databricks

Data Preparation for Machine Learning

Databricks Academy

Learning goals

Upon completion of this content, you should be able to:

- Describe Databricks Data Intelligence Platform and its features for machine learning.
- Explain data storage and governance features on Databricks.
- Perform exploratory data analysis and feature engineering using Spark and integrated visualization tools.
- Perform data pre-processing for missing data handling, data encoding and data standardization.
- Utilize Feature Store for storing and retrieving features.

Prerequisites/Technical Considerations

Things to keep in mind before you work through this course

Prerequisites

1. Familiarity with Databricks workspace and notebooks
2. Familiarity with Delta Lake and Lakehouse
3. Intermediate level knowledge of Python

Technical Considerations

1. Cluster running on **DBR ML 16.3**
2. Unity Catalog enabled workspace

AGENDA

1. Managing and Exploring Data	DEMO	LAB
Managing and Exploring Data in the Lakehouse	✓	✓
02. Data Preparation and Feature Engineering		
Fundamentals of Data Preparation and Feature Engineering		
Data Imputation	✓	✓
Data Encoding		
Data Standardization		
03. Feature Store		
Introduction to Feature Store	✓	✓

databricks

Managing and Exploring Data

Data Preparation for Machine Learning

Learning objectives

Things you'll be able to do after completing this module

- Understand Databricks Machine Learning as a data-centric platform.
- Explain the benefits of Delta Lake for machine learning, including version control and data integrity.
- Describe the benefits of Unity Catalog for machine learning, including data discovery and collaboration.
- Load and write data from/to Delta tables.
- Use data profiling and visualization to explore and analyze machine learning data.

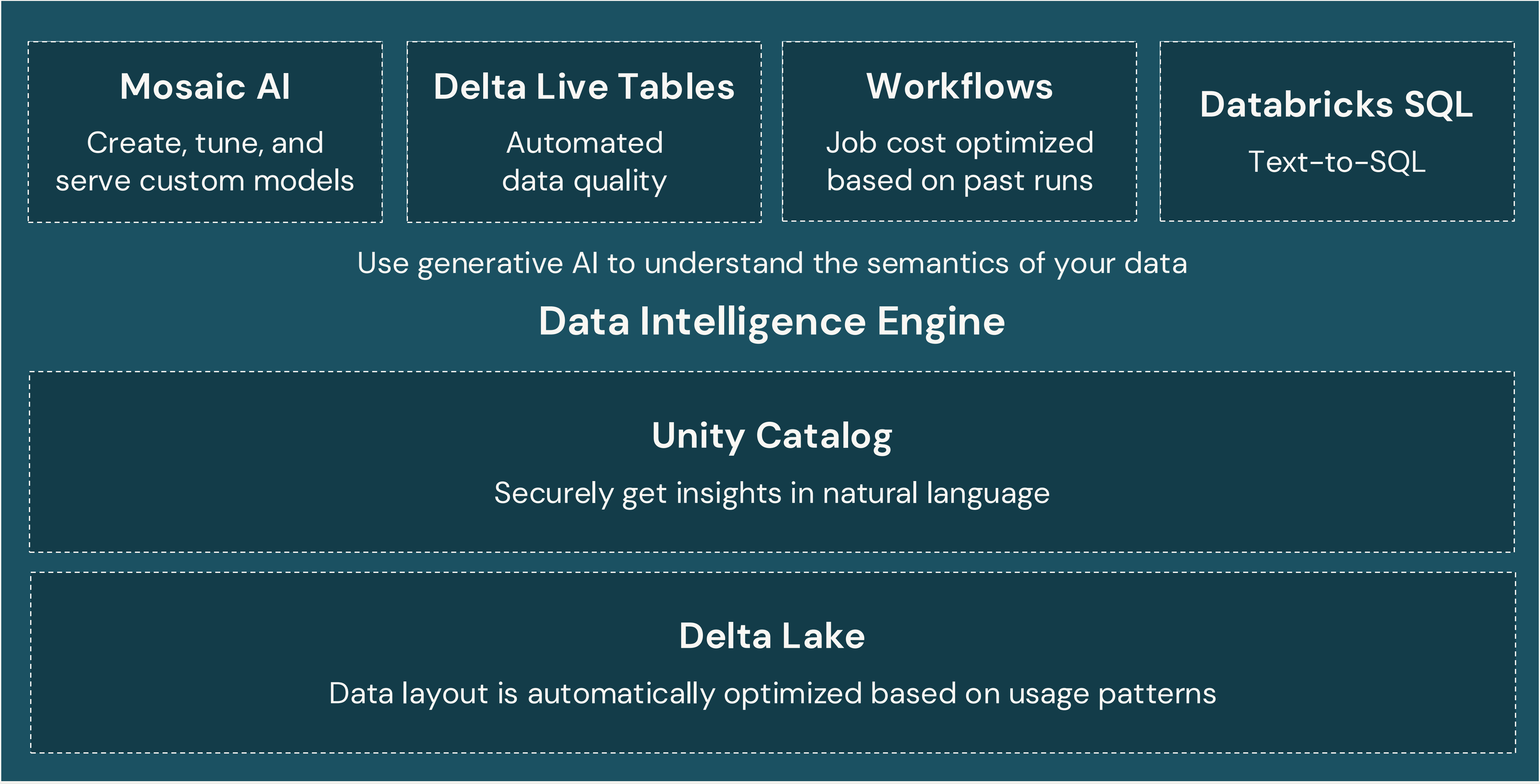
databricks

Managing and Exploring Data

LECTURE

Managing and Exploring Data in the LakeHouse

Databricks Data Intelligence Platform



Mosaic AI

Create, tune, and serve custom models

Delta Live Tables

Automated data quality

Workflows

Job cost optimized based on past runs

Databricks SQL

Text-to-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)

Data Science & AI on Databricks

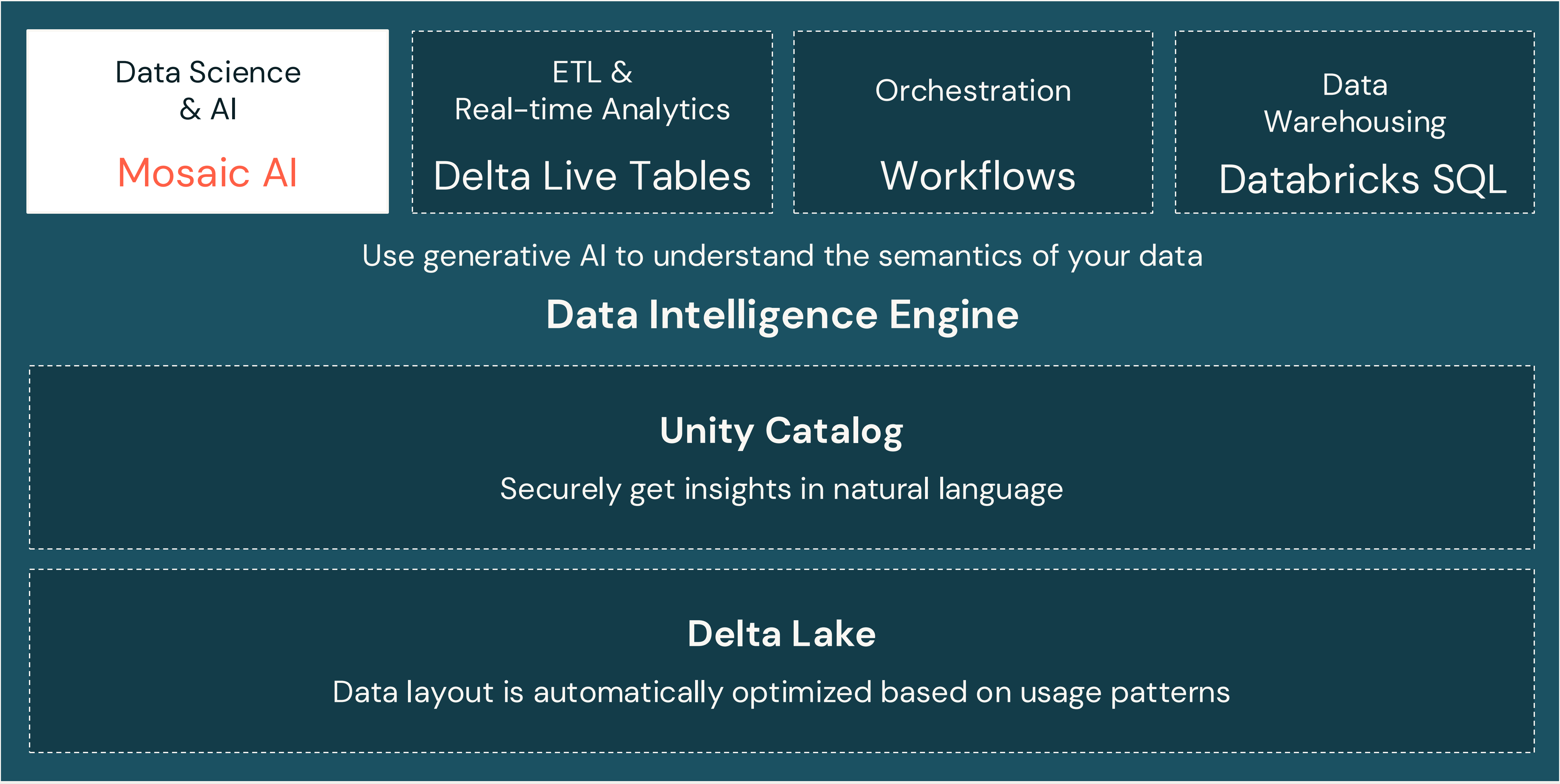
Mosaic AI

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

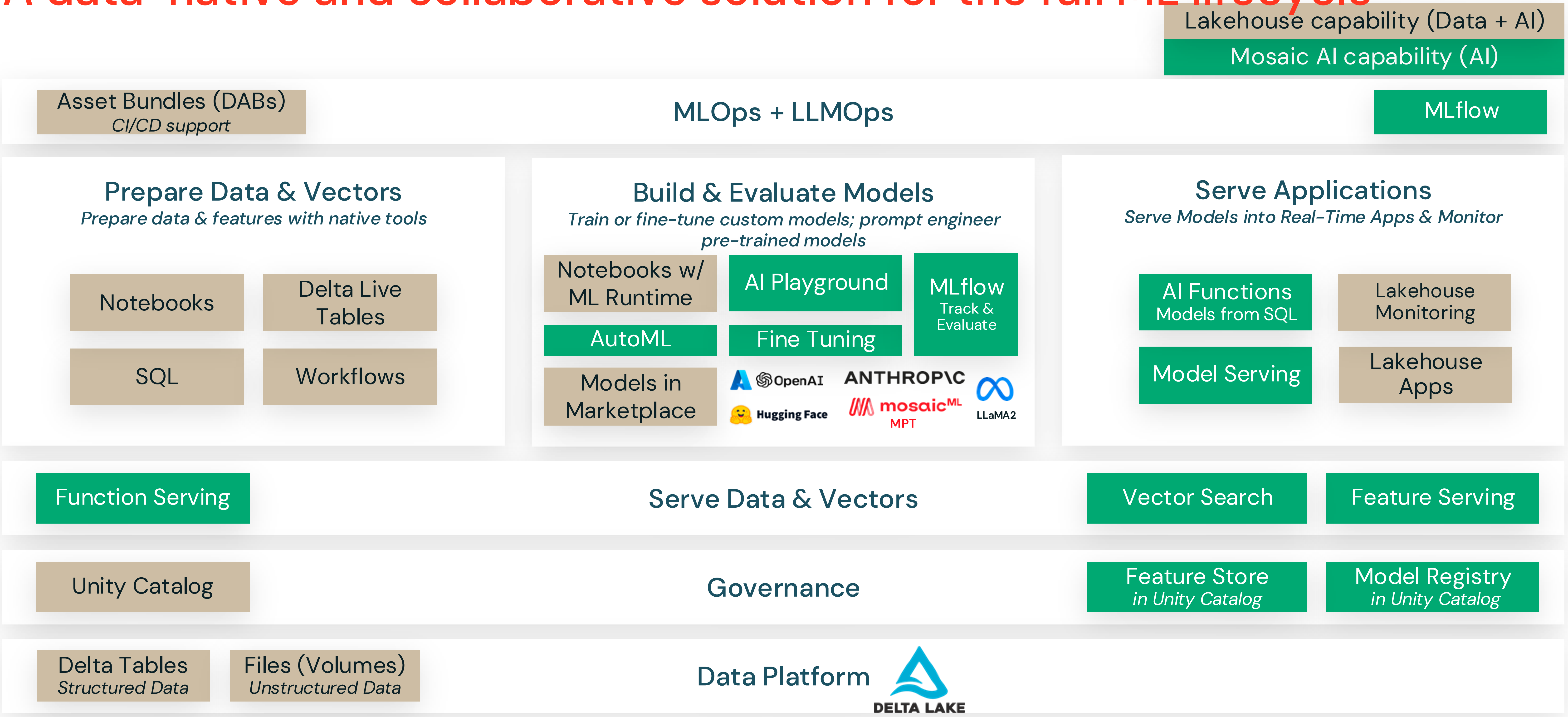
Data layout is automatically optimized based on usage patterns

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)

MLOps + LLMOps

Asset Bundles (DABs) *CI/CD support*

MLflow

Prepare Data & Vectors

Prepare data & features with native tools

Notebooks

Delta Live Tables

SQL

Workflows

Build & Evaluate Models

Train or fine-tune custom models; prompt engineer pre-trained models

Notebooks w/ ML Runtime

AI Playground

MLflow Track & Evaluate

AutoML

Fine Tuning

Models in Marketplace

OpenAI

ANTHROP\C

Hugging Face

mosaicML MPT

LLaMA2

Serve Applications

Serve Models into Real-Time Apps & Monitor

AI Functions Models from SQL

Lakehouse Monitoring

Model Serving

Lakehouse Apps

Serve Data & Vectors

Function Serving

Vector Search

Feature Serving

Governance

Unity Catalog

Feature Store *in Unity Catalog*

Model Registry *in Unity Catalog*

Data Platform

Delta Tables *Structured Data*

Files (Volumes) *Unstructured Data*

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

Collaborative Multi-Language Notebooks

Collaborative, reproducible, and enterprise ready

Multi-Language

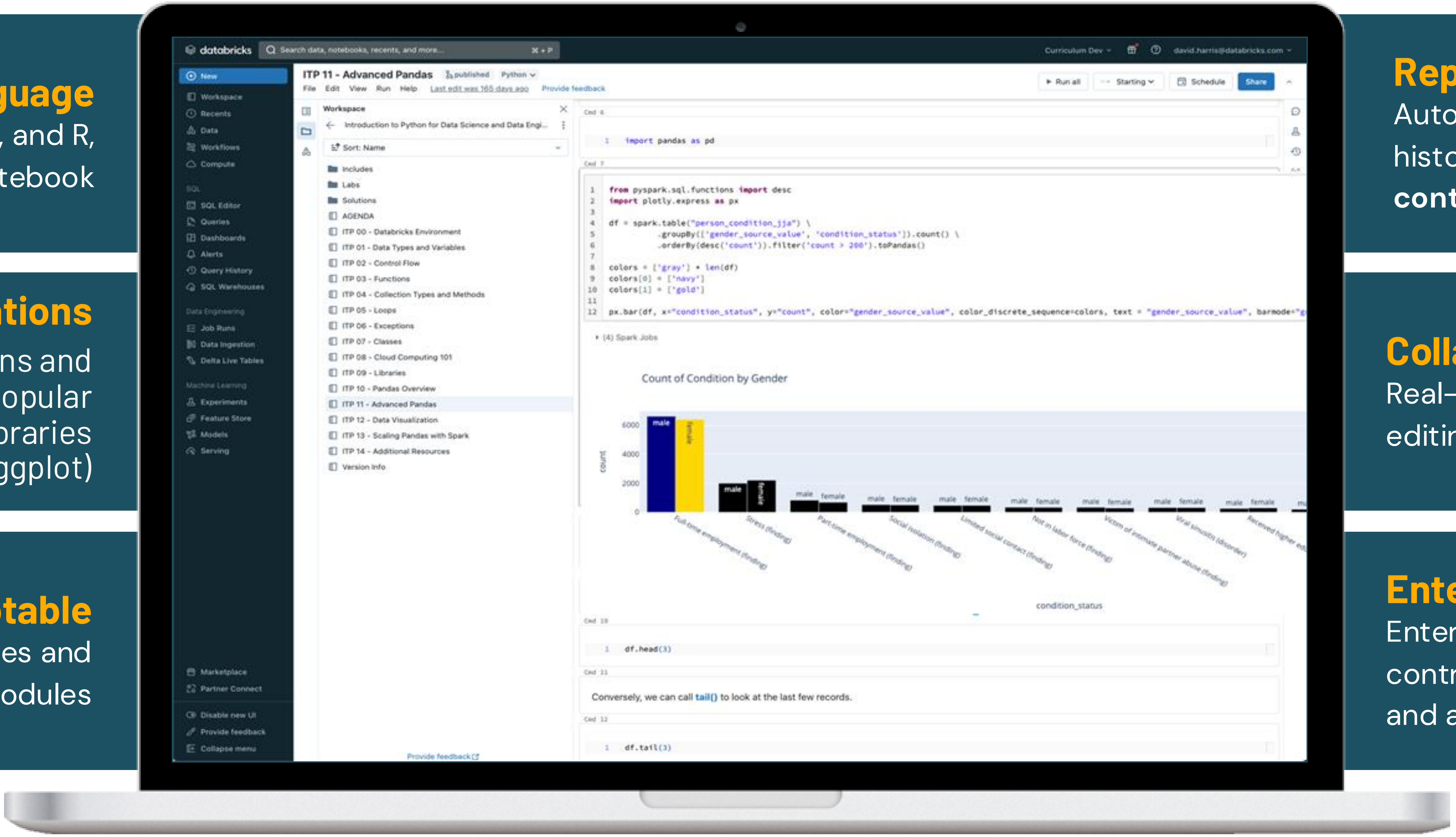
Use Python, SQL, Scala, and R, all in one Notebook

Visualizations

Built-in visualizations and support for the most popular visualization libraries (e.g. matplotlib, ggplot)

Adaptable

Install standard libraries and use local modules


databricks
Search data, notebooks, recents, and more...
ITP 11 - Advanced Pandas
Python
Run all
Starting
Schedule
Share
Workspace
Introduction to Python for Data Science and Data Engi...
Includes
Labs
Solutions
AGENDA
ITP 00 - Databricks Environment
ITP 01 - Data Types and Variables
ITP 02 - Control Flow
ITP 03 - Functions
ITP 04 - Collection Types and Methods
ITP 05 - Loops
ITP 06 - Exceptions
ITP 07 - Classes
ITP 08 - Cloud Computing 101
ITP 09 - Libraries
ITP 10 - Pandas Overview
ITP 11 - Advanced Pandas
ITP 12 - Data Visualization
ITP 13 - Scaling Pandas with Spark
ITP 14 - Additional Resources
Version Info
Count of Condition by Gender
count
condition_status
male
female

```
import pandas as pd
```

```
from pyspark.sql.functions import desc
import plotly.express as px

df = spark.table("person_condition_jja") \
          .groupBy(['gender_source_value', 'condition_status']).count() \
          .orderBy(desc('count')).filter('count > 200').toPandas()

colors = ['gray'] * len(df)
colors[0] = ['navy']
colors[1] = ['gold']

px.bar(df, x="condition_status", y="count", color="gender_source_value", color_discrete_sequence=colors, text = "gender_source_value", barmode="g
```

```
df.head(3)
```

Conversely, we can call tail() to look at the last few records.

```
df.tail(3)
```

Reproducible

Automatically track version history, and use **git version control** with Repos

Collaborative

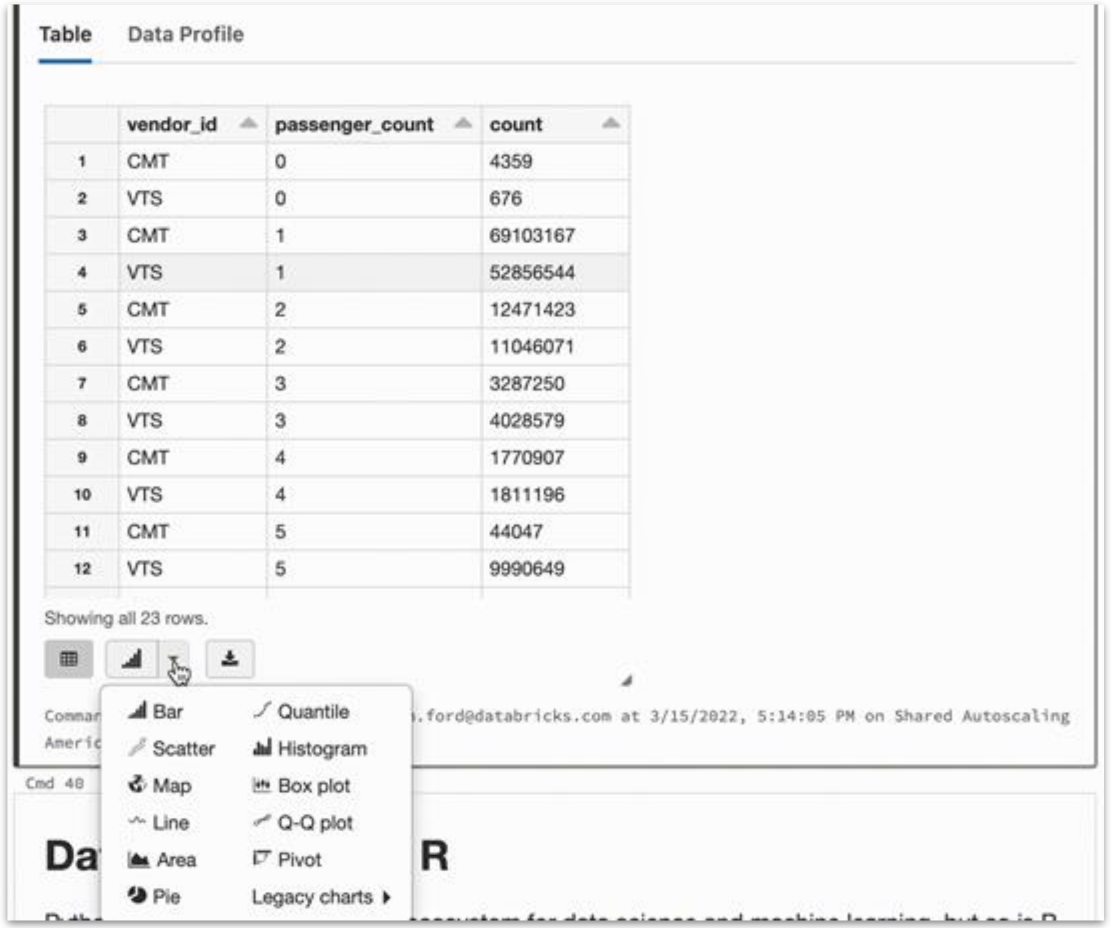
Real-time co-presence, co-editing, and commenting

Enterprise Ready

Enterprise-grade access controls, identity management, and auditability

Quick Exploratory Data Analysis

Native tools for visualizing and understanding data in ML workflow



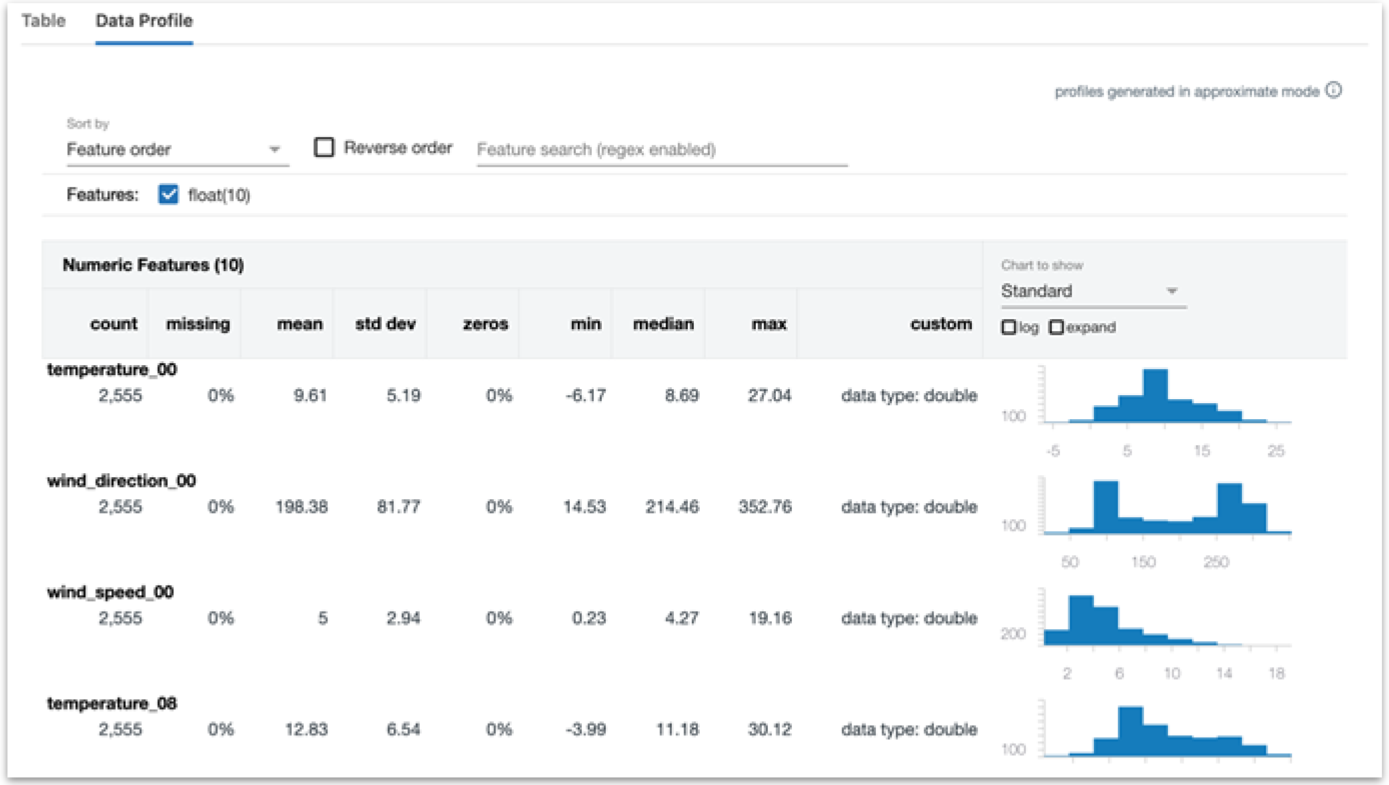
	vendor_id	passenger_count	count
1	CMT	0	4359
2	VTS	0	676
3	CMT	1	69103167
4	VTS	1	52856544
5	CMT	2	12471423
6	VTS	2	11046071
7	CMT	3	3287250
8	VTS	3	4028579
9	CMT	4	1770907
10	VTS	4	1811196
11	CMT	5	44047
12	VTS	5	9990649

Showing all 23 rows.

```
Comman .ford@databricks.com at 3/15/2022, 5:14:05 PM on Shared Autoscaling
Americ
Cmd 40
```

