

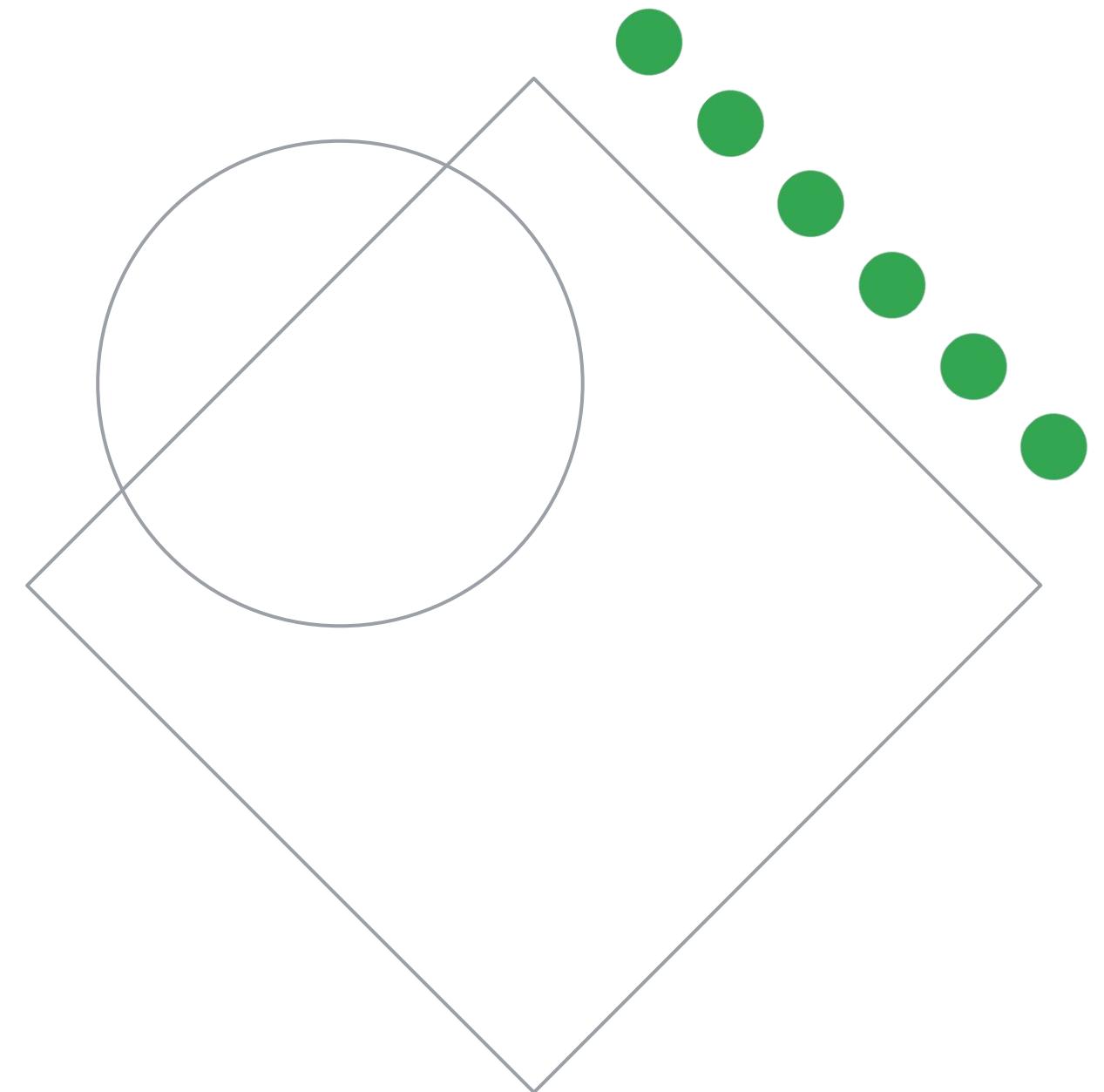
# Feature Engineering

# In this course, you learn to ...

- 01 Use the Vertex AI Feature Store
- 02 Describe how to move from raw data to features
- 03 Perform Feature Engineering in BigQuery ML and Keras
- 04 Preprocess features using Apache Beam and Cloud DataFlow
- 05 Use tf.Transform



# Introduction to Vertex AI Feature Store



# In this module, you learn to ...

- 01 Understand the benefits of Feature Store
- 02 Define Feature Store terminology and concepts
- 03 Summarize the Feature Store data model
- 04 Create a featurestore
- 05 Describe how to serve batch and online requests



# Feature Store **benefits**

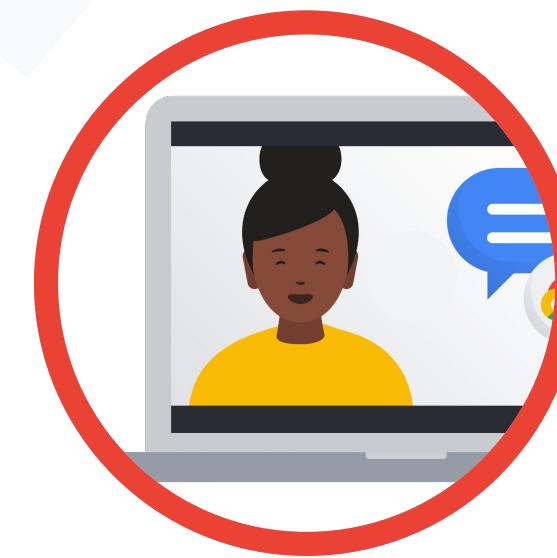
# Business use case



Software developer



Data analyst



Data scientist

Multiple teams are re-creating the same features...

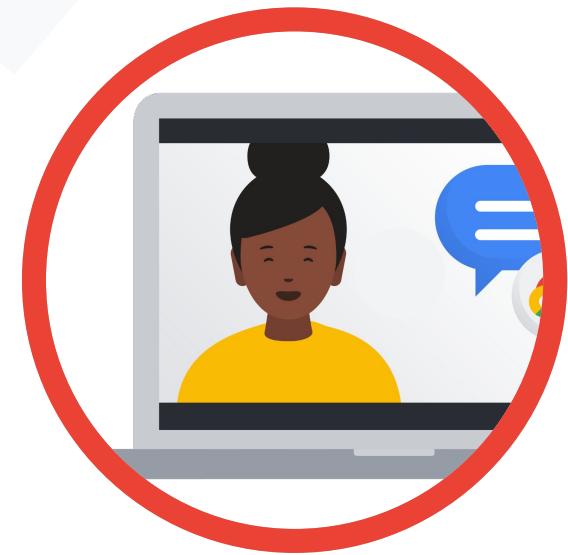
# Business use case



Software developer



Data analyst



Data scientist

...and we're dependent on the Ops team to add new features to models.

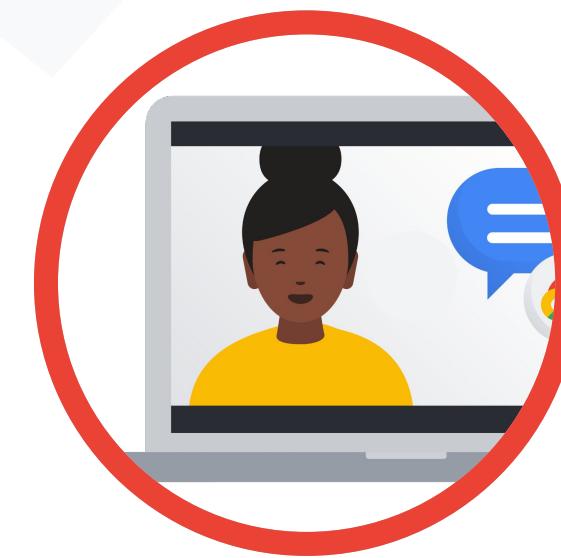
# Business use case



Software developer



Data analyst

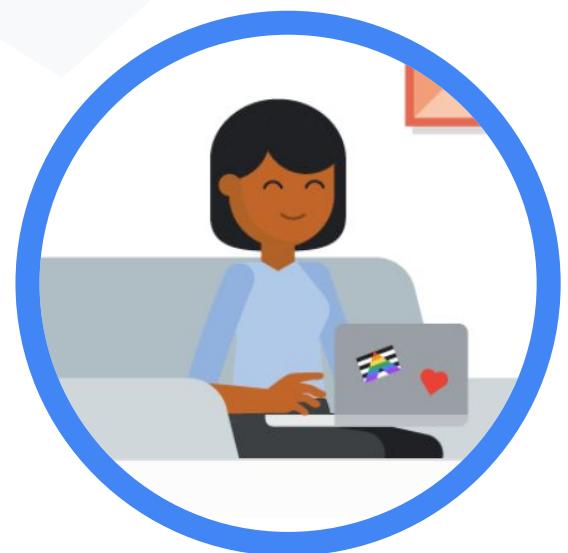


Data scientist

We're spending too much time trying to manage the lifecycle of a feature: from training to deployment and serving.

# Business use case

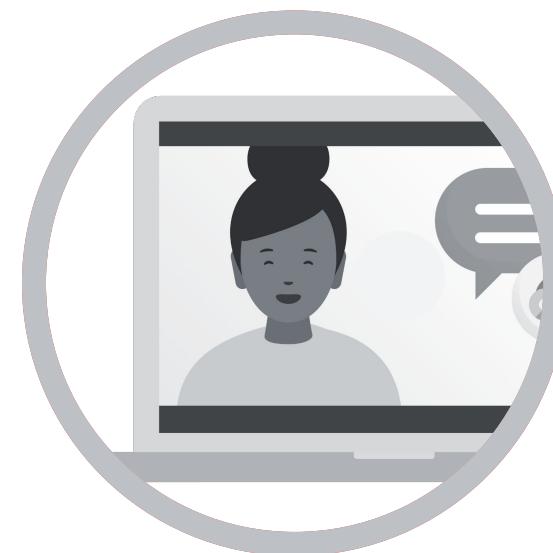
I need to understand how to add ML to our apps via online prediction endpoints without having to figure out how to generate features.



Software developer



Data analyst



Data scientist

# Business use case

I'm new to the XYZ team. I need to understand how to monitor and maintain feature serving and management infrastructure.



DevOps



Software developer



Data analyst



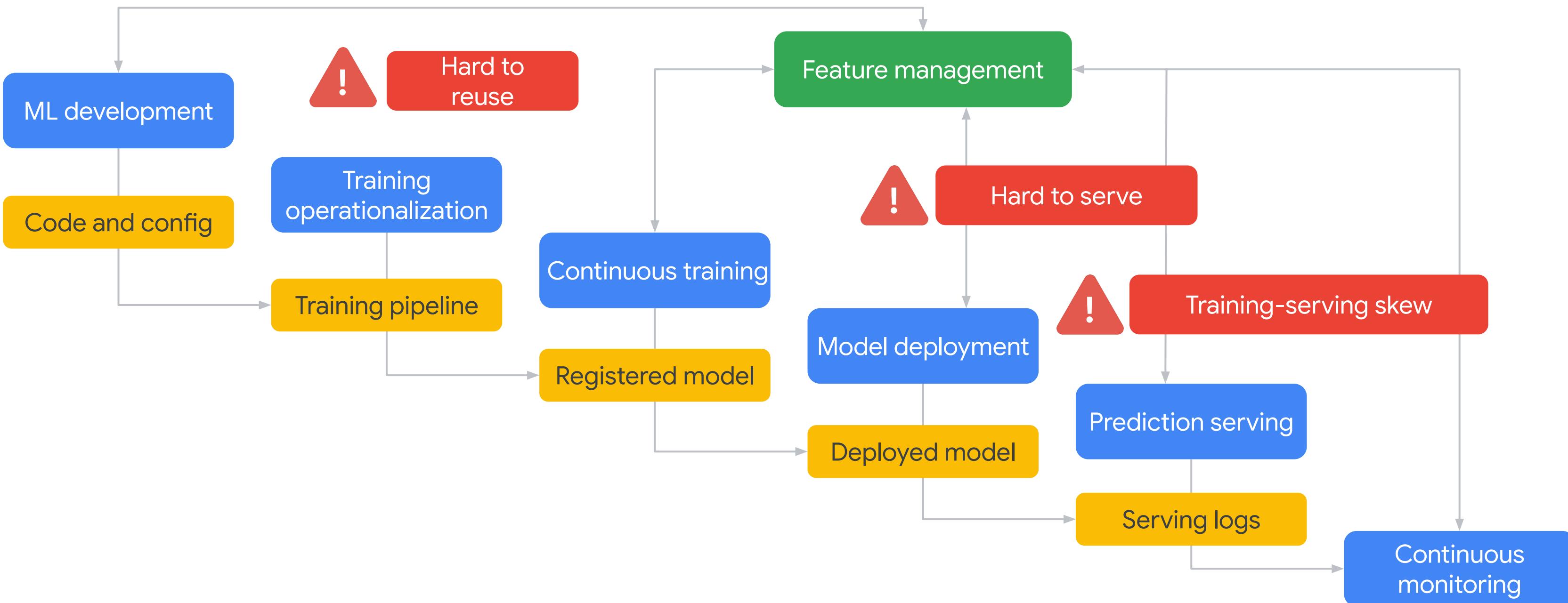
Data scientist

# Feature challenges

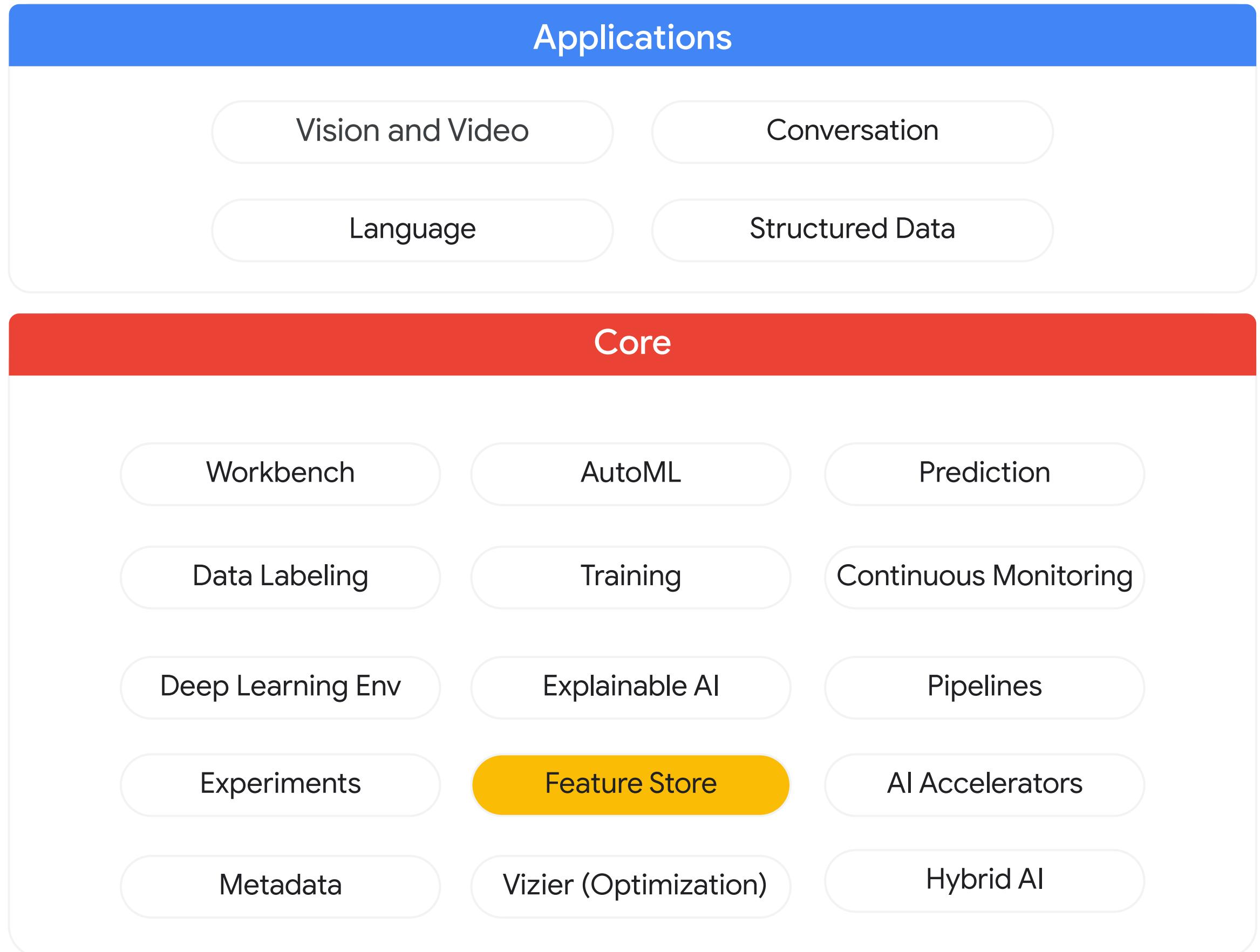
- Hard to share and reuse
- Hard to reliably serve in production with low latency
- Inadvertent skew in feature values between training and serving



# Feature management pain points

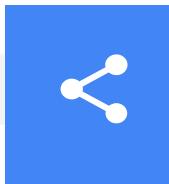


# Vertex AI



# Fully managed Feature Store

A rich feature repository to serve, share, and re-use ML features.



## Share and reuse ML features across use cases

Centralized feature repository with easy APIs to search and discover features, fetch them for training/serving, and manage permissions.



## Serve ML features at scale with low latency

Offload the operational overhead of handling infrastructure for low-latency scalable feature serving.



## Alleviate training-serving skew

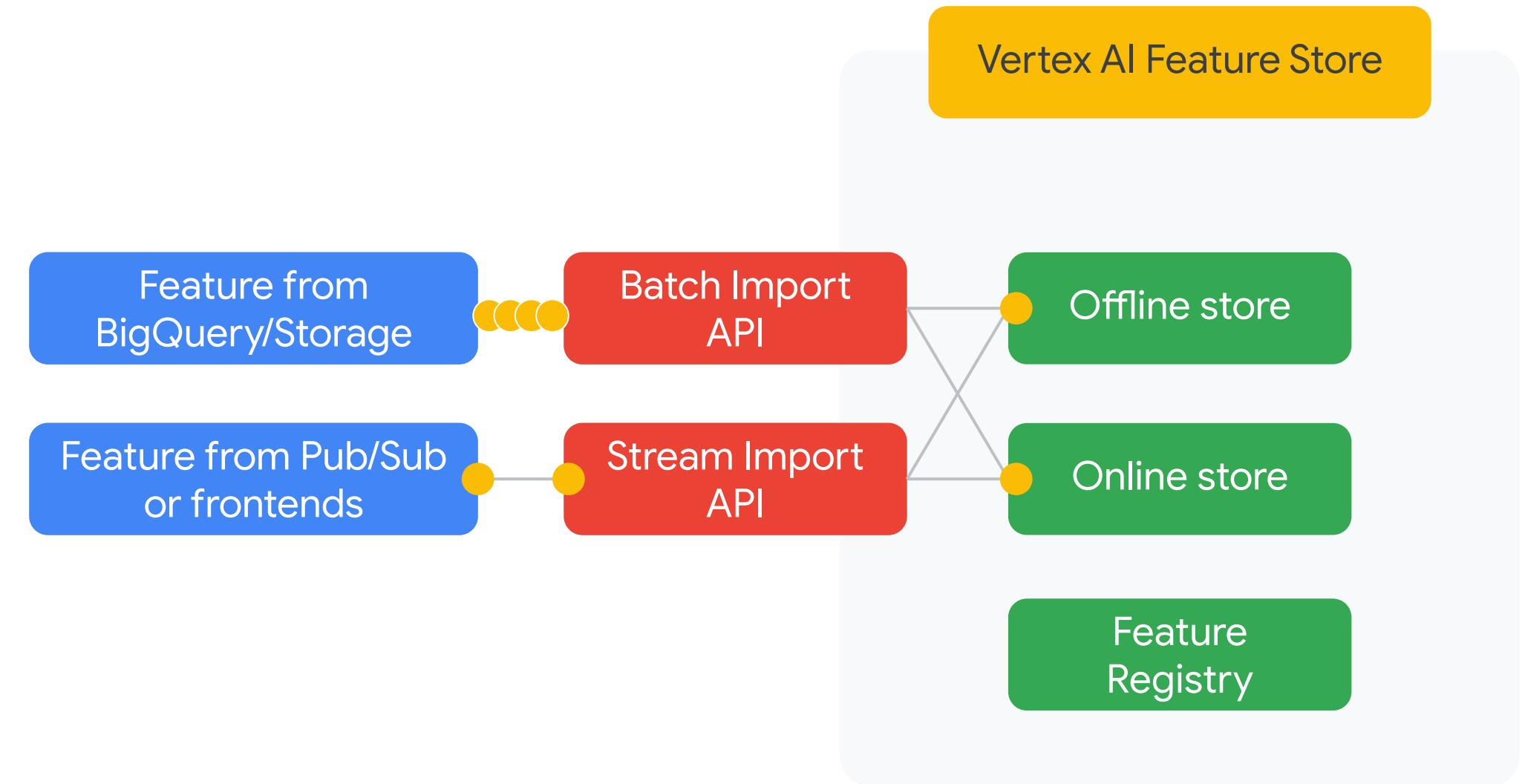
Compute feature values once; re-use for training and serving. Track and monitor for drift and other quality issues.



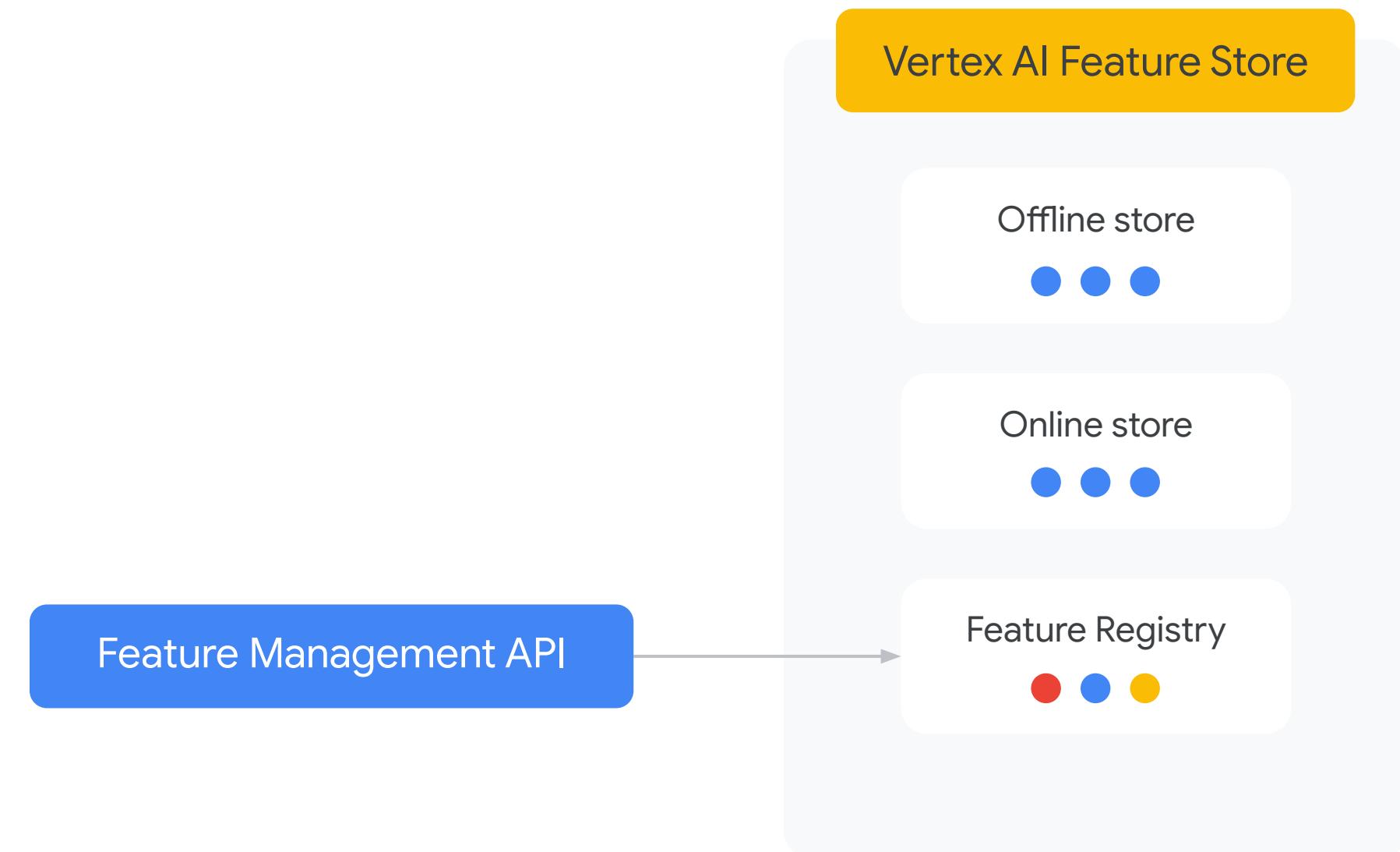
## Use batch and streaming feature ingestion

Ingest features efficiently in large batches or in real time as data streams in. ETA for streaming ingestion is Q3 2021.

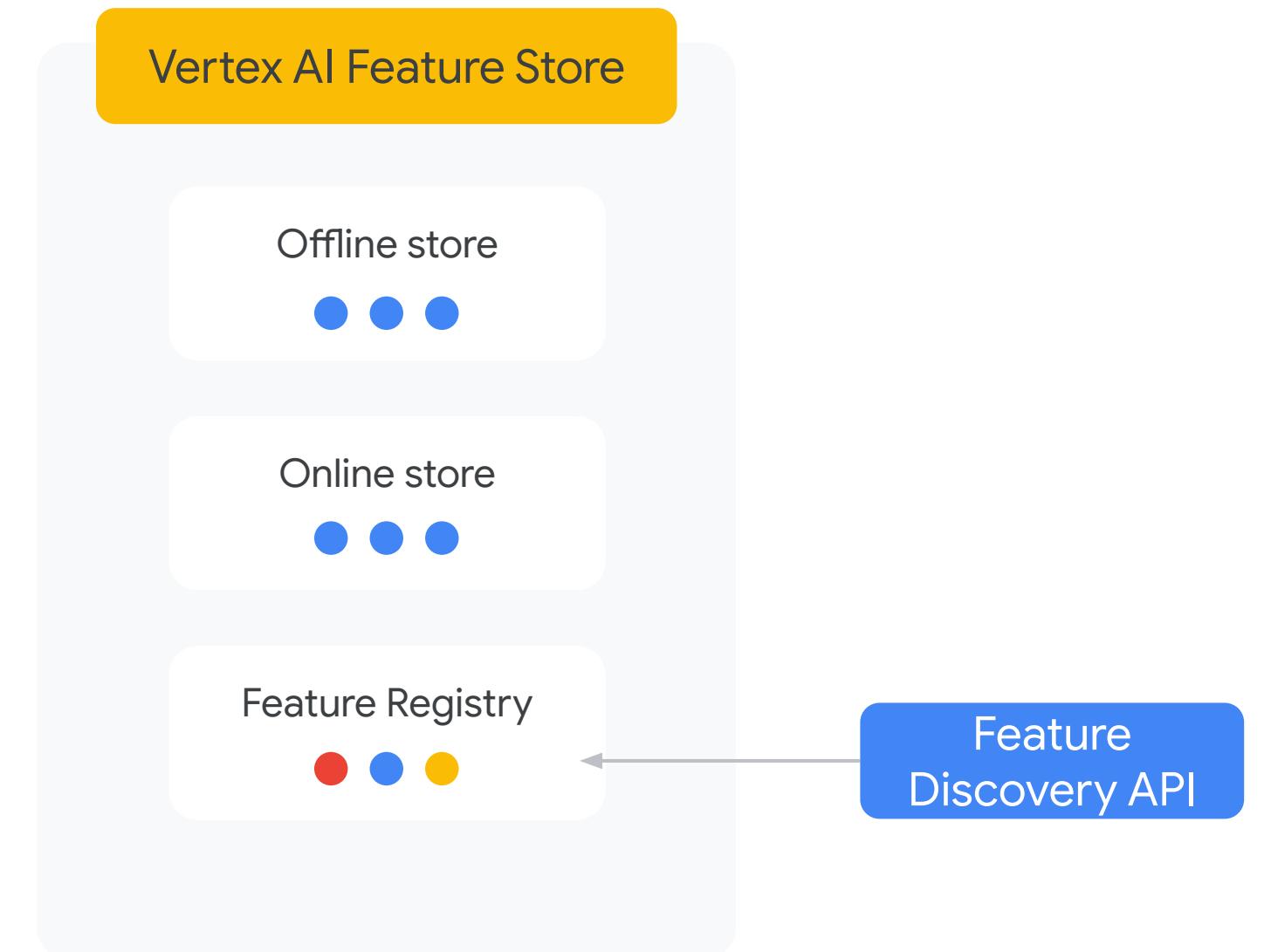
**With Vertex AI Feature Store, you can store features with Batch and Stream Import APIs...**



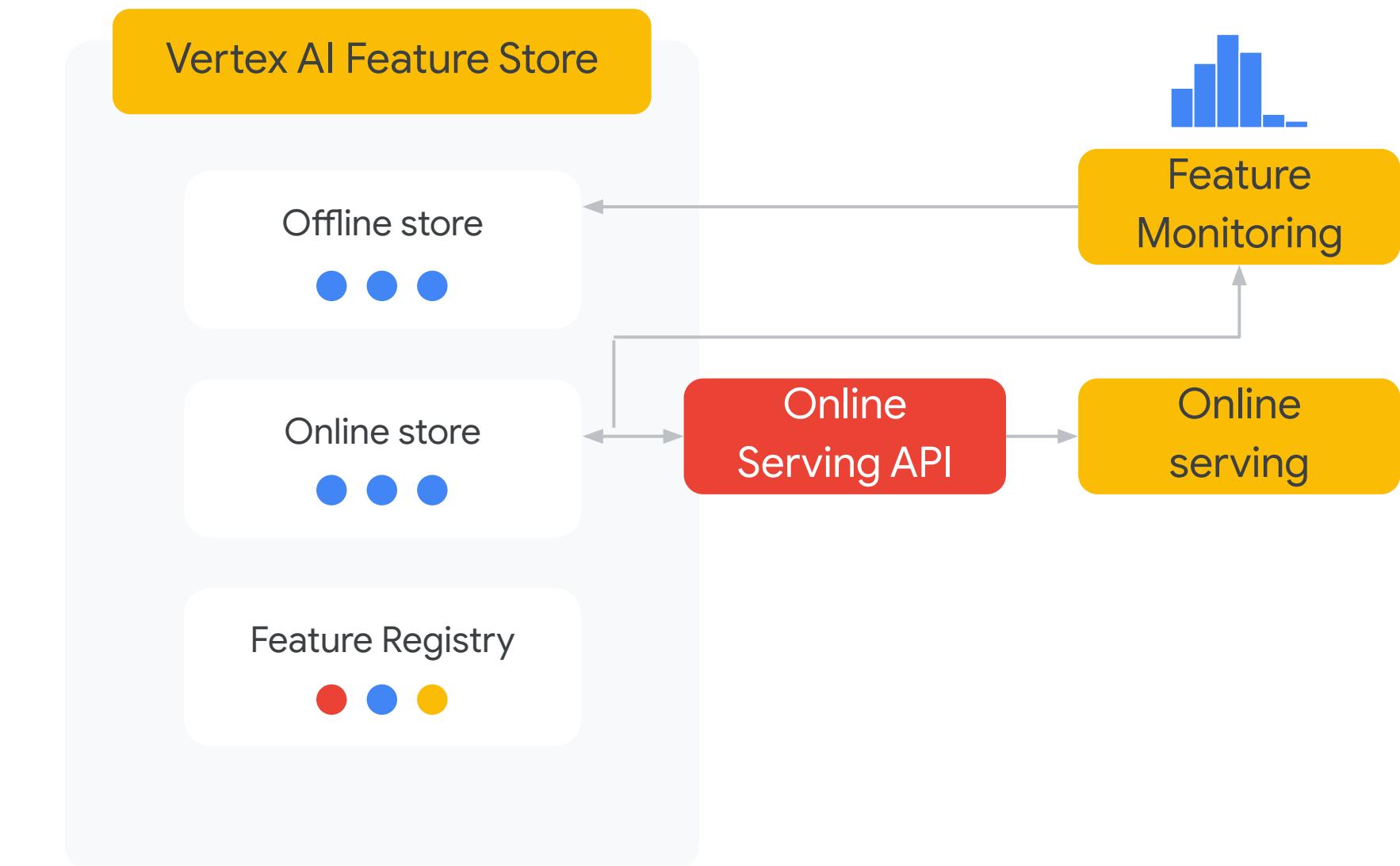
**...and register the  
feature to its  
Feature Registry**



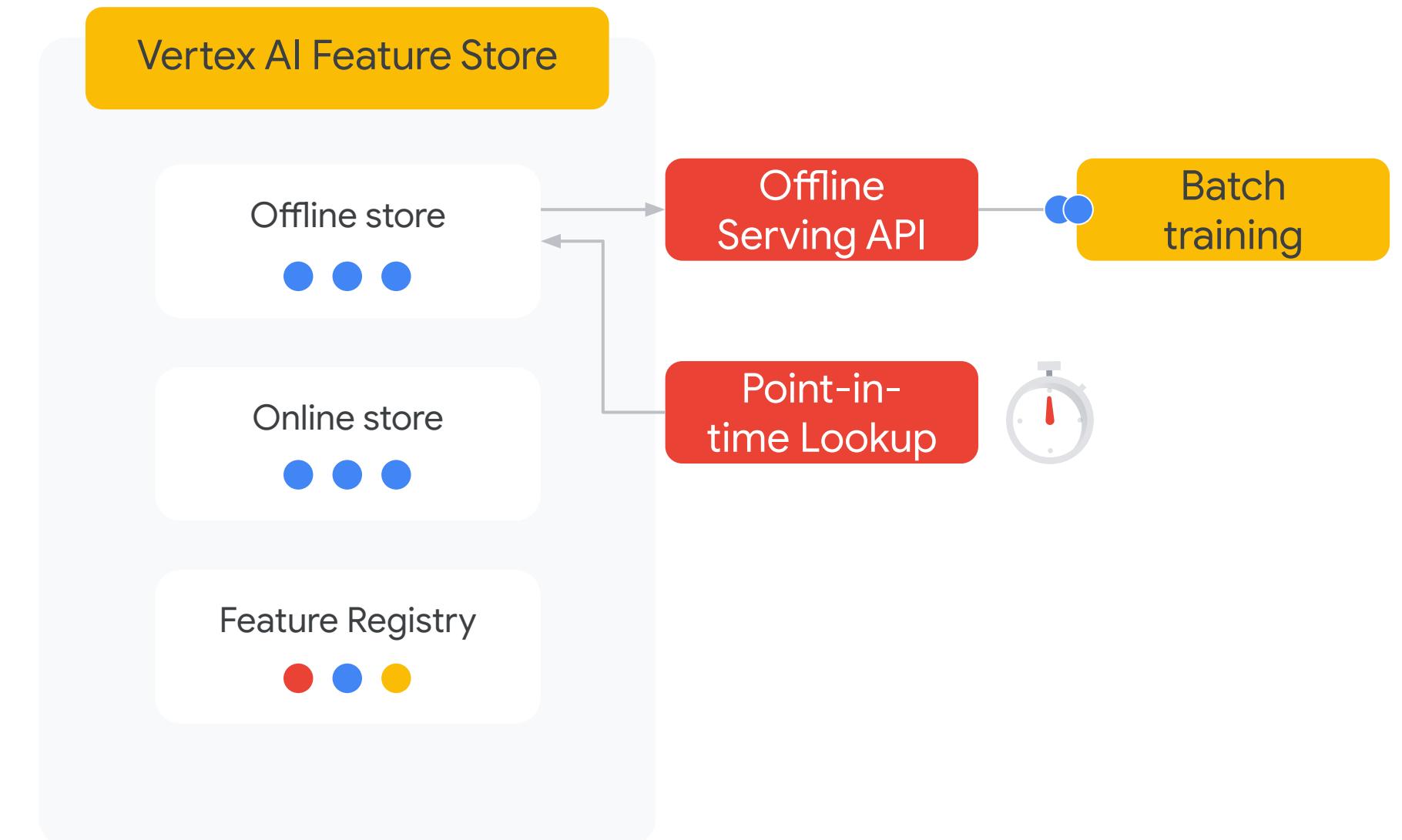
**So, other researchers  
and ML engineers can  
easily find the feature  
with Discovery API...**



**...and retrieve the  
feature value for fast  
online serving with  
continuous Feature  
Monitoring**



Also, retrieve  
feature batches  
for training jobs,  
with  
point-in-time  
lookup to prevent  
data leaks



# Feature Store

## terminology and concepts

# Feature Store

Features    [+ CREATE ENTITY TYPE](#)    [VIEW INGESTION JOBS](#)    [VIEW BATCH SERVING JOBS](#)

Vertex Feature Store enables storing, sharing and serving machine learning features at scale. The feature values can be fetched for training, as well as served with low latency for online prediction. [Learn more](#)

Region: us-central1 (Iowa)    [?](#)

**Filter** Enter a property name

Feature	Entity type	Featurestore	Description
budget_id	budget	hello_world	entity id column
newspaper_budget	budget	hello_world	—
radio_budget	budget	hello_world	—
sales_total	budget	hello_world	—
tv_budget	budget	hello_world	—

- Feature Store is a top-level container for features and their values
- Permitted users can add and share their features
- Users can define features and ingest values from various sources

# Entity type

Features    [+ CREATE ENTITY TYPE](#)    [VIEW INGESTION JOBS](#)    [VIEW BATCH SERVING JOBS](#)

Vertex Feature Store enables storing, sharing and serving machine learning features at scale. The feature values can be fetched for training, as well as served with low latency for online prediction. [Learn more](#)

Region: us-central1 (Iowa) ▾ ?

Filter Enter a property name

Feature	Entity type	Featurestore	Description
budget_id	budget	hello_world	entity id column
newspaper_budget	budget	hello_world	—
radio_budget	budget	hello_world	—
sales_total	budget	hello_world	—
tv_budget	budget	hello_world	—

- An *entity type* is a collection of semantically related features
- You define your own entity types based on the concepts that are relevant to your use case

# Entity

Features    [+ CREATE ENTITY TYPE](#)    [VIEW INGESTION JOBS](#)    [VIEW BATCH SERVING JOBS](#)

Vertex Feature Store enables storing, sharing and serving machine learning features at scale. The feature values can be fetched for training, as well as served with low latency for online prediction. [Learn more](#)

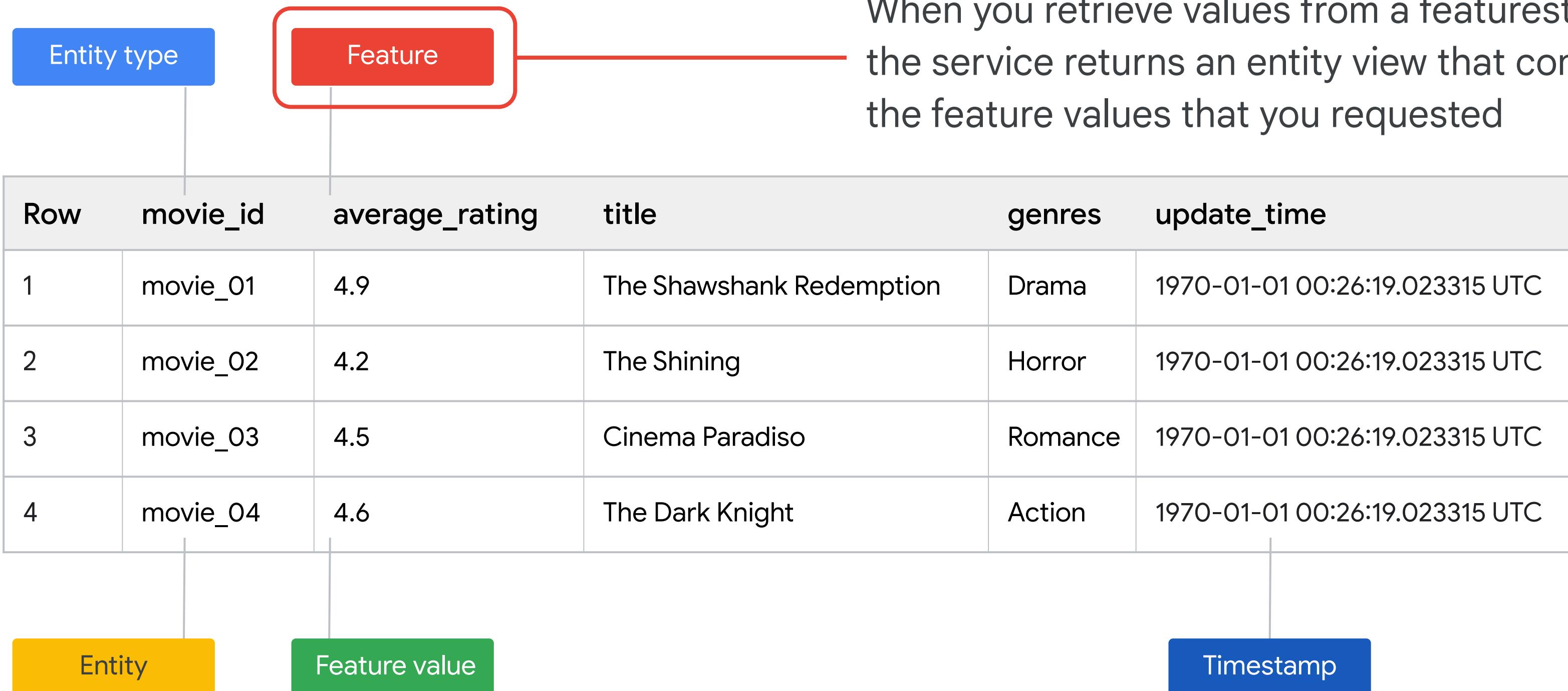
Region: us-central1 (Iowa)    [?](#)

**Filter** Enter a property name

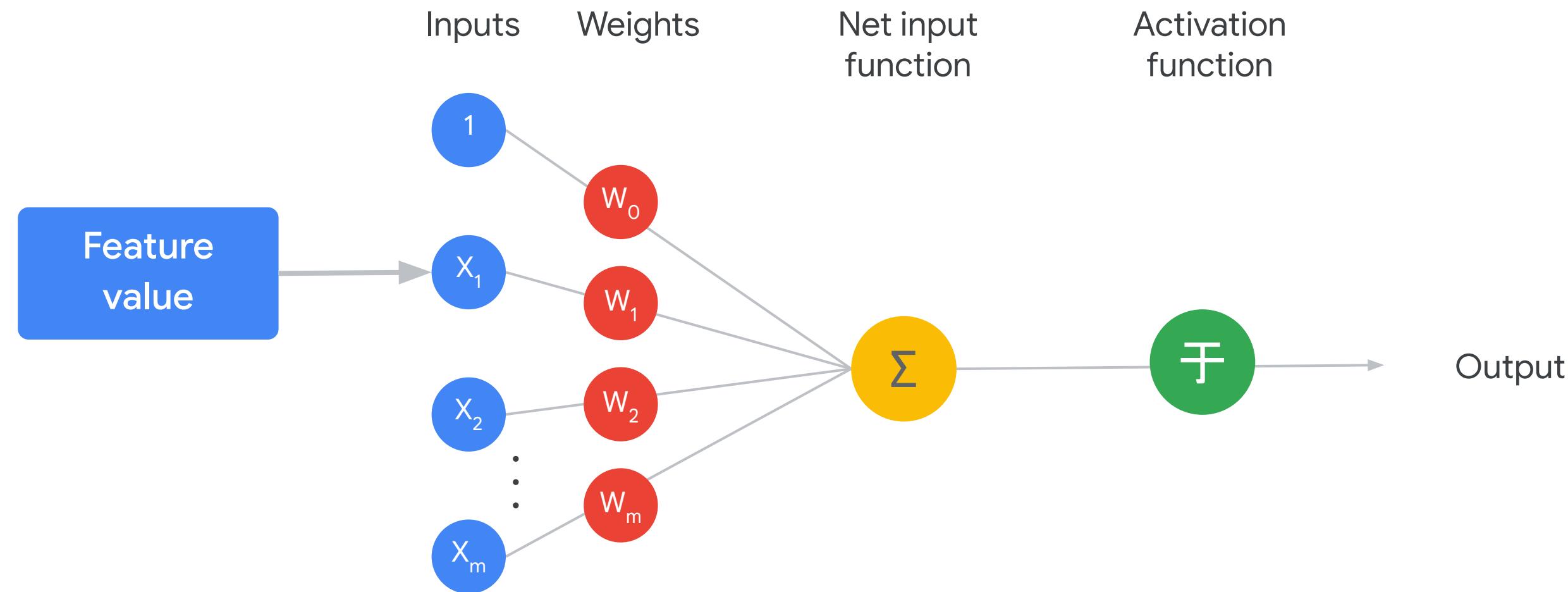
Feature	Entity type	Featurestore	Description
budget_id	budget	hello_world	entity id column
newspaper_budget	budget	hello_world	—
radio_budget	budget	hello_world	—
sales_total	budget	hello_world	—
tv_budget	budget	hello_world	—

- An **entity** is an instance of an entity type
- *Movie\_01* and *movie\_02* are entities of the *movies* entity type
- Each entity must have a unique ID and must be of type STRING

# Entity view

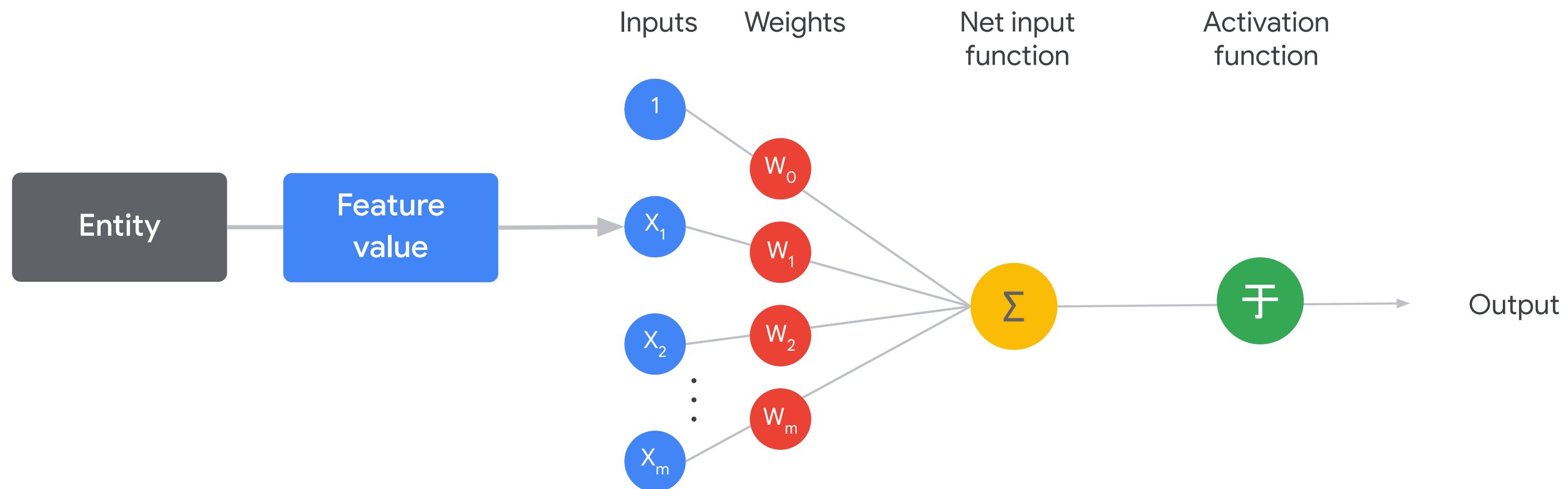


# What is a feature?



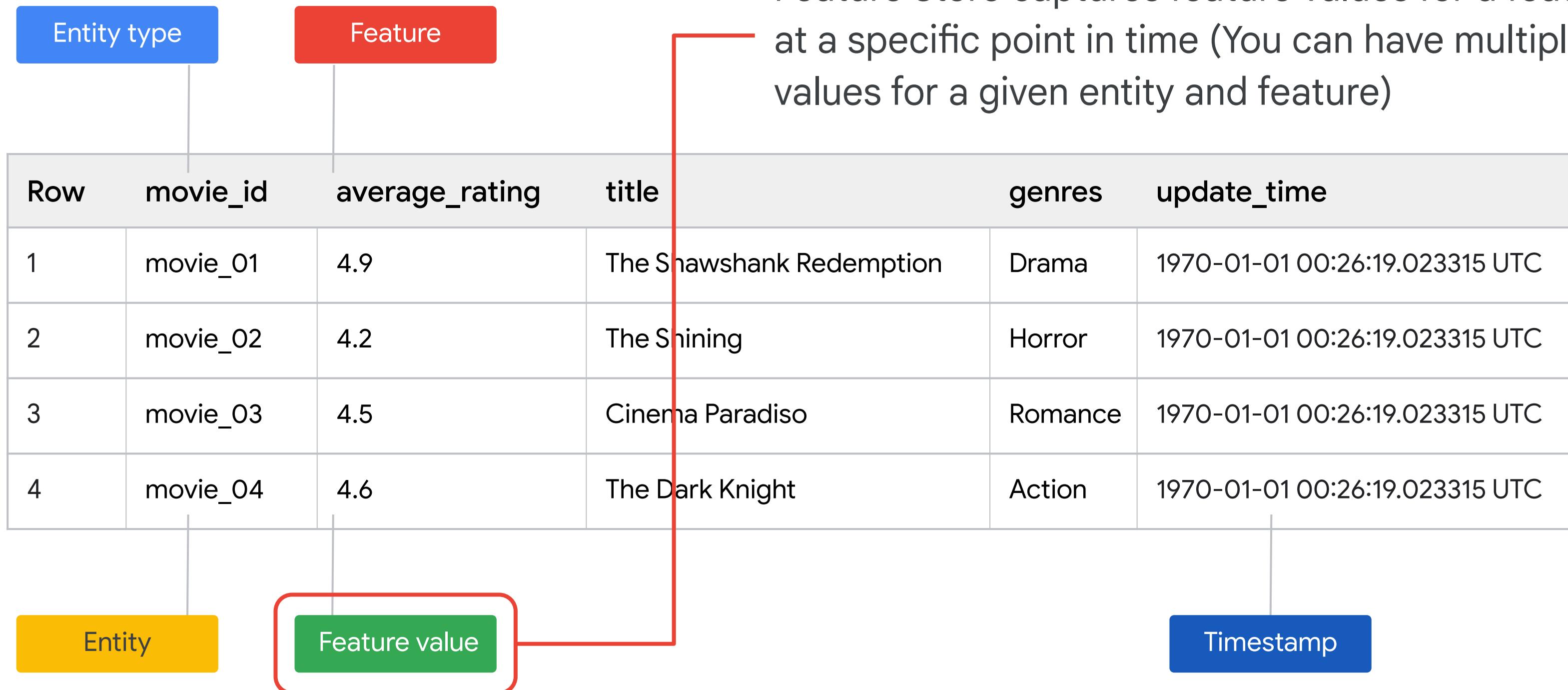
A measurable or recordable property of an entity,  
which is passed as input to a machine learning model

# A feature describes some entity



Examples: Age of user, price of product, category of web page

# Feature value

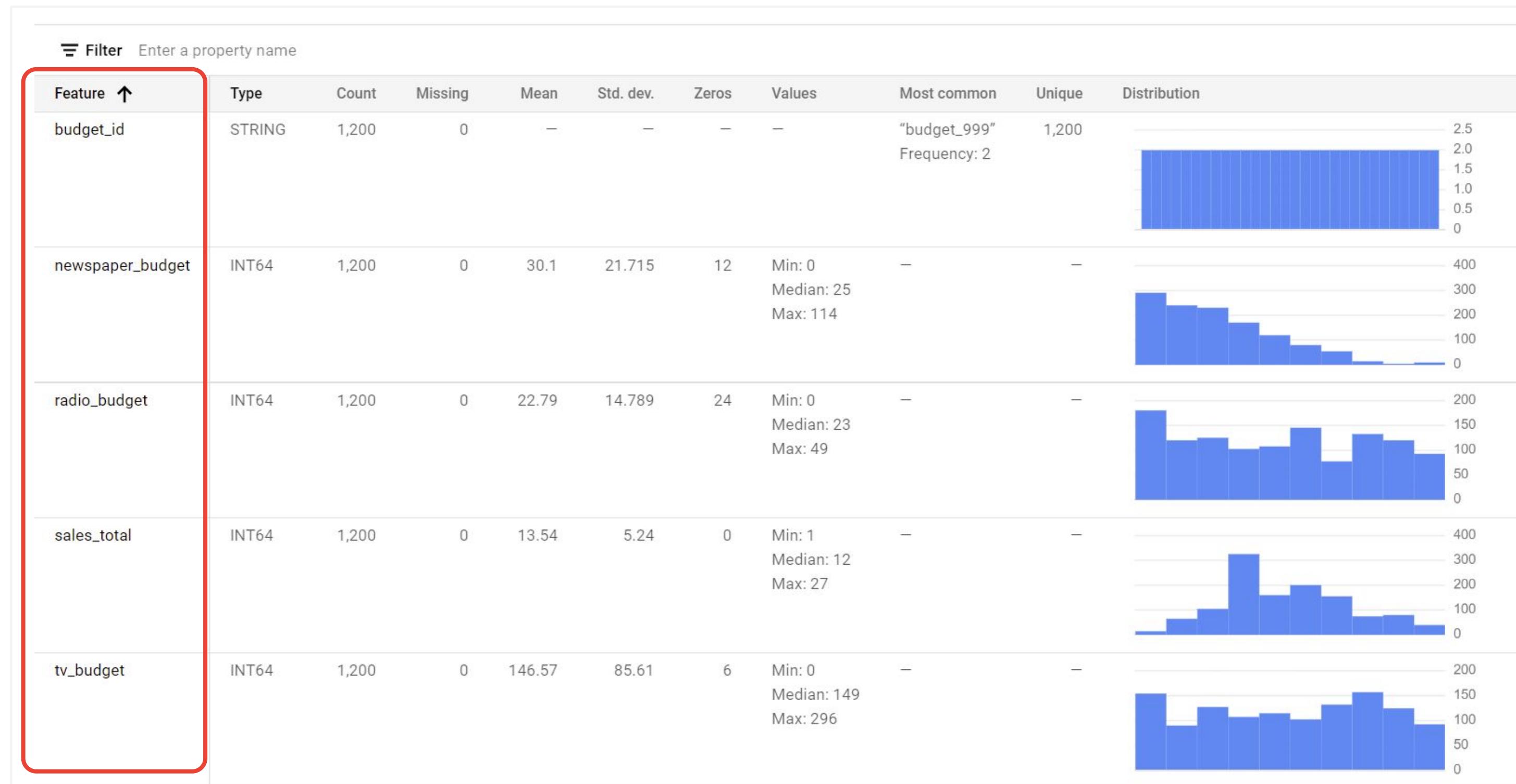


# Timestamp

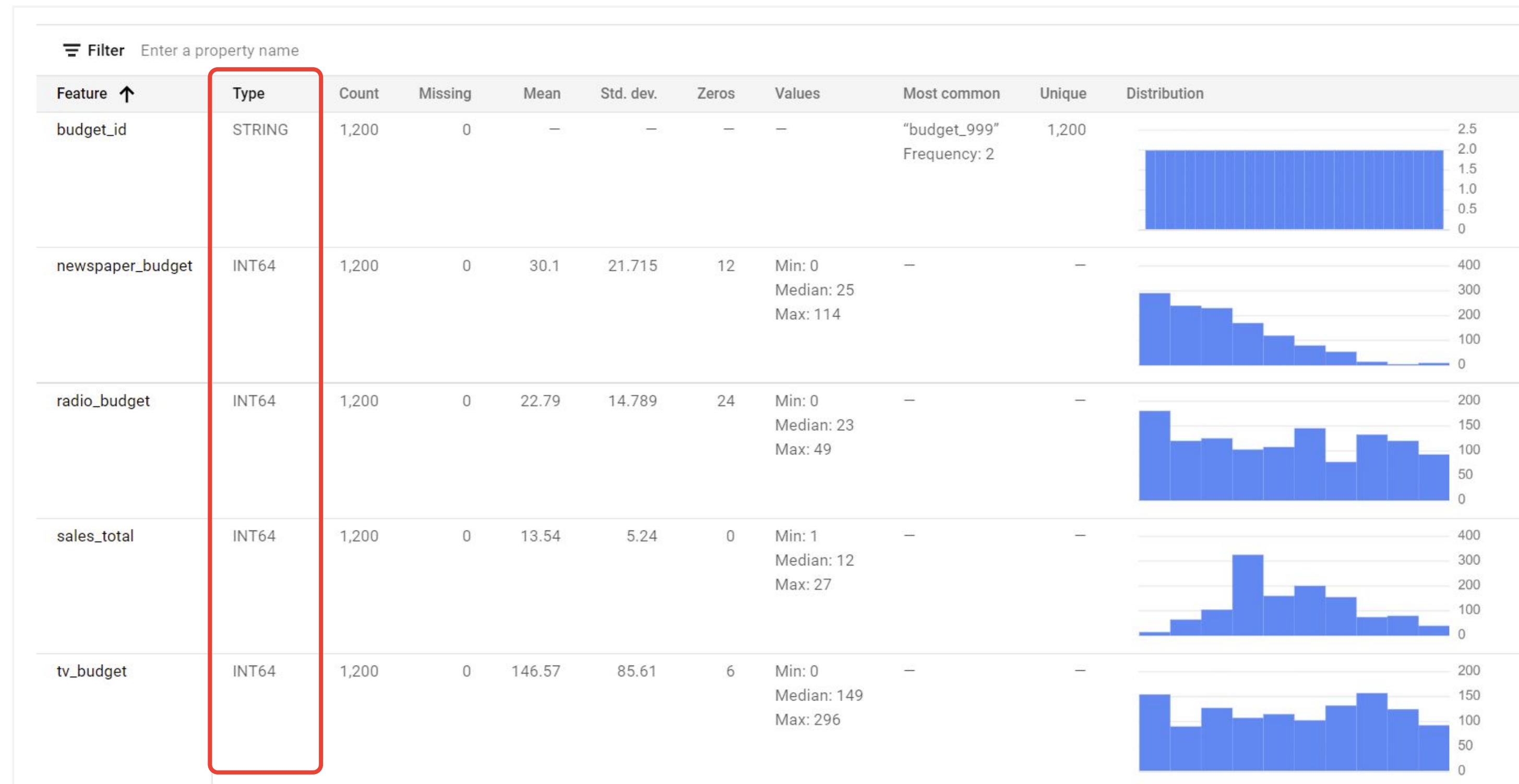
The diagram illustrates the structure of a movie dataset. It features two rows of colored boxes at the top: a blue box labeled "Entity type" and a red box labeled "Feature". A vertical line connects these boxes to a horizontal line above a table. The table has six columns: Row, movie\_id, average\_rating, title, genres, and update\_time. Below the table, three more colored boxes are aligned horizontally: a yellow box labeled "Entity", a green box labeled "Feature value", and a blue box labeled "Timestamp". A red bracket on the right side of the table groups the "Timestamp" box and the last column of the table ("update\_time").

Row	movie_id	average_rating	title	genres	update_time
1	movie_01	4.9	The Shawshank Redemption	Drama	1970-01-01 00:26:19.023315 UTC
2	movie_02	4.2	The Shining	Horror	1970-01-01 00:26:19.023315 UTC
3	movie_03	4.5	Cinema Paradiso	Romance	1970-01-01 00:26:19.023315 UTC
4	movie_04	4.6	The Dark Knight	Action	1970-01-01 00:26:19.023315 UTC

