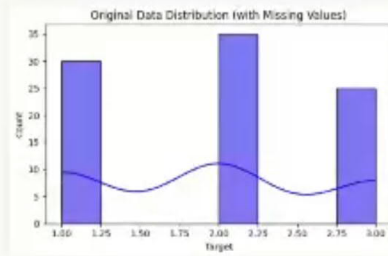
Splitting the dataset is a fundamental step in data preparation to facilitate effective model training, validation, and testing.

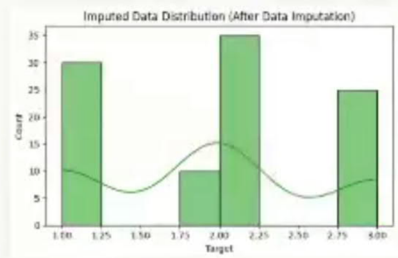


Data Imputation

Data imputation is **the process of filling in missing values** in a dataset with estimated or predicted values.

The goal of data imputation is to enhance the quality and completeness of the dataset, ultimately improving the performance and reliability of the machine learning model.

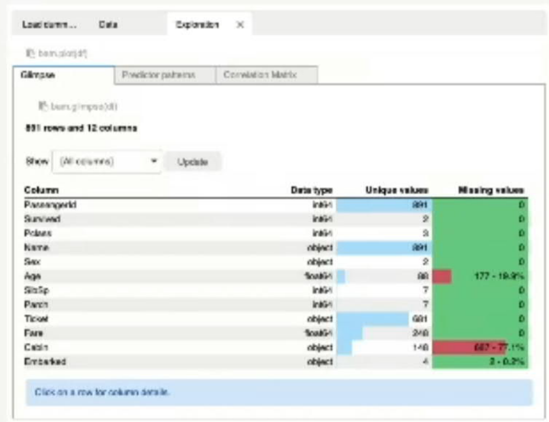




Problems with Missing Data

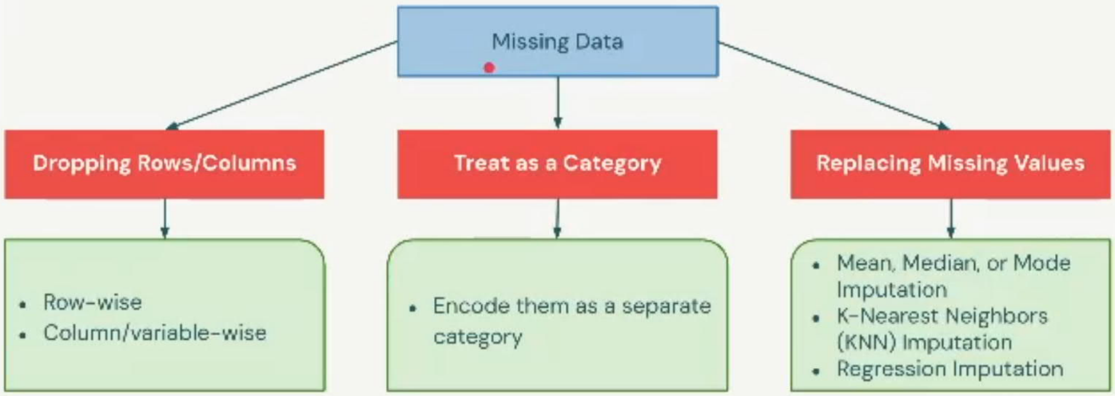
Impacting the performance and reliability of ML models

- Reduced Model Performance
- Biased Inferences
- Imbalanced Representations
- Increased Complexity in Model Handling



How to Handle Missing Data

Data imputation methods



Replacing Missing Values

Data imputation methods

Mean - Mode Imputation

Before	After
10	10.0
15	15.0
-	18.3
20	20.0
25	25.0

K-Nearest Neighbors (KNN with k=2)

Before	After
8	8.0
-	10.0
12	12.0
15	15.0
-	13.0

Multiple Imputation (Regression)

F1	F2	Before	After
X	X	10	10.0
X	X	15	15.0
X	$Y=\beta_0+\beta_1X_1+\beta_2X_2+\varepsilon$		Y
X	X	20	20.0
X	X	25	25.0

Marking Imputed Data (*Best Practice*)

Keep track of imputed data

Important to mark imputed data, for:

- Model Evaluation
- Data Quality Assessment
- Enabling Transparency of Dataset
- Error Identification

ID	Name	Age	Age_imputed
1	Alice	25.0	0
2	Bob	30.0	0
3	Charlie	26.0	1
4	David	28.0	0
5	Eva	22.0	0