Da R

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



profiles generated in approximate mode ⓘ

Sort by: Feature order

□ Reverse order

Feature search (regex enabled)

Features: ☑ float(10)

Chart to show: Standard

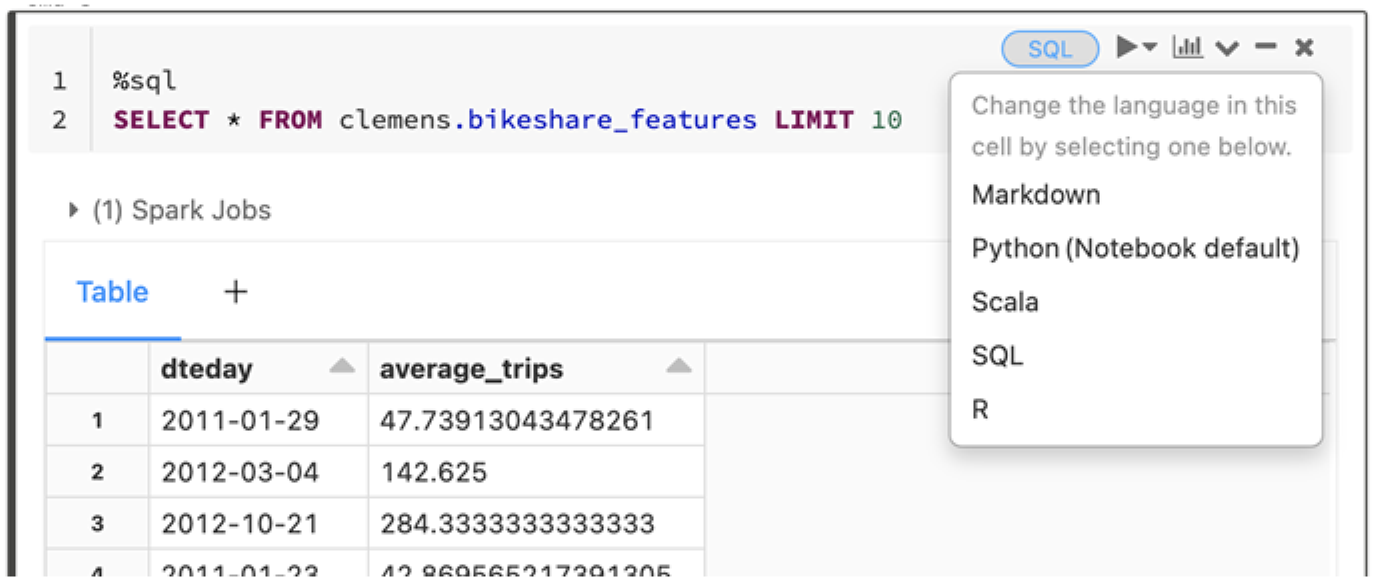
□ log □ expand

Numeric Features (10)	count	missing	mean	std dev	zeros	min	median	max	custom
temperature_00	2,555	0%	9.61	5.19	0%	-6.17	8.69	27.04	data type: double
wind_direction_00	2,555	0%	198.38	81.77	0%	14.53	214.46	352.76	data type: double
wind_speed_00	2,555	0%	5	2.94	0%	0.23	4.27	19.16	data type: double
temperature_08	2,555	0%	12.83	6.54	0%	-3.99	11.18	30.12	data type: double

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

Tools for Quick ML Model Development

Multi-language support, use standard libraries and custom modules

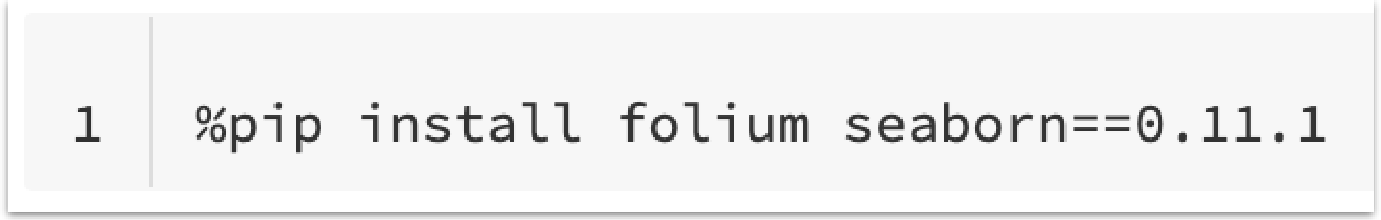


```
%sql
SELECT * FROM clemens.bikeshare_features LIMIT 10
```

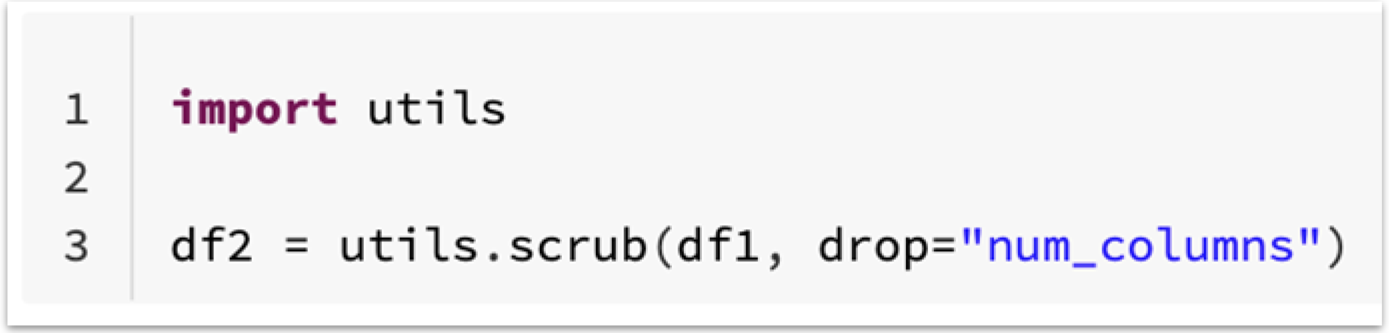
Change the language in this cell by selecting one below.

	dteday	average_trips
1	2011-01-29	47.73913043478261
2	2012-03-04	142.625
3	2012-10-21	284.3333333333333
4	2011-01-23	42.869565217391305

Mix and match languages based on use case and preferred workflow, choosing from **Python, SQL, Scala, and R**



```
%pip install folium seaborn==0.11.1
```



```
import utils

df2 = utils.scrub(df1, drop="num_columns")
```

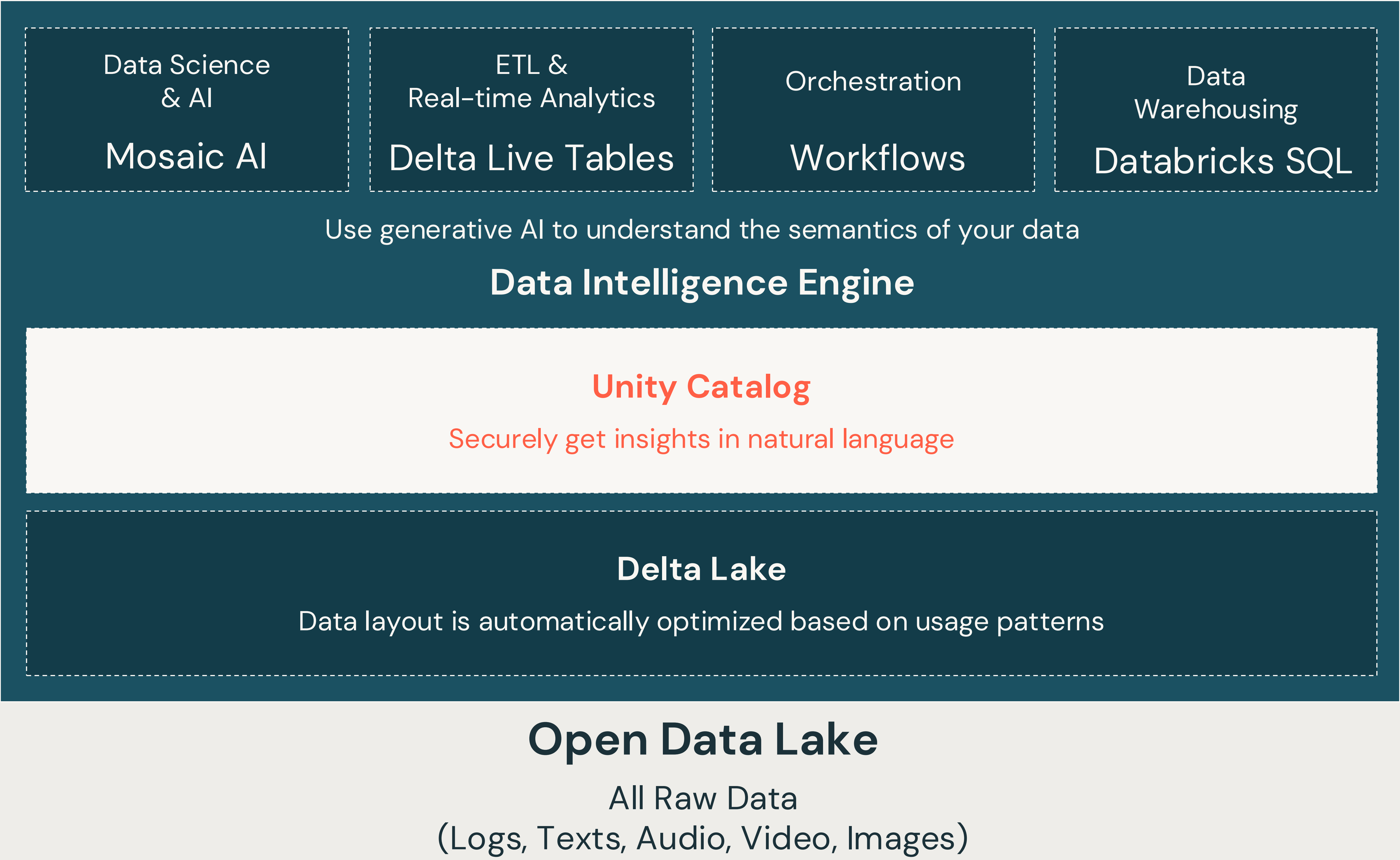
Install Python libraries for a notebook without affecting other users with *%pip*

Import local modules using **arbitrary file support** when working in Repos

Data Science & AI on Databricks

Unity Catalog

- Context-aware search
- Auto describe tables and columns
- Automated lineage
- End-to-end observability and monitoring
- Sharing AI models



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

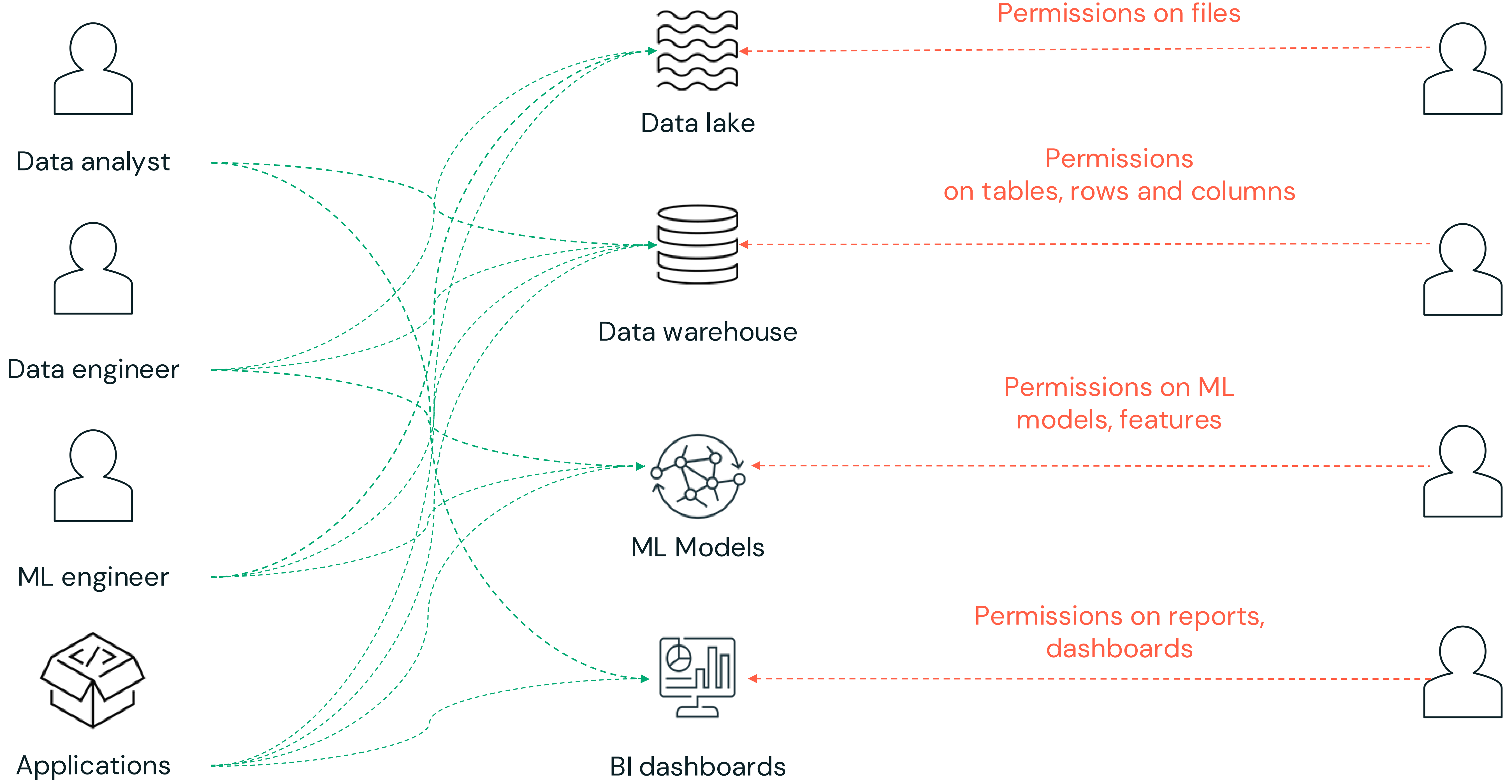
All Raw Data
(Logs, Texts, Audio, Video, Images)

Today, data and AI governance is complex

Data Consumers

“Where to **discover** the datasets, models, notebooks, dashboards?”

“Can I **trust** the data and ML models?”



Data Governance Team

“How to **secure** these assets?”

“Who is **accessing** these assets and how?”

“Are we meeting the regulatory **compliance**?”

Unity Catalog (UC)

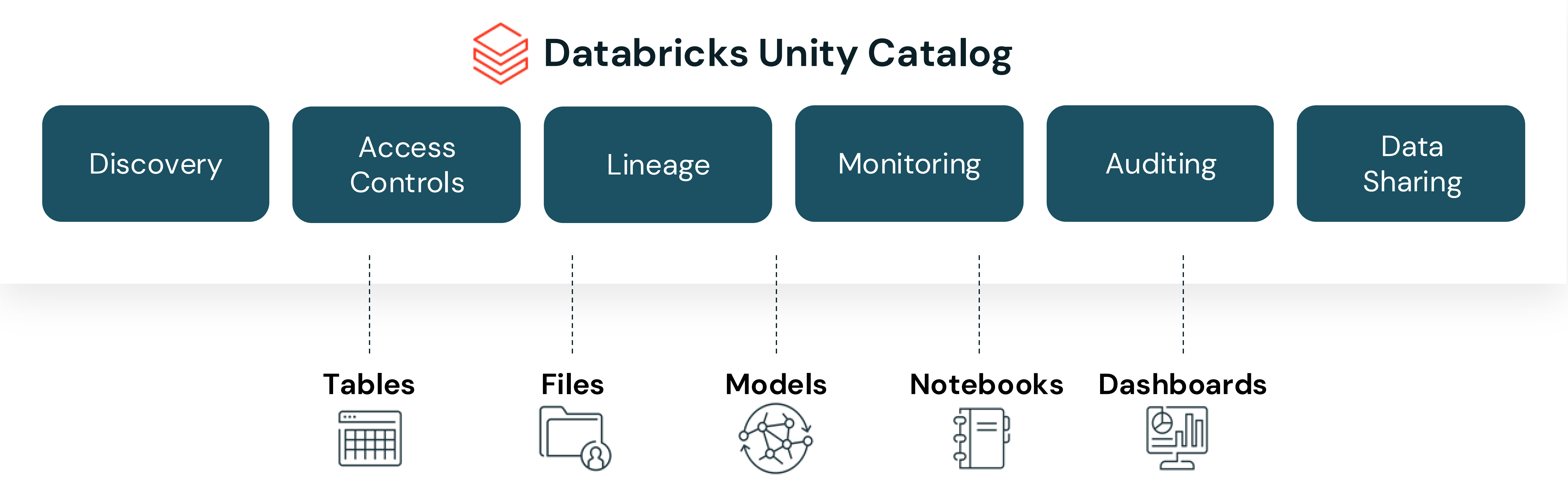
Unified governance for AI; data, code, models

Unified visibility into data and AI

Single permission model for data and AI

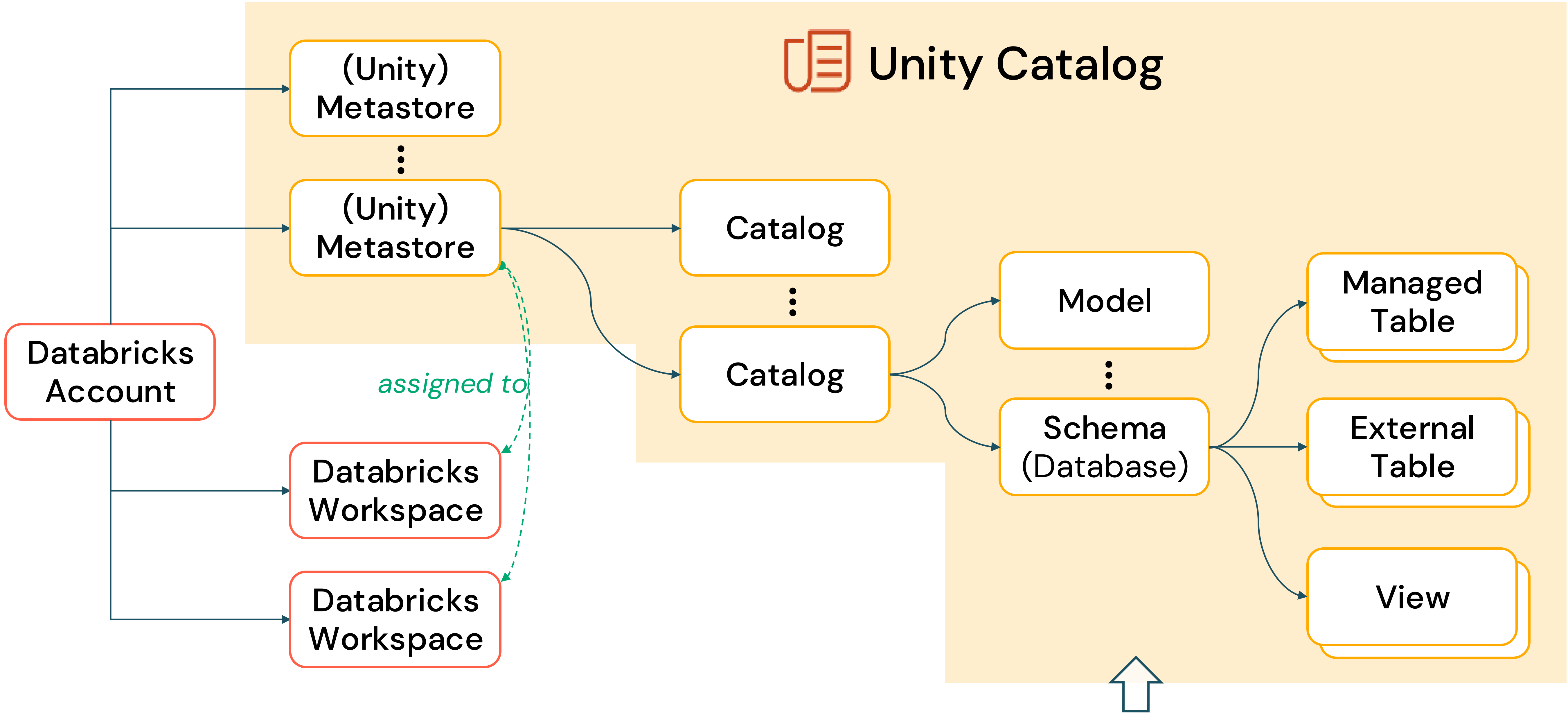
AI-powered monitoring and observability

Open data sharing



The 3-level Namespace of UC

How to use UC

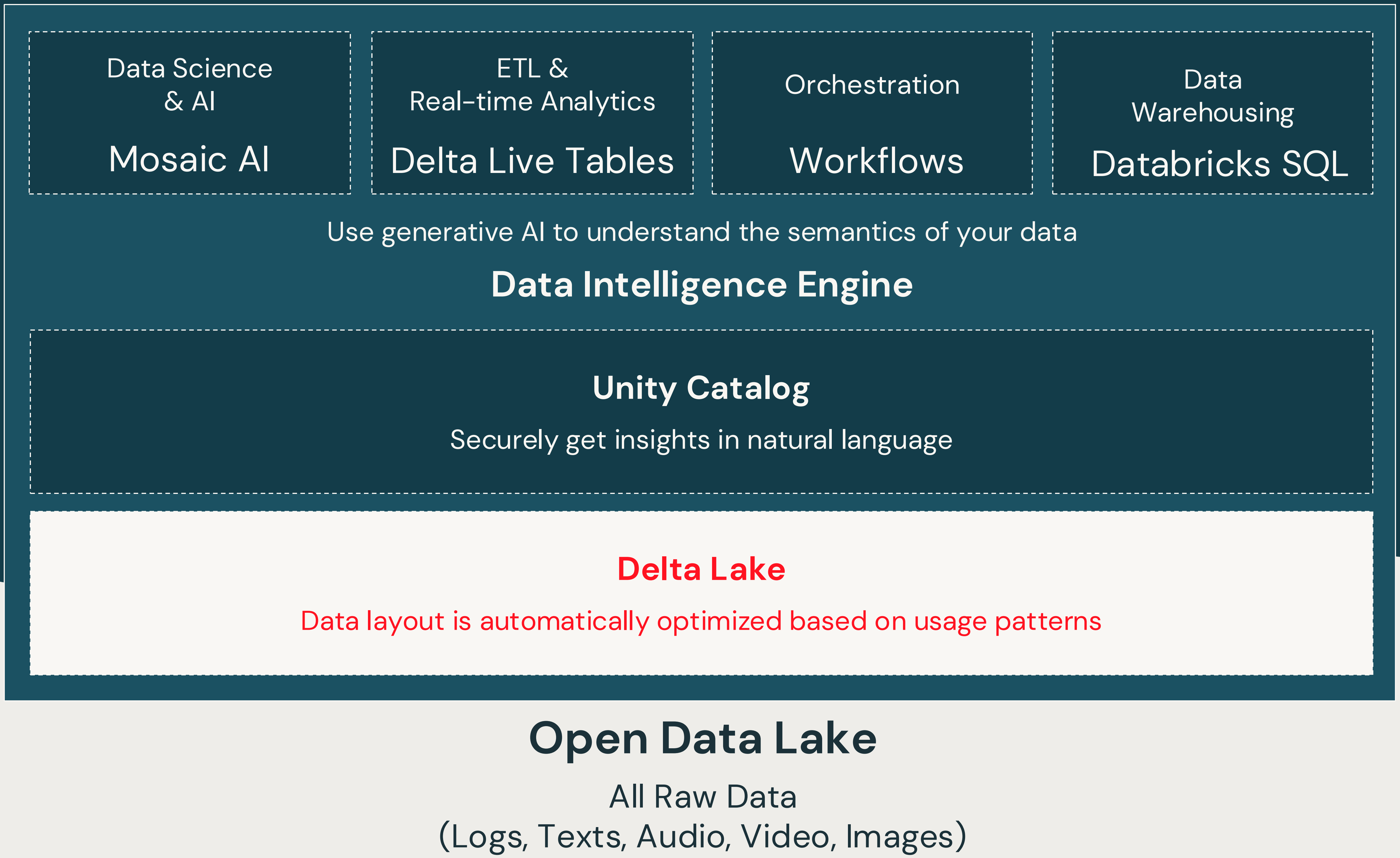


```
SELECT * FROM catalog1.database1.table1;
```

Data Science & AI on Databricks

Delta Lake

- Open-source
- Unified data management layer
- Reliable and fast
- Optimization features for storing data in the cloud



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

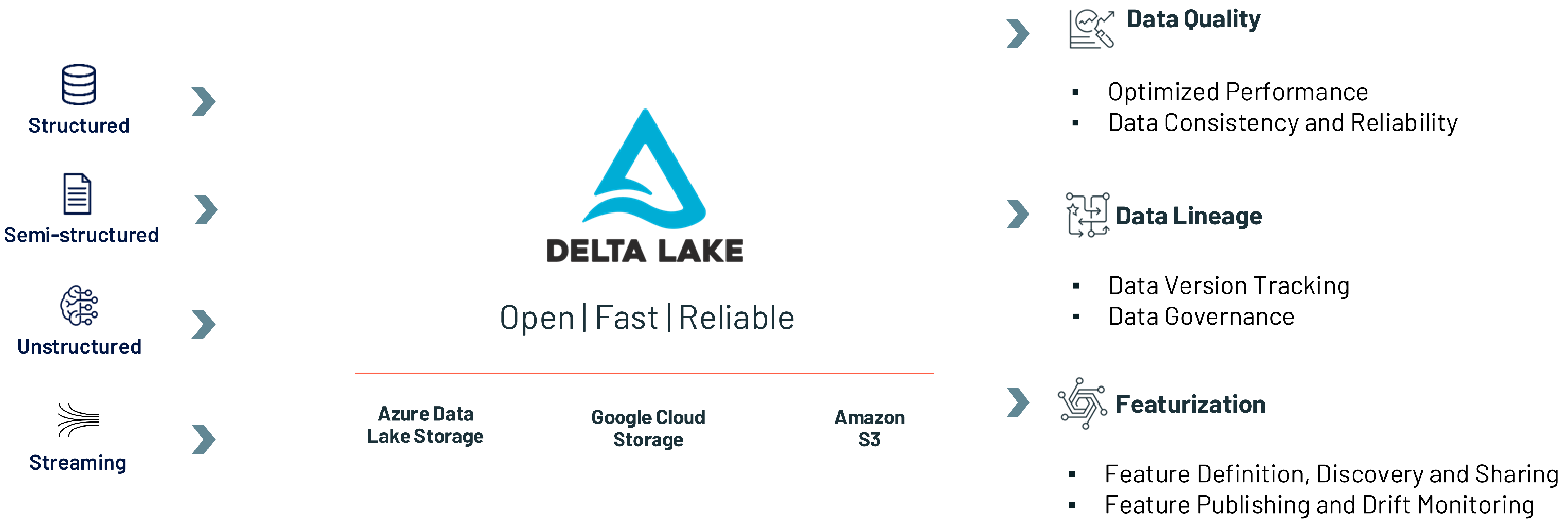
Data layout is automatically optimized based on usage patterns

Open Data Lake

All Raw Data
(Logs, Texts, Audio, Video, Images)

What is Delta Lake?