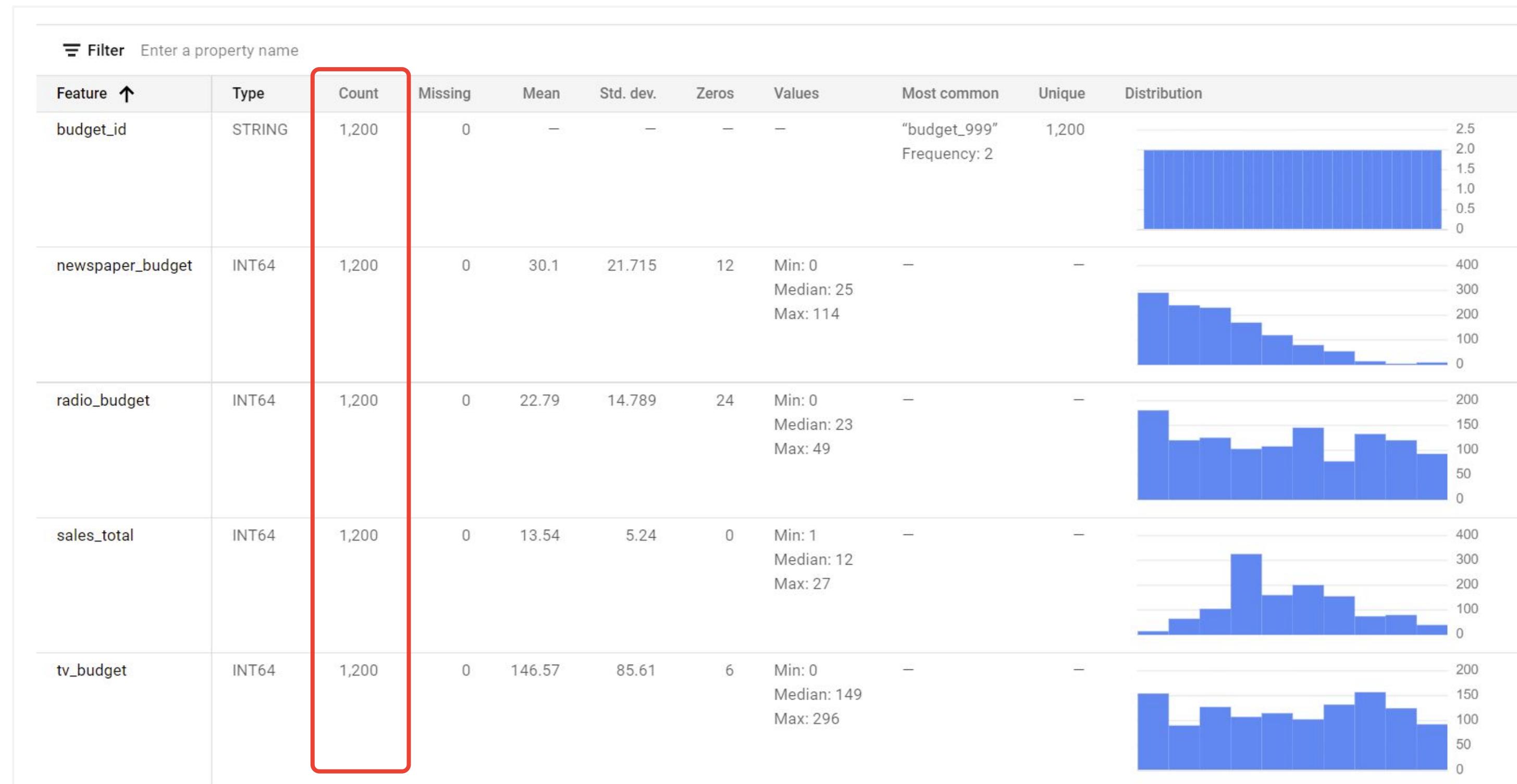
# Feature list



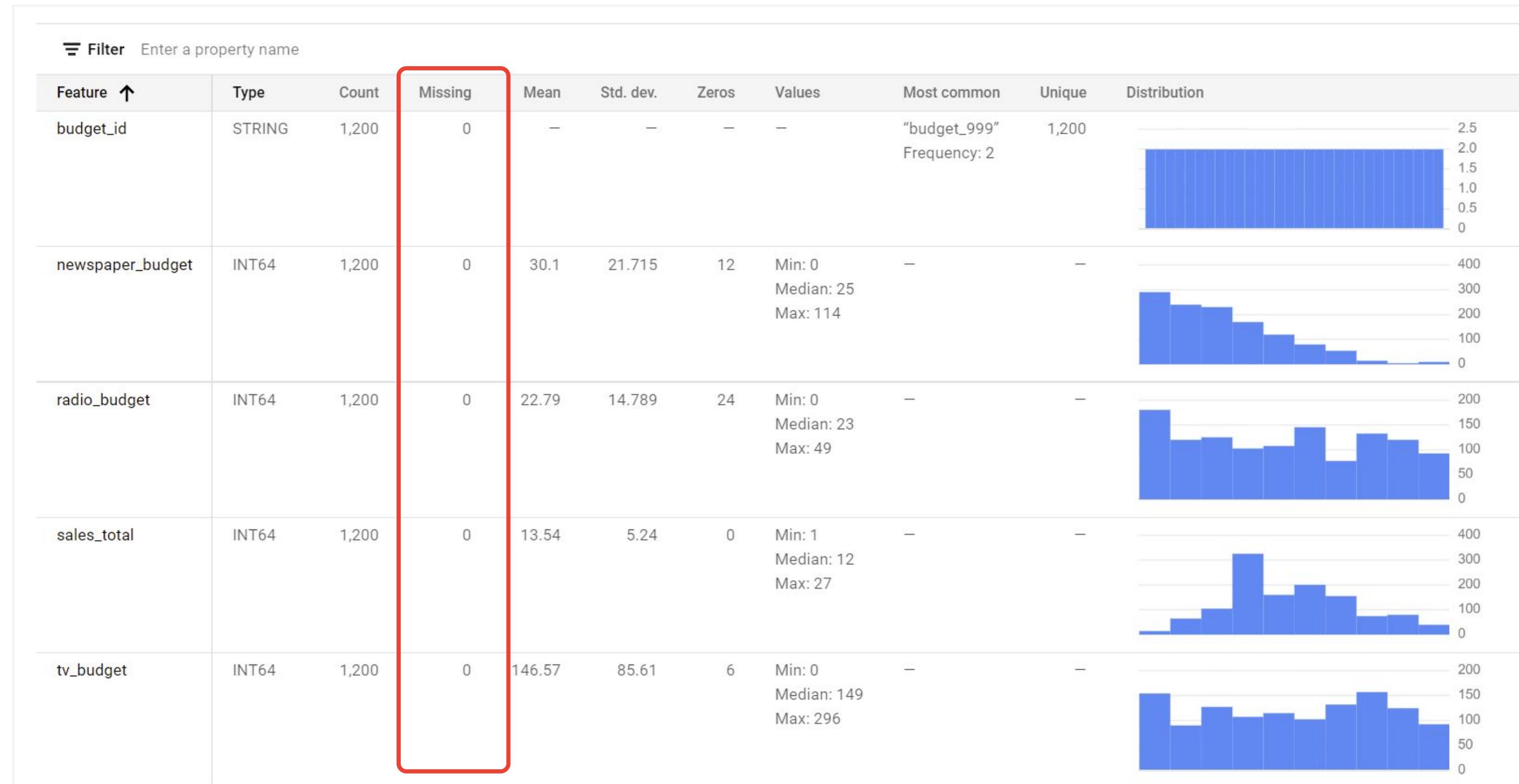
# Feature datatype



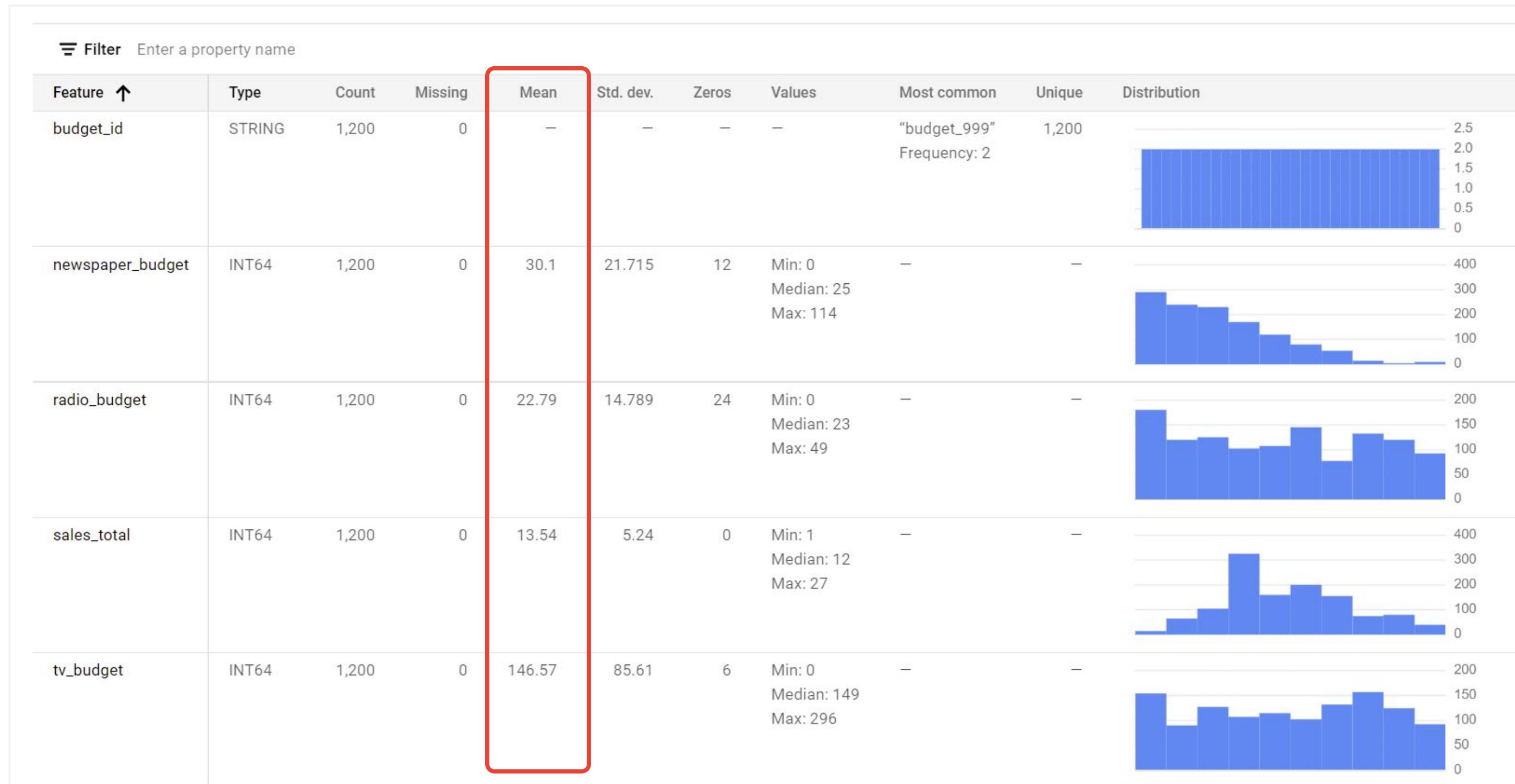
# Feature count



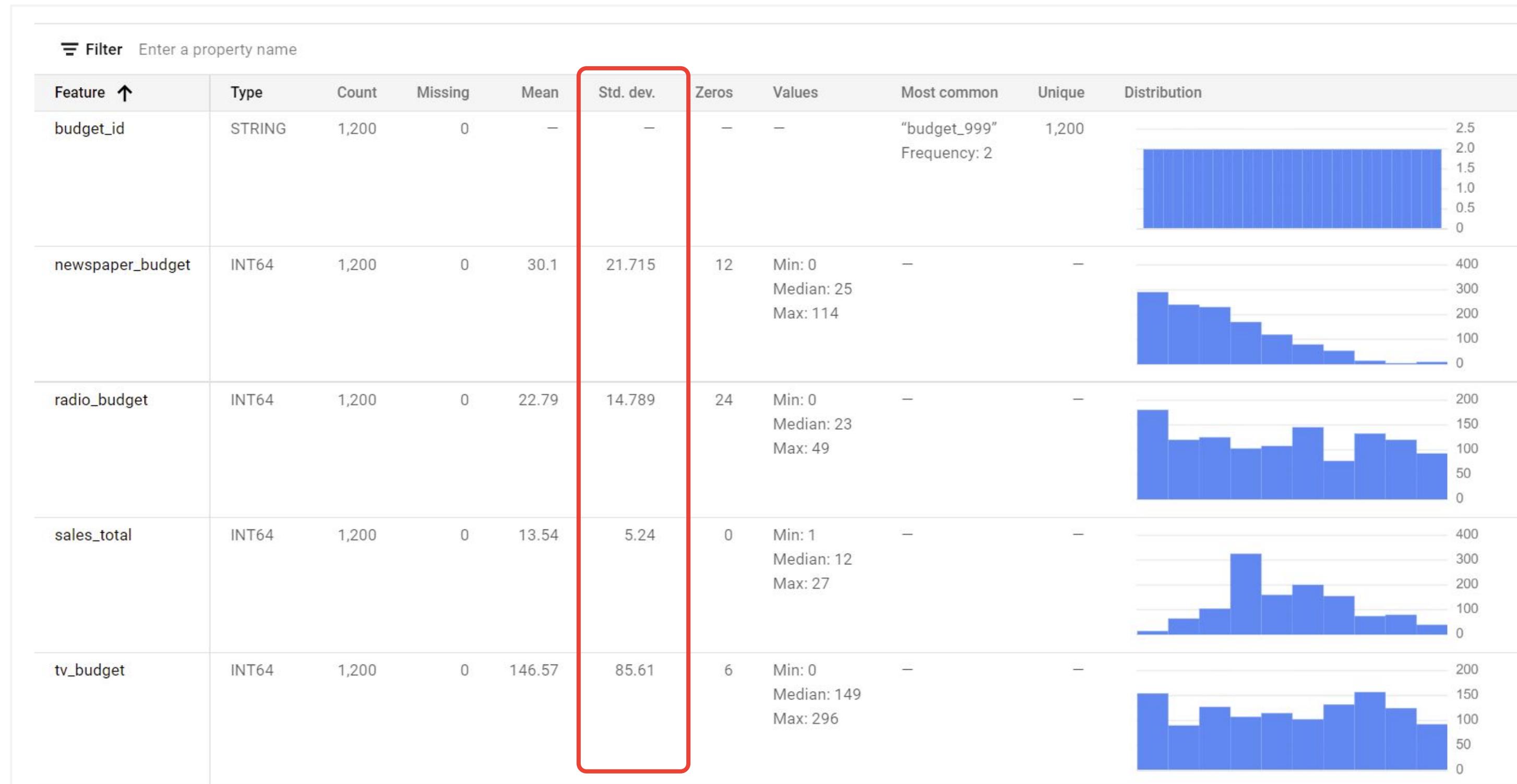
# Missing feature values



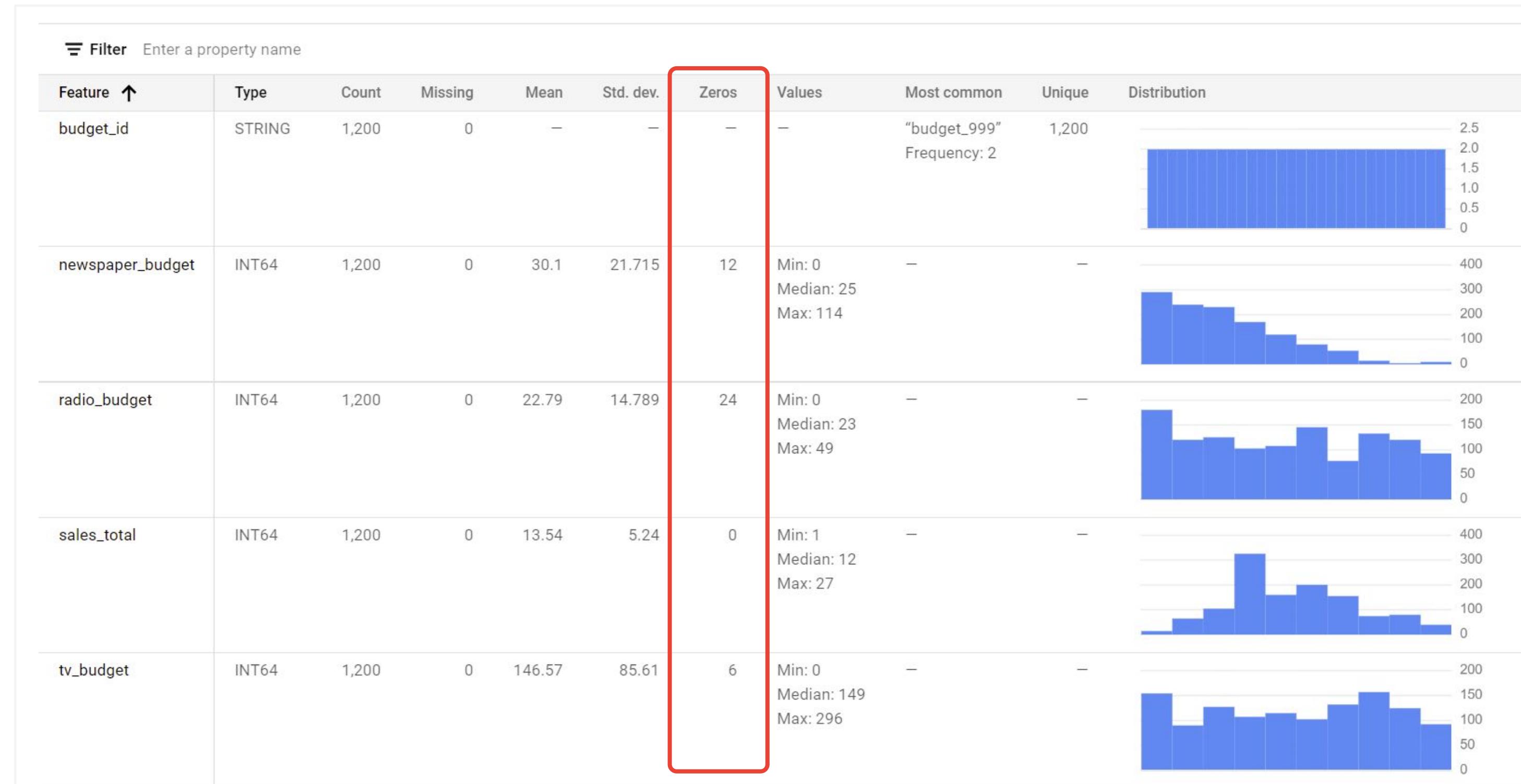
# Mean



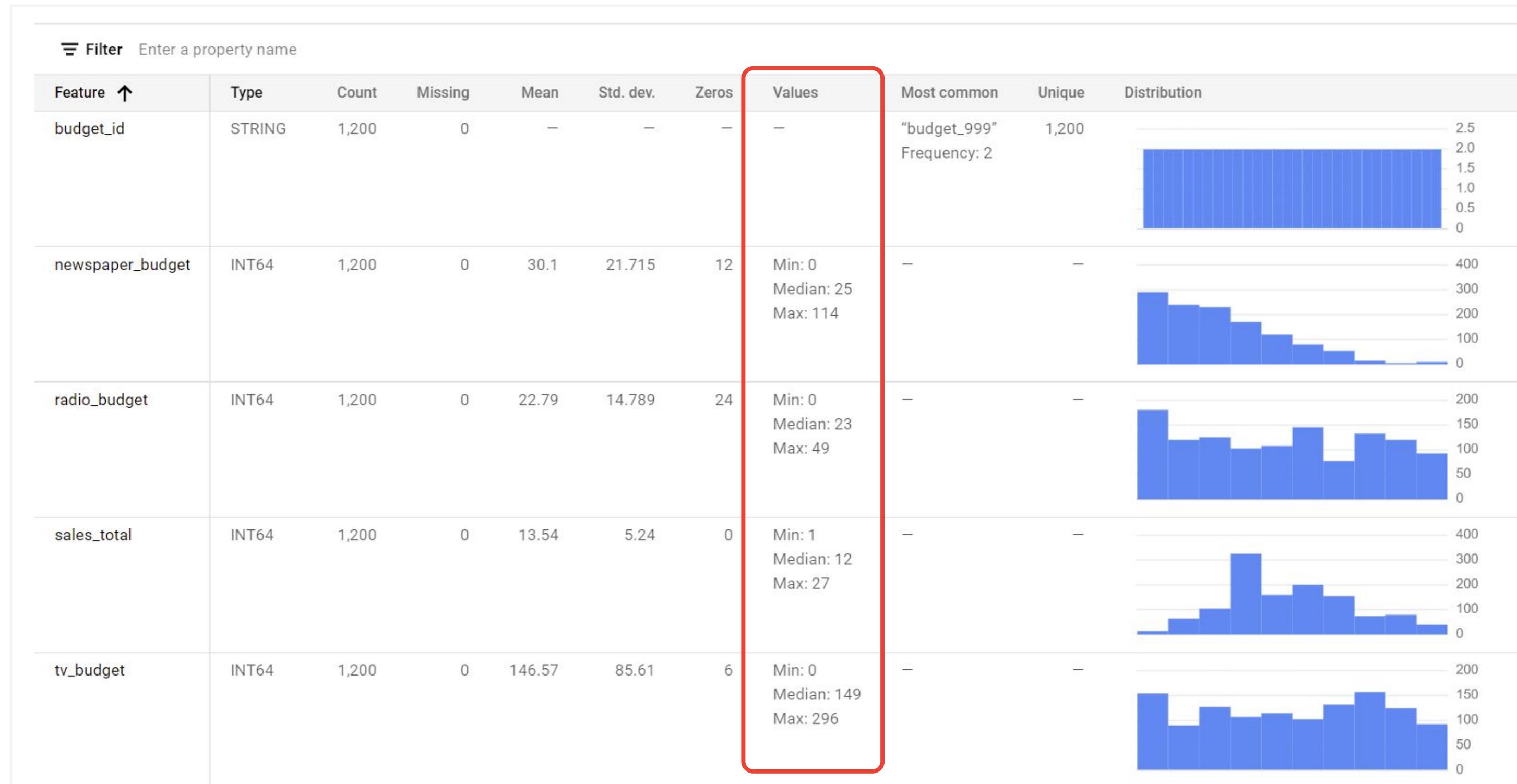
# Standard deviation



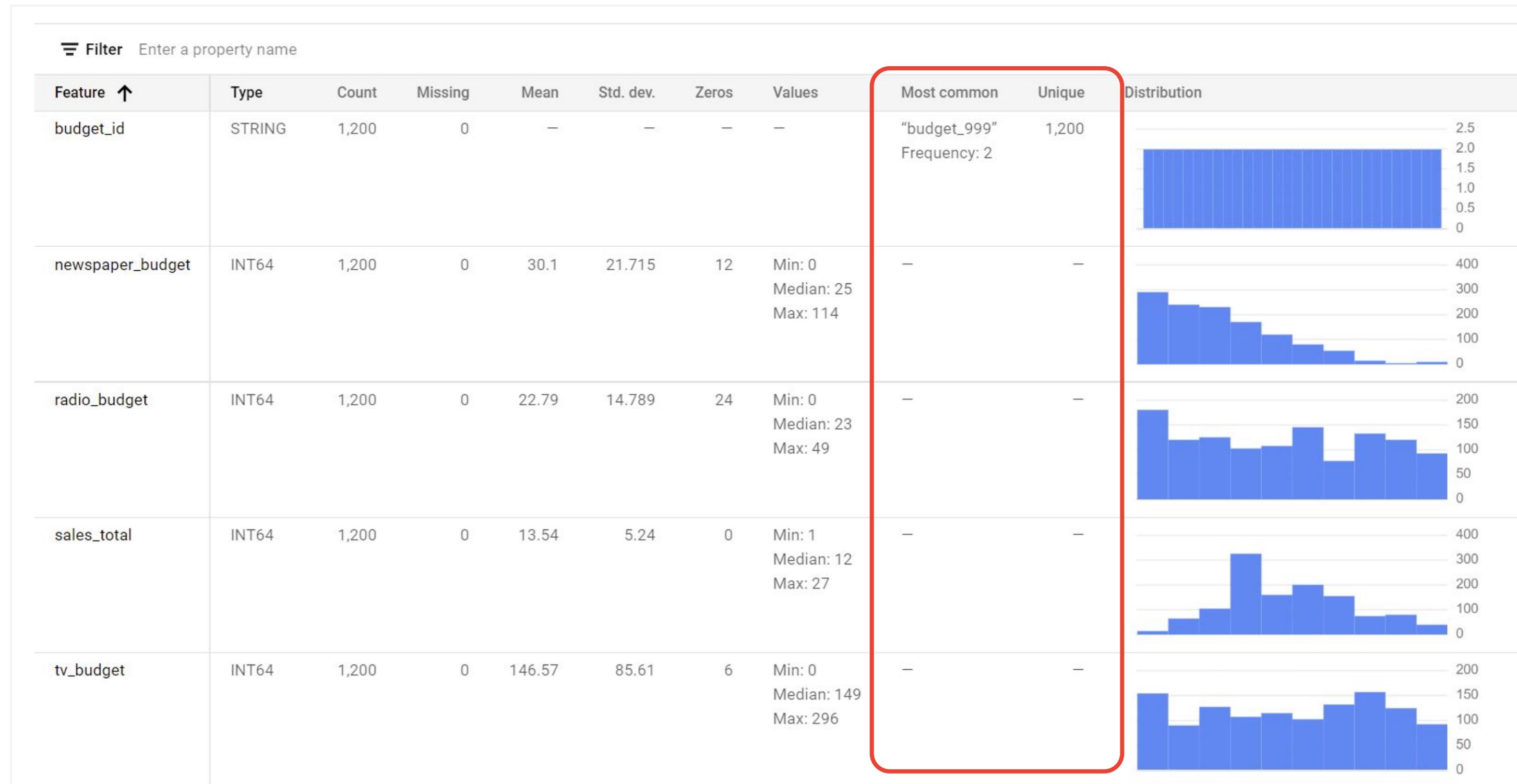
# Number of zeroes



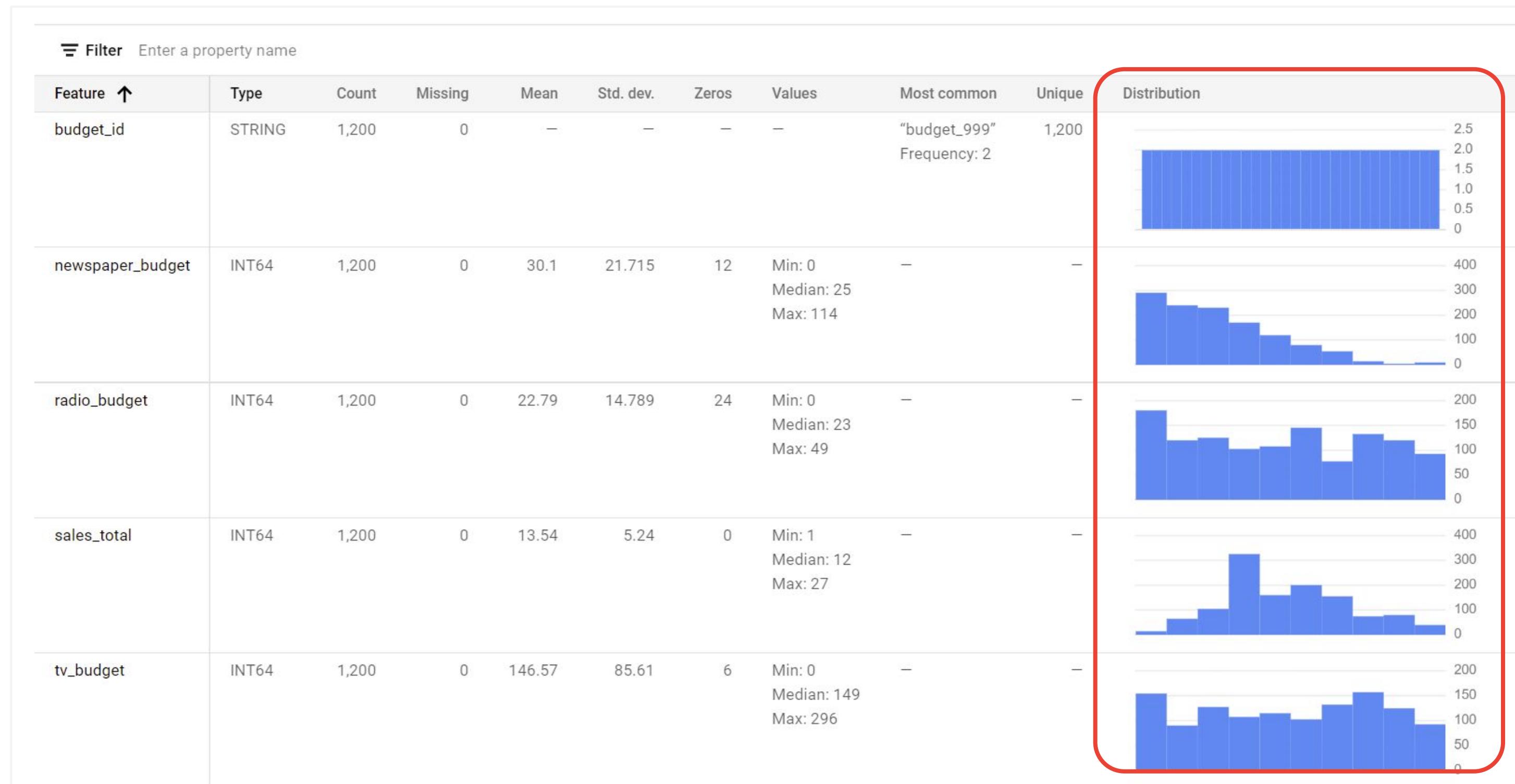
# Aggregation values



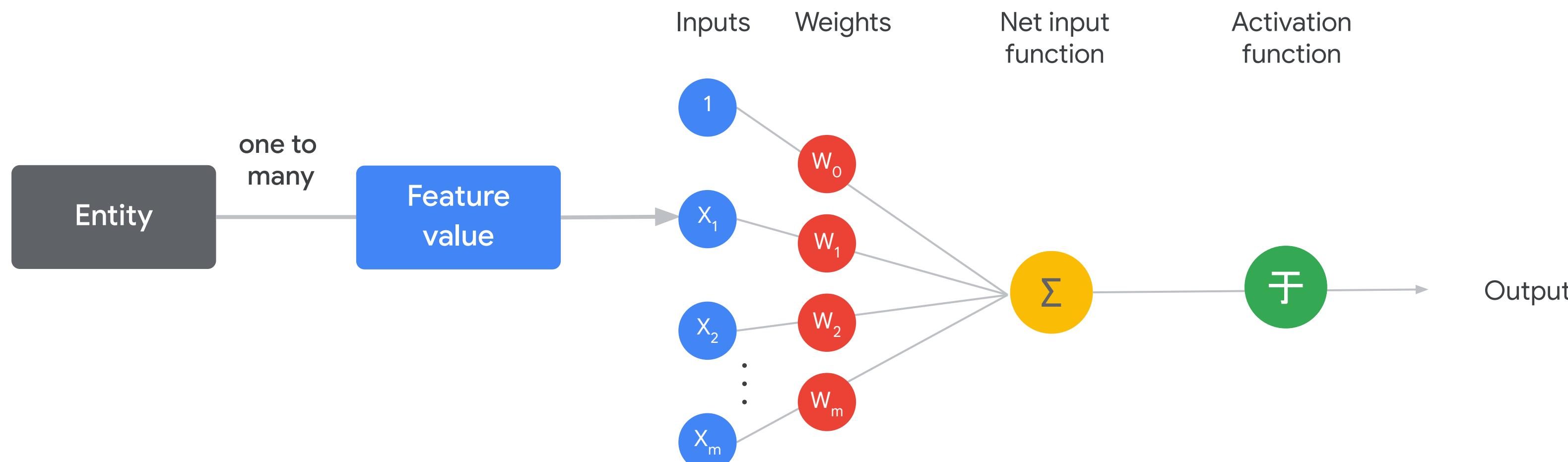
# Entity\_id common and unique



# Feature distribution



# A one-to-many relationship between an entity and a feature



# In Vertex AI Feature Store

Features	 CREATE ENTITY TYPE	 VIEW INGESTION JOBS	 VIEW BATCH SERVING JOBS	
Vertex Feature Store enables storing, sharing and serving machine learning features at scale. The feature values can be fetched for training, as well as served with low latency for online prediction. <a href="#">Learn more</a>				
Region	 us-central1 (Iowa)  			
 Filter Enter a property name				
Feature	Entity type	Featurestore	Description	Last updated
budget_id	budget	hello_world	entity id column	Oct 8, 2021, 3:08:06 PM
newspaper_budget	budget	hello_world	—	Oct 8, 2021, 3:08:06 PM
radio_budget	budget	hello_world	—	Oct 8, 2021, 3:08:06 PM
sales_total	budget	hello_world	—	Oct 8, 2021, 3:08:06 PM
tv_budget	budget	hello_world	—	Oct 8, 2021, 3:08:06 PM

One entity type  
(budget)

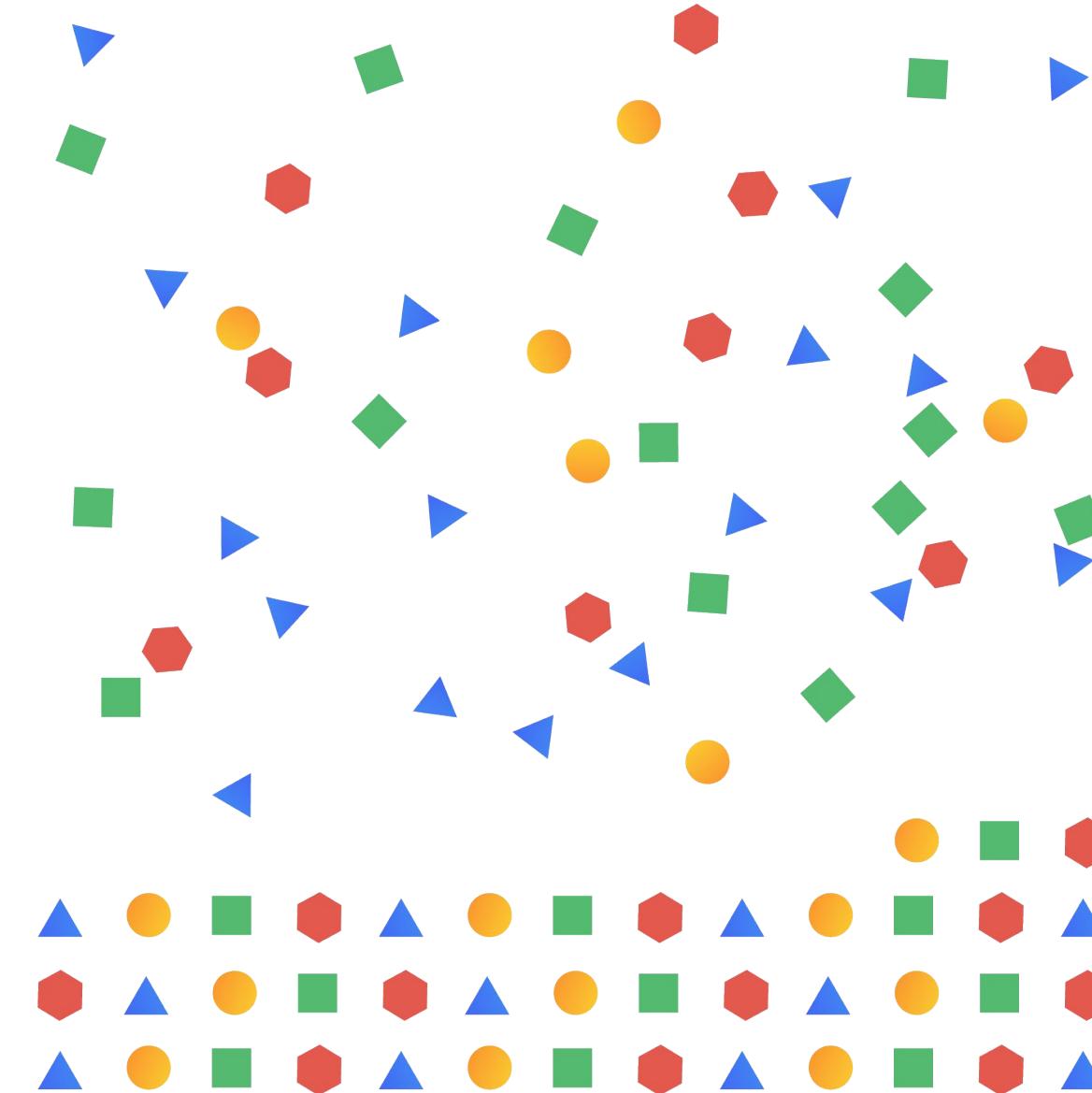
# In Vertex AI Feature Store

Features	 CREATE ENTITY TYPE	 VIEW INGESTION JOBS	 VIEW BATCH SERVING JOBS	
Vertex Feature Store enables storing, sharing and serving machine learning features at scale. The feature values can be fetched for training, as well as served with low latency for online prediction. <a href="#">Learn more</a>				
Region	us-central1 (Iowa)			
 Filter Enter a property name				
Feature	Entity type	Featurestore	Description	Last updated
budget_id	budget	hello_world	entity id column	Oct 8, 2021, 3:08:06 PM
newspaper_budget	budget	hello_world	—	Oct 8, 2021, 3:08:06 PM
radio_budget	budget	hello_world	—	Oct 8, 2021, 3:08:06 PM
sales_total	budget	hello_world	—	Oct 8, 2021, 3:08:06 PM
tv_budget	budget	hello_world	—	Oct 8, 2021, 3:08:06 PM

Many values

# Feature ingestion

- *Feature ingestion* is the process of importing feature values computed by feature engineering jobs into a featurestore
- Entity type and features must be defined in the featurestore
- Batch ingestion means you can do a bulk ingestion of values into a featurestore



# Feature serving

- *Feature serving* is the process of exporting stored feature values for training or inference
- Feature Store offers two methods for serving features:
  - *Batch serving* is for high throughput and serving large volumes of data for offline processing
  - *Online serving* is for low-latency data retrieval of small batches of data for real-time processing (like for online predictions)



# Feature Store **data model**

# Feature Store data model

Row	movie_id	average_rating	title	genres	update_time
1	movie_01	4.9	The Shawshank Redemption	Drama	1970-01-01 00:26:19.023315 UTC
2	movie_02	4.2	The Shining	Horror	1970-01-01 00:26:19.023315 UTC
3	movie_03	4.5	Cinema Paradiso	Romance	1970-01-01 00:26:19.023315 UTC
4	movie_04	4.6	The Dark Knight	Action	1970-01-01 00:26:19.023315 UTC

# Feature Store **creation**

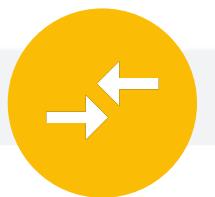
# Create a featurestore to address the key feature management challenges



Hard to share  
and reuse



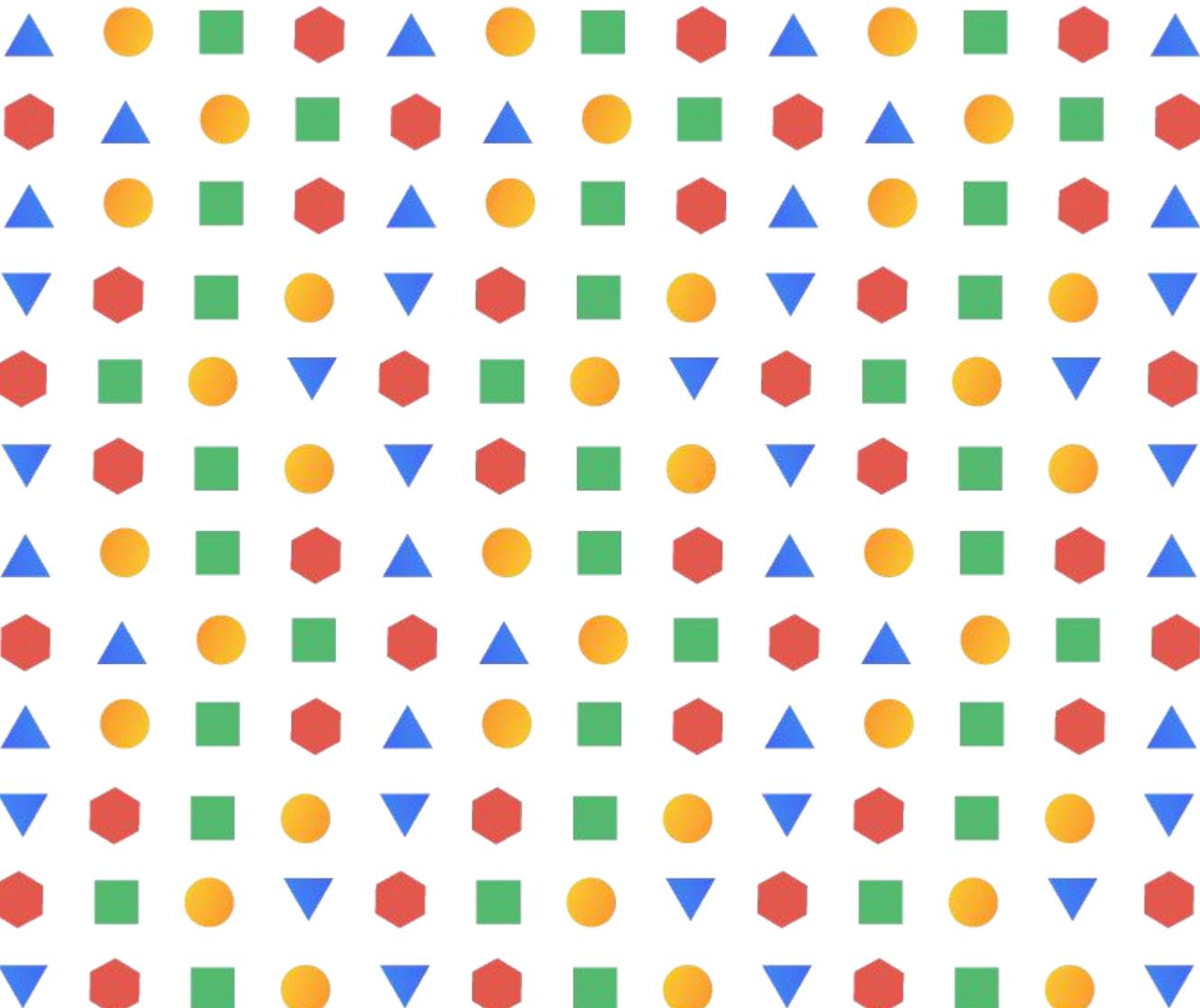
Hard to reliably serve in  
production with low  
latency



Inadvertent skew in  
feature values between  
training and serving

# Prerequisites

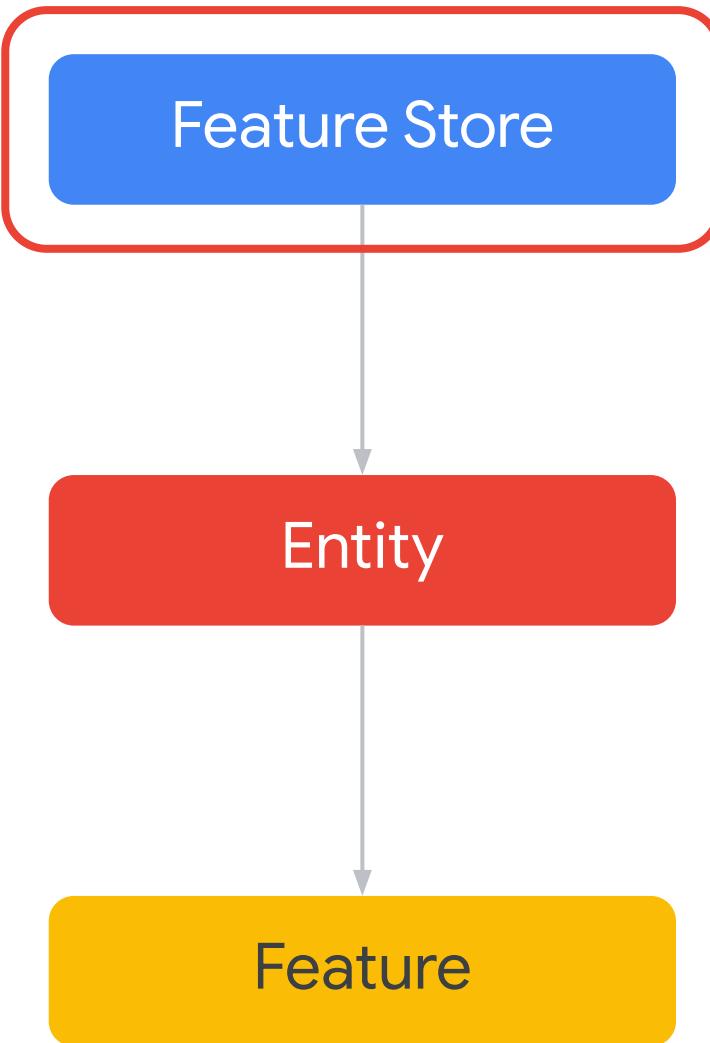
Data is pre-processed and feature values are clean and tidy: no missing values, correct data type, one-hot encoding of categorical values already done



# Source data requirements

- Include a column for entity IDs, and the values must be of type STRING
- Your source data value types must match the value types of the destination feature in the featurestore
- All columns must have a header that is of type STRING. There are no restrictions on the names of the headers. The column header depends on your file type:
  - BigQuery: Column name
  - Avro: Defined by the Avro schema that is associated with the binary data
  - CSV files: First row
- If you provide a column for feature generation timestamps, use right format for your file type:
  - BigQuery tables: TIMESTAMP column
  - Avro: Type long and logical type timestamp-micros
  - CSV files: RFC 3339 format
- CSV files cannot include array data types; use Avro or BigQuery instead
- For array types, you cannot include a null value in the array, although you can include an empty array

## Step 1. Create a featurestore



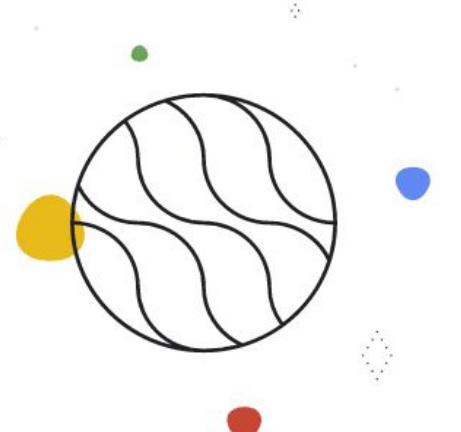
 Features  CREATE ENTITY TYPE  VIEW INGESTION JOBS  VIEW BATCH SERVING JOBS

Vertex Feature Store enables storing, sharing and serving machine learning features at scale. The feature values can be fetched for training, as well as served with low latency for online prediction. [Learn more](#)

Region  
us-central1 (Iowa)  

 Filter Enter a property name

Feature	Entity type	Featurestore	Description	Last updated



You don't have any features in this region yet  
Get started by creating a featurestore. [Learn more](#)

 **Features**    [+ CREATE ENTITY TYPE](#)    [VIEW INGESTION JOBS](#)    [VIEW BATCH SERVING JOBS](#)

 Vertex Feature Store enables storing, sharing and serving machine learning features at scale. The feature values can be fetched for training, as well as served with low latency for online prediction. [Learn more](#)

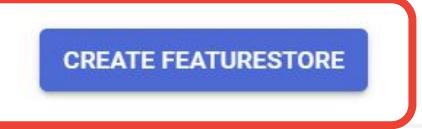
 **Region**  
us-central1 (Iowa)  

 **Filter** Enter a property name

Feature	Entity type	Featurestore	Description	Last updated



You don't have any features in this region yet  
Get started by creating a featurestore. [Learn more](#)

 [CREATE FEATURESTORE](#)

## Create featurestore

Name \*

hello\_world



The name of your featurestore

Region

us-central1 (Iowa)



The region containing your featurestore

Number of online serving nodes \*

1

The number of nodes to allocate for your featurestore

### Encryption

Use a customer-managed encryption key (CMEK)

▲ SHOW LESS

CREATE

CANCEL

## Create featurestore

Name \*

hello\_world



The name of your featurestore

Region

us-central1 (Iowa)



The region containing your featurestore

Number of online serving nodes \*

1

The number of nodes to allocate for your featurestore

### Encryption

Use a customer-managed encryption key (CMEK)

▲ SHOW LESS

**CREATE**

CANCEL

## Create featurestore

Name \*

hello\_world



The name of your featurestore

Region

us-central1 (Iowa)



The region containing your featurestore

Number of online serving nodes \*

1

The number of nodes to allocate for your featurestore

### Encryption

Use a customer-managed encryption key (CMEK)

▲ SHOW LESS

CREATE

CANCEL

## Create featurestore

Name \*

hello\_world



The name of your featurestore

Region

us-central1 (Iowa)



The region containing your featurestore

Number of online serving nodes \*

1

The number of nodes to allocate for your featurestore

### Encryption

Use a customer-managed encryption key (CMEK)

▲ SHOW LESS

CREATE

CANCEL

 **Features**  **CREATE ENTITY TYPE**  **VIEW INGESTION JOBS**  **VIEW BATCH SERVING JOBS**

 Vertex Feature Store enables storing, sharing and serving machine learning features at scale. The feature values can be fetched for training, as well as served with low latency for online prediction. [Learn more](#)

 **Region**   



 **Filter** Enter a property name

Feature	Entity type	Featurestore	Description	Last u
—	—	<a href="#">Featurestore</a> <a href="#">hello_world</a> 	—	—

 [hello\\_world](#) [!\[\]\(629622f261e33faed8d2761f1b8550ed\_img.jpg\) EDIT CONFIGURATION](#)

[!\[\]\(c95c59c27479da8a2ca201bdf4a9df1f\_img.jpg\) PROPERTIES](#) [!\[\]\(559caf7f972ec210192754f0eaed6352\_img.jpg\) ENTITY TYPES](#) [!\[\]\(4d0a23492e31a393313939babccf659f\_img.jpg\) METRICS](#)

**Basic info**

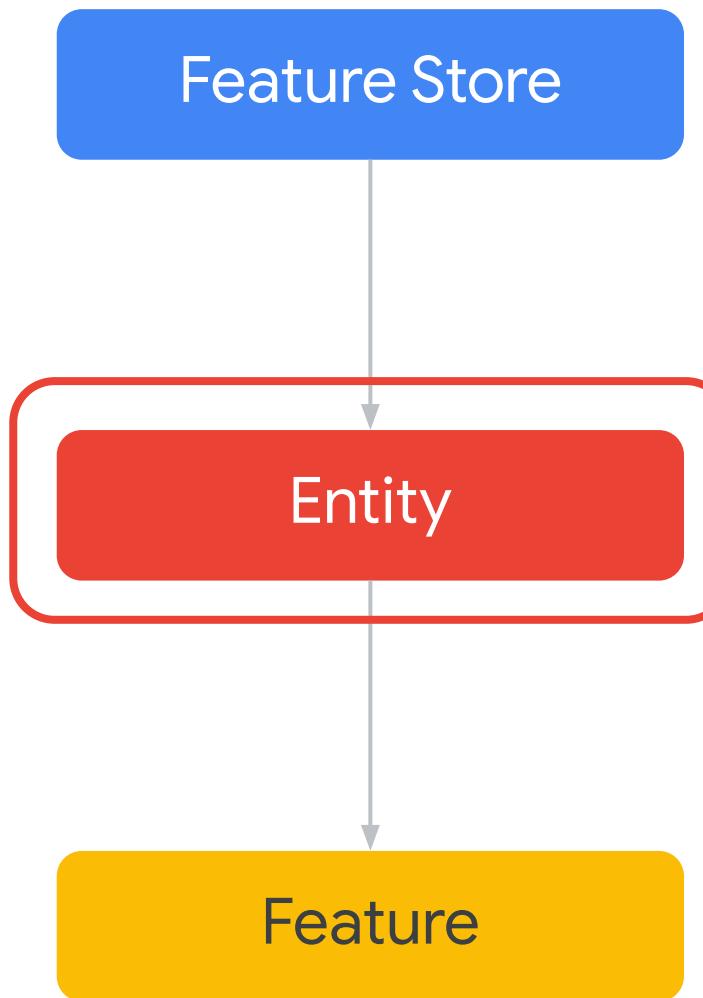
Status	 Active
Created	Oct 6, 2021, 10:38:32 PM
Last updated	Oct 6, 2021, 10:38:32 PM
Region	us-central1
Encryption type	Google-managed key

**Online serving configuration**

Nodes per cluster	1
-------------------	---

## Step 2.

### Create an entity type



Vertex AI

Features

**+ CREATE ENTITY TYPE**

**VIEW INGESTION JOBS**

**VIEW BATCH SERVING JOBS**

---

Dashboard

Datasets

**Features**

Labeling tasks

Workbench

Region  
us-central1 (Iowa) ▾ ?

Filter Enter a property name

 EDIT CONFIGURATION

 PROPERTIES  ENTITY TYPES  METRICS

  
**Entity types**

Entity types group and contain related features. For example, a “movies” entity type might contain features like “title” and “genre”. [Learn more](#)

 **CREATE ENTITY TYPE**

 Filter Enter a property name

Entity type ↑	Description
No entity types have been created in this featurestore	

# Create entity type

Entity types group and contain related features. For example, a “movies” entity type might contain features like “title” and “genre”. [Learn more](#)

Region

us-central1 (Iowa) ▾ ?

Featurestore \*

hello\_world

Entity type name \*

budget\_id

Must start with a letter or underscore. Can use letters, numbers, and underscores.

Description

Optional text description of the entity type

Feature monitoring PREVIEW

Feature monitoring PREVIEW

Provides descriptive statistics and distribution shapes. Enables feature monitoring for all features in the entity type. You can also edit feature monitoring at the feature level, which will override this setting.

Disabled

Monitoring time interval

1 days

CREATE CANCEL

# Create entity type

Entity types group and contain related features. For example, a “movies” entity type might contain features like “title” and “genre”. [Learn more](#)

Region

us-central1 (Iowa) ▾ ?

Featurestore \*

hello\_world

Entity type name \*

budget\_id

Must start with a letter or underscore. Can use letters, numbers, and underscores.

Description

Optional text description of the entity type

Feature monitoring PREVIEW

Feature monitoring PREVIEW

Provides descriptive statistics and distribution shapes. Enables feature monitoring for all features in the entity type. You can also edit feature monitoring at the feature level, which will override this setting.

Disabled

Monitoring time interval

1 days

CREATE CANCEL

# Create entity type

Entity types group and contain related features. For example, a “movies” entity type might contain features like “title” and “genre”. [Learn more](#)

**Region** us-central1 (Iowa) ▾ ?

**Featurestore \*** hello\_world ▾

**Entity type name \*** budget\_id

Must start with a letter or underscore. Can use letters, numbers, and underscores.

**Description** |

Optional text description of the entity type

**Feature monitoring** PREVIEW

**Feature monitoring** PREVIEW

Provides descriptive statistics and distribution shapes. Enables feature monitoring for all features in the entity type. You can also edit feature monitoring at the feature level, which will override this setting.

Disabled

Monitoring time interval 1 days

**CREATE** CANCEL

# Create entity type

Entity types group and contain related features. For example, a “movies” entity type might contain features like “title” and “genre”. [Learn more](#)

Region

us-central1 (Iowa) ▾ ?

Featurestore \*

hello\_world

Entity type name \*

budget\_id

Must start with a letter or underscore. Can use letters, numbers, and underscores.

Description

Optional text description of the entity type

Feature monitoring PREVIEW

Feature monitoring PREVIEW

Provides descriptive statistics and distribution shapes. Enables feature monitoring for all features in the entity type. You can also edit feature monitoring at the feature level, which will override this setting.

Disabled

Monitoring time interval

1 days

**CREATE**  CANCEL

# Create entity type

Entity types group and contain related features. For example, a “movies” entity type might contain features like “title” and “genre”. [Learn more](#)

**Region** us-central1 (Iowa) ▾ ?

**Featurestore \*** hello\_world ▾

**Entity type name \*** budget\_id  
Must start with a letter or underscore. Can use letters, numbers, and underscores.

**Description**  
Optional text description of the entity type

**Feature monitoring** PREVIEW

**Feature monitoring** PREVIEW

Provides descriptive statistics and distribution shapes. Enables feature monitoring for all features in the entity type. You can also edit feature monitoring at the feature level, which will override this setting.

Disabled

Monitoring time interval 1 days

**CREATE** CANCEL

# Entity type created

Features     CREATE ENTITY TYPE     VIEW INGESTION JOBS     VIEW BATCH SERVING JOBS

Vertex Feature Store enables storing, sharing and serving machine learning features at scale. The feature values can be fetched for training, as well as served with low latency for online prediction. [Learn more](#)

Region  
us-central1 (Iowa) ▾ ?

Filter Enter a property name

Feature	Entity type	Featurestore	Description
-	budget_id	hello_world	-

FEATURES

ENTITY TYPE PROPERTIES

## Basic info

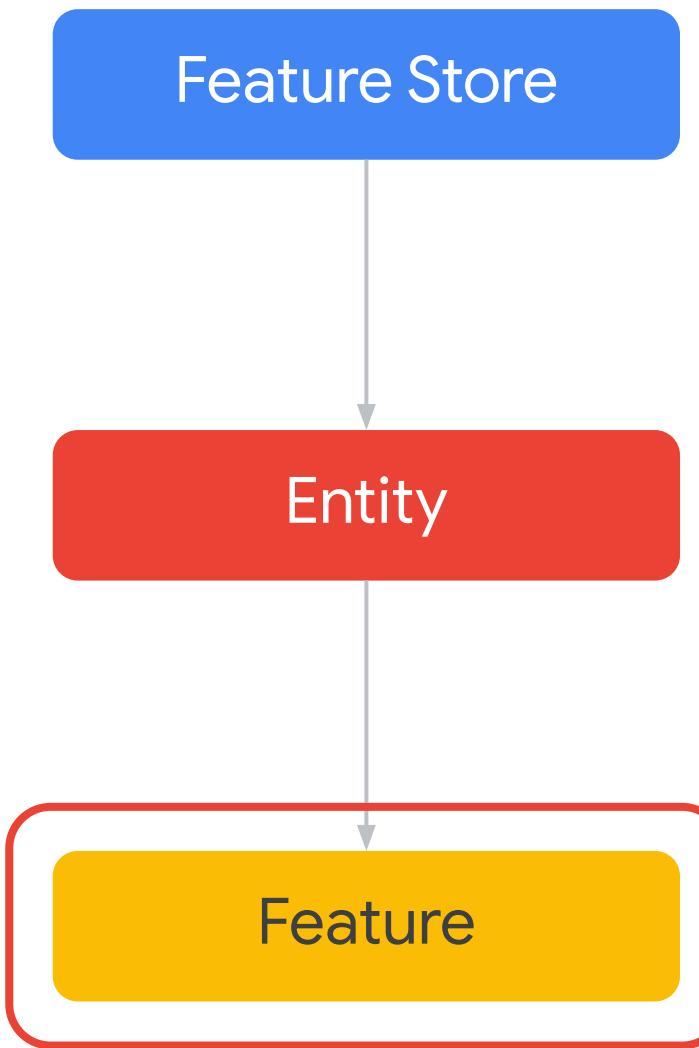
Name	budget
Region	us-central1
Featurestore	<a href="#">hello_world</a>
Created	Oct 8, 2021, 3:00:55 PM
Updated	Oct 8, 2021, 3:00:55 PM
Description	media budgets

## Feature monitoring

[PREVIEW](#)

Status	Disabled
Time interval	—

## Step 3. Add features



 Vertex AI

[← budget](#) [!\[\]\(5d09ddd50fda3c710f49a7d348c426f4\_img.jpg\) INGEST VALUES](#) [!\[\]\(efd91363b3848f264c3cbcbb43054c4a\_img.jpg\) EDIT INFO](#) [!\[\]\(463ec2982e533439a97ac1f6988a24d8\_img.jpg\) DELETE](#)

	FEATURES	ENTITY TYPE PROPERTIES
 Dashboard		
 Datasets		
 Features	<b>FEATURES</b>	ENTITY TYPE PROPERTIES
 Labeling tasks		
 Workbench		

## Features

A feature is a measurable attribute of an entity type. After you add features in your entity type, you can then associate your features with values stored in BigQuery or Cloud Storage. [Learn more](#)

**ADD FEATURES**

## Add features

A feature is a measurable attribute of an entity type. After you add features in your entity type, you can then associate your features with values stored in BigQuery or Cloud Storage. [Learn more](#)

Feature names can include lowercase letters, numbers and underscores and must start with a lowercase letter or underscore

Feature name *	Value type *	Description	Override monitoring values	Feature monitoring	Interval *
Feature name 1 *	Value type 1 *	Description 1	<input type="checkbox"/> Override	<input checked="" type="checkbox"/> Disabled	Interval 1 1 days

[+ ADD ANOTHER FEATURE](#)

[SAVE](#)

[CANCEL](#)

Before setting up  
the featurestore,  
the XYZ team  
added an entity  
type field to the  
budget table

budget_id	tv	radio	newspaper	sales
budget_01	230.1	37.8	69.2	22.1
budget_02	44.5	39.3	45.1	10.4
budget_03	17.2	45.9	69.3	9.3
budget_04	151.5	41.3	58.5	18.5
budget_05	180.8	10.8	58.4	12.9
budget_06	8.7	48.9	75	7.2

## Features

[\*\*CREATE ENTITY TYPE\*\*](#)[\*\*VIEW INGESTION JOBS\*\*](#)[\*\*VIEW BATCH SERVING JOBS\*\*](#)

Vertex Feature Store enables storing, sharing and serving machine learning features at scale. The feature values can be fetched for training, as well as served with low latency for online prediction. [Learn more](#)

**Region**

us-central1 (Iowa)

**Filter** Enter a property name

Feature	Entity type	Featurestore	Description	Last updated	Created
budget_id	budget	hello_world	entity id column	Oct 8, 2021, 3:08:06 PM	Oct 8, 2021, 3:08:06 PM
newspaper_budget	budget	hello_world	—	Oct 8, 2021, 3:08:06 PM	Oct 8, 2021, 3:08:06 PM
radio_budget	budget	hello_world	—	Oct 8, 2021, 3:08:06 PM	Oct 8, 2021, 3:08:06 PM
sales_total	budget	hello_world	—	Oct 8, 2021, 3:08:06 PM	Oct 8, 2021, 3:08:06 PM
tv_budget	budget	hello_world	—	Oct 8, 2021, 3:08:06 PM	Oct 8, 2021, 3:08:06 PM

Vertex AI

[budget\\_id](#)

[EDIT INFO](#) [DELETE](#)

[FEATURE PROPERTIES](#) [METRICS](#) [PREVIEW](#) [INGESTION JOBS](#) [BATCH SERVING JOBS](#)

**Basic info**

Name	budget_id
Region	us-central1
Featurestore	<a href="#">hello_world</a>
Entity type	<a href="#">budget</a>
Type	STRING
Created	Oct 8, 2021, 3:08:06 PM
Updated	Oct 8, 2021, 3:08:06 PM
Description	entity id column
Labels	—

**Entity type monitoring config**

Status	Disabled
Time interval	—

Vertex AI    [← budget](#)    [INGEST VALUES](#)    [EDIT INFO](#)    [DELETE](#)

FEATURES    ENTITY TYPE PROPERTIES

### Basic info

Name	budget
Region	us-central1
Featurestore	<a href="#">hello_world</a>
Created	Oct 8, 2021, 3:00:55 PM
Updated	Oct 8, 2021, 3:00:55 PM
Description	media budgets

### Feature monitoring PREVIEW

Status	Disabled
Time interval	—

### Feature value type distribution

BOOL	0 (0%)	INT64_ARRAY	0 (0%)
BOOL_ARRAY	0 (0%)	STRING	1 (20%)
DOUBLE	0 (0%)	STRING_ARRAY	0 (0%)
DOUBLE_ARRAY	0 (0%)	BYTES	0 (0%)
INT64	4 (80%)		

Vertex AI

← budget ↴ INGEST VALUES EDIT INFO DELETE

FEATURES ENTITY TYPE PROPERTIES

Features

A feature is a measurable attribute of an entity type. After you add features in your entity type, you can then associate your features with values stored in BigQuery or Cloud Storage. [Learn more](#)

The screenshot shows the Vertex AI Features page. On the left, there's a sidebar with icons for Dashboard, Datasets, Features (which is selected and highlighted in blue), and Labeling tasks. The main content area has a breadcrumb navigation bar with 'budget' and 'INGEST VALUES'. Below it are two tabs: 'FEATURES' (selected) and 'ENTITY TYPE PROPERTIES'. The 'FEATURES' tab contains a section titled 'Features' with a descriptive text about what features are and a link to learn more. A red box highlights the 'EDIT INFO' button, which has a pencil icon and a hand cursor icon pointing at it.

# Edit entity type info

Entity types group and contain related features. For example, a “movies” entity type might contain features like “title” and “genre”. [Learn more](#)

Entity type name \*

budget

Must start with a letter or underscore. Can use letters, numbers, and underscores.

Description

media budgets

Optional text description of the entity type

Feature monitoring PREVIEW

Provides descriptive statistics and distribution shapes. Enables feature monitoring for all features in the entity type. You can also edit feature monitoring at the feature level, which will override this setting.

Enabled

Monitoring time interval

1

days

**UPDATE**

CANCEL

FEATURES

ENTITY TYPE PROPERTIES

## Basic info

Name budget

Region us-central1

Featurestore hello\_world

Created Oct 8, 2021, 3:00:55 PM

Updated Oct 17, 2021, 10:01:07 PM

Description media budgets

## Feature monitoring PREVIEW

Status Enabled

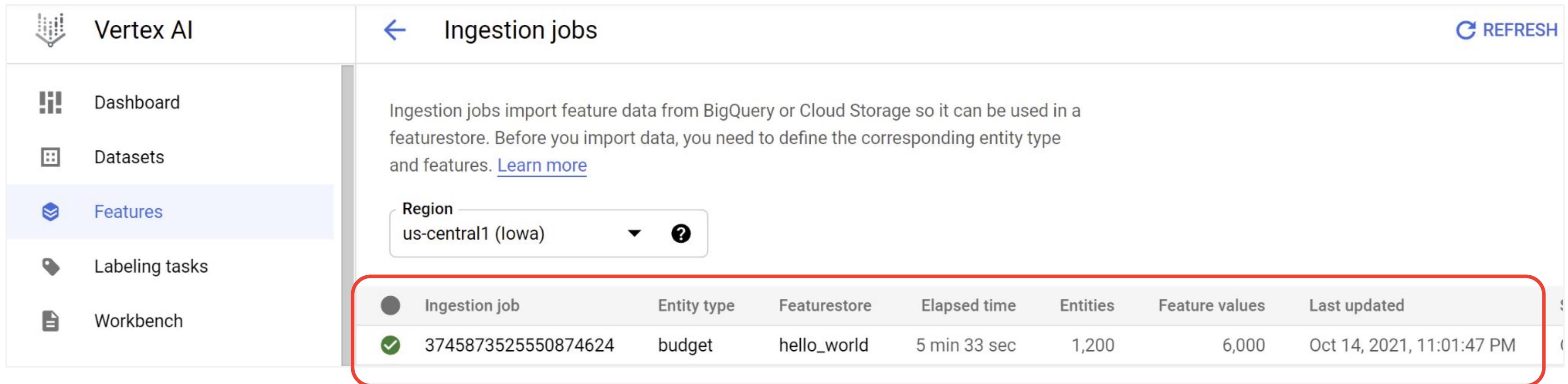
Time interval 1 day

## Feature value type distribution

BOOL	0 (0%)	INT64_ARRAY	0 (0%)
BOOL_ARRAY	0 (0%)	STRING	1 (20%)
DOUBLE	0 (0%)	STRING_ARRAY	0 (0%)
DOUBLE_ARRAY	0 (0%)	BYTES	0 (0%)
INT64	4 (80%)		



# Ingestion jobs



The screenshot shows the Vertex AI web interface. On the left is a sidebar with icons for Dashboard, Datasets, Features (which is selected and highlighted in blue), Labeling tasks, and Workbench. The main area has a header with a back arrow, the title "Ingestion jobs", and a "REFRESH" button. Below the header is a descriptive text about ingestion jobs and a "Region" dropdown set to "us-central1 (Iowa)". A table lists an "Ingestion job" with the ID 3745873525550874624, entity type "budget", featurestore "hello\_world", elapsed time "5 min 33 sec", entities "1,200", feature values "6,000", and last updated "Oct 14, 2021, 11:01:47 PM". This row is highlighted with a red box.

Ingestion job	Entity type	Featurestore	Elapsed time	Entities	Feature values	Last updated
3745873525550874624	budget	hello_world	5 min 33 sec	1,200	6,000	Oct 14, 2021, 11:01:47 PM

## Vertex AI

- Dashboard
- Datasets
- Features**
- Labeling tasks
- Workbench
- Pipelines
- Training
- Experiments
- Marketplace

<

### Ingestion job details

FEATURES PROPERTIES

This ingestion job finished on October 14, 2021 at 11:01:47 PM GMT-7

Status	Finished
Job ID	3745873525550874624
Created	Oct 14, 2021, 10:56:13 PM
Elapsed time	5 min 33 sec
Region	us-central1
Workers	1
Data source	<a href="#">bq://cloud-training-demos.xyz_team_dataset.tidyadvertising_1_string_int</a>
Entity type	<a href="#">budget</a>
Featurestore	<a href="#">hello_world</a>
Ingested entities	1,200

# Format of input data for batch ingestion

Data for ingestion should have the following columns:

- Entity\_id: The ID of ingested entity
- Timestamp: The timestamp at which the feature was generated/computed (this timestamp is critical for correct feature serving)
- Feature columns that match the destination feature name

Example:

user_id	timestamp	Feature 1	Feature 2	Feature 3
88	1552521600	true	177	“abc”
88	1552521600	true	666	“xyz”
96	1569542400	false	235	“foobar”

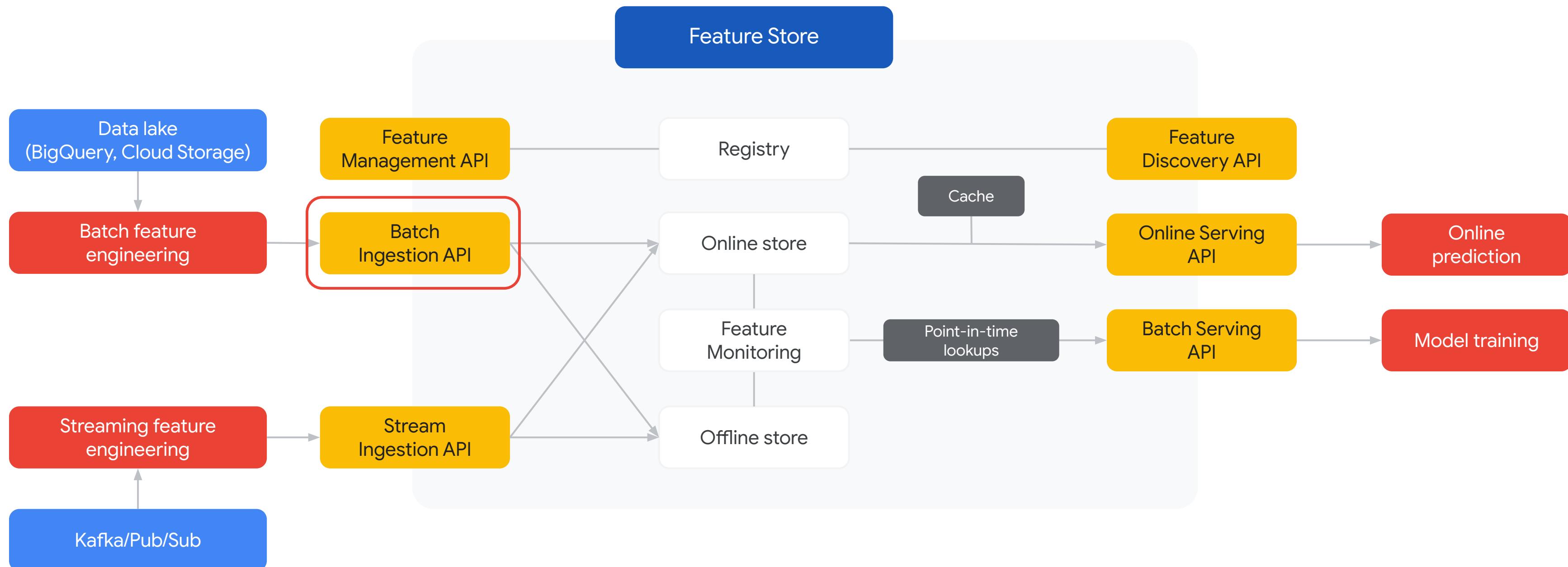
# Feature serving: Batch and online methods

# Feature serving

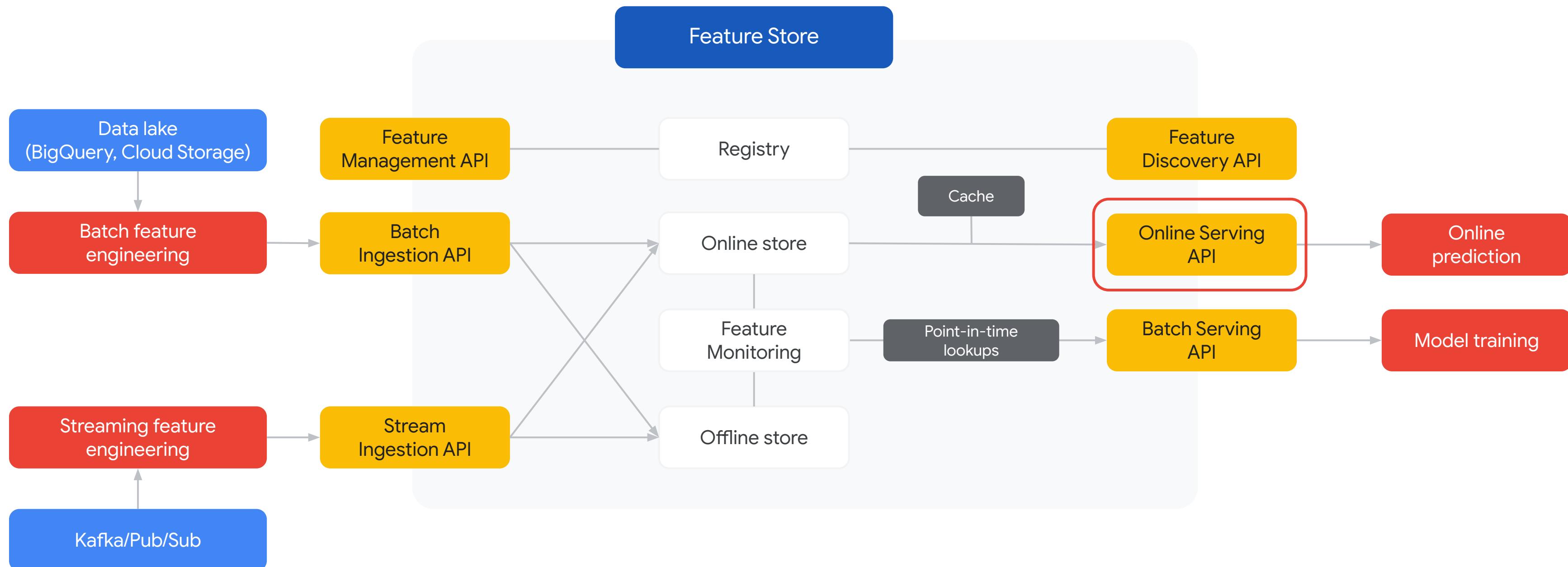
- *Feature serving* is the process of exporting stored feature values for training or inference
- Feature Store offers two methods for serving features: batch and online



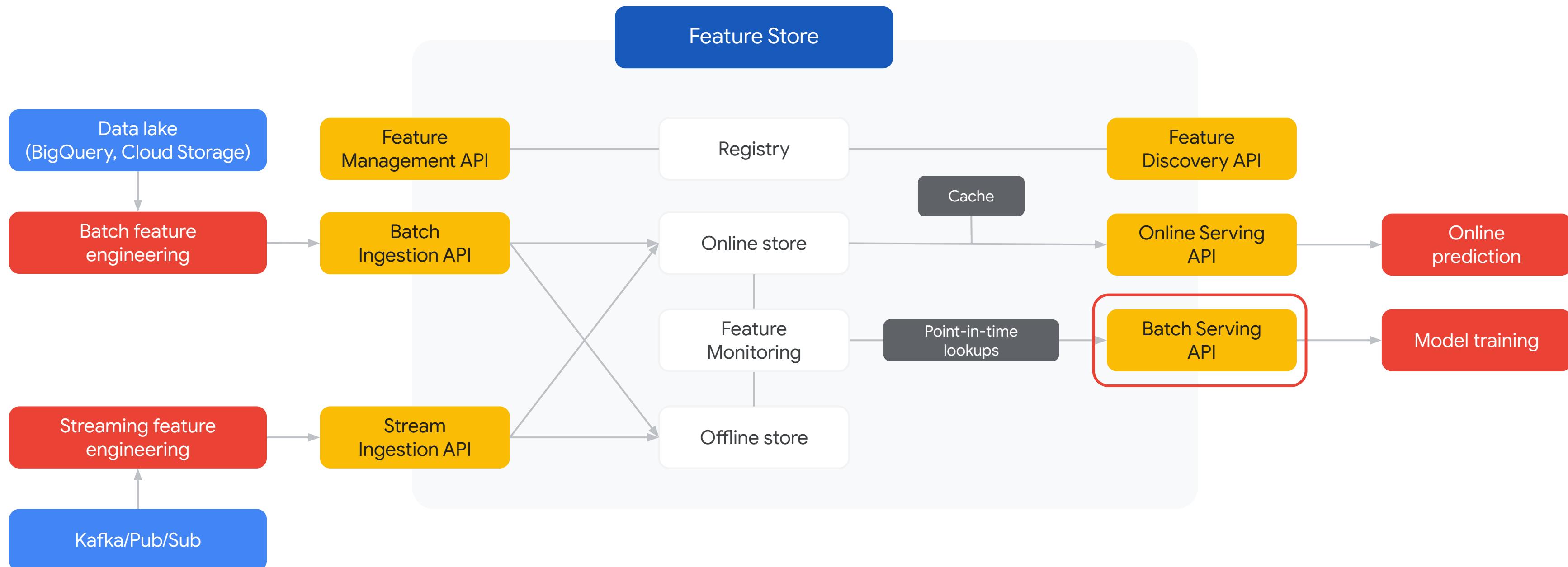
# Simple data scientist—friendly APIs and SDKs abstract away the underlying complexity



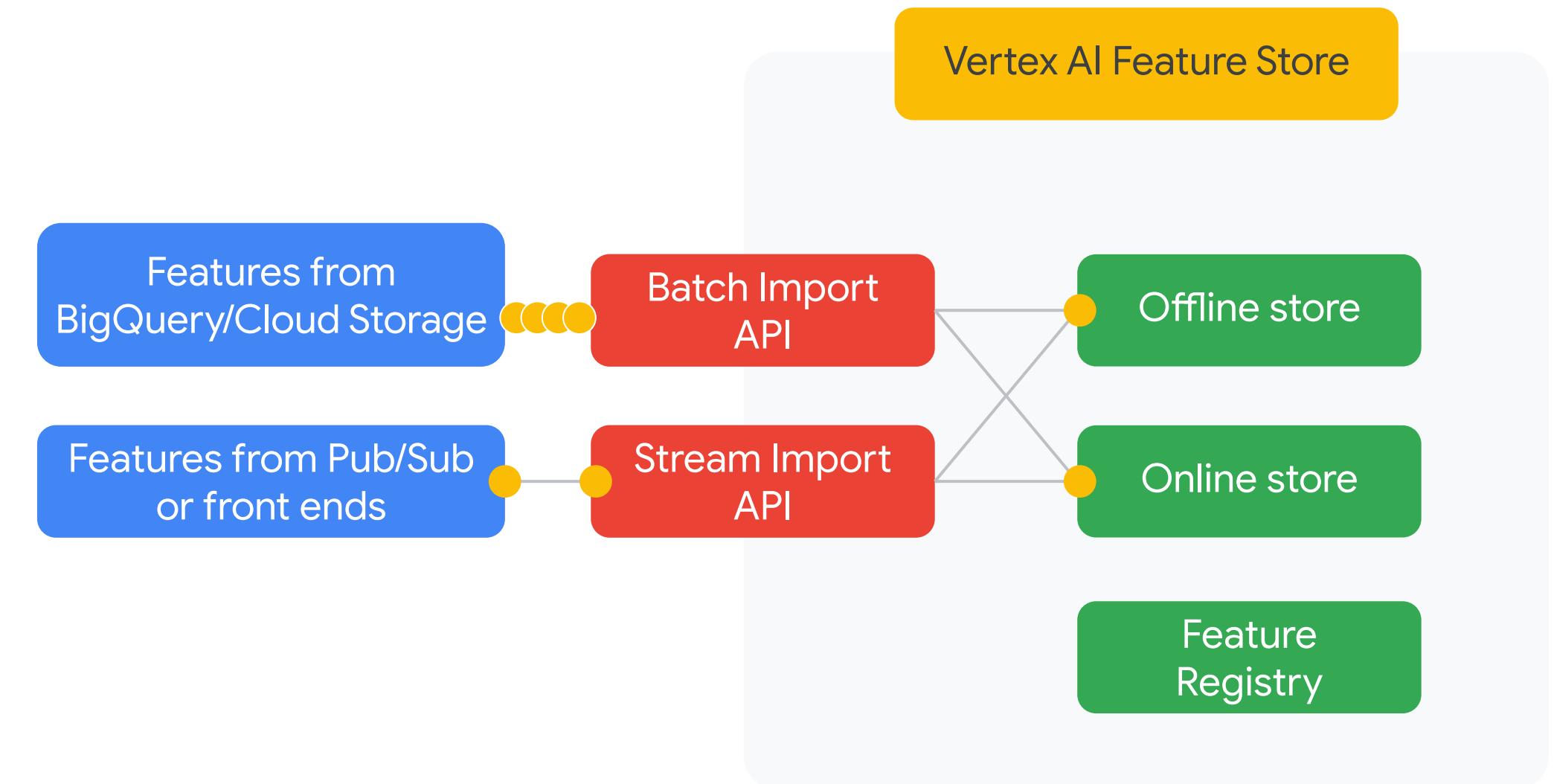
# Simple data scientist—friendly APIs and SDKs abstract away the underlying complexity



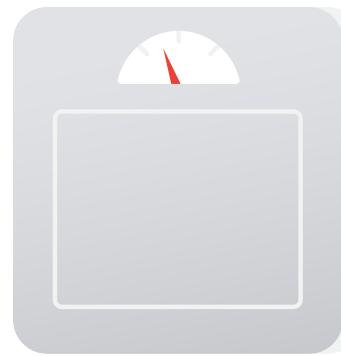
# Simple data scientist—friendly APIs and SDKs abstract away the underlying complexity



# With Vertex AI Feature Store, you can store features with Batch and Stream Import APIs



# Our goal is to predict the weight of newborns so that all babies can get the care they need



---

Predict the  
weight of  
newborns



---

Identify babies  
who may need  
special facilities



---

Get babies the  
care they need

# An open dataset of births is available in **BigQuery**

Births recorded in the 50 states of the USA from 1969 to 2008

Table ID	bigquery-public-data:samples.natality
Table size	21.9 GB
Long term storage size	21.9 GB
Number of rows	137,826,763

<https://bigquery.cloud.google.com/table/bigquery-public-data:samples.natality>

# The data set includes details about the pregnancy

	year	INTEGER	NULLABLE	Four-digit year of the birth. Example: 1975.
Date of birth	month	INTEGER	NULLABLE	Month index of the date of birth, where 1=January.
	day	INTEGER	NULLABLE	Day of birth, starting from 1.
	wday	INTEGER	NULLABLE	Day of week, where 1 is Sunday and 7 is Saturday.
Location of birth (US state)	state	STRING	NULLABLE	The two character postal code for the state. Entries after 2004 do not include this value.
Baby's birth weight (lbs)	weight_pounds	FLOAT	NULLABLE	Weight of the child, in pounds.
Mother's age at birth	mother_age	INTEGER	NULLABLE	Reported age of the mother when giving birth.
Duration of pregnancy	gestation_weeks	INTEGER	NULLABLE	The number of weeks of the pregnancy.
Mother's weight gain (lbs)	weight_gain_pounds	INTEGER	NULLABLE	Number of pounds gained by the mother during pregnancy.

