Feature Engineering - the art of extracting value from data

Maciej Kępa

Roadmap

Data Community

01 AI/ML architecture - how it all fits together

Understand the full ML stack from data sources to production, including roles, cloud reference architectures, layers, tools, and a simple ETL + model demo.

02 Data Preparation - practical foundations

Learn practical techniques for building ML data pipelines, cleaning data, validating datasets, and preparing a dataset in PySpark

03 Feature Engineering – the art of extracting value from data

Discover how to create effective features, handle different types of data, apply encoding and normalization, monitor feature drift, and track features with Spark + MLflow.

04 ML Algorithms - the classical approach

Get familiar with core algorithms like regression, trees, and gradient boosting, understand classification, regression, clustering, their pros and cons, and try MLlib demos in Databricks.

05 Model Training in Practice

Learn how to split data, perform hyperparameter tuning, scale training, and train models with code and metrics using MLflow.

06 Deep Learning - leveling up

Understand neural network basics, choose between TensorFlow and PyTorch, leverage GPU/TPU scaling, and build a simple CNN with visualization.

07 ML Pipelines - automation and CI/CD

Build repeatable ML workflows with orchestration tools, CI/CD pipelines, model registry/versioning, and a Databricks pipeline demo.

08 MLOps - manage your ML solution

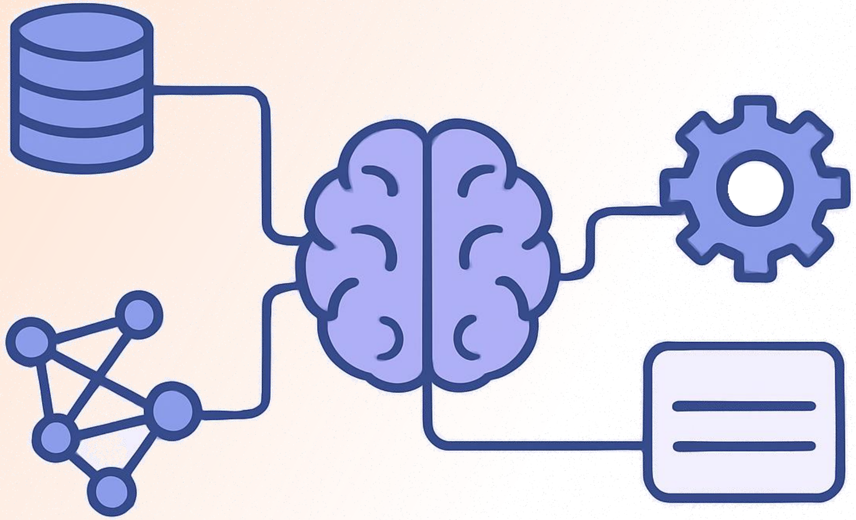
Explore ML lifecycle management with tracking, model monitoring, data drift handling, canary deployments, A/B testing, and monitoring demos.

09 Generative AI and LLMs - the new wave of technology

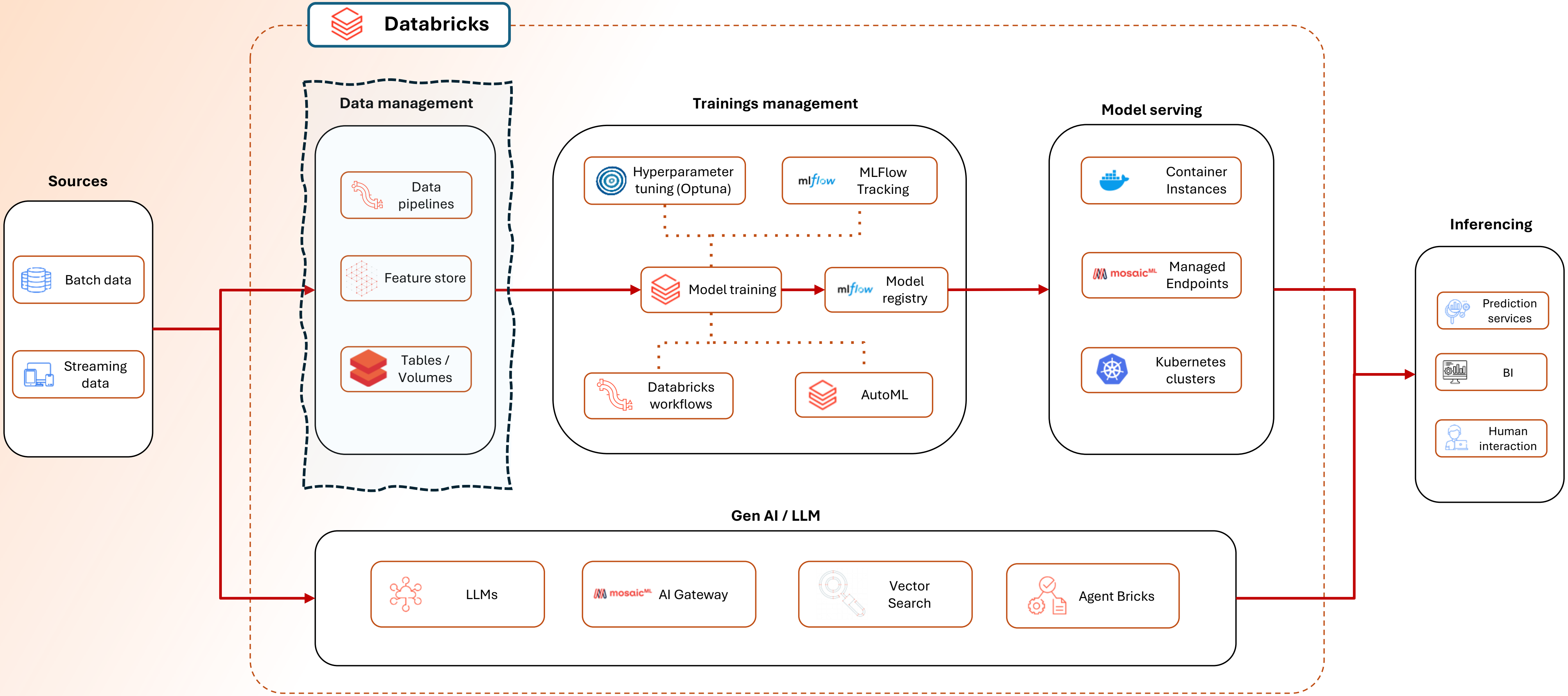
Learn how LLMs differ from classical ML, understand transformers, RAG, AI agent orchestration, and try a simple RAG demo in Azure OpenAI.