- A **unified data management layer** that brings data reliability and fast analytics to cloud data lakes. It is the optimized storage layer that provides the foundation for storing data and tables in the Databricks DI Platform.



Structured

Semi-structured

Unstructured

Streaming

DELTA LAKE

Open | Fast | Reliable

Azure Data Lake Storage

Google Cloud Storage

Amazon S3

Data Quality

- Optimized Performance
- Data Consistency and Reliability

Data Lineage

- Data Version Tracking
- Data Governance

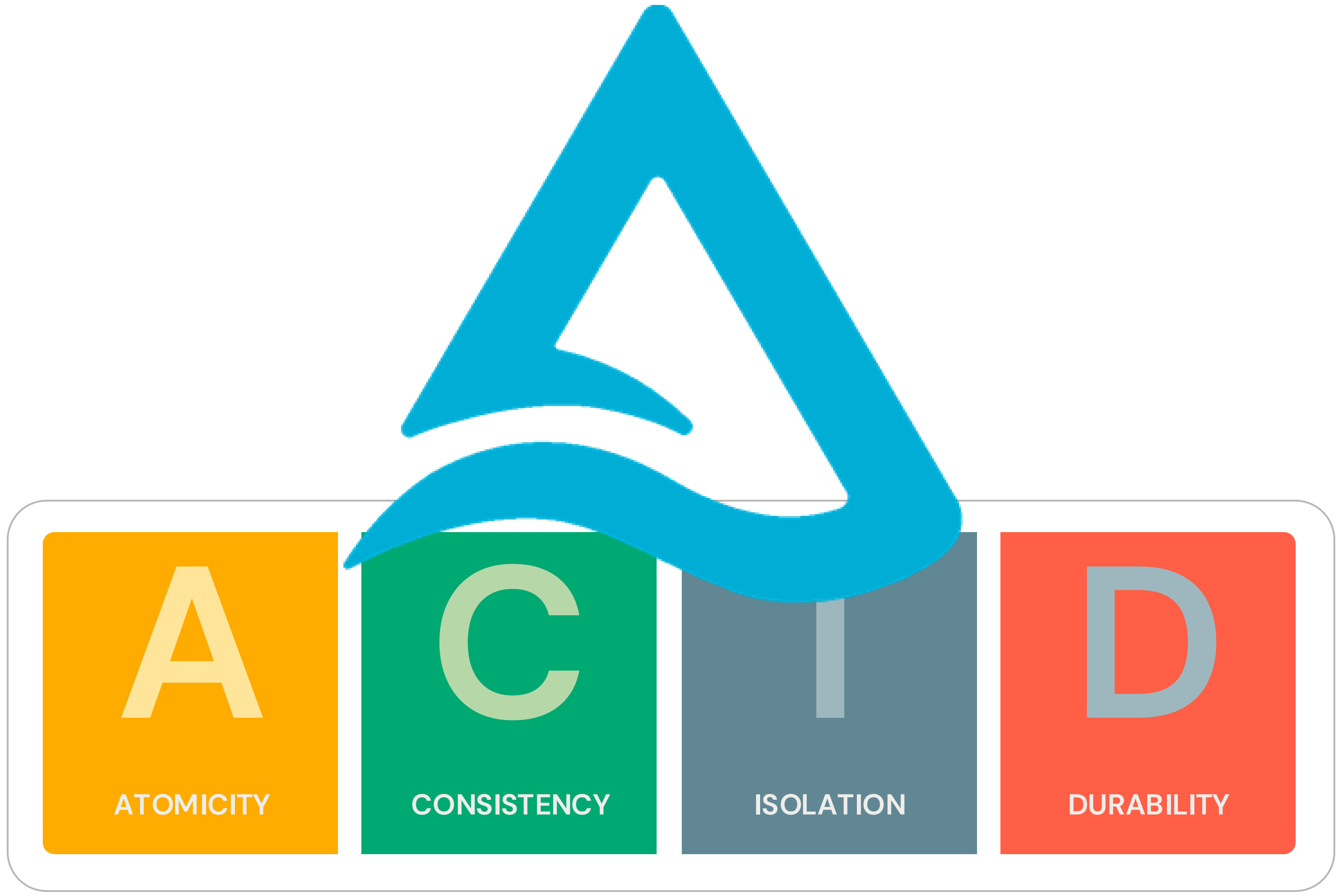
Featurization

- Feature Definition, Discovery and Sharing
- Feature Publishing and Drift Monitoring

Delta Lake and Its Features

Open-source, default storage format on Databricks

- Delta Lake is an **open-source** project.
- It is the **default format** for the tables created in Databricks.
- Delta Lake **optimizes performance** with large datasets, providing **ACID transactions** and **scalable metadata handling**.
- Designed to improve data reliability, quality, and performance in data lakes.



A ATOMICITY

C CONSISTENCY

I ISOLATION

D DURABILITY

Delta Lake features

Key features

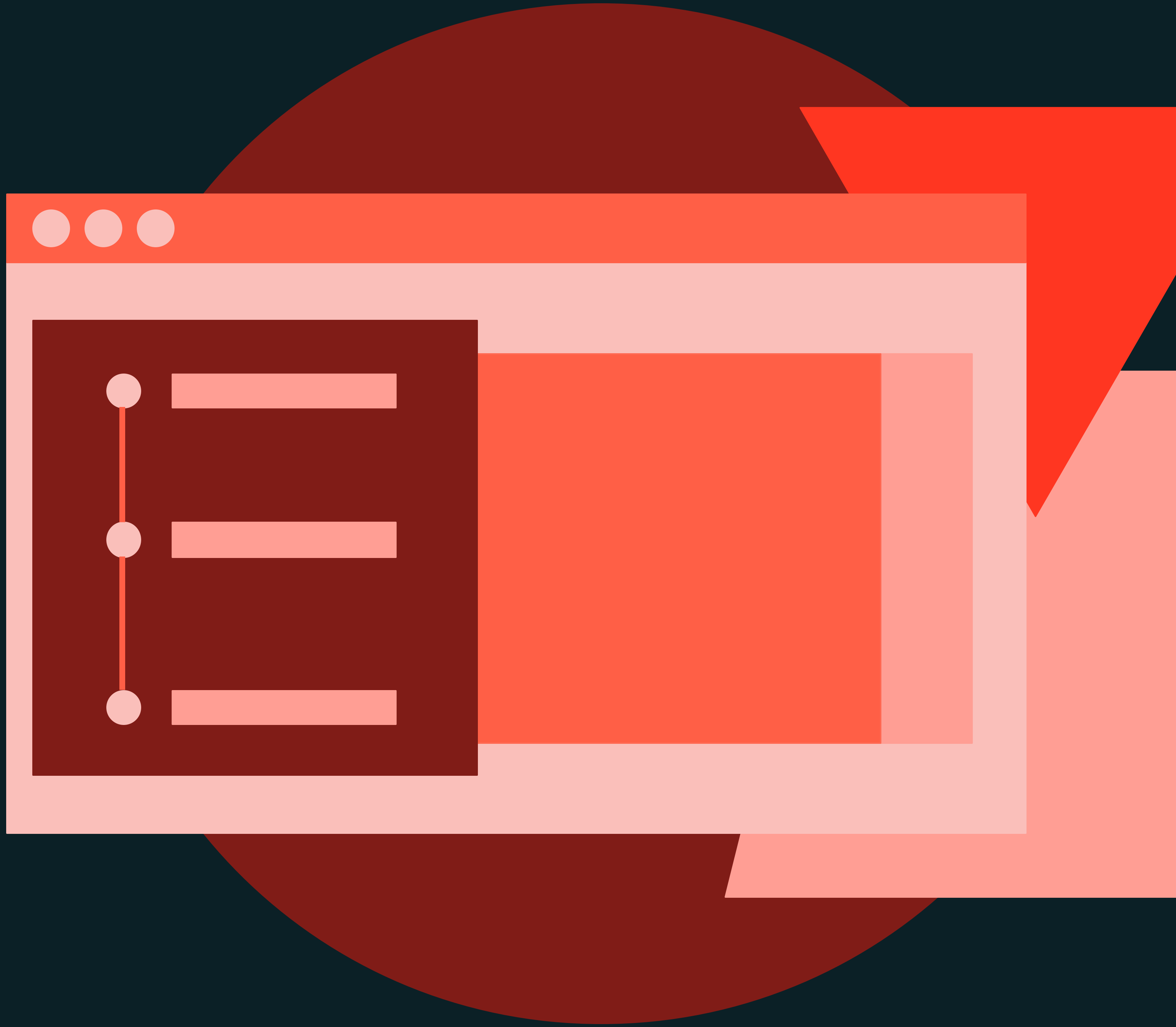
- Unified **batch** and **streaming**
- Automatic **schema validation**
- Support upserts using the merge operation
- Update your table schema without rewriting data.
- Track row-level changes with Change Data Feed
- **Time-travel**; querying previous versions of a table based on version number of timestamp
- **Performance optimization** with data skipping and liquid clustering
- Supports multiple programming languages like Python, Scala, and SQL.

databricks

Managing and Exploring Data

DEMONSTRATION

Load and Explore Data



Demo

Outline

What we'll cover:

- Explore data with summary statistics
- Data visualization
  - Integrated visualization tools
  - External visualization tools
- Time-travel with Delta

databricks

Managing and Exploring Data

LAB EXERCISE

Load and Explore Data

Lab

Outline

What you'll do:

- Read data from Delta table
- Manage data permissions
- Show summary statistics
- Use Data Profiler to explore data
- Time-travel to older versions of data
- Revert to previous versions of the Delta table

Data Preparation and Feature Engineering

Data Preparation for Machine Learning

Learning objectives

Things you'll be able to do after completing this module

- Explain data preparation and data splitting for ML, including train-test-holdout and cross-validation.
- Effectively handle missing values and understand the importance of indicator variables.
- Manage categorical features using encoding methods, considering efficiency and associated risks.
- Describe the benefits of feature standardization and recognize challenges in result interpretation.

Learning objectives

Things you'll be able to do after completing this module

- Build a data imputation and transformation pipeline for diverse datasets.
- Develop an advanced feature engineering pipeline, along with the process of saving it for future use.

databricks

Data Preparation and Feature Engineering

LECTURE

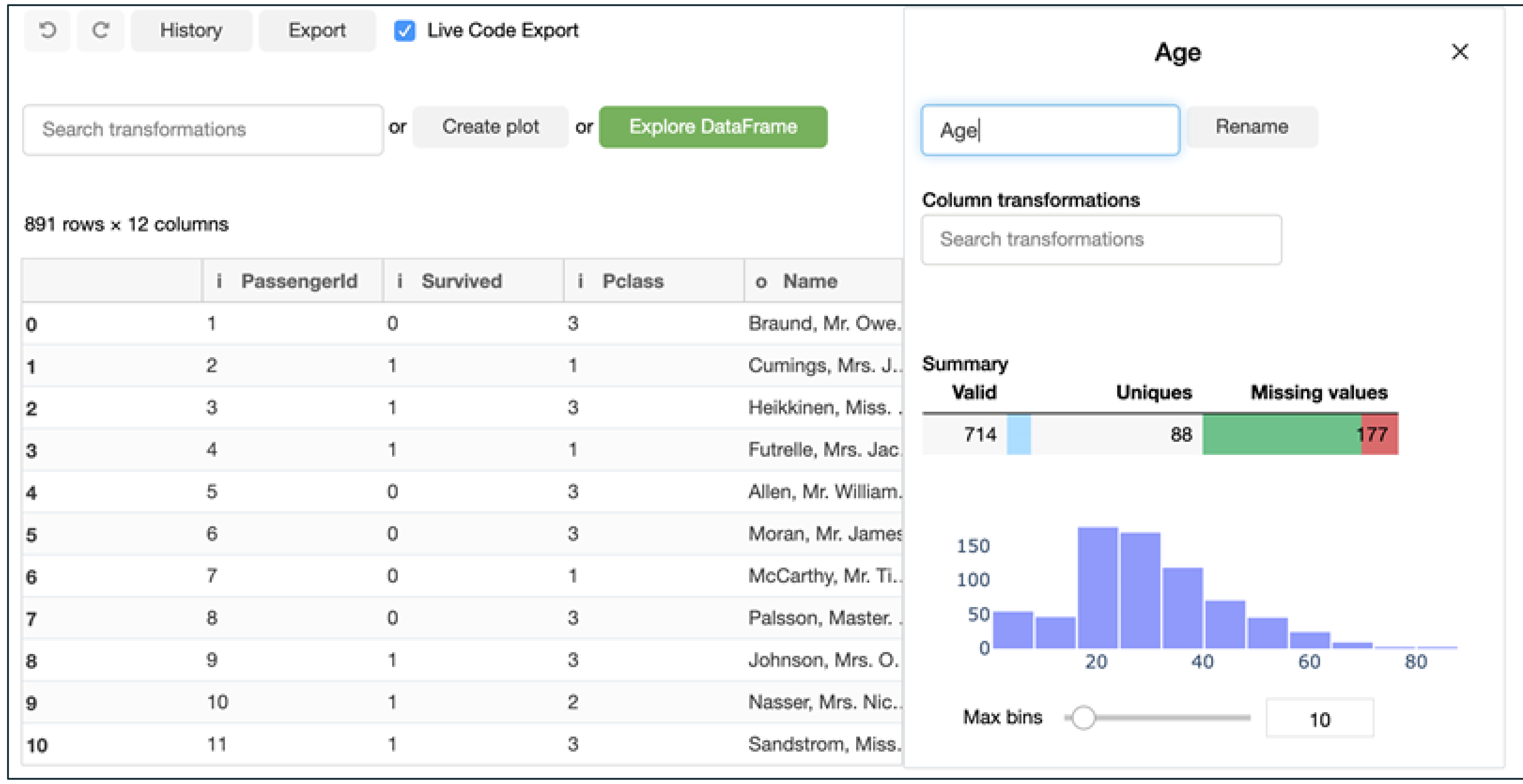
Fundamentals of Data Preparation and Feature Engineering

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**.



891 rows × 12 columns

	i PassengerId	i Survived	i Pclass	o Name
0	1	0	3	Braund, Mr. Owe.
1	2	1	1	Cumings, Mrs. J..
2	3	1	3	Heikkinen, Miss. .
3	4	1	1	Futrelle, Mrs. Jac
4	5	0	3	Allen, Mr. William.
5	6	0	3	Moran, Mr. James
6	7	0	1	McCarthy, Mr. Ti..
7	8	0	3	Palsson, Master. .
8	9	1	3	Johnson, Mrs. O.
9	10	1	2	Nasser, Mrs. Nic..
10	11	1	3	Sandstrom, Miss.

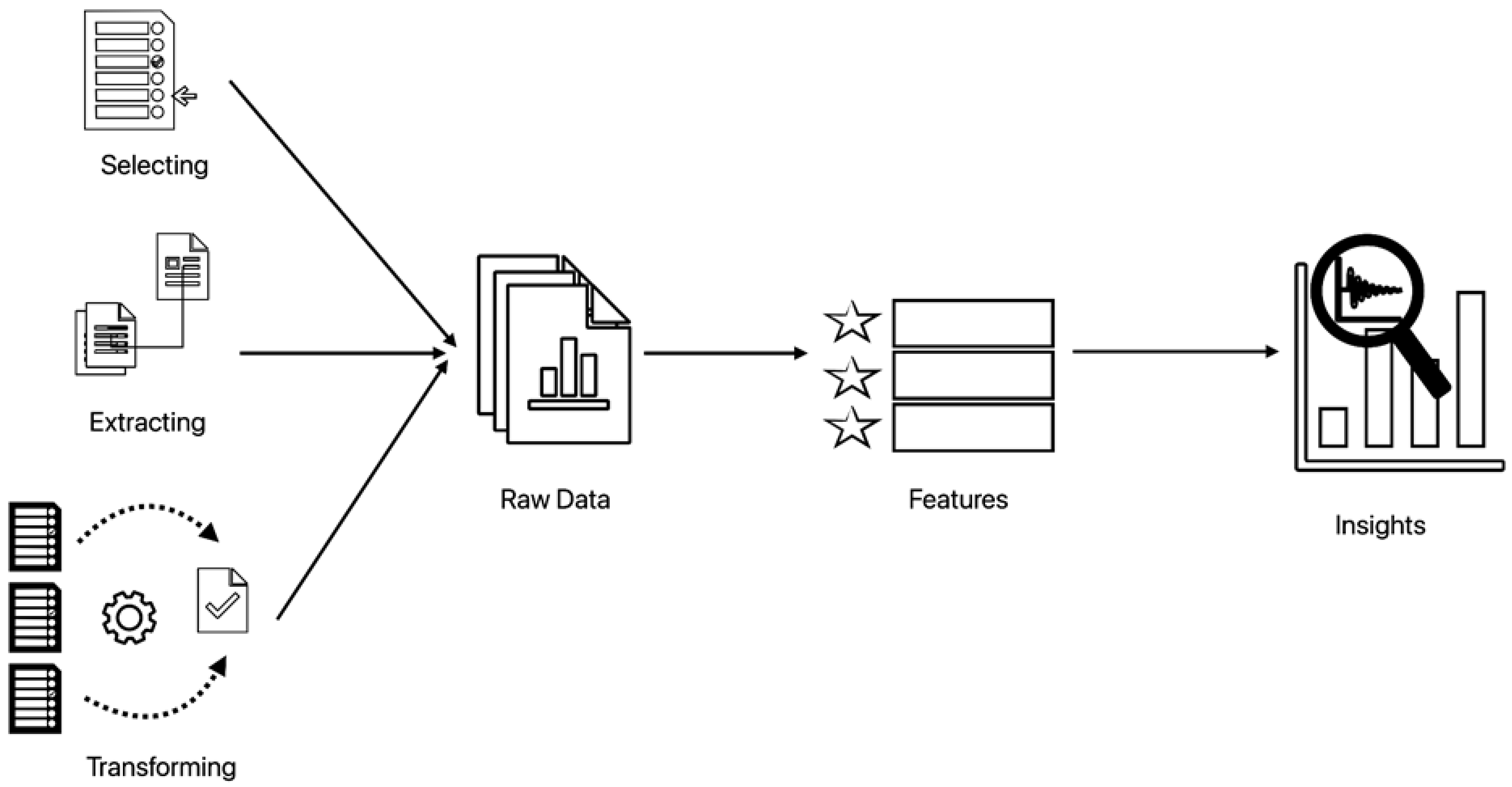
Summary

Valid	Uniques	Missing values
714	88	177

Feature Engineering

Transforming raw data into model-friendly features

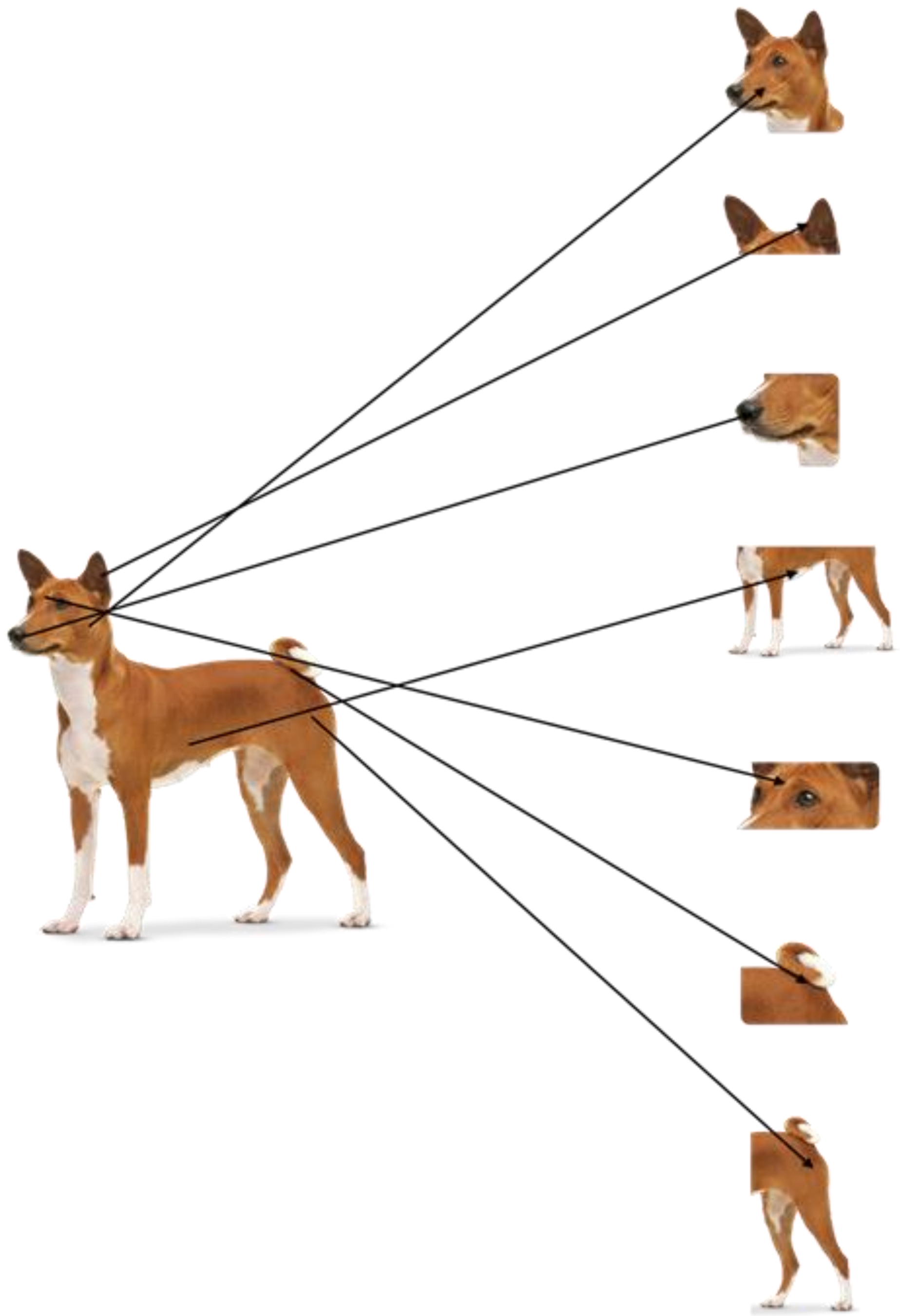
- It is a critical step in preparing data for machine learning models.
- It transforms raw data into a format that allows ML models to extract meaningful patterns and make accurate predictions.



Feature Extraction

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

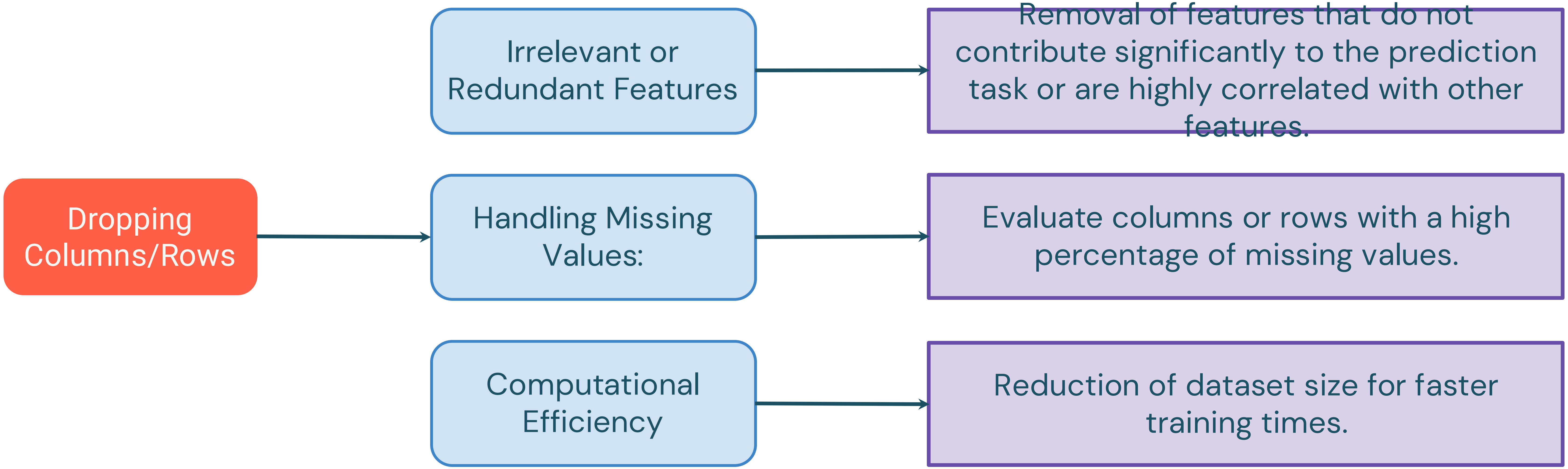
- Transforming Raw Data for Enhanced Modeling
- Dimensionality Reduction for Improved Performance
- Simplifying Feature Engineering



Feature Selection: Dropping Columns/Rows

Enhancing Model Efficiency

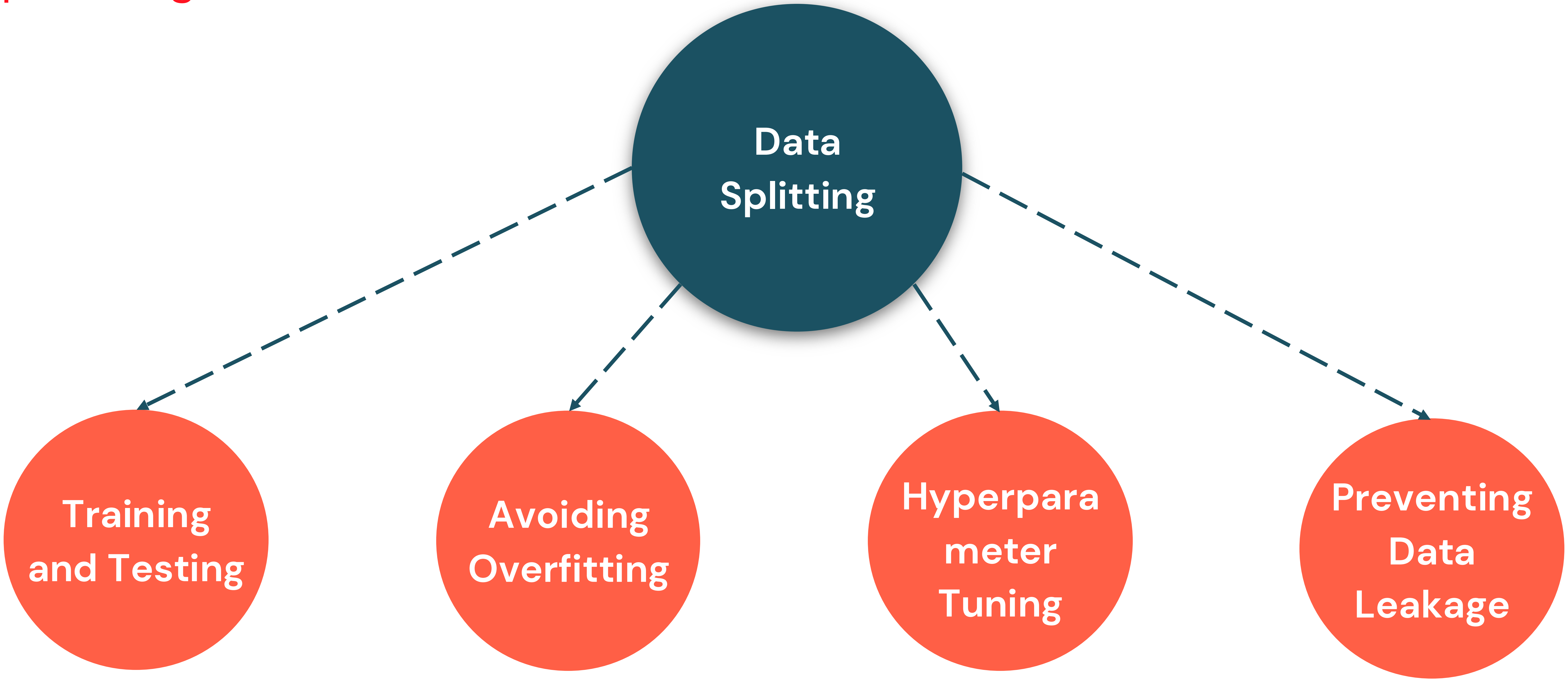
Streamlining the dataset by selectively removing columns (features) or rows (instances) to improve model efficiency and effectiveness.



