



**Program:** MSc of Data Science  
**Module:** Big Data Tools and Techniques

Week 7

Machine Learning in Spark and the  
Machine Learning Lifecycle

2025

# Expectations

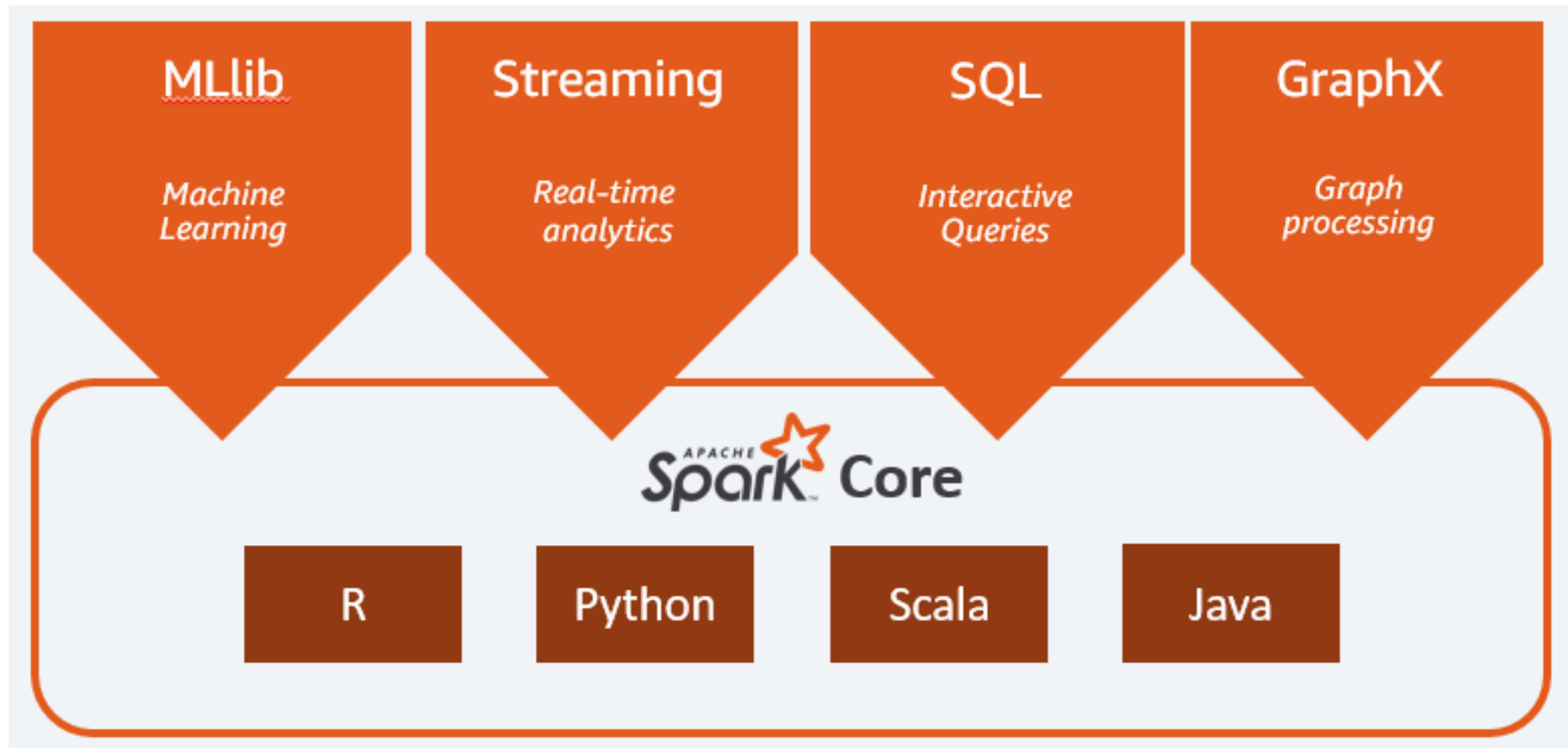
1. Choose a quiet place to attend the class and please concentrate during the lecture
2. Don't raise hand or ask a question during the lecture
3. Put your questions in Padlet (not Teams' chat box) and I will review them in the due time (Padlet link is in the Bb, week 7, Lecture) – we will also end on a Q&A if there is time!
4. Turn off your mic during the lecture
5. We will have 5 mins break after the first hour of the lecture (please remind me)
6. Jisc code will be shared during the break time