Agenda

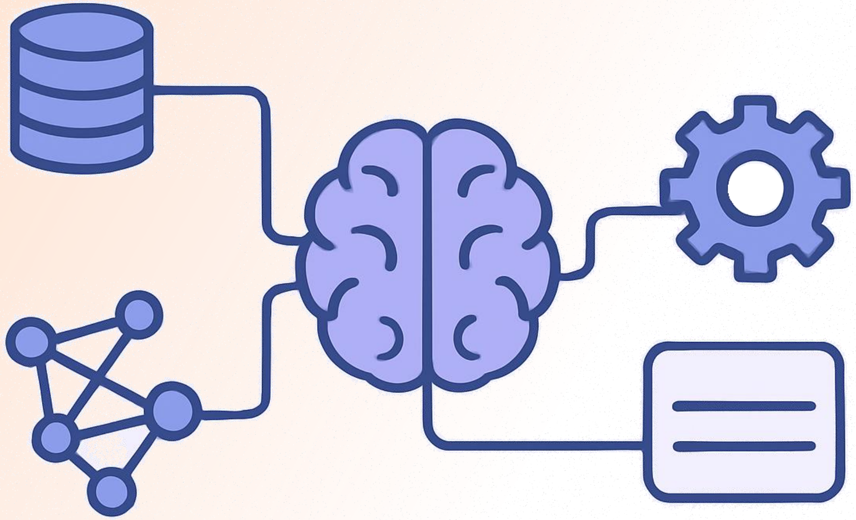
1. Introduction
2. Feature engineering concepts
3. Feature store and data drift
4. Workshop: feature engineering
5. Workshop: feature store
6. Workshop: data drift monitoring
7. Best practices