Dropping Columns/Rows
Irrelevant or Redundant Features
Removal of features that do not contribute significantly to the prediction task or are highly correlated with other features.
Handling Missing Values:
Evaluate columns or rows with a high percentage of missing values.
Computational Efficiency
Reduction of dataset size for faster training times.

Why Do We Need Splitting Data

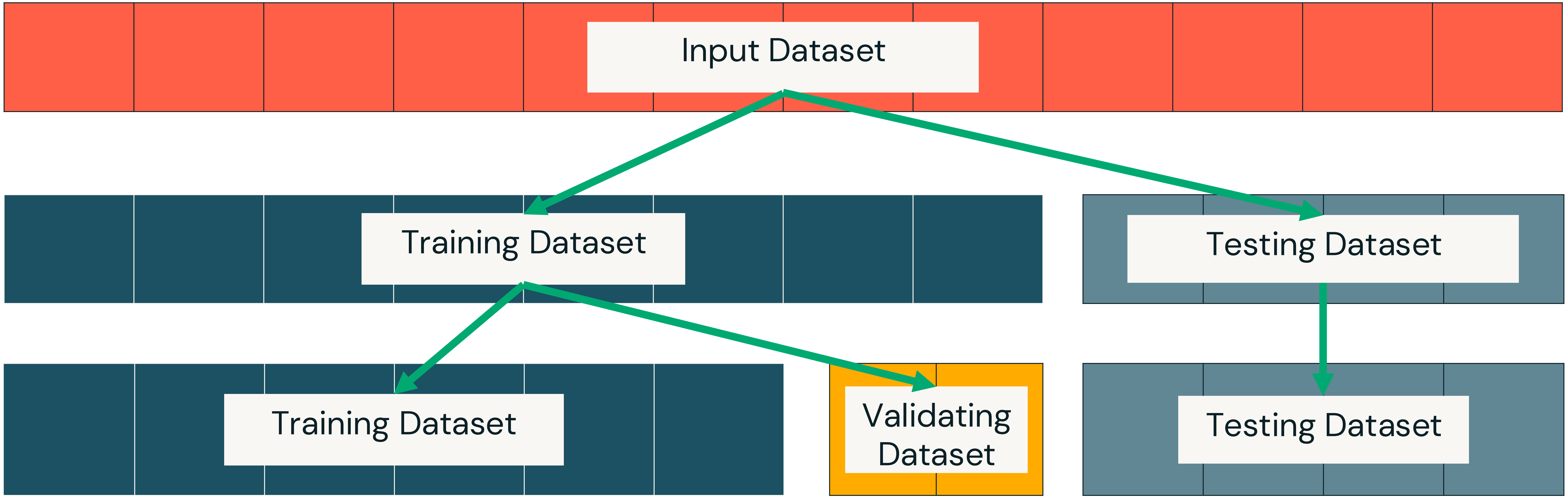
Optimizing Model Evaluation



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.



Sampling Methods

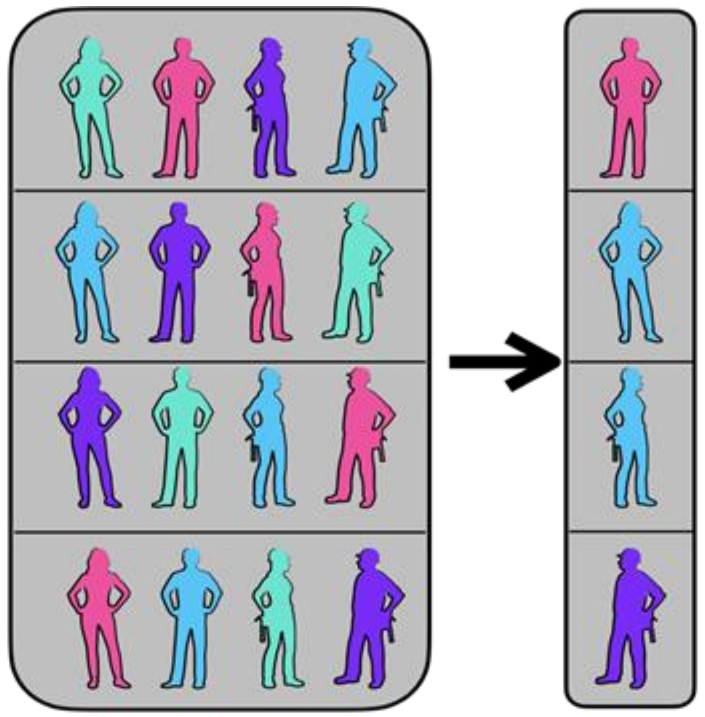
Sampling methods play a crucial role in selecting subsets of data for training, validation, and testing, influencing the performance and generalization of machine learning models. Always take care in how you are performing your sampling for your particular dataset.

Sampling Methods

Sampling methods play a crucial role in selecting subsets of data for training, validation, and testing, influencing the performance and generalization of machine learning models. Always take care in how you are performing your sampling for your particular dataset.

Here's an overview of common sampling methods:

Random Sampling:



This is the most common and basic sampling technique in machine learning. It randomly selects data points from a dataset without following any specific pattern, assuming that each data point has an equal probability of being chosen.

Example with a pandas DataFrame

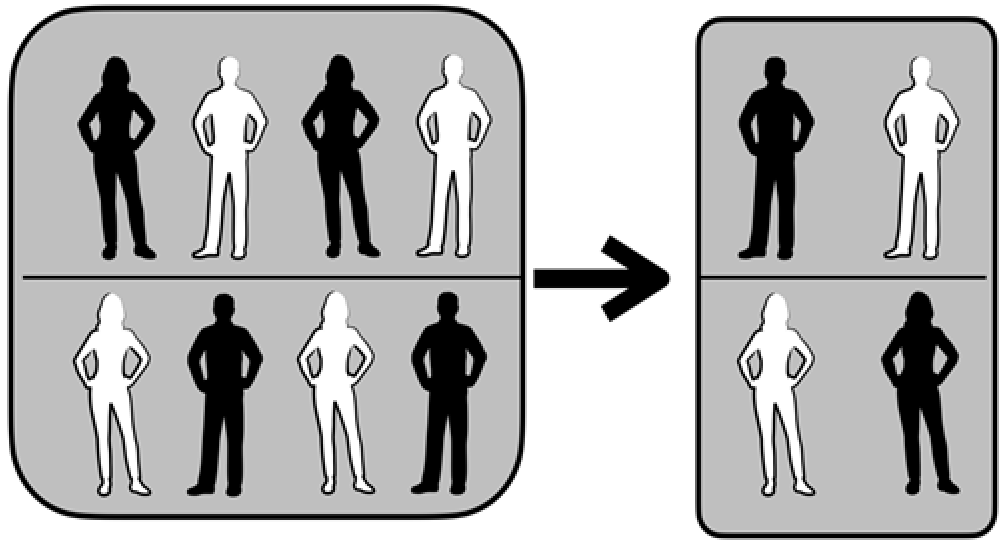
```
# my_data is a pandas DataFrame and we want to sample 50 data
points

my_random_sample = my_data.sample(n=50)
```

Sampling Methods

Sampling methods play a crucial role in selecting subsets of data for training, validation, and testing, influencing the performance and generalization of machine learning models. Always take care in how you are performing your sampling for your particular dataset.

Here's an overview of common sampling methods:

Stratified Sampling:



This sampling method is used when a dataset consists of subgroups, ensuring that samples are taken from each. The population is divided into homogeneous subgroups, known as strata, and an appropriate number of instances is selected from each stratum to ensure the sample accurately represents the entire population.

Example with a pandas DataFrame & scikit-learn

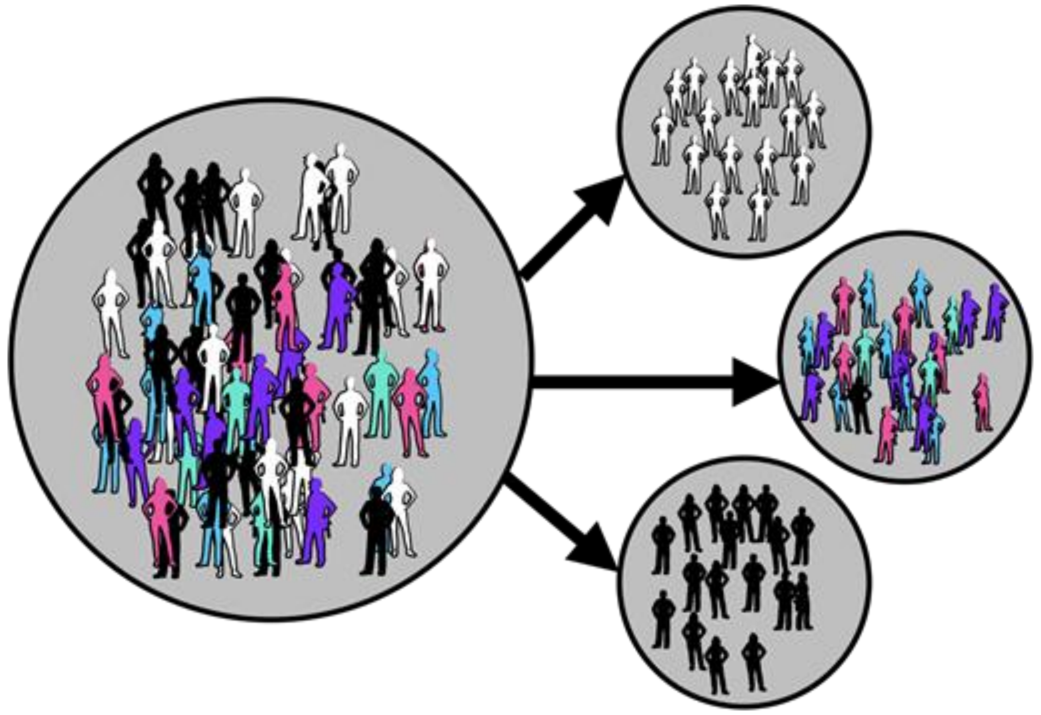
```
# my_data is a pandas DataFrame
# group_label is the column containing subgroups

from sklearn.model_selection import train_test_split

stratified_train, stratified_test = train_test_split(my_data, test_size =0.2, stratify =
my_data[‘group_label])
```

Sampling Methods

Sampling methods play a crucial role in selecting subsets of data for training, validation, and testing, influencing the performance and generalization of machine learning models. Always take care in how you are performing your sampling for your particular dataset.

Here's an overview of common sampling methods:

Cluster Sampling:



Cluster sampling is a sampling technique where the population is divided into naturally occurring groups, or clusters, and a subset of these clusters is randomly selected for study. Instead of sampling individuals directly, researchers collect data from all individuals within the chosen clusters.

Example with a pandas DataFrame and numpy

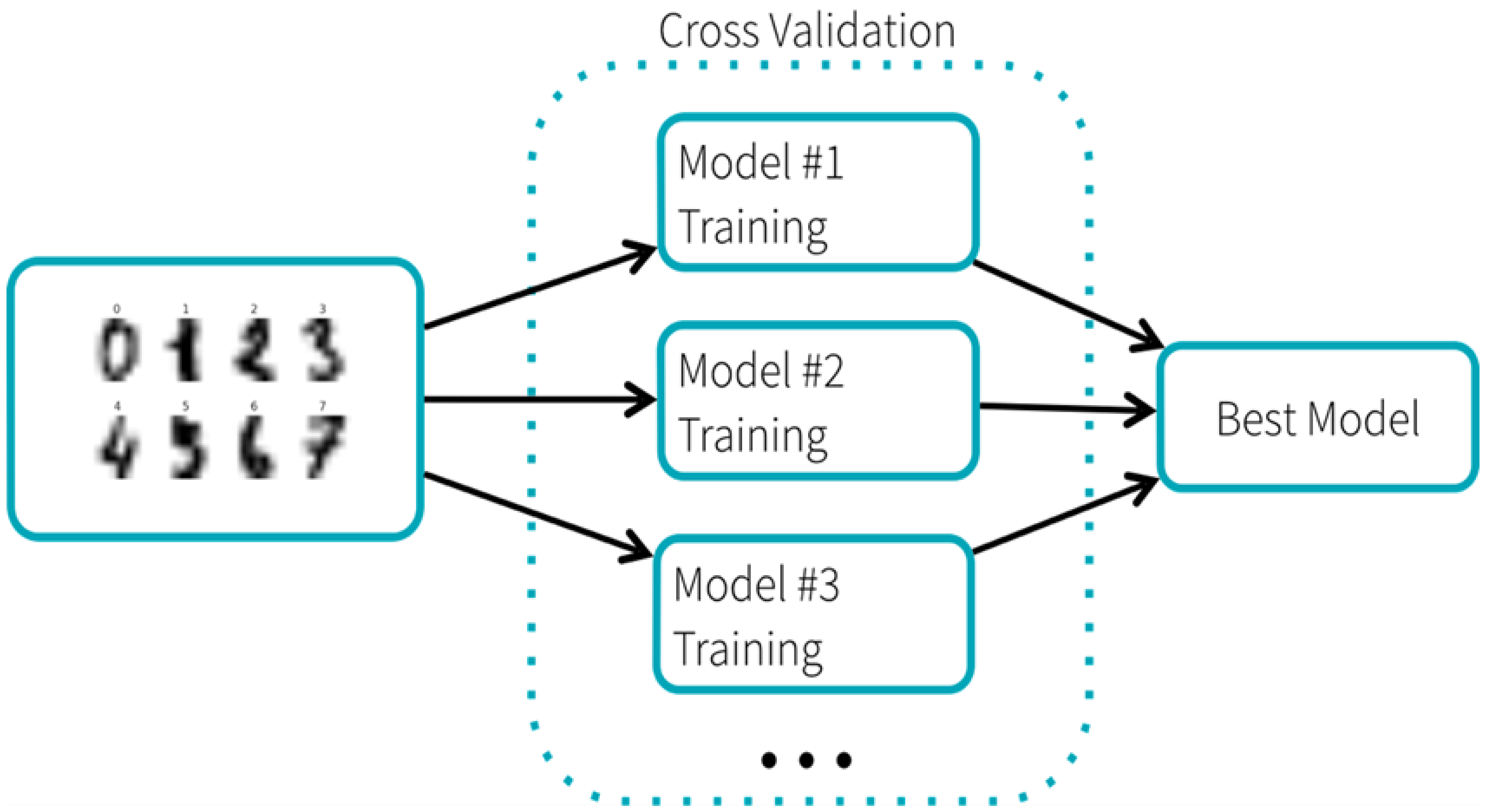
```
# my_data is a pandas DataFrame
# my_col is the column containing subgroup identifiers
# Randomly select M clusters from the range 1 to N

import pandas as pd
import numpy as np

clusters = np.random.choice(np.arange(1, N+1), size=M, replace=False)
cluster_sample = my_data[my_data['my_col'].isin(clusters)]
```

Cross-Validation

Cross-validation is a resampling technique used to assess the performance of a machine learning model. It involves partitioning the dataset into subsets to train and evaluate the model multiple times, providing a more robust estimate of its generalization performance.

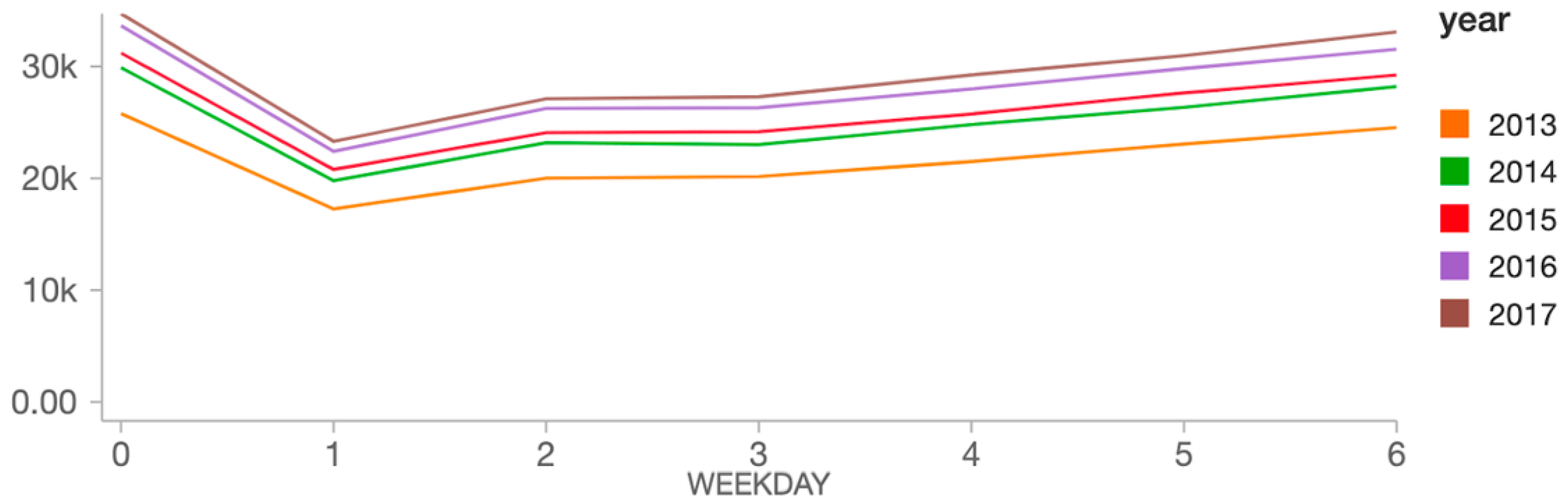
Here are the key elements of cross-validation:

- K-Fold Cross-Validation
- Stratified K-Fold Cross-Validation
- Shuffle Split Cross-Validation
- Nested Cross-Validation



Sampling for Time-Series Data

Sampling for **time-series data requires special consideration** due to the temporal dependencies inherent in the data. **Traditional random sampling or shuffling may not be appropriate**, as the order of events in time is crucial.



databricks

Data Preparation and Feature Engineering

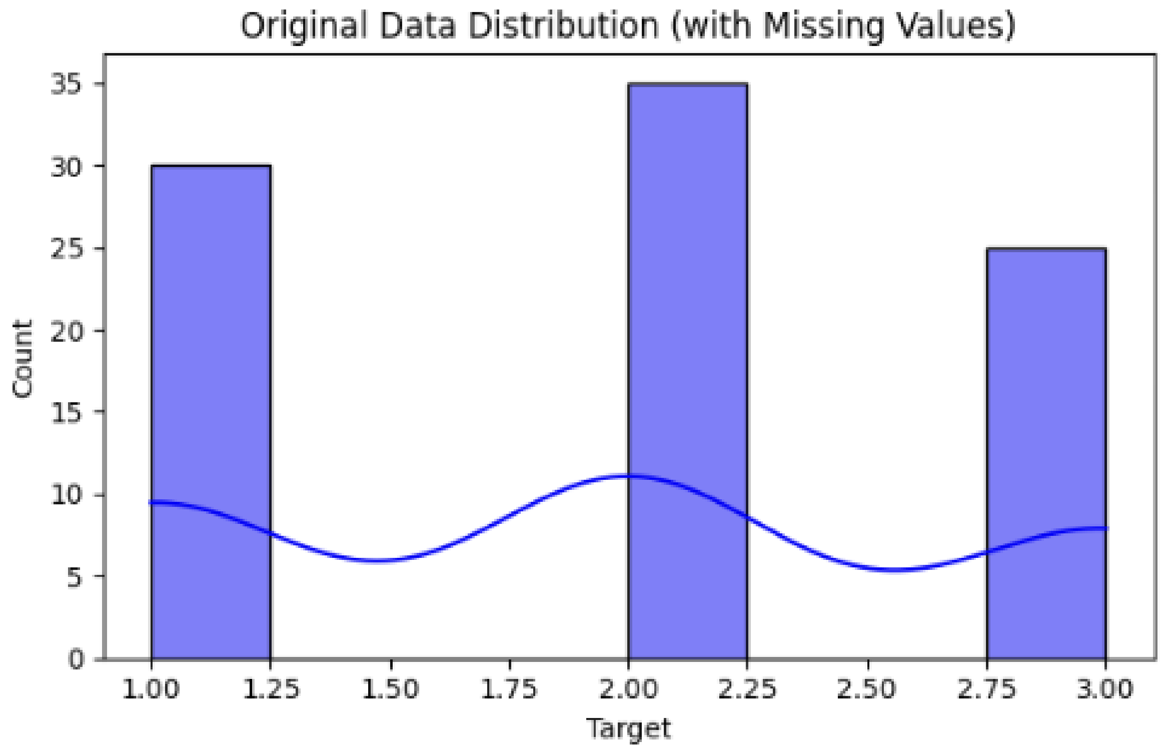
LECTURE

Data Imputation

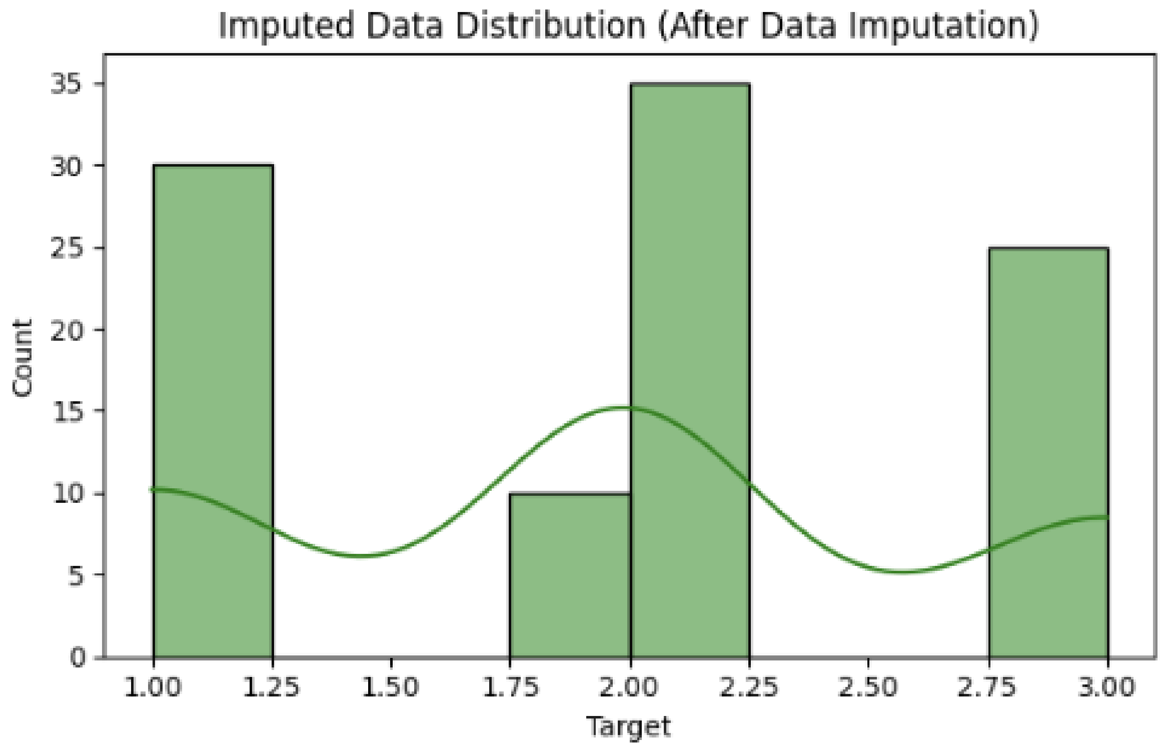
Data Imputation

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.