# Learning Outcomes

By the end of this lecture and workshop, you will be able to:

1. Discuss how supervised machine learning can be used to solve problems in real-world scenarios
2. Apply classification algorithms to a dataset using the MLlib library, interpret the result and evaluate model performance
3. Explain how tools such as MLflow can support the machine learning lifecycle

# Apache Spark



<https://aws.amazon.com/what-is/apache-spark/>

# What we'll cover (and what we won't!)

- The aim of today's lecture is to cover some core machine learning concepts, which we will be applying in the workshop. **For some of you this will be an introduction, for others a refresh.**
- The MLDM provides a more comprehensive, thorough and complete introduction – however, we want you to see how machine learning fits within the wider picture when working with Big Data in Spark (which is why we cover it here)
- We'll also cover the machine learning lifecycle and discuss how a platform like MLflow can help an organisation manage the machine learning lifecycle effectively

# What is machine learning?

*“Computer programming requires a programmer. Humans instruct a computer to solve a problem by specifying each and every step through many lines of code. But with machine learning... you can let the computer try to solve the problem itself.”*

- Kaz Sato, Developer Advocate, Google Cloud, “Understanding neural networks with TensorFlow Playground”

<https://cloud.google.com/blog/products/ai-machine-learning/understanding-neural-networks-with-tensorflow-playground>

# What is machine learning?


*“Computer programming requires a programmer. Humans instruct a computer to solve a problem by specifying each and every step through many lines of code. But with machine learning... you can let the computer try to solve the problem itself.”*

- Kaz Sato, Developer Advocate, Google Cloud, “Understanding neural networks with TensorFlow Playground”

<https://cloud.google.com/blog/products/ai-machine-learning/understanding-neural-networks-with-tensorflow-playground>

# Classical Machine Learning

Task Driven



## Supervised Learning

( Pre Categorized Data )



Classification

( Divide the  
socks by Color )

Eg. Identity  
Fraud Detection



Regression

( Divide the  
Ties by Length )

Eg. Market  
Forecasting

Data Driven



## Unsupervised Learning

( Unlabelled Data )



Clustering

( Divide by  
Similarity )

Eg. Targeted  
Marketing



Association

( Identify  
Sequences )

Eg. Customer  
Recommendation



Dimensionality  
Reduction

( Wider  
Dependencies )

Eg. Big Data  
Visualization

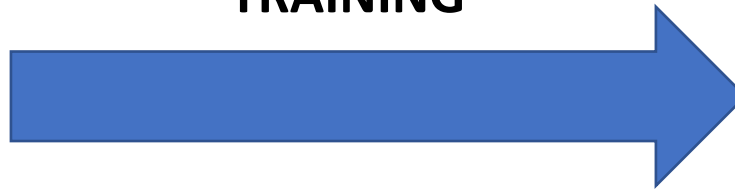


# Supervised Learning

Labelled data which includes the input variables and the output variable we want to predict



TRAINING



Following training, we have a model which can predict the output variable based on the input variables

In supervised learning we use a labelled dataset to train a model which can predict the output given a particular input

# Supervised Learning

What's the question we want to answer?	What INPUT data might we have?	What OUTPUT are we trying to predict?	Type of Problem
How can I predict if a patient has diabetes?	Medical records (e.g., blood sugar levels, urine test results etc.)	Diabetes Present = TRUE or FALSE	Classification (Binary)
How can I predict if a customer is going to renew their car insurance next year?	Price of insurance, number of years they have renewed, make / model of car. etc.	Insurance Renewed = TRUE or FALSE	Classification (Binary)
How can I predict the price of a house?	Size of house, number of bedrooms, location, etc.	Predicted house price in £	Regression