Introduction

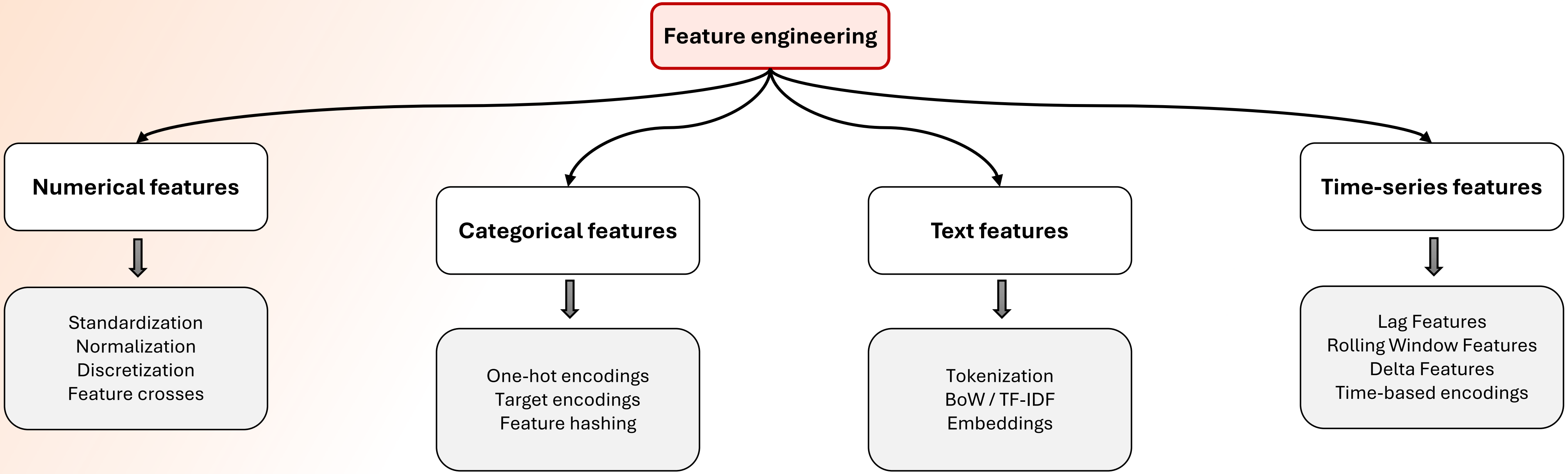


Data Community



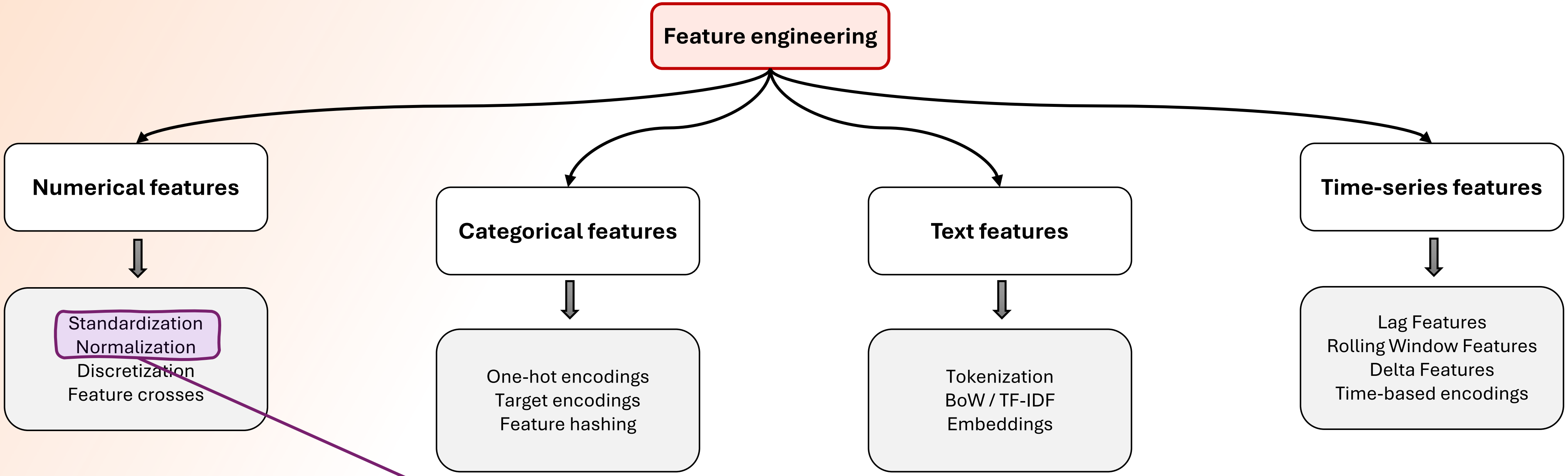
Feature engineering concepts

Feature engineering concepts



Feature engineering concepts

Data Community



Brings features to a comparable scale

YES	BUT
• Faster and more stable training • Prevents dominance of large-scale features • Required for distance-based and linear models	• Tree-based models don't need it • Sensitive to outliers • Scaling breaks business interpretability • Transformation must be applied to inferencing in the same way • Is often a source to data leak

x		x'
10000	→	0
30000	→	0.33
70000	→	1
20000	→	0.16
50000	→	0.66

Feature engineering concepts

Data Community

Feature engineering

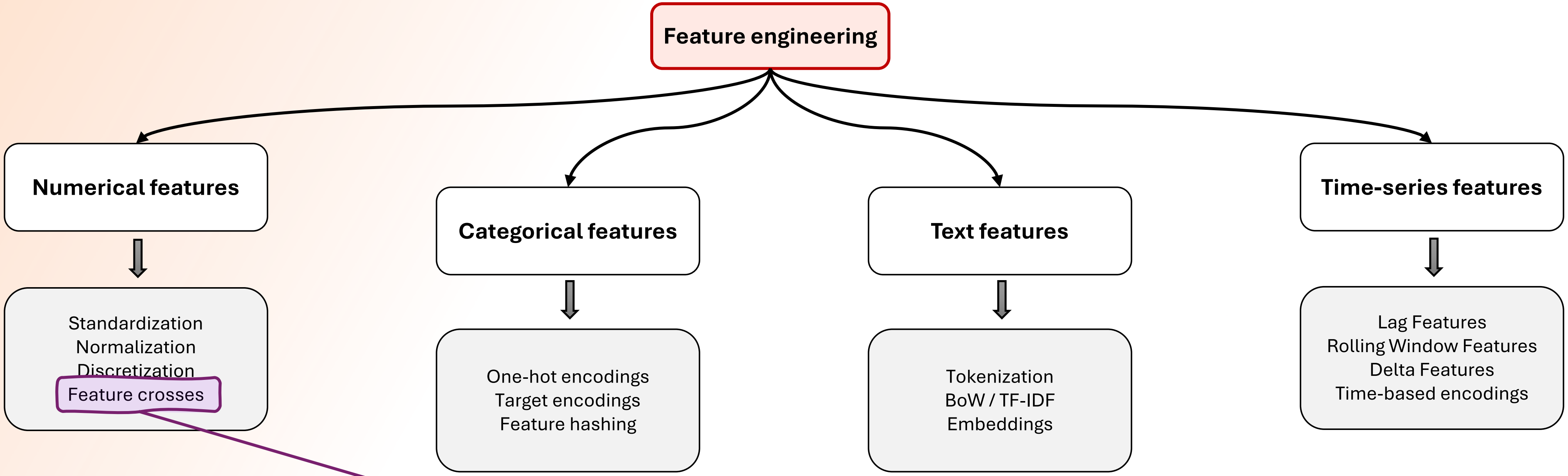
- **Numerical features**
  - Standardization
  - Normalization
  - Discretization
  - Feature crosses
- **Categorical features**
  - One-hot encodings
  - Target encodings
  - Feature hashing
- **Text features**
  - Tokenization
  - BoW / TF-IDF
  - Embeddings
- **Time-series features**
  - Lag Features
  - Rolling Window Features
  - Delta Features
  - Time-based encodings