Data Encoding

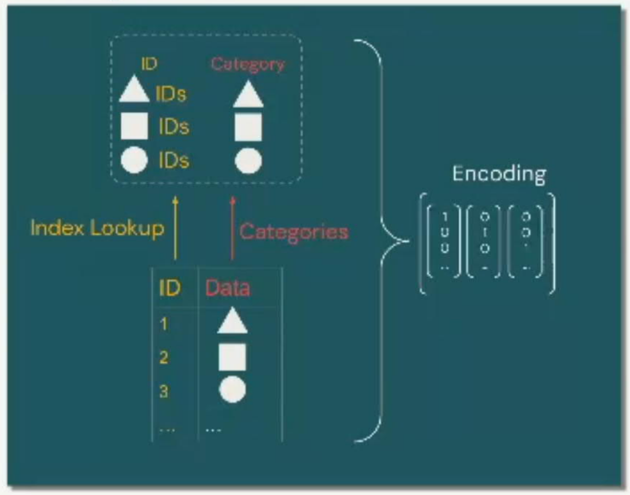
Why Encoding?

Data encoding is a important pre-processing step in **preparing categorical data for machine learning algorithms**, as vast majority of algorithms accept numerical input exclusively.

Issues with classic ML

- Handling high-cardinality features
- Introducing unintended relationships
- Overfitting
- Increased computational cost
- Possible lack of interpretability

Encoding Process



Working with Categorical Features

High Cardinality

Issue: Large set of categories

Many unique values in a categorical feature can lead to a large number of dummy variables, increasing dimensionality and potentially causing issues. This is known as the high-cardinality problem.

Possible Solutions:

Group Rare Categories:

ID	Category
0	A
1	B
2	A
3	C
4	D
5	D
6	D
7	D

ID	Category	Cat_Group
0	A	A
1	B	Rare
2	A	A
3	C	Rare
4	D	D
5	D	D
6	D	D
7	D	D

Top-N Categories:

ID	Category
0	A
1	B
2	A
3	C
4	D
5	D
6	D
7	D

ID	Category	Cat_Group
0	A	A
1	B	Other
2	A	A
3	C	Other
4	D	D
5	D	D
6	D	D
7	D	D

Working with Categorical Features

Missing Values

Issue: Categorical gaps

Categorical features often have missing values, which need to be addressed before model training.

Possible Solutions:

Imputation:

ID	Feature A	Feature B
0	1.0	10.0
1	2.0	NaN
2	NaN	30.0
3	4.0	40.0
4	5.0	50.0

→

ID	Feature A	Feature B
0	1.0	10.0
1	2.0	32.5
2	3.0	30.0
3	4.0	40.0
4	5.0	50.0

Consider Missing as a Separate Category:

ID	Feature A	Feature B
0	1.0	10.0
1	2.0	NaN
2	NaN	30.0
3	4.0	40.0
4	5.0	50.0

→

ID	Feature A	Feature B
0	1.0	10.0
1	2.0	-1.0
2	-1.0	30.0
3	4.0	40.0
4	5.0	50.0

Working with Categorical Features

Encoding Categories

Issue: String types

Models require numerical input, and categorical variables need to be encoded.

Possible Solutions:

One-Hot Encoding

ID	Category
0	A
1	B
2	A
3	C
4	A

ID	Cat_A	Cat_B	Cat_C
0	1	1	1
1	0	D	0
2	1	1	1
3	0	0	0
4	1	1	1

Label Encoding:

ID	Category
0	A
1	B
2	A
3	C
4	A

ID	Category	Label
0	A	0
1	B	1
2	A	0
3	C	2
4	A	0

Ordinal Encoding:

ID	Category
0	A
1	B
2	A
3	C
4	A

ID	Category	Ordinal
0	A	0.0
1	B	1.0
2	A	0.0
3	C	2.0
4	A	0.0

Label Encoding for Ordinal Features

Convert categorical data into numerical labels *(aka "String Indexing")*

Procedure:

1. **Assign numeric labels:** Map each category to a numeric value based on its natural order.
2. **Transform the Feature:** Replace each categorical value in the feature column with its corresponding numeric label.

ID	High School Grade Level	Age
1	Freshman	14
2	Senior	17
3	Junior	16
4	Freshman	15
5	Sophomore	16

ID	High School Grade Level	Age
1	1	14
2	4	17
3	3	16
4	1	15
5	2	16

Example:

String Value	Numeric Value
Freshman	1
Sophomore	2
Junior	3
Senior	4

databricks

Feature Store

Data Preparation for Machine Learning

What is a Feature Store?

A **feature store** manages features, or input data to a machine learning model.

In a model that predicts **customer churn**, for example, features could be:

- Aggregations of raw data over time windows, like **trailing 7-day purchases**
- Joined **combinations of data sets**, like customer demographic information joined to transaction features
- Complex functions of customer information, like **estimated customer lifetime value**

The process of creating these values from data is **feature engineering**.