# Supervised Learning

What's the question we want to answer?	What INPUT data might we have? (Features)	What OUTPUT are we trying to predict?	Type of Problem
How can I predict if a patient has diabetes?	Medical records (e.g., blood sugar levels, urine test results etc.)	Diabetes Present = TRUE or FALSE	Classification (Binary)
How can I predict if a customer is going to renew their car insurance next year?	Price of insurance, number of years they have renewed, make / model of car. etc.	Insurance Renewed = TRUE or FALSE	Classification (Binary)
How can I predict the price of a house?	Size of house, number of bedrooms, location, etc.	Predicted house price in £	Regression

# Supervised Learning

What's the question we want to answer?	What INPUT data might we have?	What OUTPUT are we trying to predict?	Type of Problem
How can I predict if a patient has diabetes?	Medical records (e.g., blood sugar levels, urine test results etc.)	Diabetes Present = TRUE or FALSE	Classification (Binary)
How can I predict if a customer is going to renew their car insurance next year?	Price of insurance, number of years they have renewed, make / model of car. etc.	Insurance Renewed = TRUE or FALSE	Classification (Binary)
How can I predict the price of a house?	Size of house, number of bedrooms, location, etc.	Predicted house price in £	Regression

# Breakout Session

10 mins in breakout groups to come up with suggestions for use cases for classification. Come up with as many ideas as you can and share on Padlet.

Think about what output you're trying to predict and what data (features) you might want to use to help make the prediction.

Try to think of use cases across different industries, e.g.:

- Financial Services
- Ecommerce
- Healthcare
- Manufacturing

# Examples of Classification

- Classifying credit card transactions as legitimate or fraudulent
- Predicting whether a customer or web visitor is going to buy a particular product based on their past habits and / or their demographic information (e.g. location, age)
- Predicting whether a customer is likely to churn (stop buying our services)
- Predicting whether or not an individual has a particular disease (e.g. whether a tumour is benign or malignant)
- Predicting the credit risk of an individual or business before we offer them a loan
- Classifying human activity based on phone / smart watch accelerometer data (e.g. is the person walking, running, swimming etc)

# Classification Example

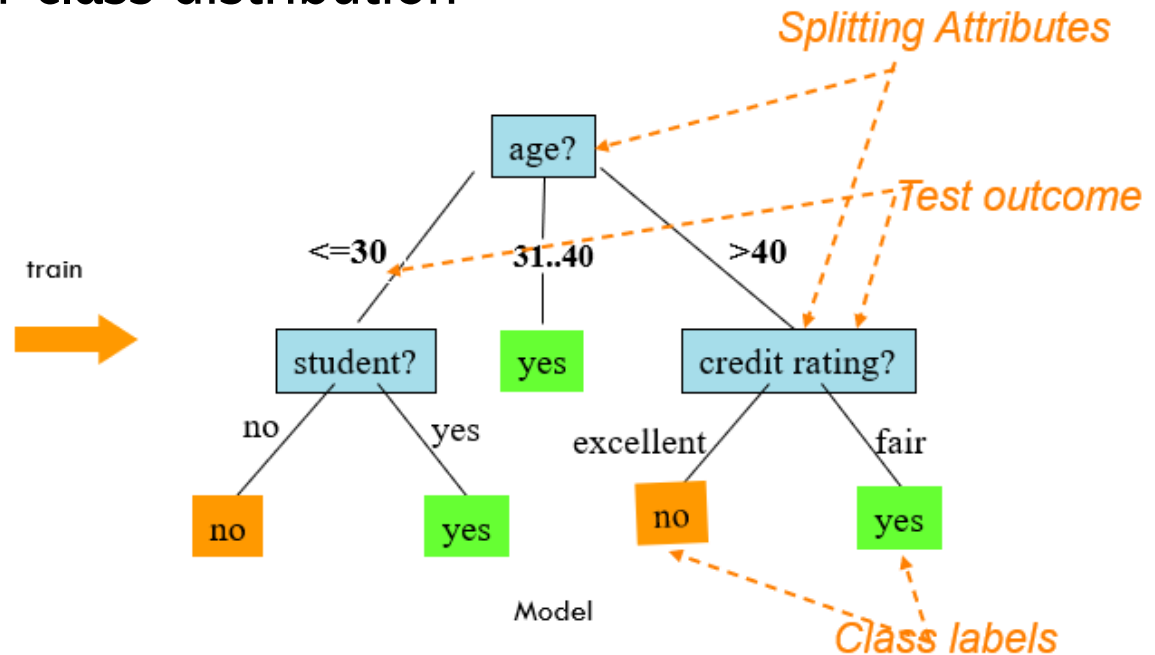
- Goal: Predict fraudulent cases in credit card transactions.
- Approach:
  - Use credit card transactions and the information on its account-holder as attributes.
  - When does a customer buy, what they buy, how often they pay on time, etc
  - Label past transactions as fraud or fair transactions. This forms the target that we want the model to learn how to predict
  - Train a model for the type of transaction (i.e., legitimate or fraudulent).
  - Use this model to detect fraud in future by observing credit card transactions on an account.