Converts continuous values into discrete intervals

YES	BUT
• Robustness to noise • Simplifies model training • Useful for limited training data	• Information loss • Poor bin boundaries kill signal • Not suitable for precise regression tasks

x	**x’**
10000	0 - 30k
30000	30k - 60k
70000	60k - 90k
20000	0 - 30k
50000	30k - 60k

Feature engineering concepts



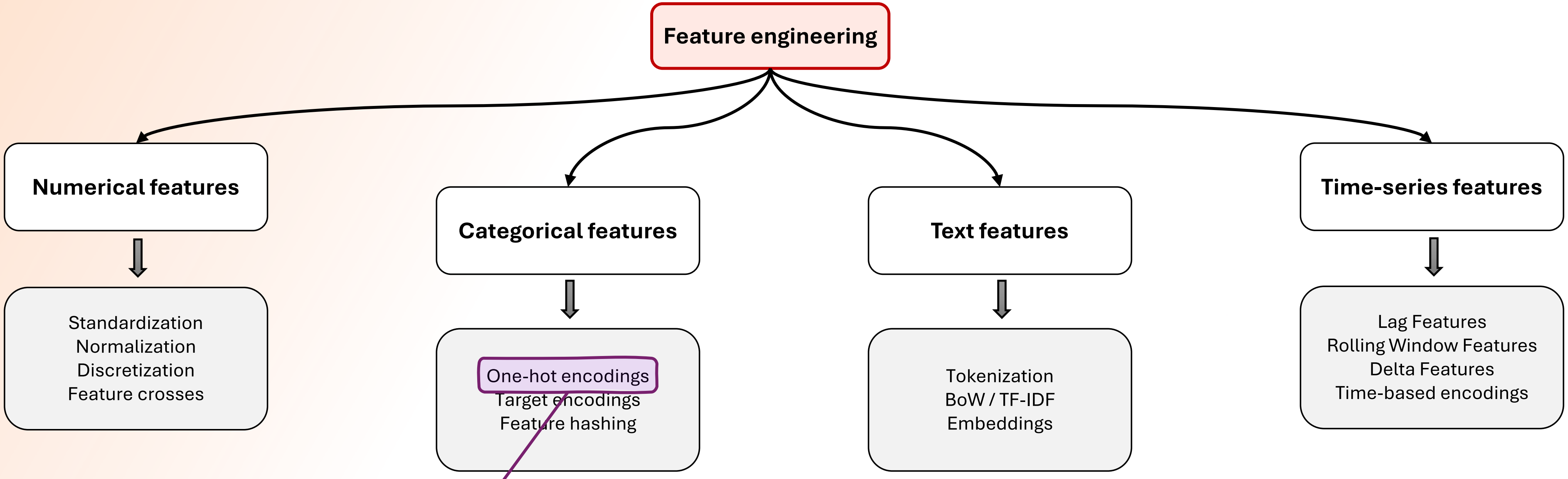
Combines two or more features into new one

YES	BUT
• Captures non-linear relationships • Injects domain knowledge • Allows simple linear models to learn non-linear relations • May speed-up the training for bigger models (like neural networks)	• Feature explosion • Multicollinearity • Easy to overfit • May require domain knowledge

marriage	kids
Yes	0
No	2
Yes	1
Yes	2
No	0

xy
Yes, 0
No, 2
Yes, 1
Yes, 2
No, 0

Feature engineering concepts



Creates a binary column for each category

YES	BUT
• Simple and interpretable • No ordering assumptions	• High cardinality leads to one-hot explosion • Unseen categories at inference • Poor scalability

x	x_cc	x_c	x_o
Credit Card	1	0	0
Cash	0	1	0
Online	0	0	1
Credit Card	1	0	0
Online	0	0	1