**“Historical data”  
because the  
babies have  
already been born**

year	INTEGER	NULLABLE	Four-digit year of the birth. Example: 1975.
month	INTEGER	NULLABLE	Month index of the date of birth, where 1=January.
day	INTEGER	NULLABLE	Day of birth, starting from 1.
wday	INTEGER	NULLABLE	Day of week, where 1 is Sunday and 7 is Saturday.
state	STRING	NULLABLE	The two character postal code for the state. Entries after 2004 do not include this value.
weight_pounds	FLOAT	NULLABLE	Weight of the child, in pounds.
mother_age	INTEGER	NULLABLE	Reported age of the mother when giving birth.
gestation_weeks	INTEGER	NULLABLE	The number of weeks of the pregnancy.
weight_gain_pounds	INTEGER	NULLABLE	Number of pounds gained by the mother during pregnancy.

# This is what we will build

### Baby weight predictor

*Example application to predict a baby's weight.*

Mother's age

Gestation weeks

Plurality

Baby's gender  Male  Female  Unknown

**PREDICT**

Prediction 4.36 lbs.

### Baby weight predictor

*Example application to predict a baby's weight.*

Mother's age

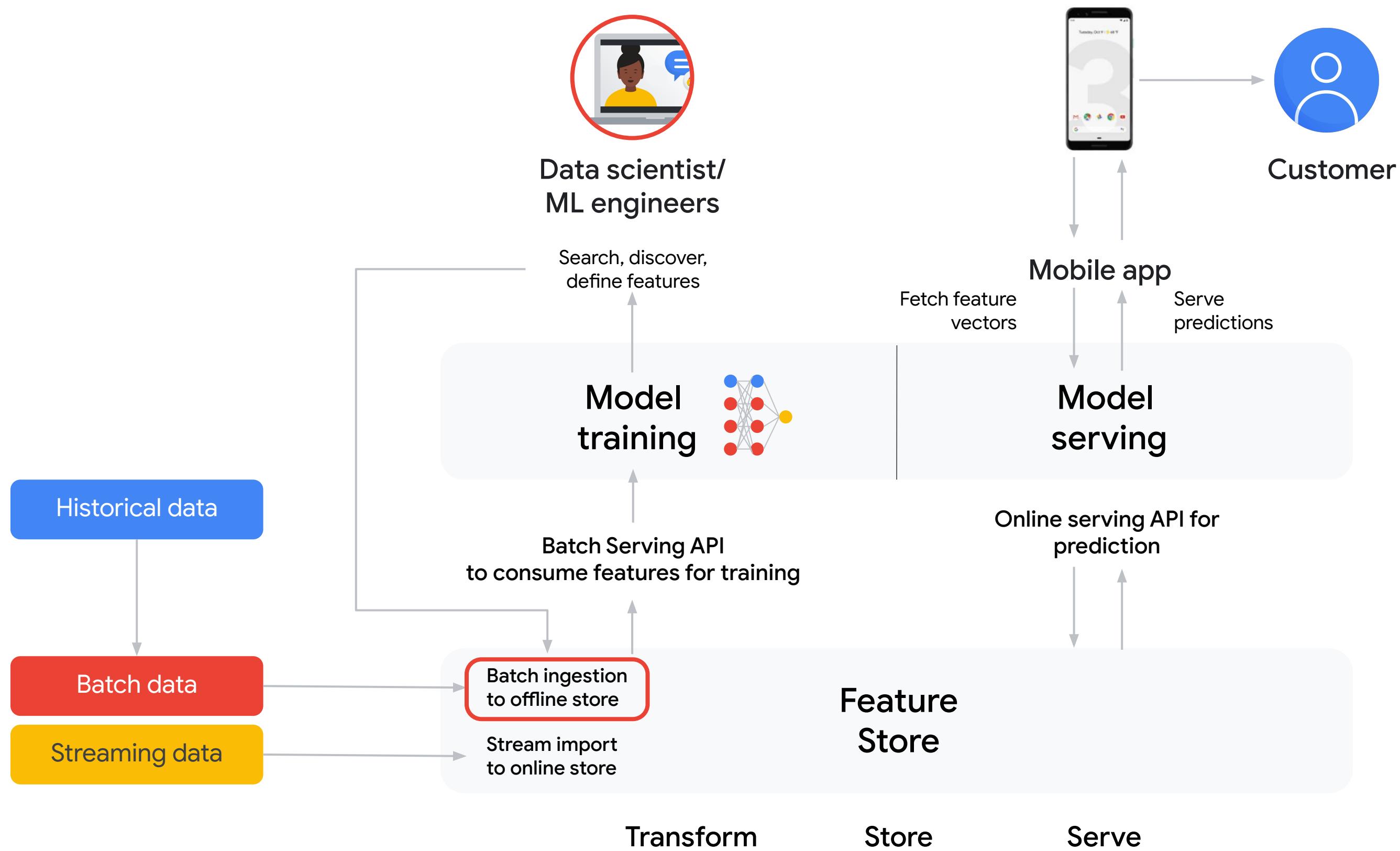
Gestation weeks

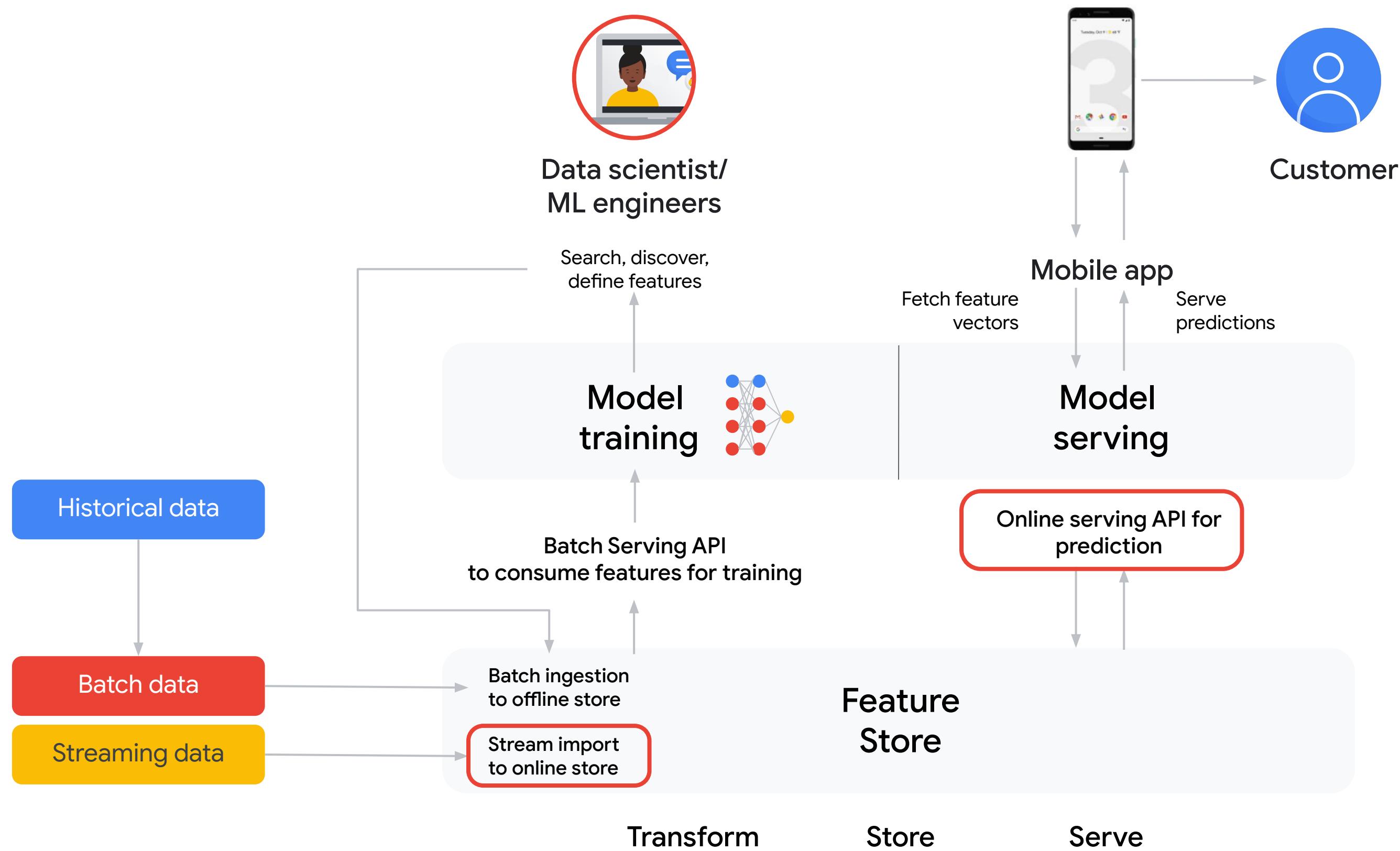
Plurality

Baby's gender  Male  Female  Unknown

**PREDICT**

Prediction 7.19 lbs.





# Batch serving specifying required feature values

## Generate the labeled training data:

As part of a batch serving request, the following information is required:

- A list of existing features to get values for
- A read-instance list that contains information for each training example
- The destination URI and format where the output is written. In the output, Feature Store essentially joins the table from the read instances list and the feature values from the featurestore. Specify one of the following formats and locations for the output:
  - CSV file in a regional or multi-regional Cloud Storage bucket. But if your feature values include arrays, you must choose another format.
  - Tfrecord file in a Cloud Storage bucket

Column	Description
user_id	ID of user
item_id	ID of item
ds	Impression timestamp
label	Binary label whether user bought the item

# Feature serving

Vertex AI

Batch serving jobs

REFRESH

Region: us-central1 (Iowa)

Batch serving job Featurestore Elapsed time Entity types Feature values Last updated Started Status

Features

Labeling tasks

Workbench

Pipelines

Training

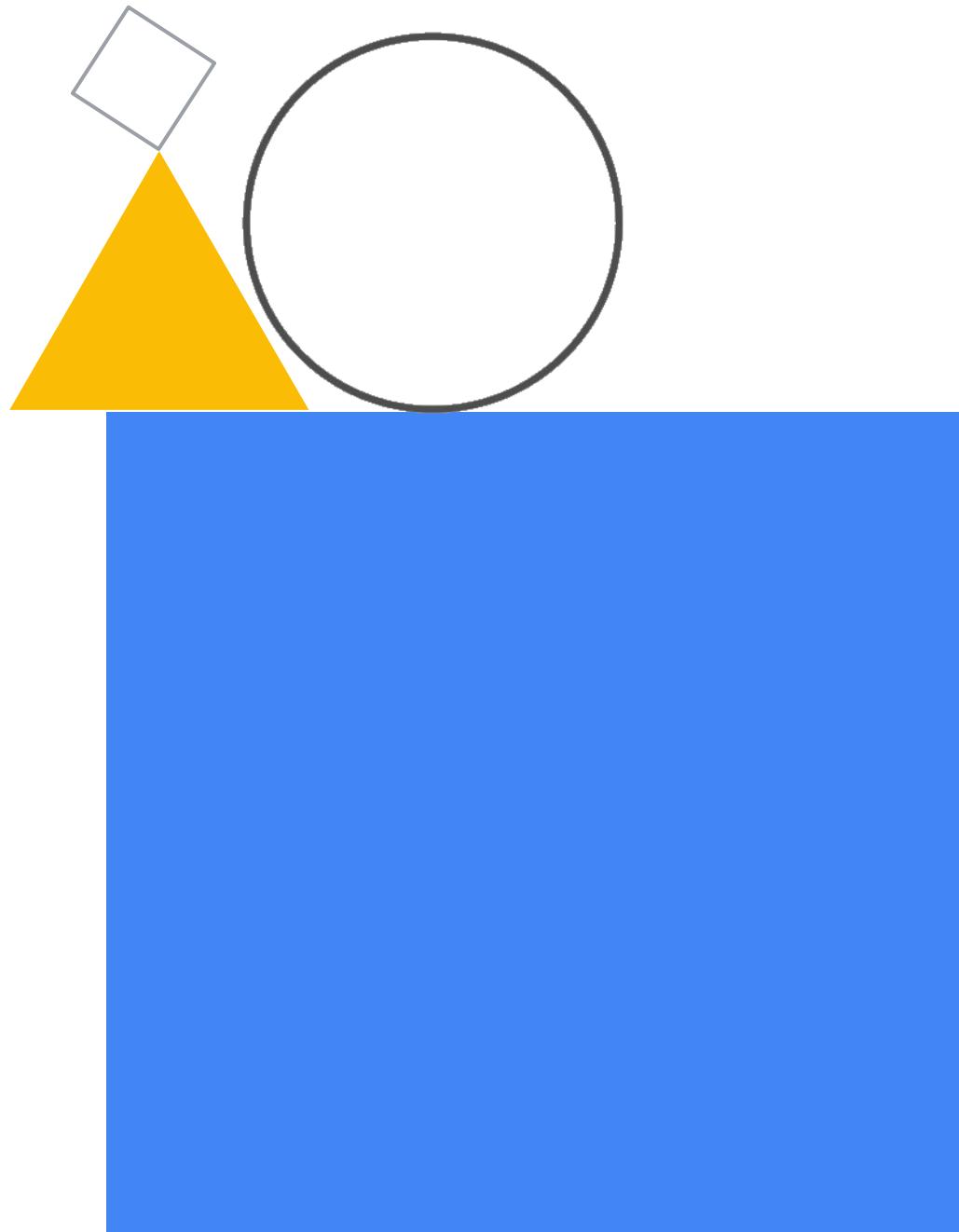
Experiments

Marketplace

Show debug pane

You don't have any batch serving jobs in this project yet  
You can create a batch serving job through the Feature Store API

# Raw Data to Features

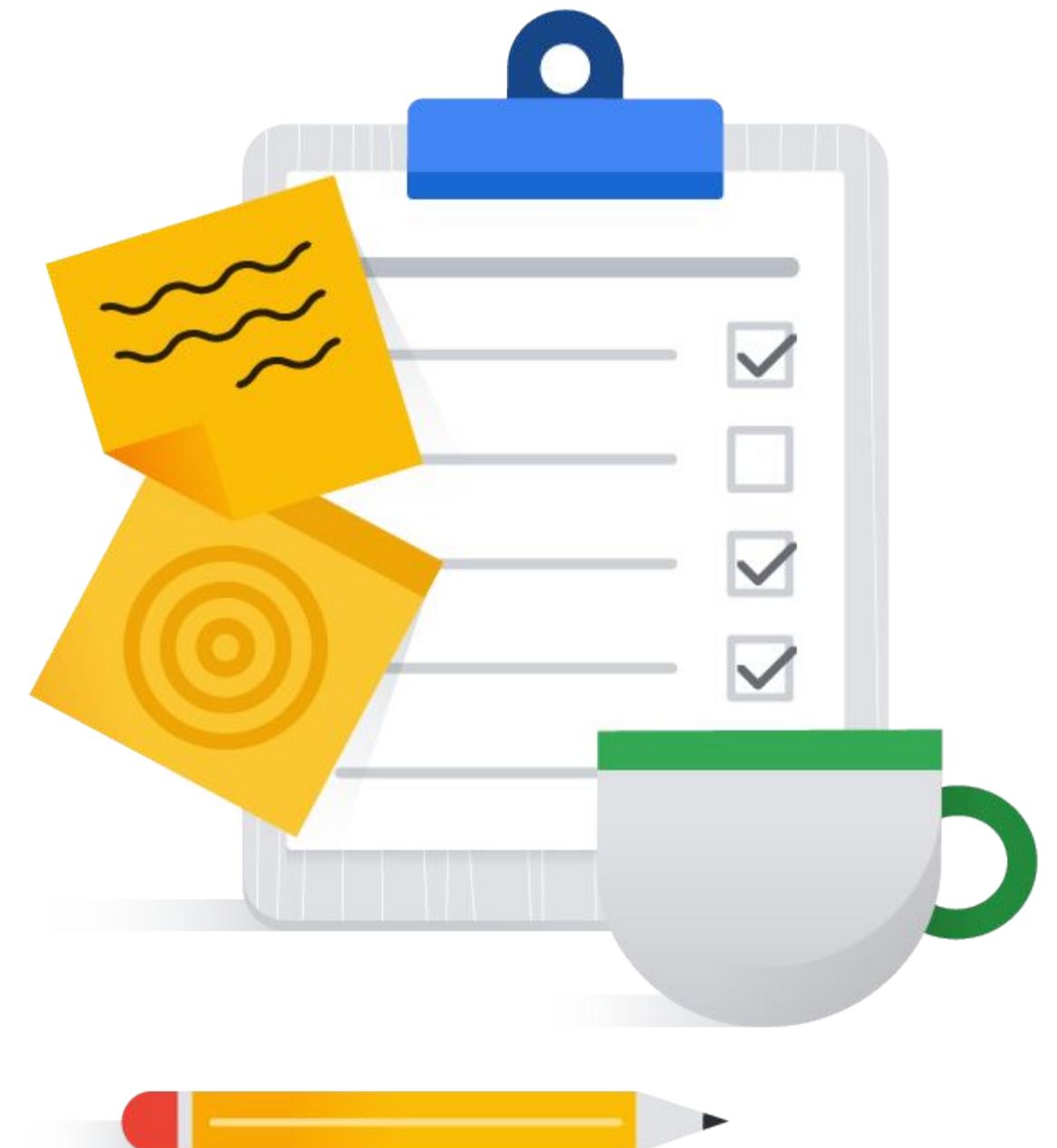


# In this module, you learn to ...

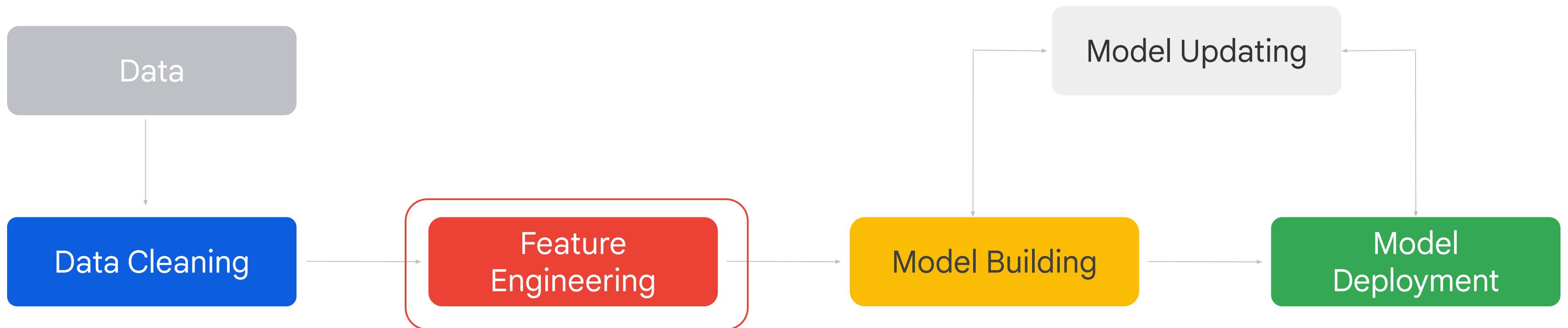
01 Turn raw data into features

02 Compare good features versus bad features

03 Represent features



# Feature engineering in predictive modeling



# 1 - Process (What?) of feature engineering

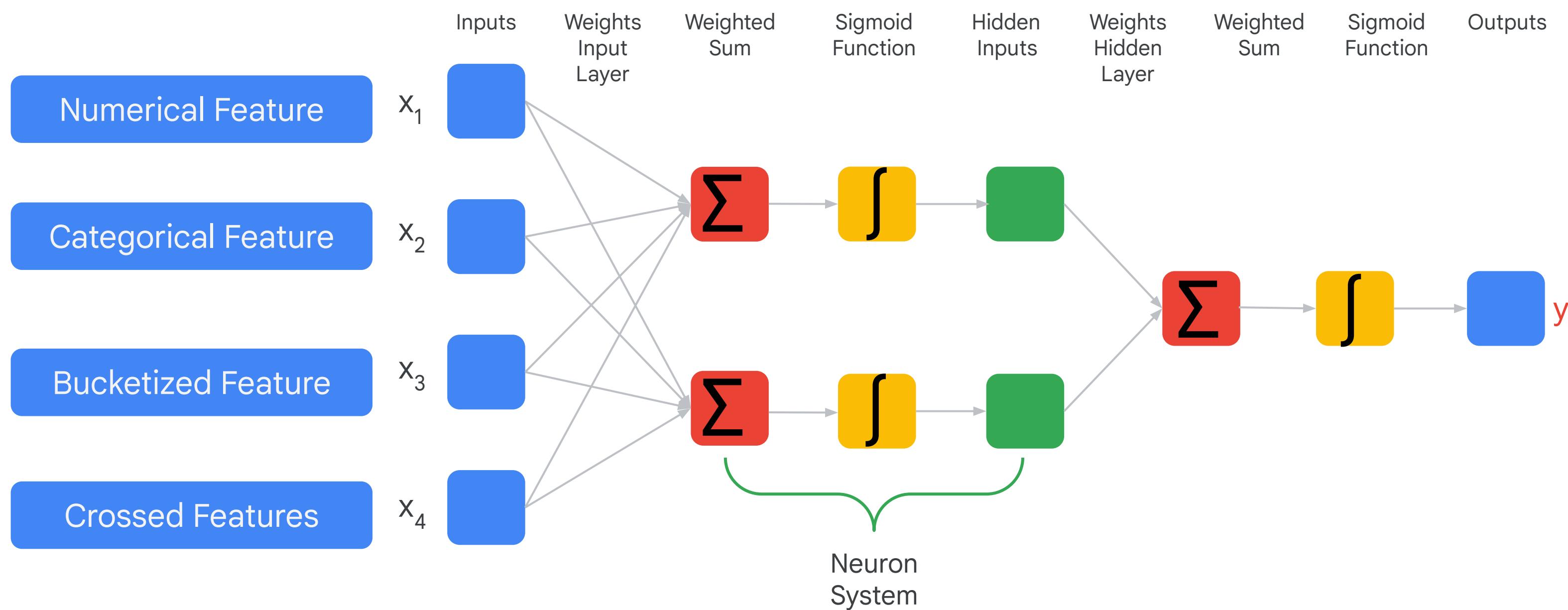
## Definition

This process attempts to create additional relevant features from the existing raw features in the data and to increase the predictive power of the learning algorithm.

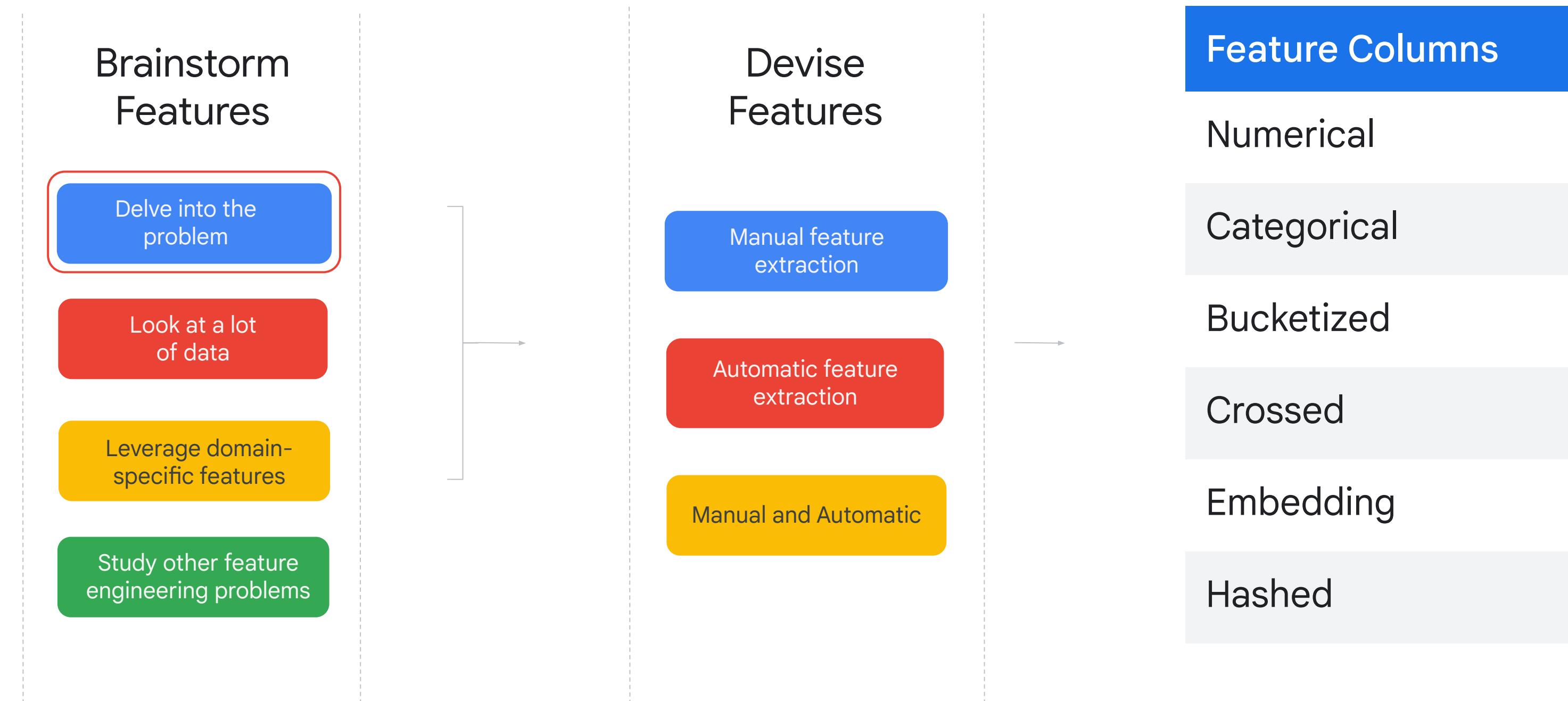
“ Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data. ”

Prof. Andrew Ng

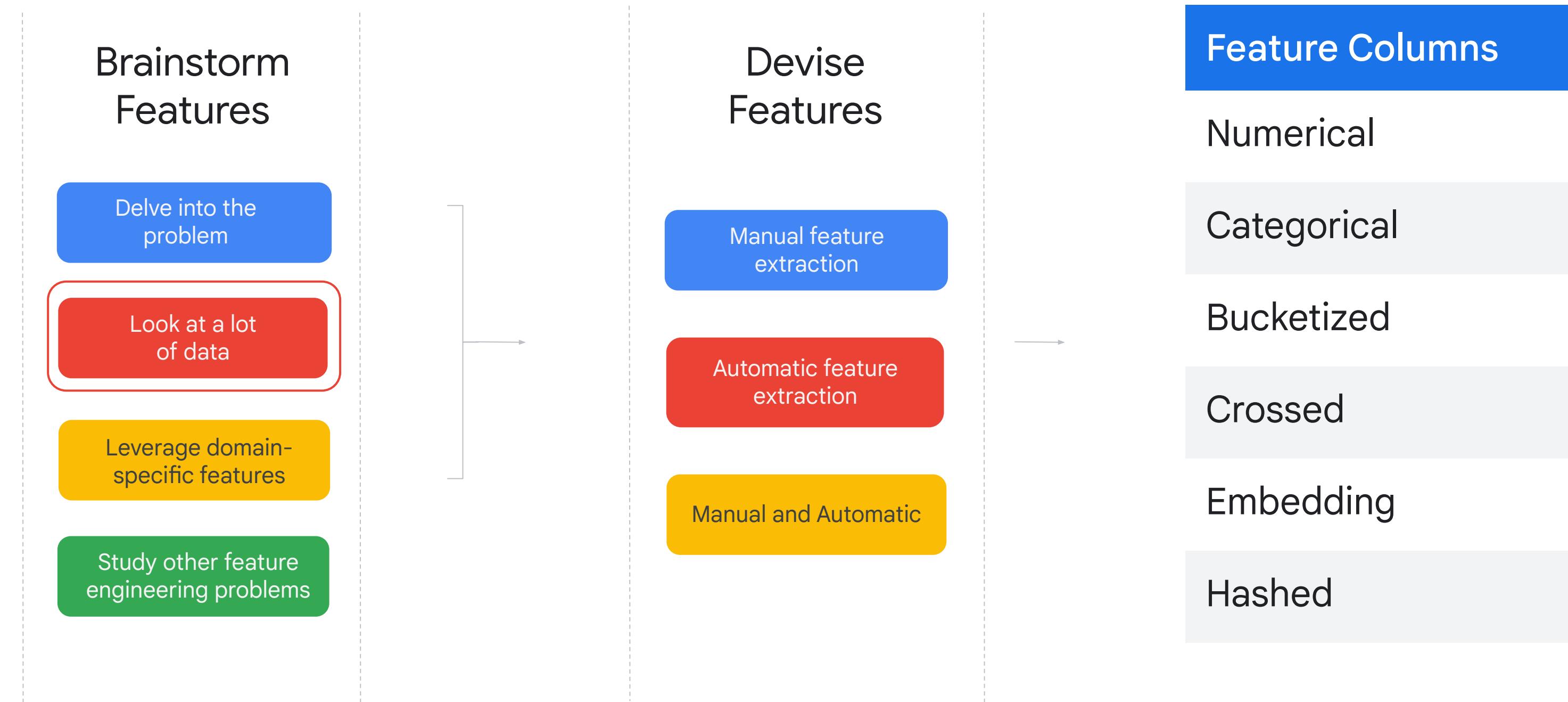
## 2 - Purpose (Why): To improve the accuracy of models by increasing the predictive power of learning algorithms



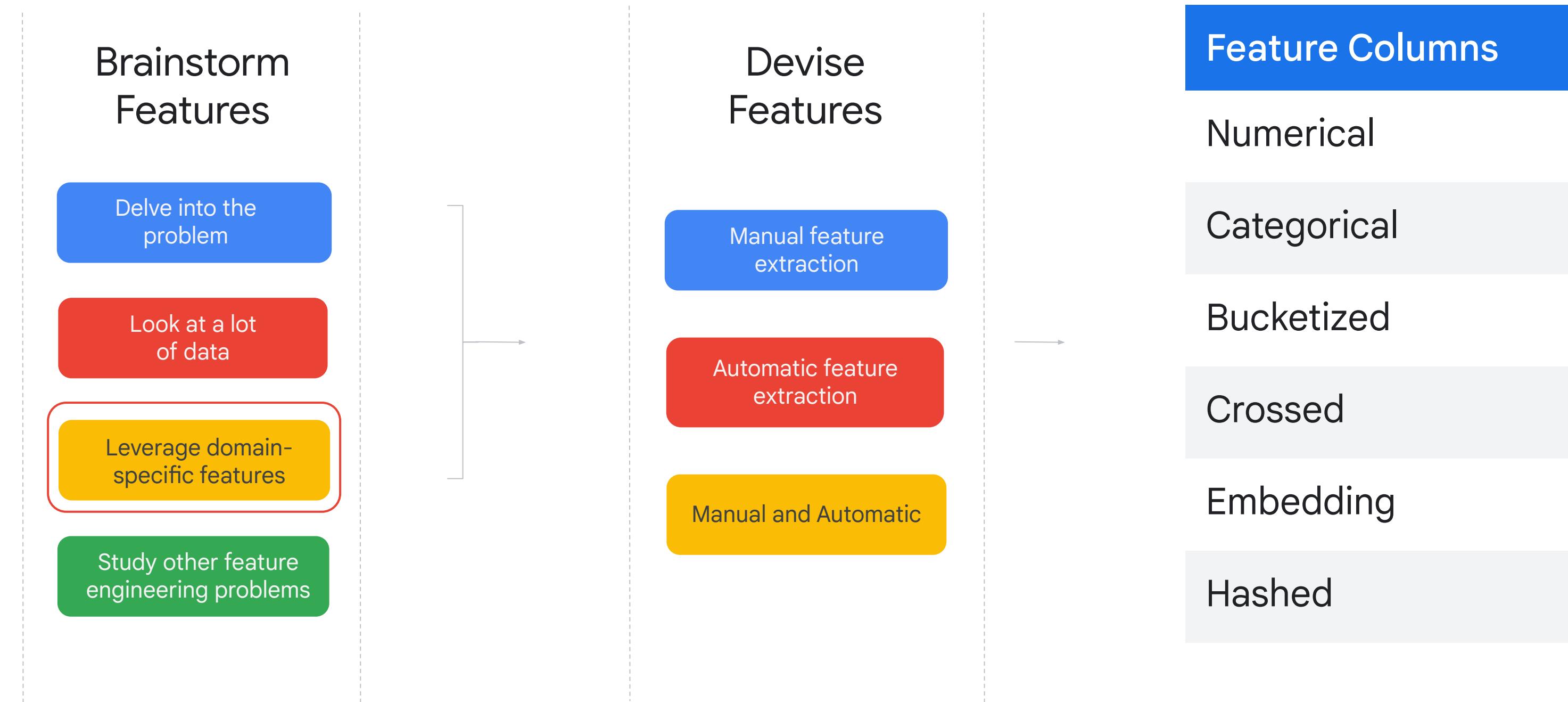
# 3 - Process (How?) of feature engineering



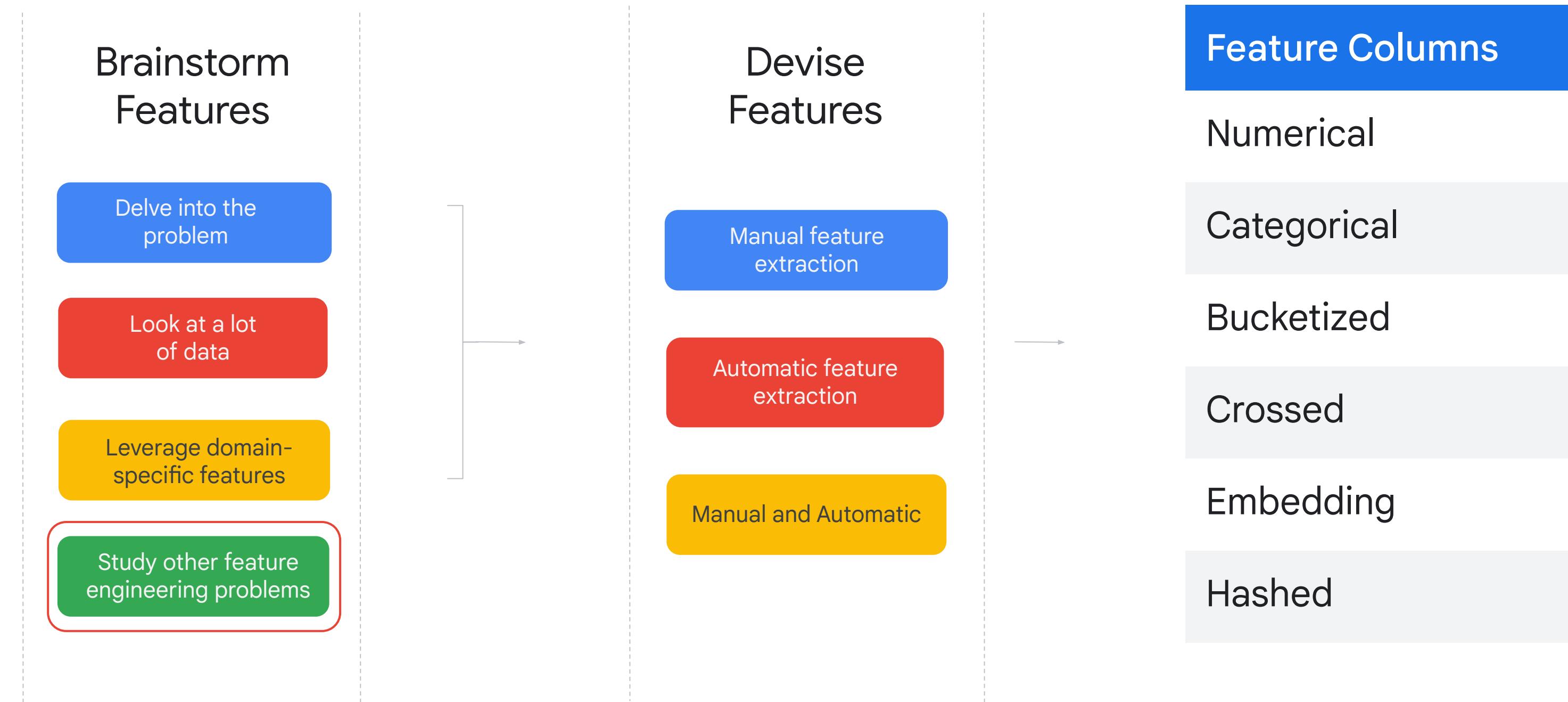
# 3 - Process (How?) of feature engineering



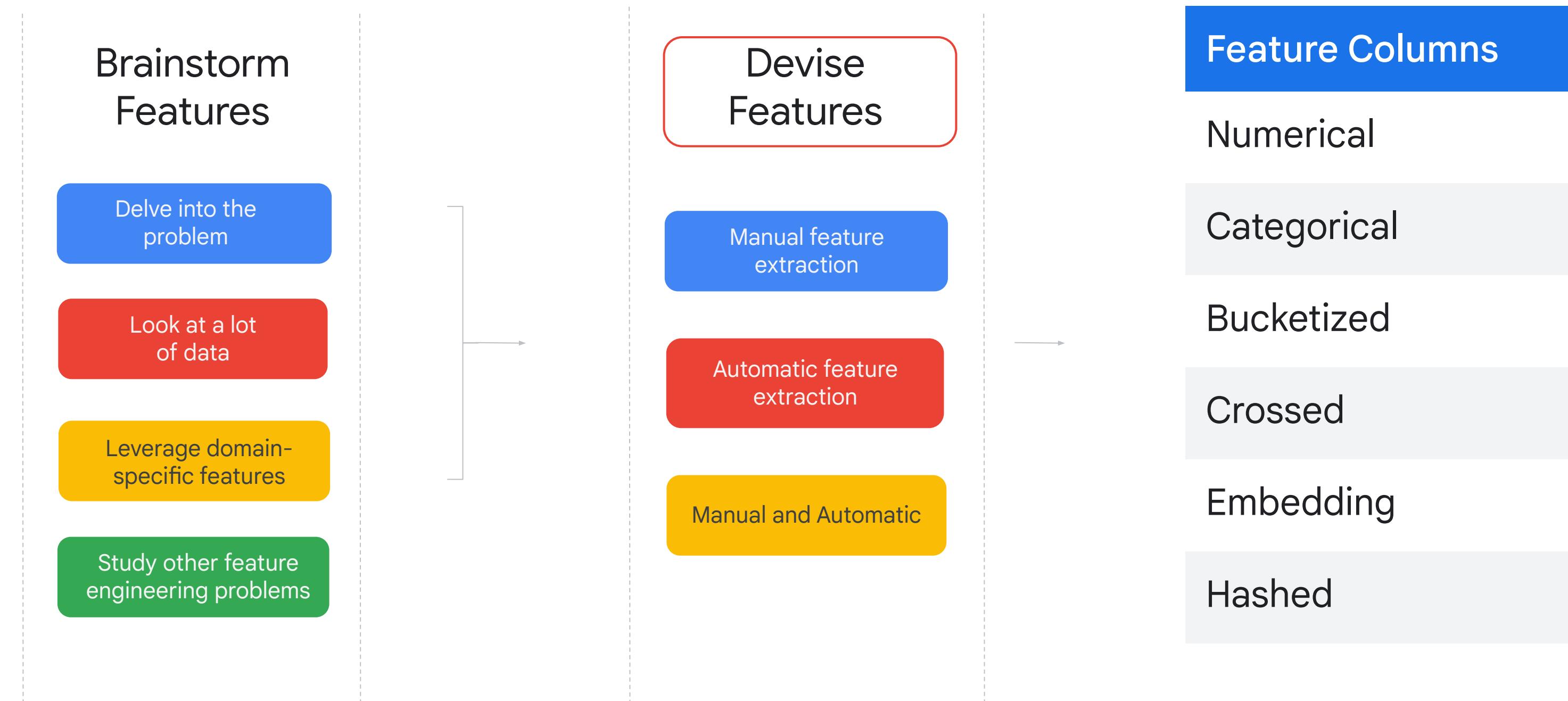
# 3 - Process (How?) of feature engineering



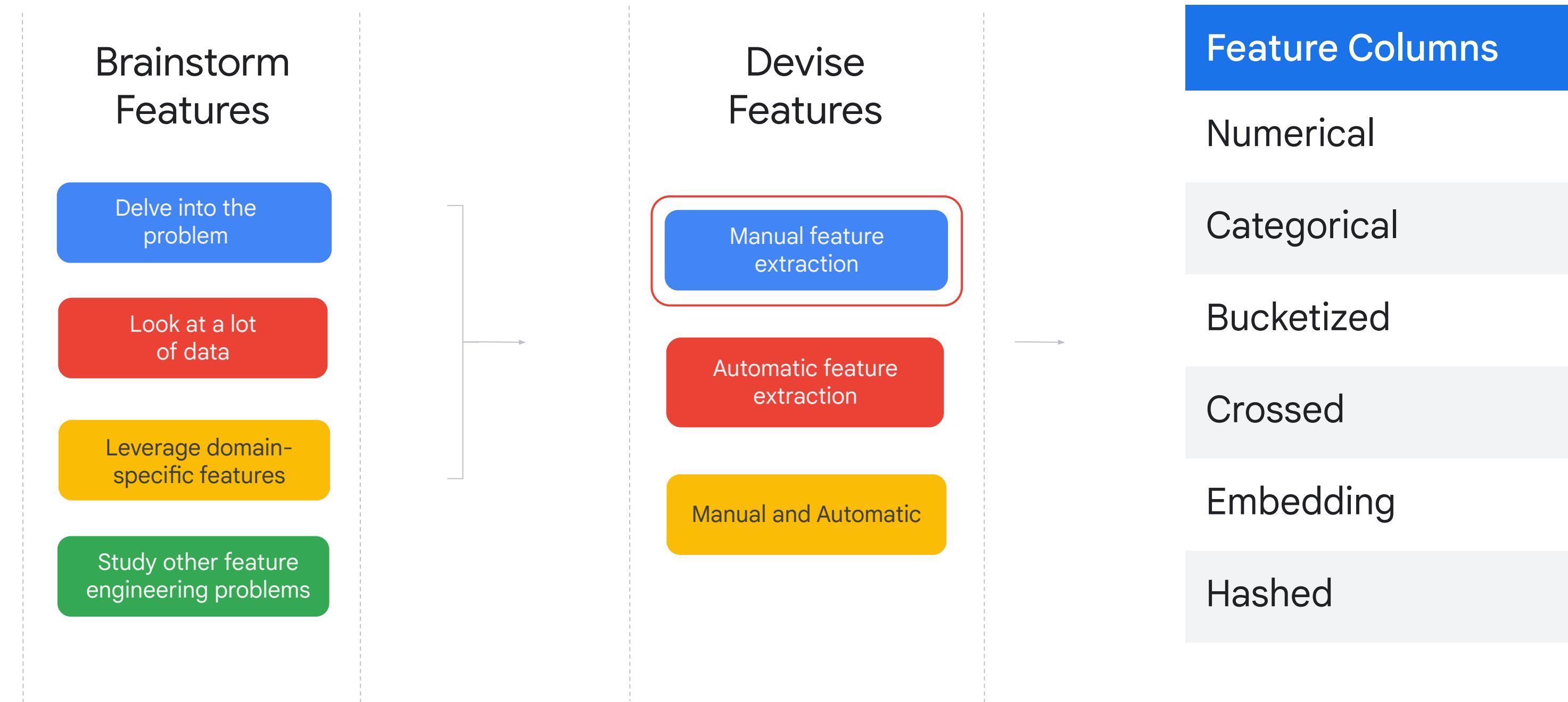
# 3 - Process (How?) of feature engineering



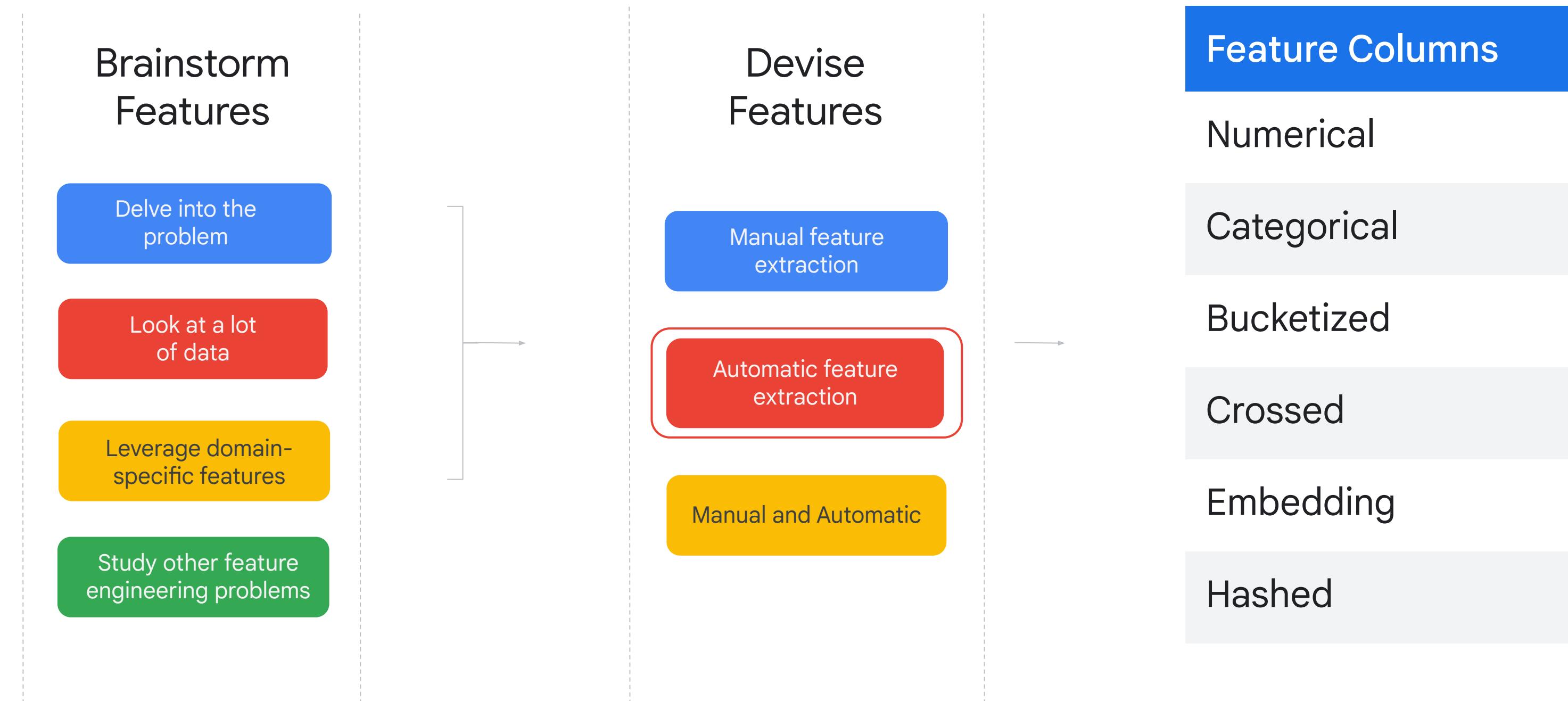
# 3 - Process (How?) of feature engineering



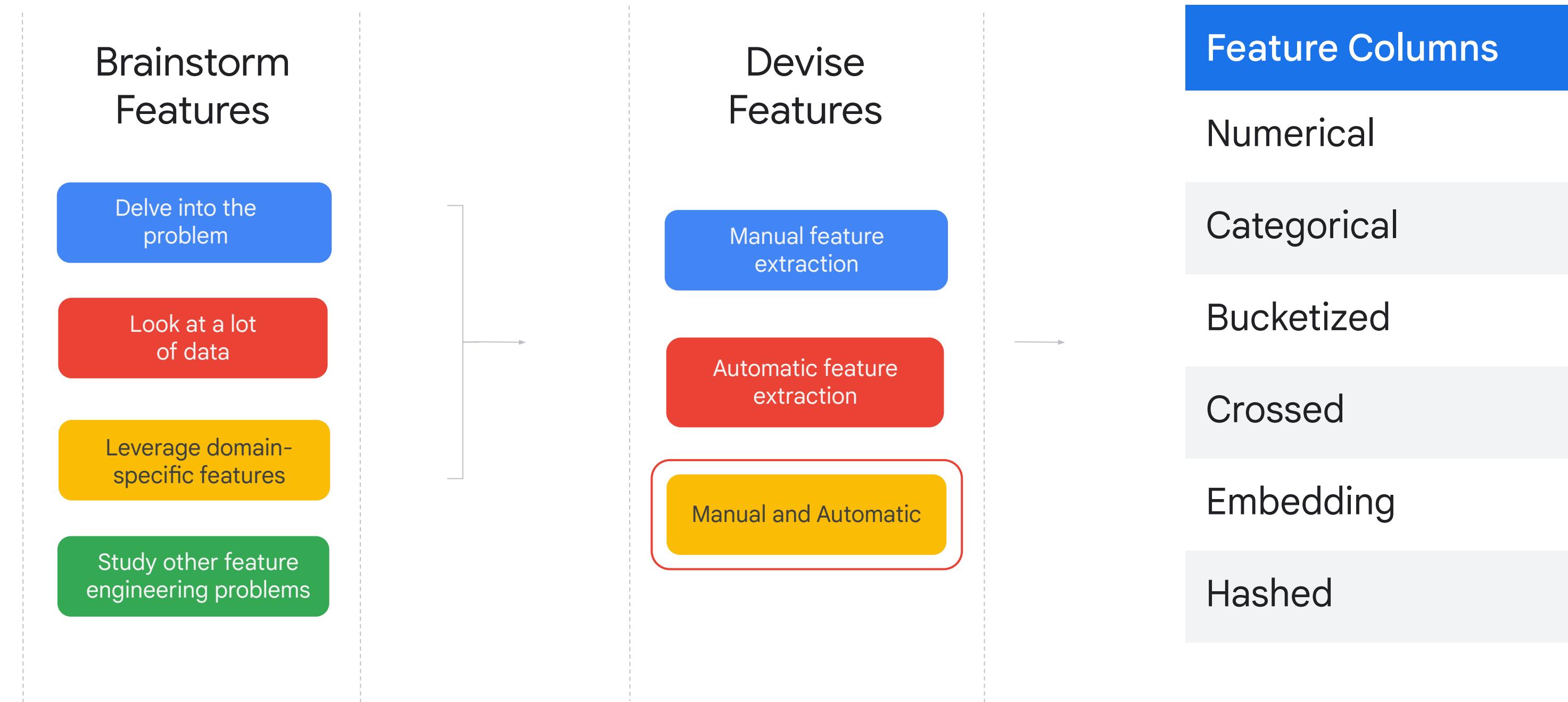
# 3 - Process (How?) of feature engineering



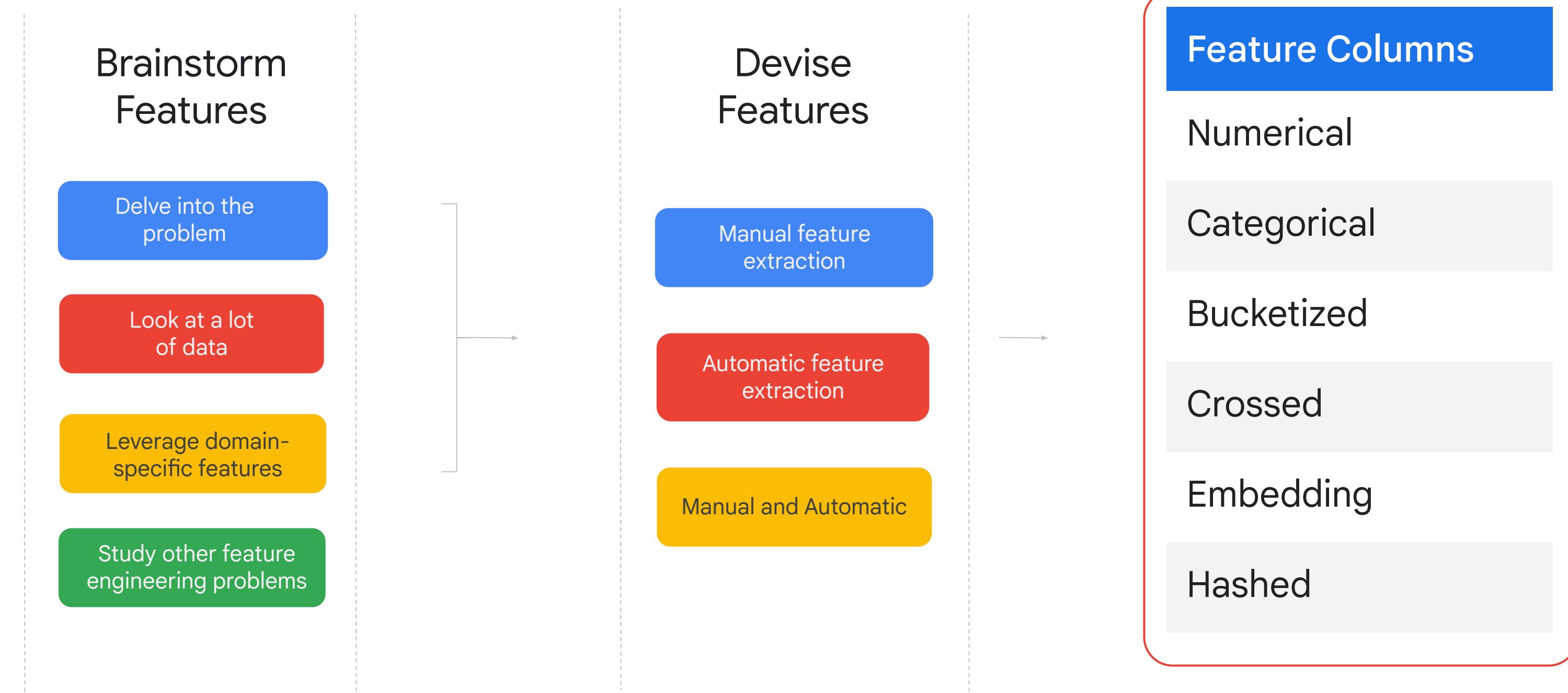
# 3 - Process (How?) of feature engineering



# 3 - Process (How?) of feature engineering



# 3 - Process (How?) of feature engineering



# Different problems in the same domain may need different features

The key learning is that different problems in the same domain may need different features.

It depends on you and your subject matter expertise to determine which fields you want to start with for your hypothesis.

“

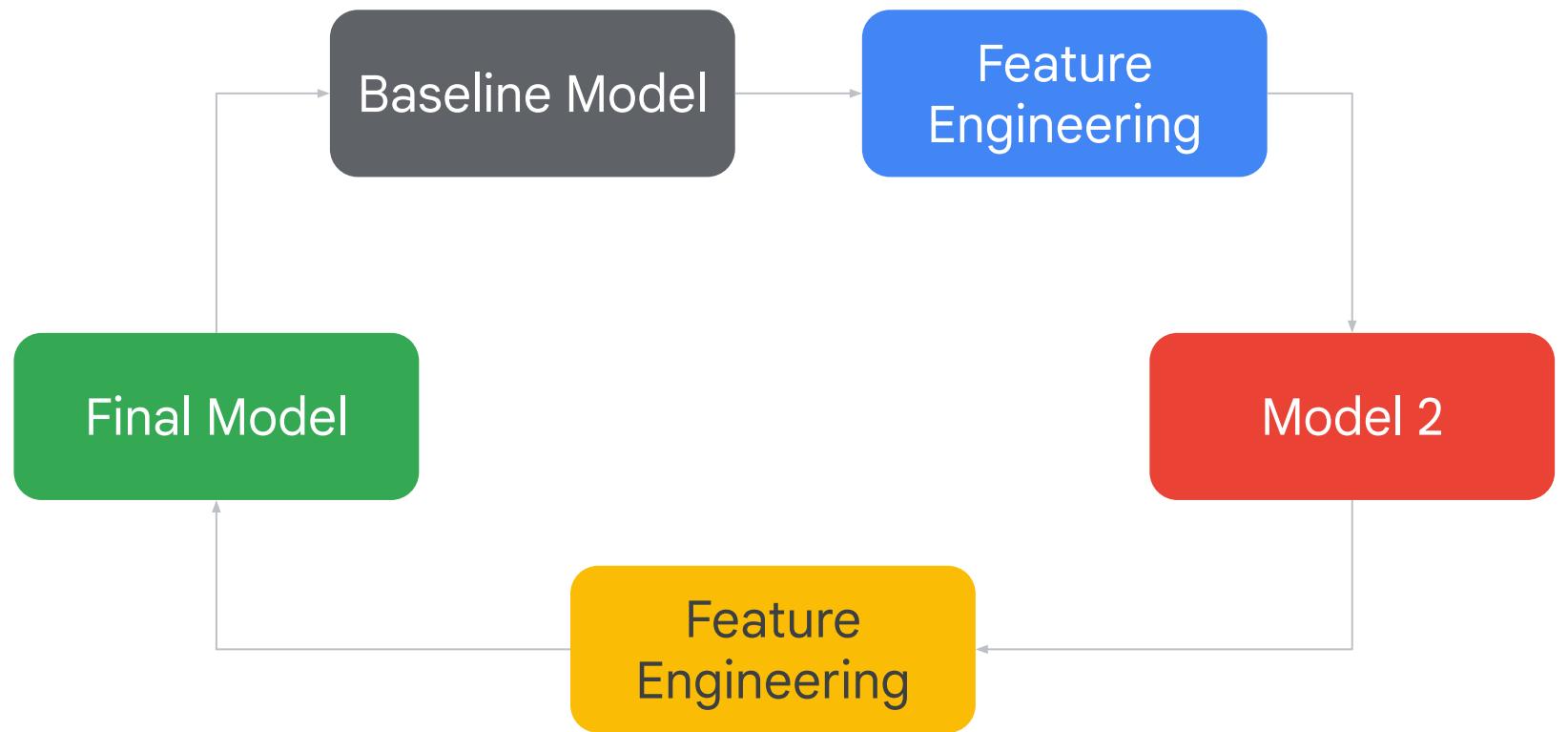
Coming up with features is difficult, time-consuming, [and] requires expert knowledge. ‘Applied machine learning’ is basically feature engineering.

”

Prof. Andrew Ng

## 4 - Iterative Process

Example: Process can continue until  
RMSE is lowest



# 5 - Feature engineering types

Type	Example
Using indicator variables to isolate key information.	Isolates a specific area for our training dataset.
Highlighting interactions between two or more features.	Sum of two features, product of two features, etc.
	Create a new feature "grade" with "Elementary School," "Middle School," and "High School" as classes.
Representing the same feature in a different way.	Group similar classes, and then group the remaining ones into a single "Other" class.
	Transform categorical features into dummy variables.

# 5 - Feature engineering types

Type	Example
Using indicator variables to isolate key information.	Isolates a specific area for our training dataset.
Highlighting interactions between two or more features.	Sum of two features, product of two features, etc.
	Create a new feature "grade" with "Elementary School," "Middle School," and "High School" as classes.
Representing the same feature in a different way.	Group similar classes, and then group the remaining ones into a single "Other" class.
	Transform categorical features into dummy variables.

# 5 - Feature engineering types

Type	Example
Using indicator variables to isolate key information.	Isolates a specific area for our training dataset.
Highlighting interactions between two or more features.	Sum of two features, product of two features, etc.
	Create a new feature "grade" with "Elementary School," "Middle School," and "High School" as classes.
Representing the same feature in a different way.	Group similar classes, and then group the remaining ones into a single "Other" class.  Transform categorical features into dummy variables.

# 5 - Feature engineering types

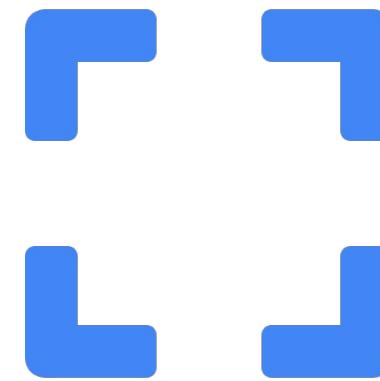
Type	Example
Using indicator variables to isolate key information.	Isolates a specific area for our training dataset.
Highlighting interactions between two or more features.	Sum of two features, product of two features, etc.
	Create a new feature "grade" with "Elementary School," "Middle School," and "High School" as classes.
Representing the same feature in a different way.	Group similar classes, and then group the remaining ones into a single "Other" class.
	Transform categorical features into dummy variables.

# 5 - Feature engineering types

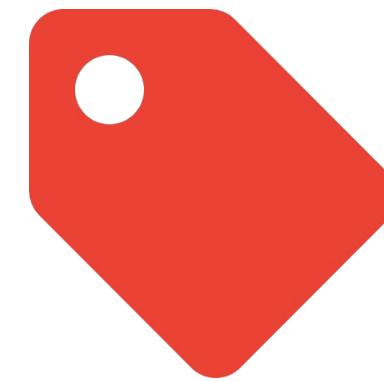
Type	Example
Using indicator variables to isolate key information.	Isolates a specific area for our training dataset.
Highlighting interactions between two or more features.	Sum of two features, product of two features, etc.
Representing the same feature in a different way.	Create a new feature "grade" with "Elementary School," "Middle School," and "High School" as classes.  Group similar classes, and then group the remaining ones into a single "Other" class.  Transform categorical features into dummy variables.

**Raw data to feature vectors**

# What raw data do we need to collect to predict the price of a house?



Lot size  
Number of rooms

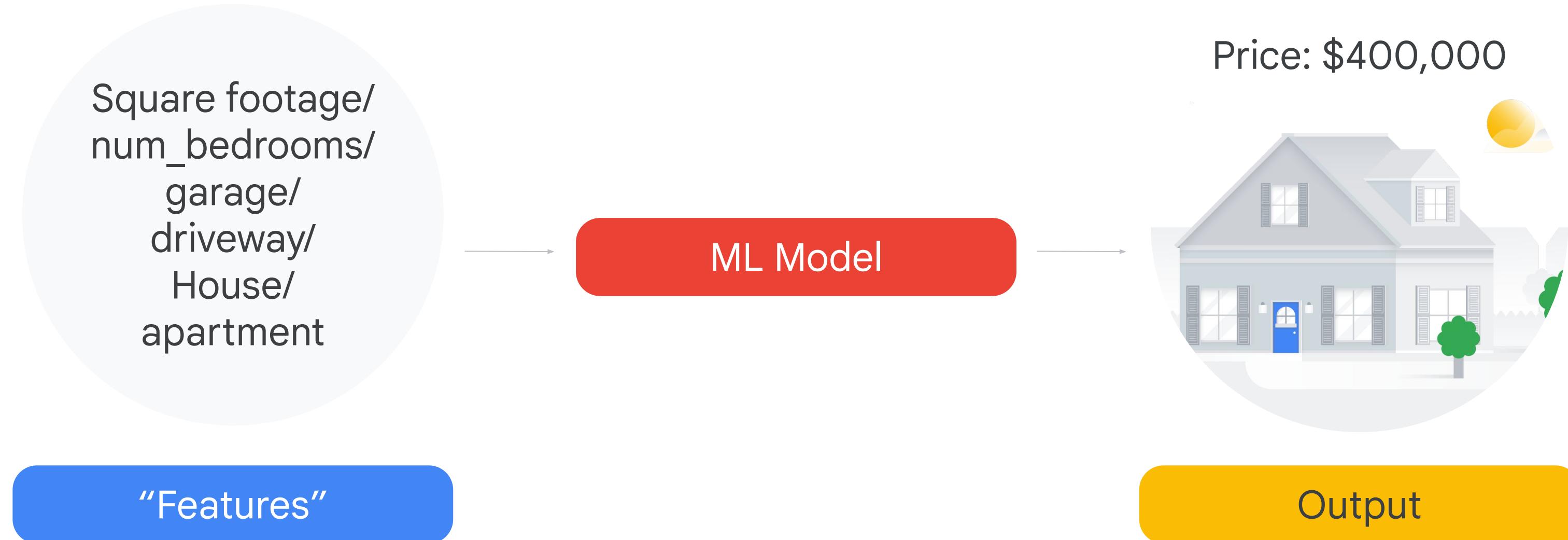


Historical  
sale price



Location, location,  
location

# Predict property value from historical data



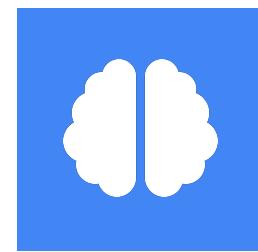
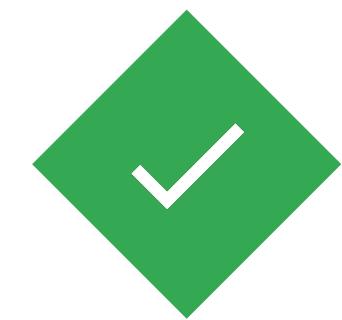
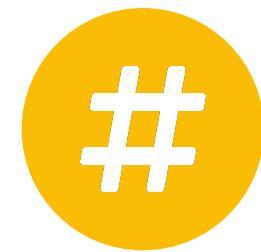
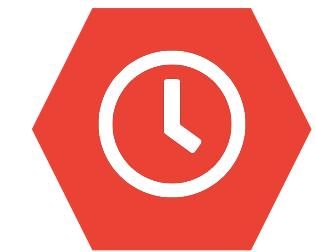
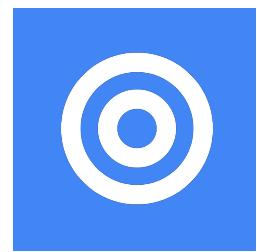
# Raw data must be mapped into numerical feature vectors

```
0 : {  
    house_info : {  
        ✓ num_rooms: 6  
        ✓ num_bedrooms: 3  
        ✓ street_name: "Main Street"  
        ✓ num_basement_rooms: -1  
        ...  
    }  
}
```



```
[  
  6.0,  
  1.0,  
  0.0,  
  0.0,  
  0.0,  
  9.321,  
  -2.20,  
  1.01,  
  0.0,  
  ...  
,  
]
```

# What makes a good feature?



— 1 —

— 2 —

— 3 —

— 4 —

— 5 —

Be related to the  
objective

Be known at  
prediction-time

Be numeric with  
meaningful  
magnitude

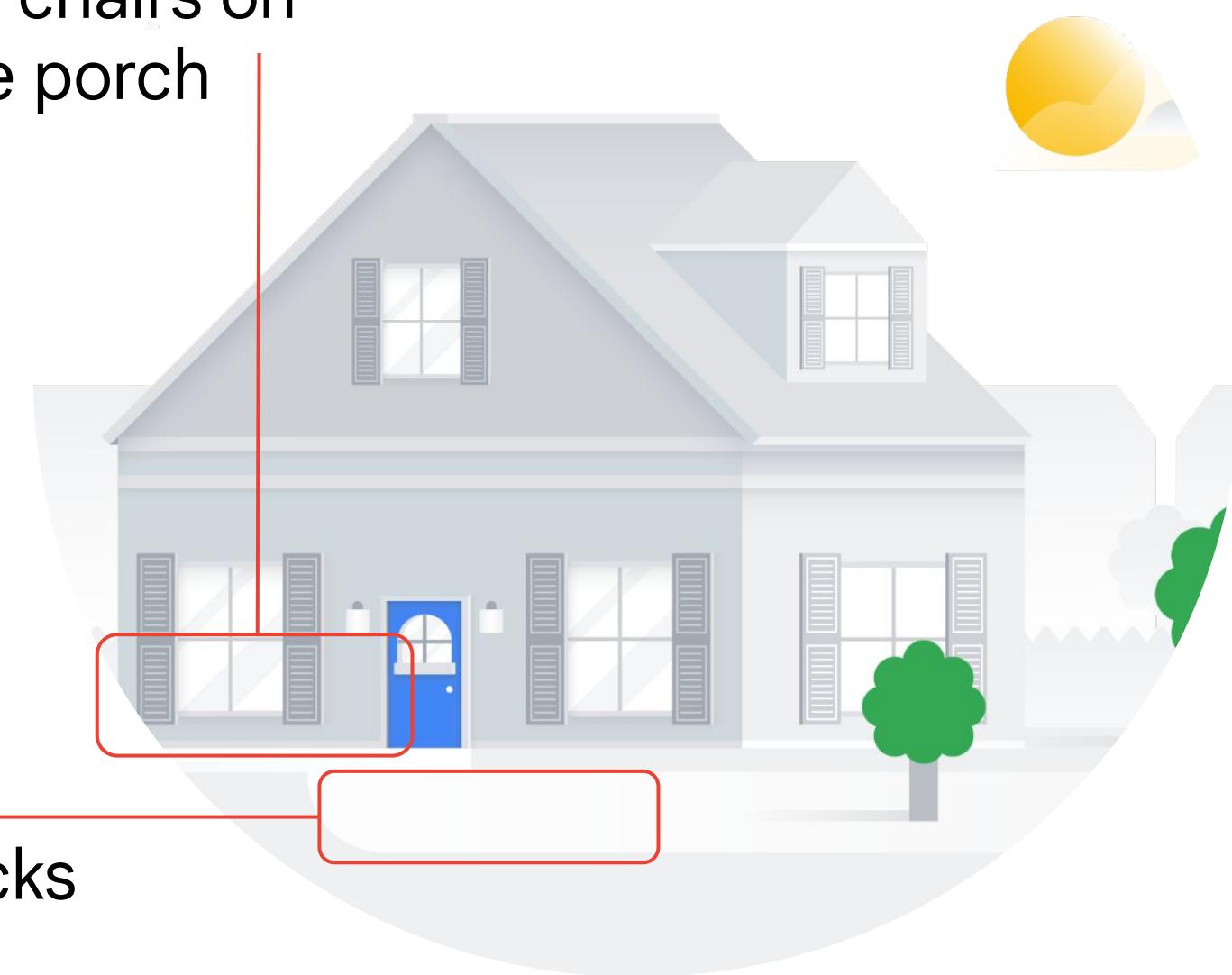
Have enough  
examples

Bring human  
insight to  
problem

**Be related to  
the objective**

No chairs on  
the porch

Number of  
concrete blocks  
in driveway



# 1 - Is related to the objective

Raw Data

```
{  
  "vendor_id": "CMT",  
  "pickup_datetime": "2010-02-05 01:20:05 UTC",  
  "dropoff_datetime": "2010-02-05 01:41:28 UTC",  
  "pickup_longitude": "-73.979935",  
  "pickup_latitude": "40.761105",  
  "dropoff_longitude": "-73.96623",  
  "dropoff_latitude": "40.689831",  
  "rate_code": "1",  
  "passenger_count": "1",  
  "trip_distance": "84.8",  
  "payment_type": "CAS",  
  "fare_amount": "0.0",  
  "extra": "0.0",  
  "mta_tax": "0.0",  
  "imp_surcharge": null,  
  "tip_amount": "0.0",  
  "tolls_amount": "0.0",  
  "total_amount": "0.0",  
  "store_and_fwd_flag": "0"  
}
```

Feature  
Engineering

Feature Vector

```
{  
  "fare_amount": "2.5",  
  "dayofweek": "1",  
  "pickuplon": "-73.937463",  
  "pickuplat": "40.758052",  
  "dropofflon": "-73.937477",  
  "dropofflat": "40.758062"
```

Raw data does not  
come as feature vectors

# Quiz: Logistic Regression

Are these features related to the objective or not?