Original Data Distribution (with Missing Values)

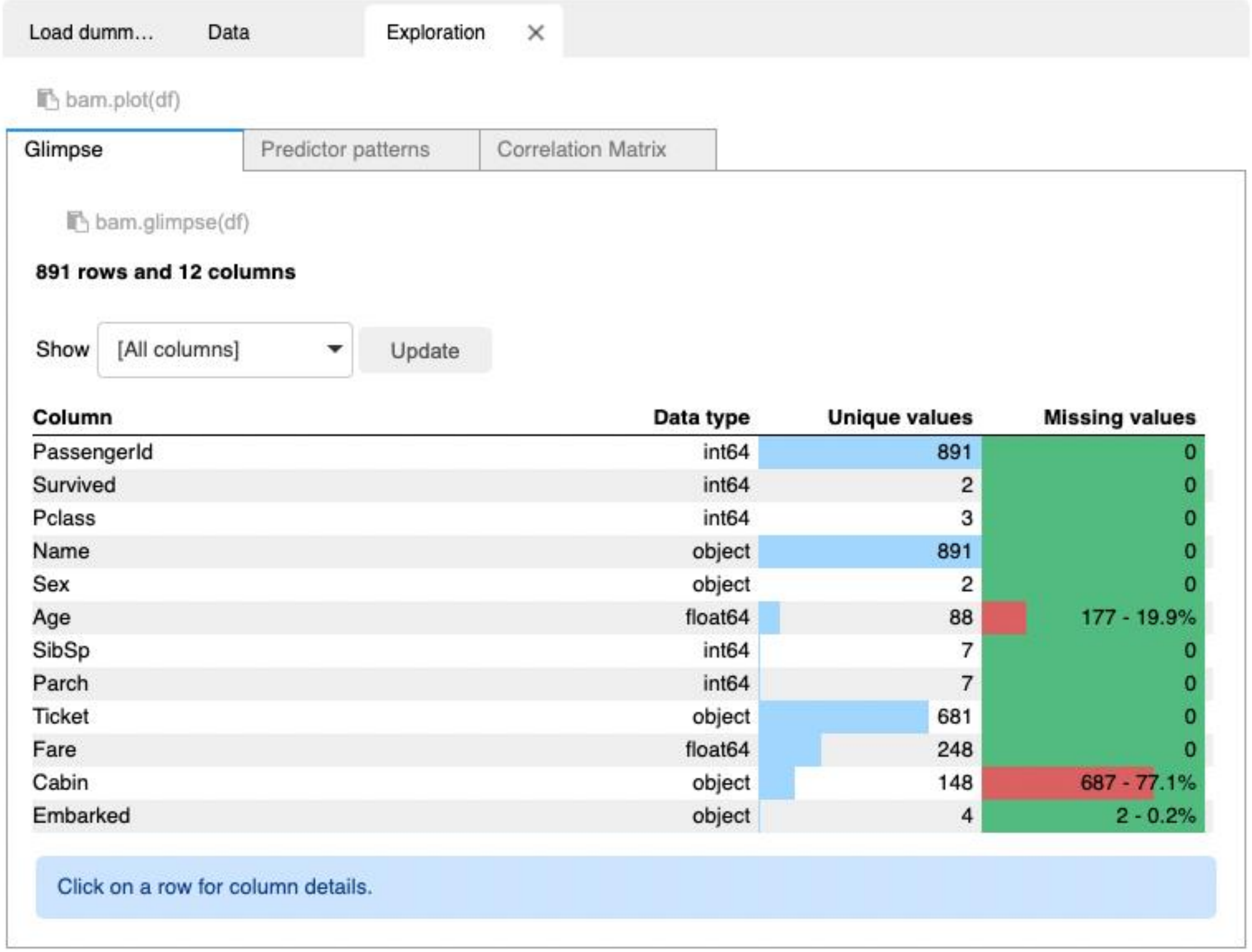


Imputed Data Distribution (After Data Imputation)

Problems with Missing Data

Impacting the performance and reliability of ML models

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



bam.plot(df)

bam.glimpse(df)

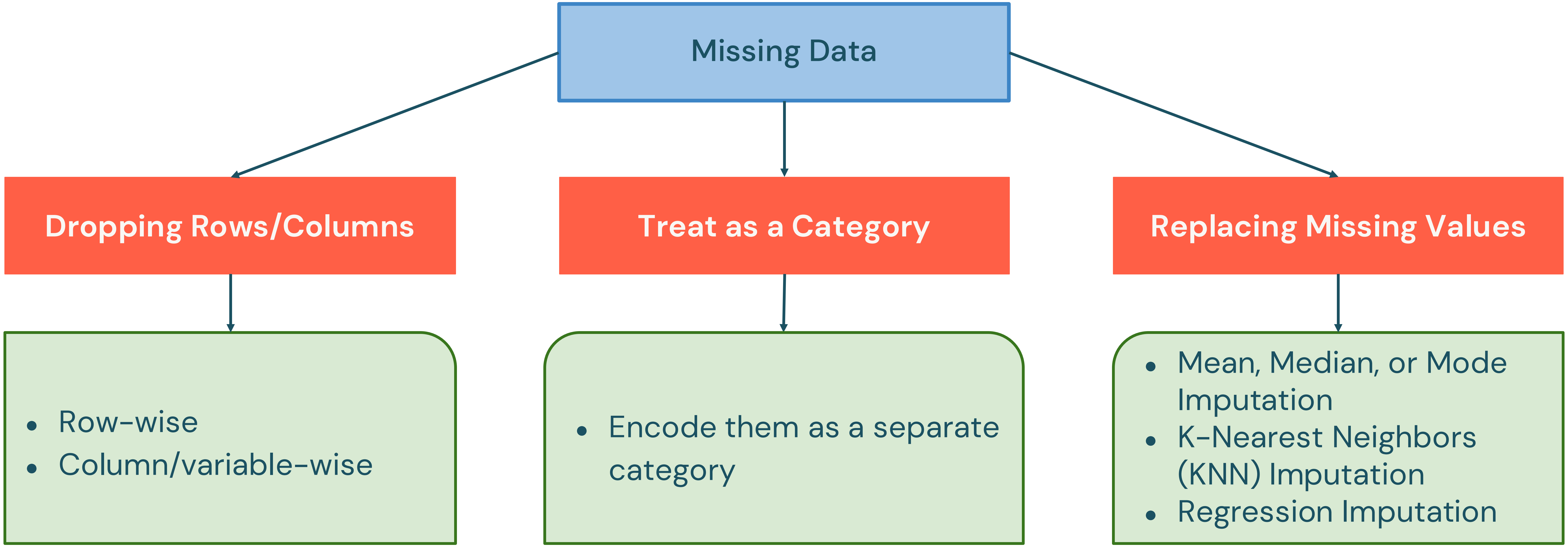
891 rows and 12 columns

Column	Data type	Unique values	Missing values
PassengerId	int64	891	0
Survived	int64	2	0
Pclass	int64	3	0
Name	object	891	0
Sex	object	2	0
Age	float64	88	177 - 19.9%
SibSp	int64	7	0
Parch	int64	7	0
Ticket	object	681	0
Fare	float64	248	0
Cabin	object	148	687 - 77.1%
Embarked	object	4	2 - 0.2%

Click on a row for column details.

How to Handle Missing Data

Data imputation methods



- Row-wise
- Column/variable-wise

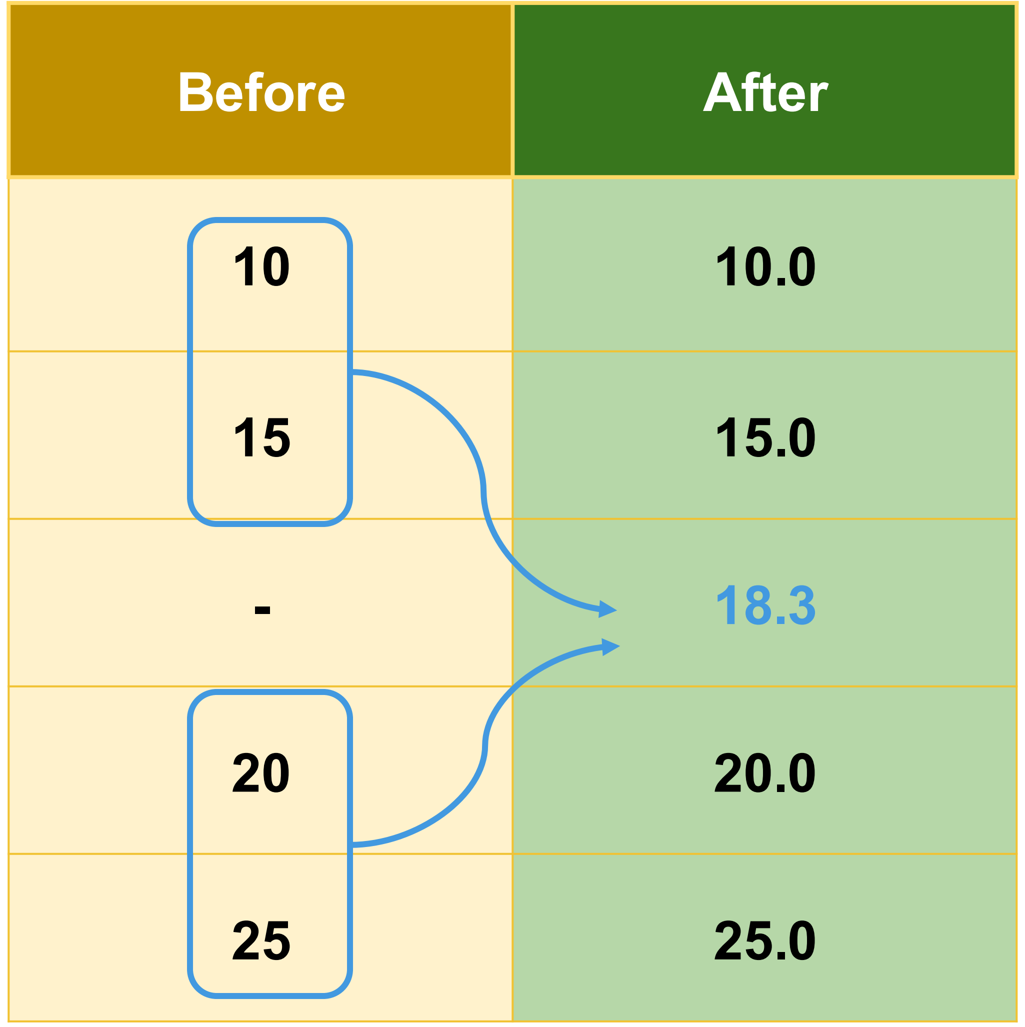
- Encode them as a separate category

- Mean, Median, or Mode Imputation
- K-Nearest Neighbors (KNN) Imputation
- Regression Imputation

Replacing Missing Values

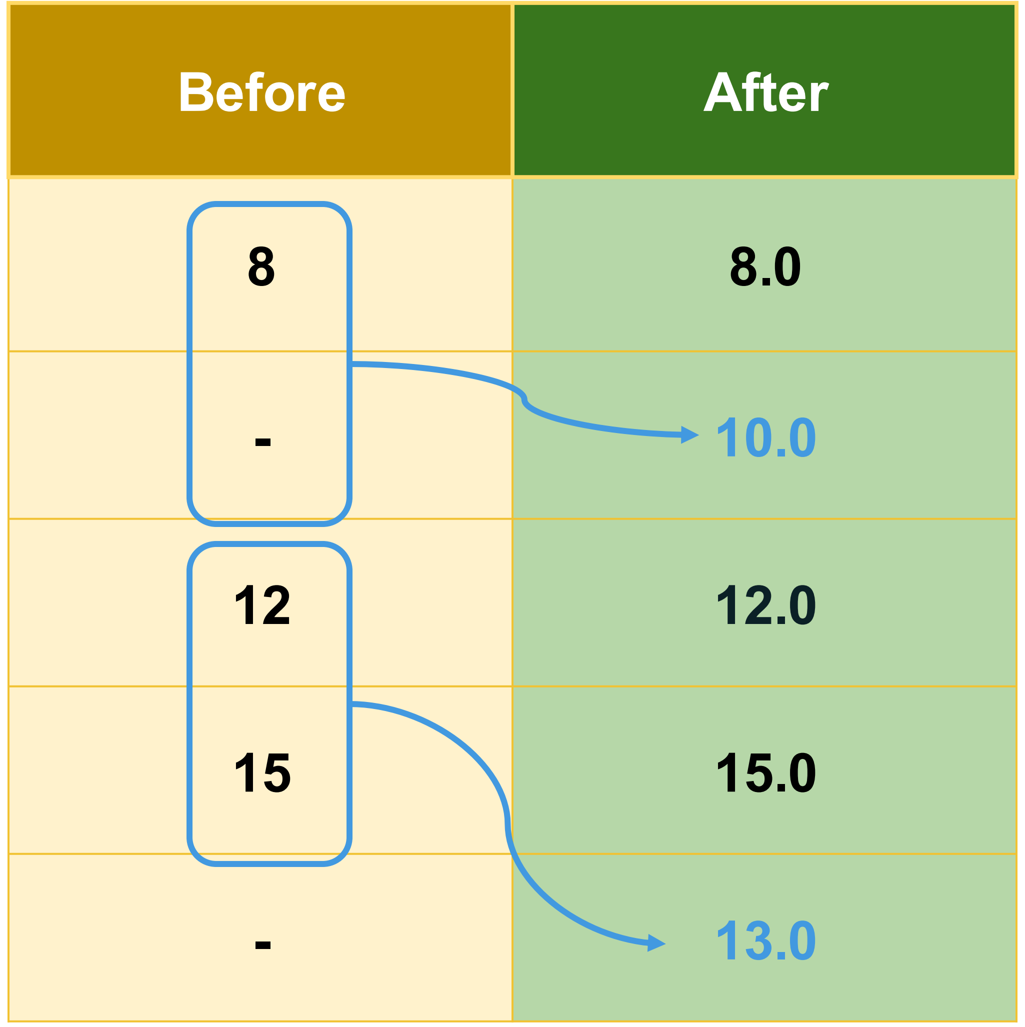
Data imputation methods

Mean – Mode Imputation



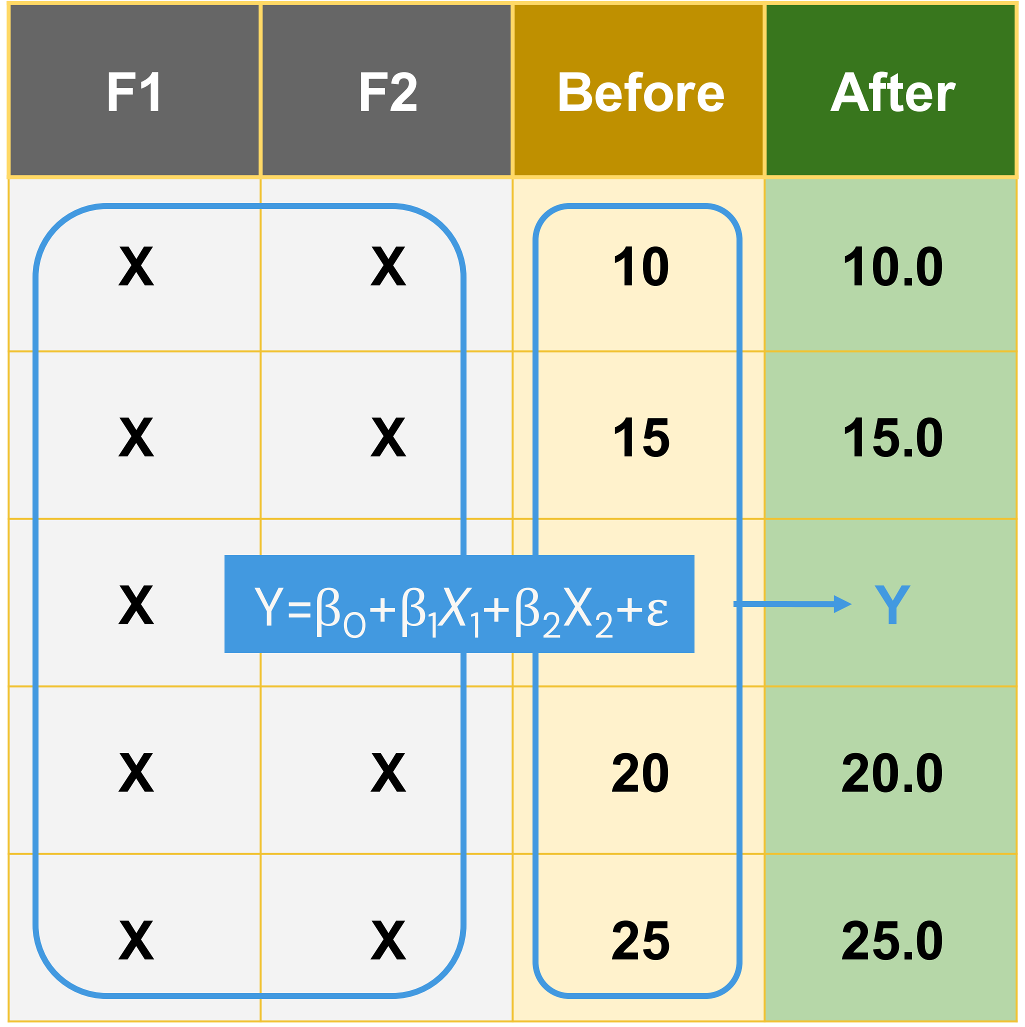
Before	After
10	10.0
15	15.0
-	18.3
20	20.0
25	25.0

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

Factors Influencing Imputation Method

There isn't a definitive best approach

Nature of Data

- Is data type is **continuous, ordinal or categorical**?
- Data **distribution**: For example, median imp. Might work better for non-normal distributions

Amount of Missing Data

- How much of data is missing?
- Will imputing data improve the quality of dataset or will it add more bias?
- Multiple imp. and KNN might better for large missingness.

Form of Missingness

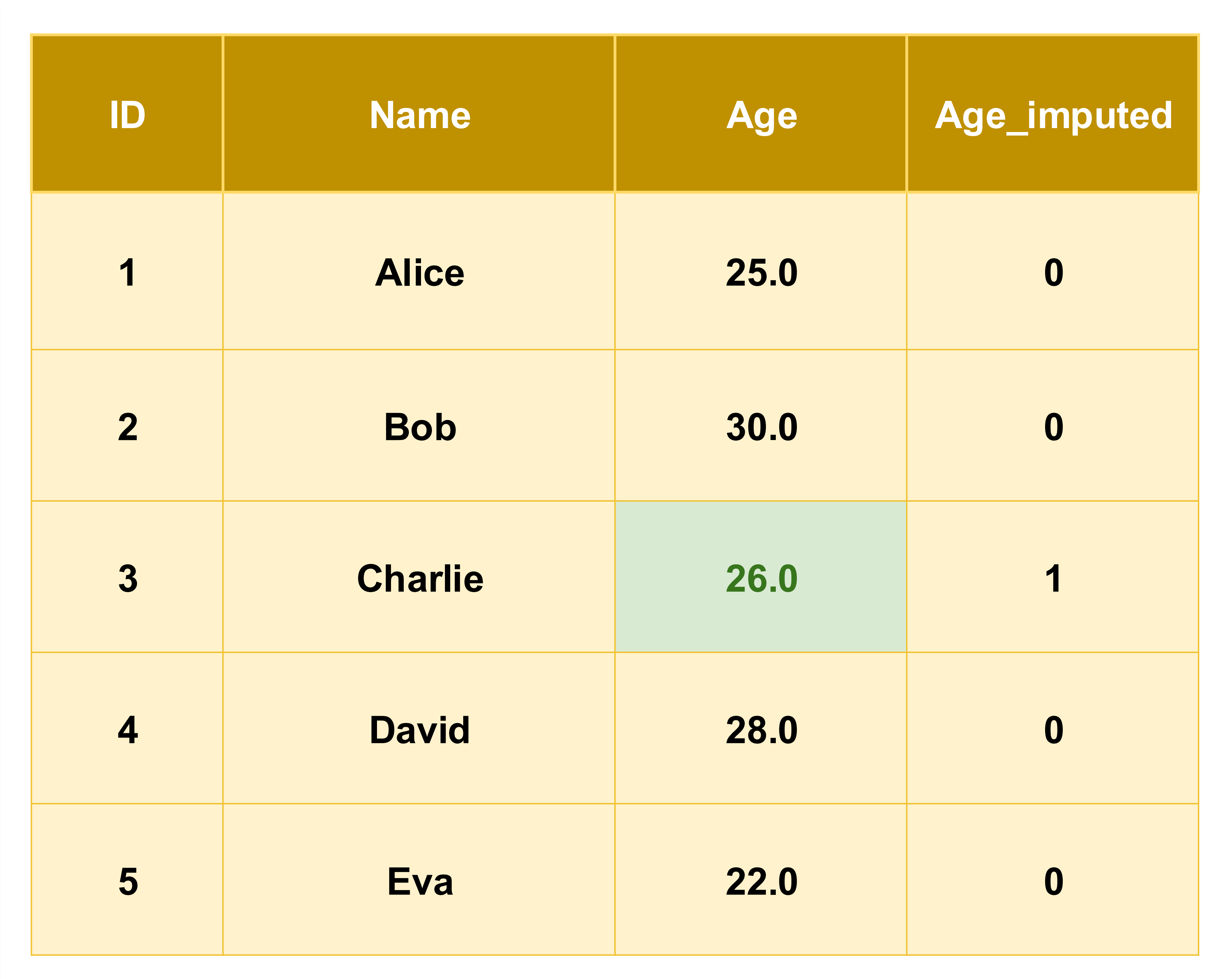
- Is data is **missing at random** or is there a **missingness pattern**.
- Domain knowledge and how would imputation affect downstream analysis.

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

databricks

Data Preparation and Feature Engineering

LECTURE

Data Encoding

Data Encoding

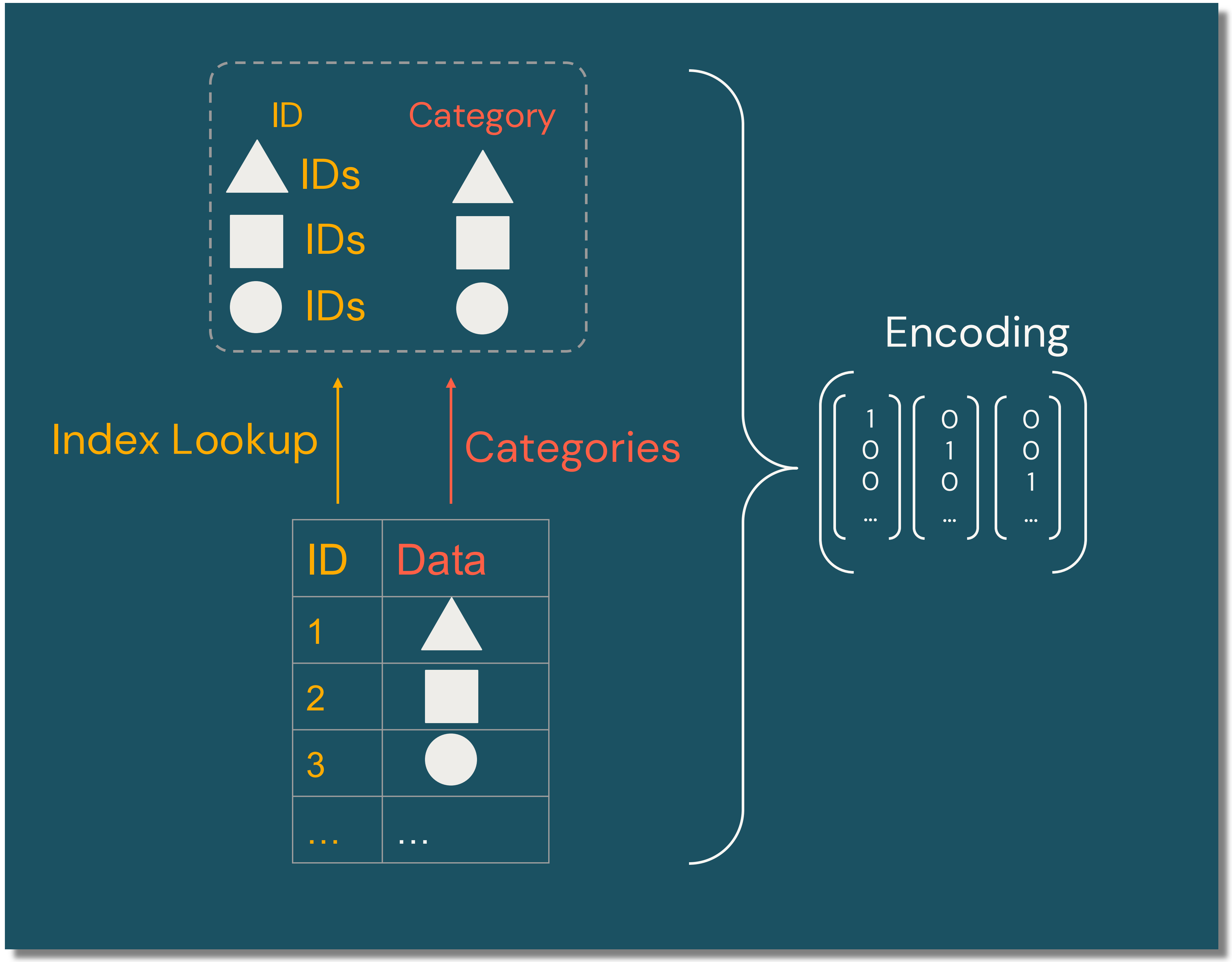
Why Encoding?

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	0	0
1	0	1	0
2	1	0	0
3	0	0	1
4	1	0	0

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

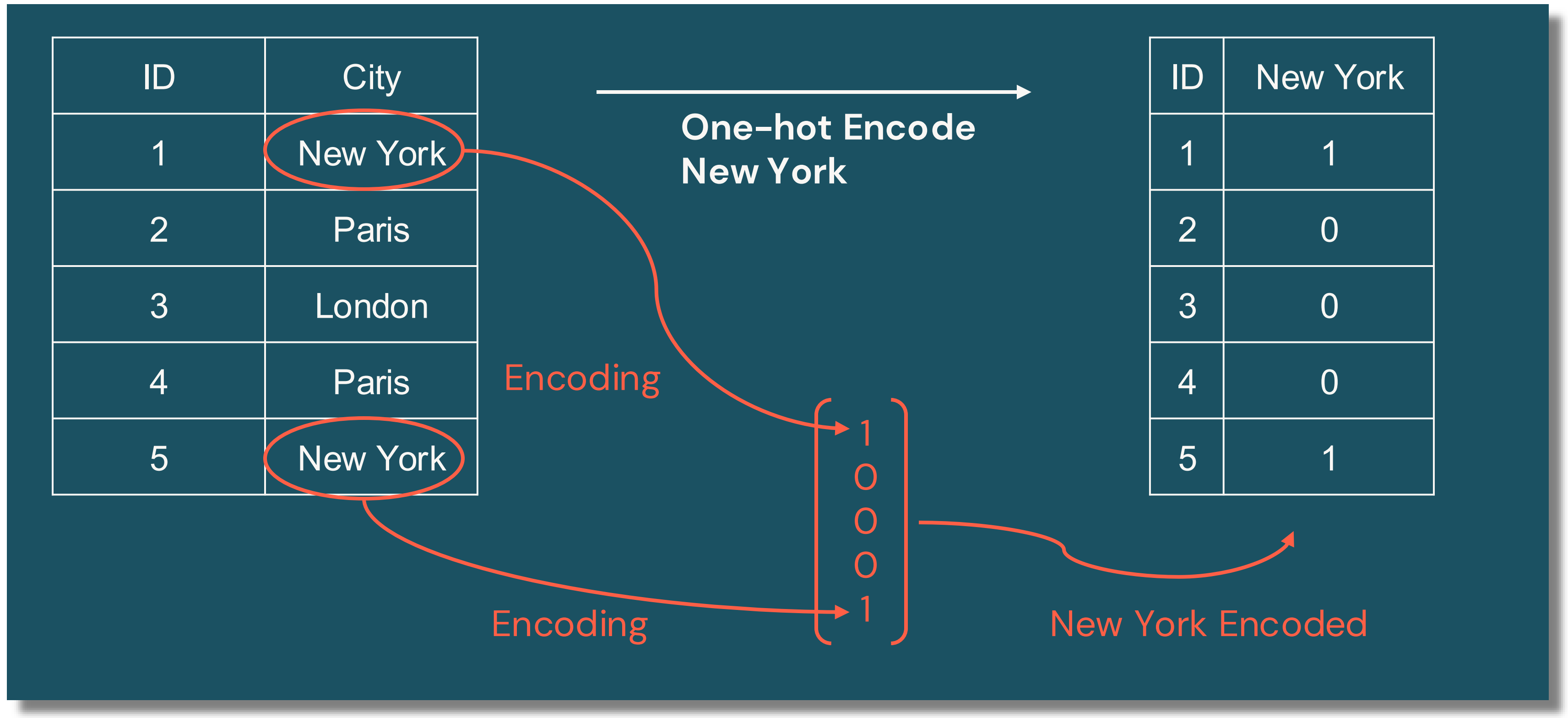
One-hot Encoding

Represent categorical variables as binary vectors

Procedure:

1. **Add binary columns** (0 or 1) for each unique value in categorical columns.
2. Each category is now a column that map to either 0 or 1 in reference to the ID column

Example:



ID	City
1	New York
2	Paris
3	London
4	Paris
5	New York

ID	New York
1	1
2	0
3	0
4	0
5	1

ID	City
1	New York
2	Paris
3	London
4	Paris
5	New York

ID	New York	Paris	London
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0
5	1	0	0

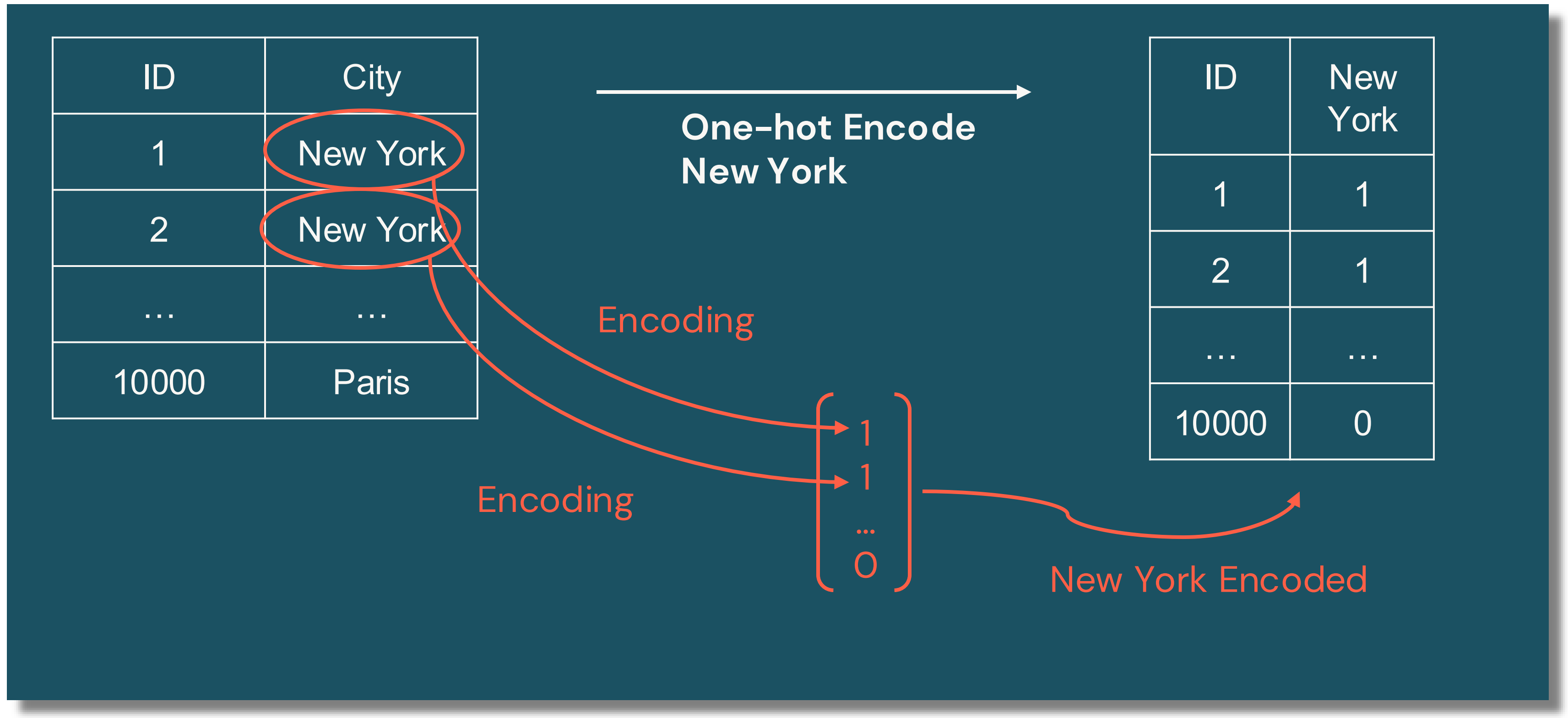
One-hot Encoding

Represent categorical variables as binary vectors

Drawbacks:

1. Induces sparsity
2. Limites split options for tree-based models
3. Inefficiency with high-cardinality variables
4. Obscures feature importance
5. Increases computational costs

Example:



ID	City
1	New York
2	New York
…	…
10000	Paris

ID	New York
1	1
2	1
…	…
10000	0

ID	City
1	New York
2	Paris
3	London
4	Paris
5	New York

ID	New York	Paris	London
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0
5	1	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.