Feature engineering concepts

Feature engineering

Numerical features

Standardization
Normalization
Discretization
Feature crosses

Categorical features

One-hot encodings
Target encodings
Feature hashing

Text features

Tokenization
BoW / TF-IDF
Embeddings

Time-series features

Lag Features
Rolling Window Features
Delta Features
Time-based encodings

Replaces categories with target statistics

YES	BUT
• Strong predictive signal • Compact representation • Handles high cardinality	• Very high leakage risk • Requires cross-validation or smoothing • Unstable for rare categories

x	x’
Credit Card	0.42
Cash	0.15
Online	0.76
Credit Card	0.42
Online	0.76

Feature engineering concepts

Feature engineering

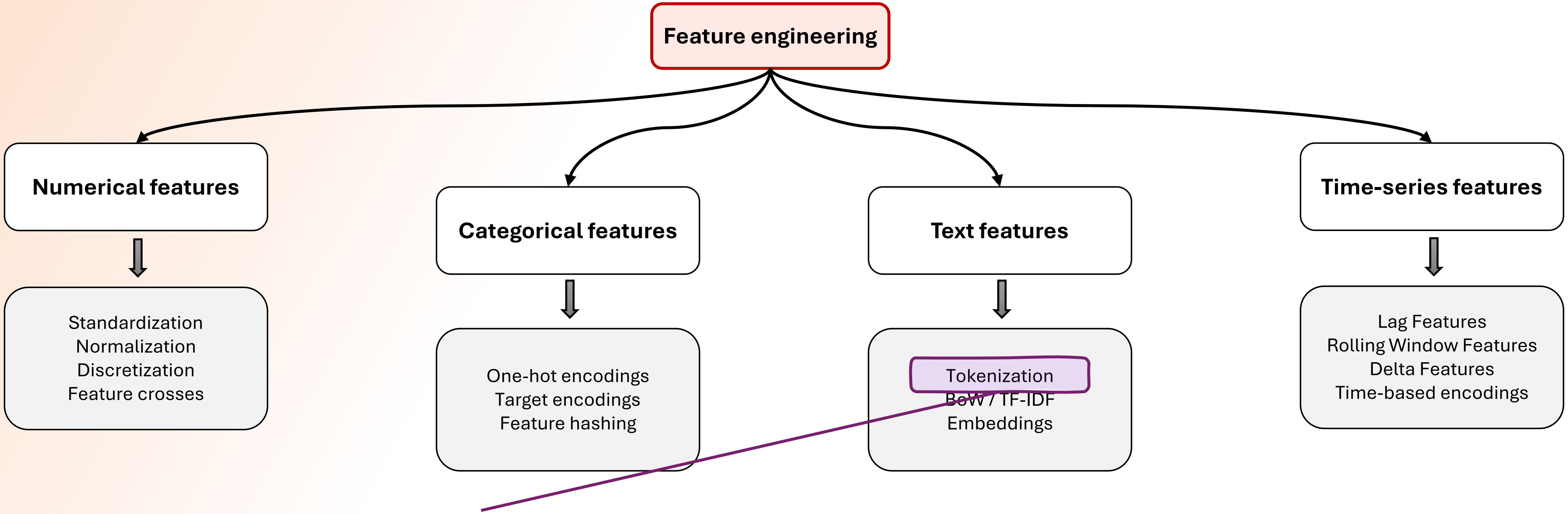
- **Numerical features**
  - Standardization
  - Normalization
  - Discretization
  - Feature crosses
- **Categorical features**
  - One-hot encodings
  - Target encodings
  - Feature hashing
- **Text features**
  - Tokenization
  - BoW / TF-IDF
  - Embeddings
- **Time-series features**
  - Lag Features
  - Rolling Window Features
  - Delta Features
  - Time-based encodings

Maps categories to fixed-size buckets using a hash function

YES	BUT
• Scales to massive cardinality • Constant memory footprint • No dictionary required	• Hash collisions • No interpretability • Hard to debug

x	x’
Credit Card	17
Cash	92
Online	35
Credit Card	17
Online	35

Feature engineering concepts



Splits text into tokens (words, subwords, n-grams)

YES	BUT
• Converts text into processable units • Vocabulary control • Foundation for all text models	• Language and domain dependent • High impact on model quality • Loses context

x		**x’**
"Internet is slow"		[internet, is, slow]
"Service unavailable"		[service, unavailable]
"Billing issue"		[billing, issue]
"Connection drops"		[connection, drops]
"Support was helpful"		[support, was, helpful]

Feature engineering concepts

Feature engineering

- **Numerical features**
  - Standardization
  - Normalization
  - Discretization
  - Feature crosses
- **Categorical features**
  - One-hot encodings
  - Target encodings
  - Feature hashing
- **Text features**
  - Tokenization
  - BoW / TF-IDF
  - Embeddings
- **Time-series features**
  - Lag Features
  - Rolling Window Features
  - Delta Features
  - Time-based encodings

Represents text as token frequency vectors

YES	BUT
• Simple, fast baseline • Interpretable keyword importance • Good for short, technical texts	• Very high dimensional • No semantics or word order • Sparse and noisy • Temporal leakage

x	x'
"internet slow"	[1,1,0]
"slow service"	[0,1,1]
"internet service"	[1,0,1]
"slow slow internet"	[1,2,0]
"service"	[0,0,1]

Feature engineering concepts

Feature engineering

Numerical features
- Standardization
- Normalization
- Discretization
- Feature crosses