Why Would You Need a Feature Store?

Basic Motivations

Discovery

Multiple Data Scientists are trying to solve similar modeling tasks and come up with different definitions of the same features. **How can I find the features**?

Lineage

Model governance requires documentation of the features used to train a model, as well as the **upstream lineage** of a feature to reliably use it. **How is it computed, and who owns it**?

Skew

When multiple teams manage feature computation and ML models in production, minor yet significant **skew in upstream data** at the input of a feature pipeline can be very hard to detect and fix.

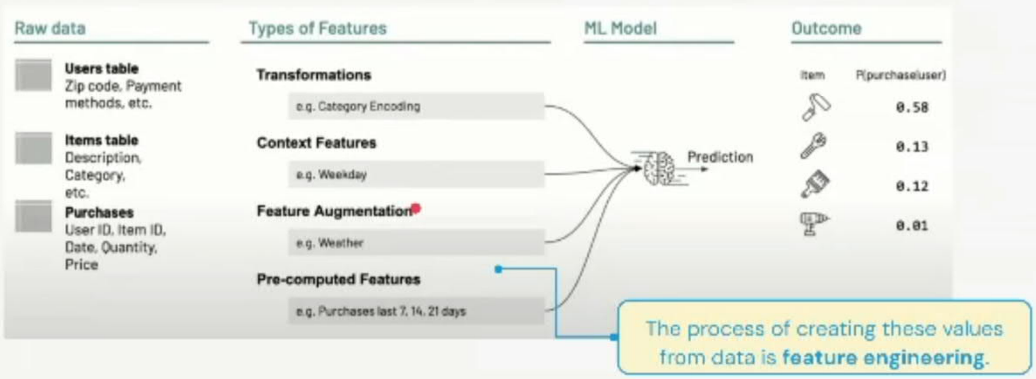
Online Serving

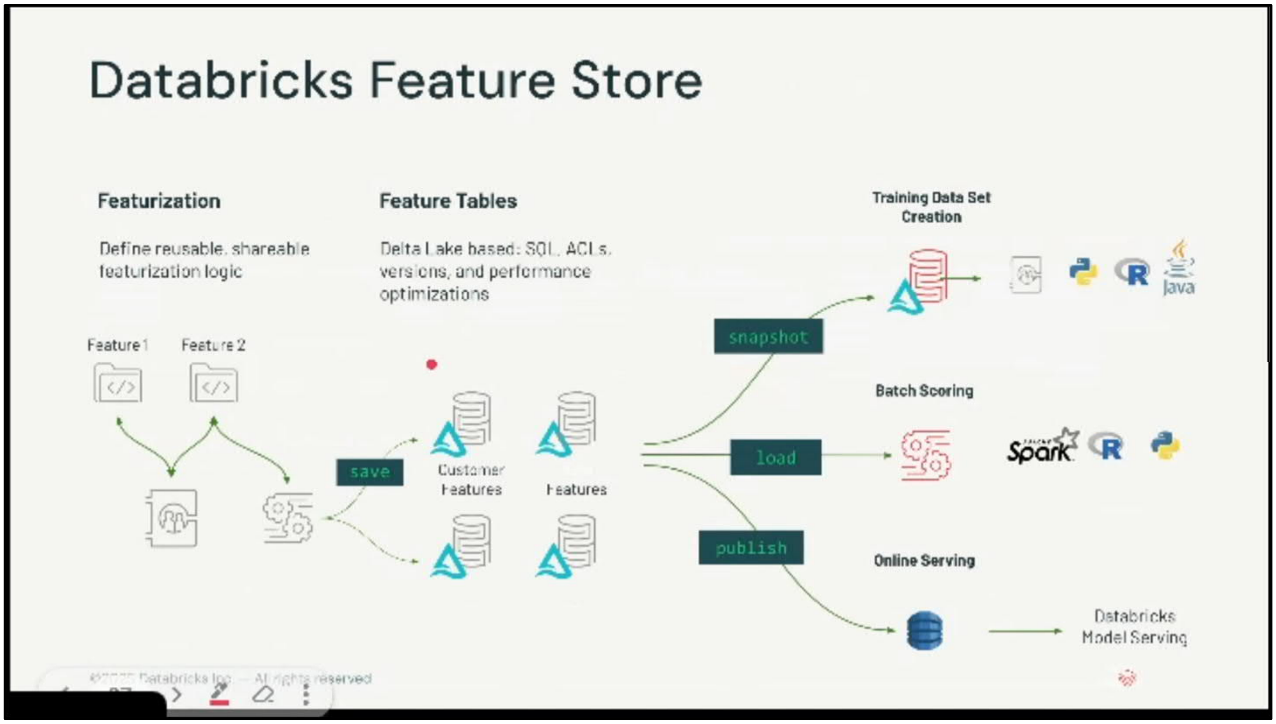
During exploration and model experimentation phases features are implemented in frameworks that do not scale to production.