# Example: Decision Trees

- **Internal node** denotes a decision node (splitting attributes)
- **Branch** shows the values of the attribute
- **Leaf nodes** represent class labels or class distribution

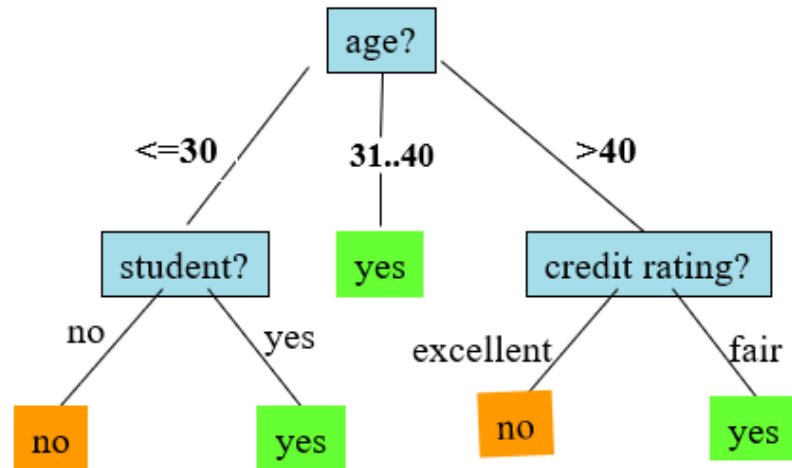
age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no