# Predict the total number of customers who will use a certain discount coupon

01

Font of the text with which the discount is advertised on partner websites.

02

Price of the item the coupon applies to.

03

Number of items in stock.

# Predict whether a credit card transaction is fraudulent or not

01

Whether the cardholder has purchased these items at this store before.

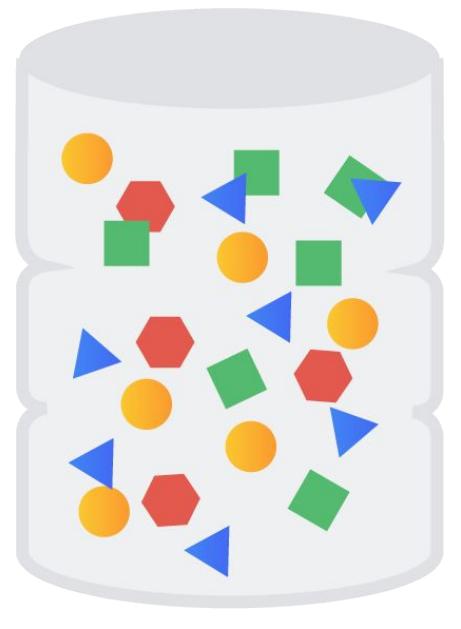
02

Category of the purchased item.

03

Expiry date of credit card.

**Prediction-time:**  
Some data could be known  
immediately, and some other  
data is **not known in real time**

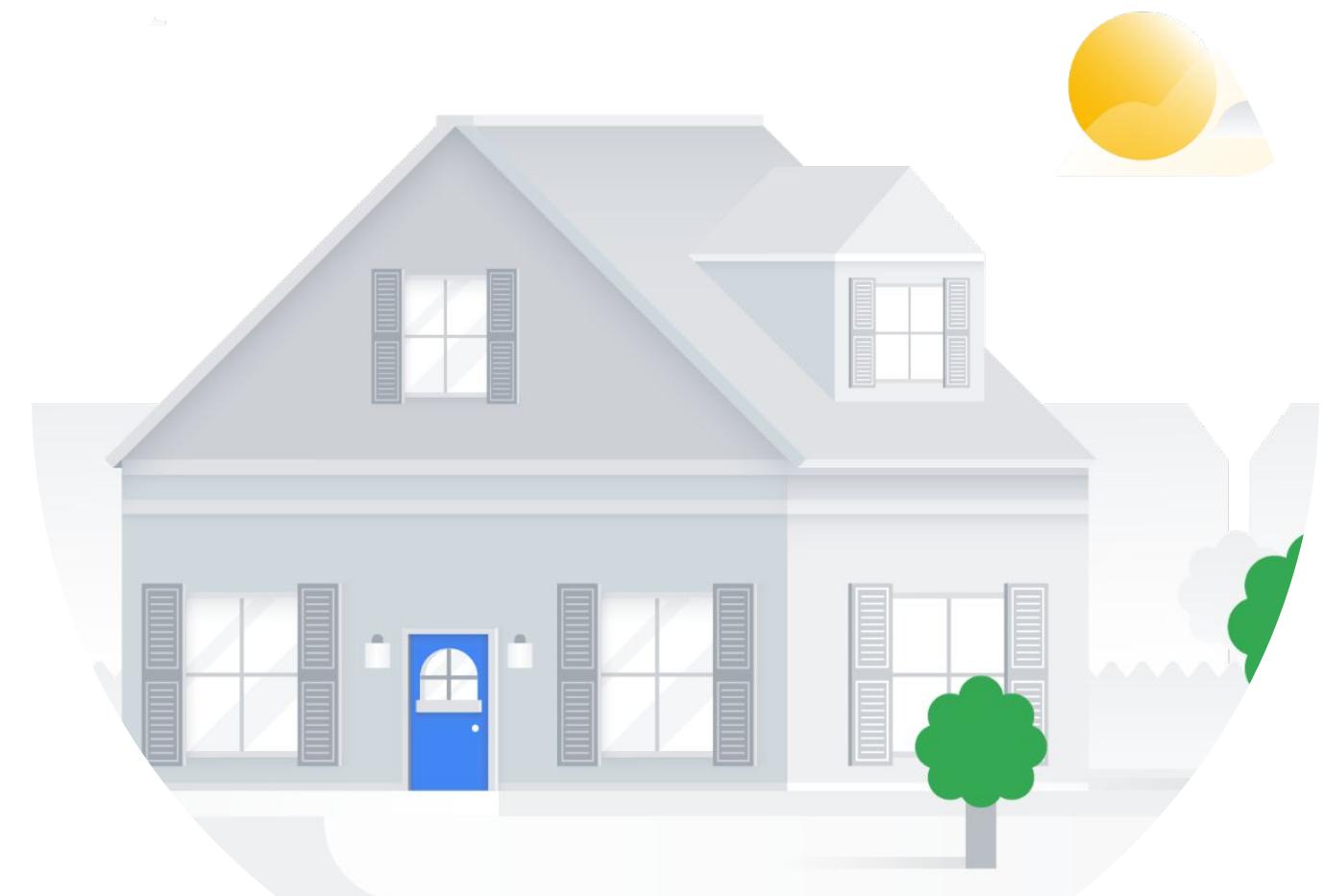


Sales Data



Today's sales  
transactions

You cannot feed  
your model what  
**you don't know** at  
prediction-time



Sold: \$300,000

# What's wrong with the second feature?



city\_id:"br/sao\_paulo"



inferred\_city\_cluster\_id:219

# Quiz: Logistic Regression

Is the value knowable at prediction time or not?



# Predict the total number of customers who will use a certain discount coupon

01

Total number of discountable items sold.

02

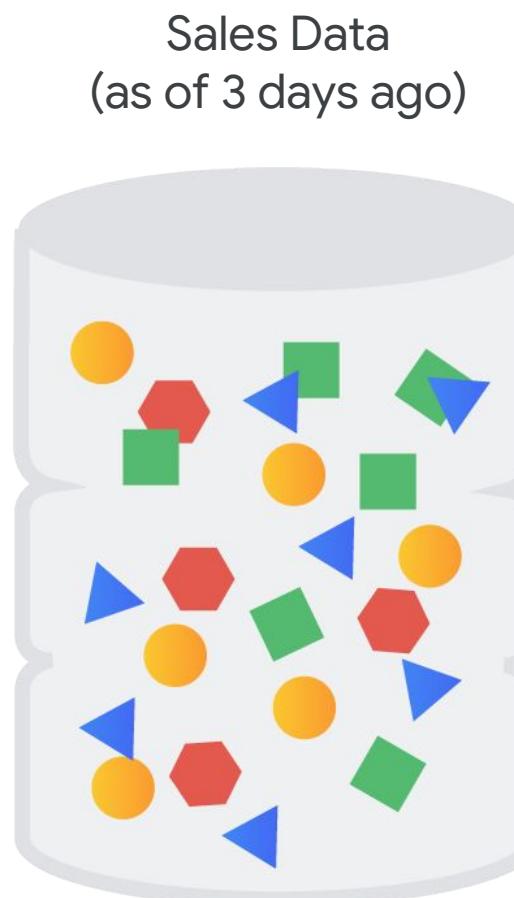
Number of discountable items sold the previous month.

03

Number of customers who viewed ads about item.

# You cannot train with current data and predict with stale data

If sales data at prediction time is only available with a three day lag, you should only train with sales data at least three days old



Sales Data  
(as of 3 days ago)

```
SELECT name, COUNT(trans_id) AS count
FROM sales_warehouse
WHERE
    # filter out last three days
    trans_time <
    TIMESTAMP_SUB(
        CURRENT_TIMESTAMP(), INTERVAL 3 DAY
    )
```

# Predict whether a credit card transaction is fraudulent or not

01

Whether the item is new at the store (and can not have been purchased before).

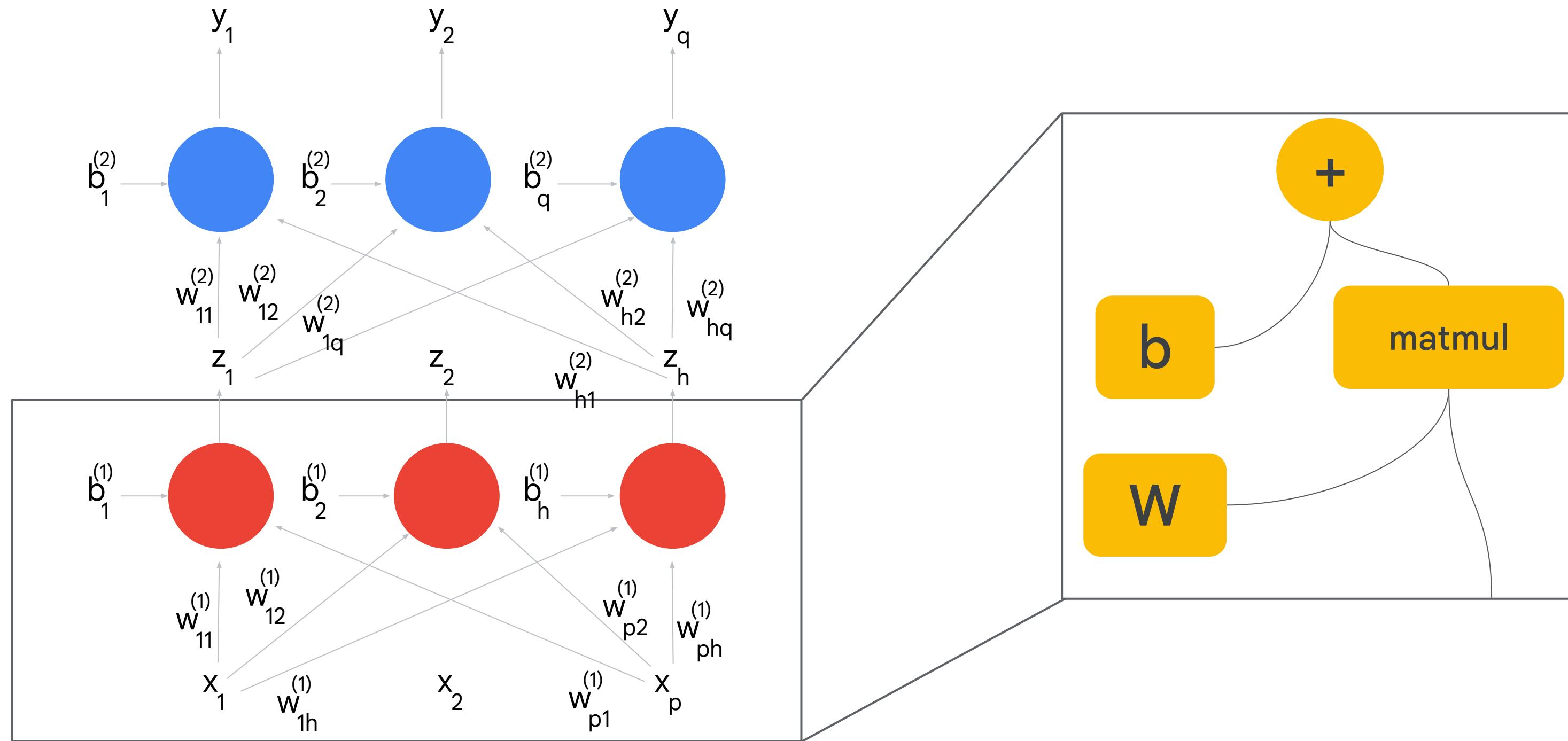
02

Category of item being purchased.

03

Online or in-person purchase?

# Neural networks are weighing and adding machines



# Quiz: Logistic Regression

Which of these are numeric with meaningful magnitude?



# Predict the total number of customers who will use a certain discount coupon

01

Percent value of the discount  
(e.g., 10% off, 20% off, etc.)

02

Size of the coupon  
(e.g., 4 cm<sup>2</sup>, 24 cm<sup>2</sup>, 48 cm<sup>2</sup>, etc.)

03

Font an advertisement is in  
(Arial, Times New Roman, etc.)

# Predict the total number of customers who will use a certain discount coupon

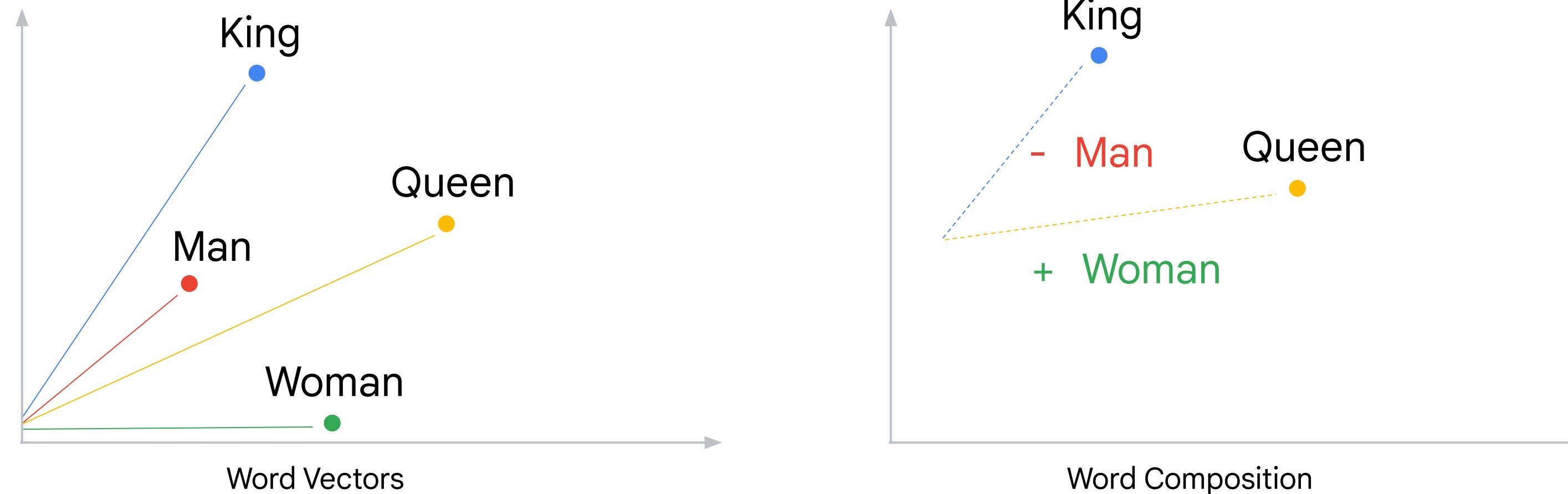
04

Color of coupon  
(red, black, blue,  
etc.)

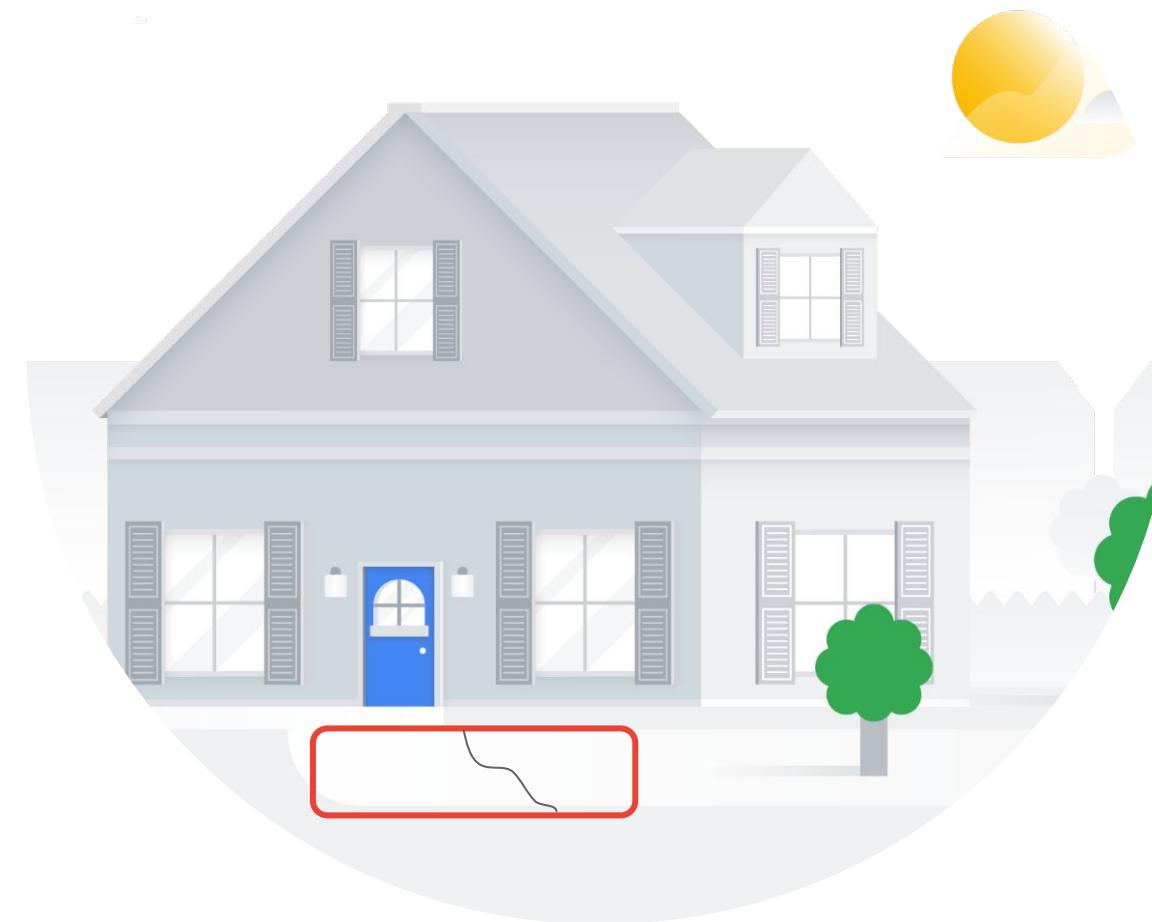
05

Item category (1 for  
dairy, 2 for deli, 3 for  
canned goods, etc.)

# Use Word2vec to make word vectors



# Have enough examples of feature value in the dataset

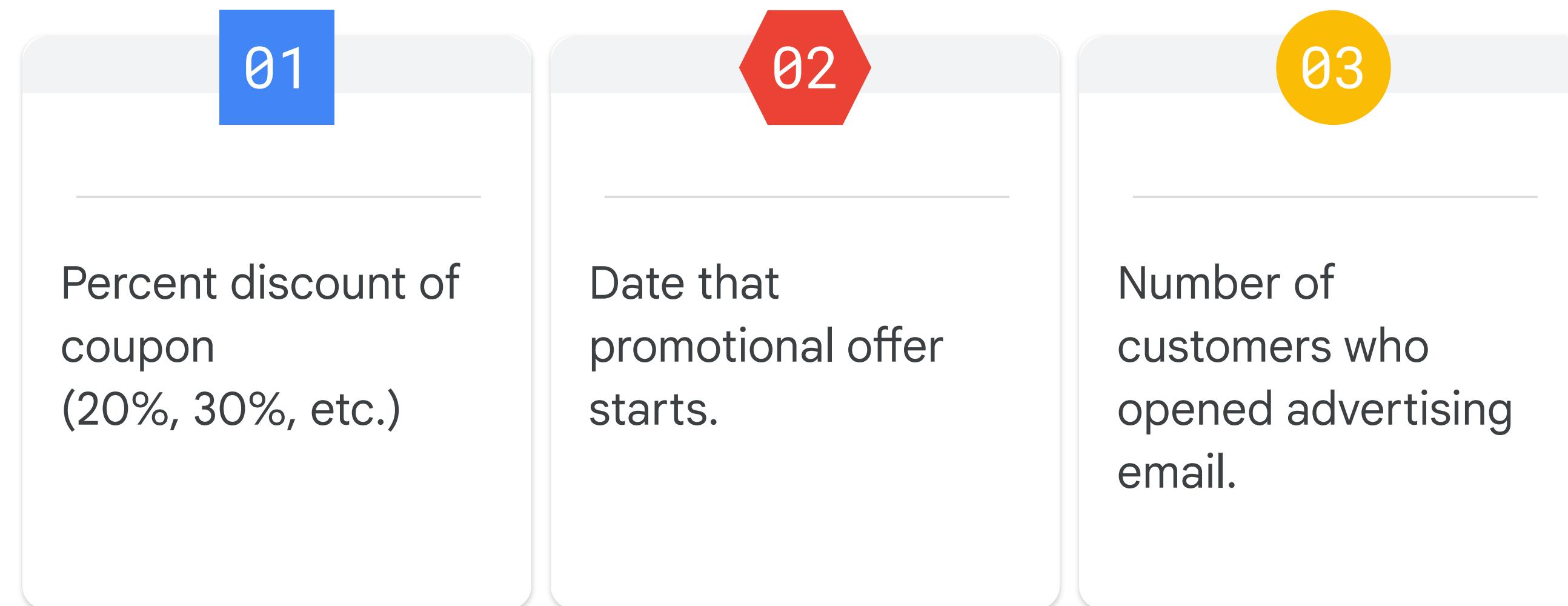


# Quiz: Logistic Regression

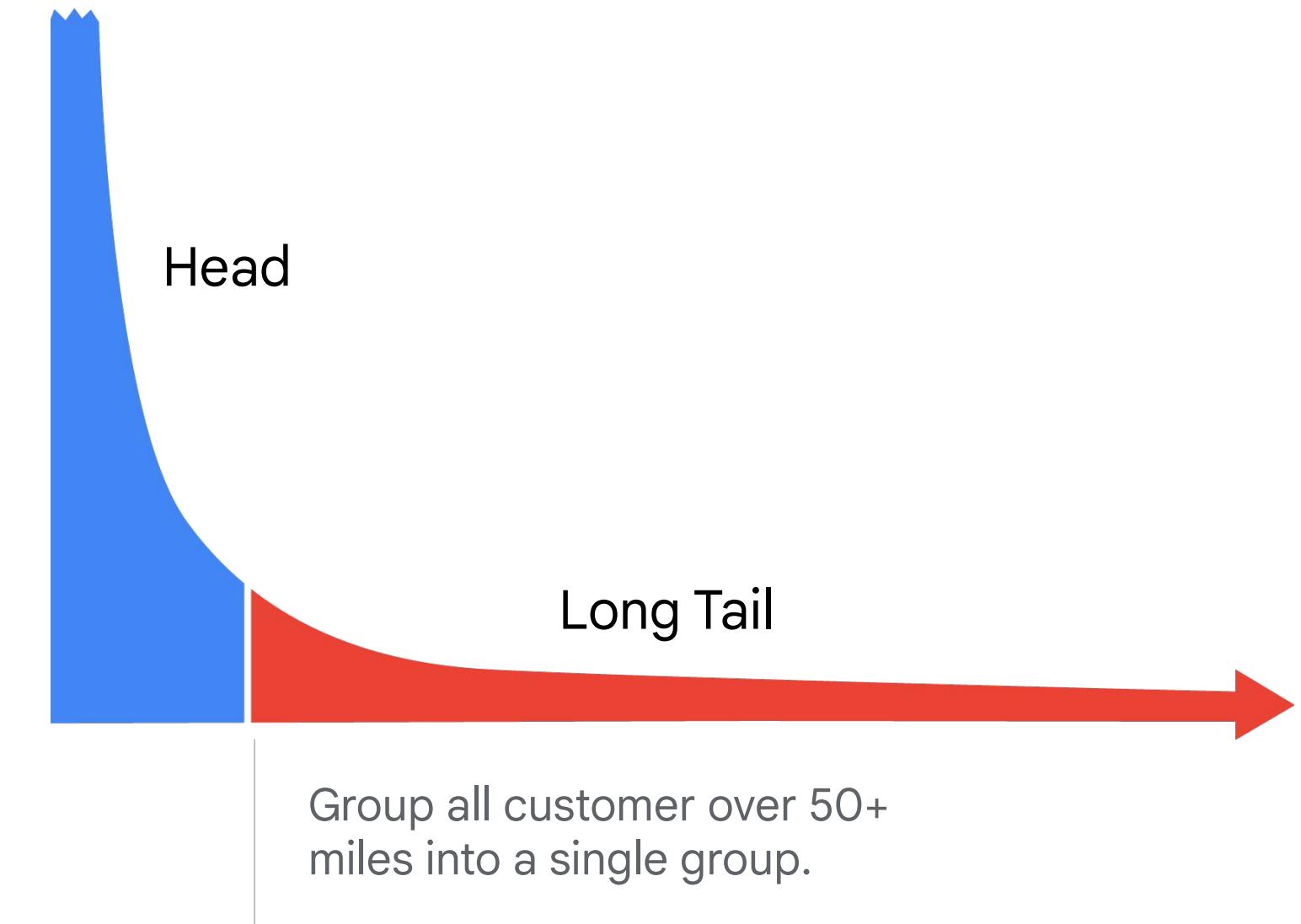
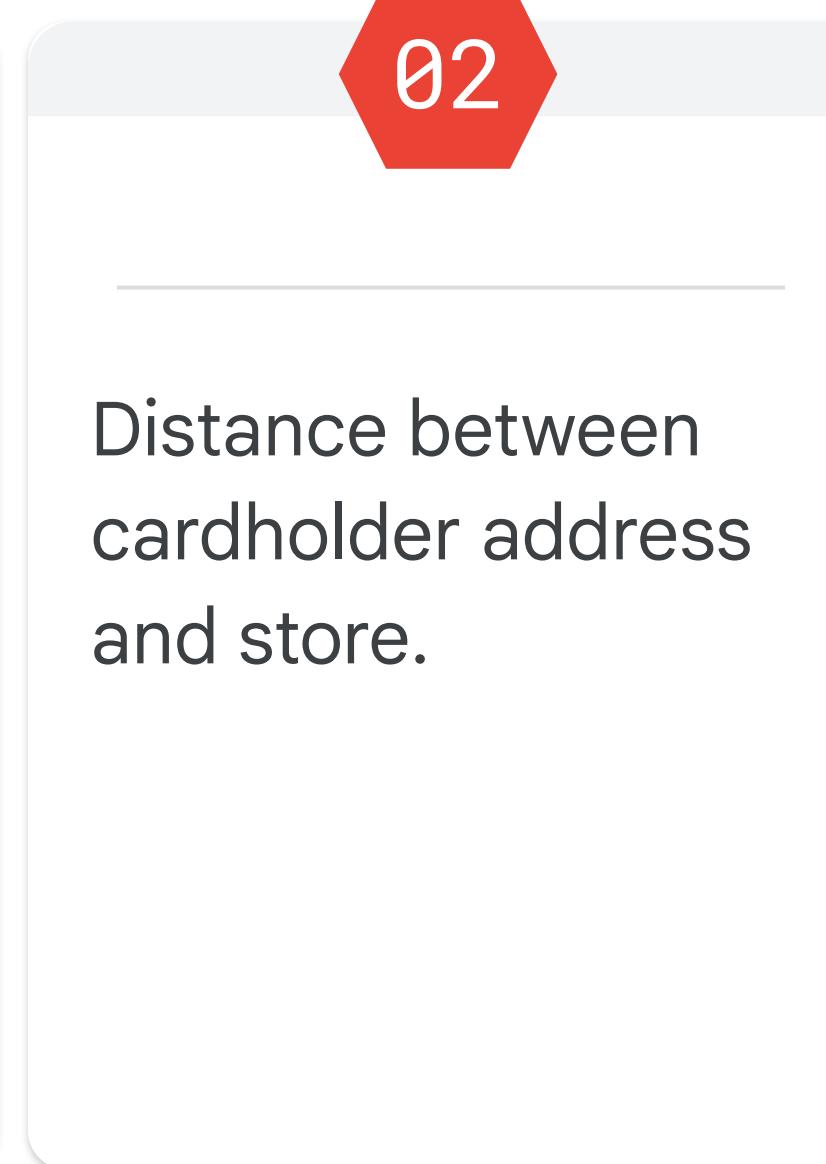
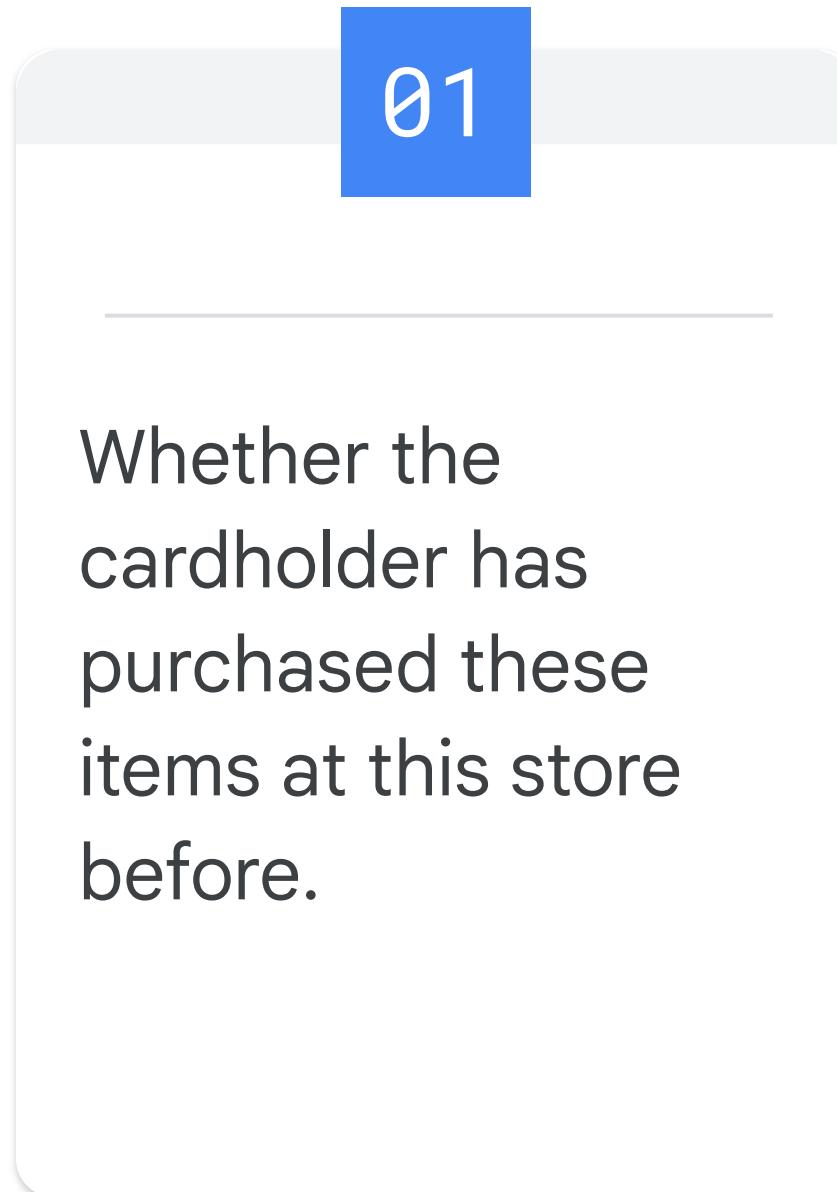
Which of these will it be difficult to get enough examples?



# Predict the total number of customers who will use a certain discount coupon



# Predict whether a credit card transaction is fraudulent



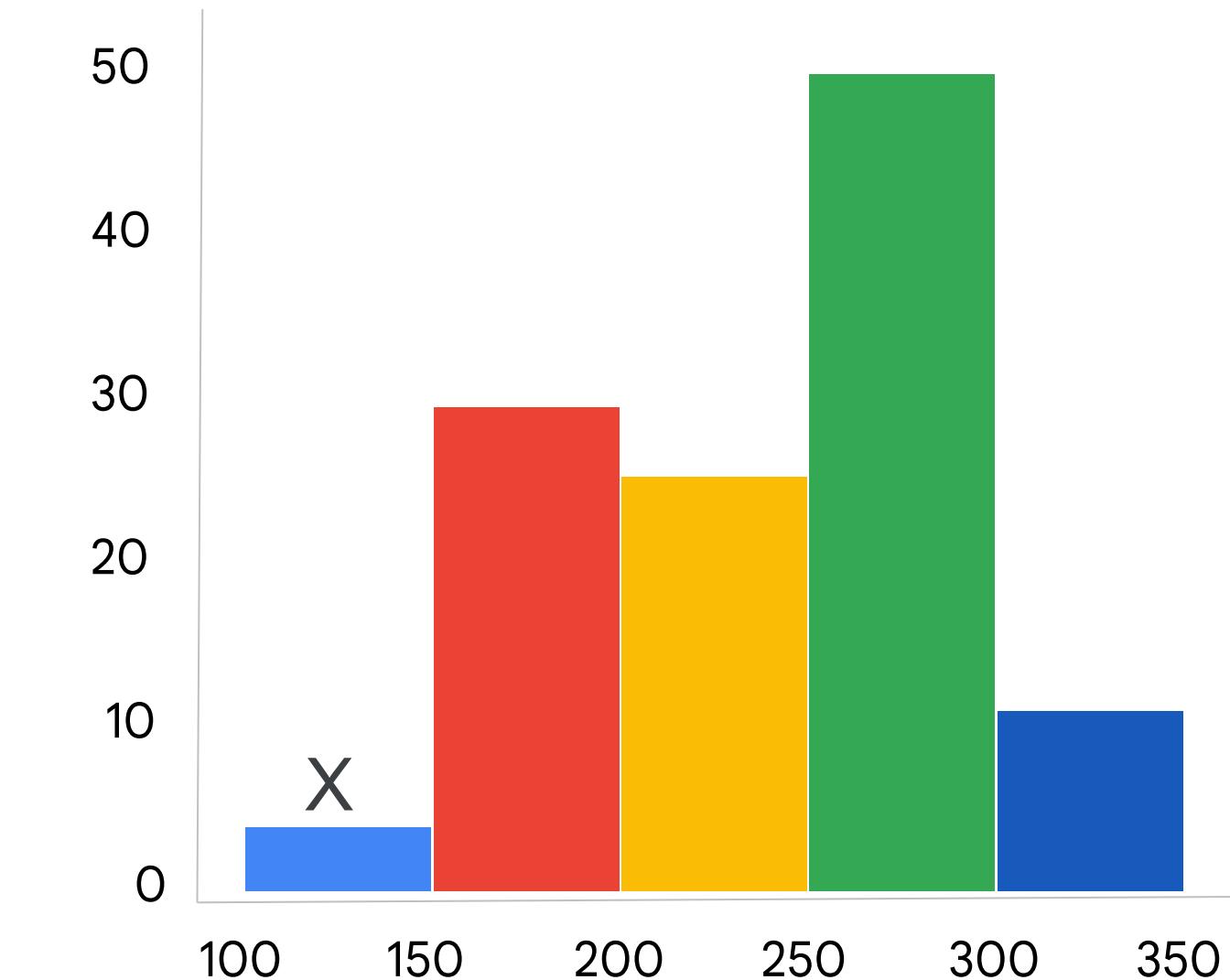
# Predict whether a credit card transaction is fraudulent

01

Whether the cardholder has purchased these items at this store before.

02

Distance between cardholder address and store.



# Predict whether a credit card transaction is fraudulent

01

Whether the cardholder has purchased these items at this store before.

02

Distance between cardholder address and store.

03

Category of item being purchased.

04

Online or in-person purchase?

# Bring human insight to problem

You need to have subject matter expertise and a curious mind to think of all of the ways a data field could be used as a feature.

Remember that feature engineering is not done in a vacuum. After you train your first model you can always come back and add or remove features for model two.



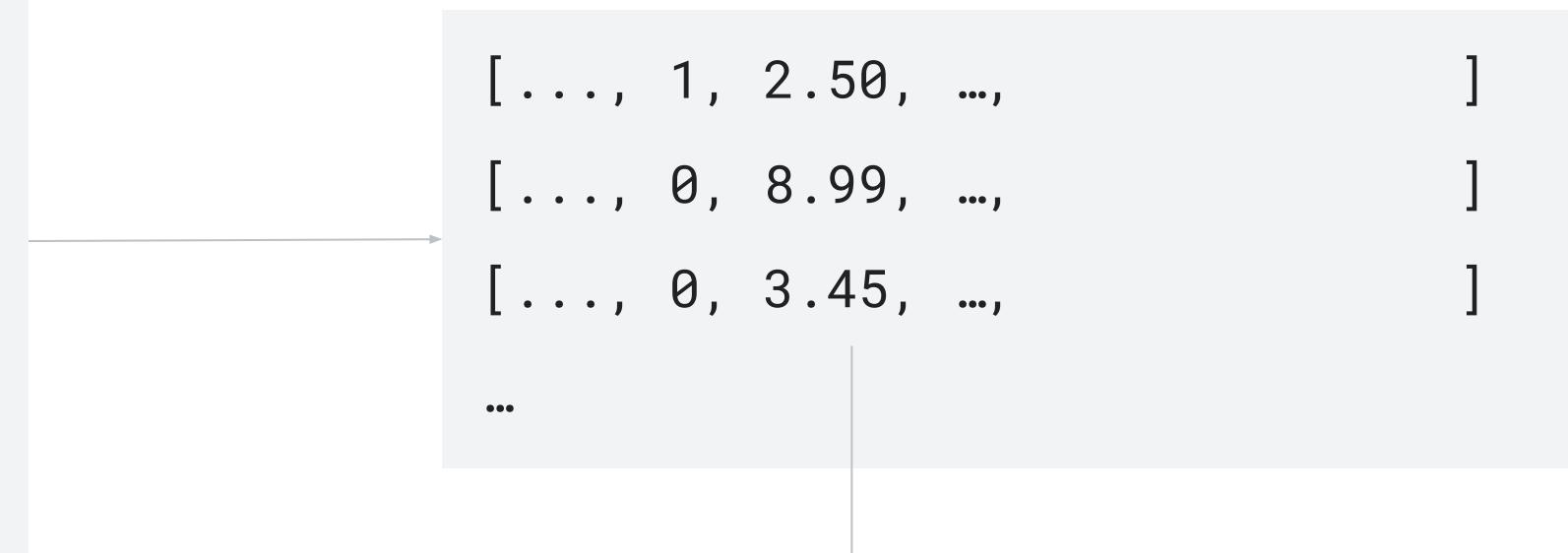
# Represent features

# Raw data are converted to numeric features in different ways

```
{  
  "transactionId": 42,  
  "name": "Ice Cream",  
  "price": 2.50,  
  "tags": ["cold", "dessert"],  
  "servedBy": {  
    "employeeId": 72365,  
    "waitTime": 1.4,  
    "customerRating": 4},  
  "storeLocation": {  
    "latitude": 35.3,  
    "longitude": -98.7}  
},
```

# Raw data are converted to numeric features in different ways

```
{  
  "transactionId": 42,  
  "name": "Ice Cream",  
  "price": 2.50,  
  "tags": ["cold", "dessert"],  
  "servedBy": {  
    "employeeId": 72365,  
    "waitTime": 1.4,  
    "customerRating": 4},  
  "storeLocation": {  
    "latitude": 35.3,  
    "longitude": -98.7}  
},
```



In estimator API,  
this is a feature column.

# Numeric values can be used as-is

```
{  
  "transactionId": 42,  
  "name": "Ice Cream",  
  "price": 2.50,  
  "tags": ["cold", "dessert"],  
  "servedBy": {  
    "employeeId": 72365,  
    "waitTime": 1.4,  
    "customerRating": 4},  
  "storeLocation": {  
    "latitude": 35.3,  
    "longitude": -98.7}  
},
```

→ [ ... , **2.50**, ... , **1.4** , ... ]  
...  
...

```
INPUT_COLUMNS = [  
  ...,  
  tf.feature_column.numeric_column('price'),  
  ...  
]
```

numeric\_column is a type of  
feature column.

# Overly specific attributes should be discarded

```
{  
  "transactionId": 42,  
  "name": "Ice Cream",  
  "price": 2.50,  
  "tags": ["cold", "dessert"],  
  "servedBy": {  
    "employeeId": 72365,  
    "waitTime": 1.4,  
    "customerRating": 4},  
  "storeLocation": {  
    "latitude": 35.3,  
    "longitude": -98.7}  
},
```

# Overly specific attributes should be discarded

```
{  
  "transactionId": 42,  
  "name": "Ice Cream",  
  "price": 2.50,  
  "tags": ["cold", "dessert"],  
  "servedBy": {  
    "employeeId": 72365,  
    "waitTime": 1.4,  
    "customerRating": 4},  
  "storeLocation": {  
    "latitude": 35.3,  
    "longitude": -98.7}  
},
```



# Overly specific attributes should be discarded

```
{  
  "transactionId": 42,  
  "name": "Ice Cream",  
  "price": 2.50,  
  "tags": ["cold", "dessert"],  
  "servedBy": {  
    "employeeId": 72365,  
    "waitTime": 1.4,  
    "customerRating": 4},  
  "storeLocation": {  
    "latitude": 35.3,  
    "longitude": -98.7}  
},
```



# Overly specific attributes should be discarded

```
{  
  "transactionId": 42,  
  "name": "Ice Cream",  
  "price": 2.50,  
  "tags": ["cold", "dessert"],  
  "servedBy": {  
    "employeeId": 72365,  
    "waitTime": 1.4,  
    "customerRating": 4},  
  "storeLocation": {  
    "latitude": 35.3,  
    "longitude": -98.7}  
},
```



# Categorical variables should be one-hot encoded

```
{  
  "transactionId": 42,  
  "name": "Ice Cream",  
  "price": 2.50,  
  "tags": ["cold", "dessert"],  
  "servedBy": {  
    "employeeId": 72365,  
    "waitTime": 1.4,  
    "customerRating": 4},  
  "storeLocation": {  
    "latitude": 35.3,  
    "longitude": -98.7}  
}
```



8345	72365	87654	98723	23451
0	1	0	0	0

# Categorical variables should be one-hot encoded

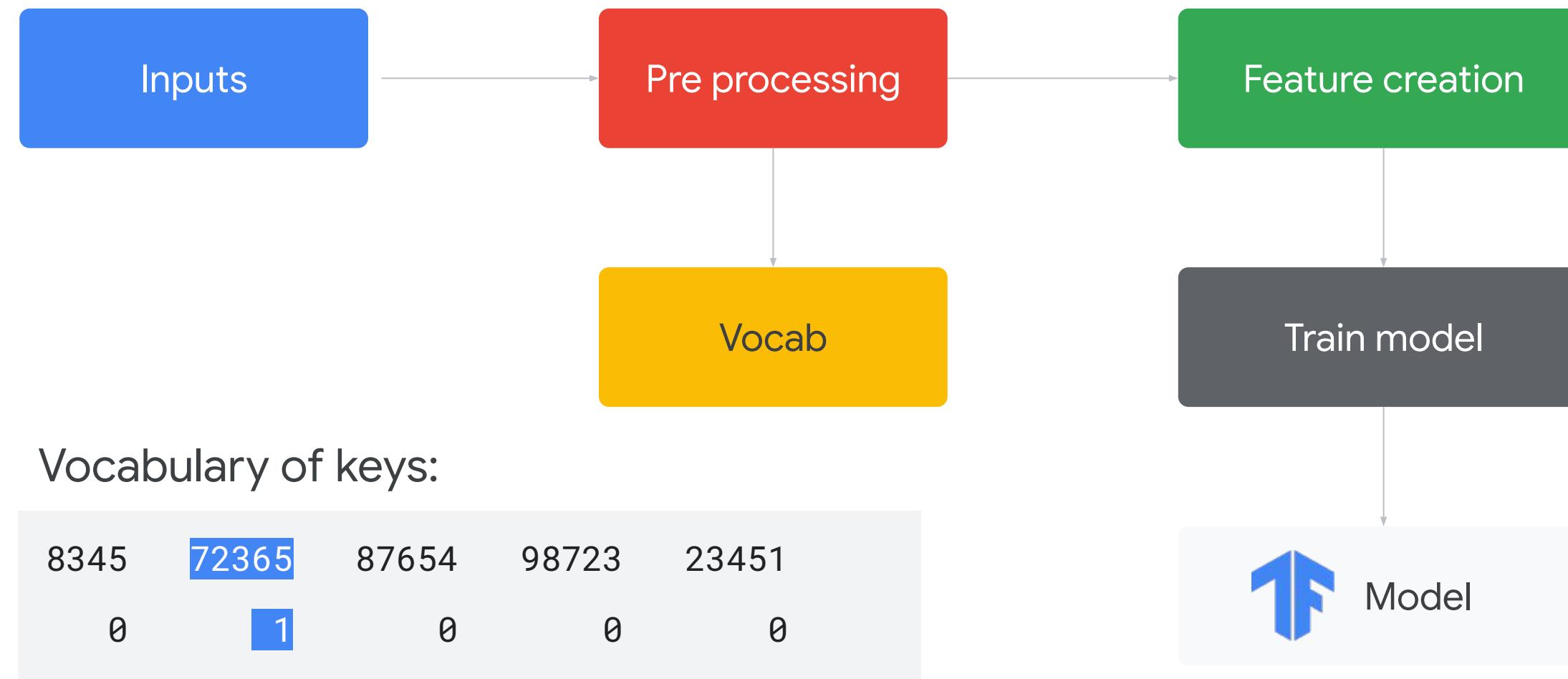
```
{  
    "transactionId": 42,  
    "name": "Ice Cream",  
    "price": 2.50,  
    "tags": ["cold", "dessert"],  
    "servedBy": {  
        "employeeId": 72365,  
        "waitTime": 1.4,  
        "customerRating": 4},  
    "storeLocation": {  
        "latitude": 35.3,  
        "longitude": -98.7}  
}
```



8345	72365	87654	98723	23451
0	1	0	0	0

```
tf.feature_column.categorical_column_with_vocabulary_list(  
    'employeeId',  
    Vocabulary_list = ['8345', '72365', '87654', '98723', '23451']),
```

# Preprocess data to create a vocabulary of keys



The vocabulary and  
the mapping of the  
vocabulary needs to  
be identical at  
**prediction time**

8345	72365	87654	98723	??????
0	0	0	0	0

# Options for encoding categorical data

If you know the keys beforehand:

```
tf.feature_column.categorical_column_with_vocabulary_list('employeeId',  
    vocabulary_list = [ '8345' , '72345' , '87654' , '98723' , '23451' ]),
```

# Options for encoding categorical data

If you know the keys beforehand:

```
tf.feature_column.categorical_column_with_vocabulary_list('employeeId',  
    vocabulary_list = [ '8345' , '72345' , '87654' , '98723' , '23451' ]),
```

If your data is already indexed; i.e., has integers in [0-N):

```
tf.feature_column.categorical_column_with_identity('employeeId',  
    num_buckets = 5)
```

# Options for encoding categorical data

If you know the keys beforehand:

```
tf.feature_column.categorical_column_with_vocabulary_list('employeeId',  
    vocabulary_list = [ '8345' , '72345' , '87654' , '98723' , '23451' ]),
```

If your data is already indexed; i.e., has integers in [0-N):

```
tf.feature_column.categorical_column_with_identity('employeeId',  
    num_buckets = 5)
```

If you don't have a vocabulary of all possible values:

```
tf.feature_column.categorical_column_with_hash_bucket('employeeId',  
    hash_bucket_size = 500)
```

# Categorical variables should be one-hot encoded

```
{  
    "transactionId": 42,  
    "name": "Ice Cream",  
    "price": 2.50,  
    "tags": ["cold", "dessert"],  
    "servedBy": {  
        "employeeId": 72365,  
        "waitTime": 1.4,  
        "customerRating    "storeLocation": {  
        "latitude": 35.3,  
        "longitude": -98.7}  
},
```

# Categorical variables should be one-hot encoded

```
{  
  "transactionId": 42,  
  "name": "Ice Cream",  
  "price": 2.50,  
  "tags": ["cold", "dessert"],  
  "servedBy": {  
    "employeeId": 72365,  
    "waitTime": 1.4,  
    "customerRating": 4",  
  "storeLocation": {  
    "latitude": 35.3,  
    "longitude": -98.7}  
},
```

[ ..., 4, ...]

(OR)

[ ..., 0,0,0,**1**,0, ...]

# Don't mix magic numbers with data

```
{  
  "transactionId": 42,  
  "name": "Ice Cream",  
  "price": 2.50,  
  "tags": ["cold", "dessert"],  
  "servedBy": {  
    "employeeId": 72365,  
    "waitTime": 1.4,  
    "customerRating": -1},  
  "storeLocation": {  
    "latitude": 35.3,  
    "longitude": -98.7}  
},
```

# Don't mix magic numbers with data

```
{  
  "transactionId": 42,  
  "name": "Ice Cream",  
  "price": 2.50,  
  "tags": ["cold", "dessert"],  
  "servedBy": {  
    "employeeId": 72365,  
    "waitTime": 1.4,  
    "customerRating": -1},  
  "storeLocation": {  
    "latitude": 35.3,  
    "longitude": -98.7}  
},
```

```
[ . . . , 4, 1, ...] # 4  
[ . . . , 0, 0, ...] # -1
```

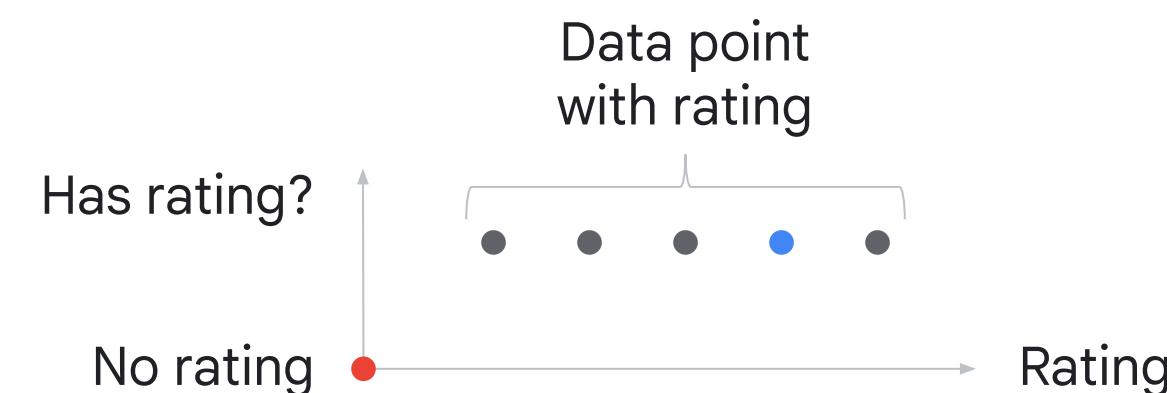
# Don't mix magic numbers with data

```
{  
  "transactionId": 42,  
  "name": "Ice Cream",  
  "price": 2.50,  
  "tags": ["cold", "dessert"],  
  "servedBy": {  
    "employeeId": 72365,  
    "waitTime": 1.4,  
    "customerRating": -1},  
  "storeLocation": {  
    "latitude": 35.3,  
    "longitude": -98.7}  
},
```

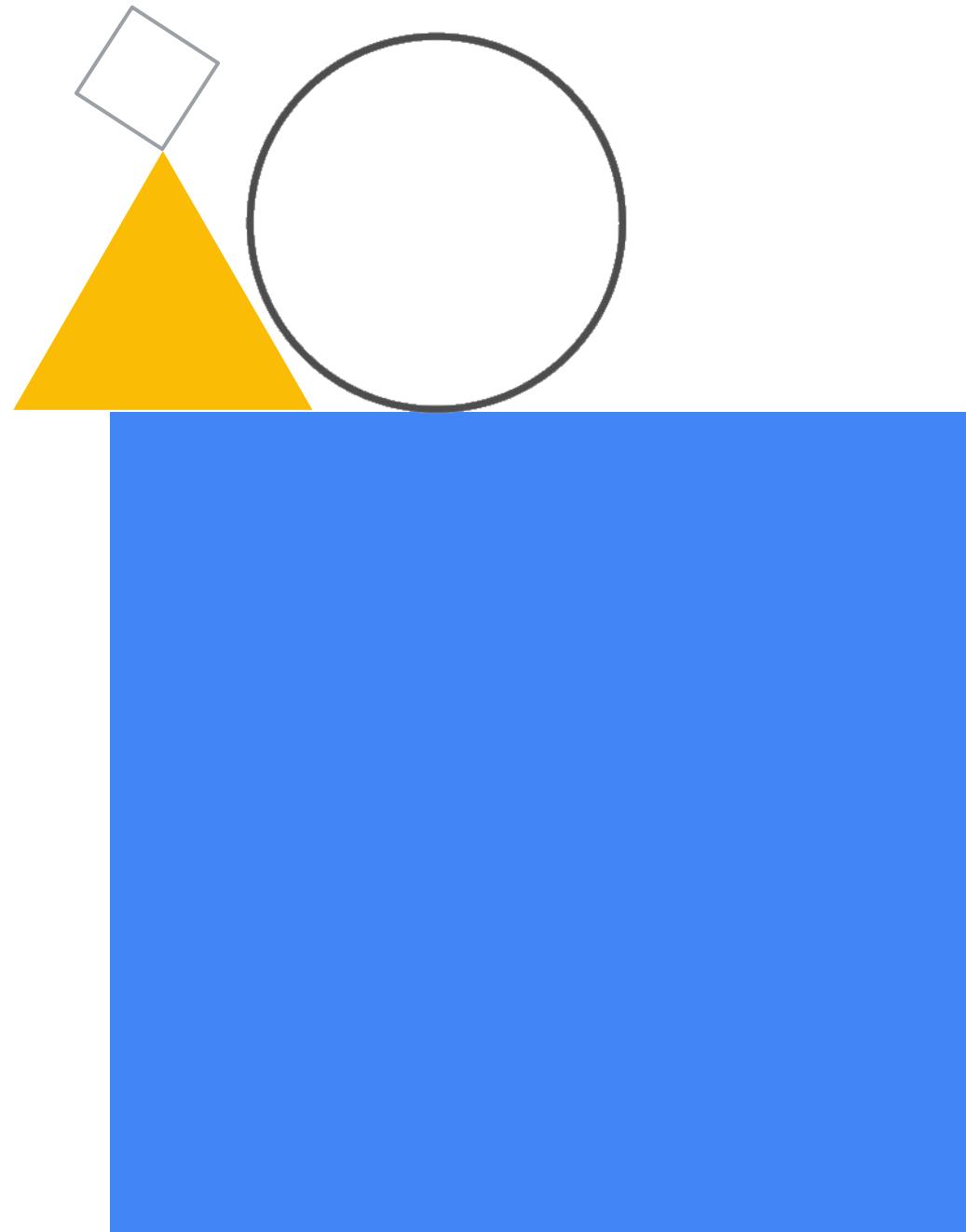
```
[..., 4, 1, ...] # 4  
[..., 0, 0, ...] # -1
```

(OR)

```
[..., 0, 0, 0, 1, 1, ...] # 4  
[..., 0, 0, 0, 0, 0, ...] # -1
```



# Feature Engineering

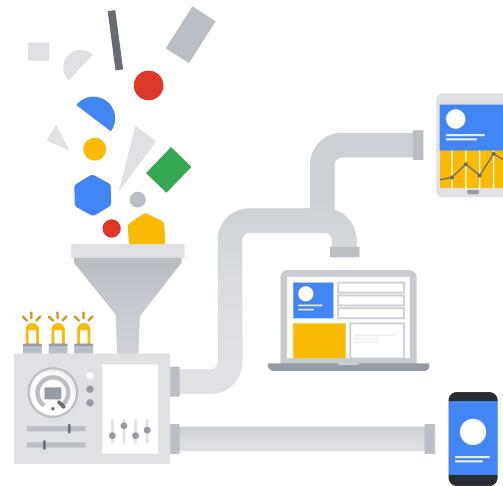


# In this module, you learn to ...

- 01 Distinguish machine learning from statistics
- 02 Perform feature engineering using BigQuery ML
- 03 Perform feature engineering using Keras



# Machine Learning versus Statistics



## Machine Learning

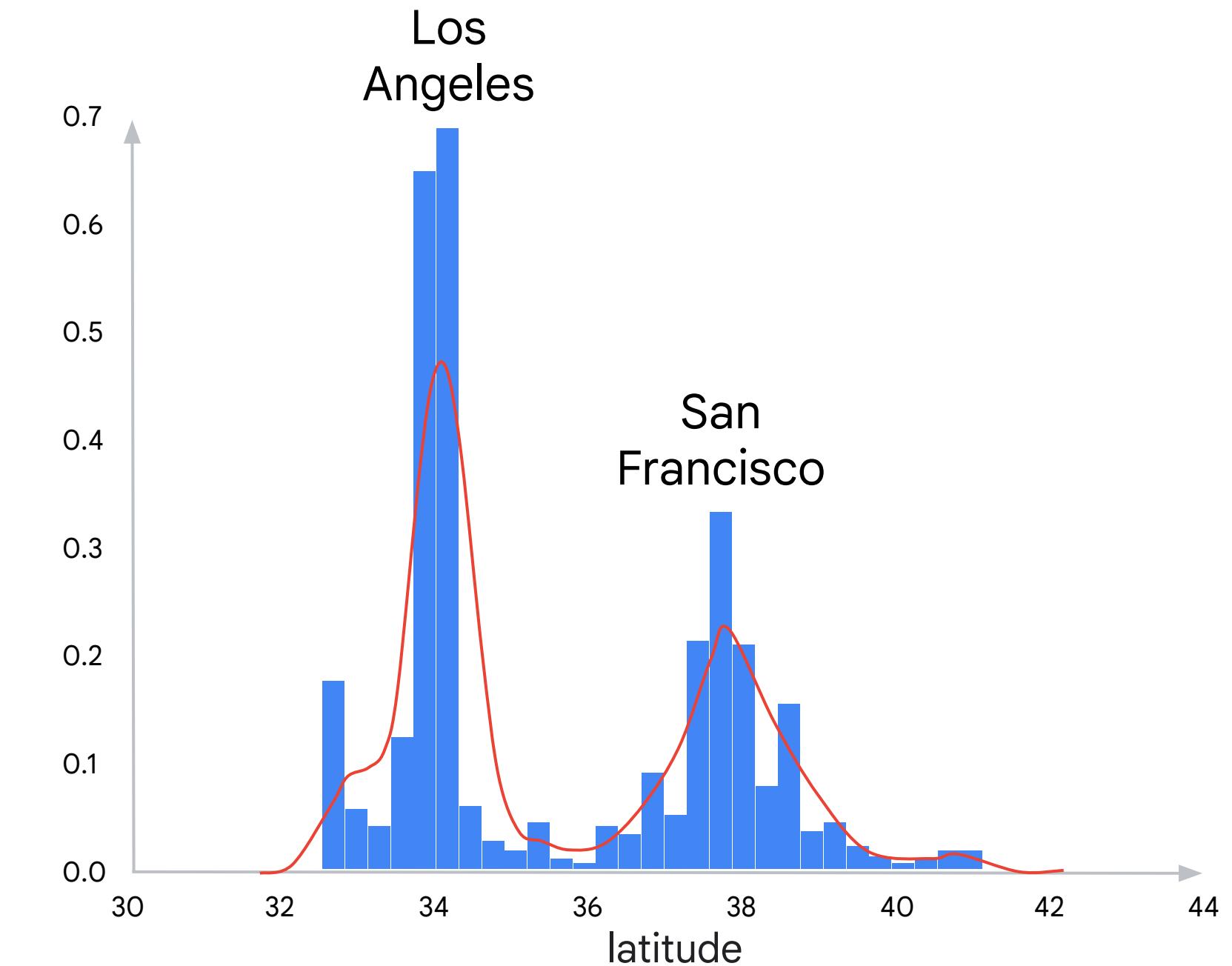
Lots of data, keep outliers  
and build models for them.



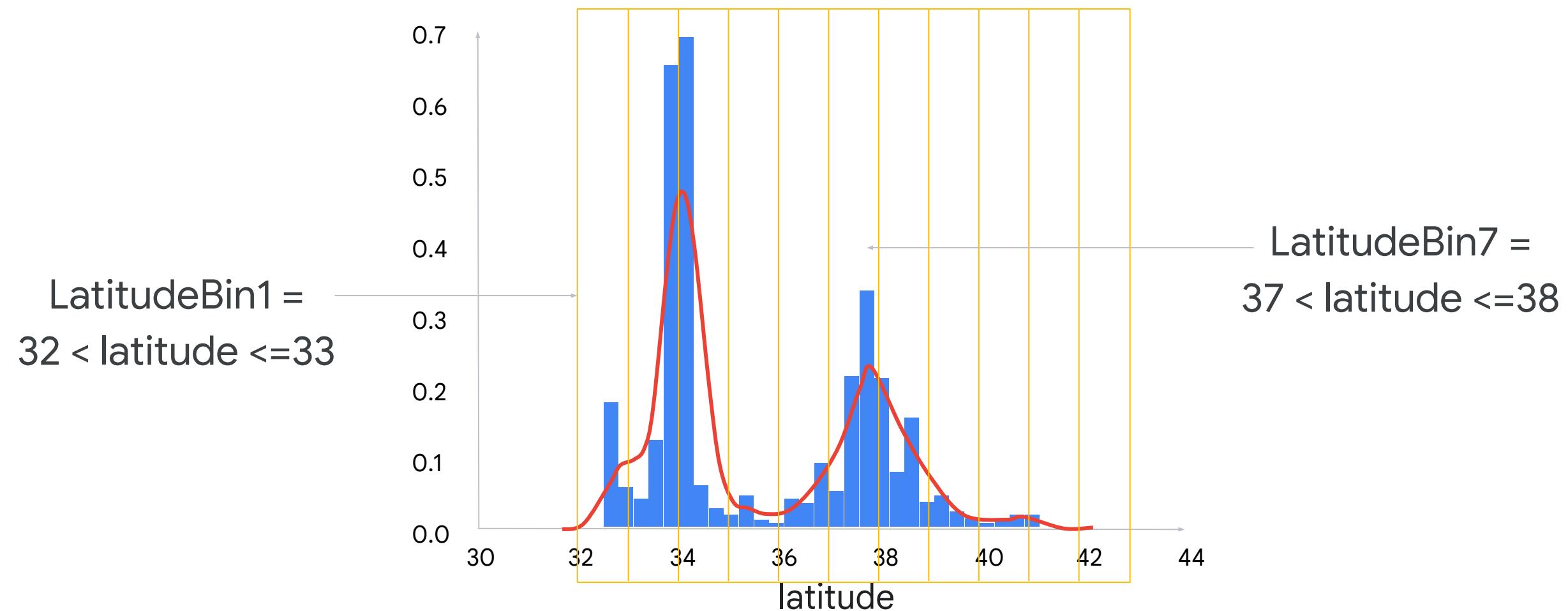
## Statistics

"I've got all the data I'll  
ever get," throw away  
outliers.

**Exact floats are  
not meaningful**

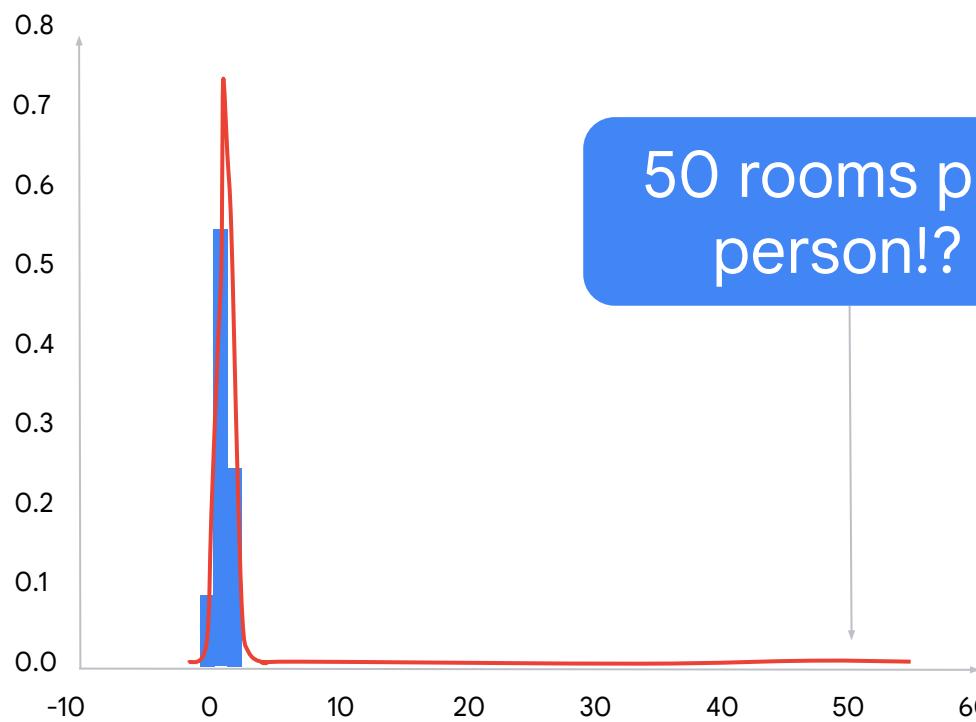


# Discretize floating point values into bins



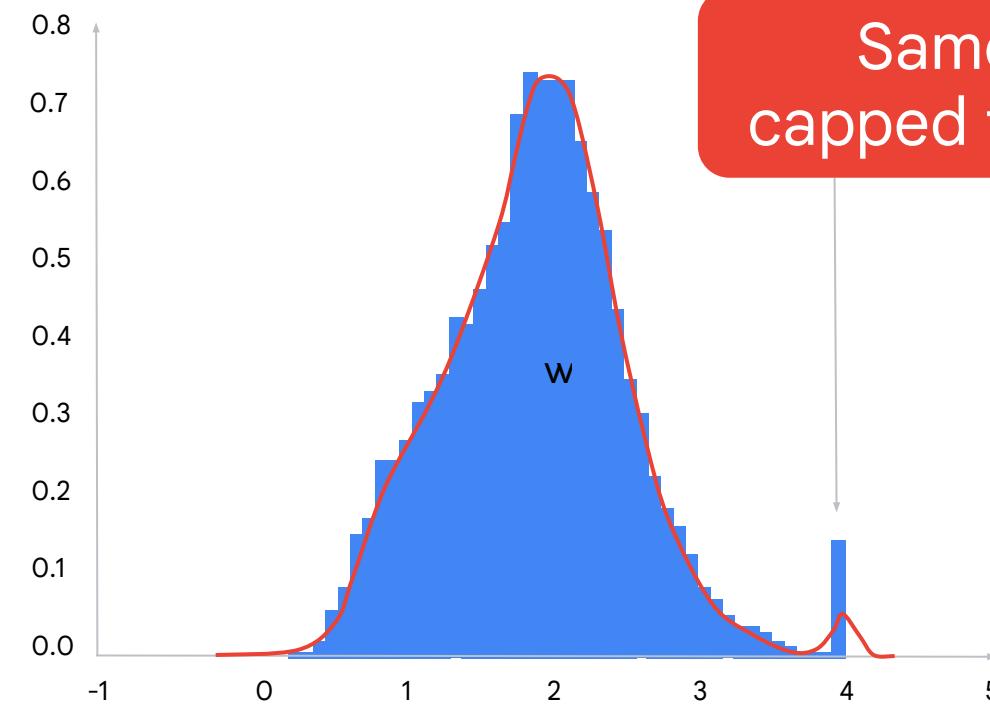
```
lat = tf.feature_column.numeric_column('latitude')
dlat = tf.feature_column.bucketized_column(
    lat, boundaries=np.arange(32,42,1).tolist())
```

# Crazy outliers will hurt trainability



Rooms Per Person

50 rooms per person!?



Capped Rooms Per Person

Same feature,  
capped to max of 4.0

```
features['capped_rooms'] = tf.clip_by_value(  
    features['rooms'] ,  
    clip_value_min=0,  
    clip_value_max=4  
)
```

**Ideally, features should have a similar range**

Typically [0, 1] or [-1, 1]

```
features['scaled_price'] =  
    (features['price'] - min_price) /  
    (max_price - min_price)
```

# BigQuery preprocessing - Feature Engineering

## Representation transformation

---

- Convert a numeric feature to a categorical feature (bucketization)
- Convert categorical features to a numeric representation (one-hot encoding, learning with counts, sparse feature embeddings, etc.)
- Some models work with numeric or categorical features
- Other models handle mixed type features

## Feature construction

---

- Create new features either by using typical techniques (polynomial expansion by using univariate mathematical functions, or feature crossing to capture feature interactions)
- Can also be constructed by using business logic from the domain of the ML use case

# BigQuery ML supports two types of feature preprocessing

## Automatic preprocessing

---

BigQuery ML performs automatic preprocessing during training.

## Manual preprocessing

---

BigQuery ML provides the [TRANSFORM](#) clause for you to define custom preprocessing using the [manual preprocessing functions](#). You can also use these functions outside the TRANSFORM clause.