What is a Feature Store?

An example of a recommendation system.

A **feature store** manages features, or input data to a machine learning model.

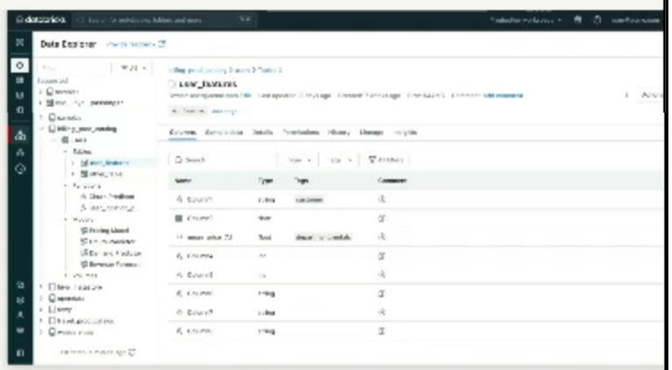



Databricks Feature Store
Featurization
Define reusable, shareable featurization logic
Feature Tables
Delta Lake based: SQL, ACLs, versions, and performance optimizations
Feature 1
Feature 2
save
Customer Features
Features
snapshot
load
publish
Training Data Set Creation
Batch Scoring
Online Serving
Java
Spark
Databricks Model Serving

Complete Integration-FS with Unity Catalog

Any table can be a feature table

- Feature Tables become regular UC Tables with additional metadata.
- Shared properties are unified.
  - Feature table description == table comment.
  - Feature table schema == table schema
- Three-level namespace convention



Feature Engineering Demo

Load Dataset

```
# Load dataset with spark
shared_volume_name = 'telco' # From Marketplace
csv_name = 'telco-customer-churn' # CSV file name
dataset_path = f"{DA.paths.datasets.telco}/{shared_volume_name}/{csv_name}.csv" # Full path
telco_df = spark.read.csv(dataset_path, header="true", inferSchema="true", multiLine="true", escape='"')

# # Drop the target column
telco_df = telco_df.drop("Churn")

# # View dataset
display(telco_df)
```

Create Feature Table

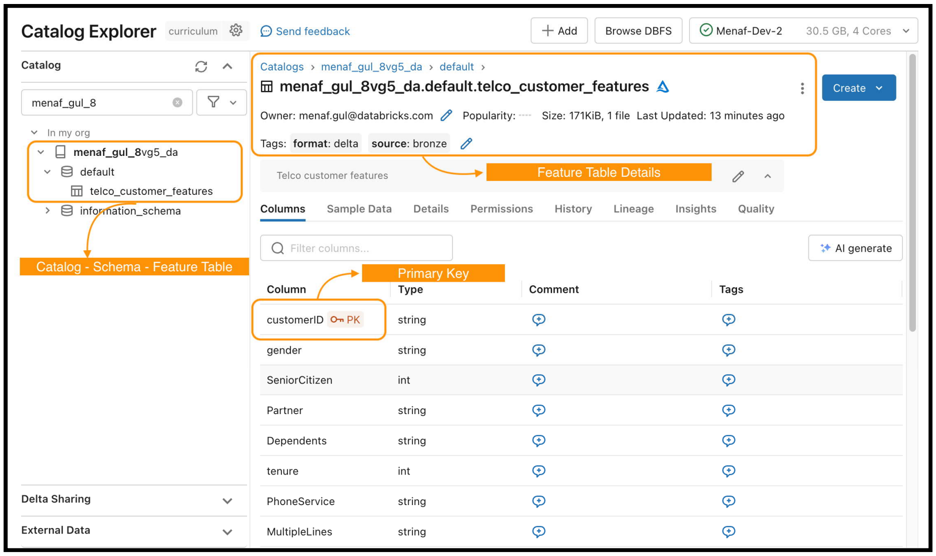
```python
# # create a feature table from the dataset
table_name = f"{DA.catalog_name}.{DA.schema_name}.telco_customer_features"

fe.create_table(
    name=table_name,
    primary_keys=["customerID"],
    df=telco_df,
    #partition_columns=["InternetService"] for small datasets partitioning is not recommended
    description="Telco customer features",
    tags={"source": "bronze", "format": "delta"}
)
```

```python
from databricks.feature_engineering import FeatureEngineeringClient


fe = FeatureEngineeringClient()
```