Training dataset





# Example: Decision Trees

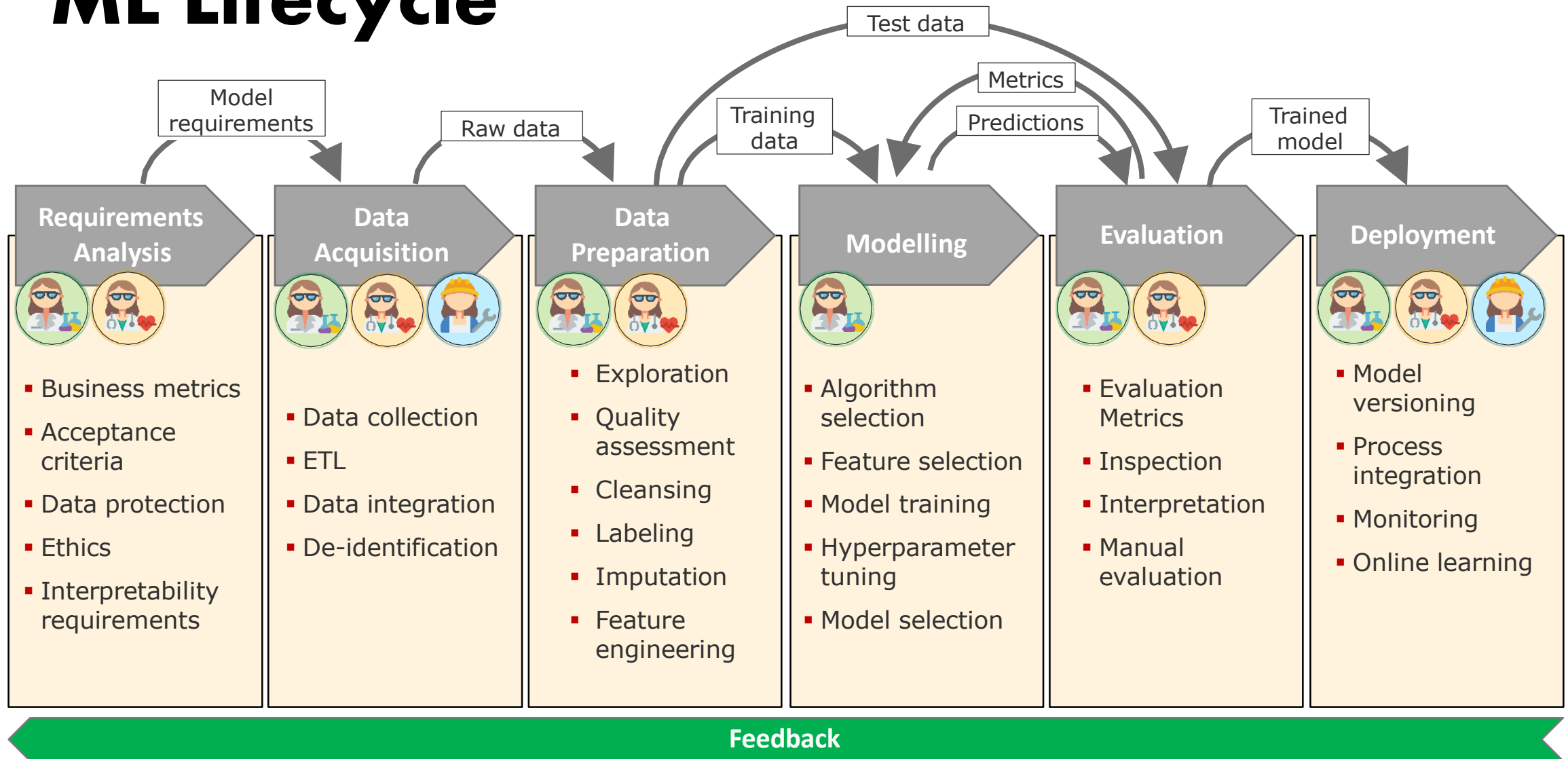


Apply

age	Income	Student	credit_rating	buys_computer
≤30	No	Yes	fair	?

Test data

# ML Lifecycle



# Machine Learning Libraries



TensorFlow



Keras



PyTorch





# Why use MLlib?

- There are numerous packages for performing machine learning on a single machine – e.g. Scikit Learn, Tensorflow.
- However, these have limits (either in terms of size of dataset you can train on or the processing time)
- If you reach those scalability issues – MLlib makes distributed machine learning easy
- Provides implementations of ML algorithms which can be efficiently distributed across a cluster (e.g., Decision Trees but not KNN)
- Spark MLlib provides tools for feature engineering and data preprocessing as well as for ML

# Why use MLlib?

- Two use cases for using Spark for machine learning:
  - Use Spark for data pre-processing and feature engineering to produce training and test datasets from large amounts of data and then use a single-machine learning library to train a model on this dataset
  - When training data or model size becomes too difficult or inconvenient for one machine use Spark's full machine learning capabilities

Remember, the purpose of using Spark is when we are working with Big Data and processing on a single node / machine becomes difficult or impractical. This applies to using Spark for Machine Learning too!