Categorical features
- One-hot encodings
- Target encodings
- Feature hashing

Text features
- Tokenization
- BoW / TF-IDF
- Embeddings

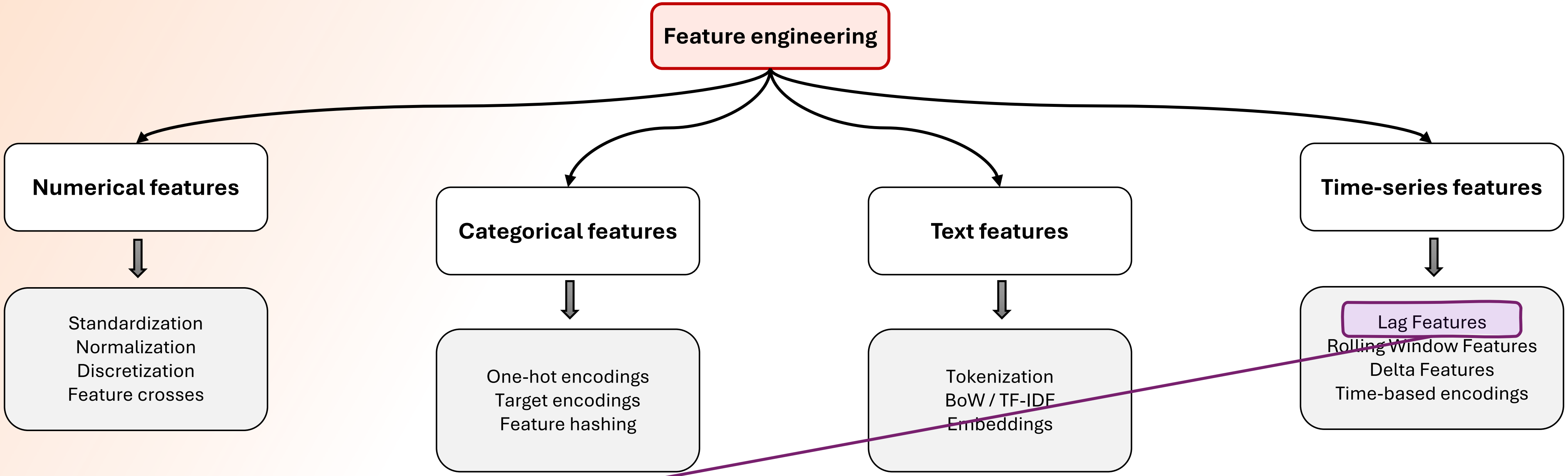
Time-series features
- Lag Features
- Rolling Window Features
- Delta Features
- Time-based encodings

Dense vector representations learned by models

YES	BUT
• Captures semantic meaning • Handles synonyms and context • Low dimensional, dense signal	• Black-box behavior • Domain mismatch • Embedding drift over time • Over-trust

x	x’
"Internet is slow"	[0.12, -0.44, 0.78, ...]
"Service unavailable"	[0.55, 0.10, -0.21, ...]
"Billing issue"	[0.48, -0.33, 0.62, ...]
"Connection drops"	[0.31, 0.27, -0.50, ...]
"Support was helpful"	[-0.12, 0.66, 0.08, ...]

Feature engineering concepts



Past values shifted in time

YES	BUT
• Temporal dependency • Trend detection	• Easy data leakage • Requires strict time ordering

x	t
January	50
February	55
March	53
April	60
May	57

(t-1)
null
50
55
53
60

Feature engineering concepts

Data Community

Feature engineering

Numerical features
- Standardization
- Normalization
- Discretization
- Feature crosses

Categorical features
- One-hot encodings
- Target encodings
- Feature hashing

Text features
- Tokenization
- BoW / TF-IDF
- Embeddings

Time-series features
- Lag Features
- Rolling Window Features
- Delta Features
- Time-based encodings

Aggregations over sliding time windows

YES	BUT
• Smooths noise • Captures recent behavior	• Window size matters a lot • Computationally expensive • Leakage if window crosses target time

x	avg
January	56.7
February	59.3
March	48.5
April	51.1
May	53.9

Feature engineering concepts

Data Community

Feature engineering

- **Numerical features**
  - Standardization
  - Normalization
  - Discretization
  - Feature crosses
- **Categorical features**
  - One-hot encodings
  - Target encodings
  - Feature hashing
- **Text features**
  - Tokenization
  - BoW / TF-IDF
  - Embeddings
- **Time-series features**
  - Lag Features
  - Rolling Window Features
  - Delta Features
  - Time-based encodings