Explore Feature Table with the UI



Load Feature Table

- We can also look at the metadata of the feature store via the FeatureStore client by using get_table().
- As feature table is a Delta table we can load it with Spark as normally we do for other tables.

```
ft = fe.get_table(name=table_name)
print(f"Feature Table description: {ft.description}")
print(ft.features)
```

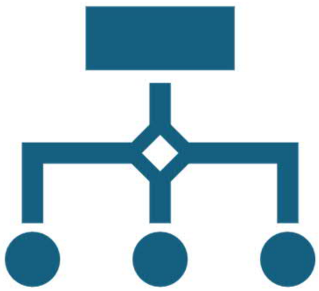
```
Feature Table description: Telco customer features
['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBac
kup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges', 'TotalCharges']
```

20

```
display(fe.read_table(name=table_name))
```

Table

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Int
1	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL
2	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL
3	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL
4	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL



Update Feature Table

- In some cases we might need to update an existing feature table by adding new features or deleting existing features. In this section, we will show to make these type of changes.
- Add a New Feature
    - To illustrate adding a new feature, let's redefine an existing one. In this case, we'll transform the tenure column by categorizing it into three groups: short, mid, and long, representing different tenure durations.
    - Then we will write the dataset back to the feature table. The important parameter is the mode parameter, which we should set to "**merge**".

Update Feature Table

pyspark.sql.DataFrame.withColumn

`DataFrame.withColumn(colName: str, col: pyspark.sql.column.Column) → pyspark.sql.dataframe.DataFrame`

Returns a new `DataFrame` by adding a column or re

The column expression must be an expression over `DataFrame` will raise an error.

```
>>> df = spark.createDataFrame([(2, "Alice"), (5,
>>> df.withColumn('age2', df.age + 2).show()
+---+-----+----+
|age| name|age2|
+---+-----+----+
|  2|Alice|   4|
|  5|  Bob|   7|
+---+-----+----+
```

```python
from pyspark.sql.functions import when

telco_df_updated = telco_df.withColumn("tenure_group",
    when((telco_df.tenure >= 0) & (telco_df.tenure <= 25), "short")
    .when((telco_df.tenure > 25) & (telco_df.tenure <= 50), "mid")
    .when((telco_df.tenure > 50) & (telco_df.tenure <= 75), "long")
    .otherwise("invalid")
)
```

```python
fe.write_table(
    name=table_name,
    df=telco_df_updated.select("customerID","tenure_group"), # primary_key and column to add
    mode="merge"
)
```