Example:

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

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

Label Encoding for Ordinal Features

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

Drawbacks:

1. Arbitrary Assignments Can Mislead Models.
2. Misinterpretation of Relationships.
3. Model Sensitivity.
4. Loss of Information.
5. Not Suitable for all algorithms.

Example:

String Value		Numeric Value
Freshman	Less Important	1
Sophomore	More Important	2
Junior		3
Senior		4

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

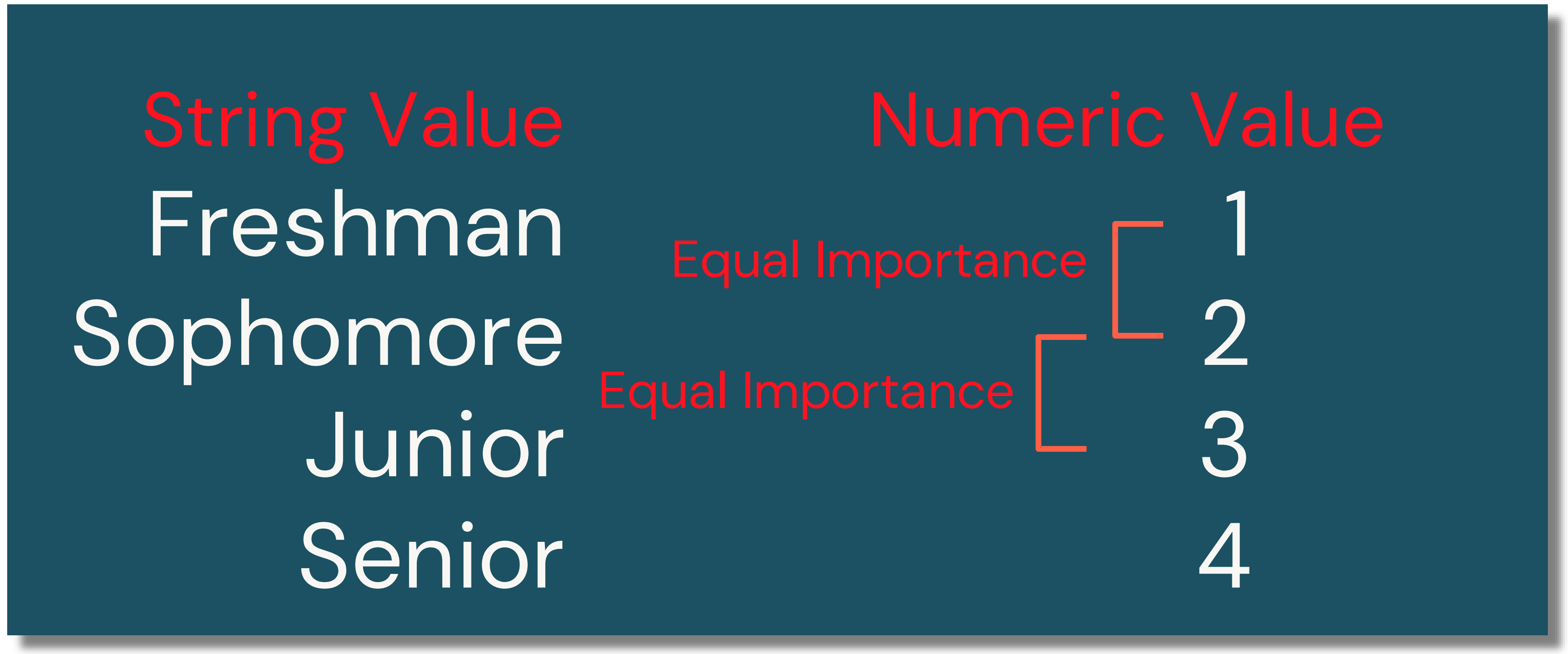
Label Encoding for Ordinal Features

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

Drawbacks:

1. Arbitrary Assignments Can Mislead Models.
2. Misinterpretation of Relationships.
3. Model Sensitivity.
4. Loss of Information.
5. Not Suitable for all algorithms.

Example:



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

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

Target Encoding

Encoding categorical variables: replacement by mean of target variable

Procedure:

1. **Compute the mean** of the target variable for each category.
2. **Substitute** category instances with their mean values.

ID	City	Clicked
1	New York	1
2	Paris	0
3	London	1
4	Paris	1
5	New York	0

ID	City	Clicked
1	0.5	1
2	0.5	0
3	1.0	1
4	0.5	1
5	0.5	0

Example:

City	Clicked Values	Mean
New York	[1,0]	0.5
Paris	[0,1]	0.5
London	[1]	1

Target Encoding

Encoding categorical variables: replacement by mean of target variable

Drawbacks:

1. Overfitting risk
2. Handling new categories
3. Dependency of target distribution
4. Limited applicability to small datasets
5. Risk of bias

Example:

City	Clicked Values	Mean
New York	[1,0]	0.5
Paris	[0,1]	0.5
London	**[1]**	**1**

Can lead to overconfident prediction

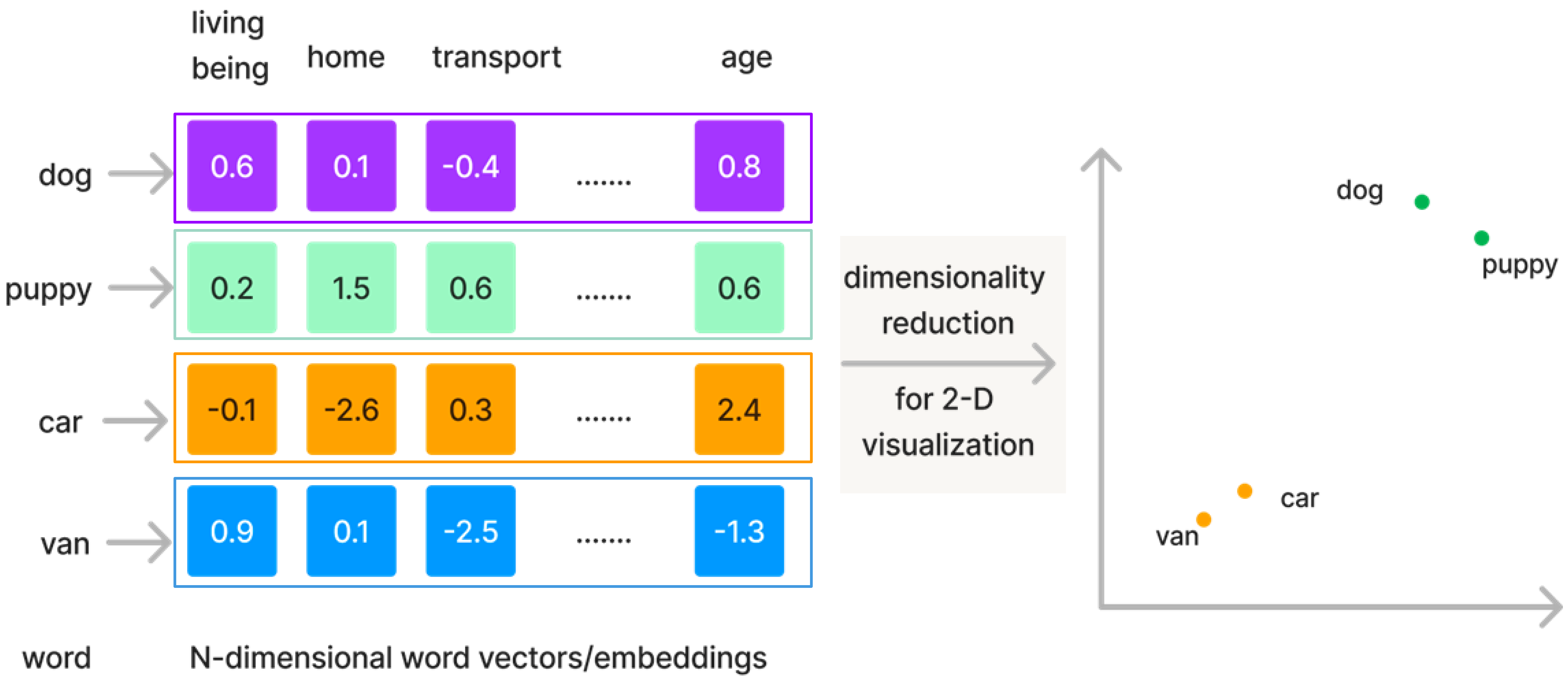
ID	City	Clicked
1	New York	1
2	Paris	0
3	London	1
4	Paris	1
5	New York	0

ID	City	Clicked
1	0.5	1
2	0.5	0
3	1.0	1
4	0.5	1
5	0.5	0

What are Embeddings?

Embeddings: Representing Data in a Meaningful Way

- Embeddings are ***dense vector representations*** of data, capturing semantic relationships.
- Unlike traditional encoding methods, embeddings ***place similar items closer together*** in a lower-dimensional space.
- Used widely in ***text***, ***categorical data***, ***images***, and ***audio processing***.



word	living being	home	transport		age
dog	0.6	0.1	-0.4	....	0.8
puppy	0.2	1.5	0.6	....	0.6
car	-0.1	-2.6	0.3	....	2.4
van	0.9	0.1	-2.5	....	-1.3

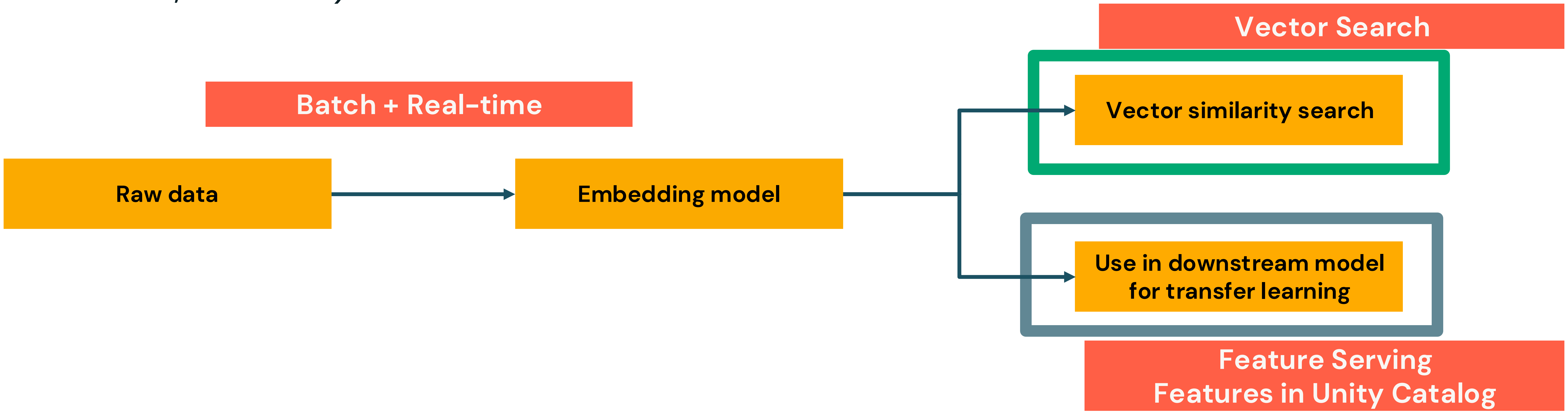
word N-dimensional word vectors/embeddings

dimensionality reduction for 2-D visualization

How Embeddings Work?

The Role of Embeddings in Machine Learning

- Embeddings transform high-dimensional categorical or textual data into a **compact**, **dense vector space**.
- These representations capture relationships and context among different entities.
- Used in **Recommendation Systems**, **NLP**, **Image Search**, **and more**.
- Can be learned from data using neural networks or retrieved from **pretrained models** (e.g., Word2Vec, FastText).



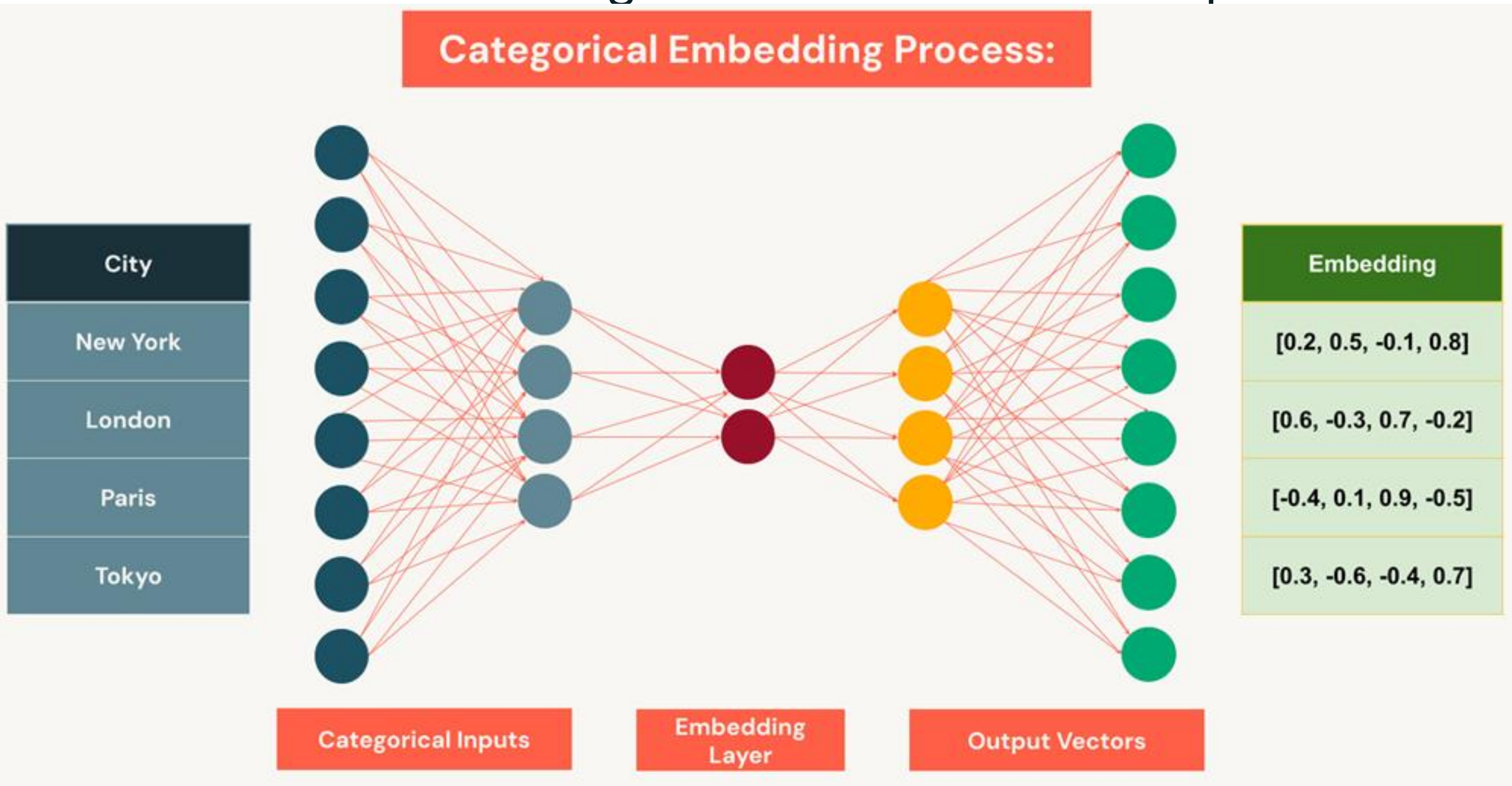
Categorical Embedding

Transforming categorical variables into meaningful, continuous vectors

- Categorical embeddings replace sparse categorical encoding methods with lower-dimensional, dense vector representations.
- These embeddings capture relationships between different categories based on learned patterns.

Process:

1. Convert categorical variables into unique indices.
2. Pass through an embedding layer (neural network or learned lookup table).
3. The model assigns dense vector representations to each category based on learned relationships.



City
New York
London
Paris
Tokyo

Embedding
[0.2, 0.5, -0.1, 0.8]
[0.6, -0.3, 0.7, -0.2]
[-0.4, 0.1, 0.9, -0.5]
[0.3, -0.6, -0.4, 0.7]

Embeddings vs. One-Hot Encoding

Choosing the Right Encoding for Categorical Data

Feature	One-Hot Encoding	Embeddings
Representation	Sparse binary vectors (high-dimensional)	Dense numerical vectors (low-dimensional)
Semantic Relationships	Does not capture relationships between categories	Places similar categories closer in vector space
Scalability	Inefficient for high-cardinality data	Scales well with large category sets
Efficiency	Inefficient for high-cardinality features (sparse matrix)	Efficient for high-cardinality features
Model Suitability	Suitable for simple models like Decision Trees	Best for Neural Networks and deep learning models

Example:	`TV → [1, 0, 0, 0]`
	`Laptop → [0, 1, 0, 0]`
	`Phone → [0, 0, 1, 0]`
		`TV → [0.6, 1.2, -0.8]`
		`Laptop → [0.5, 1.1, -0.7]`
		`Phone → [0.2, -0.4, 1.5]`

databricks

Data Preparation and Feature Engineering

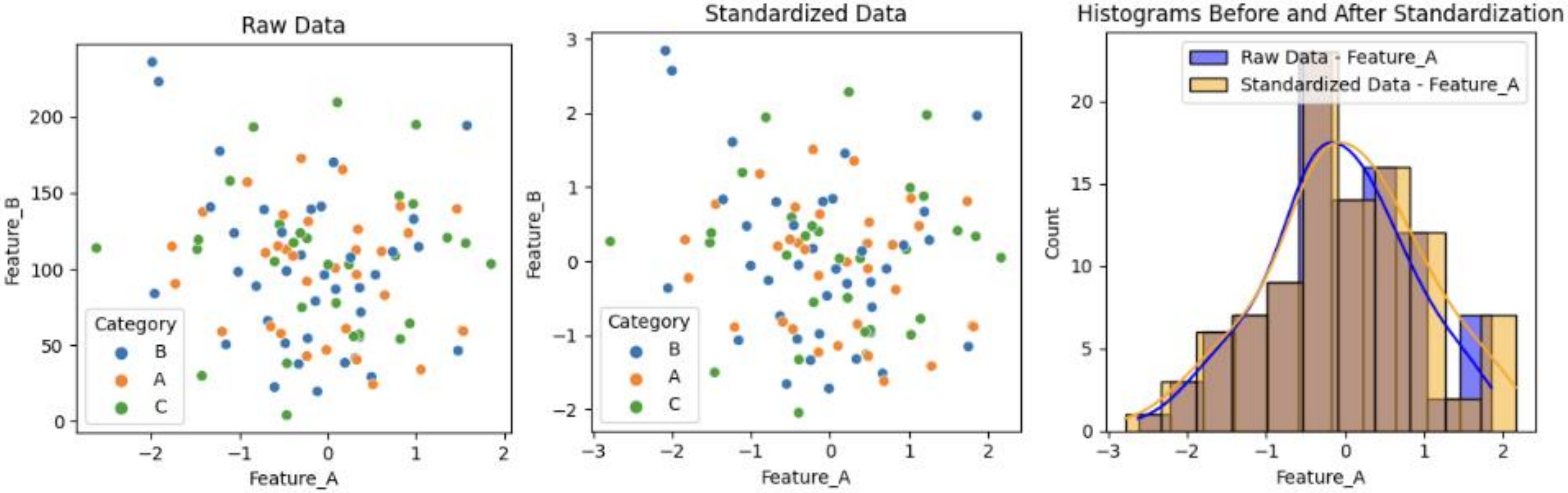
LECTURE

Data Standardization

Data Standardization

Ensuring data consistency and optimal model performance.

Data standardization is the process of creating standards and transforming data taken from different sources into a consistent format that adheres to the standards.