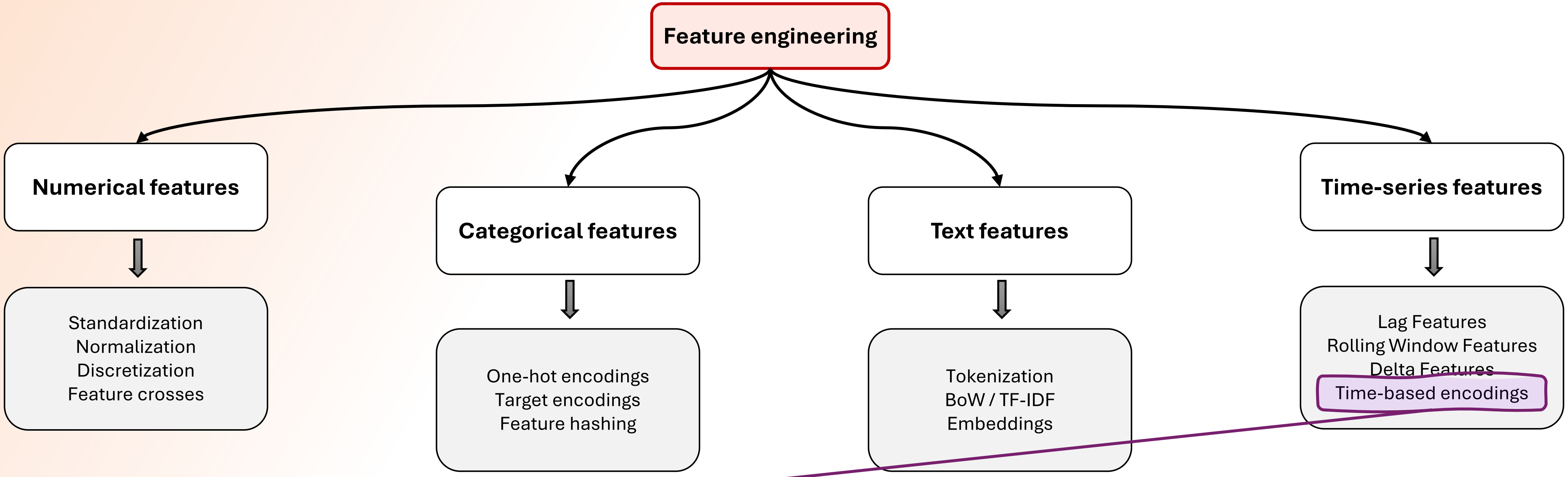
Difference or rate of change over time

YES	BUT
• Behavioral change signals • Early anomaly detection • Trend detection	• Very noisy • Sensitive to missing data

x	x’
January	+3
February	-5
March	+2
April	+7
May	-1

Feature engineering concepts

Data Community



Encodes calendar time information

YES	BUT
• Seasonality patterns • Cyclic behavior modeling	• Useless without seasonality • Needs cyclic encoding • Possible seasonal irrelevance

x	**sin(x) / cos(x)**
Monday	[0.00, 1.00]
Tuesday	[0.78, 0.62]
Wednesday	[0.97, -0.22]
Thursday	[0.43, -0.90]
Friday	[-0.43, -0.90]

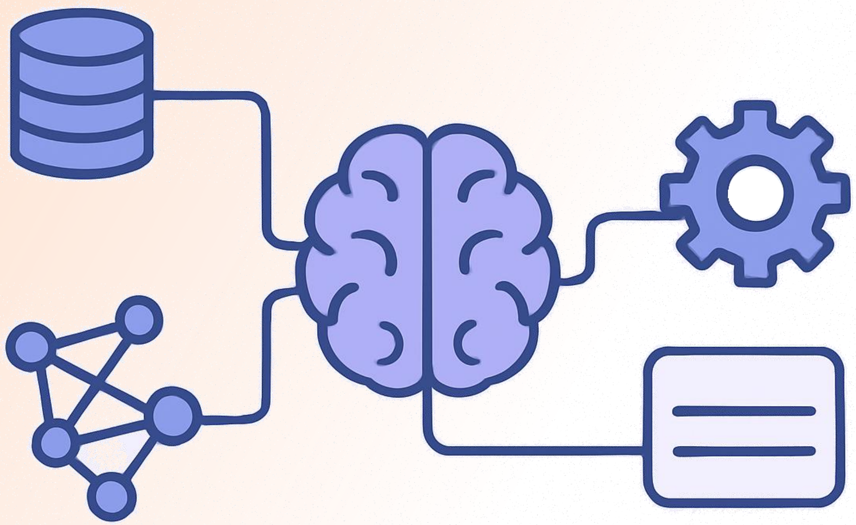
Data leak

A **data leak** is a situation when some additional information about target variable *leaks* to the training set but it's not available during prediction time.

It leads to overly optimistic test results of trained model and spectacular failures in production environments.

Data leak may be very hard to detect. Most common causes include:

- feature leakage
- random splitting of time-correlated data
- data scaling before splitting
- invalid duplication or data imputation handling
- non independent and identially distributed random data



Feature Store and data drift

Feature store

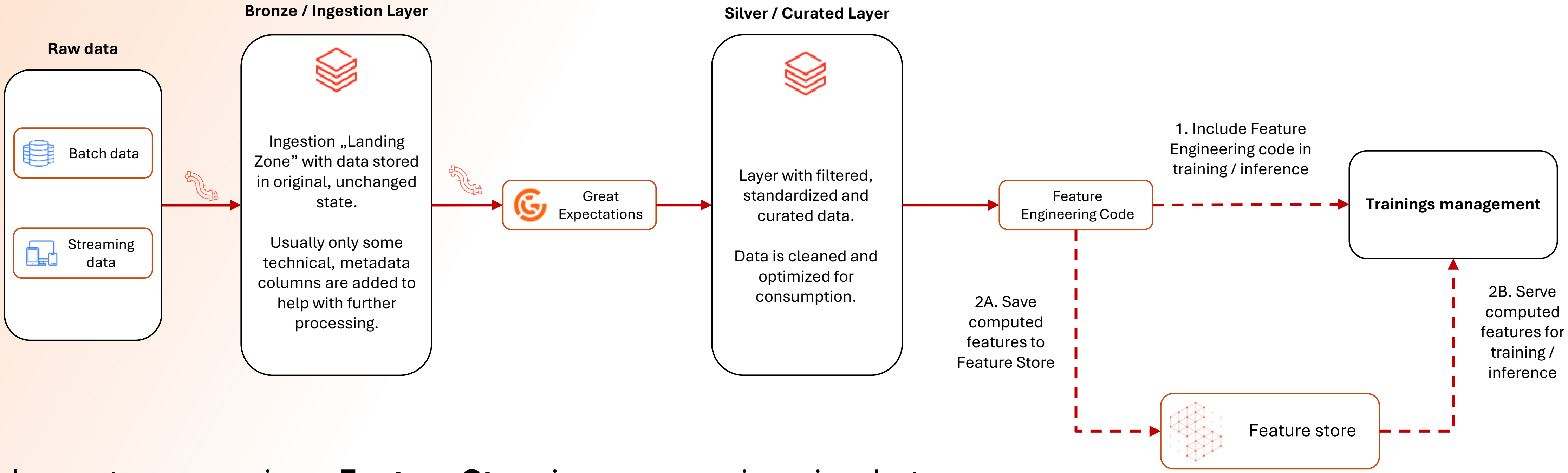
A **Feature Store** is a centralized repository for storing and serving features.