# Basic feature engineering

Example of preprocessing in BigQuery

```
SELECT
    (tolls_amount + fare_amount)
        AS fare_amount,
    DAYOFWEEK(pickup_datetime)
        AS dayofweek,
    HOUR(pickup_datetime)
        AS hourofday,
    ...
FROM
    `nyc-tlc.yellow.trips`
WHERE
    trip_distance > 0
```

# Basic feature engineering

Example of preprocessing in BigQuery

```
SELECT  
    (tolls_amount + fare_amount)  
        AS fare_amount,  
    DAYOFWEEK(pickup_datetime)  
        AS dayofweek,  
    HOUR(pickup_datetime)  
        AS hourofday,  
    ...  
FROM  
    `nyc-tlc.yellow.trips`  
WHERE  
    trip_distance > 0
```

Built-in SQL math and data processing functions

# Basic feature engineering

Example of preprocessing in BigQuery

```
SELECT  
    (tolls_amount + fare_amount)  
        AS fare_amount,  
    DAYOFWEEK(pickup_datetime)  
        AS dayofweek,  
    HOUR(pickup_datetime)  
        AS hourofday,  
    ...  
FROM  
    `nyc-tlc.yellow.trips`  
WHERE  
    trip_distance > 0
```

Built-in SQL math and data processing functions

Data processing functions

# Basic feature engineering

Example of preprocessing in BigQuery

```
SELECT  
    (tolls_amount + fare_amount)  
        AS fare_amount,  
    DAYOFWEEK(pickup_datetime)  
        AS dayofweek,  
    HOUR(pickup_datetime)  
        AS hourofday,  
    ...  
FROM  
    `nyc-tlc.yellow.trips`  
WHERE  
    trip_distance > 0
```

Built-in SQL math and data processing functions

Data processing functions

Specify SQL filtering operations

# Basic feature engineering

Example of dates and time

```
EXTRACT(DAYOFWEEK FROM pickup_datetime) AS dayofweek,  
EXTRACT(HOUR FROM pickup_datetime) AS hourofday,  
  
CONCAT(CAST(EXTRACT(DAYOFWEEK FROM pickup_datetime) AS  
STRING),  
CAST(EXTRACT(HOUR FROM pickup_datetime) AS STRING)) AS  
hourofday,
```

Data studio output: From EXTRACT dates/time queries

	dayofweek	hourofday
1.	Week 6	4 AM
2.	Week 5	3 AM
3.	Week 4	2 AM
4.	Week 7	1 AM
5.	Week 3	12 AM

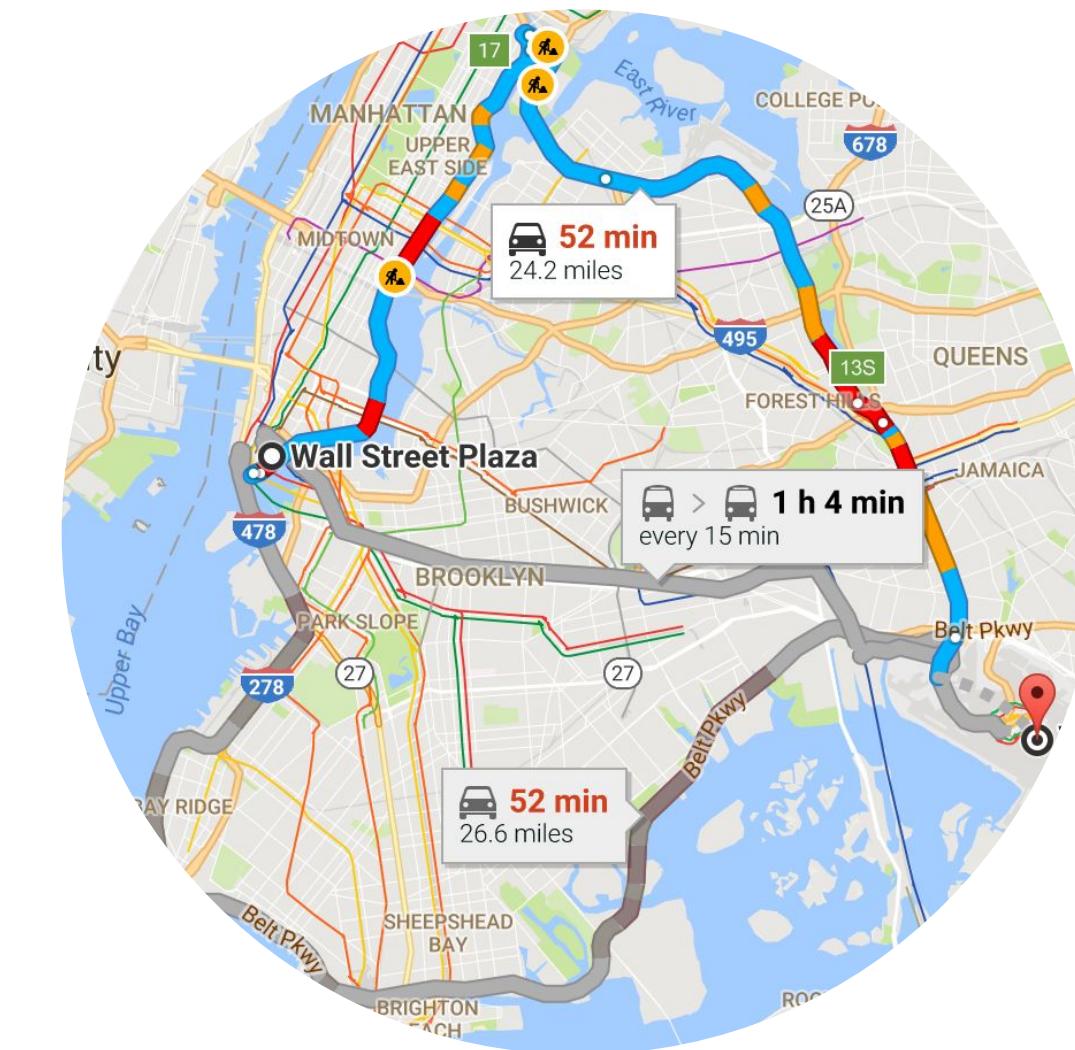
## Basic feature engineering

### Example of dates and time

**Note:** For all non-numeric columns other than TIMESTAMP, BigQuery ML performs a one-hot encoding transformation that generates a separate feature for each unique value in the column.

# BigQuery ML labs

Feature engineering in BigQuery ML

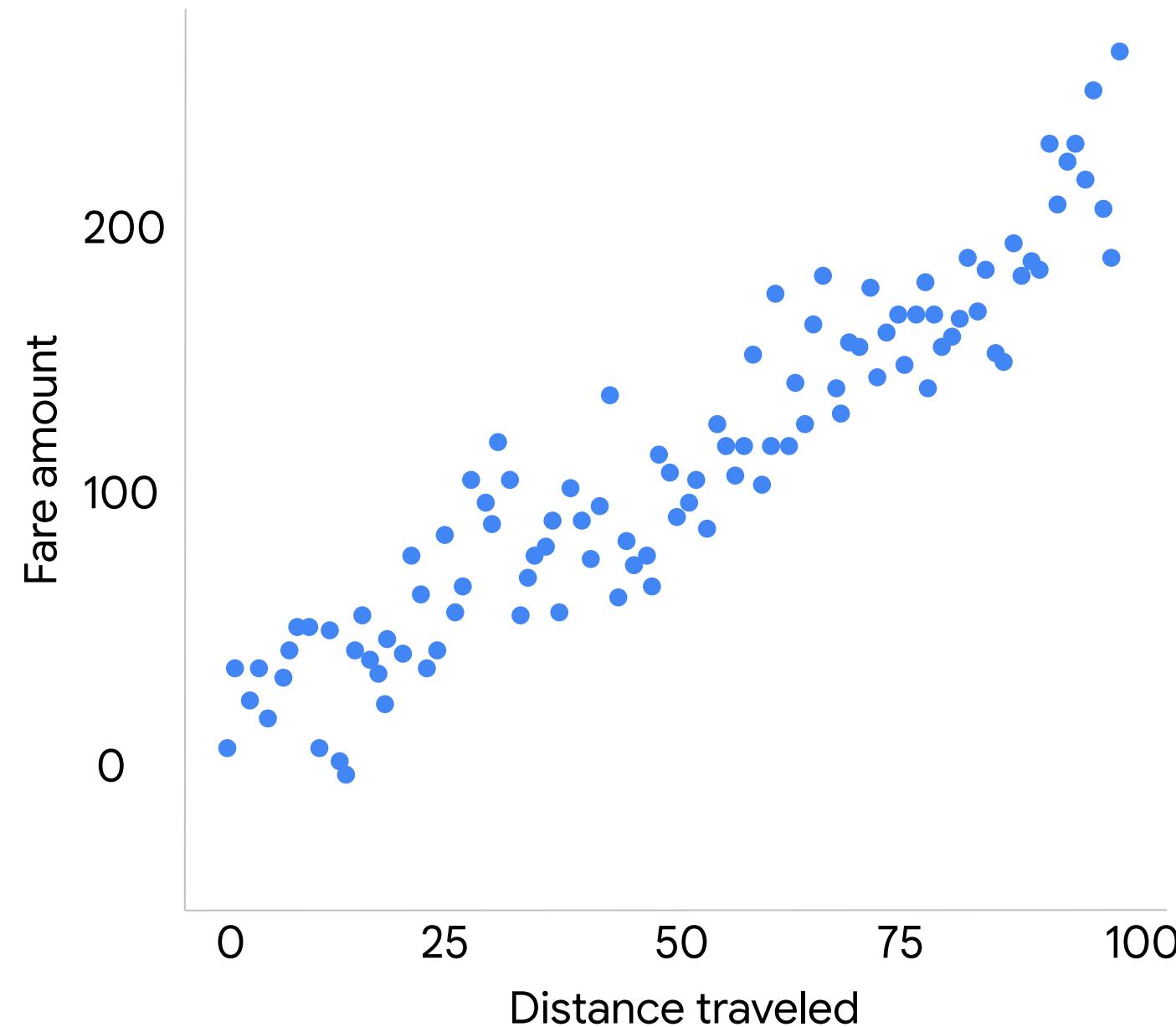


# Regression problem: Predicting taxi fare

Problem:

Predict taxi fare amount  
based on distance travelled

What is the error measure to  
optimize? RMSE



# ML problem: Estimating taxi fare

Taxi fare:

Fare amount

+

Distance travelled

+

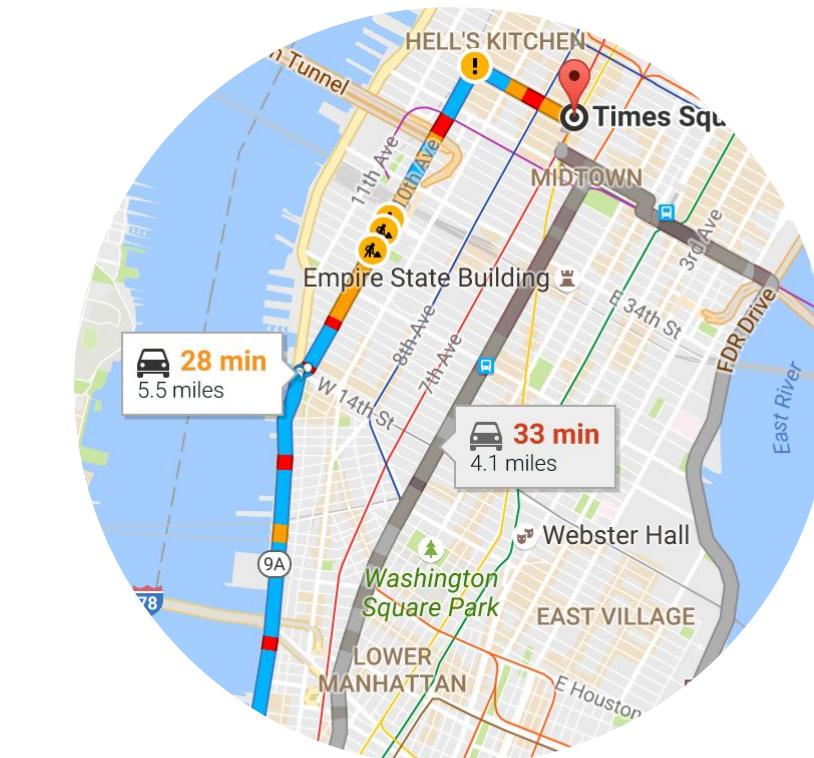
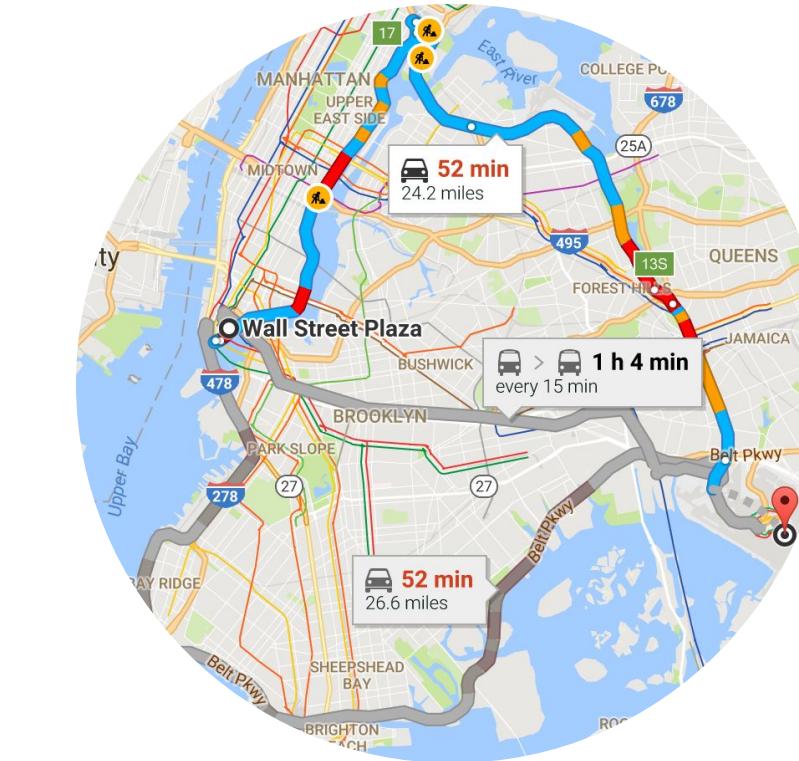
Number of passengers

+

Pickup and dropoff  
location

+

Pickup date and time

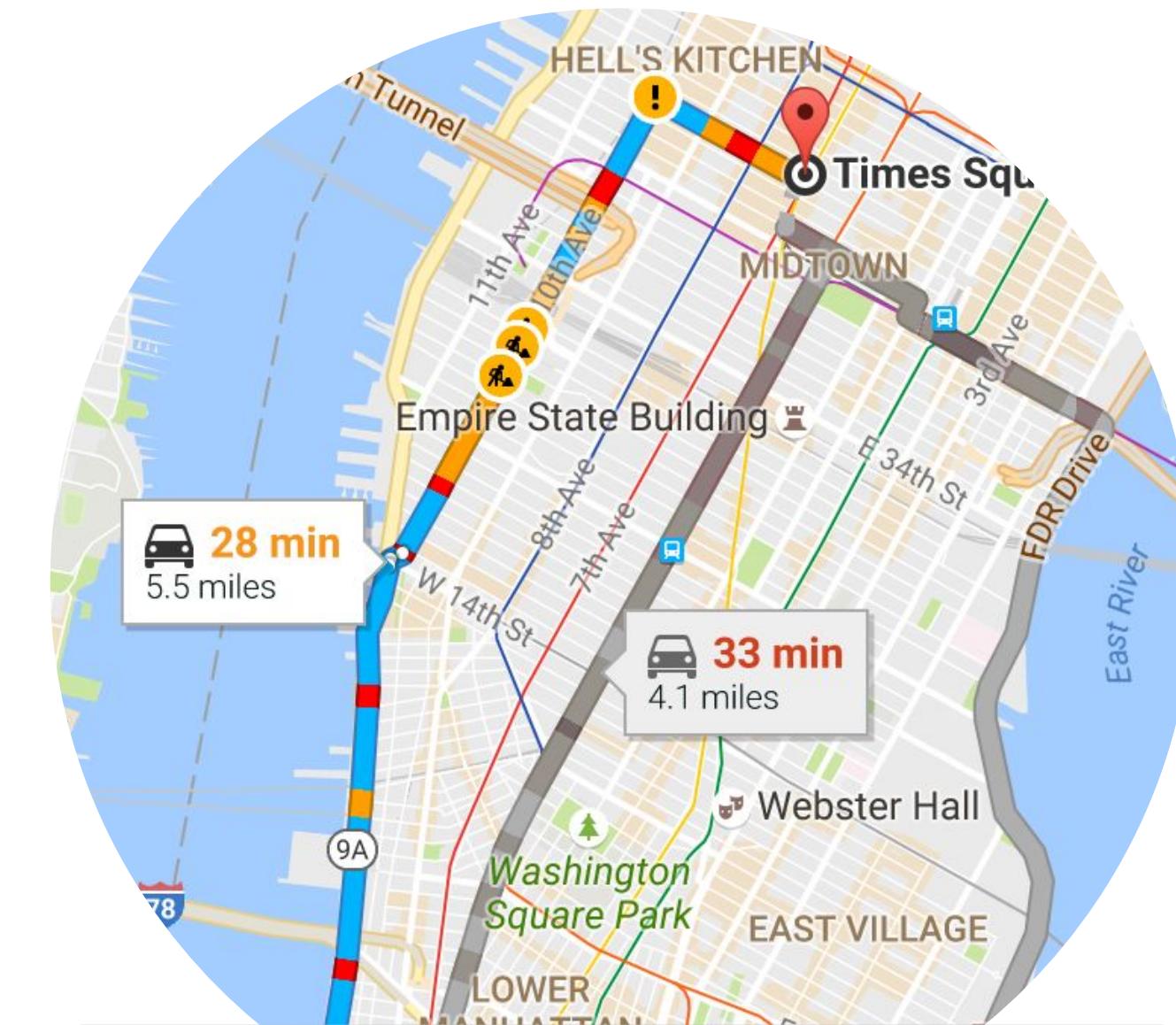


[http://www.nyc.gov/html/tlc/html/passenger/taxicab\\_rate.shtml](http://www.nyc.gov/html/tlc/html/passenger/taxicab_rate.shtml)

# Baselines are important; It helps to know what error metric is “reasonable” and/or “good” for the problem

A baseline helps you set a goal for a good value for the error metric.

Often a simple heuristic rule can function as a good benchmark.



What's a good benchmark for the taxi fare prediction?

# Evaluate model

```
SELECT * FROM ML.EVALUATE(MODEL feat_eng.baseline_model)
```

mean_absolute_error	mean_squared_error	mean_absolute_log_error	median_absolute_error	R squared
5.21	68.88	0.2581	3.790	.2261

## RMSE

(BigQuery ML splits the training data and reports evaluation statistics on the held-out set)

```
SELECT SQRT(mean_squared_error) AS rmse FROM ML.EVALUATE(MODEL feat_eng.benchmark_model)
```

rmse  
08.299

The **primary evaluation metric for this ML problem** is the root mean-squared error. RMSE measures the difference between the predictions of a model and the observed values.

# Evaluate model

```
SELECT * FROM ML.EVALUATE(MODEL feat_eng.baseline_model)
```

mean_absolute_error	mean_squared_error	mean_absolute_log_error	median_absolute_error	R squared
5.21	68.88	0.2581	3.790	.2261

## RMSE

(BigQuery ML splits the training data and reports evaluation statistics on the held-out set)

```
SELECT SQRT(mean_squared_error) AS rmse FROM ML.EVALUATE(MODEL feat_eng.benchmark_model)
```

rmse

08.299

The **primary evaluation metric for this ML problem** is the root mean-squared error. RMSE measures the difference between the predictions of a model and the observed values.

# Evaluate model

```
SELECT * FROM ML.EVALUATE(MODEL feat_eng.final_model)
```

mean_absolute_error	mean_squared_error	mean_absolute_log_error	median_absolute_error	R squared
2.26	21.65	0.0687	1.3582	0.7567

## RMSE

(BigQuery ML splits the training data and reports evaluation statistics on the held-out set)

```
SELECT SQRT(mean_squared_error) AS rmse FROM ML.EVALUATE(MODEL feat_eng.final_model)
```

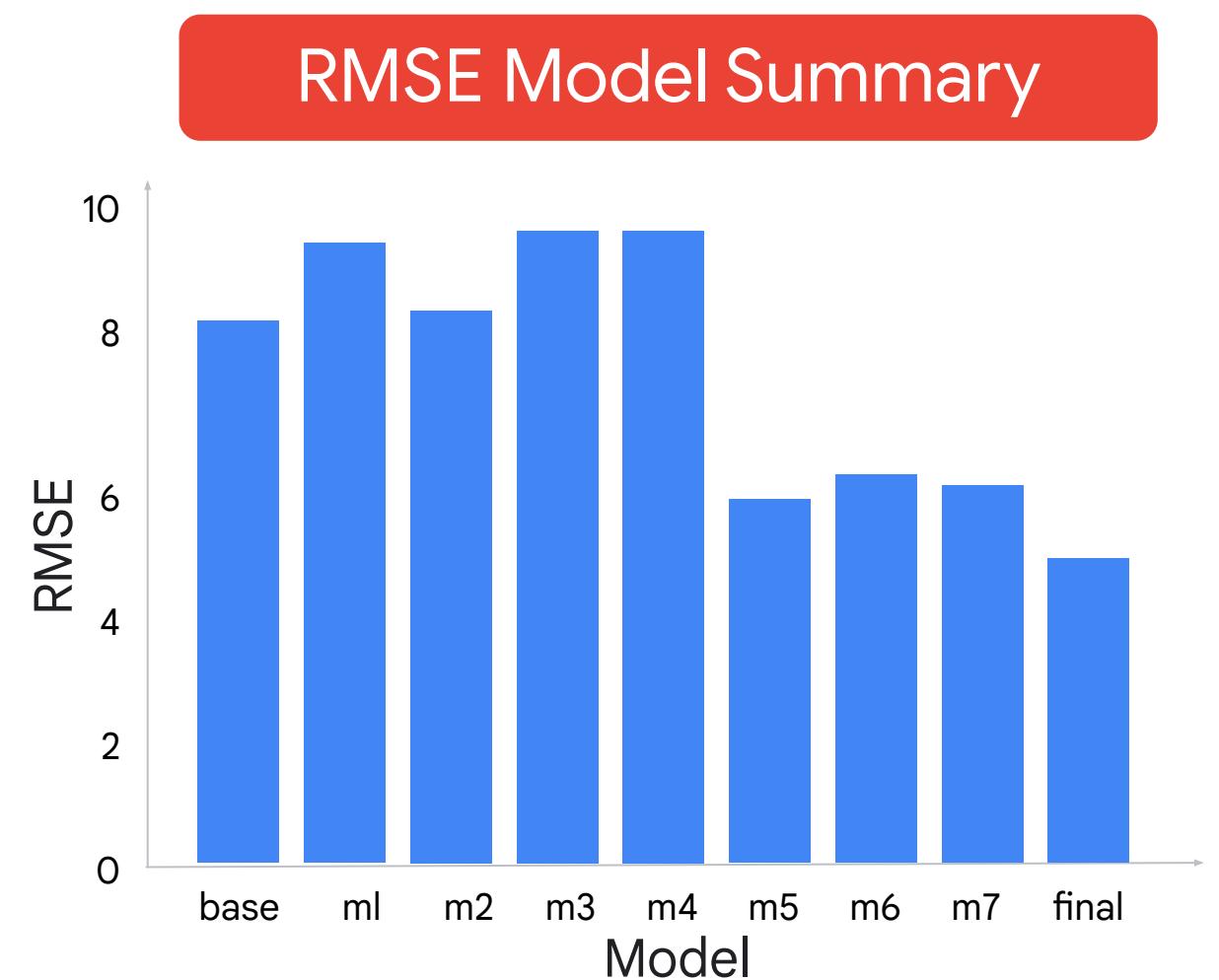
rmse

4.653

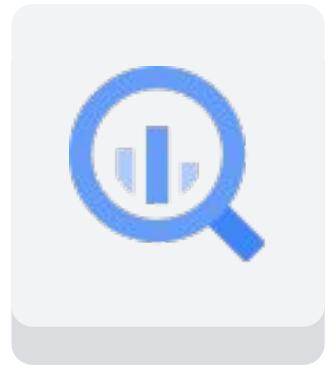
The **primary evaluation metric for this ML problem** is the root mean-squared error. RMSE measures the difference between the predictions of a model and the observed values.

# RMSE: Summary table and visualization

Model	RMSE	Description
baseline_model	8.29	--Baseline model - no feature engineering
model_1	9.431	--EXTRACT DayOfWeek from the pickup_datetime feature
model_2	8.408	--EXTRACT hourofday from the pickup_datetime feature
model_3	8.328	--Feature cross dayofweek and hourofday -Feature Cross does lead to overfitting
model_4	9.657	--Apply the ML.FEATURE_CROSS clause to categorical features
model_5	5.588	--Feature cross coordinate features to create a Euclidean feature
model_6	5.906	--Feature cross pick-up and drop-off locations features
model_7	5.75	--Apply the BUCKETIZE function
final_model	4.653	--Apply the TRANSFORM clause and L2 Regularization

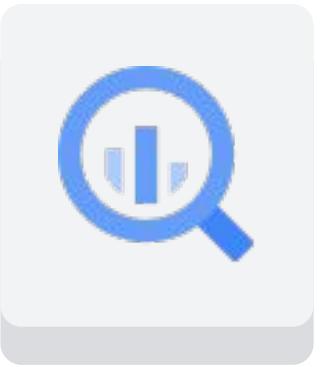


# Advanced feature engineering



BigQuery ML  
preprocessing  
functions

# Advanced feature engineering

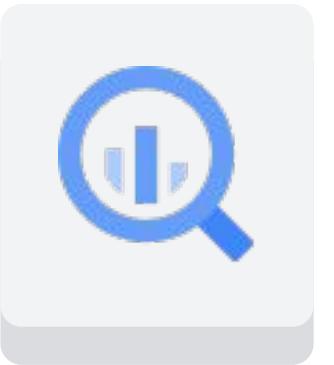


BigQuery ML  
preprocessing  
functions

```
ML.FEATURE_CROSS(STRUCT  
(features))
```

Does a feature cross of all the combinations

# Advanced feature engineering

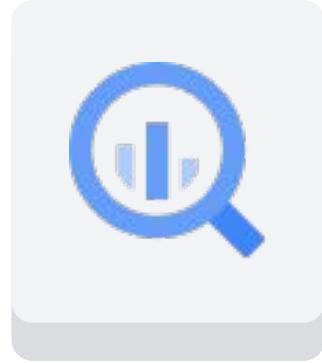


BigQuery ML  
preprocessing  
functions

```
ML.FEATURE_CROSS(STRUCT  
(features))
```

Does a feature cross of all the combinations

# Advanced feature engineering



BigQuery ML  
preprocessing  
functions

`ML.FEATURE_CROSS(STRUCT  
(features))`

Does a feature cross of all the combinations

`TRANSFORM  
(ML.FEATURE_CROSS(STRUCT  
(features)), ML.BUCKETIZE(f,  
split_points) etc...)`

Specify all preprocessing during model  
creation

`ML.BUCKETIZE(f, split_points)`

Where `split_points` is an array

# Feature crosses

01

Feature crosses  
memorize!

02

The goal of ML is  
generalization.

03

Memorization works  
when you have lots  
of data.

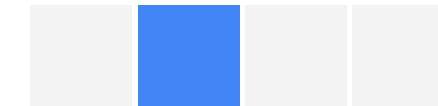
04

Feature crosses are  
powerful.

# Feature crosses lead to sparsity

Allow the model to learn traffic patterns by creating a new feature that combines the time of day and day of week (this is called a feature cross).

Hour of day



# Feature crosses lead to sparsity

Allow the model to learn traffic patterns by creating a new feature that combines the time of day and day of week (this is called a feature cross).

Hour of day

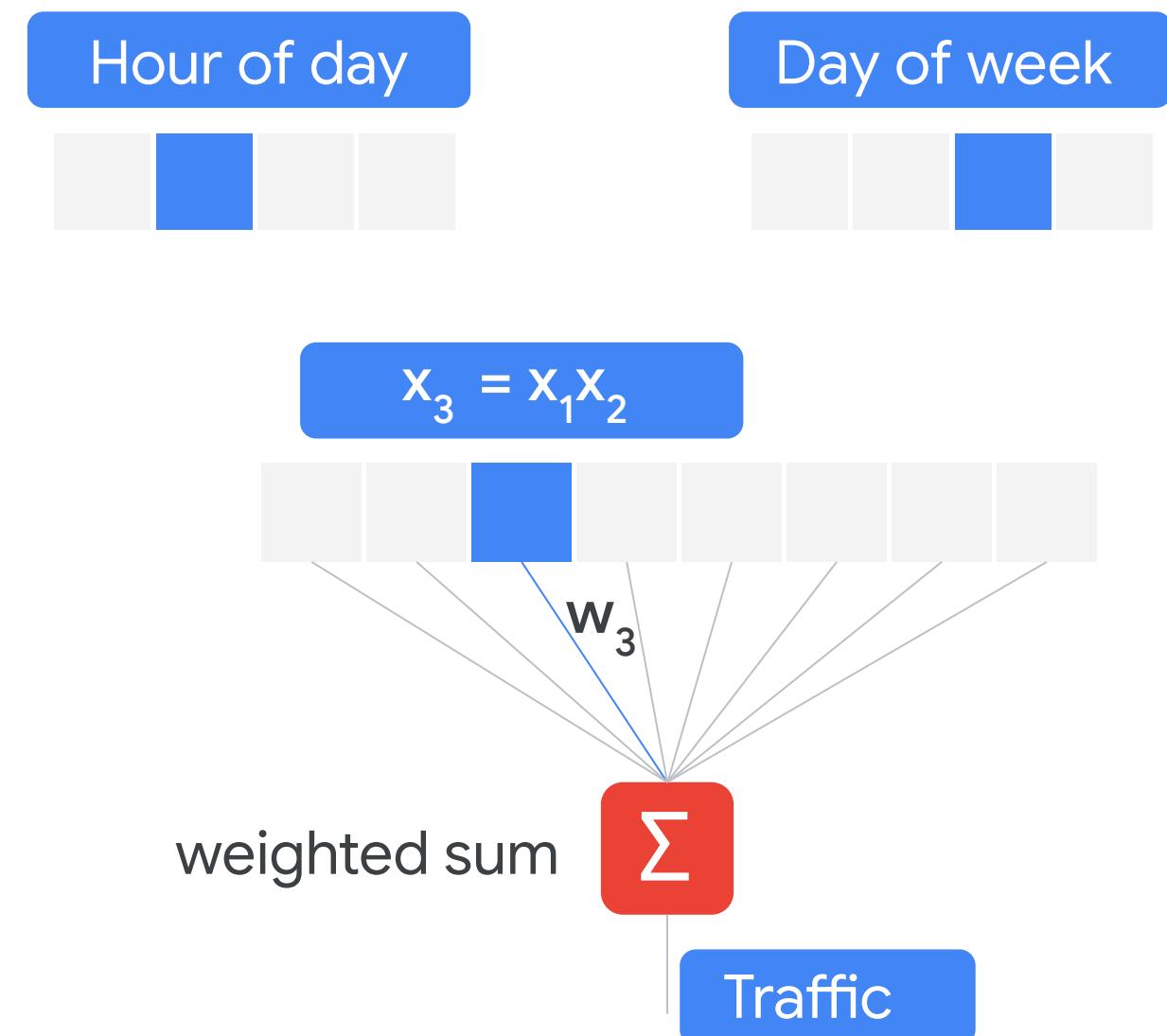


Day of week



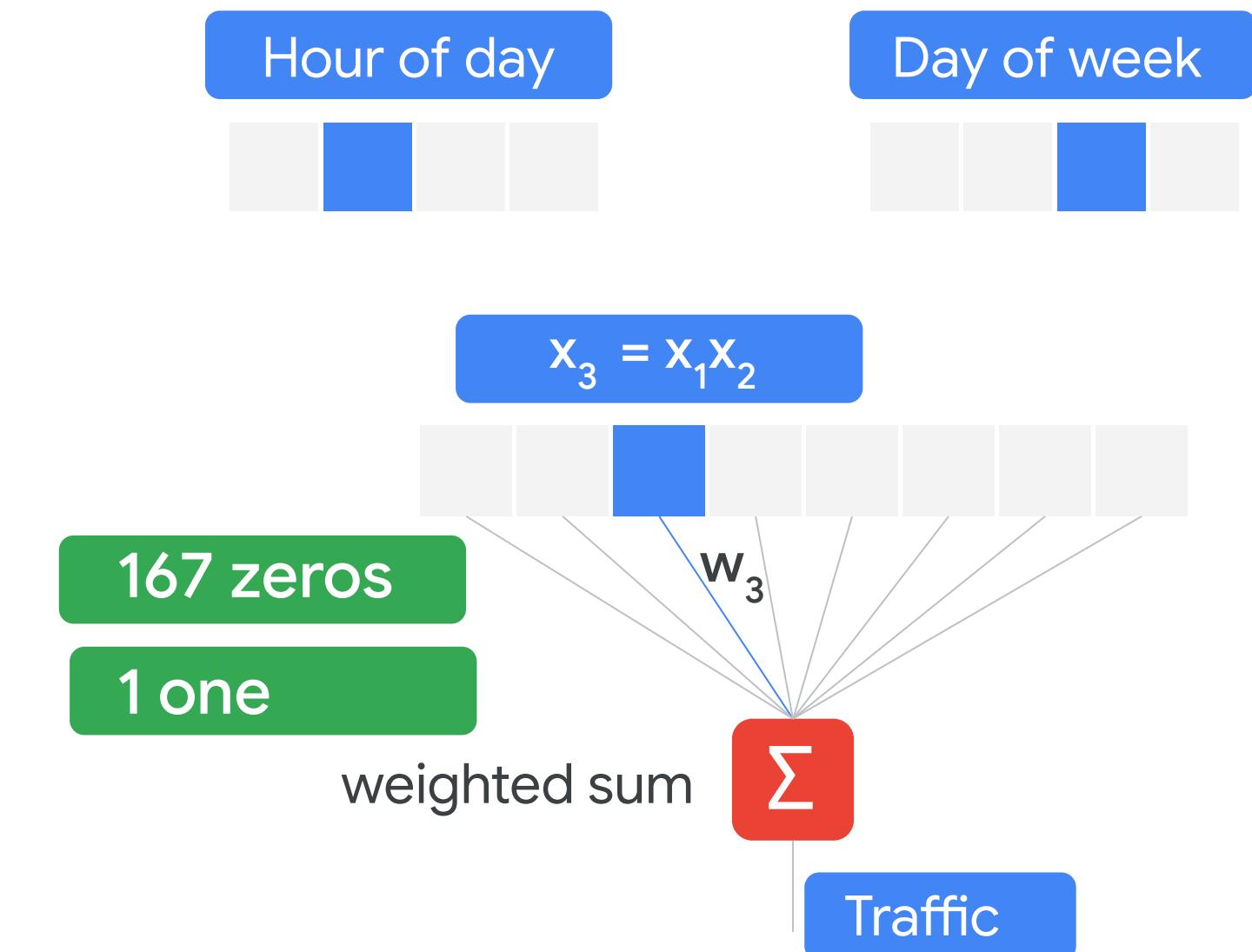
# Feature crosses lead to sparsity

Allow the model to learn traffic patterns by creating a new feature that combines the time of day and day of week (this is called a feature cross).



# Feature crosses lead to sparsity

Allow the model to learn traffic patterns by creating a new feature that combines the time of day and day of week (this is called a feature cross).



## Linear for sparse, independent features

```
[ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]  
[ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]  
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]  
[ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]  
[ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]  
[ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]  
[ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]  
[ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]  
[ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]  
[ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
```

# Spatial and temporal features

## Spatial features

```
ST_Distance(ST_GeogPoint(pickuplon, pickuplat), ST_GeogPoint(dropofflon,  
dropofflat))  
AS euclidean
```

## Temporal features

```
ML.FEATURE_CROSS(CAST(EXTRACT(DAYOFWEEK FROM pickup_datetime) AS STRING),  
CAST(EXTRACT(HOUR FROM pickup_datetime) AS STRING) AS hourofday)) AS day_hr,
```

# Feature cross

## Spatial and Features

[Schema](#)    [Details](#)    [Preview](#)

Field name	Type	Mode
fare_amount	FLOAT	NULLABLE
passengers	FLOAT	NULLABLE
euclidean	FLOAT	NULLABLE
day_hr	RECORD	NULLABLE
day_hr.dayofweek_hourofday	STRING	NULLABLE

```
{  
  "fare_amount": "2.5",  
  "passengers": "1.0",  
  "euclidean": "248.06733188206664",  
  "day_hr": {  
    "dayofweek_hourofday": :1.0"  
  },  
  "pickup_and_dropoff": "bin_402bin_370bin_403bin_370"  
},
```

# Feature cross

## Spatial and Features

[Schema](#)    [Details](#)    [Preview](#)

Field name	Type	Mode
fare_amount	FLOAT	NULLABLE
passengers	FLOAT	NULLABLE
euclidean	FLOAT	NULLABLE
day_hr	RECORD	NULLABLE
day_hr.dayofweek_hourofday	STRING	NULLABLE

```
{  
  "fare_amount": "2.5",  
  "passengers": "1.0",  
  "euclidean": "248.06733188206664",  
  "day_hr": {  
    "dayofweek_hourofday": :1.0"  
  },  
  "pickup_and_dropoff": "bin_402bin_370bin_403bin_370"  
},
```

# BUCKETIZE

## Spatial and Features

```
CONCAT(  
    ML.BUCKETIZE(pickuplon, GENERATE_ARRAY(-78, -70, 0.01)),  
    ML.BUCKETIZE(pickuplat, GENERATE_ARRAY(37, 45, 0.01)),  
    ML.BUCKETIZE(dropofflon, GENERATE_ARRAY(-78, -70, 0.01)),  
    ML.BUCKETIZE(dropofflat, GENERATE_ARRAY(37, 45, 0.01)),  
) AS pickup_and_dropoff
```

```
{  
    "fare_amount": "2.5",  
    "passengers": "1.0",  
    "euclidean": "248.06733188206664",  
    "day_hr": {  
        "dayofweek_hourofday": :1.0"  
    },  
    "pickup_and_dropoff": "bin_402bin_370bin_403bin_370"  
},
```

pickup_and_dropoff
bin_406bin_379bin_406bin_379
bin_402bin_376bin_402bin_376
bin_422bin_366bin_423bin_366
bin_404bin_378bin_405bin_380
bin_403bin_374bin_403bin_374
bin_401bin_376bin_406bin_368
bin_407bin_376bin_407bin_376
bin_404bin_378bin_404bin_377
bin_402bin_376bin_403bin_376
bin_402bin_376bin_402bin_376
bin_401bin_372bin_401bin_372
bin_401bin_375bin_401bin_375
bin_404bin_380bin_404bin_380

# TRANSFORM clause: BigQuery ML

```
CREATE OR REPLACE MODEL feat_eng.final_model
TRANSFORM(
    fare_amount,
    #SQRT( (pickuplon-dropofflon)*(pickuplon-dropofflon) +(pickuplat-dropofflat)*(pickuplat-dropofflat) ) AS euclidean,
    ST_Distance(ST_GeogPoint(pickuplon, pickuplat), ST_GeogPoint(dropofflon, dropofflat)) AS euclidean,
    ML.FEATURE_CROSS(STRUCT(CAST(EXTRACT(DAYOFWEEK FROM pickup_datetime) AS STRING) AS dayofweek,
        CAST(EXTRACT(HOUR FROM pickup_datetime) AS STRING) AS hourofday)) AS day_hr,
    CONCAT(
        ML.BUCKETIZE(pickuplon, GENERATE_ARRAY(-78, -70, 0.01)),
        ML.BUCKETIZE(pickuplat, GENERATE_ARRAY(37, 45, 0.01)),
        ML.BUCKETIZE(dropofflon, GENERATE_ARRAY(-78, -70, 0.01)),
        ML.BUCKETIZE(dropofflat, GENERATE_ARRAY(37, 45, 0.01))),
    ) AS pickup_and_dropoff
)
OPTIONS(input_label_cols=[`fare_amount`], model_type='linear_reg', 12_reg=0.1
AS
SELECT * FROM feat_eng.feateng_training_data
```

**TRANSFORM**  
ensures that  
transformations  
are automatically  
applied during  
**ML.PREDICT**

```
CREATE OR REPLACE MODEL feat_eng.final_model
TRANSFORM(
    fare_amount,
    #SQRT( (pickuplon-dropofflon)*(pickuplon-dropofflon)
+(pickuplat-dropofflat)*(pickuplat-dropofflat) ) AS euclidean,
    ST_Distance(ST_GeogPoint(pickuplon, pickuplat),
ST_GeogPoint(dropofflon, dropofflat)) AS euclidean,
    ML.FEATURE_CROSS(STRUCT(CAST(EXTRACT(DAYOFWEEK FROM
pickup_datetime) AS STRING) AS dayofweek,
    CAST(EXTRACT(HOUR FROM pickup_datetime) AS STRING) AS
hourofday)) AS day_hr,
    CONCAT(
        ML.BUCKETIZE(pickuplon, GENERATE_ARRAY(-78, -70, 0.01)),
        ML.BUCKETIZE(pickuplat, GENERATE_ARRAY(37, 45, 0.01)),
        ML.BUCKETIZE(dropofflon, GENERATE_ARRAY(-78, -70, 0.01)),
        ML.BUCKETIZE(dropofflat, GENERATE_ARRAY(37, 45, 0.01)),
    ) AS pickup_and_dropoff
)
OPTIONS(input_label_cols=[`fare_amount`], model_type='linear_reg',
12_reg=0.1
AS
SELECT * FROM feat_eng.feateng_training_data
```

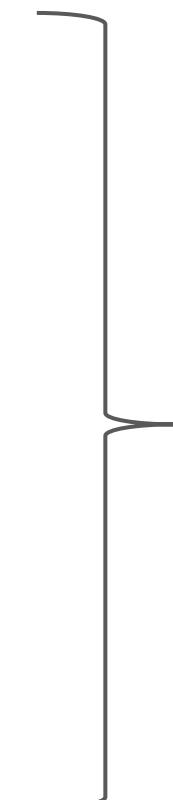
# BigQuery ML feature engineering summary

Remove examples that you don't want to train on.

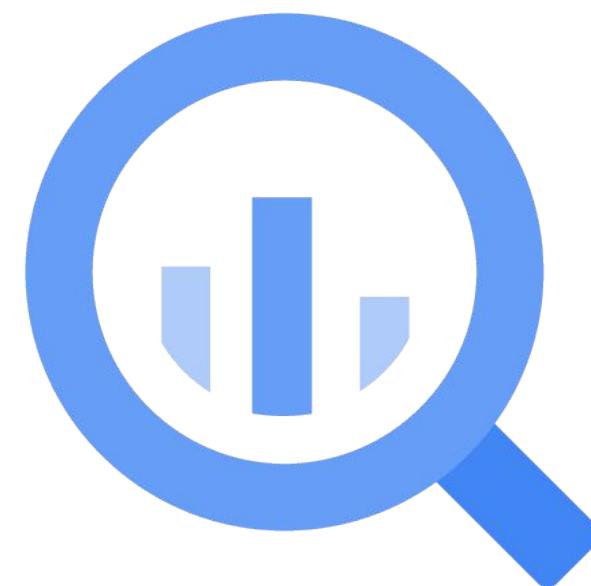
Compute vocabularies for categorical columns.

Compute aggregate statistics for numeric columns.

Consider advanced feature engineering using  
`ML.FEATURE_CROSS`, `TRANSFORM`, and `BUCKETIZE`.



In BigQuery



# ML problem: Estimating housing price

Housing Price?

Median house

+

Longitude

+

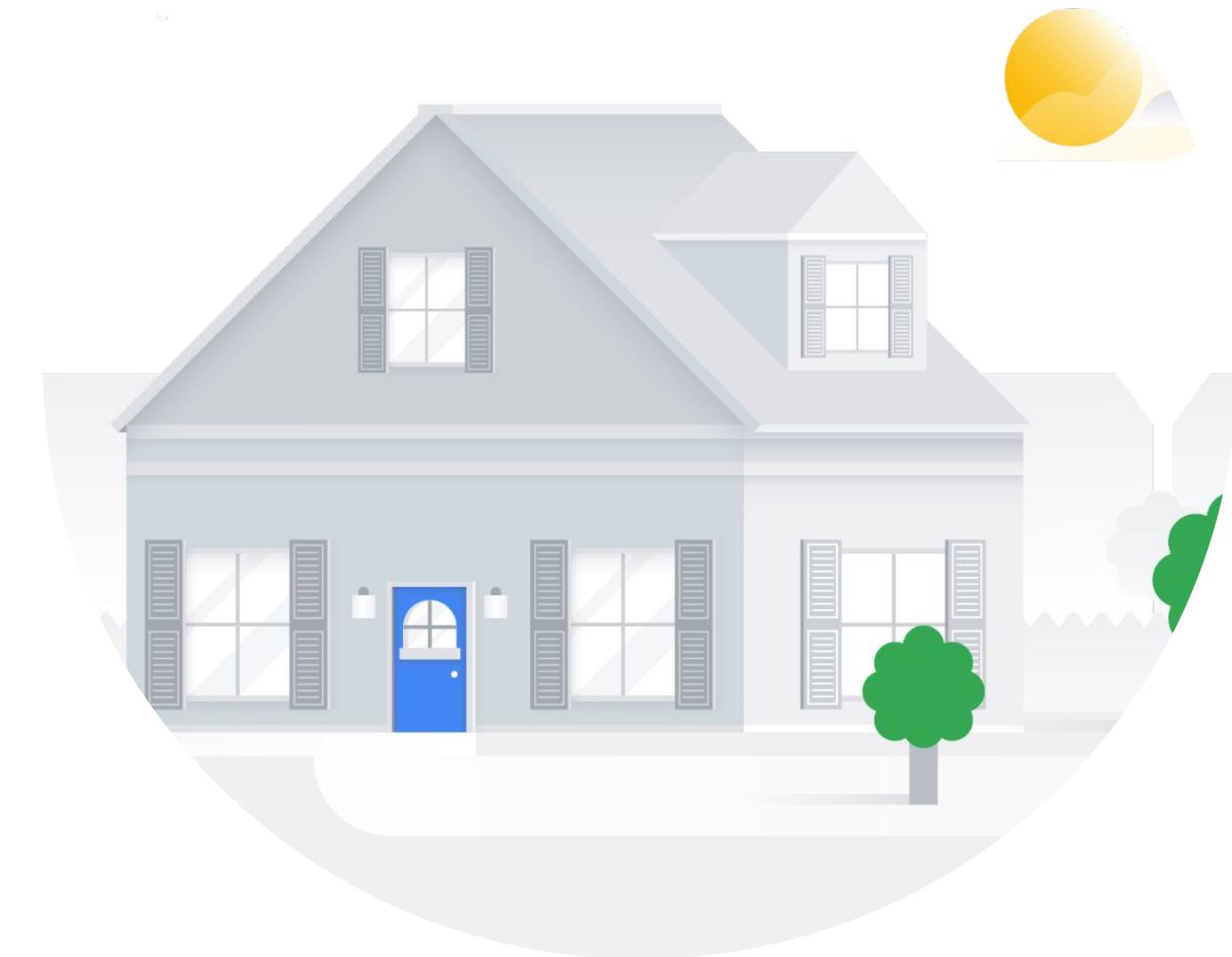
Latitude

+

Number of rooms

+

Number of bedrooms



# Wrap the dataframe with tf.data

```
def df_to_dataset(dataframe, shuffle=True, batch_size=32):
    dataframe = dataframe.copy()
    labels = dataframe.pop('median_house_value')
    ds = tf.data.Dataset.from_tensor_slices((dict(dataframe), labels))
    if shuffle:
        ds = ds.shuffle(buffer_size=len(dataframe))
    ds = ds.batch(batch_size)
    return ds
```

# Build an input data pipeline

## Numeric columns

The output of a feature column becomes the input to the model. A numeric is the simplest type of column and is used to represent real valued features. When you use this column, your model will receive the column value from the dataframe unchanged.

```
num_c = [ 'longitude',
          'latitude',
          'housing_median_age',
          'total_rooms',
          'total_bedrooms',
          'population',
          'households',
          'median_income' ]
```

# Build an input data pipeline

## Categorical columns

In this dataset, 'ocean\_proximity' is represented as a string. You cannot feed strings directly to a model; instead, you must first map them to numeric values. The categorical vocabulary columns provide a way to represent strings as a one-hot vector.

```
cat_i_c = ['ocean_proximity']
for feature_name in cat_i_c:
    vocabulary =
        dataframe[feature_name].unique()
    cat_c =
        tf.feature_column.categorical_column_with_
        vocabulary_list(feature_name, vocabulary)
    one_hot =
        feature_column.indicator_column(cat_c)
    feature_columns.append(one_hot)
```

# Build an input data pipeline

## Bucketize columns

Often, you don't want to feed a number directly into the model, but instead split its value into different categories based on numerical ranges. Consider the raw data that represents a home's age. Instead of representing the house age as a numeric column, you could split the home age into several buckets using a bucketized column. Notice that the one-hot values below describe which age range each row matches.

```
age_buckets =  
    feature_column.bucketized_column(Age,  
        boundaries=[18, 25, 30, 35, 40, 45, 50,  
        55, 60, 65])  
    feature_columns.append(age_buckets)
```

# Build an input data pipeline

## Feature cross columns

Combining features into a single feature, better known as a feature cross, enables a model to learn separate weights for each combination of features.

```
vocabulary = dataframe['ocean_proximity'].unique()
ocean_proximity =
tf.feature_column.categorical_column_with_vocabulary_list
('ocean_proximity', vocabulary)
crossed_feature =
feature_column.crossed_column([age_buckets,
ocean_proximity], hash_bucket_size=1000)
crossed_feature =
feature_column.indicator_column(crossed_feature)
feature_columns.append(crossed_feature)
```

# Feature-engineered input data pipeline

Numerical

```
num_c = ['longitude', 'latitude', 'housing_median_age', 'total_rooms', 'total_bedrooms', 'population',  
'households', 'median_income']
```

Categorical

```
cat_i_c = ['ocean_proximity']  
for feature_name in cat_i_c:  
    vocabulary = dataframe[feature_name].unique()
```

Bucketized

```
cat_c = tf.feature_column.categorical_column_with_vocabulary_list(feature_name, vocabulary)  
one_hot = feature_column.indicator_column(cat_c)  
feature_columns.append(one_hot)
```

Feature cross

```
age_buckets = feature_column.bucketized_column(Age, boundaries=[18, 25, 30, 35, 40, 45, 50, 55, 60, 65])  
feature_columns.append(age_buckets)
```

```
vocabulary = dataframe['ocean_proximity'].unique()  
ocean_proximity = tf.feature_column.categorical_column_with_vocabulary_list('ocean_proximity', vocabulary)  
crossed_feature = feature_column.crossed_column([age_buckets, ocean_proximity], hash_bucket_size=1000)  
crossed_feature = feature_column.indicator_column(crossed_feature)  
feature_columns.append(crossed_feature)
```

# Build the DNN

Keras Sequential Model API

```
feature_layer = tf.keras.layers.DenseFeatures(feature_columns)
model = tf.keras.Sequential([
    feature_layer,
```

# Build the DNN

Keras Sequential Model API

```
feature_layer = tf.keras.layers.DenseFeatures(feature_columns)
model = tf.keras.Sequential([
    feature_layer,
    layers.Dense(12,  input_dim=8, activation='relu'),
```

# Build the DNN

Keras Sequential Model API

```
feature_layer = tf.keras.layers.DenseFeatures(feature_columns)
model = tf.keras.Sequential([
    feature_layer,
    layers.Dense(12, input_dim=8, activation='relu'),
    layers.Dense(8, activation='relu'),
```

# Build the DNN

Keras Sequential Model API

```
feature_layer = tf.keras.layers.DenseFeatures(feature_columns)
model = tf.keras.Sequential([
    feature_layer,
    layers.Dense(12, input_dim=8, activation='relu'),
    layers.Dense(8, activation='relu'),
    layers.Dense(1, activation='linear', name='median_house_value')
])
```

# Build the DNN

Keras Sequential Model API

```
feature_layer = tf.keras.layers.DenseFeatures(feature_columns)
model = tf.keras.Sequential([
    feature_layer,
    layers.Dense(12, input_dim=8, activation='relu'),
    layers.Dense(8, activation='relu'),
    layers.Dense(1, activation='linear', name='median_house_value')
])
model.compile(optimizer='adam',
              loss='mse',
              metrics=['mse'])
```

# Build the DNN

Keras Sequential Model API

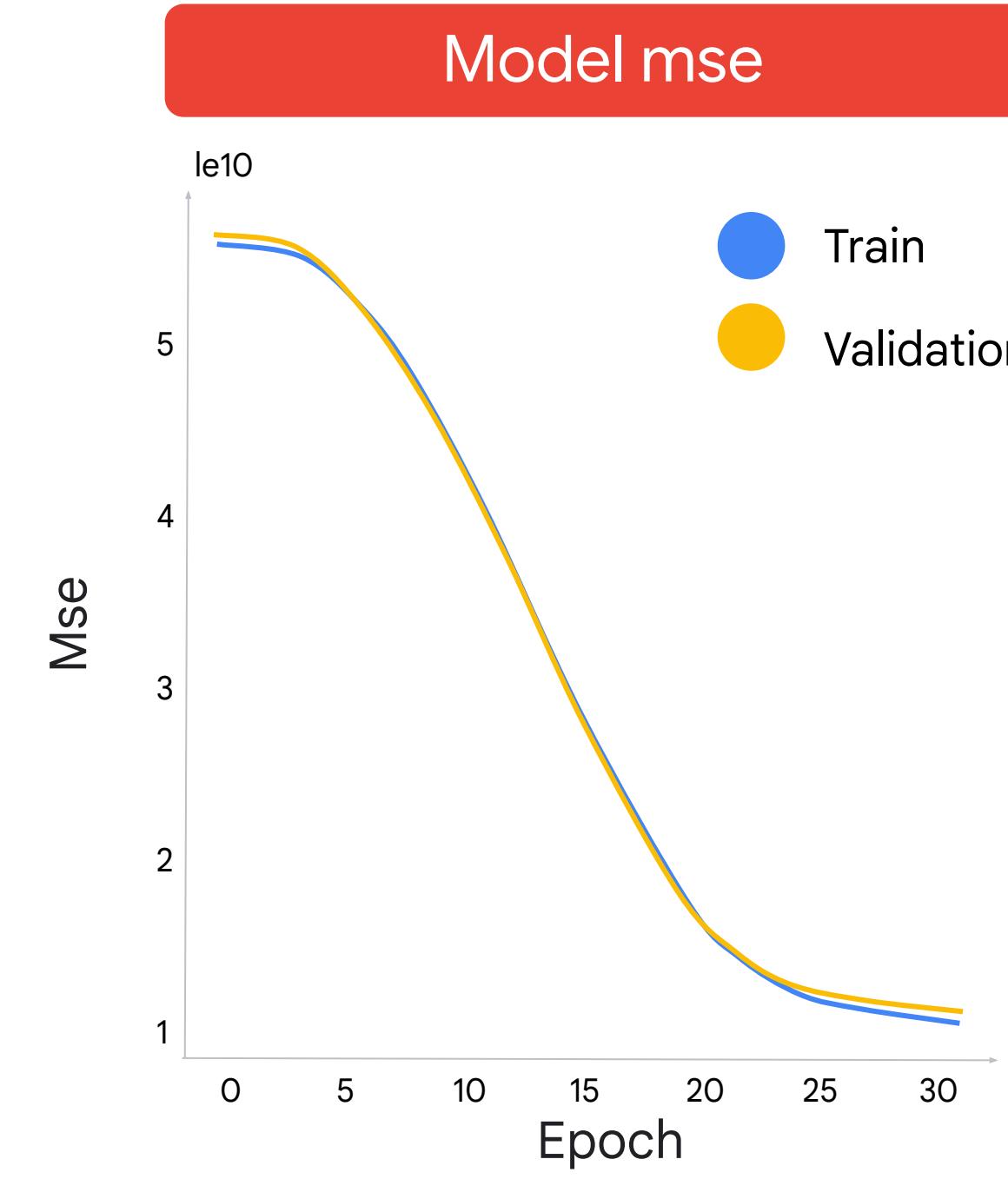
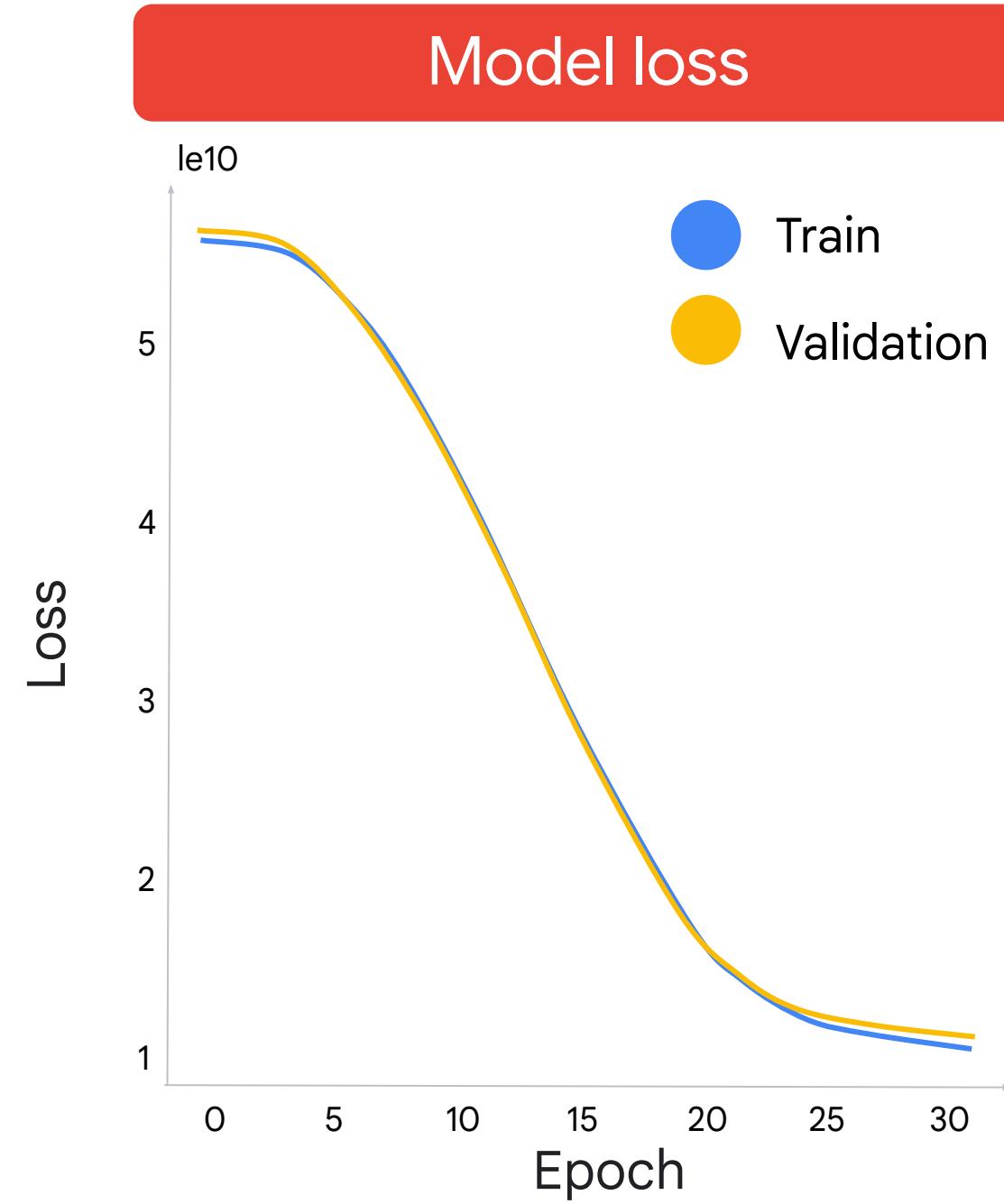
```
feature_layer = tf.keras.layers.DenseFeatures(feature_columns)
model = tf.keras.Sequential([
    feature_layer,
    layers.Dense(12, input_dim=8, activation='relu'),
    layers.Dense(8, activation='relu'),
    layers.Dense(1, activation='linear', name='median_house_value')
])
model.compile(optimizer='adam',
              loss='mse',
              metrics=['mse'])
```

# Build the DNN

Keras Sequential Model API

```
feature_layer = tf.keras.layers.DenseFeatures(feature_columns)
model = tf.keras.Sequential([
    feature_layer,
    layers.Dense(12, input_dim=8, activation='relu'),
    layers.Dense(8, activation='relu'),
    layers.Dense(1, activation='linear', name='median_house_value')
])
model.compile(optimizer='adam',
              loss='mse',
              metrics=['mse'])
history = model.fit(train_ds,
                     validation_data=val_ds,
                     epochs=32)
```

# Model loss and MSE



# Model prediction

House near the ocean

Median\_house\_value is  
\$249,000, prediction is  
\$234,000

```
model.predict({  
    'longitude': tf.convert_to_tensor([-122.43]),  
    'latitude': tf.convert_to_tensor([37.63]),  
    'housing_median_age': tf.convert_to_tensor([34.0]),  
  
    'total_rooms': tf.convert_to_tensor([4135.0]),  
    'total_bedrooms': tf.convert_to_tensor([687.0]),  
    'population': tf.convert_to_tensor([2154.0]),  
    'households': tf.convert_to_tensor([742.0]),  
    'median_income': tf.convert_to_tensor([ 4.9732]),  
    'ocean_proximity': tf.convert_to_tensor(['NEAR  
OCEAN'])}  
, steps=1)
```

# ML problem: Estimating taxi fare

Taxi fare:

Fare amount

+

Distance travelled

+

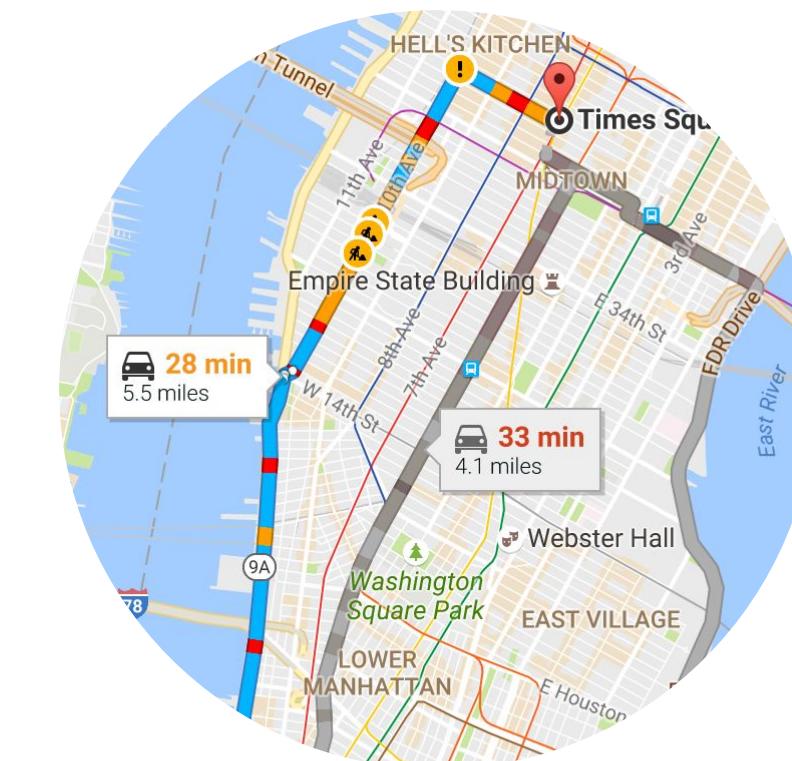
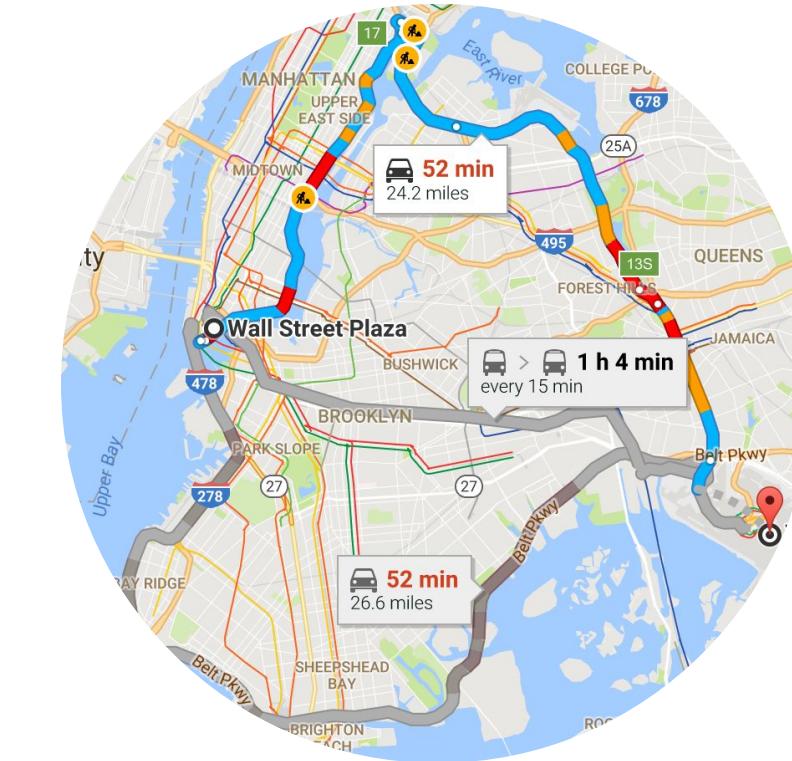
Number of passengers

+

Pickup and dropoff  
location

+

Pickup date and time



[http://www.nyc.gov/html/tlc/html/passenger/taxicab\\_rate.shtml](http://www.nyc.gov/html/tlc/html/passenger/taxicab_rate.shtml)

# Define features, default values, and label

```
CSV_COLUMNS = [
    'fare_amount',
    'pickup_datetime',
    'pickup_longitude',
    'pickup_latitude',
    'dropoff_longitude',
    'dropoff_latitude',
    'passenger_count',
    'key',
]
LABEL_COLUMN = 'fare_amount'
STRING_COLS = ['pickup_datetime']
NUMERIC_COLS = ['pickup_longitude', 'pickup_latitude',
                'dropoff_longitude', 'dropoff_latitude',
                'passenger_count']
DEFAULTS = [[0.0], ['na'], [0.0], [0.0], [0.0], [0.0], [0.0], ['na']]]
DAYS = ['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat']
```

# Create an input pipeline using tf.data

```
def features_and_labels(row_data):
    for unwanted_col in ['pickup_datetime', 'key']:
        row_data.pop(unwanted_col)
    label = row_data.pop(LABEL_COLUMN)
    return row_data, label # features, label

# load the training data
def load_dataset(pattern, batch_size=1, mode=tf.estimator.ModeKeys.EVAL):
    dataset = (tf.data.experimental.make_csv_dataset(pattern, batch_size, CSV_COLUMNS,
DEFAULTS)
               .map(features_and_labels) # features, label
               )
    if mode == tf.estimator.ModeKeys.TRAIN:
        dataset = dataset.shuffle(1000).repeat()
    dataset = dataset.prefetch(1)
    return dataset
```

# Build the DNN

Keras Functional Model API

```
def rmse(y_true, y_pred): # Root mean square error
    return tf.sqrt(tf.reduce_mean(tf.square(y_pred - y_true)))

def build_dnn_model(): # Build the DNN model
    # input layer
    inputs = {
        colname: tf.keras.layers.Input(name=colname, shape=(),
                                         dtype='float32')
        for colname in NUMERIC_COLS
    }

    # feature_columns
    feature_columns = {
        colname: tf.feature_column.numeric_column(colname)
        for colname in NUMERIC_COLS
    }
```

# Build the DNN

Keras Functional Model API

```
# Constructor for DenseFeatures takes a list of numeric columns
# The Functional API in Keras requires you specify:
LayerConstructor()(inputs)

dnn_inputs =
tf.keras.layers.DenseFeatures(feature_columns.values())(inputs)

# two hidden layers of [32, 8]
h1 = tf.keras.layers.Dense(32, activation='relu',
name='h1')(dnn_inputs)
h2 = tf.keras.layers.Dense(8, activation='relu', name='h2')(h1)

# final output is a linear activation because this is regression
output = tf.keras.layers.Dense(1, activation='linear',
name='fare')(h2)
model = tf.keras.models.Model(inputs, output)

# compile model
model.compile(optimizer='adam', loss='mse', metrics=[rmse, 'mse'])

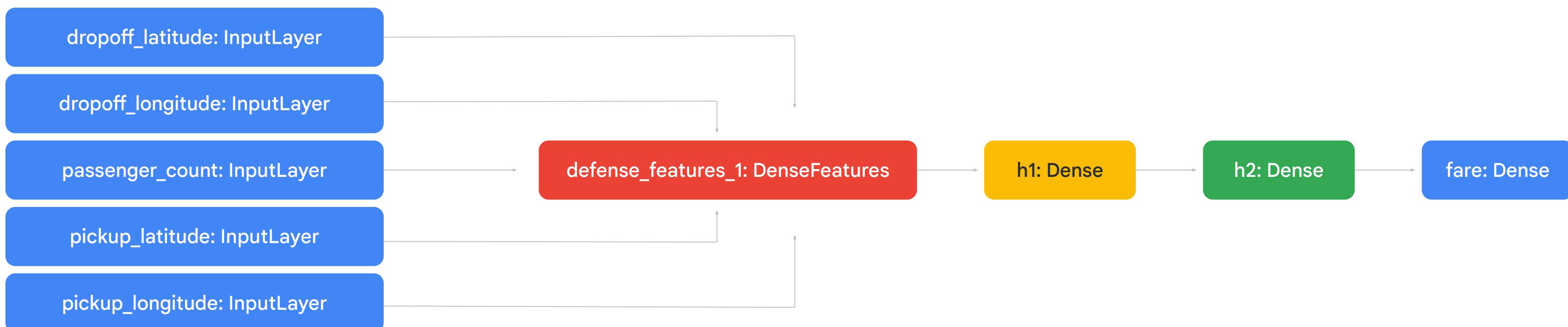
return model

print("Here is our DNN architecture so far:\n")
model = build_dnn_model()
print(model.summary())
```