Benefits of Data Standardization

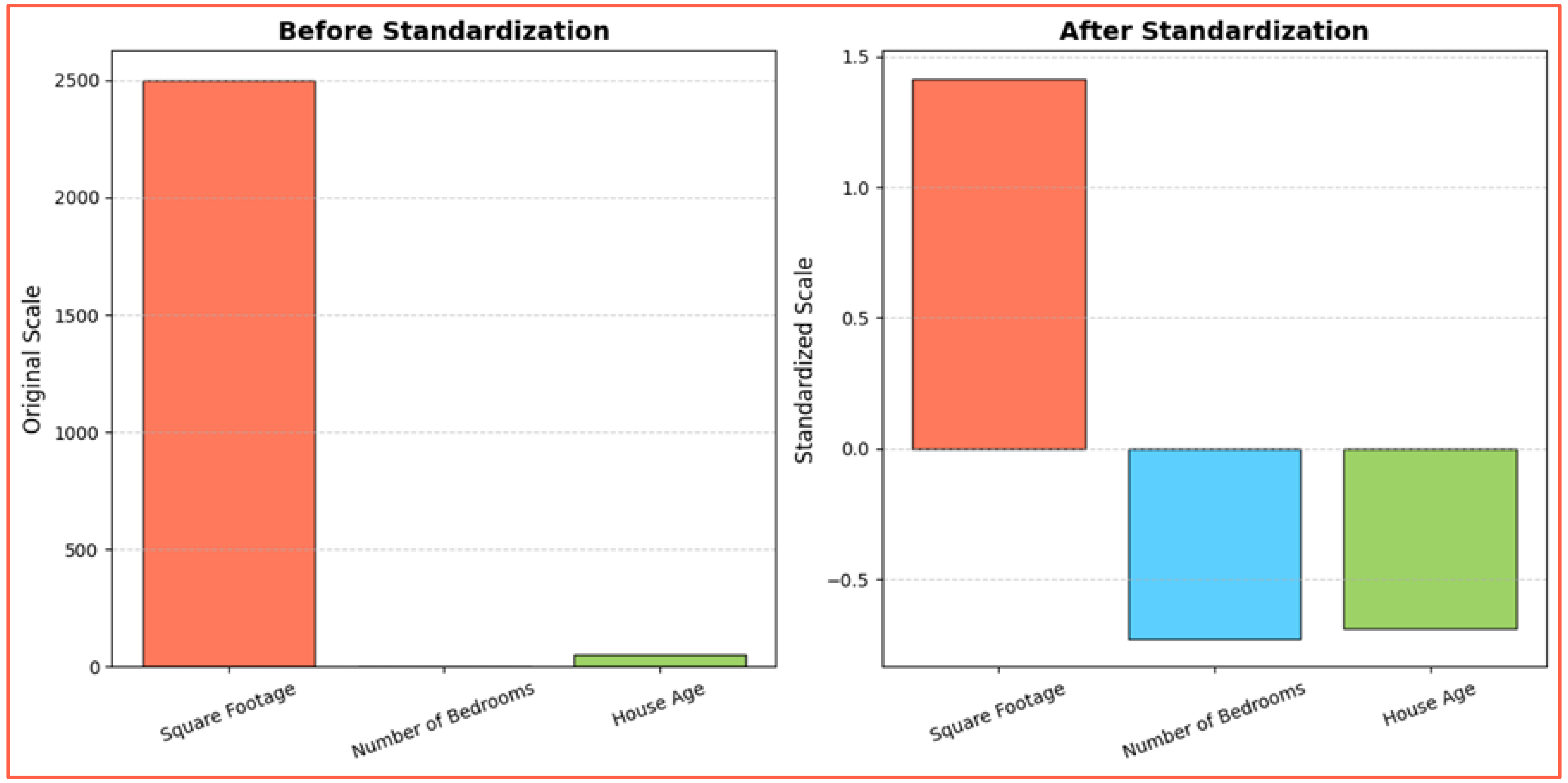
Standardizing features offers significant advantages

- Improved Interpretability
- Prevention of Model Biases
- Enhanced Convergence
- Effective Regularization
- Facilitates Comparison of Coefficients

Benefits of Data Standardization

Standardizing features offers significant advantages

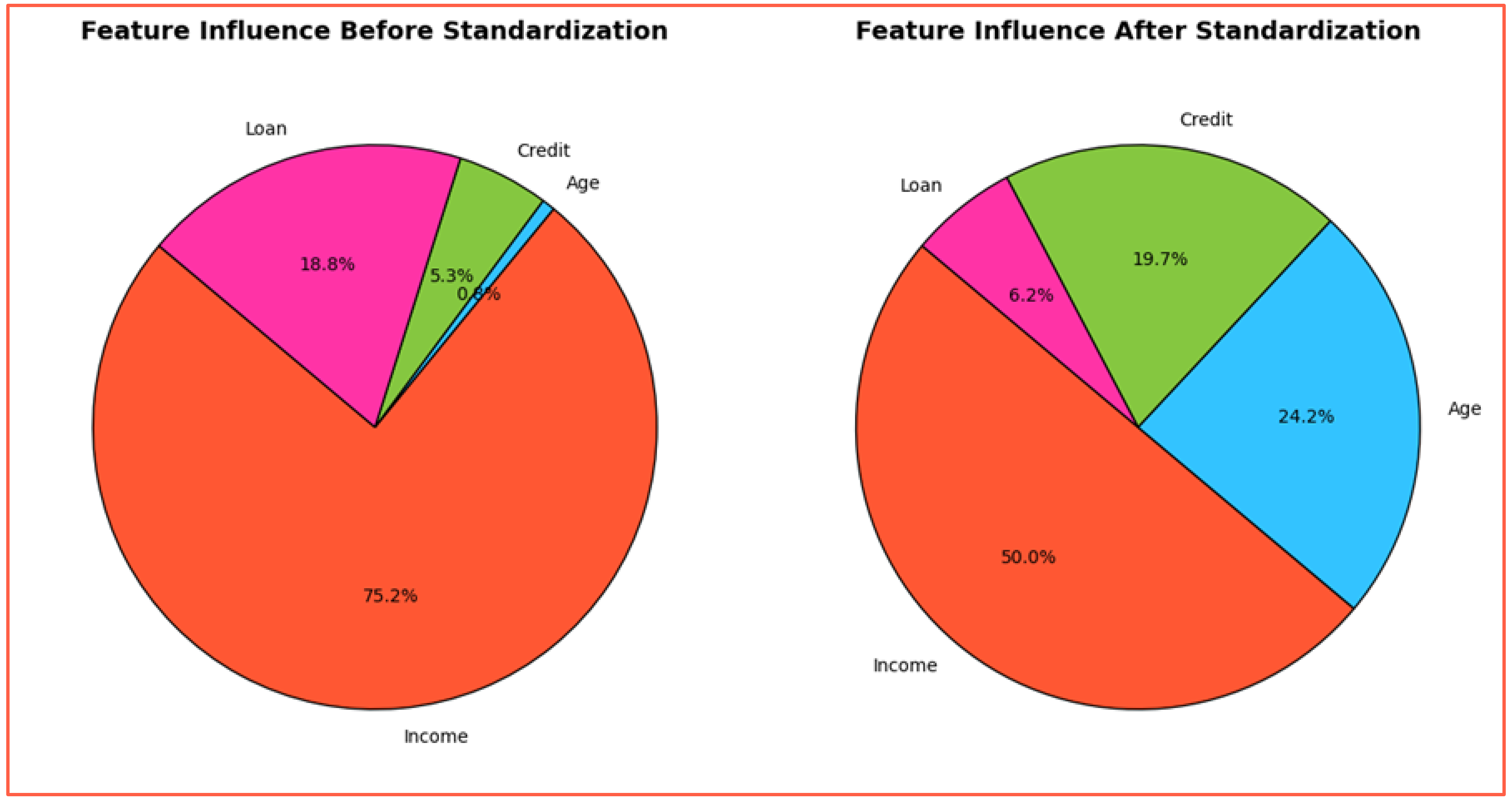
- Improved Interpretability
- Prevention of Model Biases
- Enhanced Convergence
- Effective Regularization
- Facilitates Comparison of Coefficients



Benefits of Data Standardization

Standardizing features offers significant advantages

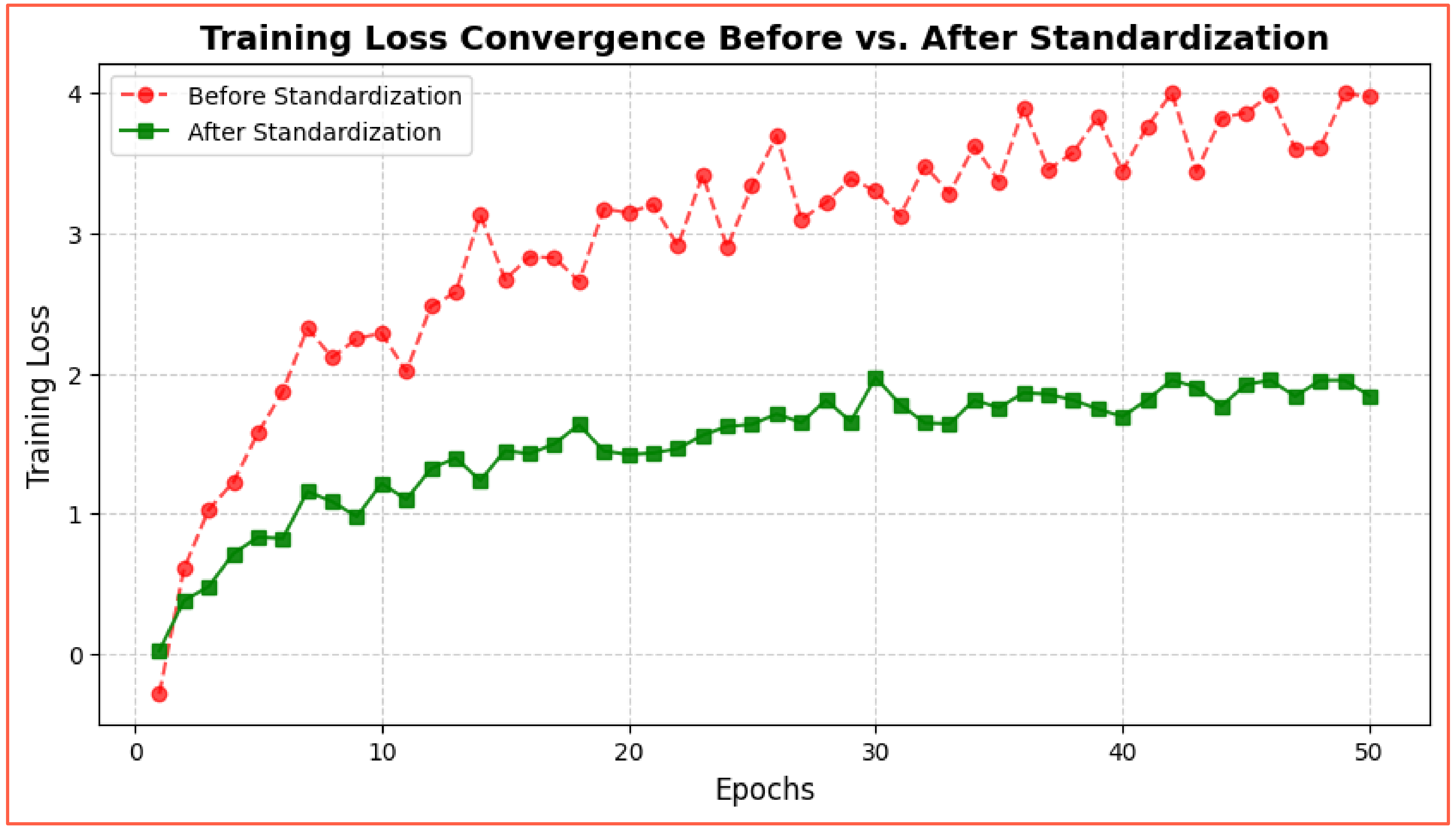
- Improved Interpretability
- **Prevention of Model Biases**
- Enhanced Convergence
- Effective Regularization
- Facilitates Comparison of Coefficients



Benefits of Data Standardization

Standardizing features offers significant advantages

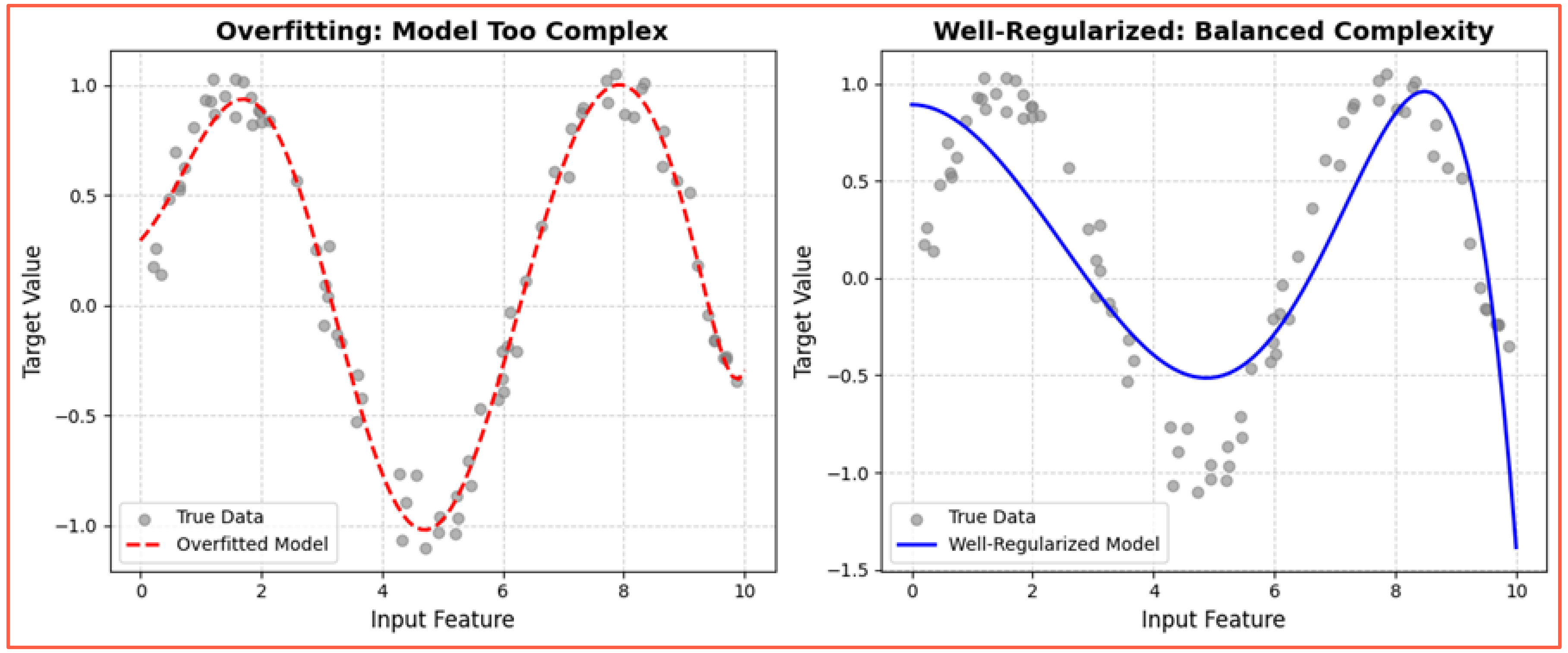
- Improved Interpretability
- Prevention of Model Biases
- **Enhanced Convergence**
- Effective Regularization
- Facilitates Comparison of Coefficients



Benefits of Data Standardization

Standardizing features offers significant advantages

- Improved Interpretability
- Prevention of Model Biases
- Enhanced Convergence
- **Effective Regularization**
- Facilitates Comparison of Coefficients



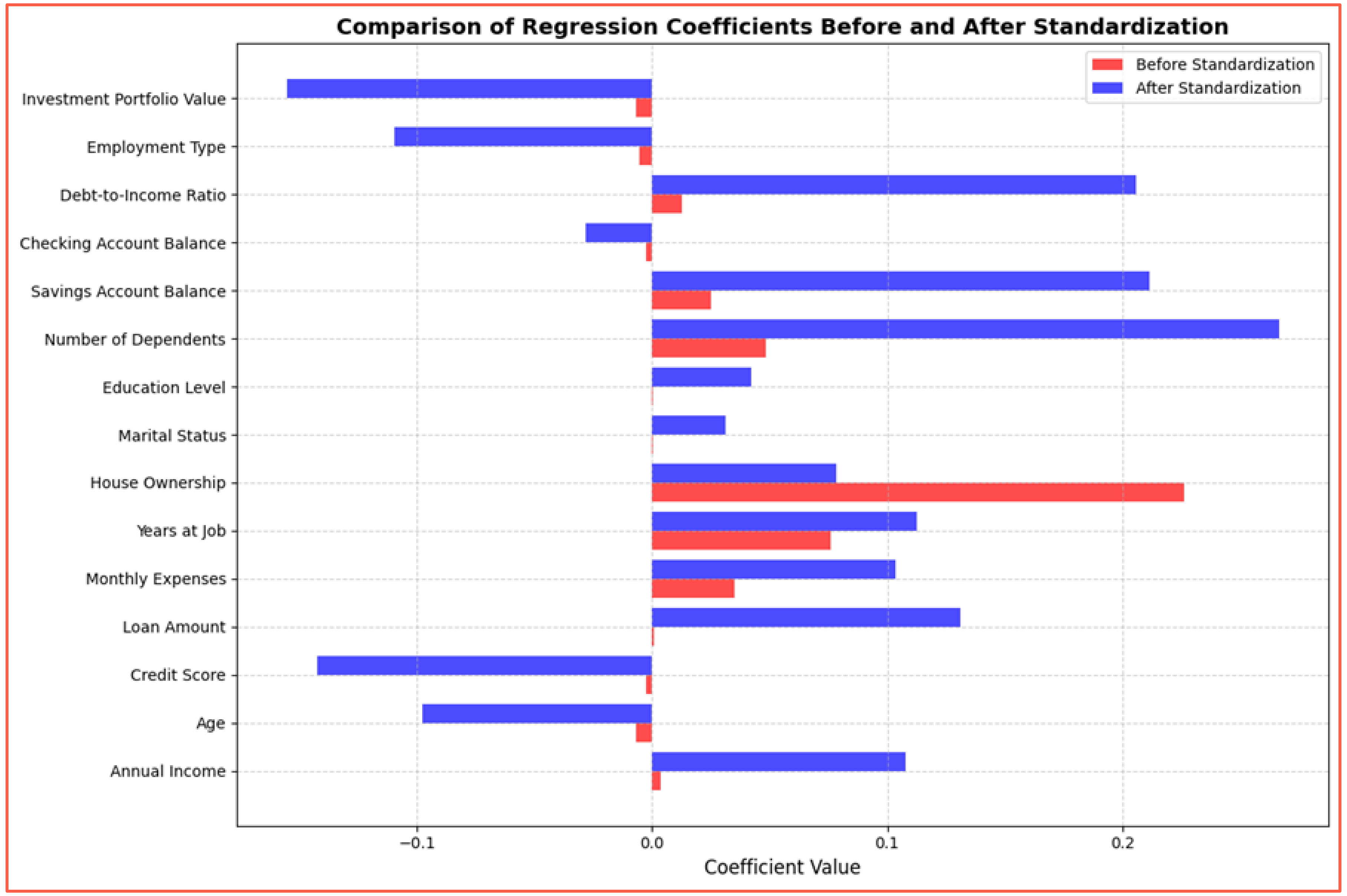
Overfitting: Model Too Complex

Well-Regularized: Balanced Complexity

Benefits of Data Standardization

Standardizing features offers significant advantages

- Improved Interpretability
- Prevention of Model Biases
- Enhanced Convergence
- Effective Regularization
- **Facilitates Comparison of Coefficients**



Which Models are Sensitive or Robust to the Scale of the Feature?

Sensitivity or robustness of ML models to the scale of features can vary

Models Sensitive to Feature Scale:

- Linear Models
- Support Vector Machines (SVM)
- K-Nearest Neighbors (KNN)
- Neural Networks

Models Robust to Feature Scale:

- Decision Trees and Random Forests
- Ensemble Models
- Tree-based Models
- Principal Component Analysis (PCA)

Process of Data Standardization

Navigating the Steps to Standardize Data for Machine Learning

1. Assess feature distributions/scales and need to standardize
2. Prep-process + Hold-Out split (train/test then train/validation)
3. Create/Push-to Feature Store table(s)
4. Create standardization pipeline and fit to train/validation & test sets

Drawbacks of Data Standardization

Limitations in the Process of Data Standardization

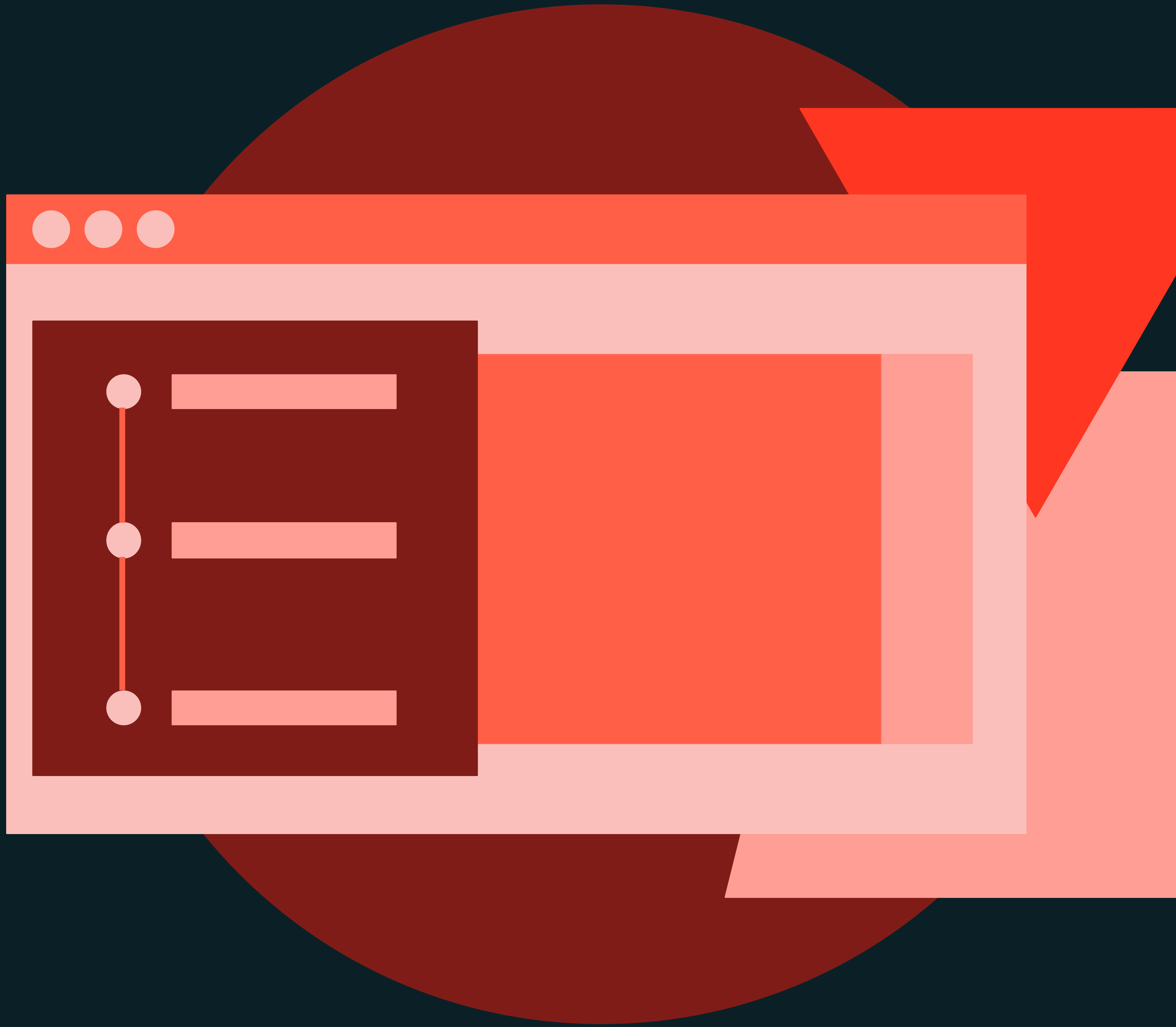
- Loss of Original Interpretation
- Challenges in Interpreting Interaction Terms
- Challenges in Result Interpretation
- *Assumption of Normally Distributed Features* (i.e. z-score)*

databricks

Data Preparation and Feature Engineering

DEMONSTRATION

Data Imputation and Transformation Pipeline



Demo

Outline

What we'll cover:

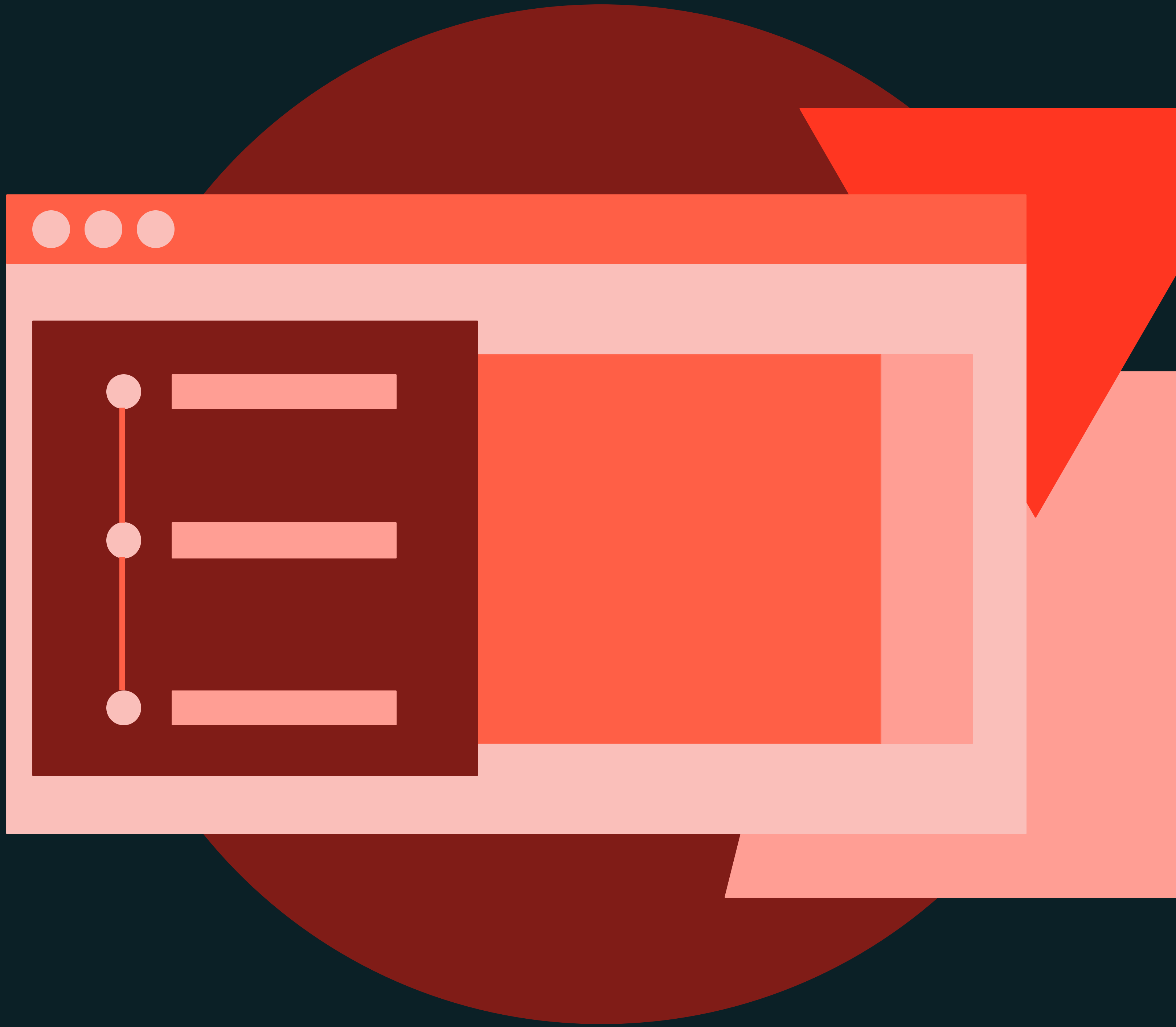
- Data cleaning and imputation
  - Coerce/fix data types
  - Handling outliers
  - Handling missing values
- Encoding categorical features
- Splitting data

databricks

Data Preparation and Feature Engineering

DEMONSTRATION

Build a Feature Engineering Pipeline with Embeddings



Demo

Outline

What we'll cover:

- Build a structured feature engineering pipeline that includes multiple preprocessing steps.
- Create a pipeline with tasks for data imputation and numerical feature scaling.
- Generate embeddings for categorical features to represent categorical data effectively.
- Apply the feature engineering pipeline to both training and test datasets.
- Save a data preparation and feature engineering pipeline to Unity Catalog for potential future use.

databricks

Data Preparation and Feature Engineering

LAB EXERCISE

Build a Feature Engineering Pipeline

Lab

Outline

What you'll do:

- Data preparation
- Split dataset
- Create a pipeline using data imputation and transformation
- Fit the pipeline
- Show transformation results
- Save the pipeline

databricks

Feature Store

Data Preparation for Machine Learning

Learning objectives

Things you'll be able to do after completing this module

- Understand Feature Store concepts and its benefits/importance in machine learning (*including improved collaboration and data consistency*).
- Explain the relationship between Feature Store tables and Unity Catalog (*and compare legacy/workspace feature store and Unity Catalog feature store*)
- Using Feature Store for Feature Engineering by creating a feature store table

databricks

Feature Store

LECTURE

Introduction to Feature Store

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:

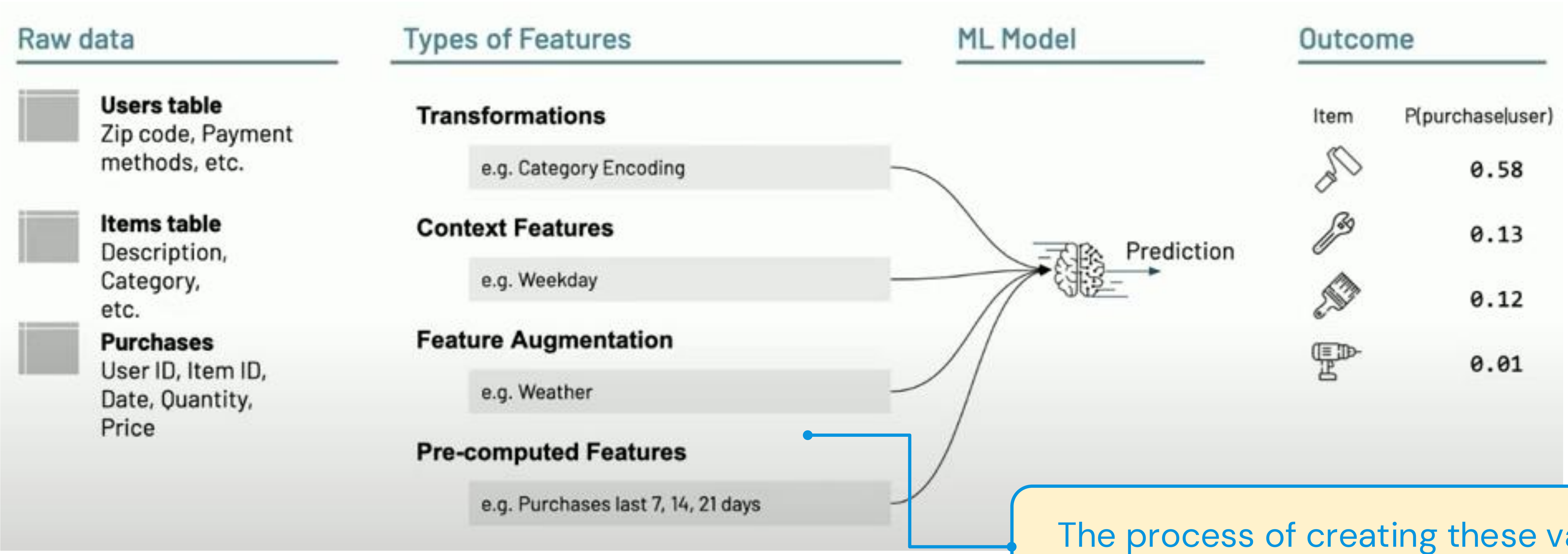
- Aggregations of raw data over time windows, like **trailing 7-day purchases**
- Joined **combinations of data sets**, like customer demographic information joined to transaction features
- Complex functions of customer information, like **estimated customer lifetime value**

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

What is a Feature Store?