Features are organized as **feature tables**. Each table must have a **primary key** and is backed by a Delta table and additional metadata.

A Feature Store can be **offline** (for training and batch inferencing), **online** (for real-time and online applications).

```
from databricks.feature_engineering import FeatureEngineeringClient, FeatureLookup, FeatureFunction

fe = FeatureEngineeringClient()

# Create feature table with `customer_id` as the primary key.
customer_feature_table = fe.create_table(
  name='ml.recommender_system.customer_features',
  primary_keys='customer_id',
  schema=customer_features_df.schema,
  description='Customer features'
)

# Create feature lookups
features = [
    FeatureLookup(
      table_name='ml.recommender_system.customer_features',
      feature_names=['total_purchases_30d', 'total_purchases_7d'],
      lookup_key='customer_id'
    ),
    FeatureFunction(
        udf_name="ml.recommender_system.extract_user_features",
        input_bindings={"name": "user_name"},
        output_name="user_features"
    )
]

# Create training set
training_set = fe.create_training_set(
  df=customer_df,
  feature_lookups=features,
  label='label',
  exclude_columns=['customer_id']
)
```

Data pipeline



In most cases, using a **Feature Store** is an overengineering, but they may be beneficial in cases of:

- online predictions
- expensive feature computation
- time-sensitive predictions
- sharing the features across many models

Data drift

Data drift occurs when the input data changes over time compared to the data used to train the model, causing model performance to degrade.

Most common data drift types:

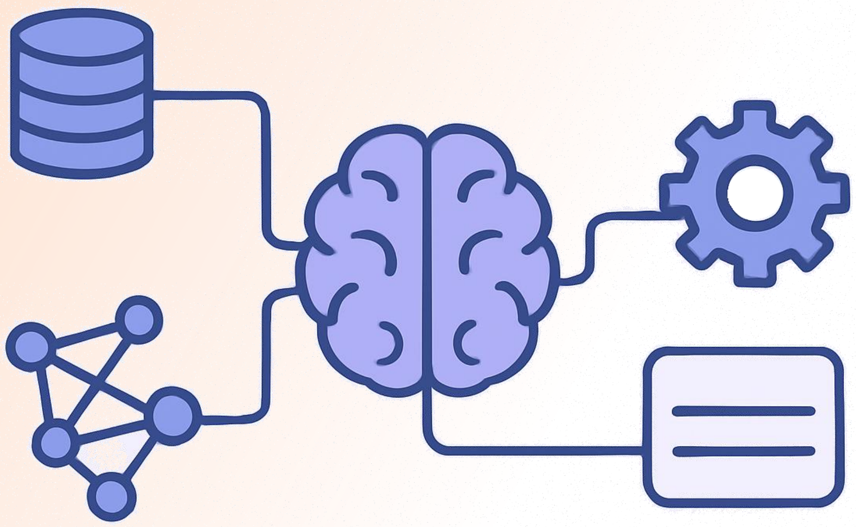
- Covariate drift (the distribution of input features changes)
- Concept drift (the relationship between features and the target changes)
- Label drift (the distribution of the target variable changes)

Data drift monitoring

Data drift detection methods:

- **Feature distribution comparison**: PSI, KL-divergence, JS-divergence, KS-test
- **Rolling statistics**: mean, standard deviation, quantiles over time windows
- **Embedding drift tracking**: changes in vector norms, centroids, cosine distance
- **Model monitoring**: changes in model predictions distribution or metrics

DEMO



Summary

Takeaways

Features encode assumptions: every transformation is a hypothesis about how the world works; if you can't explain why a feature exists, the model shouldn't use it.

Leakage beats any model architecture: a simple model with clean, well-timed features will outperform the most advanced model trained on leaked or future-aware data.

Drift is inevitable: data will change over time; the real risk is not detecting when distributions, semantics, or behavior have shifted.

Feature pipelines are production systems, not experiments: reproducibility, versioning, and consistency across training and serving matter more than squeezing out the last 0.5% of offline metrics.

Thank you!

Data Community

Contact:

https://www.linkedin.com/in/maciej-kepa

https://github.com/maciejkepa/ai-ml-in-practice