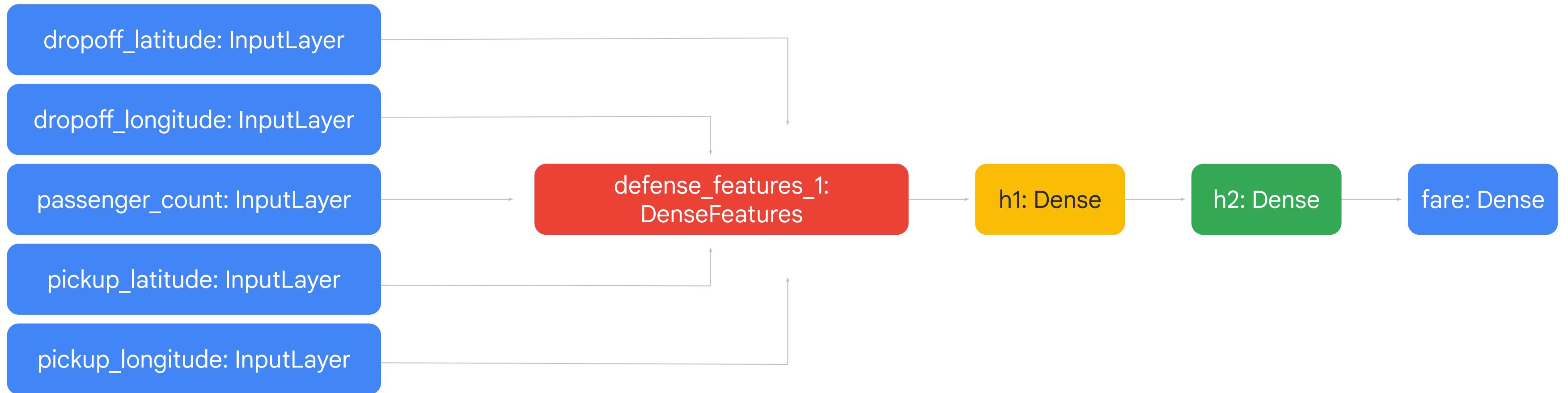
# Connect Dense Features layer to rest of model

## Summary

```
dnn_inputs = tf.keras.layers.DenseFeatures(feature_columns)(inputs)
h1 = tf.keras.layers.Dense(32, activation='relu', name='h1')(dnn_inputs)
h2 = tf.keras.layers.Dense(8, activation='relu', name='h2')(h1)
output = tf.keras.layers.Dense(1, activation='linear', name='fare')(h2)
model = tf.keras.models.Model(inputs, output)
```



# Visualize the model

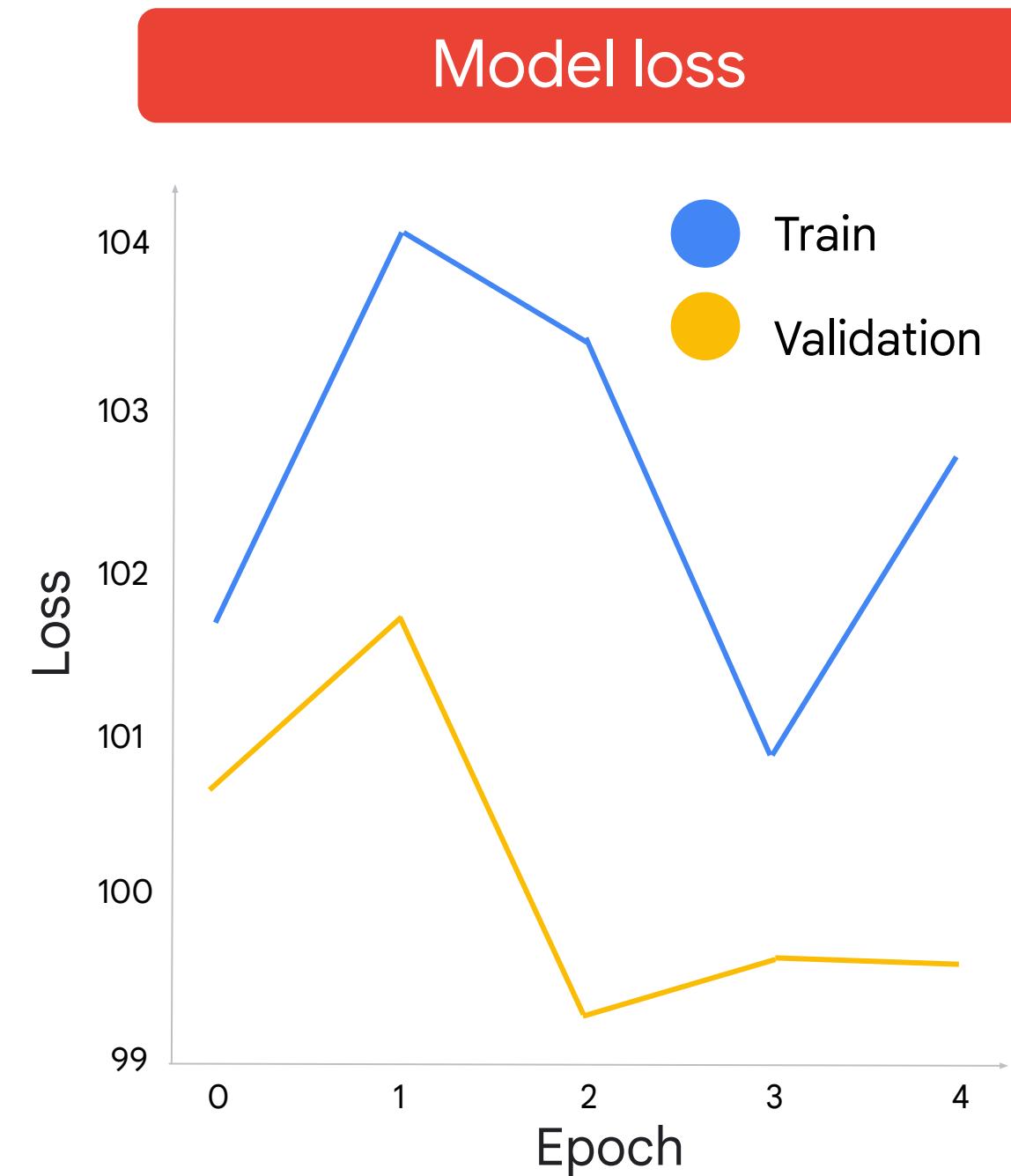


```
TRAIN_BATCH_SIZE = 32  
NUM_TRAIN_EXAMPLES = 59621 * 5  
NUM_EVALS = 5  
NUM_EVAL_EXAMPLES = 14906
```

```
trains = load_dataset(`../data/taxi-train*`,  
                      TRAIN_BATCH_SIZE,  
                      tf.estimator.ModeKeys.TRAIN)  
evalds = load_dataset(`../data/taxi-valid*`,  
                      1000,  
                      tf.estimator.ModeKeys.EVAL).take(NUM_EVAL_EXAMPLES//1000)  
  
steps_per_epoch = NUM_TRAIN_EXAMPLES// (TRAIN_BATCH_SIZE *  
NUM_EVALS)  
  
history = model.fit(trains,  
                     validation_data=evalds,  
                     epoch=NUM_EVALS,  
                     steps_per_epoch=steps_per_epoch)
```

# Train the model

# Model loss and MSE



# Model prediction

New York taxi fare at these coordinates is \$12.29

```
model.predict({  
    'pickup_longitude': tf.convert_to_tensor([-73.982683]),  
    'pickup_latitude': tf.convert_to_tensor([40.742104]),  
    'dropoff_longitude': tf.convert_to_tensor([-73.983766]),  
    'dropoff_latitude': tf.convert_to_tensor([40.755174]),  
    'passenger_count': tf.convert_to_tensor([3.0]),  
    'pickup_datetime': tf.convert_to_tensor(['2019-06-03  
04:21:29 UTC'], dtype=tf.string),  
}, steps=1)
```

# Temporal features

```
# TODO 1a
def parse_datetime(s):
    if type(s) is not str:
        s = s.numpy().decode('utf-8')
    return datetime.datetime.strptime(s, "%Y-%m-%d %H:%M:%S %Z")
```

# Temporal features

```
# TODO 1a
def parse_datetime(s):
    if type(s) is not str:
        s = s.numpy().decode('utf-8')
    return datetime.datetime.strptime(s, "%Y-%m-%d %H:%M:%S %Z")

# TODO 1b
def get_dayofweek(s):
    ts = parse_datetime(s)
    return DAYS[ts.weekday()]
```

# Temporal features

```
# TODO 1a
def parse_datetime(s):
    if type(s) is not str:
        s = s.numpy().decode('utf-8')
    return datetime.datetime.strptime(s, "%Y-%m-%d %H:%M:%S %Z")

# TODO 1b
def get_dayofweek(s):
    ts = parse_datetime(s)
    return DAYS[ts.weekday()]

# TODO 1c
@tf.function
def dayofweek(ts_in):
    return tf.map_fn(
        lambda s: tf.py_function(get_dayofweek, inp=[s],
        Tout=tf.string),
        ts_in)
```

# Computing Euclidean distance

```
# TODO 2  
def euclidean(params):  
    lon1, lat1, lon2, lat2 = params  
    londiff = lon2 - lon1  
    latdiff = lat2 - lat1  
    return tf.sqrt(londiff*londiff + latdiff*latdiff)
```

# Computing Euclidean distance

```
# TODO 2  
def euclidean(params):  
    lon1, lat1, lon2, lat2 = params  
    londiff = lon2 - lon1  
    latdiff = lat2 - lat1  
    return tf.sqrt(londiff*londiff + latdiff*latdiff)
```

Max/Min scaling latitude and longitude

```
def scale_longitude(lon_column):  
    return (lon_column + 78)/8.
```

```
def scale_latitude(lat_column):  
    return (lat_column - 37)/8.
```

# Computing Euclidean distance

```
# TODO 2  
def euclidean(params):  
    lon1, lat1, lon2, lat2 = params  
    londiff = lon2 - lon1  
    latdiff = lat2 - lat1  
    return tf.sqrt(londiff*londiff + latdiff*latdiff)
```

Max/Min scaling latitude and longitude

```
def scale_longitude(lon_column):  
    return (lon_column + 78)/8.
```

```
def scale_latitude(lat_column):  
    return (lat_column - 37)/8.
```

# Computing Euclidean distance

```
# TODO 2  
def euclidean(params):  
    lon1, lat1, lon2, lat2 = params  
    londiff = lon2 - lon1  
    latdiff = lat2 - lat1  
    return tf.sqrt(londiff*londiff + latdiff*latdiff)
```

Max/Min scaling latitude and longitude

```
def scale_longitude(lon_column):  
    return (lon_column + 78)/8.
```

```
def scale_latitude(lat_column):  
    return (lat_column - 37)/8.
```

# Computing Euclidean distance

```
# TODO 2  
def euclidean(params):  
    lon1, lat1, lon2, lat2 = params  
    londiff = lon2 - lon1  
    latdiff = lat2 - lat1  
    return tf.sqrt(londiff*londiff + latdiff*latdiff)
```

Max/Min scaling latitude and longitude

```
def scale_longitude(lon_column):  
    return (lon_column + 78)/8.
```

```
def scale_latitude(lat_column):  
    return (lat_column - 37)/8.
```

# Geolocation features

```
# Scaling longitude from range [-70, -78] to [0, 1]
for lon_col in ['pickup_longitude', 'dropoff_longitude']:
    transformed[lon_col] = layers.Lambda(
        scale_longitude,
        name="scale_{}".format(lon_col))(inputs[lon_col])

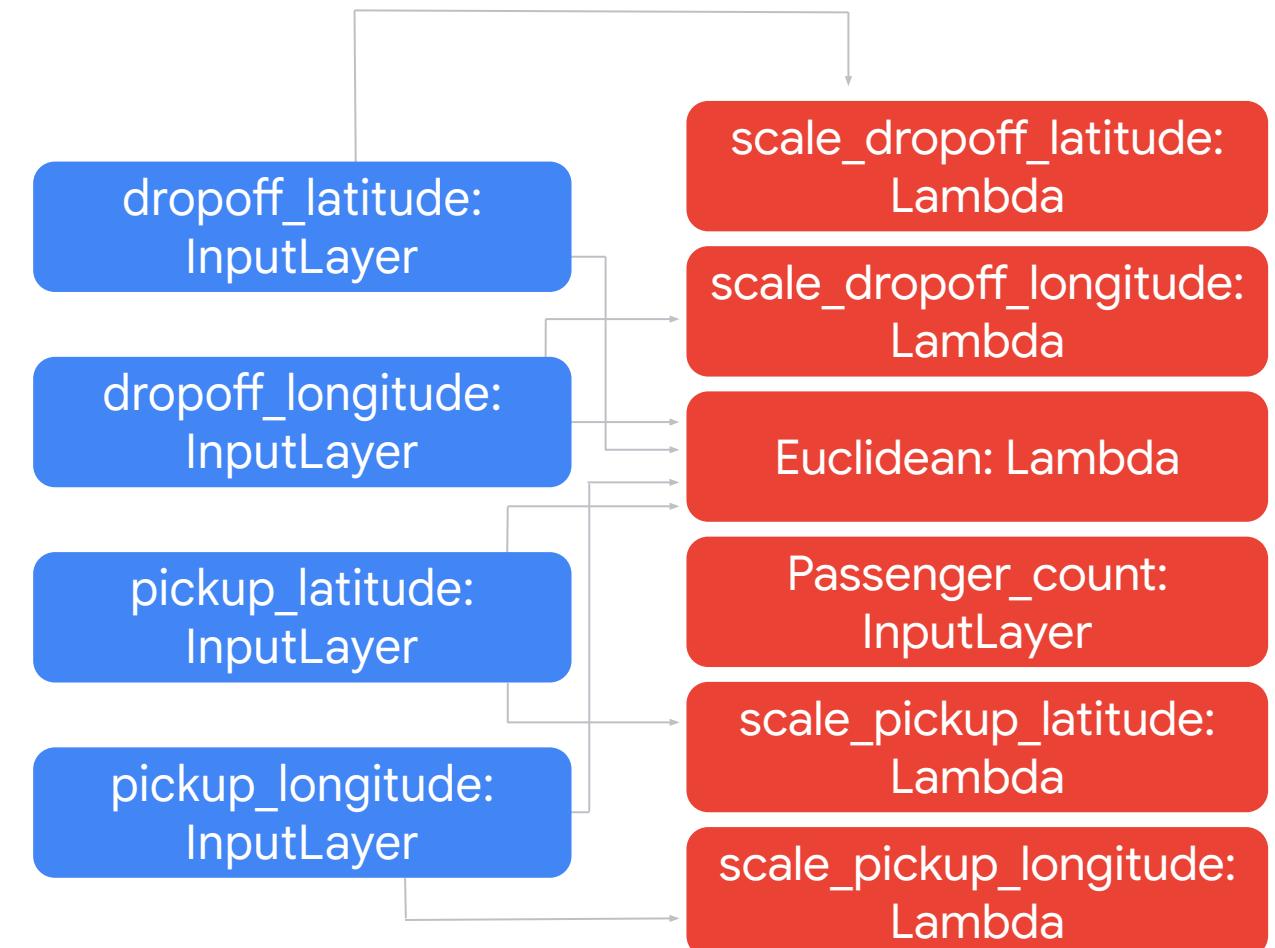
# Scaling latitude from range [37, 45] to [0, 1]
for lat_col in ['pickup_latitude', 'dropoff_latitude']:
    transformed[lat_col] = layers.Lambda(
        scale_latitude,
        name="scale_{}".format(lat_col))(inputs[lat_col])

# TODO 2
# add Euclidean distance
transformed['euclidean'] = layers.Lambda(
    euclidean,
    name='euclidean')([inputs['pickup_longitude'],
                      inputs['pickup_latitude'],
                      inputs['dropoff_longitude'],
                      inputs['dropoff_latitude']])

feature_columns['euclidean'] = fc.numeric_column('euclidean')
```

# Create new features using Lambda Layers

```
def euclidean(params):
    lon1, lat1, lon2, lat2 = params
    londiff = lon2 - lon1
    latdiff = lat2 - lat1
    return tf.sqrt(londiff*londiff + latdiff*latdiff)
...
transformed[ 'euclidean' ] = tf.keras.layers.Lambda(euclidean,
name='euclidean')([
    inputs['pickup_longitude'],
    inputs['pickup_latitude'],
    inputs['dropoff_longitude'],
    inputs['dropoff_latitude']
])
feature_columns[ 'euclidean' ] =
tf.feature_column.numeric_column('euclidean')
```



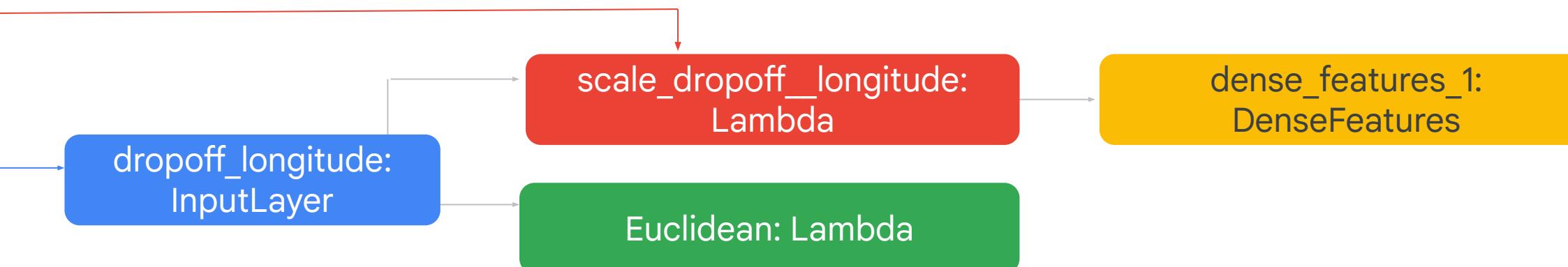
# Geolocation bucketized features

```
if True:

    # https://buganizer.corp.google.com/issues/135479527
    # featurecross lat, lon into nxn buckets, then embed
    nbuckets = NBUCKETS
    latbuckets = np.linspace(0, 1, nbuckets).tolist()
    lonbuckets = np.linspace(0, 1, nbuckets).tolist()
    b_plat = tf.feature_column.bucketized_column(feature_columns['pickup_latitude'], latbuckets)
    b_dlat = tf.feature_column.bucketized_column(feature_columns['dropoff_latitude'], latbuckets)
    b_plon = tf.feature_column.bucketized_column(feature_columns['pickup_longitude'], lonbuckets)
    b_dlon = tf.feature_column.bucketized_column(feature_columns['dropoff_longitude'], lonbuckets)
    ploc = tf.feature_column.crossed_column([b_plat, b_plon], nbuckets * nbuckets)
    dloc = tf.feature_column.crossed_column([b_dlat, b_dlon], nbuckets * nbuckets)
    pd_pair = tf.feature_column.crossed_column([ploc, dloc], nbuckets ** 4)
    feature_columns['pickup_and_dropoff'] = tf.feature_column.embedding_column(pd_pair, 100)
```

# Can replace feature columns by scaled values

```
for lon_col in ['pickup_longitude', 'dropoff_longitude']: # in range -70 to -78
    transformed[lon_col] = tf.keras.layers.Lambda(
        lambda x: (x+78)/8.0,
        name='scale_{}'.format(lon_col)
    )(inputs[lon_col])
```

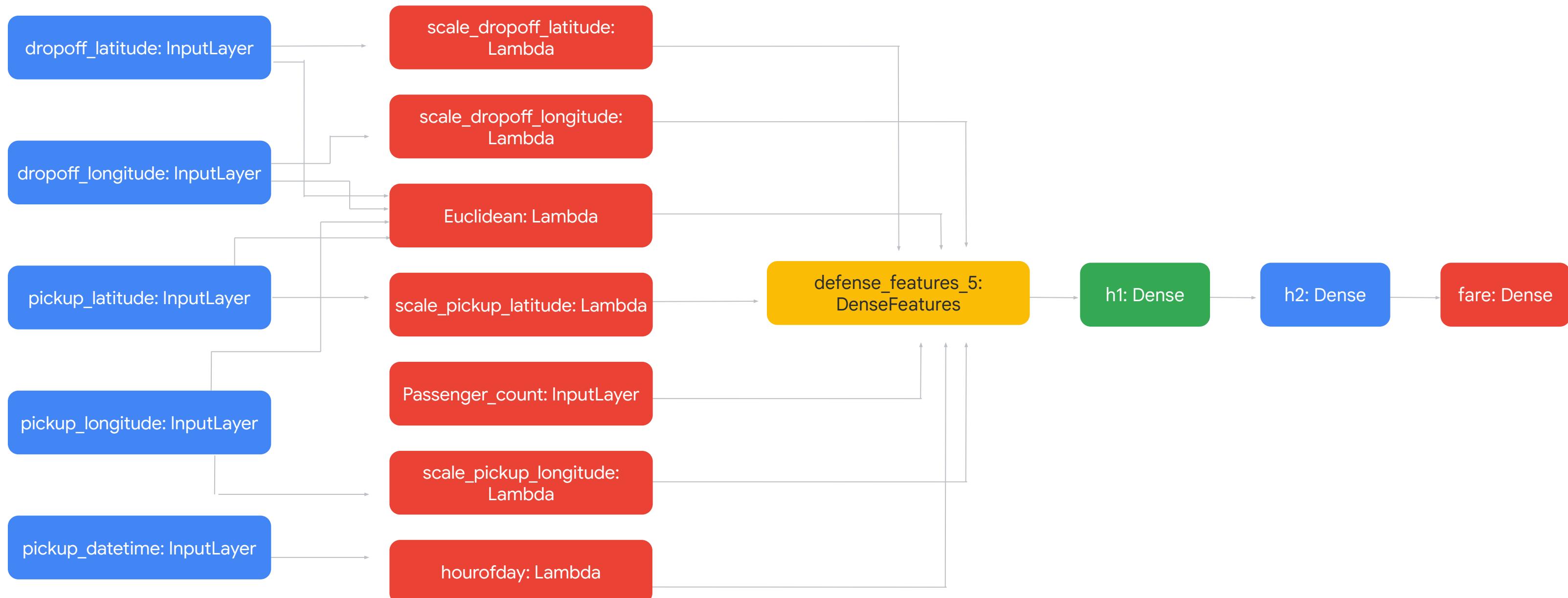


# Feature cross temporal features

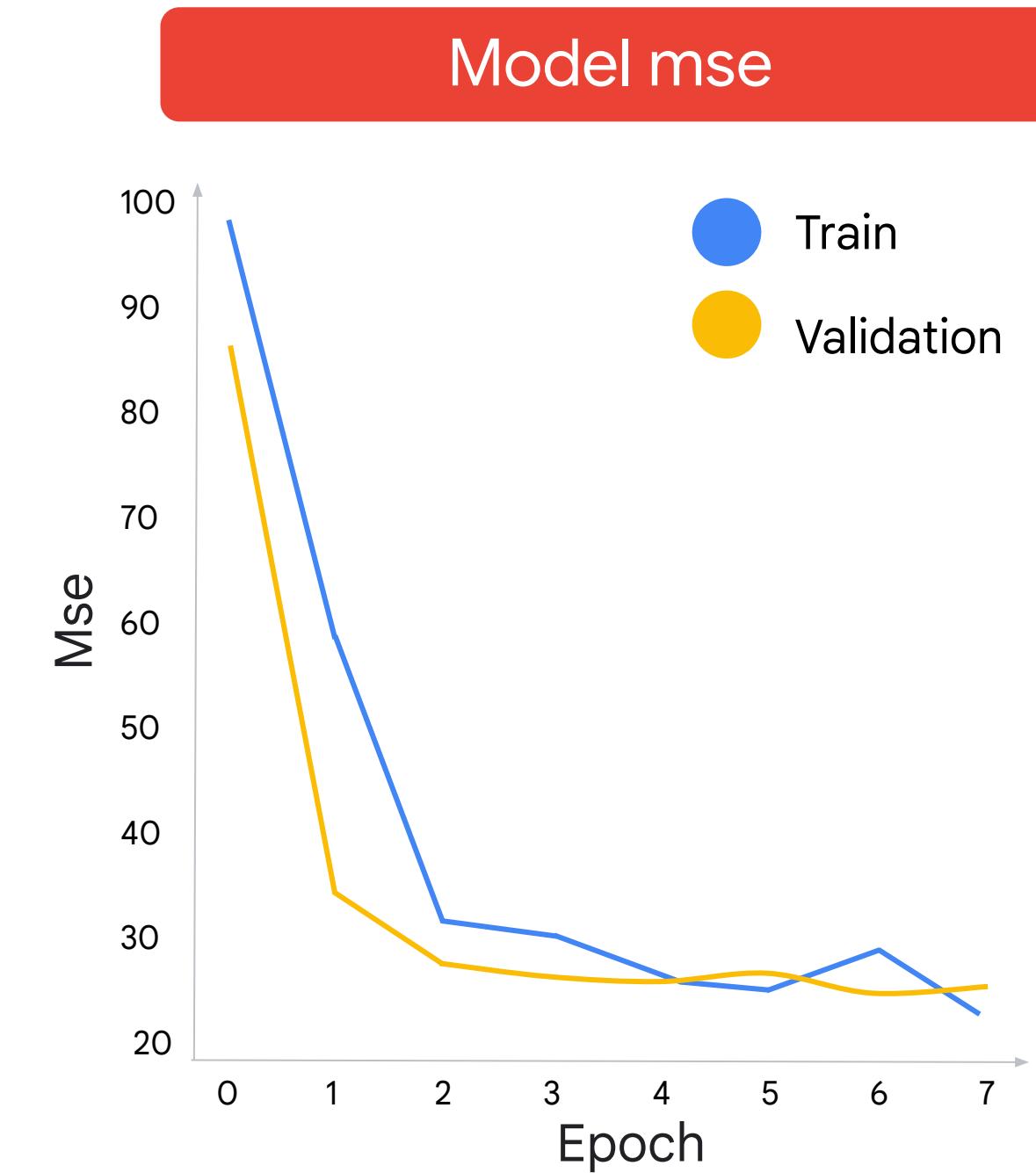
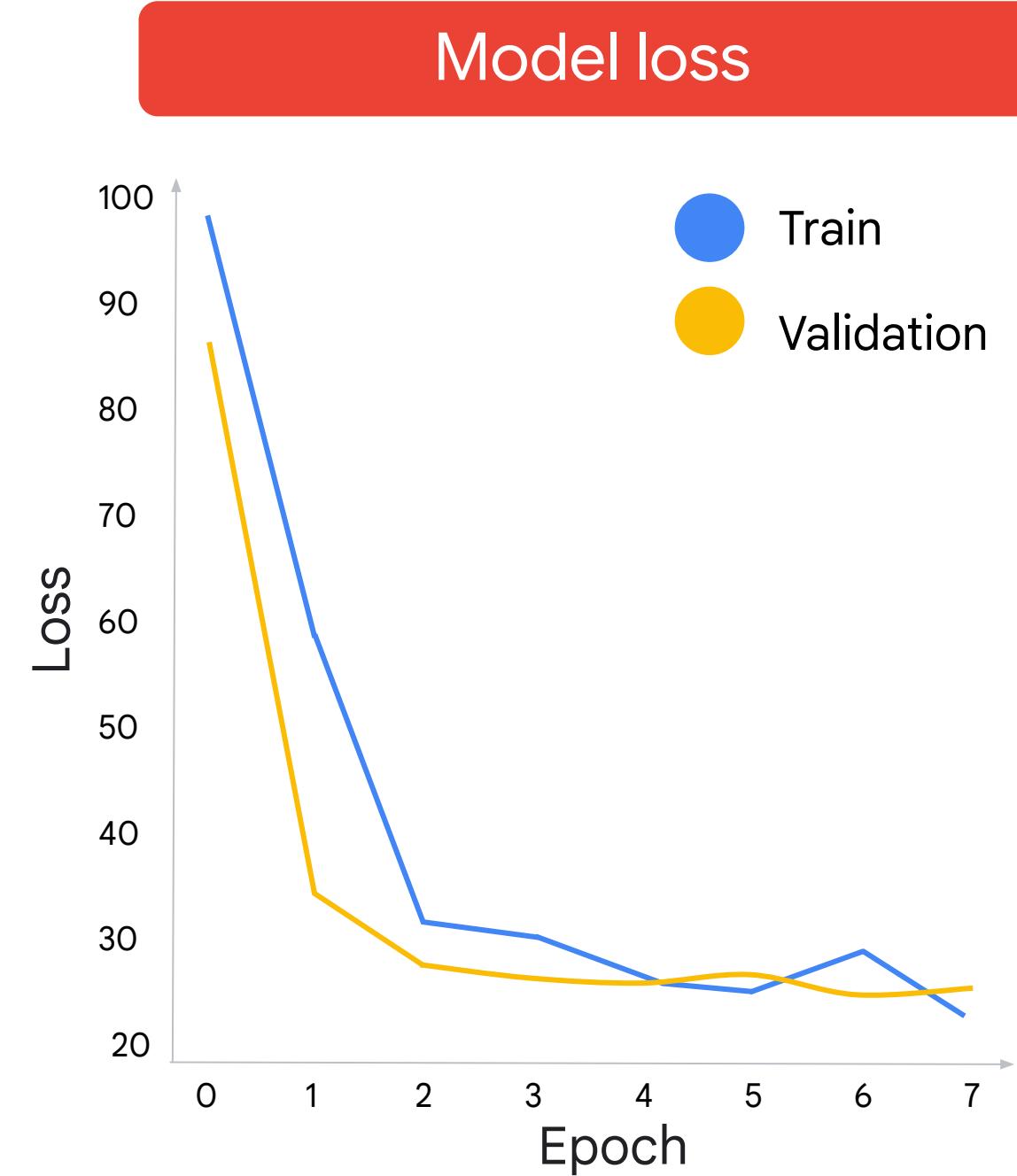
```
# hour of day from timestamp of form '2010-02-08 09:17:00+00:00'
    if True:
        transformed['hourofday'] = tf.keras.layers.Lambda(
            lambda x: tf.strings.to_number(tf.strings.substr(x, 11, 2),
out_type=tf.dtypes.int32),
            name='hourofday'
        )(inputs['pickup_datetime'])
        feature_columns['hourofday'] = tf.feature_column.indicator_column(
            tf.feature_column.categorical_column_with_identity('hourofday',
num_buckets=24))

    if False:
        # https://buganizer.corp.google.com/issues/137795281
        # day of week is hard because there is no TensorFlow function for date
handling
        transformed['dayofweek'] = tf.keras.layers.Lambda(
            lambda x: dayofweek(x),
            name='dayofweek_pyfun'
        )(inputs['pickup_datetime'])
        transformed['dayofweek'] = tf.keras.layers.Reshape((),
name='dayofweek')(transformed['dayofweek'])
        feature_columns['dayofweek'] = tf.feature_column.indicator_column(
            tf.feature_column.categorical_column_with_vocabulary_list(
                'dayofweek', vocabulary_list=DAYS))
```

# Visualize the model



# Model loss and MSE



# Model prediction

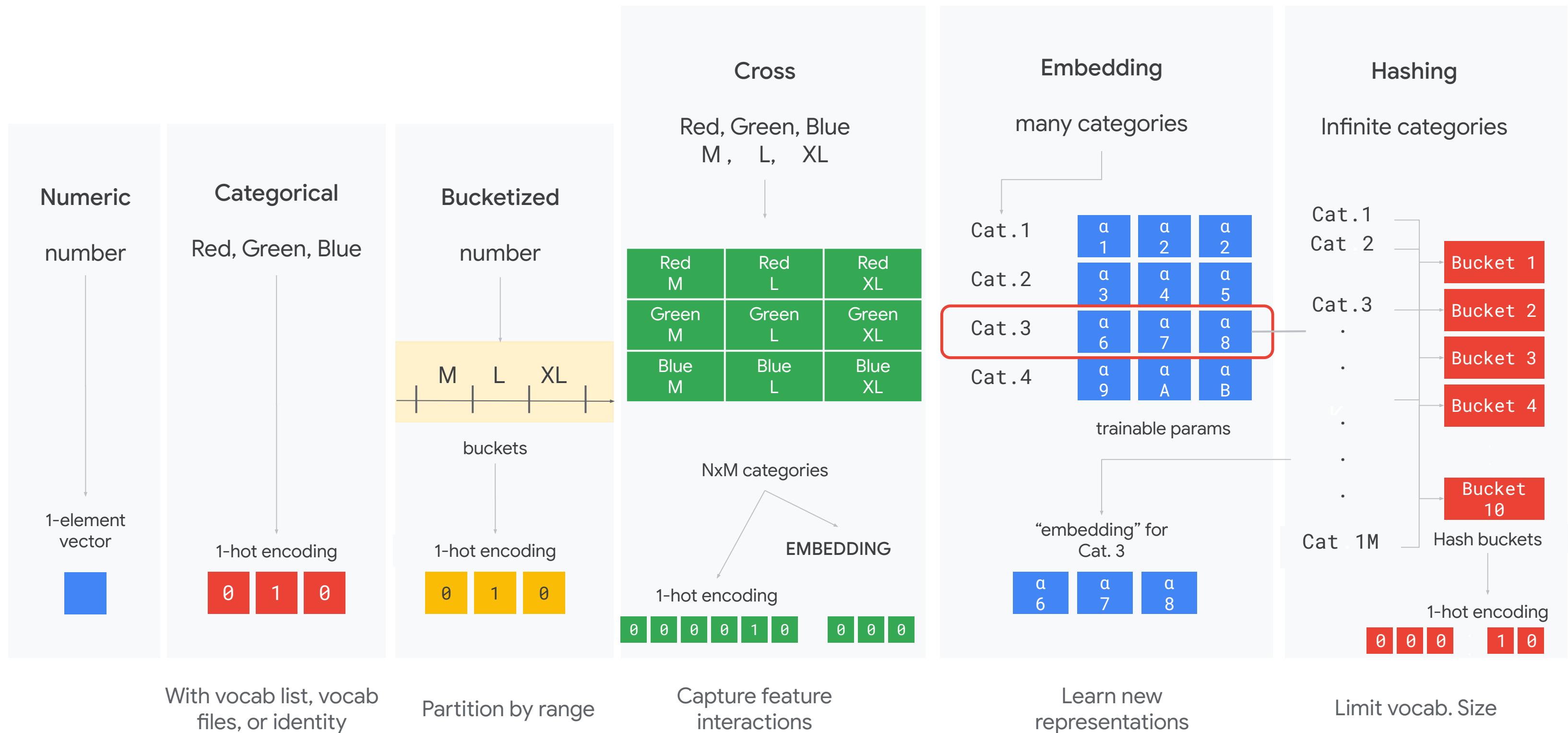
New York taxi fare at these coordinates is \$7.28

```
model.predict({  
    'pickup_longitude': tf.convert_to_tensor([-73.982683]),  
    'pickup_latitude': tf.convert_to_tensor([40.742104]),  
    'dropoff_longitude': tf.convert_to_tensor([-73.983766]),  
    'dropoff_latitude': tf.convert_to_tensor([40.755174]),  
    'passenger_count': tf.convert_to_tensor([3.0]),  
    'pickup_datetime': tf.convert_to_tensor(['2019-06-03  
04:21:29 UTC'], dtype=tf.string),  
}, steps=1)
```

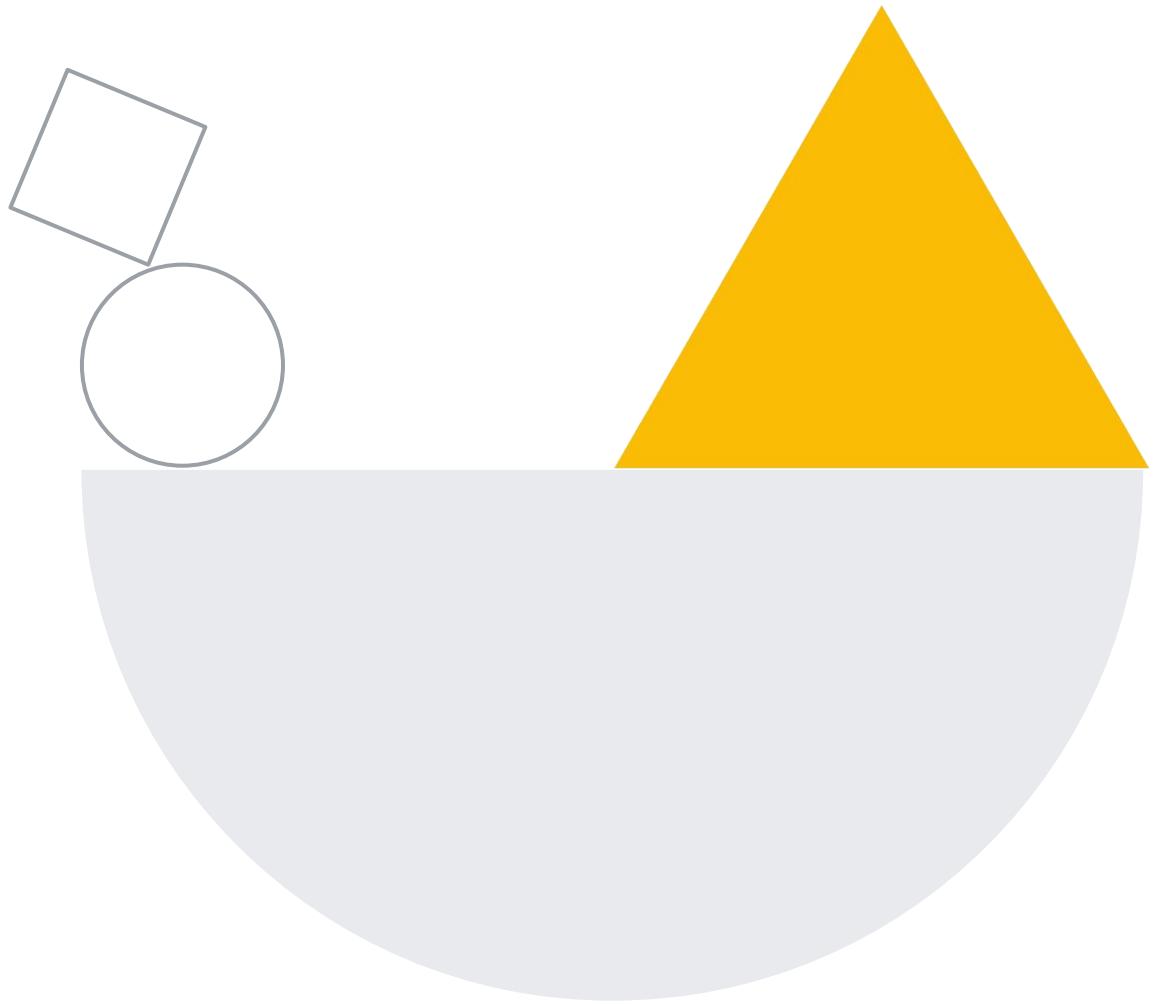
# RMSE: Summary table

Model	Taxi Fare	Description
Baseline	12.29	Baseline model - no feature engineering
Feature Engineered	07.28	Feature Engineered Model

# Feature columns



# Preprocessing and Feature Creation



# In this module, you learn to ...

01

Explain Apache Beam

02

Describe Dataflow



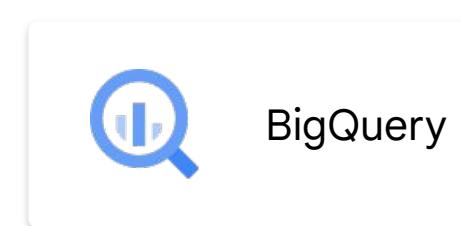
**Dataflow**  
**Apache Beam**

# Beam is a way to write elastic data processing pipelines

```
def packageHelp(record, keyword):
    count=0
    package_name='`'
    if record is not None:
        lines=record.split(`\n`)
        for line in lines:
            if line.startswith(keyword):
                package_name=line
            if `FIXME` in line or `TODO` in line
                count+=1
    packages = (getPackages(package_name))
    for p in packages:
        yield (p, count)
```

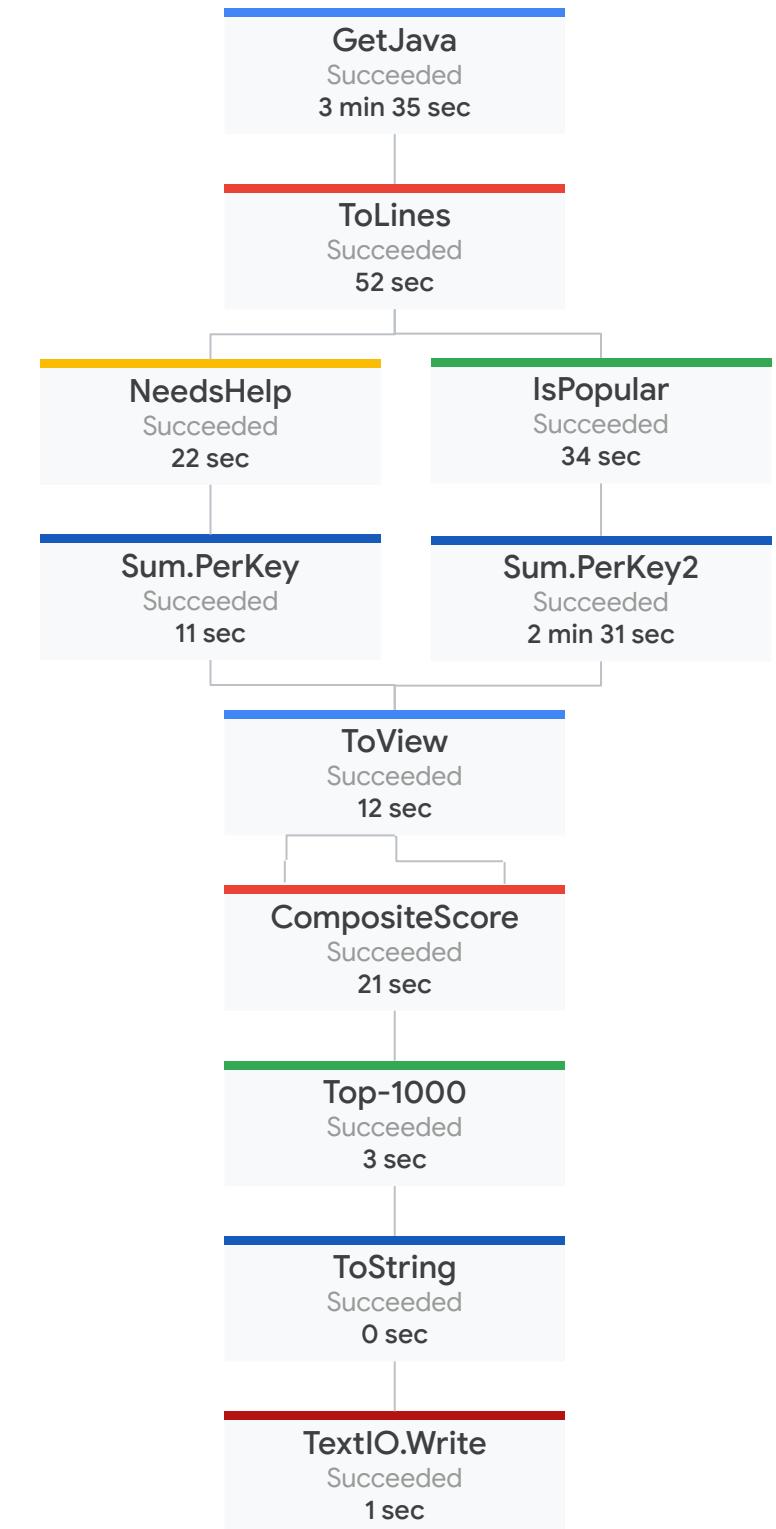
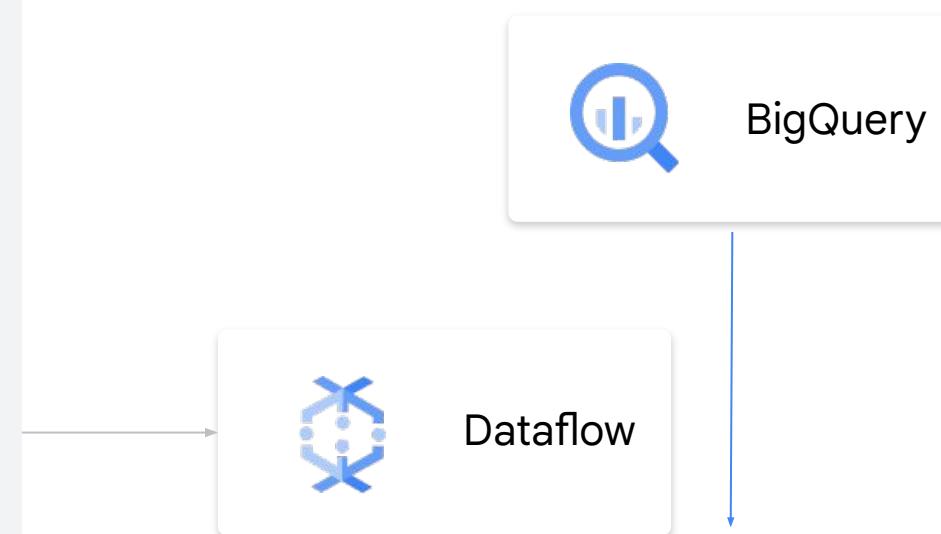


# Beam is a way to write elastic data processing pipelines



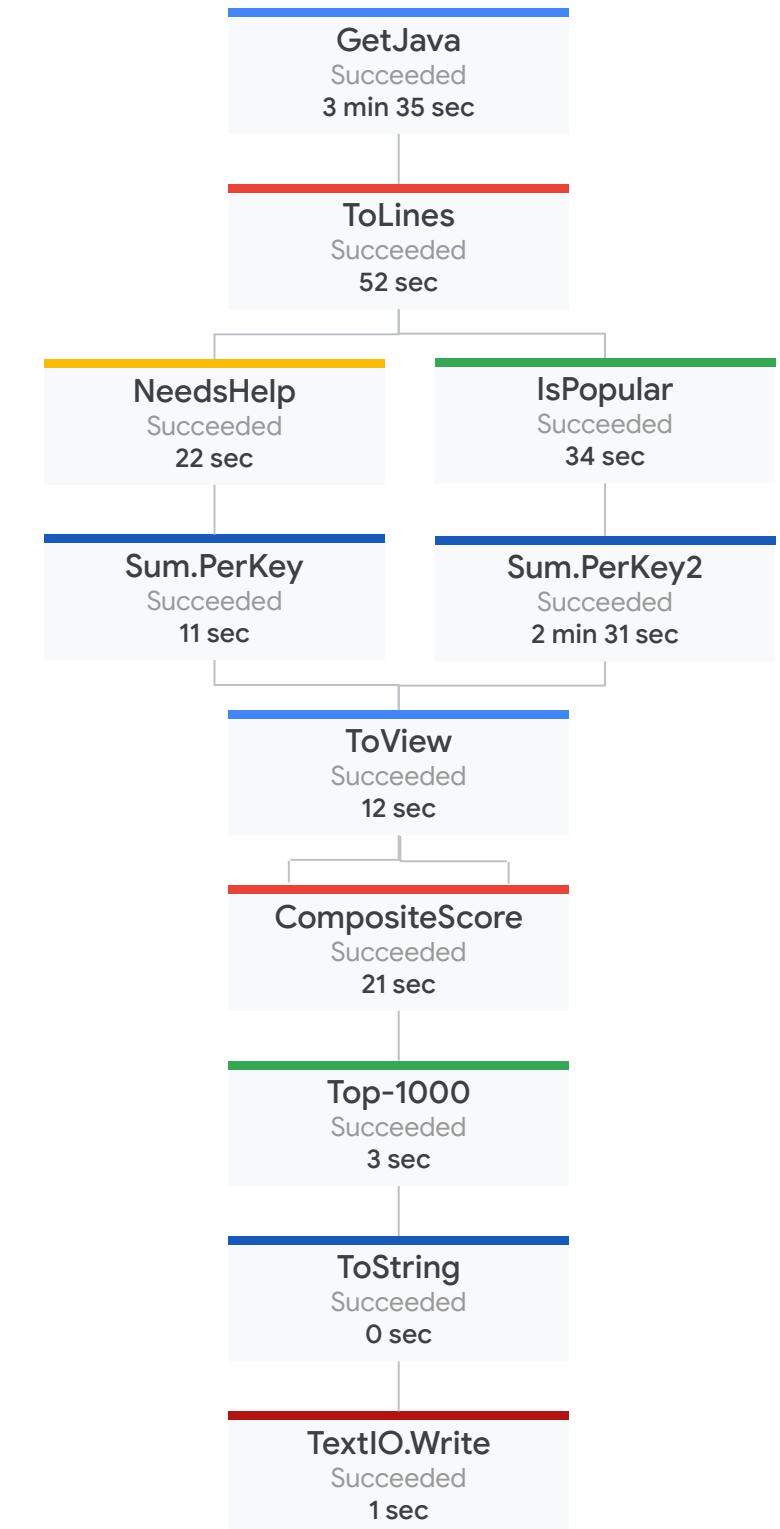
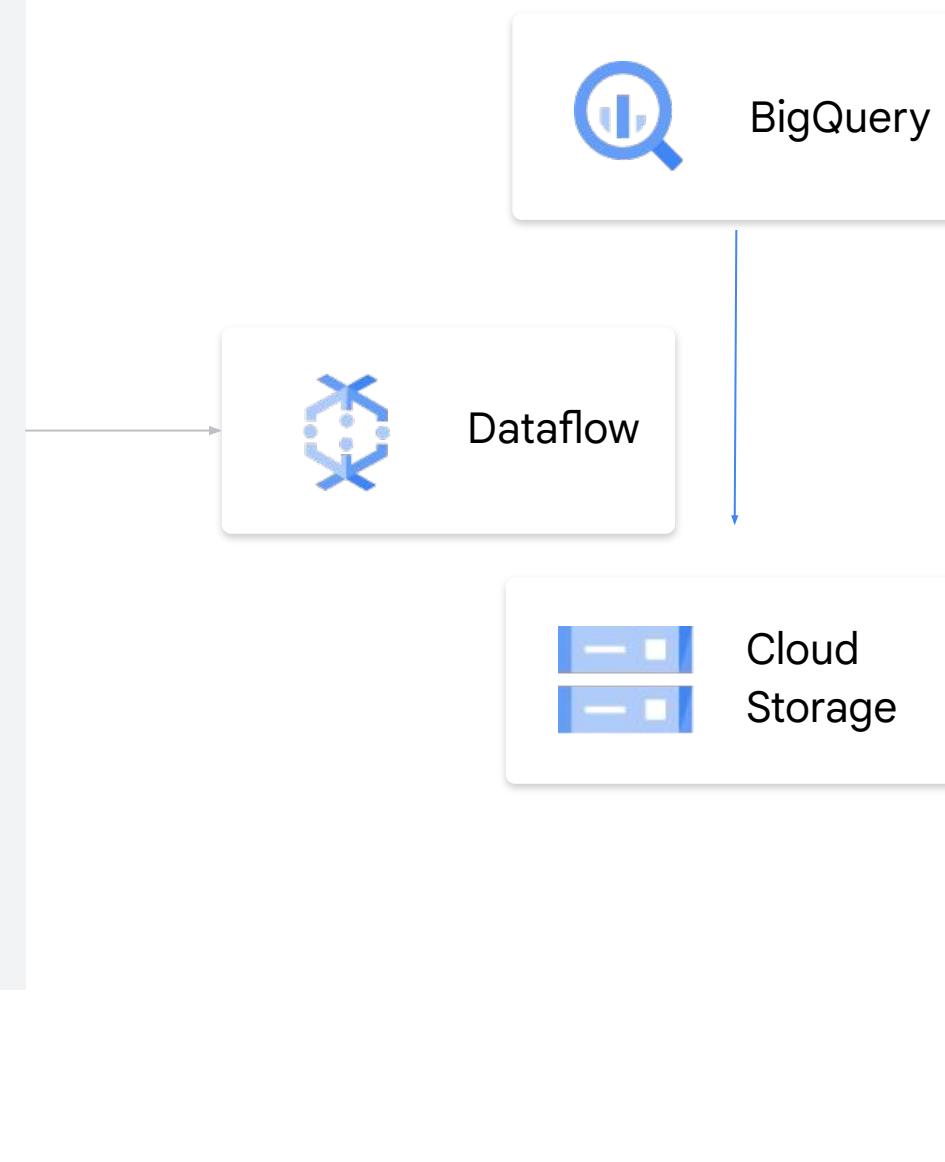
# Beam is a way to write elastic data processing pipelines

```
def packageHelp(record, keyword):
    count=0
    package_name=''
    if record is not None:
        lines=record.split(`\n`)
        for line in lines:
            if line.startswith(keyword):
                package_name=line
                if `FIXME` in line or `TODO` in line
                    count+=1
    packages = (getPackages(package_name))
    for p in packages:
        yield (p, count)
```

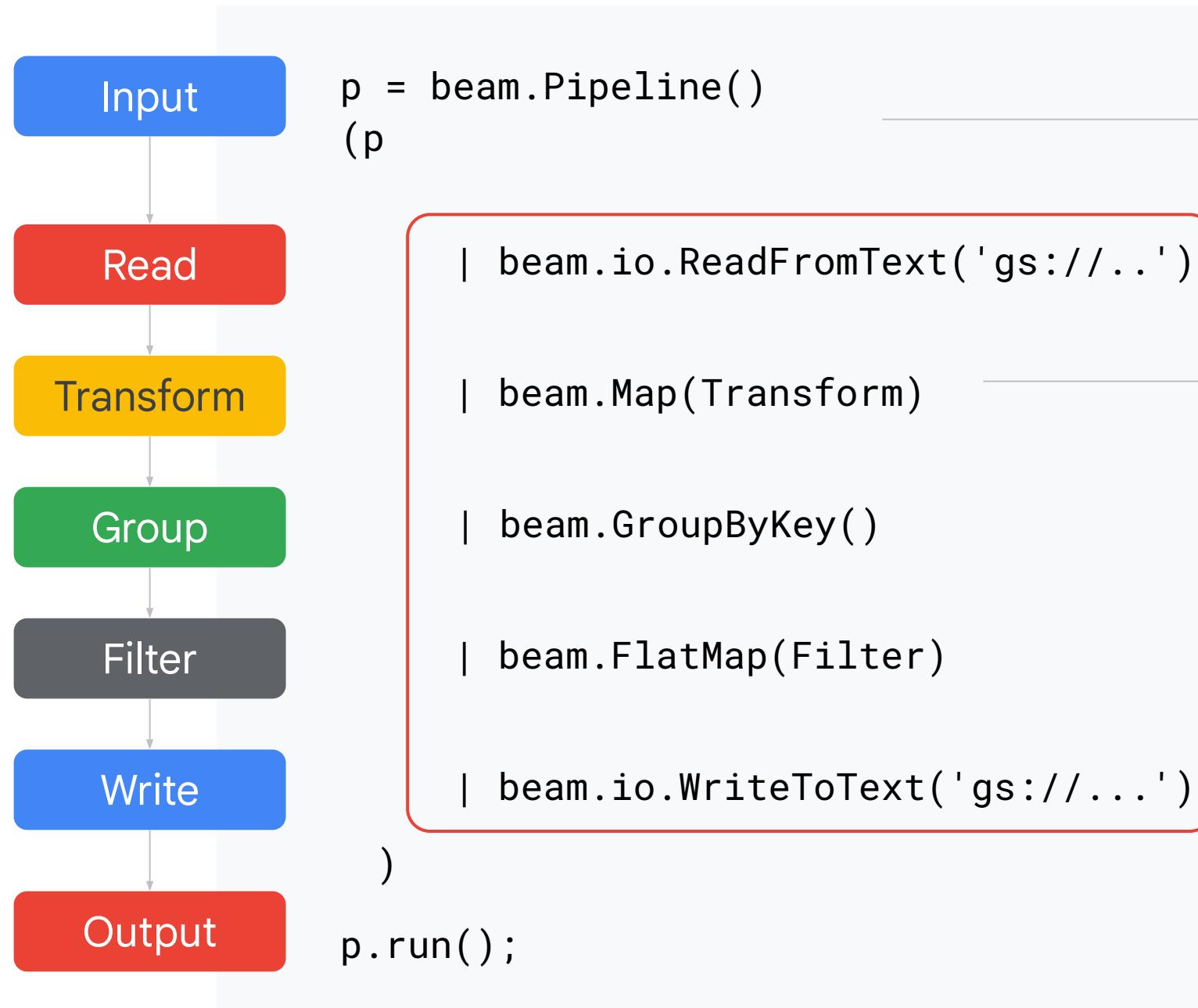


# Beam is a way to write elastic data processing pipelines

```
def packageHelp(record, keyword):
    count=0
    package_name=''
    if record is not None:
        lines=record.split(`\n`)
        for line in lines:
            if line.startswith(keyword):
                package_name=line
                if `FIXME` in line or `TODO` in line
                    count+=1
    packages = (getPackages(package_name))
    for p in packages:
        yield (p, count)
```



# Open-source API, Google infrastructure

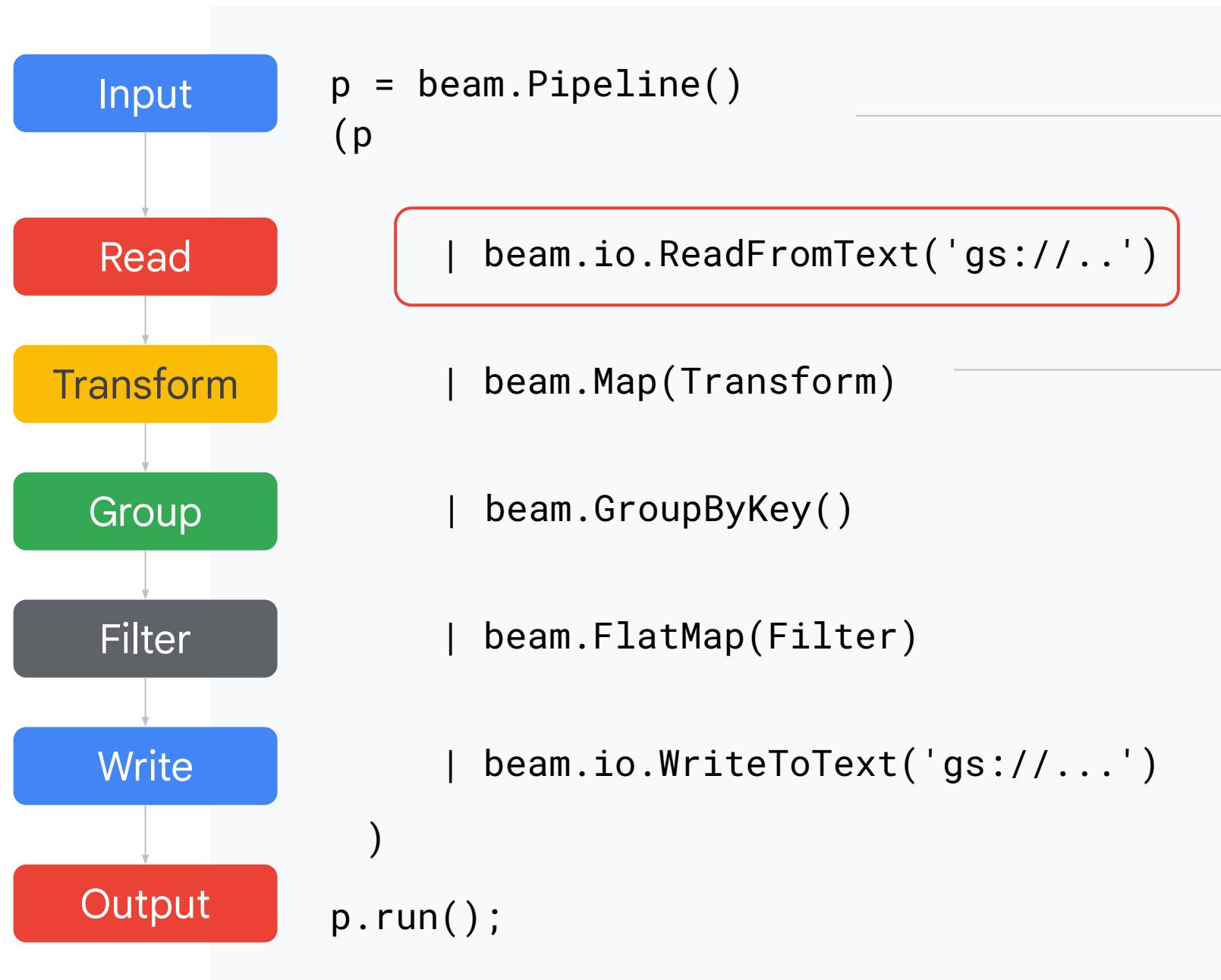


Open-source API  
(Apache Beam) can be  
executed on Flink, Spark,  
etc. also

Parallel tasks  
(autoscaled by  
execution framework)

```
def Transform(line):  
    return (count_words(line), 1)  
  
def Filter(key, values):  
    return key > 10
```

# Open-source API, Google infrastructure

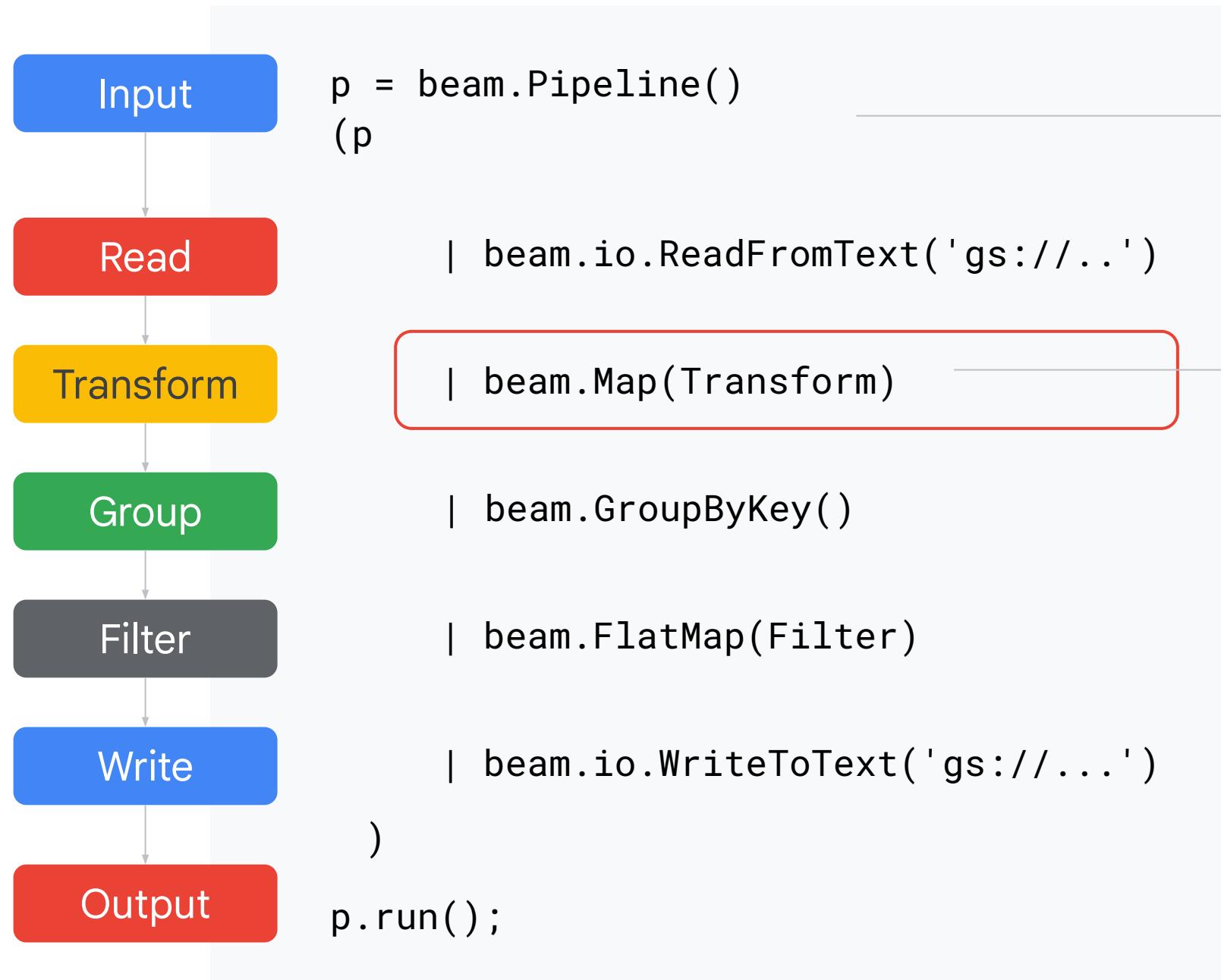


Open-source API  
(Apache Beam) can be  
executed on Flink, Spark,  
etc. also

Parallel tasks  
(autoscaled by  
execution framework)

```
def Transform(line):  
    return (count_words(line), 1)  
  
def Filter(key, values):  
    return key > 10
```

# Open-source API, Google infrastructure

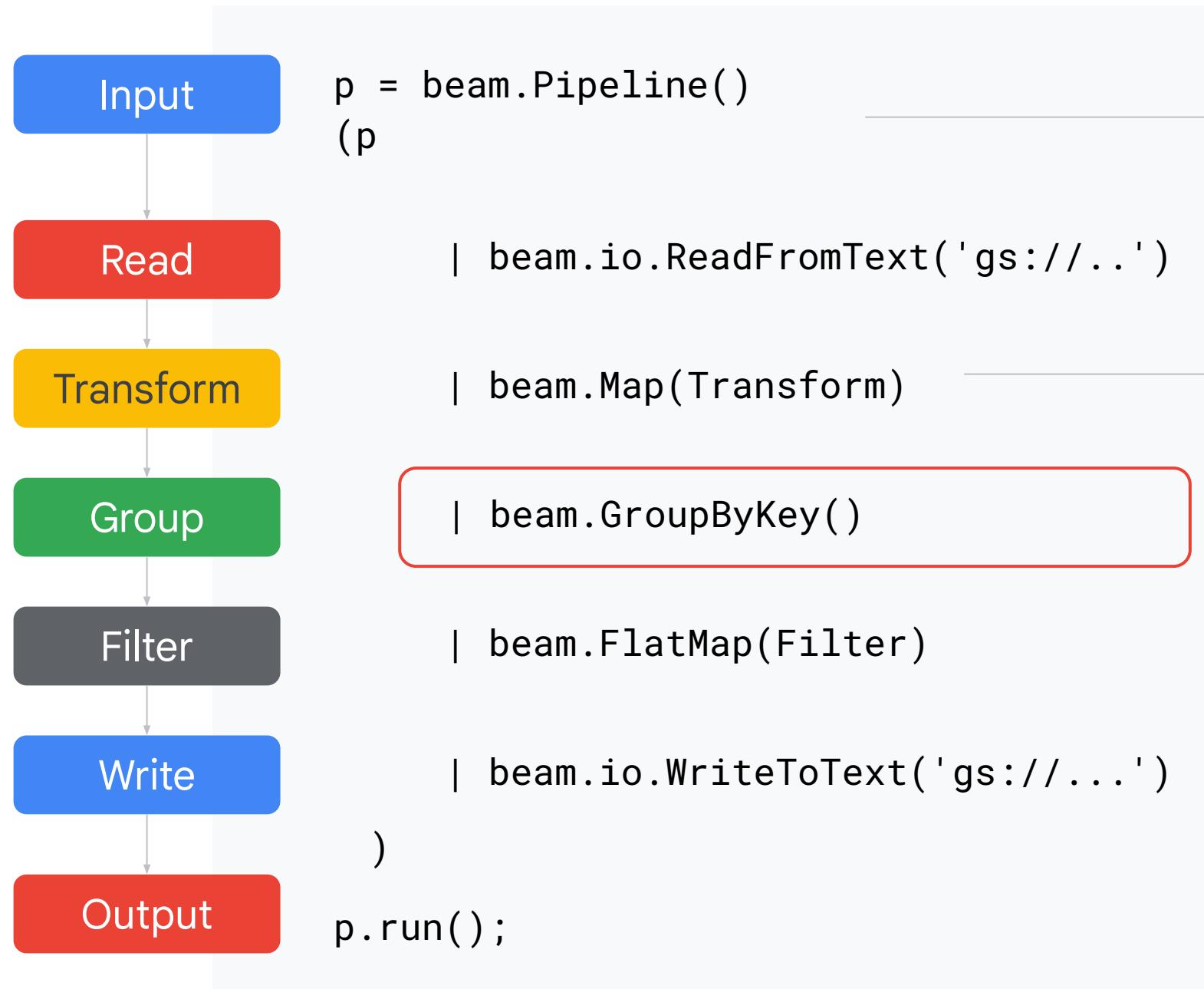


Open-source API  
(Apache Beam) can be  
executed on Flink, Spark,  
etc. also

Parallel tasks  
(autoscaled by  
execution framework)

```
def Transform(line):  
    return (count_words(line), 1)  
  
def Filter(key, values):  
    return key > 10
```

# Open-source API, Google infrastructure



Open-source API  
(Apache Beam) can be  
executed on Flink, Spark,  
etc. also

Parallel tasks  
(autoscaled by  
execution framework)

```
def Transform(line):
    return (count_words(line), 1)

def Filter(key, values):
    return key > 10
```

# Open-source API, Google infrastructure



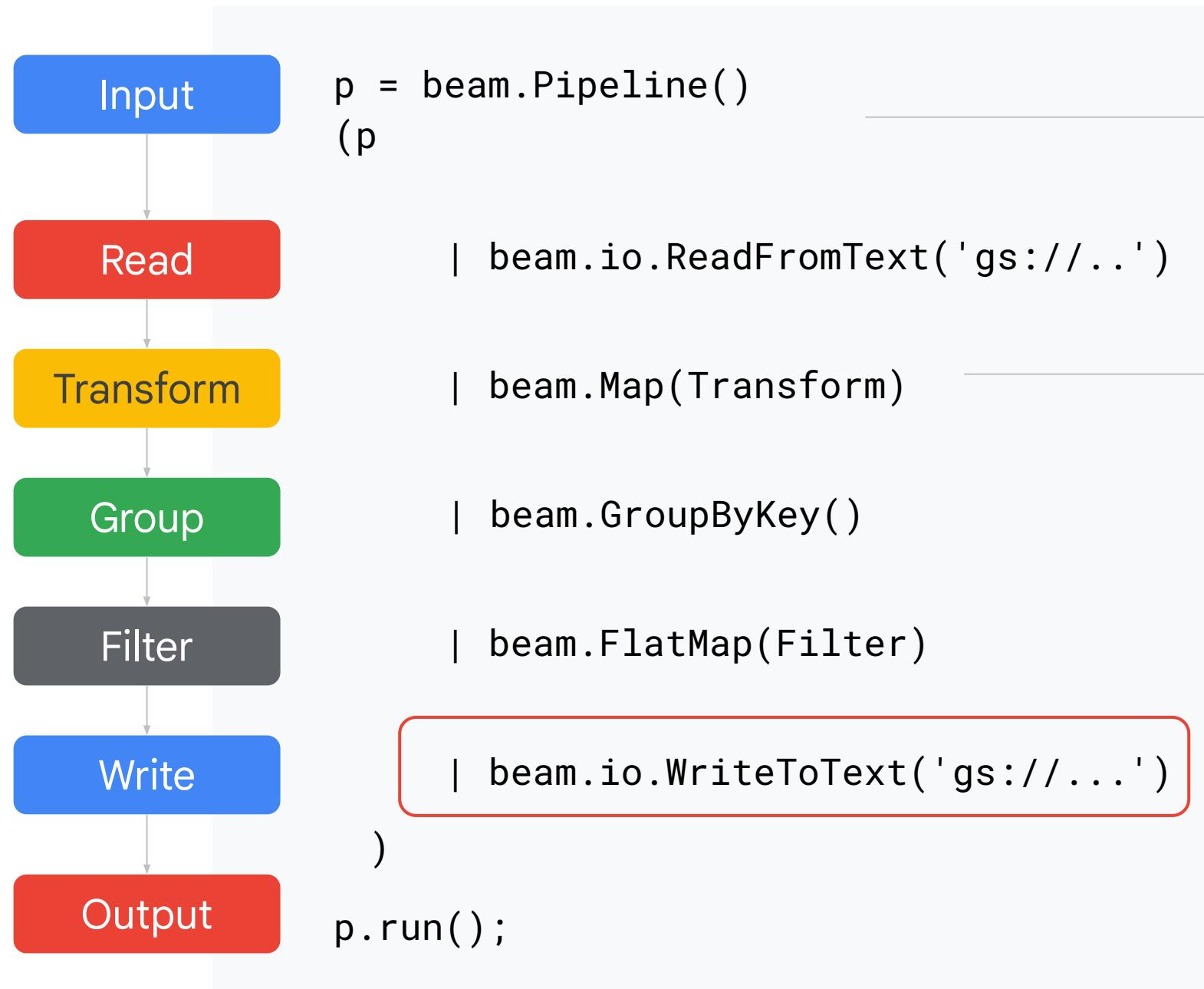
Open-source API  
(Apache Beam) can be  
executed on Flink, Spark,  
etc. also