# Why use MLlib?

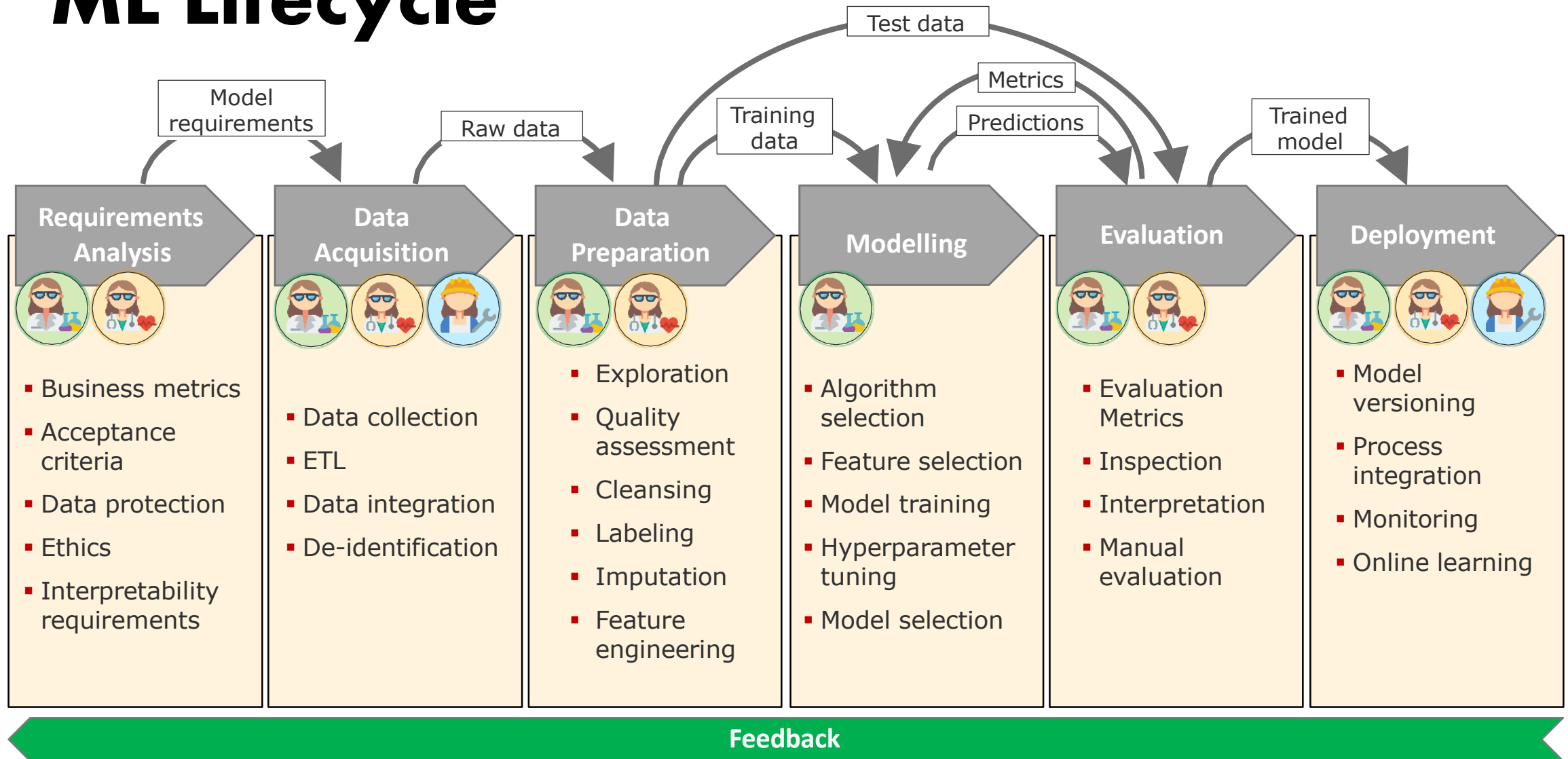
Feature	Scikit-learn	TensorFlow/PyTorch	Spark MLlib
<b>Handles large datasets</b>	No	Yes (with GPU)	Yes (distributed)
<b>Scales across multiple machines</b>	No	No	Yes
<b>Best for</b>	ML with small or medium datasets (ideal range is up to 100,000s of rows)	Deep learning	Big data ML



# Introduction to MLlib

- Spark has two machine learning packages:
  - `spark.mllib` is the original machine learning API which leverages RDDs but is now in maintenance mode
  - `spark.ml` is a newer machine learning API which is based on DataFrames – this is the one we focus on today and in the workshop
- However, the term ‘MLlib’ is still used as an umbrella term for both packages in Apache Spark, so we will refer to it as MLlib throughout.

# ML Lifecycle





# Data Acquisition

- ETL (Extract, Transform, Load) is a data integration process where data is extracted from multiple sources, transformed into a structured format, and then loaded into a target system (such as a data warehouse or data lake) for analytics or machine learning.
  - Many of the data preprocessing tasks covered in previous workshops are essential steps in an ETL pipeline, helping to prepare data for machine learning and analytics.
- Apache Spark is a powerful tool for big data ETL because it supports distributed data processing, works with various data sources (structured, semi-structured, streaming), and enables scalable feature engineering for machine learning.