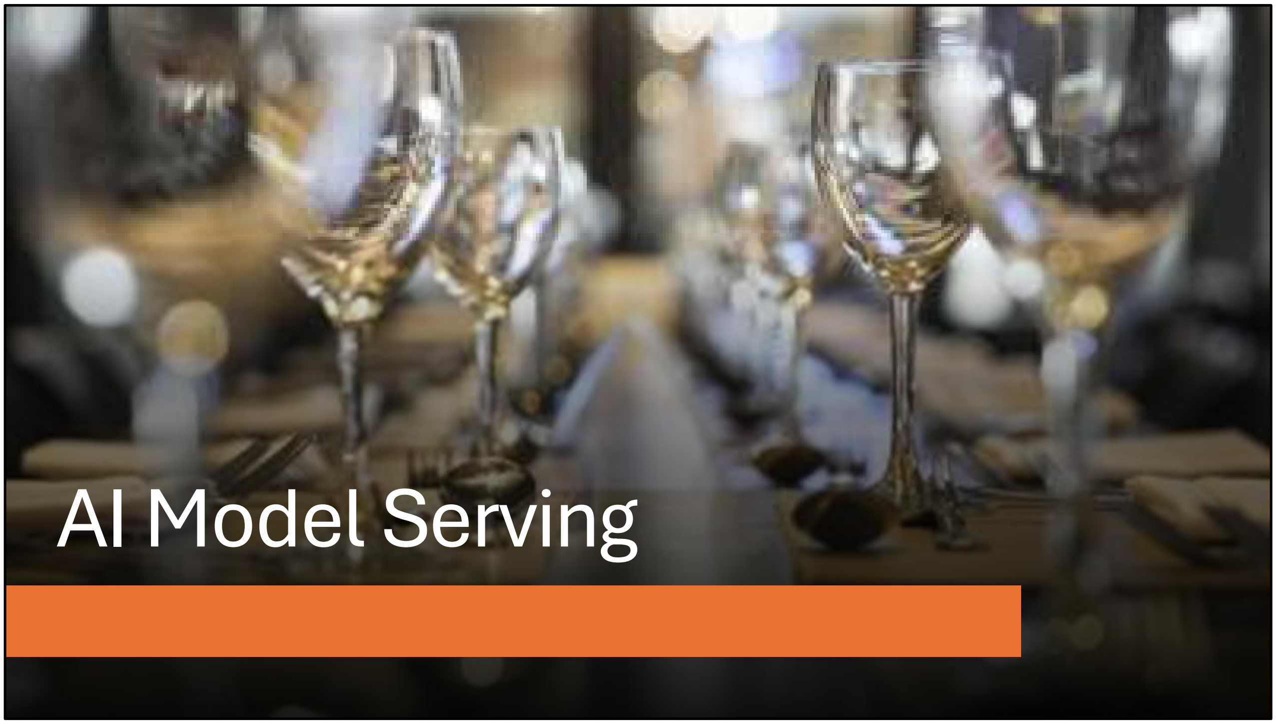
Delete Existing Feature

- To remove a feature column from the table you can just drop the column. Let's drop the original tenure column.
- We need to set Delta read and write protocol version manually to support column mapping.
- Databricks supports column mapping for Delta Lake tables, which enables metadata-only changes to mark columns as deleted or renamed without rewriting data files.
- It also allows users to name Delta table columns using characters that are not allowed by Parquet, such as spaces, so that users can directly inge
due

27

```
%sql
ALTER TABLE telco_customer_features SET TBLPROPERTIES ('delta.columnMapping.mode' = 'name', 'delta.minReaderVersion' = '2', 'delta.minWriterVersion' =
'5');
ALTER TABLE telco_customer_features DROP COLUMNS (tenure)
```

```
OK
```


AI Model Serving

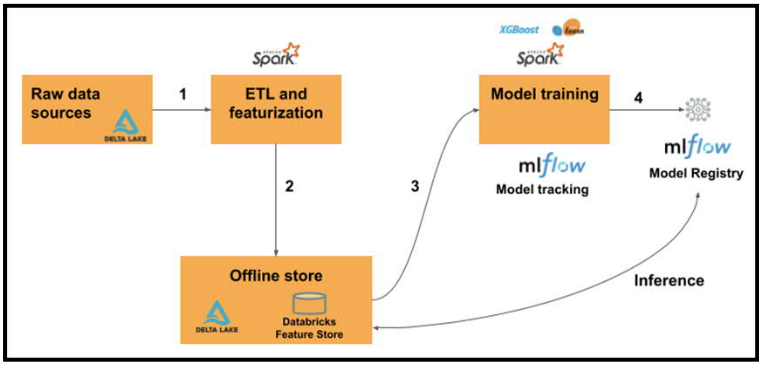
ML workflow using feature engineering

1. Write code to convert raw data into features and create a Spark DataFrame containing the desired features.
2. Create a Delta table in Unity Catalog that has a primary key.
3. Train and log a model using the feature table. When you do this, the model stores the specifications of features used for training.
4. When the model is used for inference, it automatically joins features from the appropriate feature tables.
5. Register model in Model Registry.

https://learn.microsoft.com/en-us/azure/databricks/machine-learning/feature-store/

Batch Use Case

For batch use cases, the model automatically retrieves the features it needs from Feature Store.



Real-time Use Case

For real-time serving use cases, publish the features to an online table.

Third-party online stores are also supported.

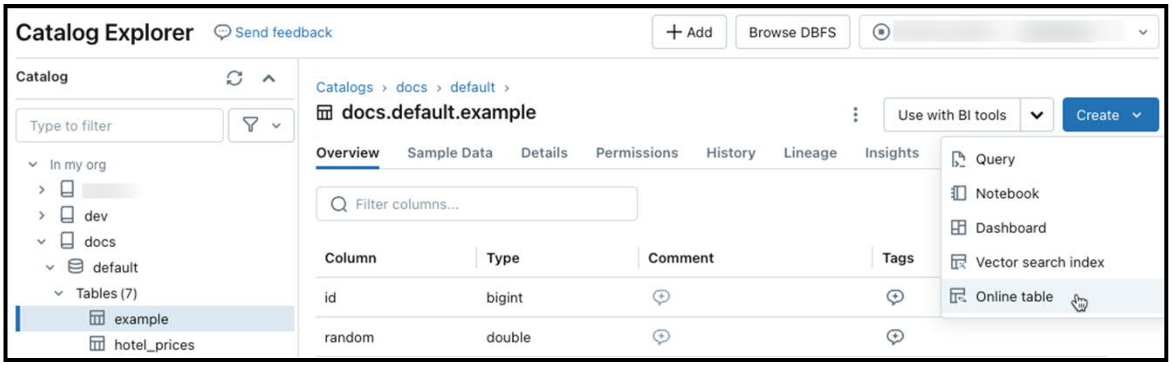
An online table is a read-only copy of a Delta Table that is stored in row-oriented format optimized for online access.

Online tables are fully serverless tables that auto-scale throughput capacity with the request load and provide low latency and high throughput access to data of any scale.