Parallel tasks  
(autoscaled by  
execution framework)

```
def Transform(line):
    return (count_words(line), 1)

def Filter(key, values):
    return key > 10
```

# Open-source API, Google infrastructure



Open-source API  
(Apache Beam) can be  
executed on Flink, Spark,  
etc. also

Parallel tasks  
(autoscaled by  
execution framework)

```
def Transform(line):  
    return (count_words(line), 1)  
  
def Filter(key, values):  
    return key > 10
```

# Open-source API, Google infrastructure

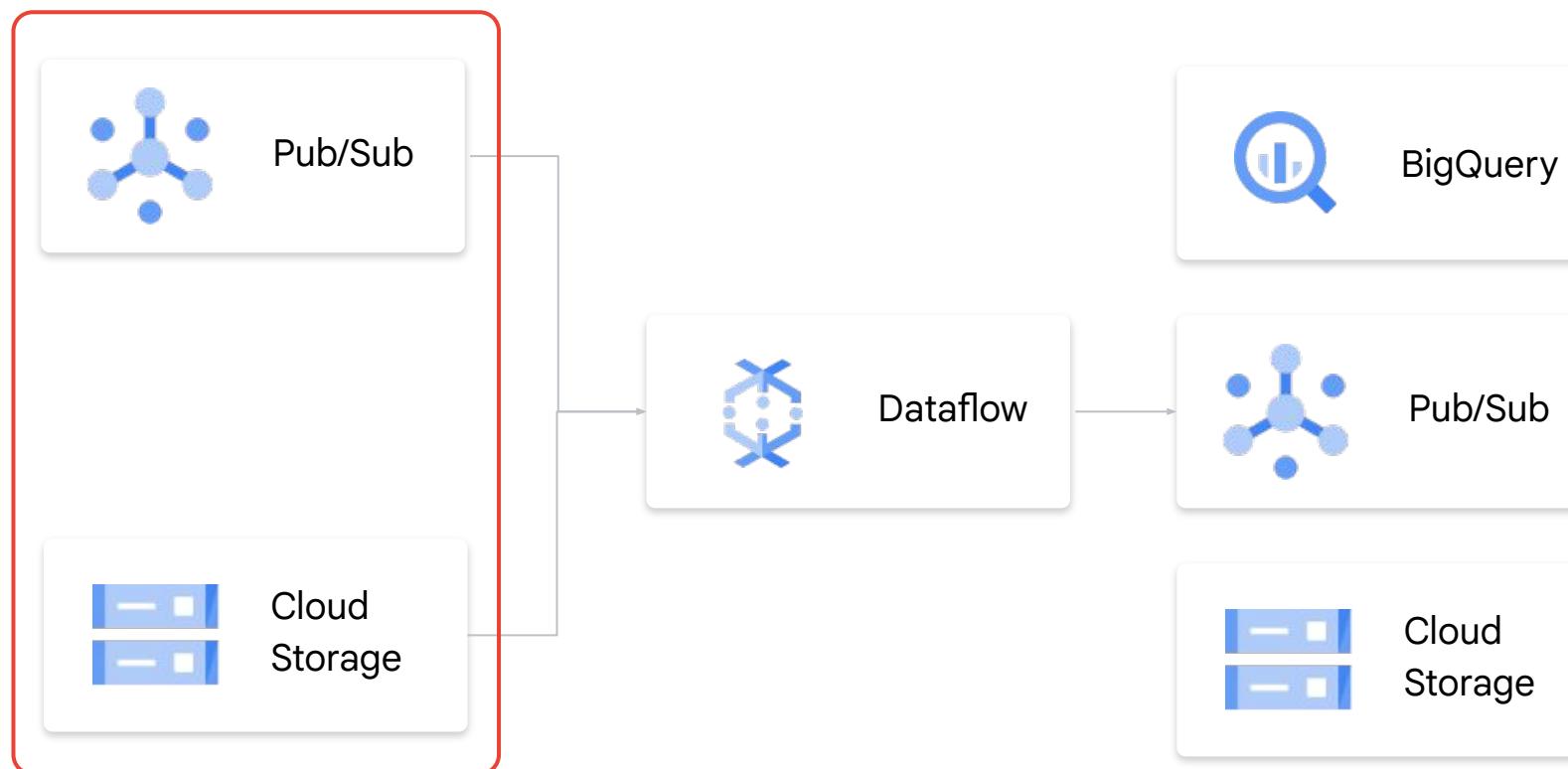


Open-source API  
(Apache Beam) can be  
executed on Flink, Spark,  
etc. also

Parallel tasks  
(autoscaled by  
execution framework)

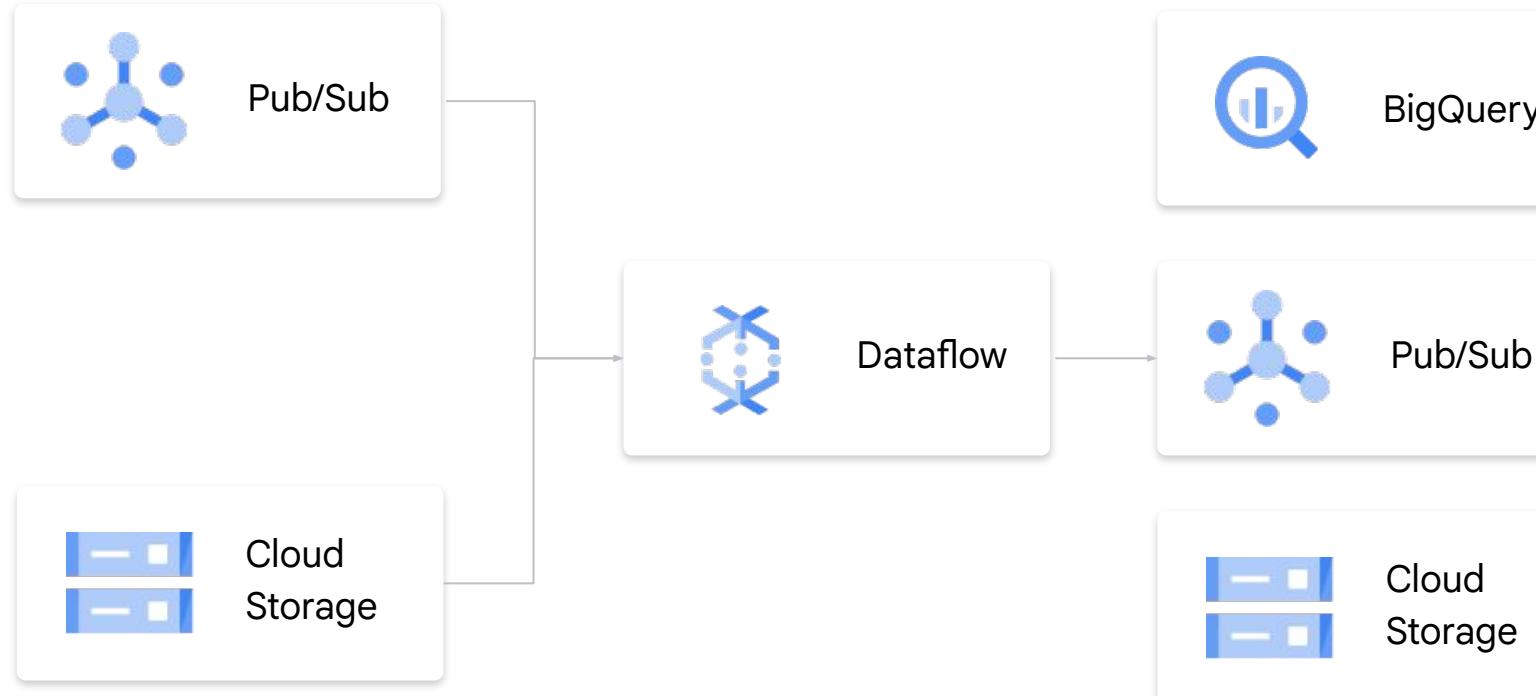
```
def Transform(line):  
    return (count_words(line), 1)  
def Filter(key, values):  
    return key > 10
```

# The code is the same between real-time and batch (Java)



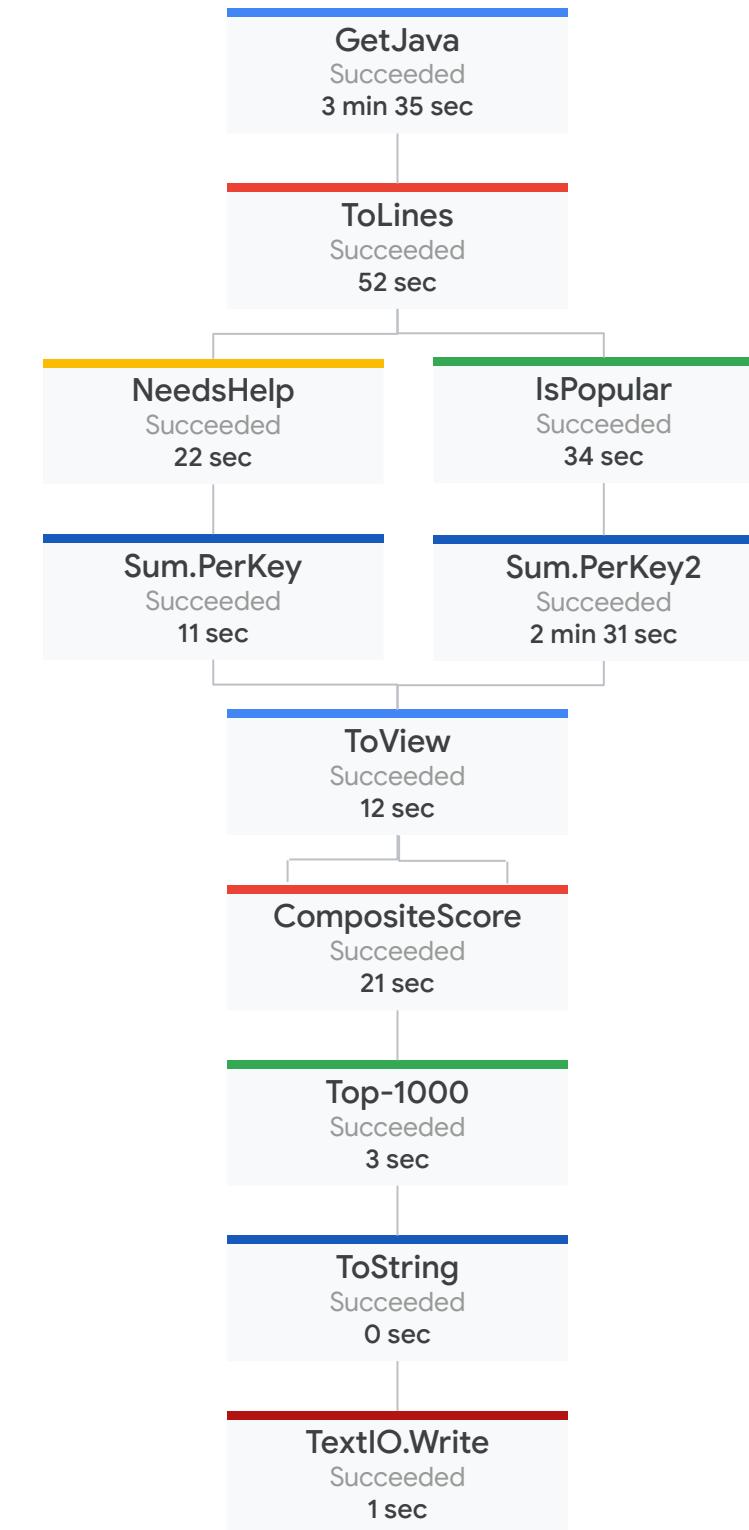
```
p = beam.Pipeline()  
(p  
| beam.io.ReadStringsFromPubSub('project/topic')  
| beam.WindowInto(SlidingWindows(60))  
| beam.Map(Transform)  
| beam.GroupByKey()  
| beam.FlatMap(Filter)  
| beam.io.WriteToBigQuery(table)  
)  
p.run()
```

# The code is the same between real-time and batch (Java)

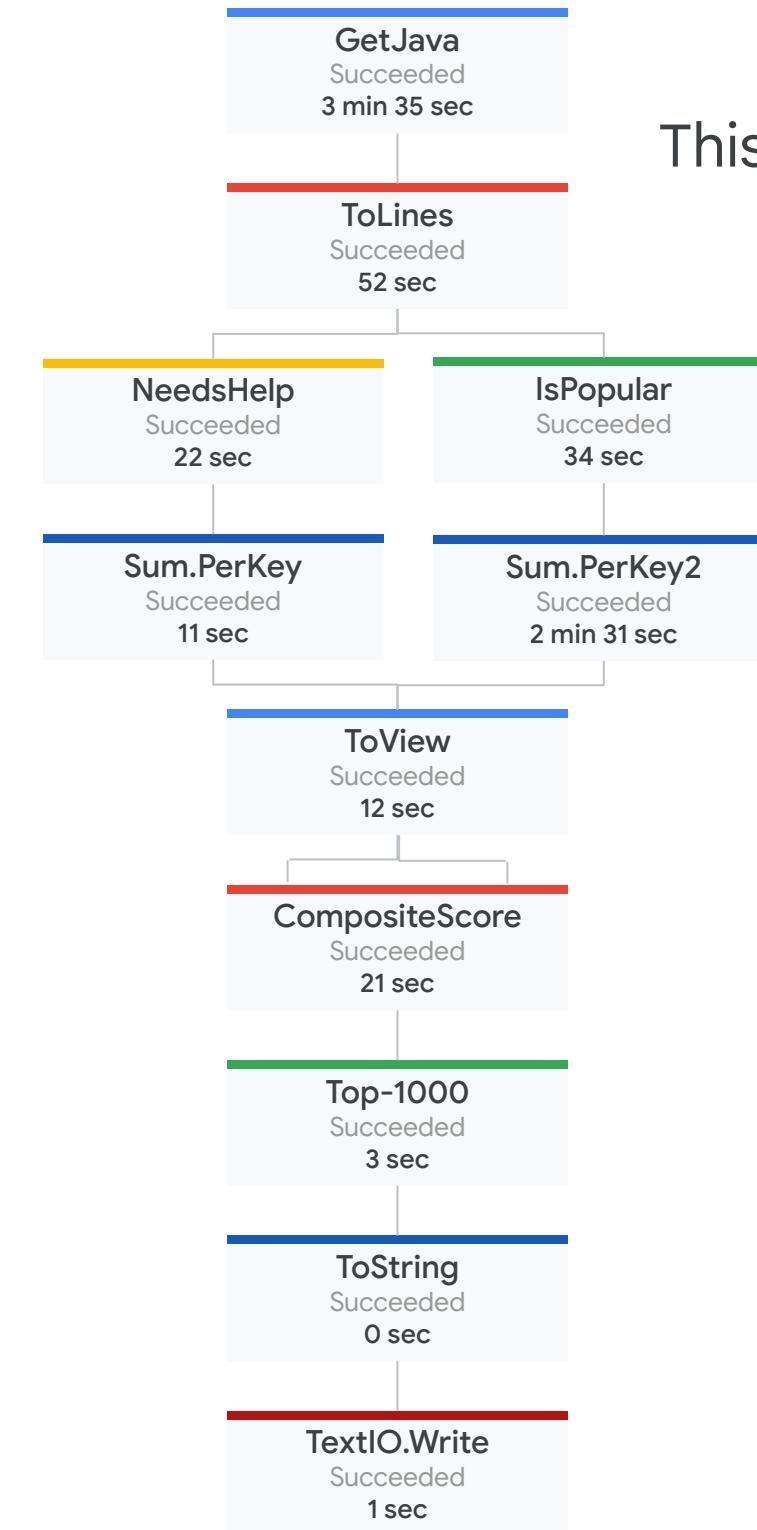


```
p = beam.Pipeline()
(p
| beam.io.ReadStringsFromPubSub('project/topic')
| beam.WindowInto(SlidingWindows(60))
| beam.Map(Transform)
| beam.GroupByKey()
| beam.FlatMap(Filter)
| beam.io.WriteToBigQuery(table)
)
p.run()
```

# Dataflow terms and concepts

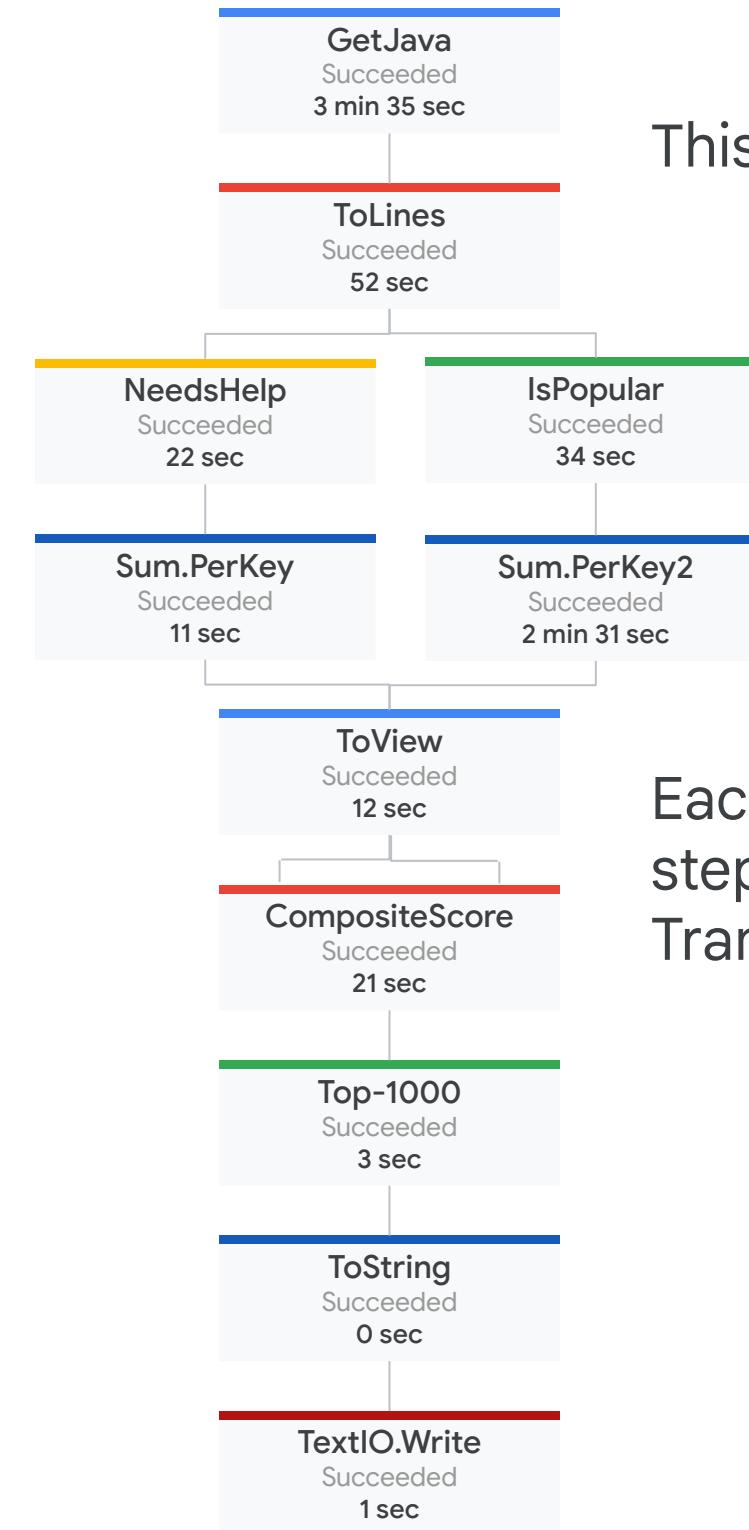


# Dataflow terms and concepts



This is a Source

# Dataflow terms and concepts



This is a Source

Each of these  
steps is a  
Transform

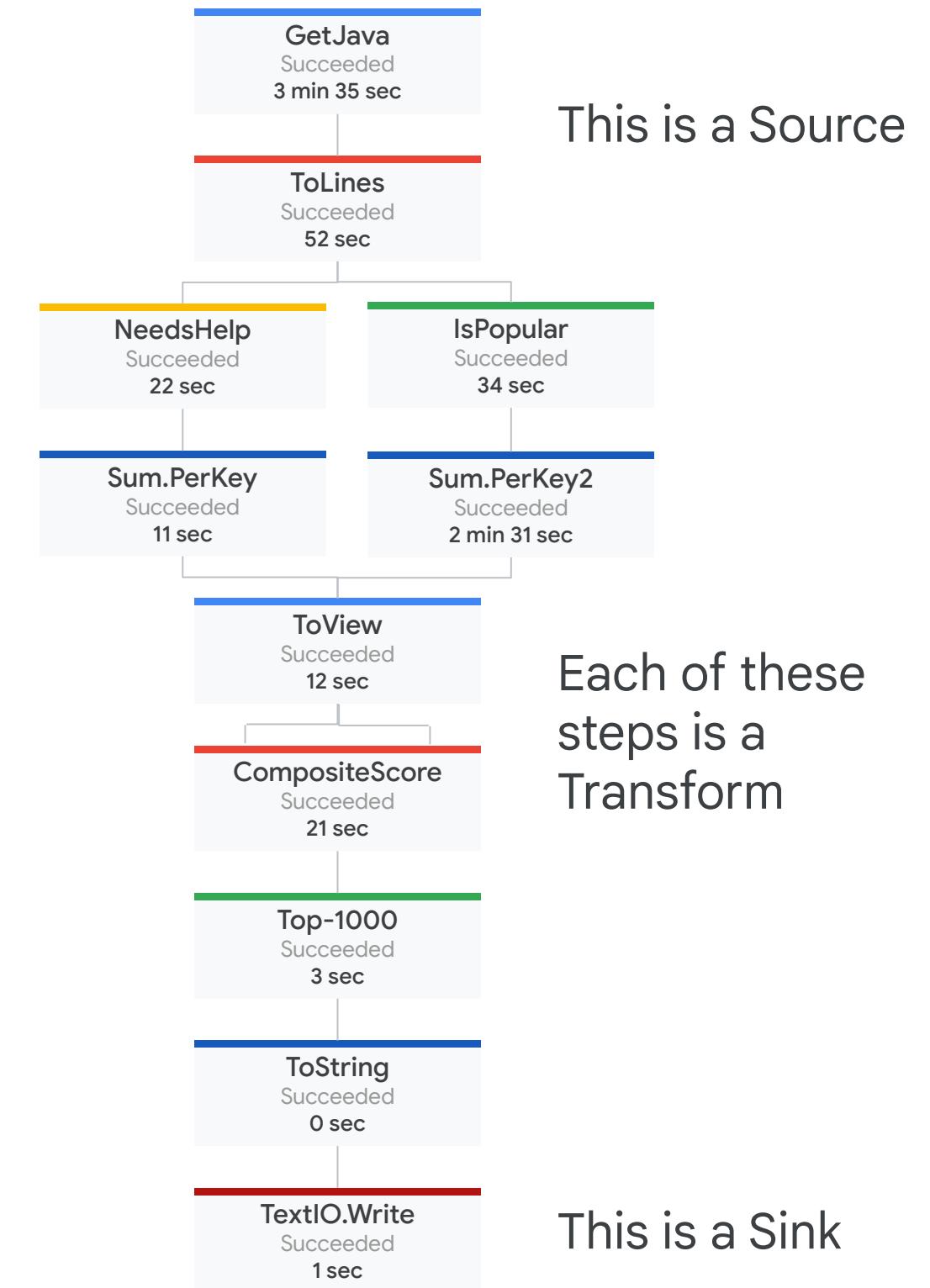
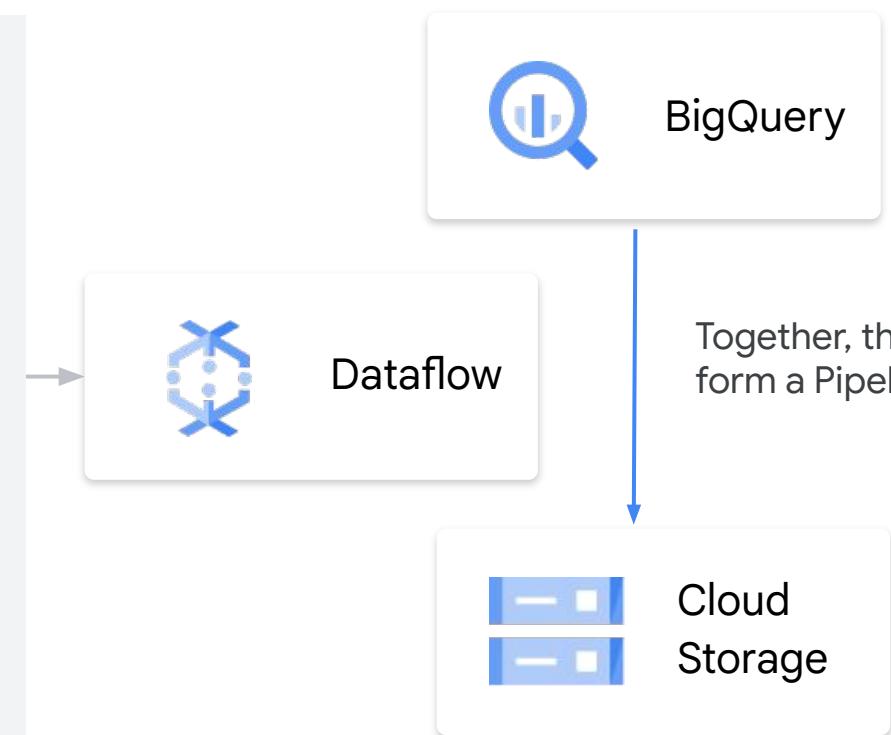
# Dataflow terms and concepts



# Dataflow terms and concepts

The Pipeline is executed on the cloud by a Runner; each step is elastically scaled

```
def packageHelp(record, keyword):
    count=0
    package_name``
    if record is not None:
        lines=record.split(`\n`)
        for line in lines:
            if line.startswith(keyword):
                package_name=line
            if `FIXME` in line or `TODO` in
                line    count+=1
    packages = (getPackages(package_name)
    for p in packages:
        yield (p, count)
```



# A Pipeline is a directed graph of steps

```
import apache_beam as beam
if __name__ == '__main__':
    # create a pipeline parameterized by commandline flags
    p = beam.Pipeline(argv=sys.argv)

    (p
        | 'Read' >> beam.io.ReadFromText('gs://...') # read input
        | 'CountWords' >> beam.FlatMap(lambda line: count_words(line))
        | 'Write' >> beam.io.WriteToText('gs://...') # write output
    )

    p.run() # run the pipeline
```

- Read in data, transform it, write out
- Can branch, merge, use if-then statements, etc.
- Pythonic syntax

# A Pipeline is a directed graph of steps

```
import apache_beam as beam
if __name__ == '__main__':
    # create a pipeline parameterized by commandline flags
    p = beam.Pipeline(argv=sys.argv)

    (p
        | 'Read' >> beam.io.ReadFromText('gs://...') # read input
        | 'CountWords' >> beam.FlatMap(lambda line: count_words(line))
        | 'Write' >> beam.io.WriteToText('gs://...') # write output
    )

p.run() # run the pipeline
```

- Read in data, transform it, write out
- Can branch, merge, use if-then statements, etc.
- Pythonic syntax

# A Pipeline is a directed graph of steps

```
import apache_beam as beam
if __name__ == '__main__':
    # create a pipeline parameterized by commandline flags
    p = beam.Pipeline(argv=sys.argv)

    (p
        | 'Read' >> beam.io.ReadFromText('gs://...') # read input
        | 'CountWords' >> beam.FlatMap(lambda line: count_words(line))
        | 'Write' >> beam.io.WriteToText('gs://...') # write output
    )

    p.run() # run the pipeline
```

- Read in data, transform it, write out
- Can branch, merge, use if-then statements, etc.
- Pythonic syntax

# A Pipeline is a directed graph of steps

```
import org.apache.beam.sdk.Pipeline; // etc.

public static void main(String[] args) {
    // Create a pipeline parameterized by commandline flags.
    Pipeline p =
        Pipeline.create(PipelineOptionsFactory.fromArgs(args));

    p.apply(TextIO.read().from("gs://...")) // Read input.
        .apply(new CountWords())           // Do some processing.
        .apply(TextIO.write().to("gs://...")); // Write output.

    // Run the pipeline.
    p.run();
}
```

- Read in data, transform it, write out
- Can branch, merge, use if-then statements, etc.

# A Pipeline is a directed graph of steps

```
import org.apache.beam.sdk.Pipeline; // etc.

public static void main(String[] args) {
    // Create a pipeline parameterized by commandline flags.
    Pipeline p =
        Pipeline.create(PipelineOptionsFactory.fromArgs(args));

    p.apply(TextIO.read().from("gs://...")) // Read input.
        .apply(new CountWords())           // Do some processing.
        .apply(TextIO.write().to("gs://...")); // Write output.

    // Run the pipeline.
    p.run();
}
```

- Read in data, transform it, write out
- Can branch, merge, use if-then statements, etc.

# Apply Transform to PCollection

- Data in a pipeline are represented by PCollection
- Supports parallel processing
- Not an in-memory collection; can be unbounded

```
lines = p | ...
```

Apply Transform to PCollection; returns PCollection

```
sizes = lines | 'Length' >> beam.Map(lambda line: len(line) )
```

# Ingesting data into a pipeline (Python)

- Read data from file system, Cloud Storage or BigQuery
- Text formats return String

```
lines = beam.io.ReadFromText('gs://.../input-*csv.gz')
```

BigQuery returns a TableRow

```
rows = beam.io.Read(beam.io.BigQuerySource(query='SELECT x, y, z' \
    'FROM [project:dataset.tablename]', project='PROJECT'))
```

# Ingesting data into a pipeline (Python)

- Read data from file system, Cloud Storage or BigQuery
- Text formats return String

```
lines = beam.io.ReadFromText('gs://.../input-*csv.gz')
```

BigQuery returns a TableRow

```
rows = beam.io.Read(beam.io.BigQuerySource(query='SELECT x, y, z' \
    'FROM [project:dataset.tablename]', project='PROJECT'))
```

# Ingesting data into a pipeline (Python)

- Read data from file system, Cloud Storage or BigQuery
- Text formats return String

```
lines = beam.io.ReadFromText('gs://.../input-*csv.gz')
```

BigQuery returns a TableRow

```
rows = beam.io.Read(beam.io.BigQuerySource(query='SELECT x, y, z' \
    'FROM [project:dataset.tablename]', project='PROJECT'))
```

# Can write data out to same formats (Python)

Write data to file system, Cloud Storage or BigQuery

```
beam.io.WriteToText(file_path_prefix='/data/output', file_name_suffix='.txt')
```

Can prevent sharding of output (do only if it is small)

```
beam.io.WriteToText(file_path_prefix='/data/output',
file_name_suffix='.txt', num_shards = 1)
```

The output must be a PCollection of Strings before writing out

# Executing pipeline (Python)

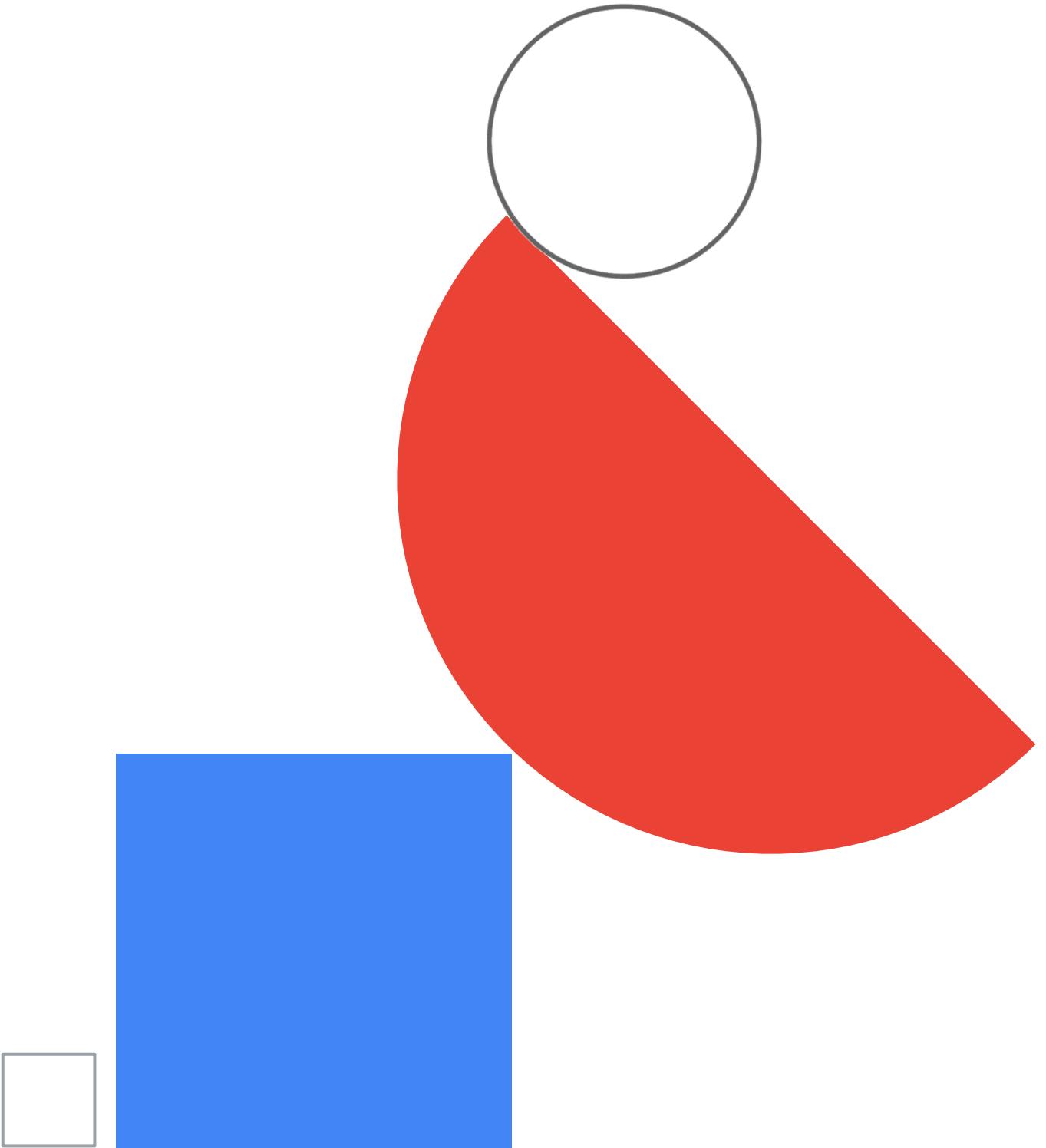
Simply running main() runs pipeline locally

```
python ./grep.py
```

To run on cloud, specify cloud parameters, and submit the job to Dataflow

```
python ./grep.py \
    --project=$PROJECT \
    --job_name=myjob \
    --staging_location=gs://$BUCKET/staging/ \
    --temp_location=gs://$BUCKET/staging/ \
    --runner=DataflowRunner
```

# Feature Crosses - TensorFlow Playground



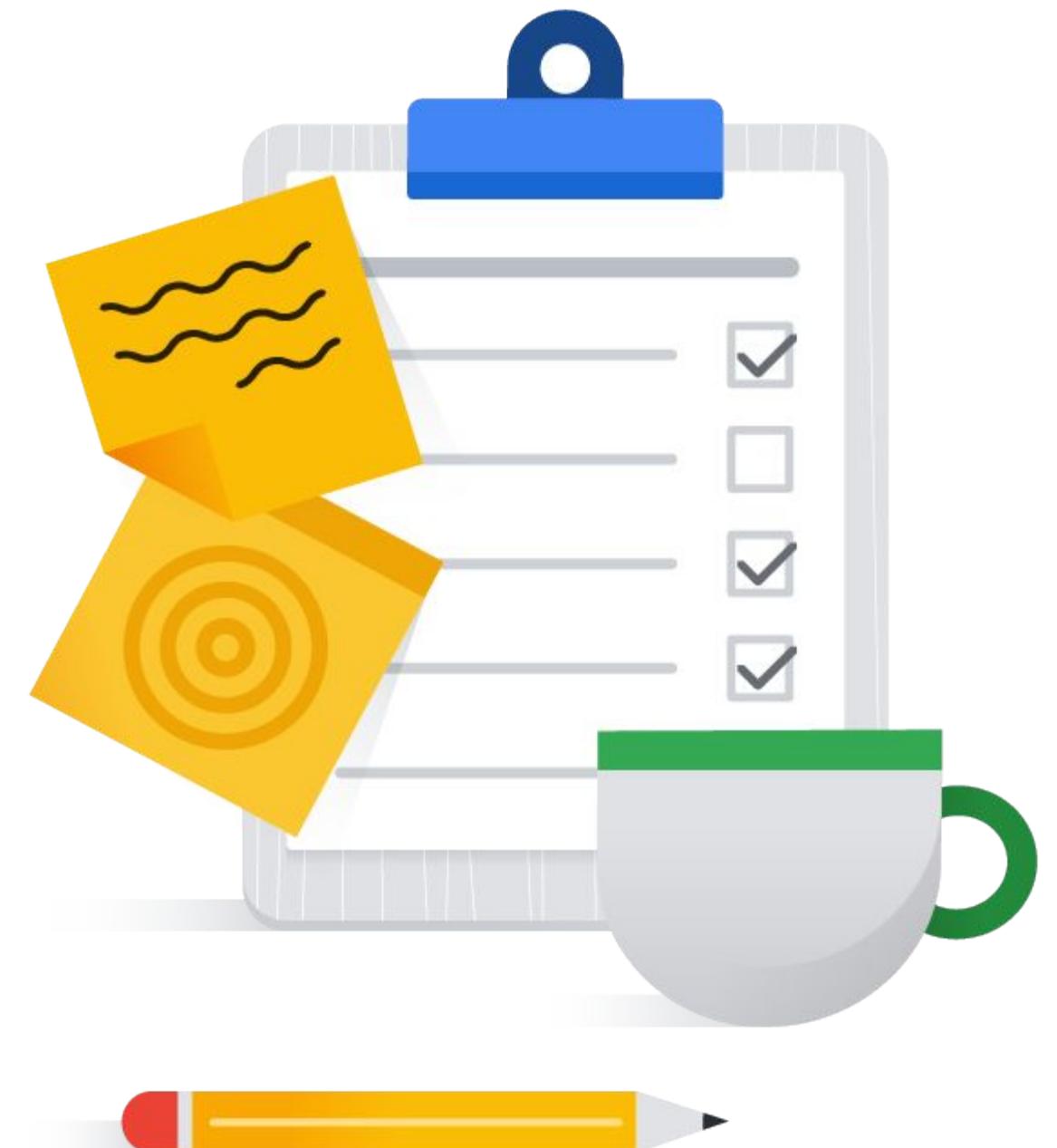
# In this module, you learn to ...

01

Describe Feature Crosses

02

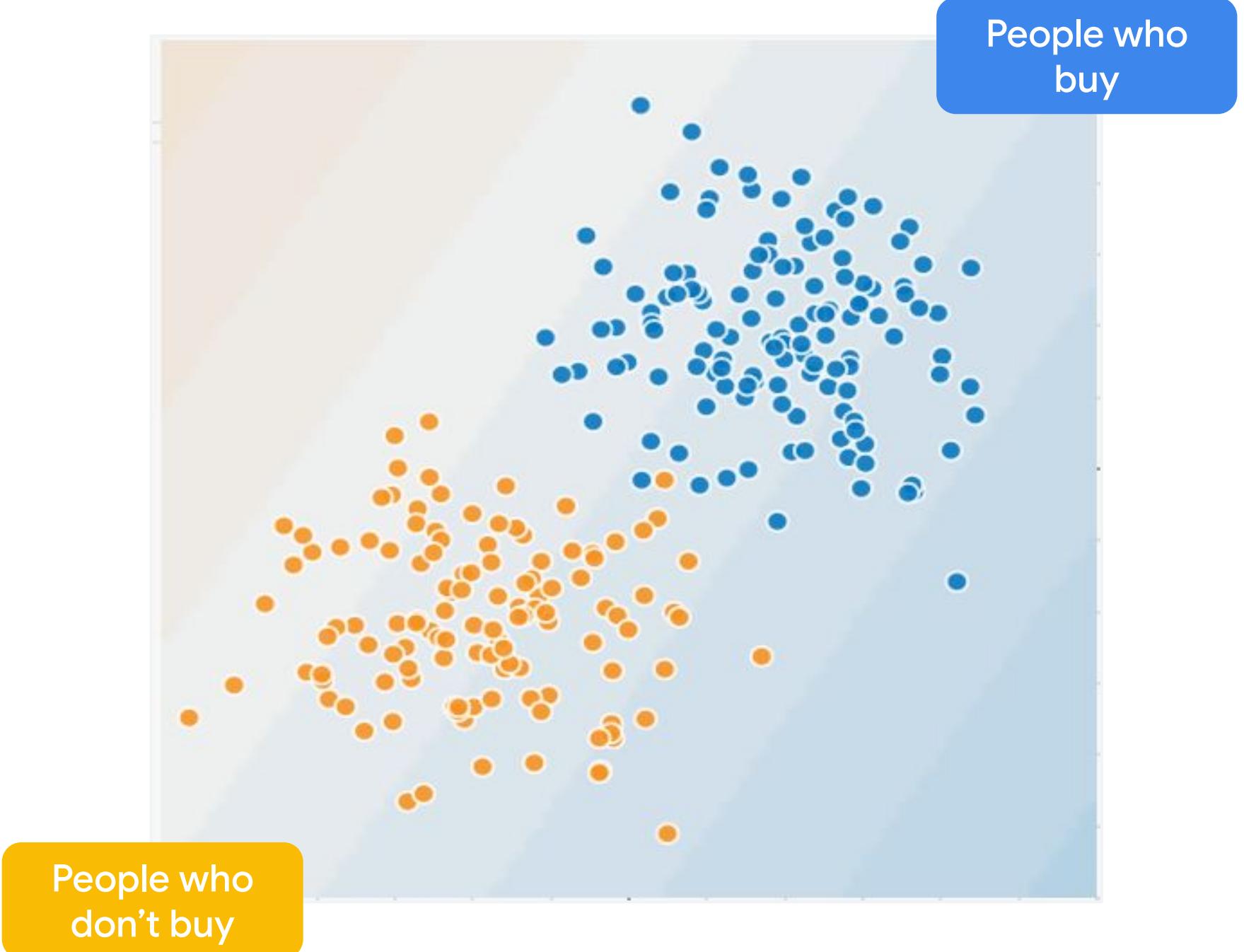
Use Feature Crosses to create a good classifier



**Can you draw a line  
that separates these  
two classes?**



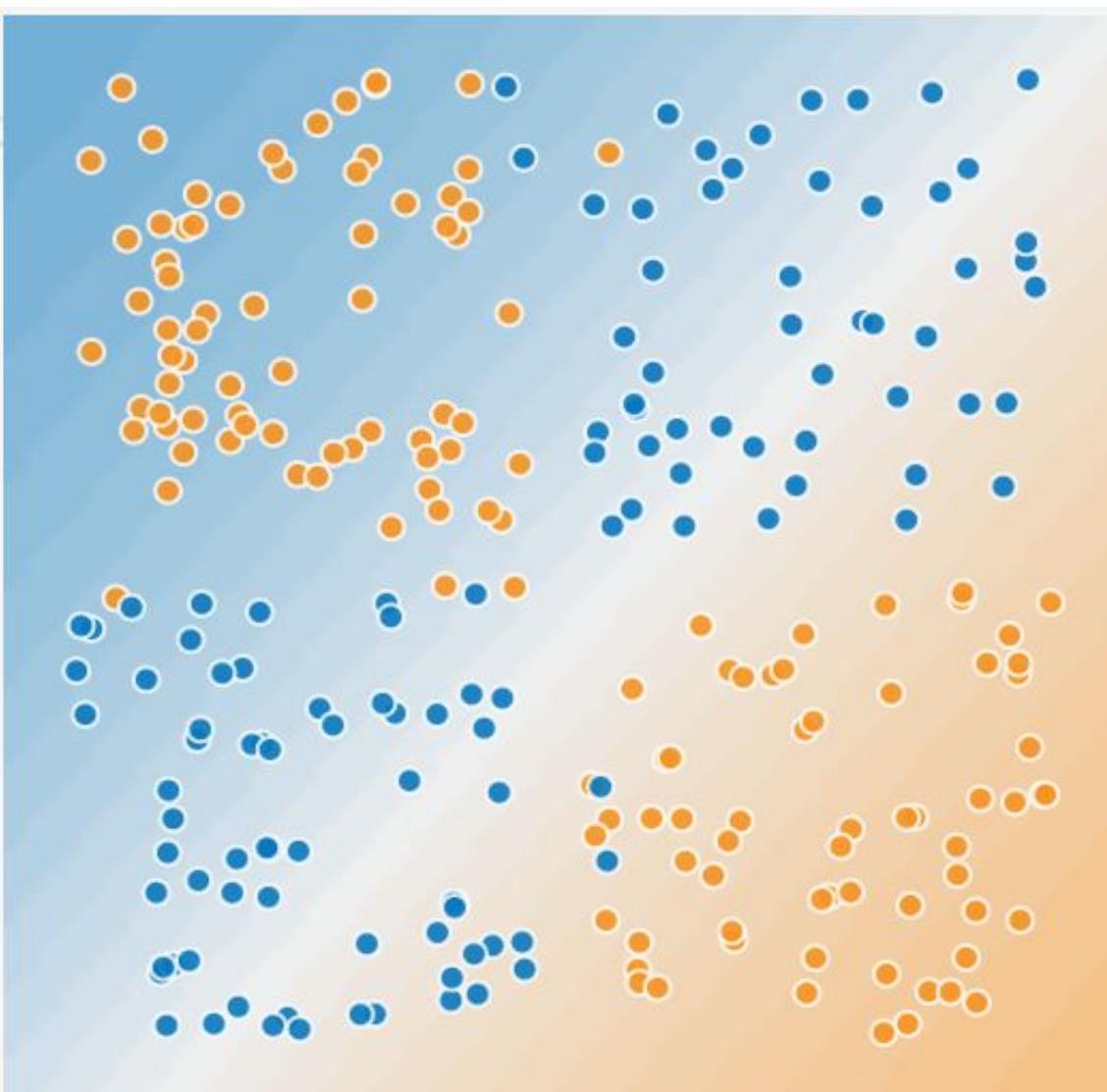
Can you draw a line  
that separates these  
two classes?



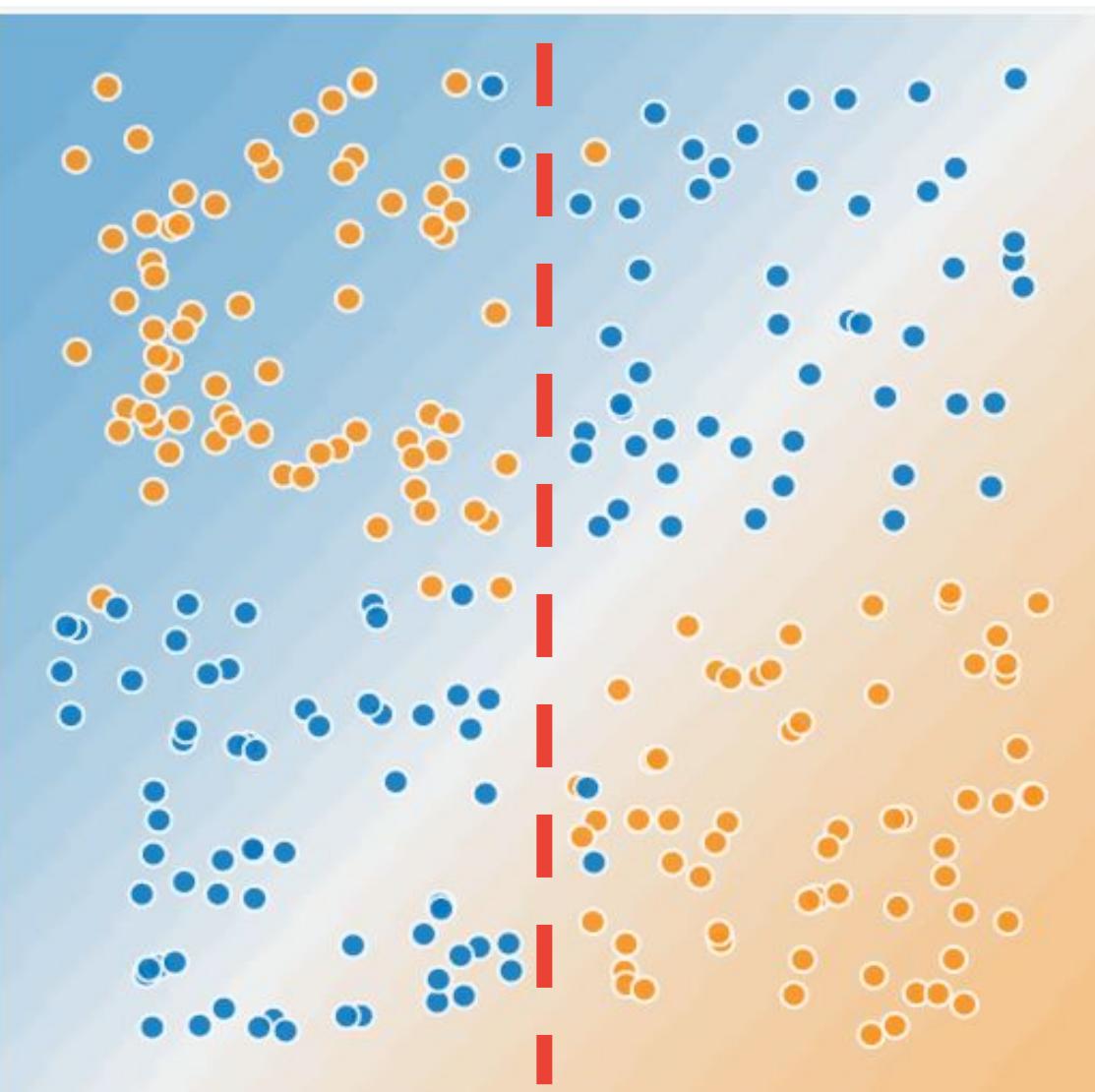
# This is a linear problem



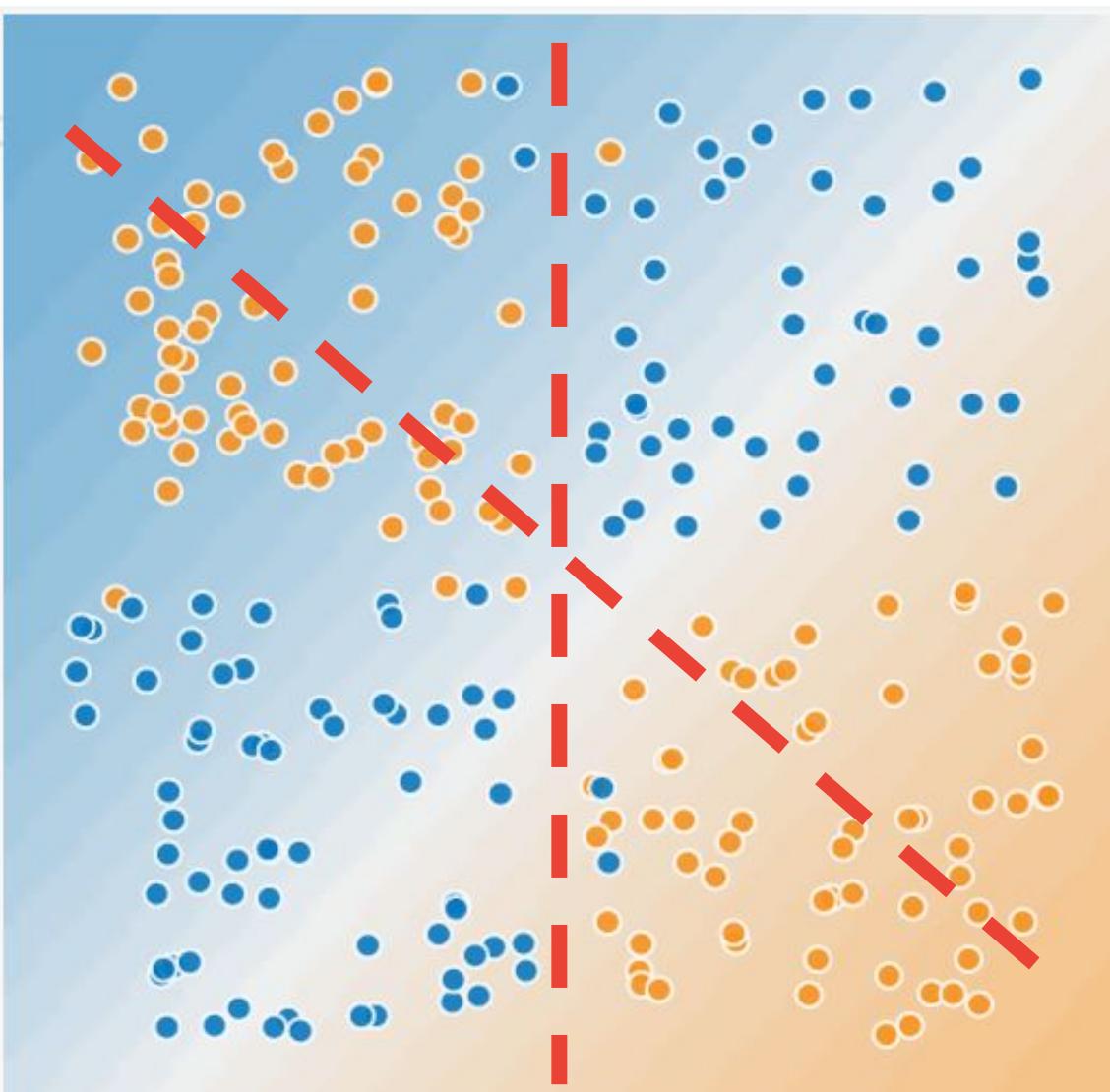
**How about this?  
Is it a linear problem?**



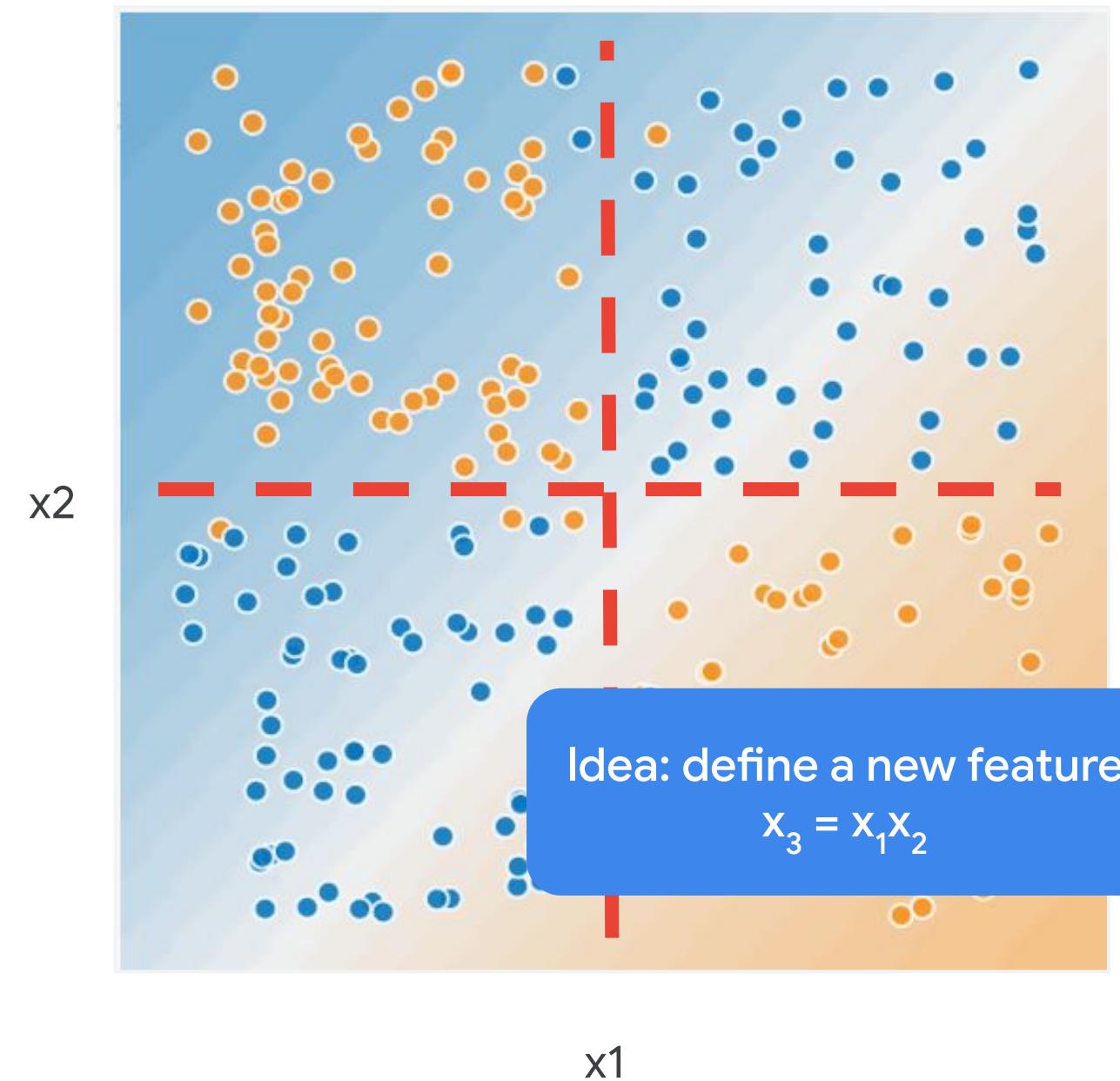
**How about this?  
Is it a linear problem?**



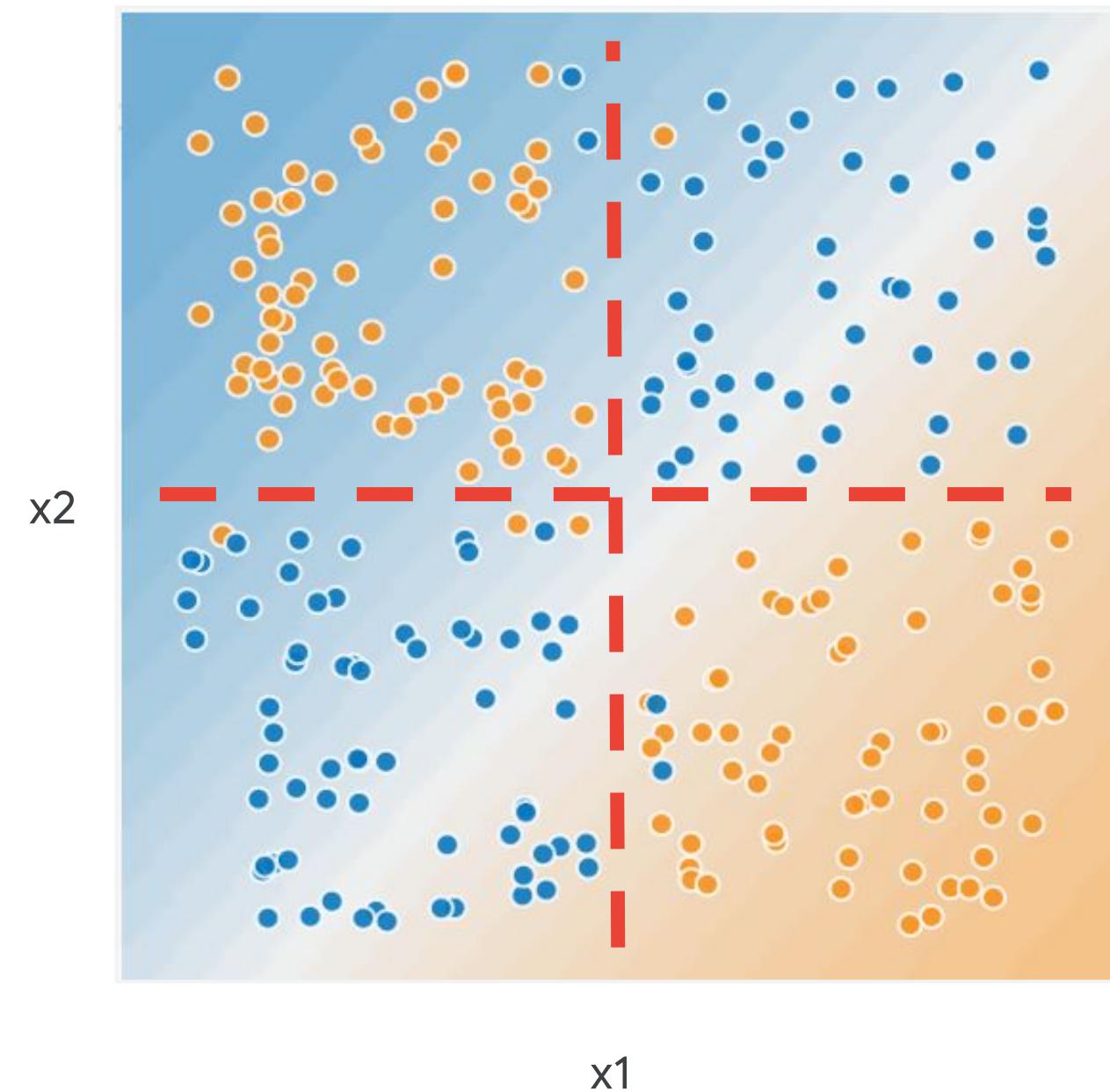
**How about this?  
Is it a linear problem?**



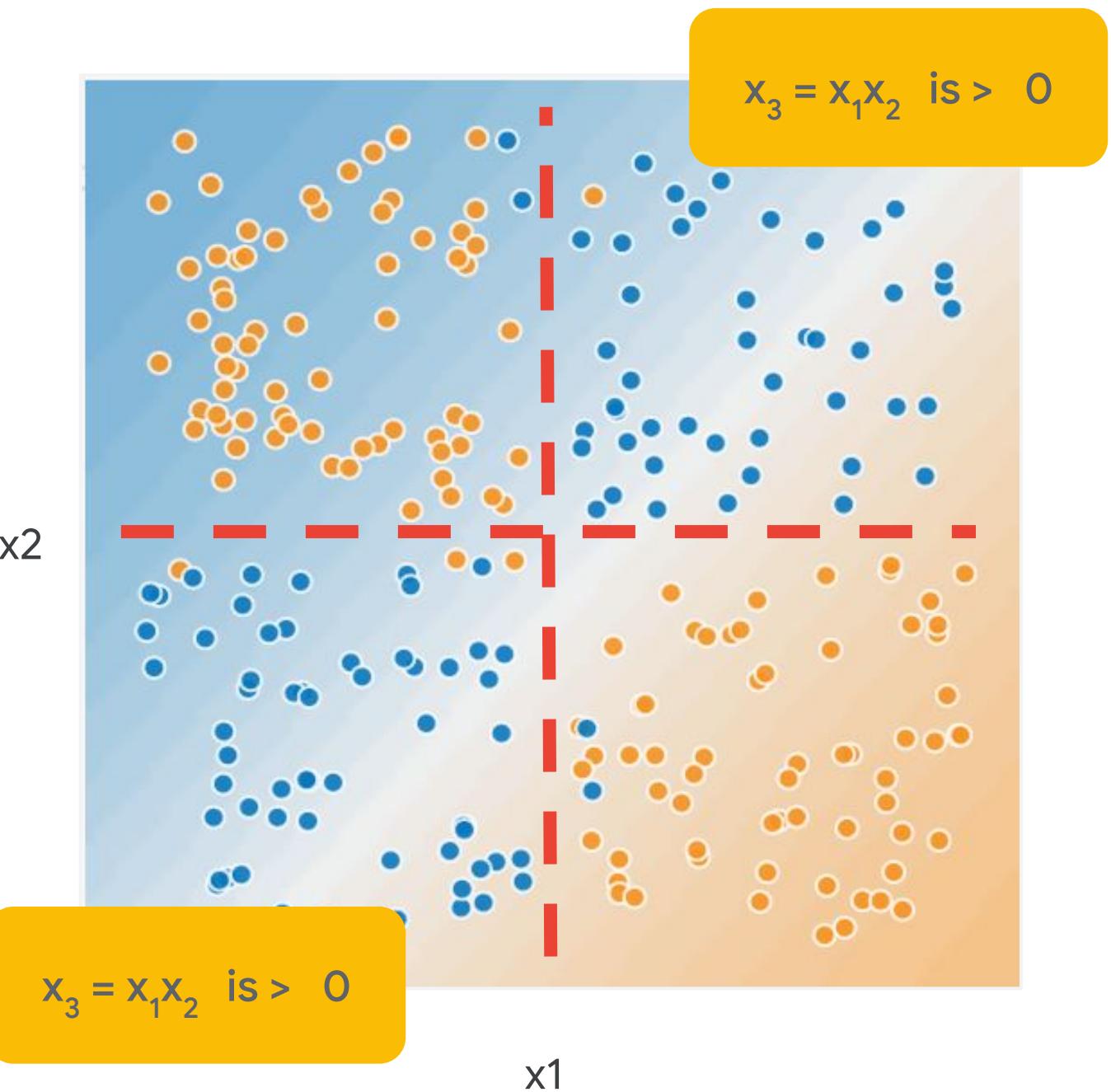
**How about this?  
Is it a linear problem?**



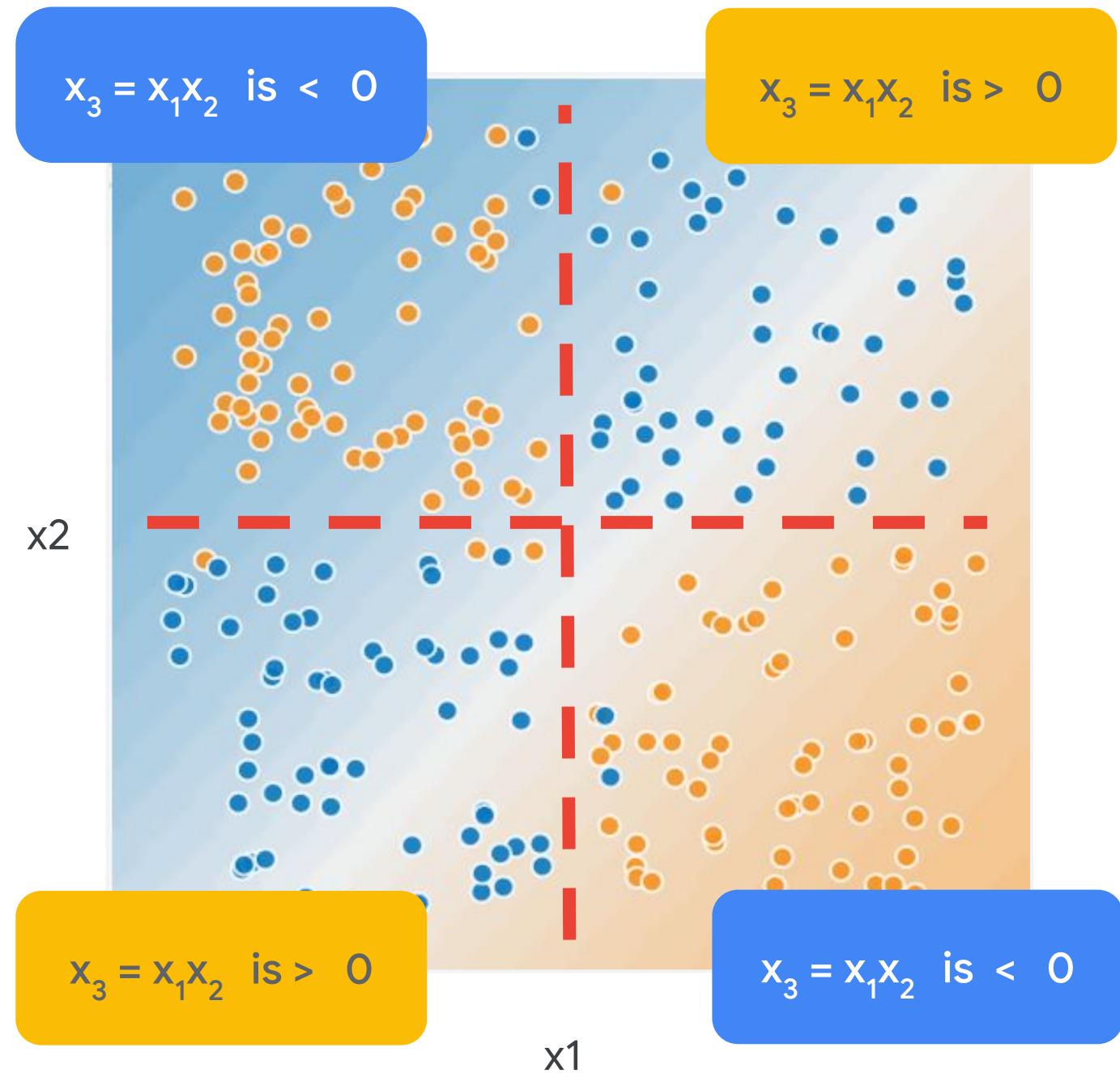
**How about this?  
Is it a linear problem?**