# Data Exploration

- Once we have gathered a data, we typically explore it before starting any modelling – this stage is known as Exploratory Data Analysis (EDA)
- This involves querying the data, creating data visualisations and computing summary statistics to understand details such as the distribution of your variables, relationships between them etc.
- Can use Spark SQL and Spark DataFrames functions to do this

# Data Pre-processing

Typically, we will need to pre-process (transform) our data in some way before we can pass it to a model to train, for example:

- Removing features / adding features
- Converting categorical features to the right format
- Scaling or normalising data
- Converting data into correct format for MLlib learning algorithm. All inputs into machine learning algorithms in MLlib must be of the following type:
  - Labels should be a single numerical value of Double type
  - Features should be a vector of Double type
  - Double type is a double-precision floating point number

# Data Pre-processing

Original features					Transformed features and labels	
Temperature	Humidity	CO2	HumidityRatio	Occupancy	features	label
21.7675	31.1225	1009.5	0.005021569	1	[21.7675,31.1225,...	1.0
21.79	31.46333333	1027.333333	0.005084053	1	[21.79,31.46333333...	1.0
21.89	31.6	1060.5	0.005137857	1	[21.89,31.6,1060....	1.0
21.89	31.73	1100.0	0.005159169	1	[21.89,31.73,1100...	1.0
21.89	30.65	896.5	0.004982159	1	[21.89,30.65,896....	1.0

MLlib provides several transformers to help transform our data to the required format (e.g. RFormula, VectorAssembler)

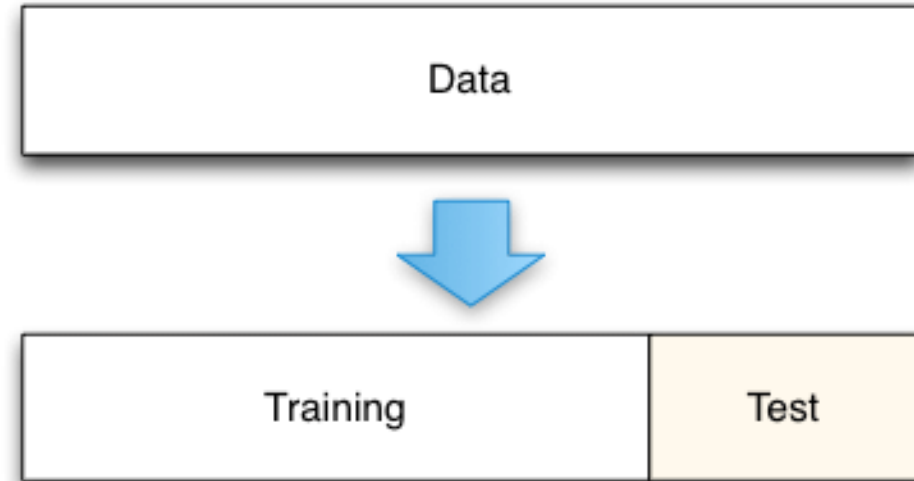
# Test / train split

We want to know if we can use our model to make predictions about the data where we don't have a class label

But we have already provided the class label for our training dataset, so this isn't a good judge of how a model will perform on new, unseen data.

The solution – split our data into two parts and only use some of it to train our model.

We can use `randomSplit` to split out DataFrame into two parts.