Online tables are designed to work with:

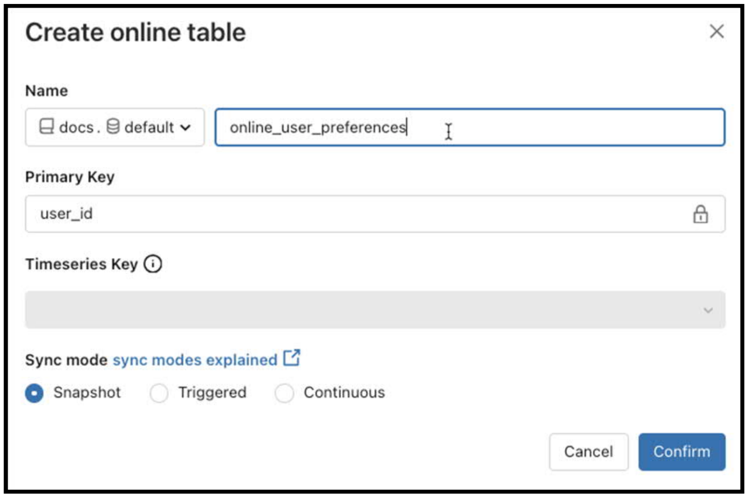
- Mosaic AI Model Serving,
- Feature Serving, and
- Retrieval-augmented generation (RAG) applications where they are used for fast data lookups.

Online Table (Databricks UI)

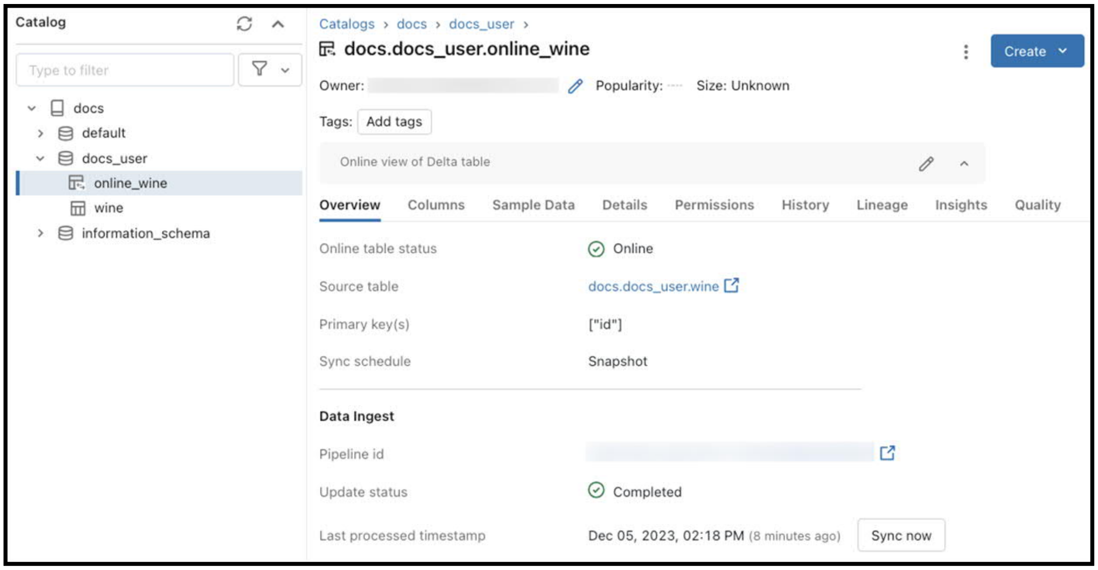


https://learn.microsoft.com/en-us/azure/databricks/machine-learning/feature-store/online-tables#api-sdk

Online Table (Databricks UI)



Online Table (Databricks UI)



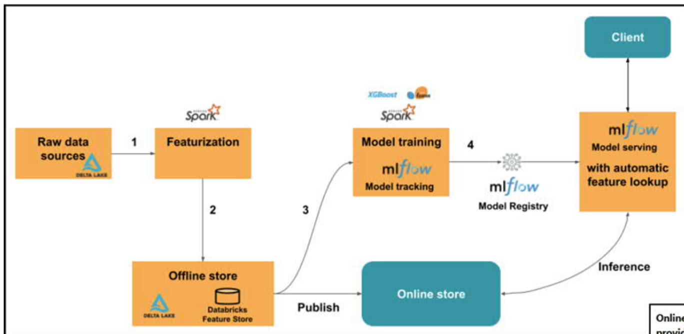
Sync Mode

Policy	Description
Snapshot	The pipeline runs once to take a snapshot of the source table and copy it to the online table. Subsequent changes to the source table are automatically reflected in the online table by taking a new snapshot of the source and creating a new copy. The content of the online table is updated atomically.
Triggered	The pipeline runs once to create an initial snapshot copy of the source table in the online table. Unlike the Snapshot sync mode, when the online table is refreshed, only changes since the last pipeline execution are retrieved and applied to the online table. The incremental refresh can be manually triggered or automatically triggered according to a schedule.
Continuous	The pipeline runs continuously. Subsequent changes to the source table are incrementally applied to the online table in real time streaming mode. No manual refresh is necessary.