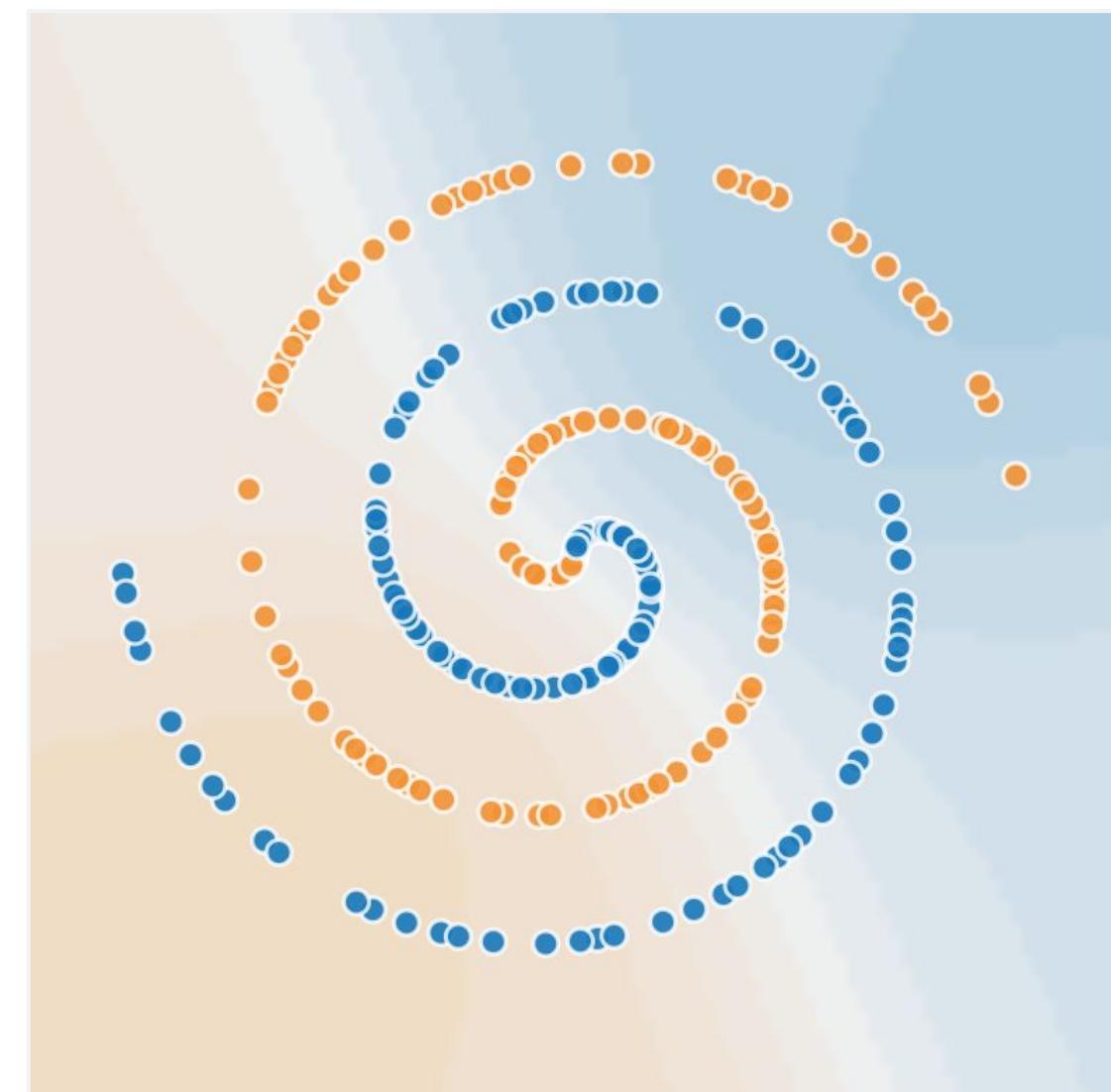
**How about this?  
Is it a linear problem?**



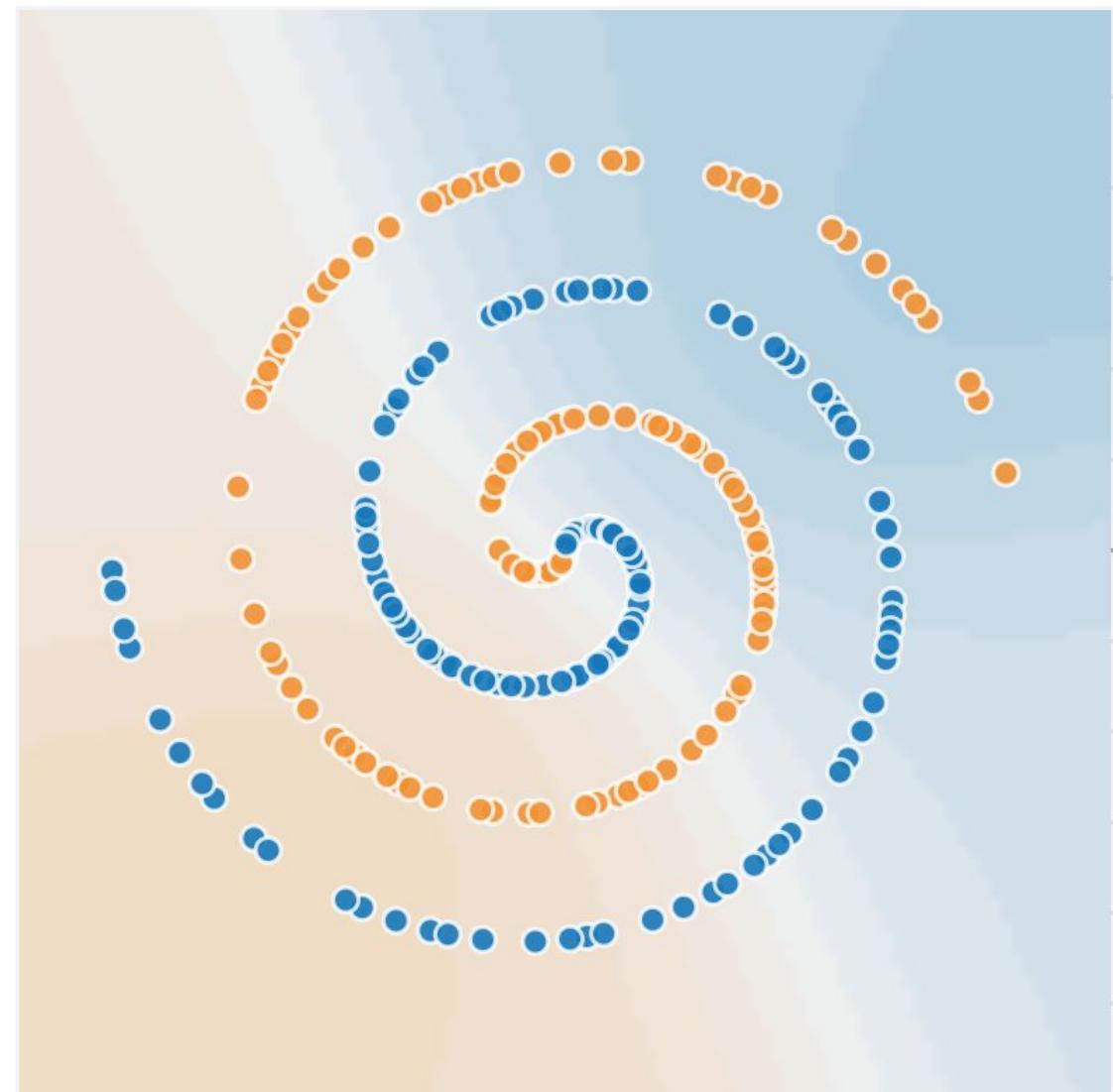
**How about this?  
Is it a linear problem?**



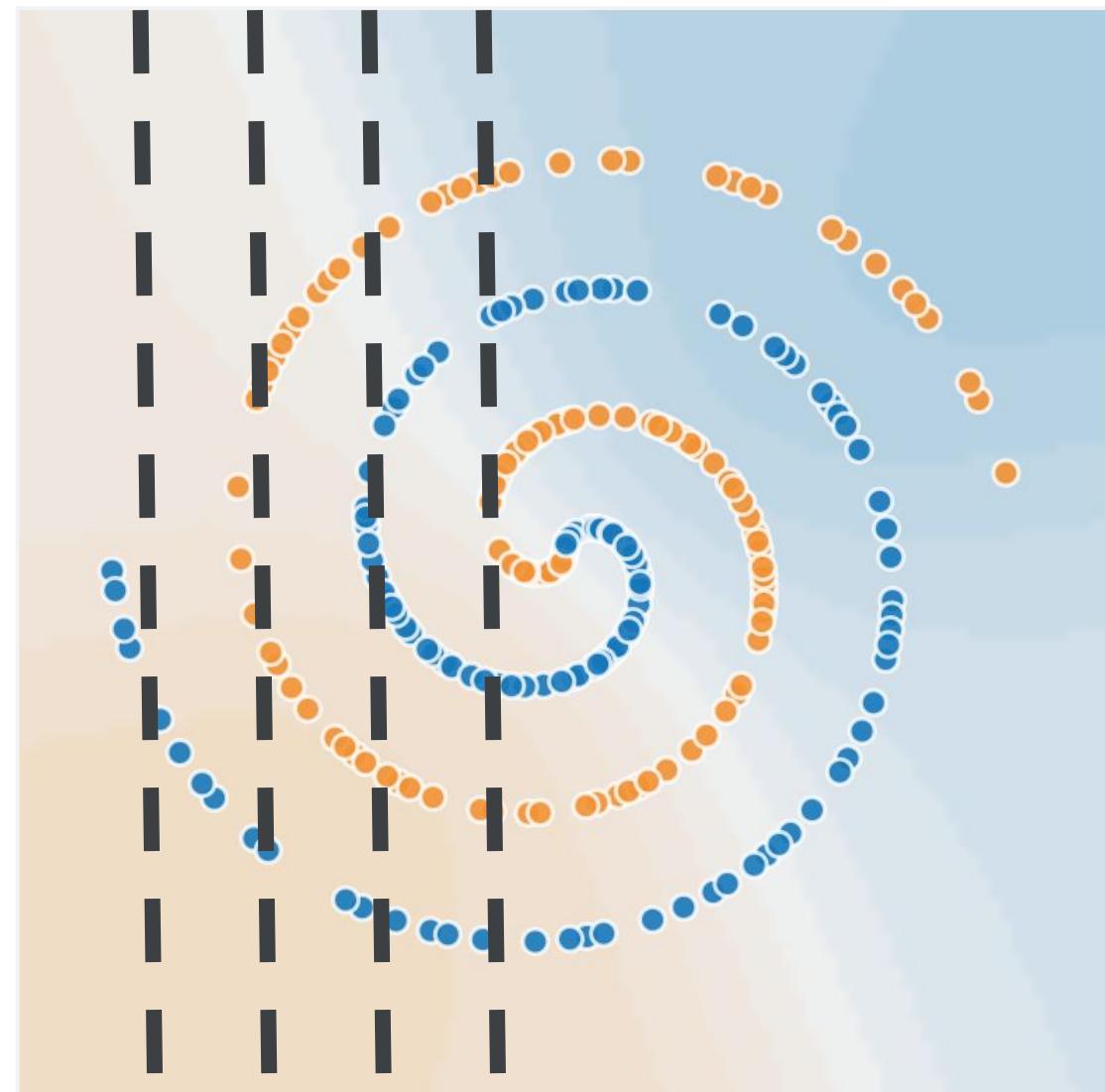
**Can a linear  
model work for  
this problem?**



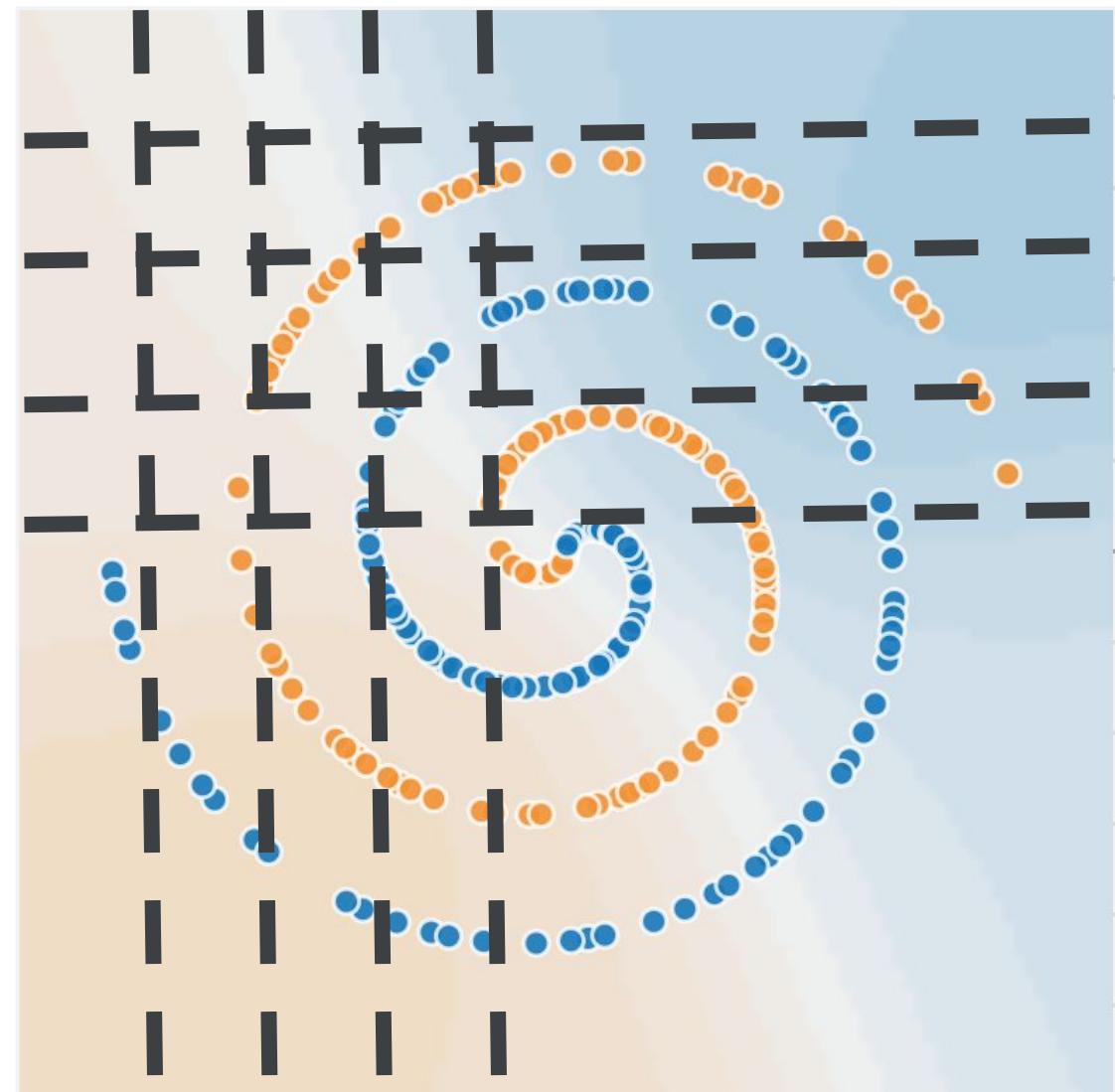
What if we  
discretize  $x_1$  and  
 $x_2$  and then  
multiply?



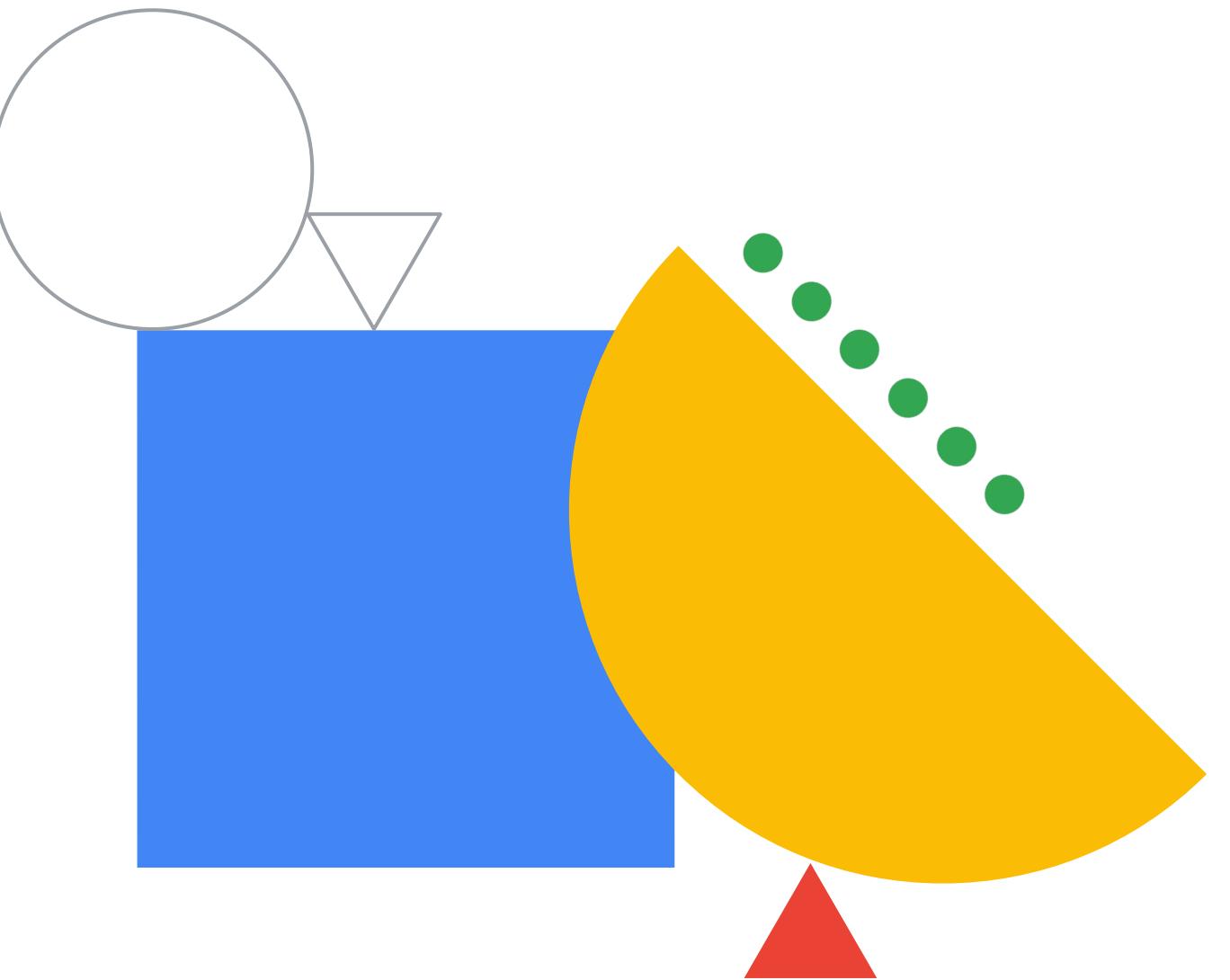
What if we  
discretize  $x_1$  and  
 $x_2$  and then  
multiply?



What if we  
discretize  $x_1$  and  
 $x_2$  and then  
multiply?



# An Introduction to TensorFlow Transform



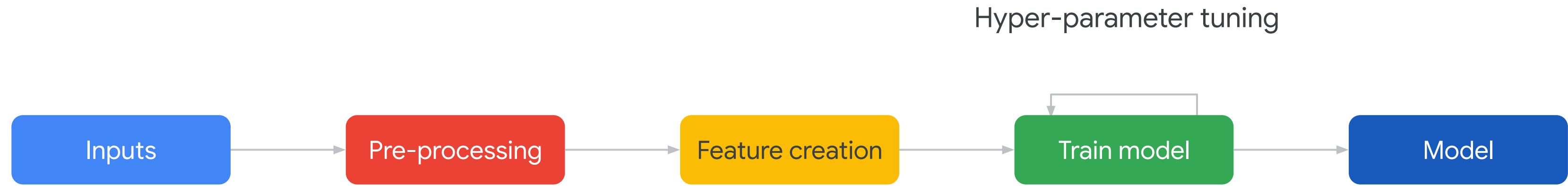
# In this module, you learn to ...

01

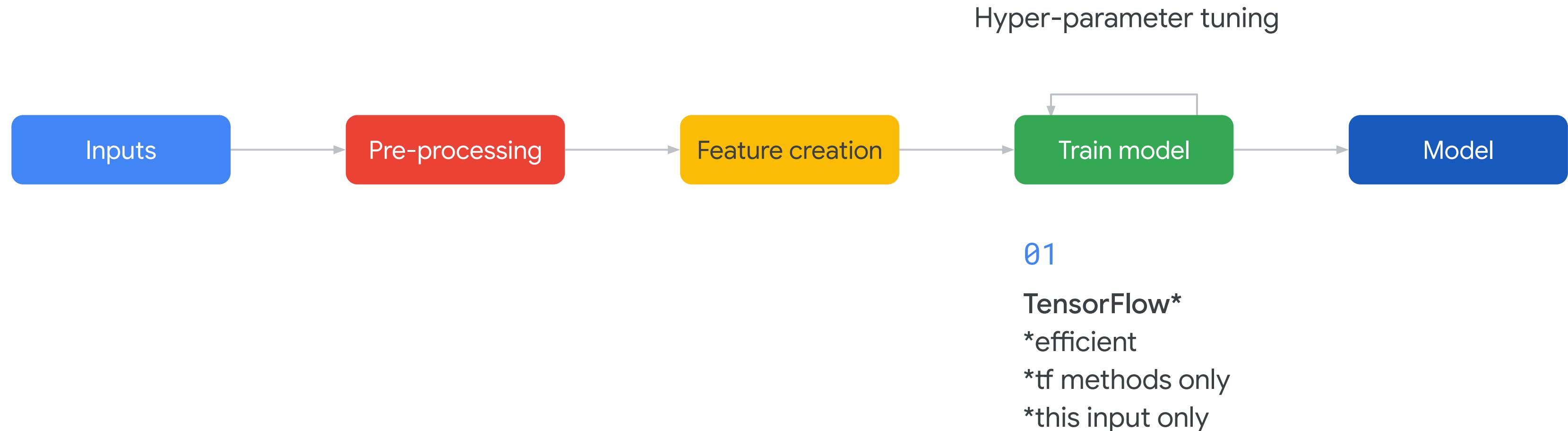
Implement feature preprocessing and  
feature creation using `tf.transform`



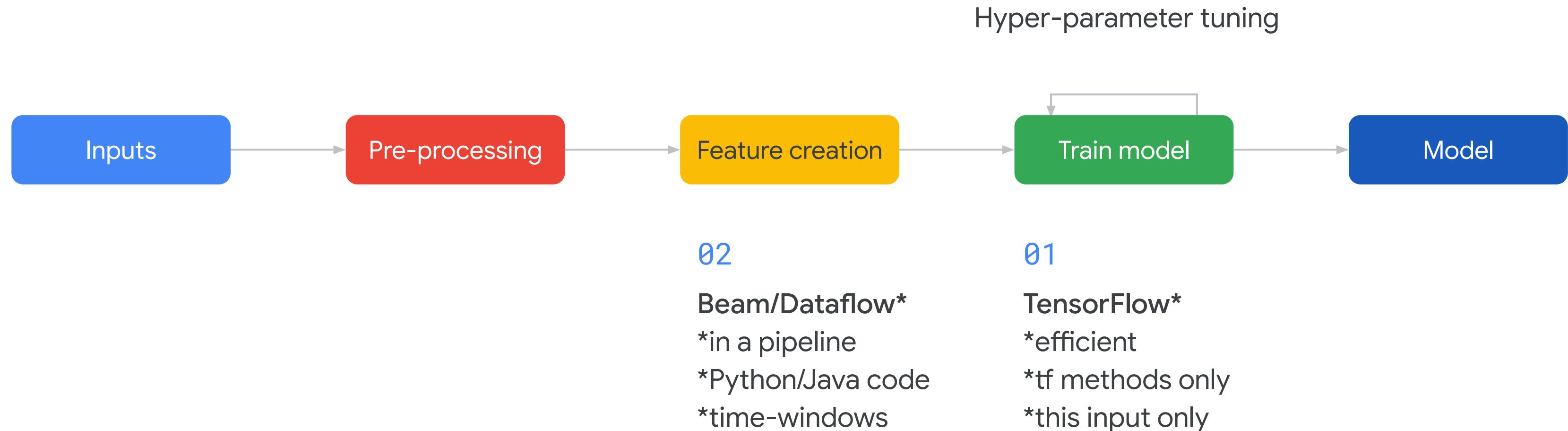
**There are three possible places to do feature engineering, each of which has its pros and cons**



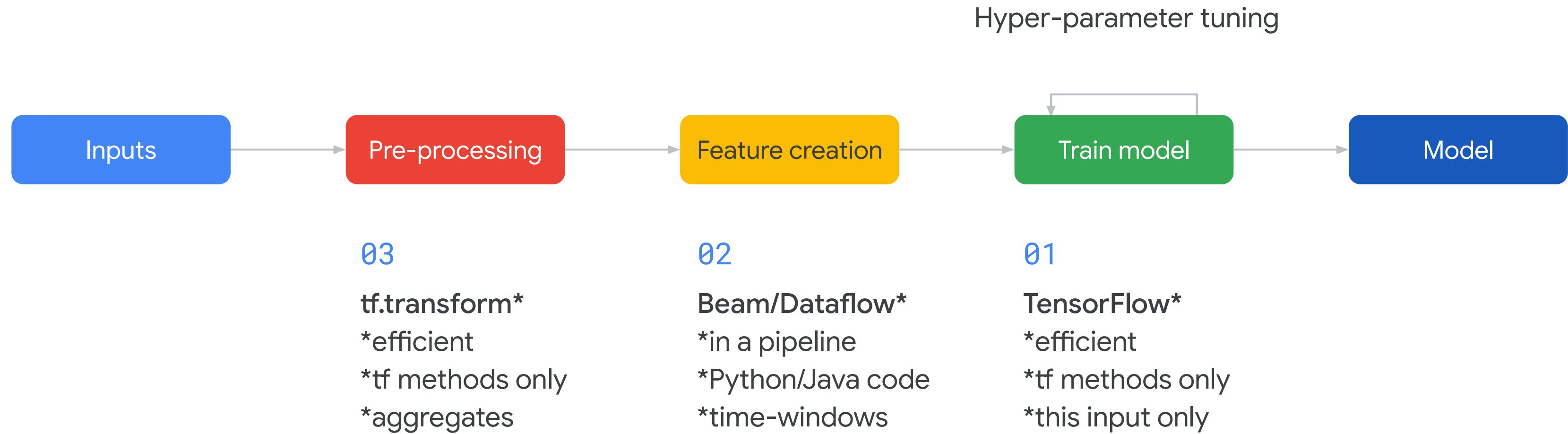
# There are three possible places to do feature engineering, each of which has its pros and cons



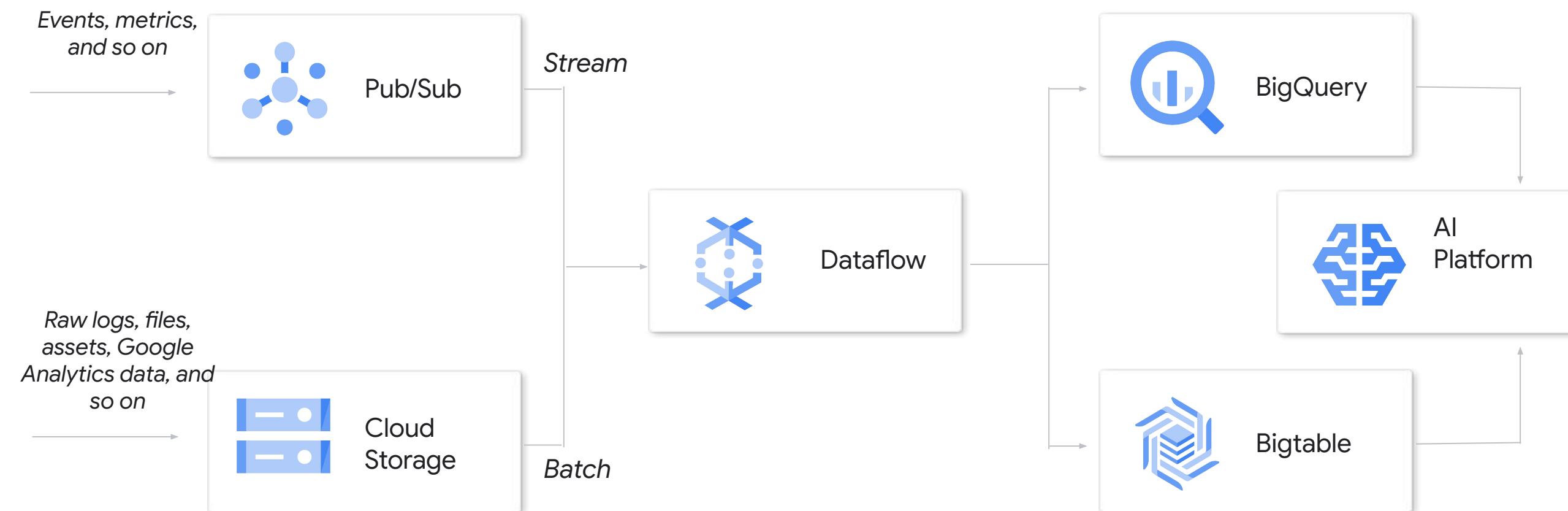
# There are three possible places to do feature engineering, each of which has its pros and cons



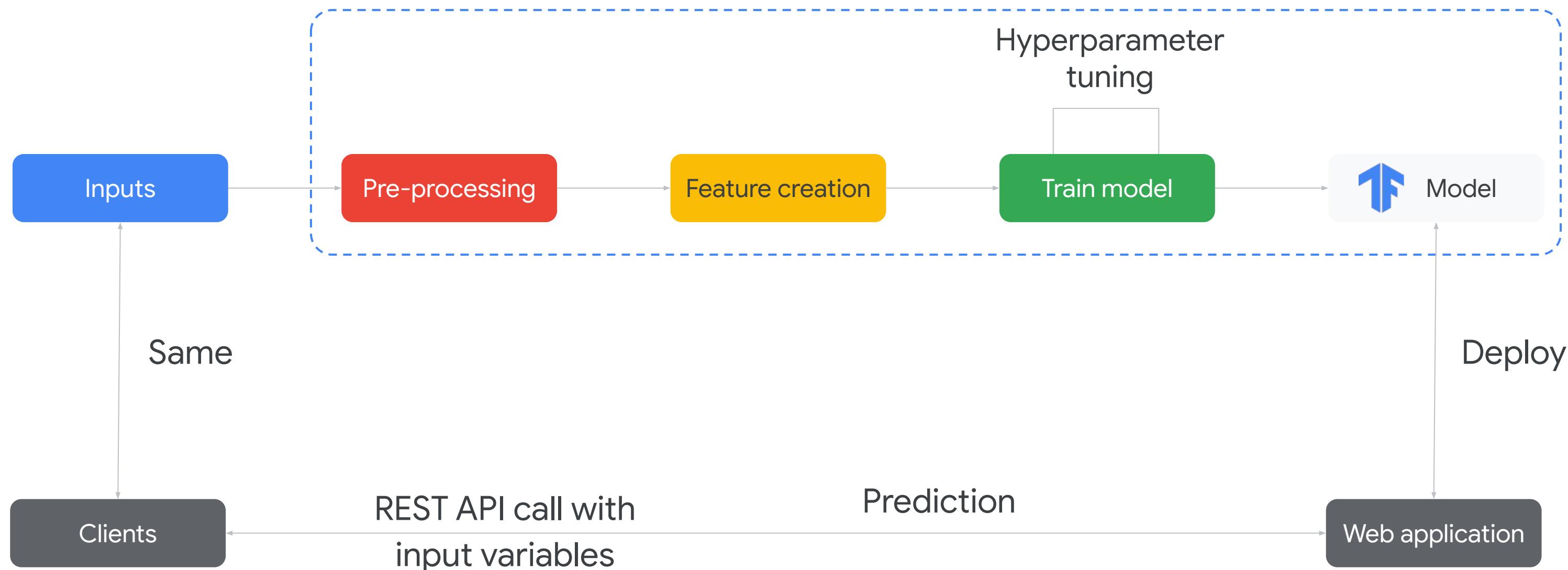
# There are three possible places to do feature engineering, each of which has its pros and cons



# Beam/Dataflow preprocessing works in the context of a pipeline



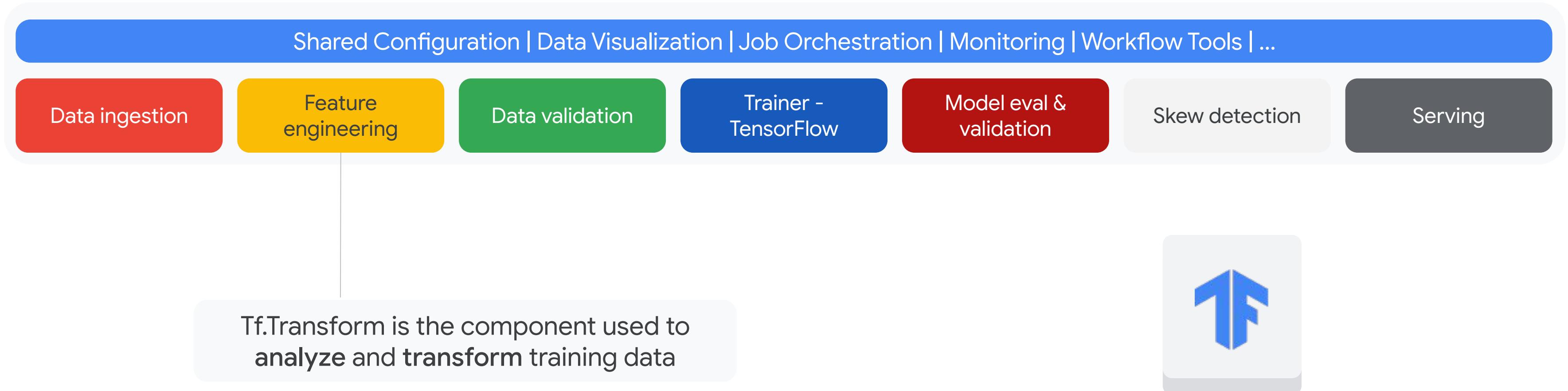
# TensorFlow is good for on-demand, on-the-fly processing



# TensorFlow Transform: A part of TFX

Productionizing machine learning requires  
more than just a learning algorithm.

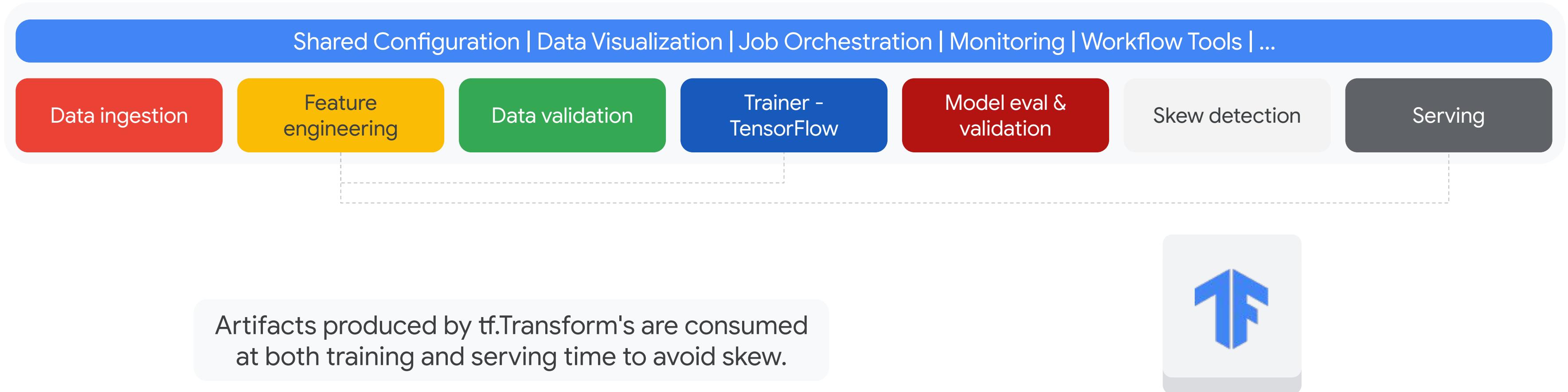
→ TFX is an end-to-end  
ML platform based on TensorFlow.



# TensorFlow Transform: A part of TFX

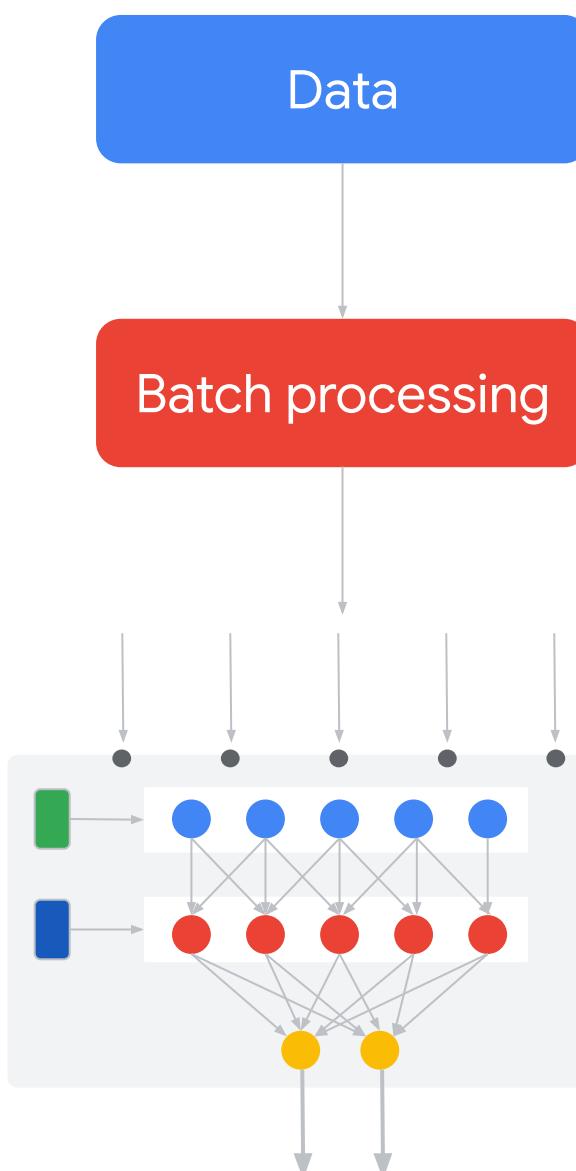
Productionizing machine learning requires  
more than just a learning algorithm.

→ TFX is an end-to-end  
ML platform based on TensorFlow.

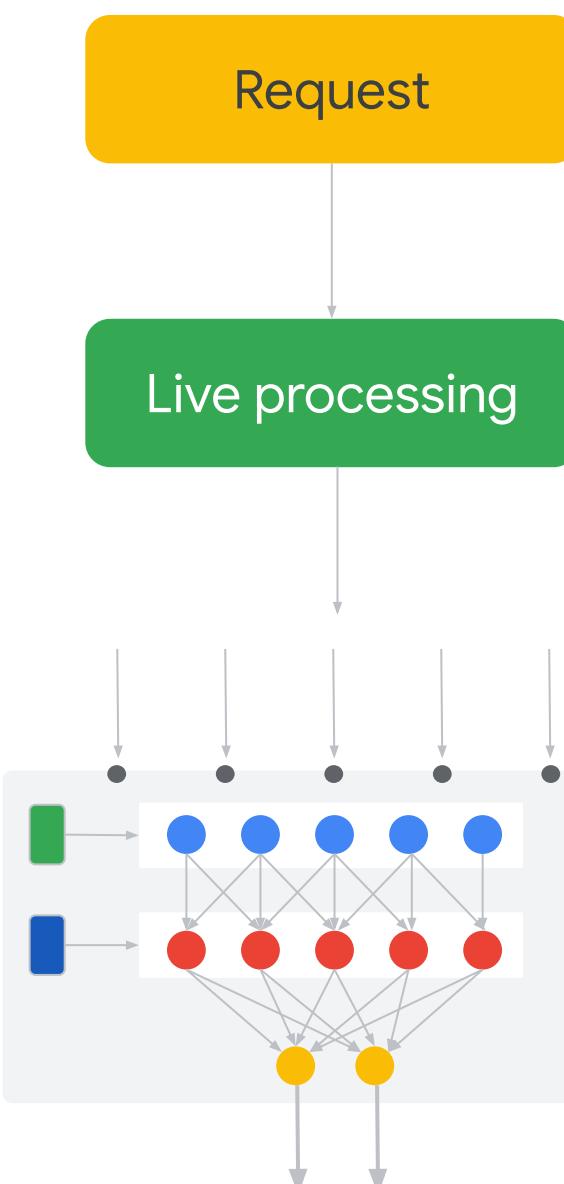


# Typical ML Pipeline

During training



During serving



# Problems

Need to keep batch and live processing in sync

All other tooling (e.g. evaluation) must also be kept in sync with batch processing



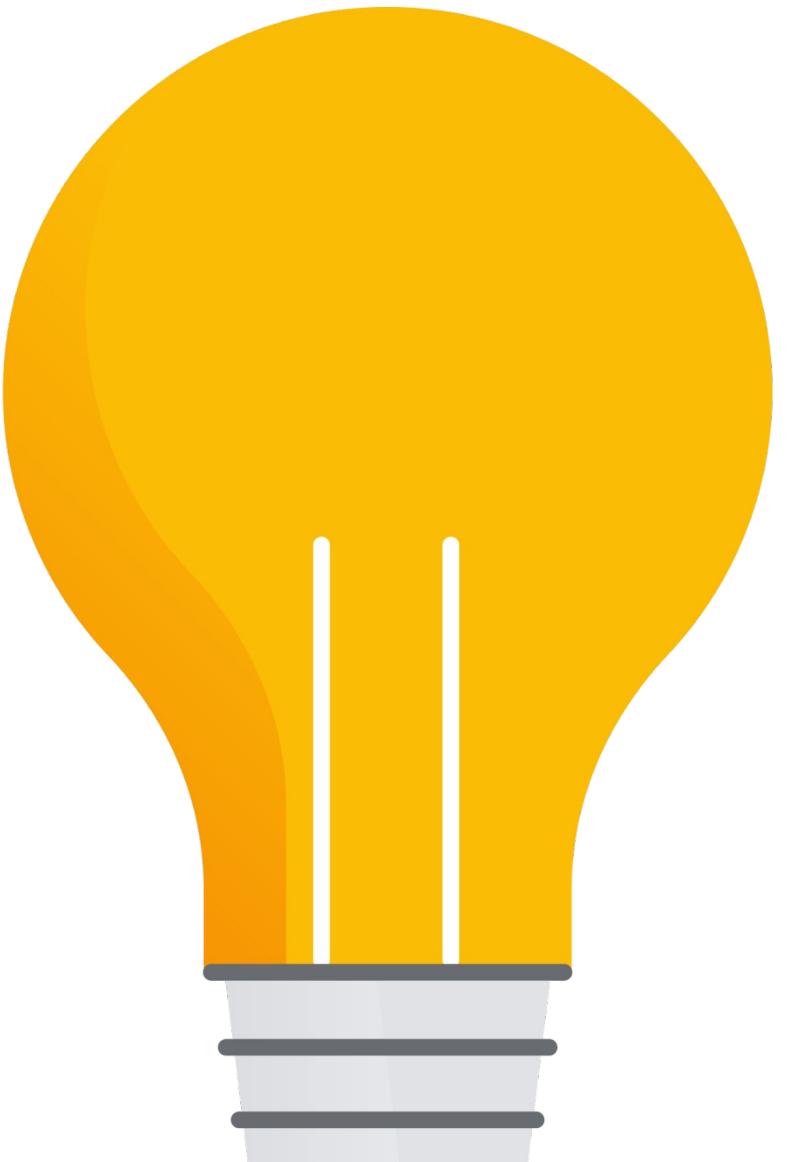
# Partial solutions

Do everything in the training graph

- Loses the benefits of materialization
- Doesn't allow for "reduces"

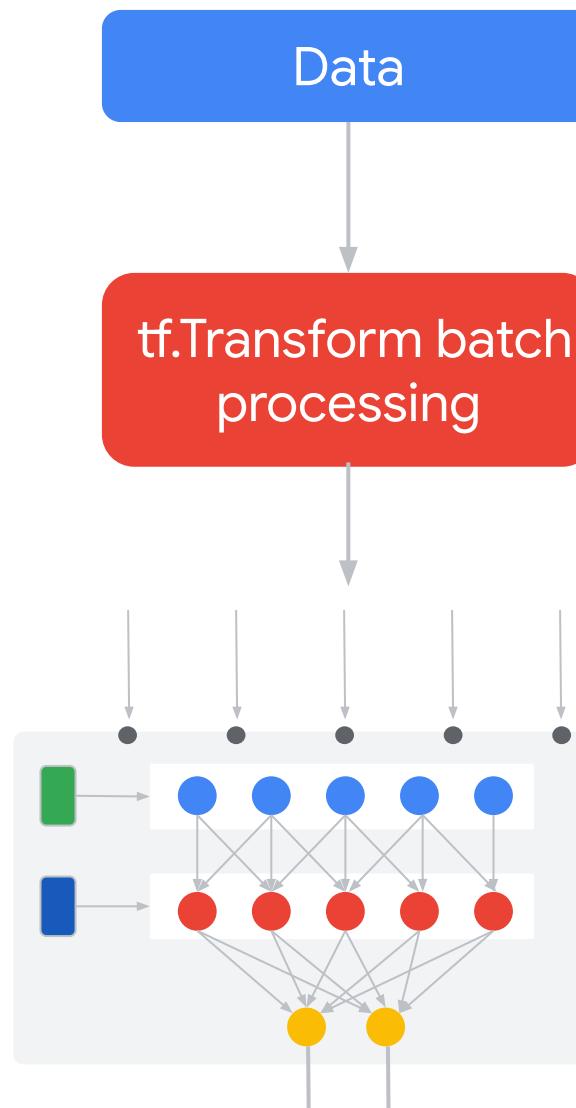
Do everything in the training graph + using statistics/vocabs generated from raw data

- Only allows for stats/vocabs on raw data
- Doesn't address materialization

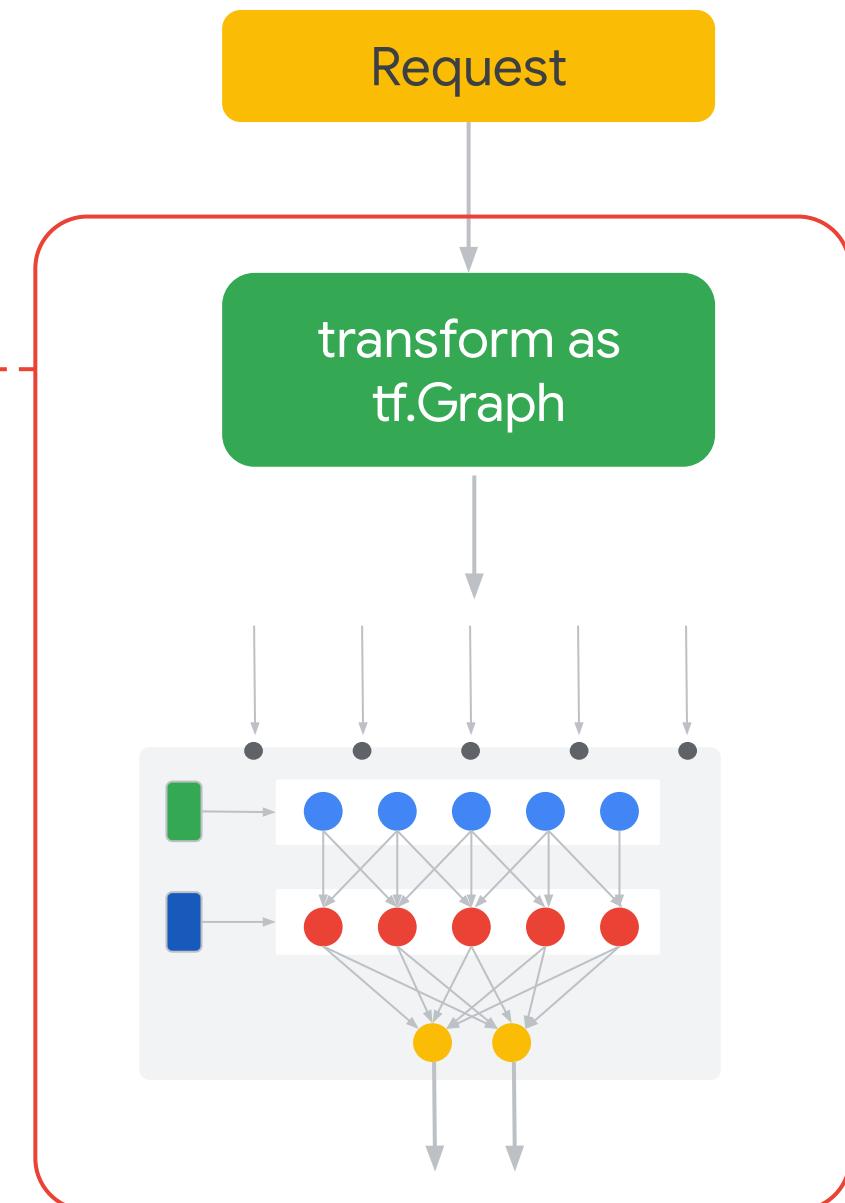


# tf.Transform

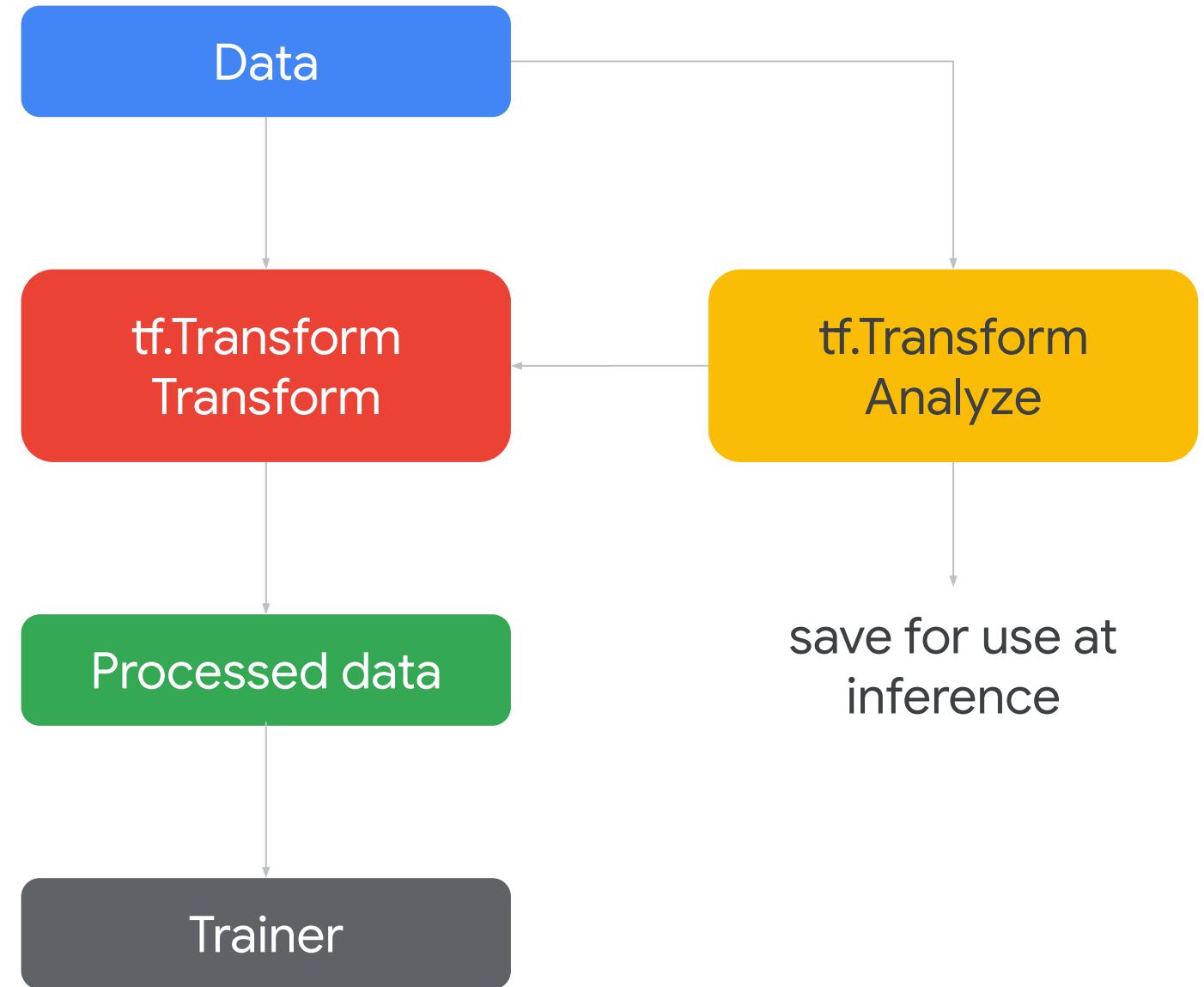
During training



During serving



# tf.Transform



`tf.transform` is a hybrid of  
Apache Beam and TensorFlow

**tf.transform** is a  
hybrid of Apache  
Beam and  
TensorFlow

Find min/max value of a numeric  
feature

**tf.transform** is a  
hybrid of Apache  
Beam and  
TensorFlow

Find min/max value of a numeric  
feature

Scale inputs by the min & max

**tf.transform** is a  
hybrid of Apache  
Beam and  
TensorFlow

Find min/max value of a numeric  
feature

Scale inputs by the min & max

Find all the unique values of a  
categorical feature

# **tf.transform is a hybrid of Apache Beam and TensorFlow**

Find min/max value of a numeric  
feature

Scale inputs by the min & max

Find all the unique values of a  
categorical feature

One-hot encode inputs based on  
set of unique values

# **tf.transform** is a hybrid of Apache Beam and TensorFlow

Find min/max value of a numeric  
feature

Scale inputs by the min & max

Find all the unique values of a  
categorical feature

One-hot encode inputs based on  
set of unique values

Analyze

# **tf.transform is a hybrid of Apache Beam and TensorFlow**

Find min/max value of a numeric  
feature

Scale inputs by the min & max

Find all the unique values of a  
categorical feature

One-hot encode inputs based on  
set of unique values

Analyze

Transform

# **tf.transform is a hybrid of Apache Beam and TensorFlow**

Find min/max value of a numeric  
feature

Find all the unique values of a  
categorical feature

Analyze

Beam

Scale inputs by the min & max

One-hot encode inputs based on  
set of unique values

Transform

TensorFlow

# tf.transform provides two PTransforms

## AnalyzeAndTransformDataset

---

- Executed in Beam to create the training dataset
- Similar in purpose to Scikit-learn's `fit_transform` method

## TransformDataset

---

- Executed in Beam to create the evaluation dataset
- The underlying transformations are executed in TensorFlow at prediction time
- Similar in purpose to Scikit-learn's `transform` method

# tf.transform has two phases

## Analysis phase

---

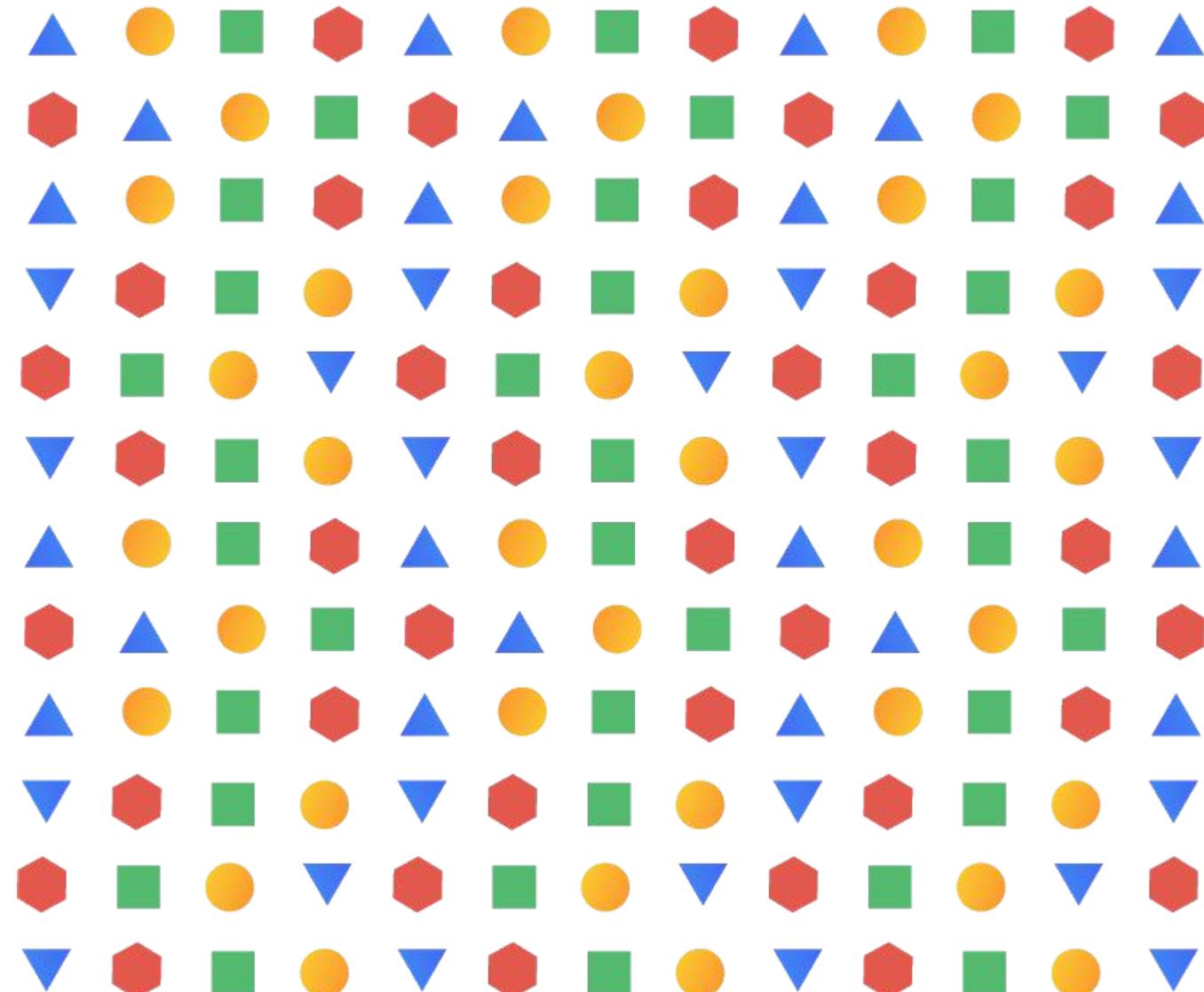
- Executed in Beam while creating training dataset

## Transform phase

---

- Executed in TensorFlow during prediction
- Executed in Beam to create training/evaluation datasets

First, set up the  
schema of the  
training dataset



# First, set up the schema of the training dataset

```
raw_data_schema = {  
    colname : dataset_schema.ColumnSchema(tf.string, ...)  
    for colname in 'dayofweek,key'.split(',',')  
}
```

# First, set up the schema of the training dataset

```
raw_data_schema = {  
    colname : dataset_schema.ColumnSchema(tf.string, ...)  
        for colname in 'dayofweek,key'.split(',',')  
}
```

1. TensorFlow type for input column

# First, set up the schema of the training dataset

```
raw_data_schema = {  
    colname : dataset_schema.ColumnSchema(tf.string, ...)  
    for colname in 'dayofweek,key'.split(',',')  
}  
  
raw_data_schema.update({  
    colname : dataset_schema.ColumnSchema(tf.float32, ...)  
    for colname in 'fare_amount,pickuplon, ... ,dropofflat'.split(',',')  
})
```

1. TensorFlow type for input column

2. float32

# First, set up the schema of the training dataset

```
raw_data_schema = {  
    colname : dataset_schema.ColumnSchema(tf.string, ...)  
        for colname in 'dayofweek,key'.split(',',')  
}  
  
raw_data_schema.update({  
    colname : dataset_schema.ColumnSchema(tf.float32, ...)  
        for colname in 'fare_amount,pickuplon, ... ,dropofflat'.split(',',')  
})  
  
raw_data_metadata =  
    dataset_metadata.DatasetMetadata(dataset_schema.Schema(raw_data_schema))
```

1. TensorFlow type for input column

2. float32

# First, set up the schema of the training dataset

```
raw_data_schema = {  
    colname : dataset_schema.ColumnSchema(tf.string, ...)  
        for colname in 'dayofweek,key'.split(',',')  
}  
  
raw_data_schema.update({  
    colname : dataset_schema.ColumnSchema(tf.float32, ...)  
        for colname in 'fare_amount,pickuplon, ... ,dropofflat'.split(',',')  
})  
  
raw_data_metadata =  
    dataset_metadata.DatasetMetadata(dataset_schema.Schema(raw_data_schema))
```

1. TensorFlow type for input column

2. float32

3. Use the schema to create metadata “template”

Next, run the analyze-and-transform PTransform on training dataset to get back preprocessed training data and the transform function

```
raw_data = (p
    | beam.io.Read(bean.io.BigQuerySource(query=myquery,
use_standard_sql=True))
    | beam.Filter(is_valid))

transformed_dataset, transform_fn = ((raw_data,
raw_data_metadata)
    | beam_impl.AnalyzeAndTransformDataset(preprocess))
```

Next, run the analyze-and-transform PTransform on training dataset to get back preprocessed training data and the transform function

```
raw_data = (p
    | beam.io.Read(bean.io.BigQuerySource(query=myquery,
use_standard_sql=True))
    | beam.Filter(is_valid))

transformed_dataset, transform_fn = ((raw_data,
raw_data_metadata)
    | beam_impl.AnalyzeAndTransformDataset(preprocess))
```

1. Read in data as usual for Beam

Next, run the analyze-and-transform PTransform on training dataset to get back preprocessed training data and the transform function

```
raw_data = (p
    | beam.io.Read(bean.io.BigQuerySource(query=myquery,
use_standard_sql=True))
    | beam.Filter(is_valid))
```

1. Read in data as usual for Beam
2. Filter out data that you don't want to train with

```
transformed_dataset, transform_fn = ((raw_data,
raw_data_metadata)
    | beam_impl.AnalyzeAndTransformDataset(preprocess))
```

# Next, run the analyze-and-transform PTransform on training dataset to get back preprocessed training data and the transform function

```
raw_data = (p
    | beam.io.Read(bean.io.BigQuerySource(query=myquery,
use_standard_sql=True))
    | beam.Filter(is_valid))

transformed_dataset, transform_fn = ((raw_data,
raw_data_metadata)
    | beam_impl.AnalyzeAndTransformDataset(preprocess))
```

1. Read in data as usual for Beam
2. Filter out data that you don't want to train with
3. Pass raw data + metadata template to AnalyzeAndTransformDataset

# Next, run the analyze-and-transform PTransform on training dataset to get back preprocessed training data and the transform function

```
raw_data = (p
    | beam.io.Read(bean.io.BigQuerySource(query=myquery,
use_standard_sql=True))
    | beam.Filter(is_valid))

transformed_dataset, transform_fn = ((raw_data, raw_data_metadata)
| beam_impl.AnalyzeAndTransformDataset(preprocess))
```

1. Read in data as usual for Beam
2. Filter out data that you don't want to train with
3. Pass raw data + metadata template to AnalyzeAndTransformDataset
4. Get back transformed dataset and a reusable transform function

# Write out the preprocessed training data into TFRecords, the most efficient format for TensorFlow

```
transformed_data | tfrecordio.WriteToTFRecord(  
    os.path.join(OUTPUT_DIR, 'train'),  
    coder=ExampleProtoCoder(  
        transformed_metadata.schema)  
    )
```

# Write out the preprocessed training data into TFRecords, the most efficient format for TensorFlow

```
transformed_data | tfrecordio.WriteToTFRecord(  
    os.path.join(OUTPUT_DIR, 'train'),  
    coder=ExampleProtoCoder(  
        transformed_metadata.schema)  
    )
```

1. The filenames will be like  
train-0003-of-0015

# Write out the preprocessed training data into TFRecords, the most efficient format for TensorFlow

```
transformed_data | tfrecordio.WriteToTFRecord(  
    os.path.join(OUTPUT_DIR, 'train'),  
    coder=ExampleProtoCoder(  
        transformed_metadata.schema)  
    )
```

1. The filenames will be like  
train-0003-of-0015
2. Note that we use the  
transformed metadata  
schema here!

# The preprocessing function is restricted to TensorFlow functions

```
transformed_dataset, transform_fn = ((raw_data, raw_data_metadata)
| beam_impl.AnalyzeAndTransformDataset(
    preprocess))
```

The things you do in `preprocess()` will get added to the TensorFlow graph, and be executed in TensorFlow during serving

The preprocessing function is restricted to functions you can call from TensorFlow graph

```
def preprocess(inputs):
```

# The preprocessing function is restricted to functions you can call from TensorFlow graph

```
def preprocess(inputs):
```

```
    result = {}
```

- 
1. Create features from the input tensors and put into “results” dict

# The preprocessing function is restricted to functions you can call from TensorFlow graph

```
def preprocess(inputs):  
  
    result = {}  
  
    result['fare_amount'] = inputs['fare_amount']
```

1. Create features from the input tensors and put into “results” dict
2. Pass through

# The preprocessing function is restricted to functions you can call from TensorFlow graph

```
def preprocess(inputs):  
  
    result = {}  
  
    result['fare_amount'] = inputs['fare_amount']  
  
    result['dayofweek'] = tft.string_to_int(inputs['dayofweek'])
```

1. Create features from the input tensors and put into “results” dict
2. Pass through
3. Vocabulary

# The preprocessing function is restricted to functions you can call from TensorFlow graph

```
def preprocess(inputs):  
  
    result = {}  
  
    result['fare_amount'] = inputs['fare_amount']  
  
    result['dayofweek'] = tft.string_to_int(inputs['dayofweek'])  
  
    ...  
    result['dropofflat'] = (tft.scale_to_0_1(inputs['dropofflat']))
```

1. Create features from the input tensors and put into “results” dict
2. Pass through
3. Vocabulary
4. Scaling

# The preprocessing function is restricted to functions you can call from TensorFlow graph

```
def preprocess(inputs):  
  
    result = {}  
  
    result['fare_amount'] = inputs['fare_amount']  
  
    result['dayofweek'] = tft.string_to_int(inputs['dayofweek'])  
    ...  
    result['dropofflat'] = (tft.scale_to_0_1(inputs['dropofflat']))  
  
    result['passengers'] = tf.cast(inputs['passengers'], tf.float32)
```

1. Create features from the input tensors and put into “results” dict
2. Pass through
3. Vocabulary
4. Scaling
5. Other TF fns

# The preprocessing function is restricted to functions you can call from TensorFlow graph

```
def preprocess(inputs):  
  
    result = {}  
  
    result['fare_amount'] = inputs['fare_amount']  
  
    result['dayofweek'] = tft.string_to_int(inputs['dayofweek'])  
    ...  
    result['dropofflat'] = (tft.scale_to_0_1(inputs['dropofflat']))  
  
    result['passengers'] = tf.cast(inputs['passengers'], tf.float32)  
  
    return result
```

1. Create features from the input tensors and put into “results” dict
2. Pass through
3. Vocabulary
4. Scaling
5. Other TF fns

# Analyze and Transform happens on the training dataset

```
transformed_dataset, transform_fn = ((raw_data, raw_data_metadata)
| beam_impl.AnalyzeAndTransformDataset(preprocess))
```

Writing out the eval dataset is similar, except that we reuse the transform function computed from the training data

```
raw_test_data = (p
    |
    beam.io.Read(beam.io.BigQuerySource(...))
        | 'eval_filter' >>
    beam.Filter(is_valid))
transformed_test_dataset = (((raw_test_data,
    raw_data_metadata), transform_fn)
    | beam_impl.TransformDataset())
```

**Writing out the eval dataset is similar, except that we reuse the transform function computed from the training data**

```
raw_test_data = (p
    | beam.io.Read(bean.io.BigQuerySource(...))
    | 'eval_filter' >> beam.Filter(is_valid))
transformed_test_dataset = (((raw_test_data,
    raw_data_metadata), transform_fn)
    | beam_impl.TransformDataset())

transformed_test_data, _ = transformed_test_dataset
_ = transformed_test_data |
    tfrecordio.WriteToTFRecord(
        os.path.join(OUTPUT_DIR, 'eval'),
        coder=example_proto_coder.ExampleProtoCoder(
            transformed_metadata.schema))
```

For training and evaluation, we created preprocessed features using Beam



Created by `AnalyzeAndTransformDataset`  
Or by `TransformDataset`

**For serving, we need  
to write out the  
transformation  
metadata**

```
_ = transform_fn |  
    transform_fn_io.WriteTransformFn(  
        os.path.join(OUTPUT_DIR, 'metadata'))
```

Buckets / cloud-training-demos-ml / [taxifare](#) / [preproc\\_tft](#) / metadata

	Name	Size	Type
	rawdata_metadata/	-	Folder
	transform_fn/	-	Folder
	transformed_metadata/	-	Folder

# Change input function to read preprocessed features

```
def input_fn(tf_transform_output, transformed_examples_pattern, batch_size):  
  
    return tf.data.experimental.make_batched_features_dataset(  
        file_pattern=transformed_examples_pattern,  
        batch_size=batch_size,  
        features=tf_transform_output.transformed_feature_spec(),  
        reader=tf.data.TFRecordDataset,  
        label_key=LABEL_KEY,  
        shuffle=True).prefetch(tf.data.experimental.AUTOTUNE)
```

# Change input function to read raw features

```
dataset = tf.data.experimental.make_csv_dataset(...)

tft_layer = tf_transform_output.transform_features_layer()

def transform_dataset(data):
    ...
    transformed_features = tft_layer(raw_features)
    ...
    return (transformed_features, data_labels)

return dataset.map(
    transform_dataset,
    num_parallel_calls=tf.data.experimental.AUTOTUNE).prefetch(
        tf.data.experimental.AUTOTUNE)
```

# The serving input function accepts the raw data

```
model.tft_layer = tf_transform_output.transform_features_layer()

@tf.function
def serve_tf_examples_fn(serialized_tf_examples):
    feature_spec = RAW_DATA_FEATURE_SPEC.copy()
    feature_spec.pop(LABEL_KEY)
    parsed_features = tf.io.parse_example(serialized_tf_examples, feature_spec)
    transformed_features = model.tft_layer(parsed_features)
    outputs = model(transformed_features)
    classes_names = tf.constant([[0, 1]])
    classes = tf.tile(classes_names, [tf.shape(outputs)[0], 1])
    return {'classes': classes, 'scores': outputs}

concrete_serving_fn = serve_tf_examples_fn.get_concrete_function(
    tf.TensorSpec(shape=[None], dtype=tf.string, name='inputs'))
signatures = {'serving_default': concrete_serving_fn}

model.save(output_dir, save_format='tf', signatures=signatures)
```

The model graph includes  
the preprocessing code