An example of a recommendation system.

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



Raw data

Users table
Zip code, Payment methods, etc.

Items table
Description, Category, etc.

Purchases
User ID, Item ID, Date, Quantity, Price

Types of Features

Transformations
e.g. Category Encoding

Context Features
e.g. Weekday

Feature Augmentation
e.g. Weather

Pre-computed Features
e.g. Purchases last 7, 14, 21 days

ML Model

Prediction

Outcome

Item	P(purchase|user)
	0.58
	0.13
	0.12
	0.01

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

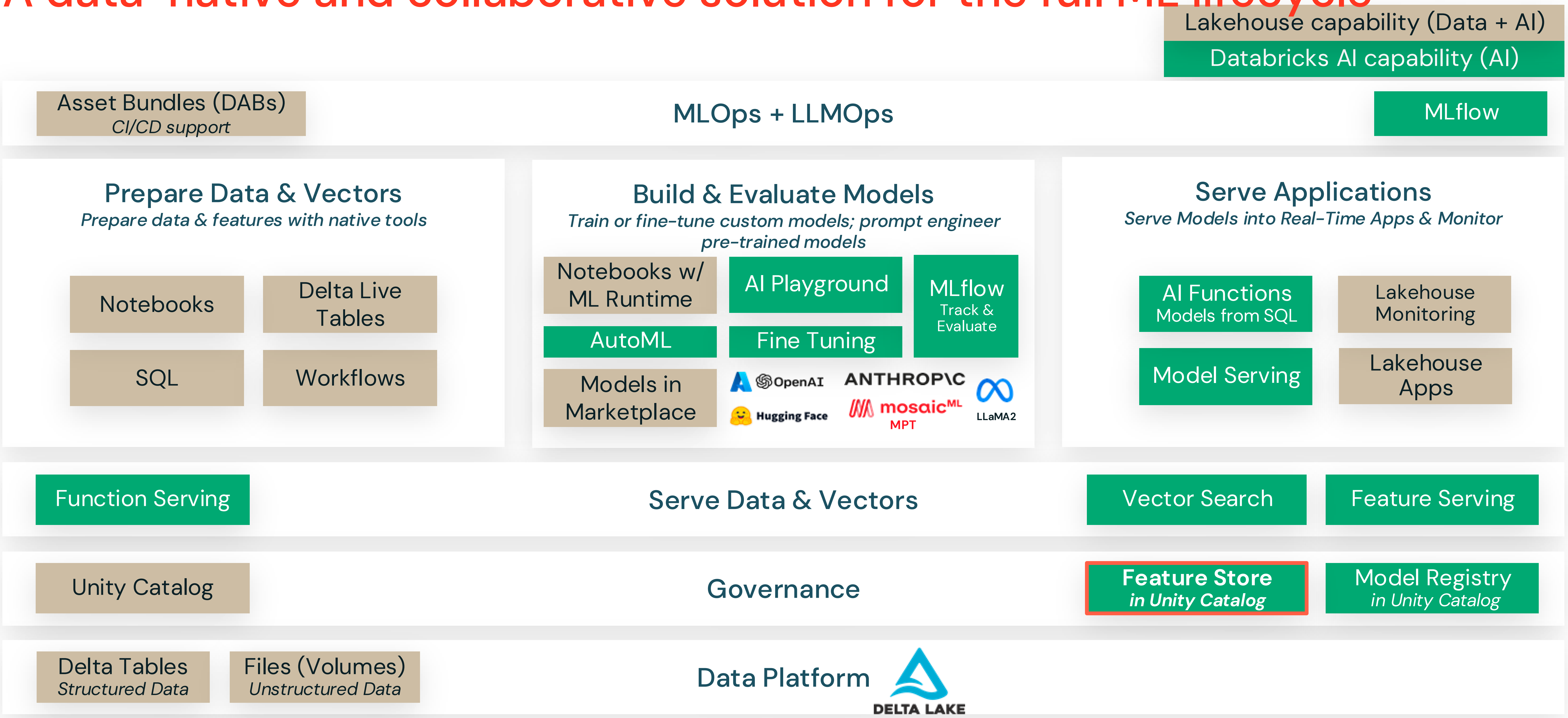
When multiple teams manage feature computation and ML models in production, minor yet significant **skew in upstream data** at the input of a feature pipeline can be very hard to detect and fix.

Online Serving

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

Databricks for Machine Learning

A data-native and collaborative solution for the full ML lifecycle



Lakehouse capability (Data + AI)

Databricks AI capability (AI)

MLOps + LLMOps

- Asset Bundles (DABs) *CI/CD support*
- MLflow

Prepare Data & Vectors

Prepare data & features with native tools

- Notebooks
- Delta Live Tables
- SQL
- Workflows

Build & Evaluate Models

Train or fine-tune custom models; prompt engineer pre-trained models

- Notebooks w/ ML Runtime
- AI Playground
- MLflow Track & Evaluate
- AutoML
- Fine Tuning
- Models in Marketplace

Serve Applications

Serve Models into Real-Time Apps & Monitor

- AI Functions Models from SQL
- Lakehouse Monitoring
- Model Serving
- Lakehouse Apps

Serve Data & Vectors

- Function Serving
- Vector Search
- Feature Serving

Governance

- Unity Catalog
- **Feature Store** ***in Unity Catalog***
- Model Registry *in Unity Catalog*

Data Platform

- Delta Tables *Structured Data*
- Files (Volumes) *Unstructured Data*

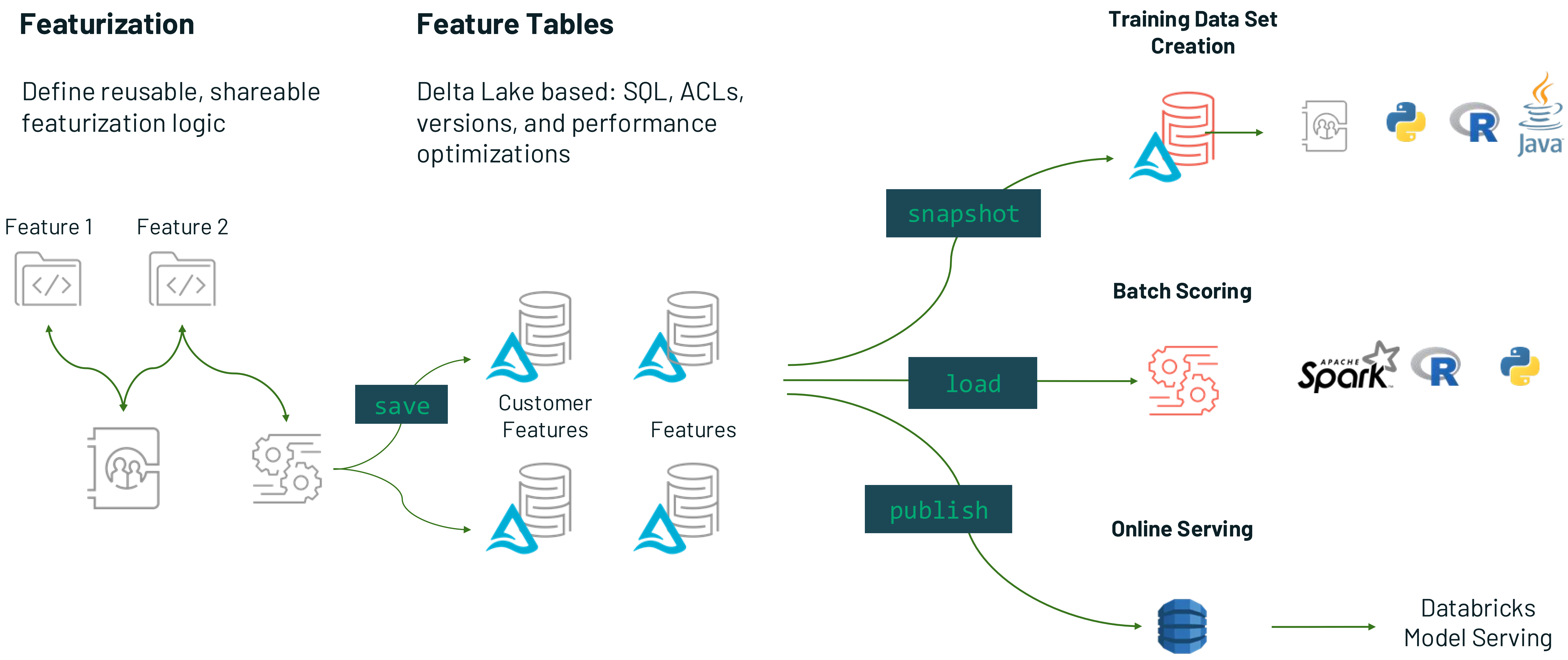
Databricks Feature Store

Featurization

Define reusable, shareable featurization logic

Feature Tables

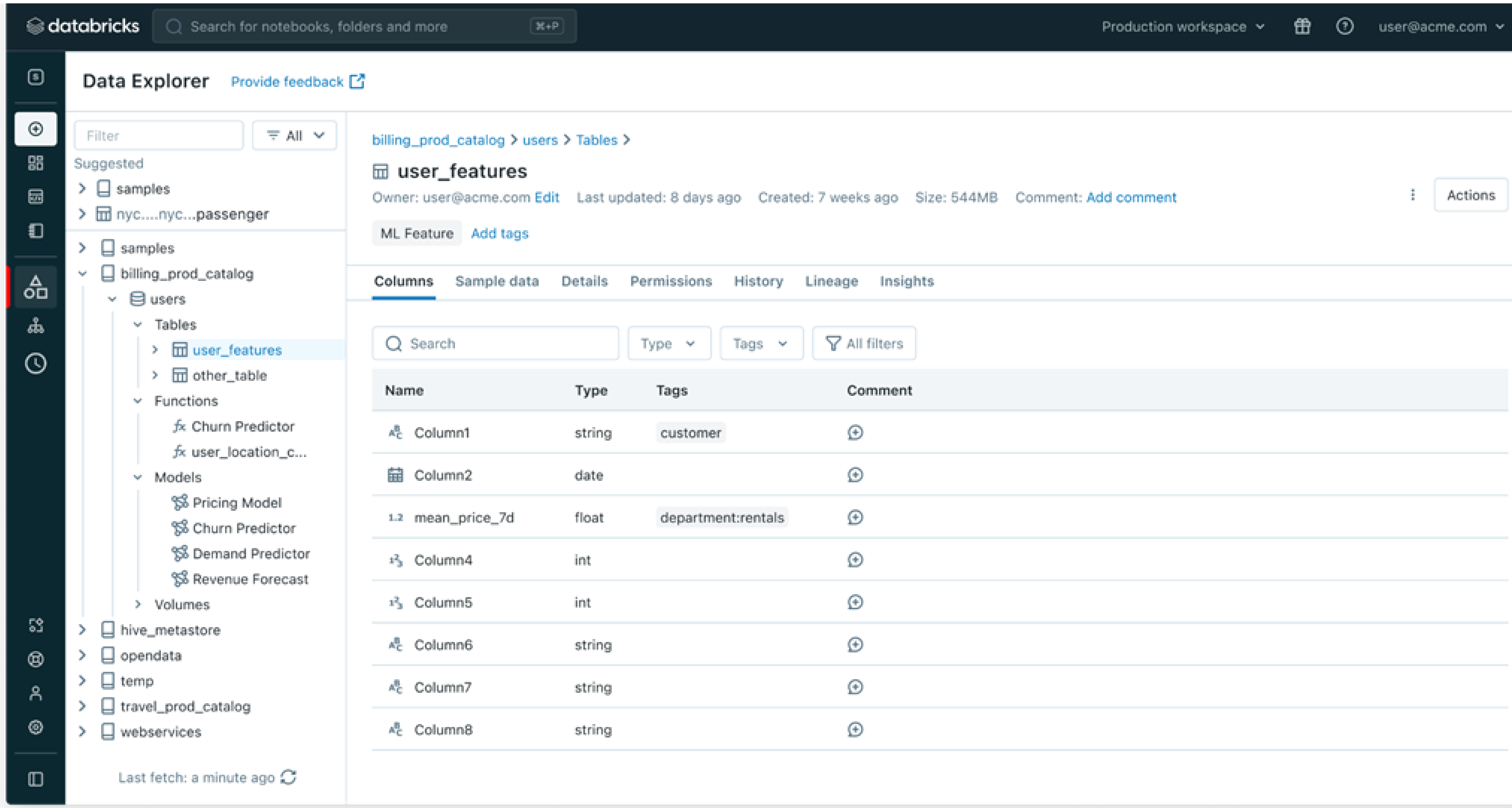
Delta Lake based: SQL, ACLs, versions, and performance optimizations



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



user_features

Owner: user@acme.com Edit   Last updated: 8 days ago   Created: 7 weeks ago   Size: 544MB   Comment: Add comment

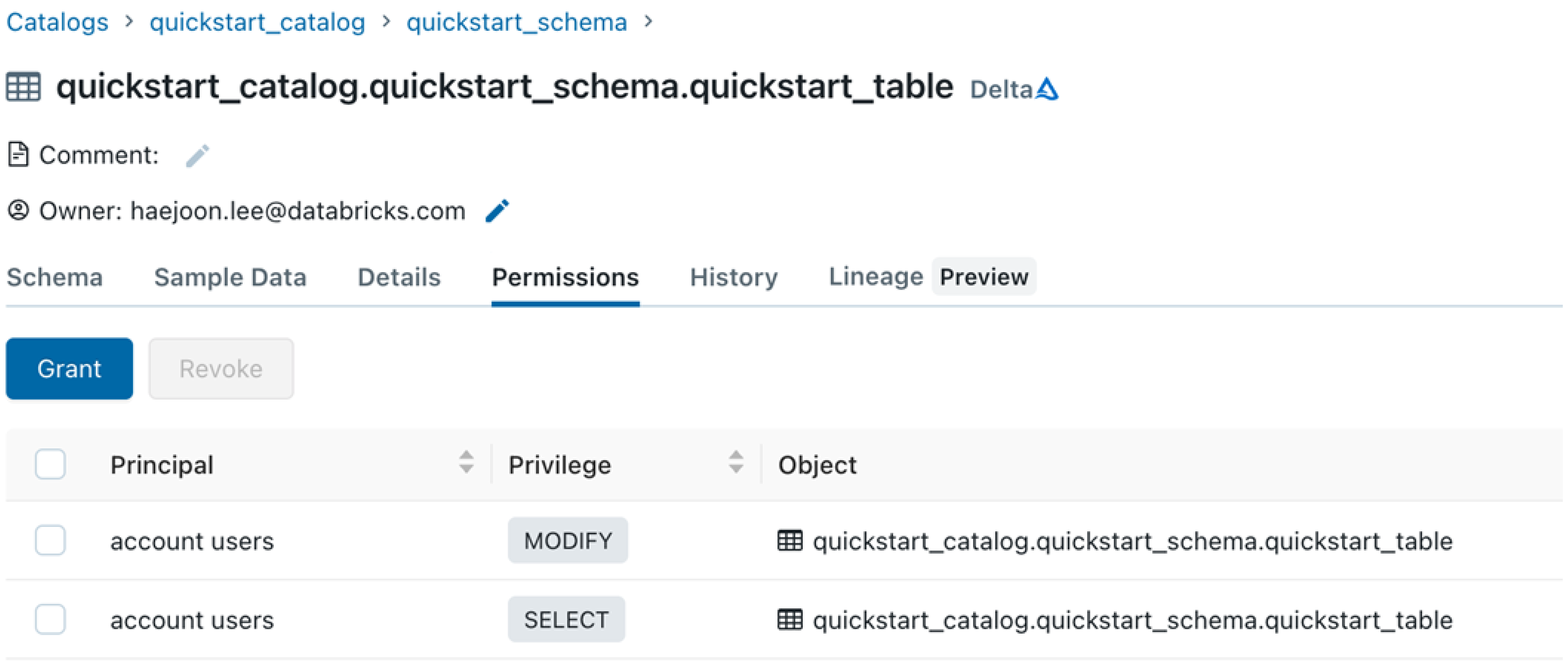
ML Feature

Name	Type	Tags	Comment
Column1	string	customer	⊕
Column2	date		⊕
mean_price_7d	float	department:rentals	⊕
Column4	int		⊕
Column5	int		⊕
Column6	string		⊕
Column7	string		⊕
Column8	string		⊕

Unified Permission Model

Secure features with built-in governance

- Feature data and metadata are governed by the Unity Catalog permission model.
- Future improvements to data governance in Unity Catalog will apply to Feature data.



quickstart_catalog.quickstart_schema.quickstart_table

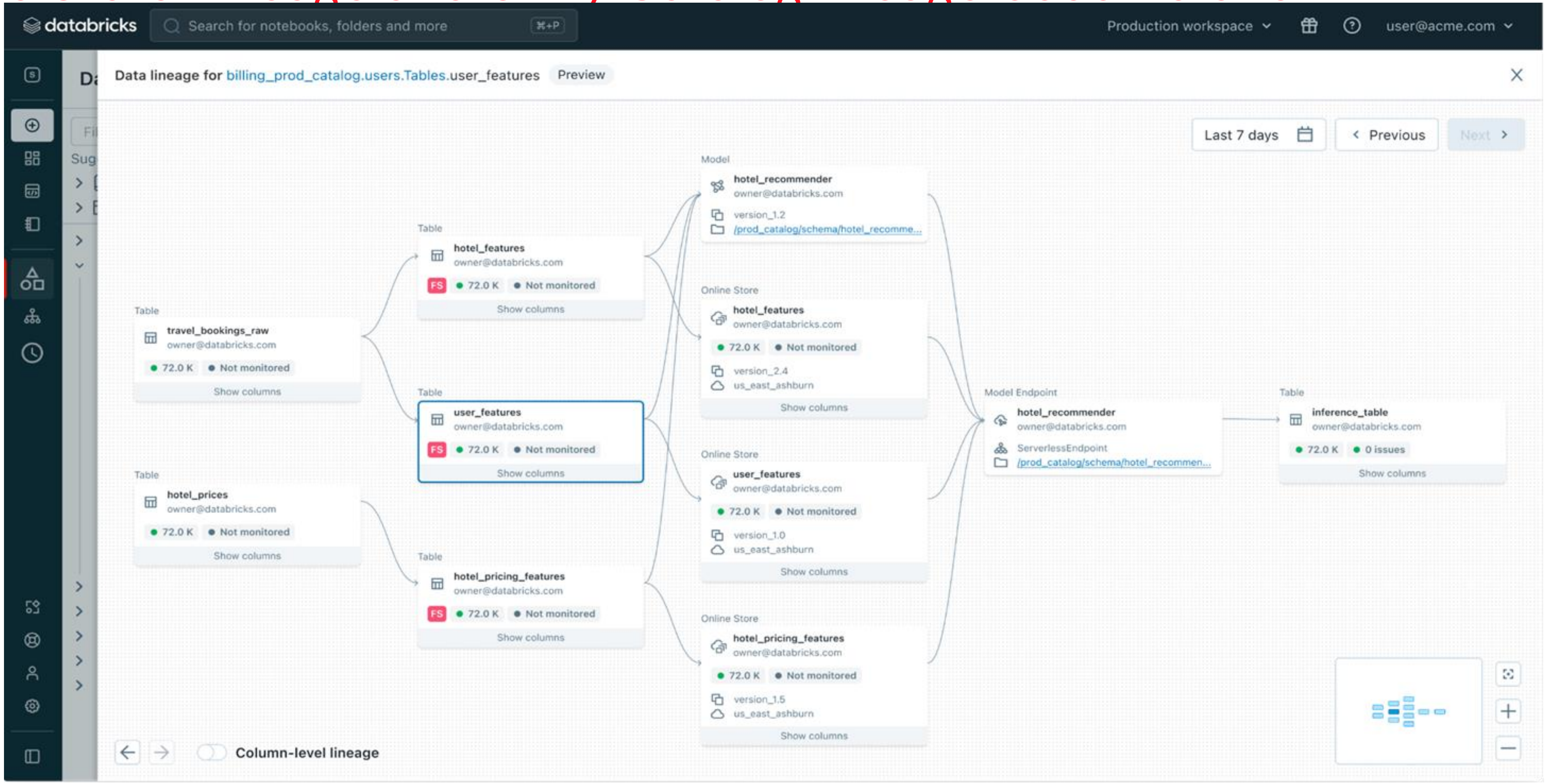
Comment:

Owner: haejoon.lee@databricks.com

□	Principal	Privilege	Object
□	account users	MODIFY	quickstart_catalog.quickstart_schema.quickstart_table
□	account users	SELECT	quickstart_catalog.quickstart_schema.quickstart_table

Unified Data Lineage

Feature Store Lineage and Unity Catalog lineage become one



Data lineage for billing_prod_catalog.users.Tables.user_features Preview

Integration with MLflow and Model Serving

Makes model deployment easier

Integration with MLflow

When a features from FS are used for training, **the model is packaged with feature metadata**.

In inference time, the model can look up features at runtime.

Integration with Model Serving

When the model is used for batch scoring or online inference, **it automatically retrieves features from Feature Store**.

Workspace FS vs. Unity Catalog FS

Workspace FS

- Specific to a single workspace
- Challenges in sharing feature tables across workspaces
- Uses `FeatureStoreClient` to create feature tables.

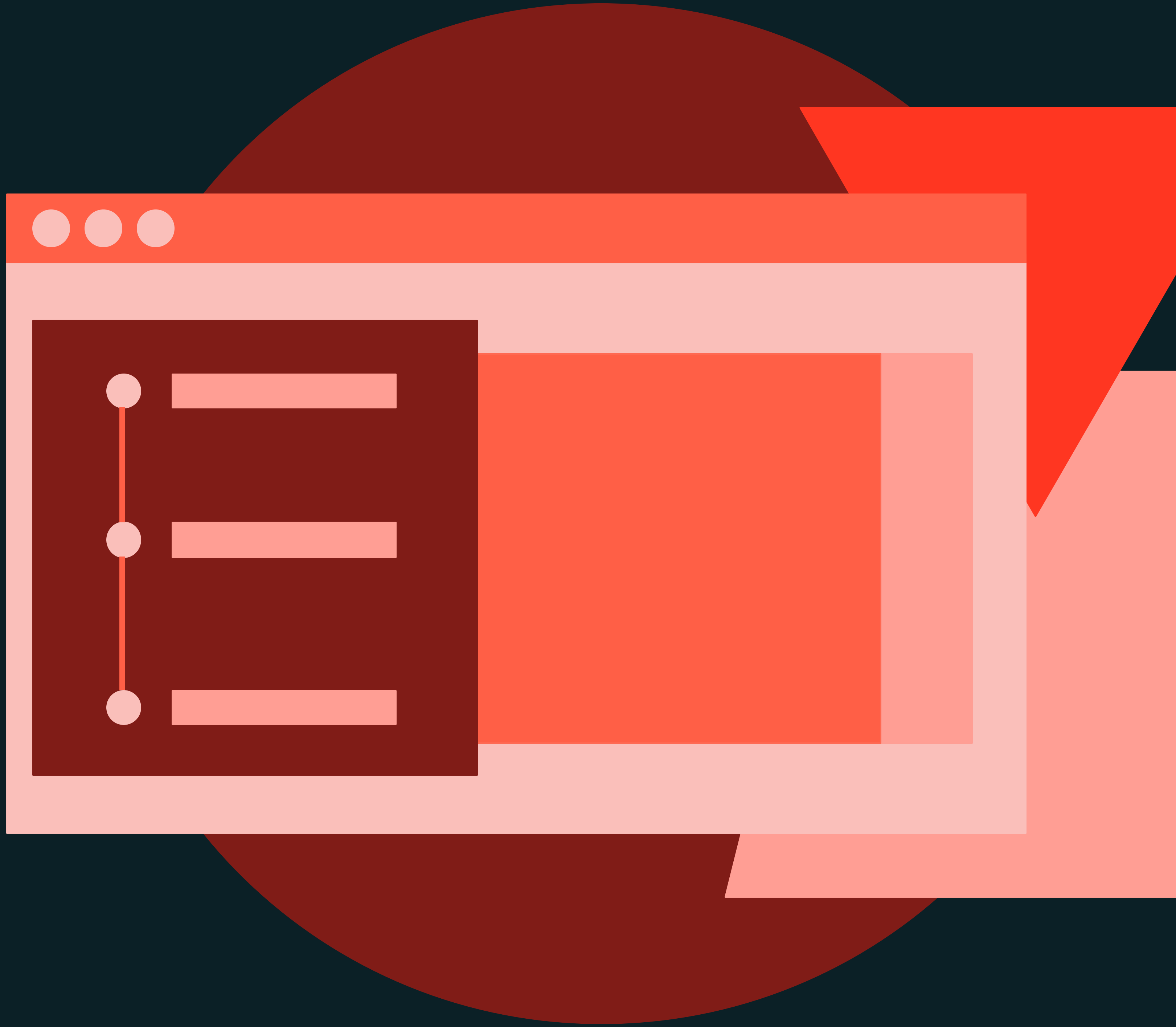
Unity Catalog FS (Recommended)

- Any Delta table with a primary key can be a feature table (DBR 13.2+)
- All UC capabilities are available; discovery, governance, lineage and cross-workspace access.
- Uses `FeatureEngineeringClient` to create feature tables.

Feature Store

DEMONSTRATION

Using Feature Store for Feature Engineering



Demo

Outline

What we'll cover:

- Feature engineering
- Save features to feature table
- Create a feature table from existing UC table
- Update features in a feature table
- Optional: Migrate workspace feature table to UC

databricks

Feature Store

LAB EXERCISE

Feature Engineering with Feature Store

Lab

Outline

What you’ll do:

- Data preparation and feature engineering
- Create a feature table
- Explore feature table with the UI
- Create a feature table from an existing UC table

Course Summary and Next Steps

Data Preparation for Machine Learning

databricks