ⓘ **Note**

To support **Triggered** or **Continuous** sync mode, the source table must have Change data feed enabled.

3rd Party Online Tables



Online store provider	Publish with Feature Engineering in Unity Catalog	Publish with legacy Workspace Feature Store	Feature lookup in Legacy MLflow Model Serving	Feature lookup in Model Serving
Azure Cosmos DB [1]	X	X (Feature Store client v0.5.0 and above)	X	X
Azure MySQL (Single Server)		X	X	
Azure SQL Server		X		

https://learn.microsoft.com/en-us/azure/databricks/machine-learning/feature-store/publish-features

3rd Party Online Tables

```python
import datetime
from databricks.feature_engineering.online_store_spec import AzureMySqlSpec
# or databricks.feature_store.online_store_spec for Workspace Feature Store
online_store = AzureMySqlSpec(
  hostname='<hostname>',
  port='<port>',
  read_secret_prefix='<read-scope>/<prefix>',
  write_secret_prefix='<write-scope>/<prefix>'
)

fs.publish_table(
  name='recommender_system.customer_features',
  online_store=online_store,
  filter_condition=f"_dt = '{str(datetime.date.today())}'",
  mode='merge'
)
```

```python
import datetime
from databricks.feature_engineering.online_store_spec import AzureCosmosDBSpec
# or databricks.feature_store.online_store_spec for Workspace Feature Store
online_store = AzureCosmosDBSpec(
  account_uri='<account-uri>',
  read_secret_prefix='<read-scope>/<prefix>',
  write_secret_prefix='<write-scope>/<prefix>'
)

fe.publish_table( # or fs.publish_table for Workspace Feature Store
  name='ml.recommender_system.customer_features',
  online_store=online_store,
  filter_condition=f"_dt = '{str(datetime.date.today())}'",
  mode='merge'
)
```

Mosaic AI Model Serving

- Mosaic AI Model Serving provides a unified interface to deploy, govern, and query AI models for real-time and batch inference.
- Each model you serve is available as a REST API that you can integrate into your web or client application.
- Model Serving provides a highly available and low-latency service for deploying models.
- The service automatically scales up or down to meet demand changes, saving infrastructure costs while optimizing latency performance.
  - This functionality uses serverless compute.
- It also provides a single UI to manage all your models and their respective serving endpoints. You can also access models directly from SQL using AI Functions for easy integration into analytics workflows.

https://docs.databricks.com/aws/en/machine-learning/model-serving/
https://docs.databricks.com/aws/en/large-language-models/ai-functions

Why use Model Serving

- **Deploy and query any models**: Model Serving provides a unified interface that so you can manage all models in one location and query them with a single API, regardless of whether they are hosted on Databricks or externally.
  - This approach simplifies the process of experimenting with, customizing, and deploying models in production across various clouds and providers.
- **Securely customize models with your private data**: Built on a Data Intelligence Platform, Model Serving simplifies the integration of features and embeddings into models through native integration with the Databricks Feature Store and Mosaic AI Vector Search.
  - For even more improved accuracy and contextual understanding, models can be fine-tuned with proprietary data and deployed effortlessly on Model Serving.

https://docs.databricks.com/aws/en/machine-learning/model-serving/
https://learn.microsoft.com/en-us/azure/databricks/resources/feature-region-support#azure-model-serving

Mosaic AI Gateway

- Mosaic AI Gateway is designed to streamline the usage and management of generative AI models and agents within an organization.
  - It is a centralized service that brings governance, monitoring, and production readiness to model serving endpoints.
  - It also allows you to run, secure, and govern AI traffic to democratize and accelerate AI adoption for your organization.

https://learn.microsoft.com/en-us/azure/databricks/resources/feature-region-support#azure-model-serving

AI Gateway Features

Feature	Definition
Permission and rate limiting	Control who has access and how much access.
Payload logging	Monitor and audit data being sent to model APIs using inference tables.
Usage tracking	Monitor operational usage on endpoints and associated costs using system tables.
AI Guardrails	Prevent unwanted and unsafe data in requests and responses. See AI Guardrails.
Fallbacks	Minimize production outages during and after deployment.
Traffic splitting	Load balance traffic across models.

https://docs.databricks.com/aws/en/ai-gateway/