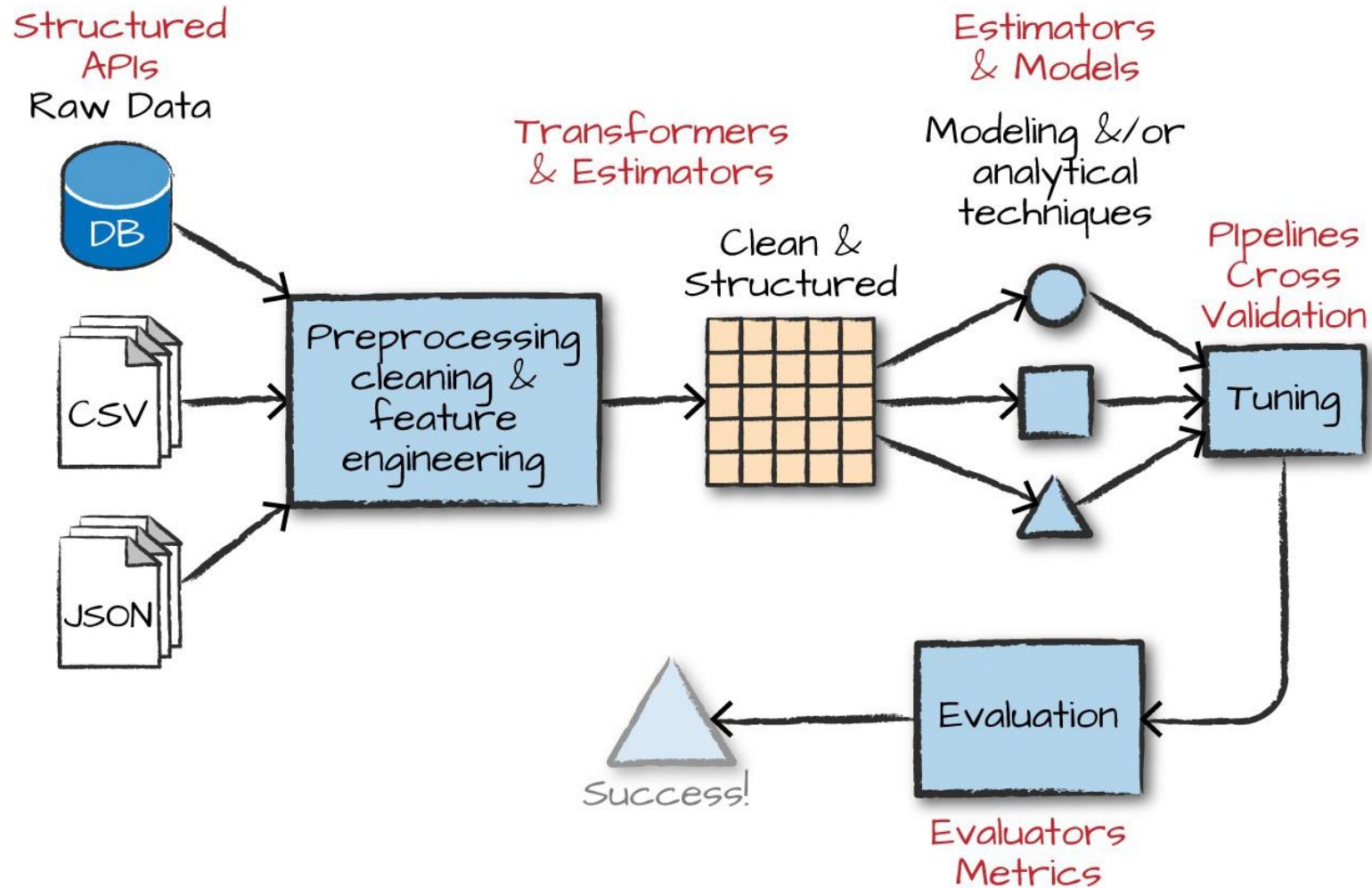
```
(trainingDF, testDF) = occupancyDF.randomSplit([0.7, 0.3])
```

Vector defines the proportion of the data we want in each split

# Model Training

- Model training is the process of applying a machine learning algorithm to a prepared dataset (training data) to create a predictive model.
- In Spark ML (MLlib), training a model typically involves:
  - Selecting an algorithm, such as Logistic Regression, Decision Tree, or Random Forest.
  - Configuring model hyperparameters, such as the number of iterations, learning rate, or tree depth.
  - Fitting the model by calling the `.fit()` method on your training dataset.
- Evaluating the performance of the model (this is what we use the test set for)

# Introduction to MLlib



*Spark: The Definitive Guide: Big Data Processing Made Simple, Matei Zaharia*

# Main MLlib Concepts

In MLlib there are several concepts we need to understand:

- Transformers
- Estimators
- Evaluators
- Pipelines



# Estimator

- Estimator: An Estimator is an algorithm which can be fit on a DataFrame to produce a Transformer.
  - E.g., a learning algorithm is an Estimator which trains on a DataFrame and produces a model.
- An Estimator implements a fit() method, in other words we can call its fit() method to train a model – i.e., to learn or ‘fit’ parameters using the DataFrame

Instantiate  
the model

```
from pyspark.ml.classification import DecisionTreeClassifier  
dt = DecisionTreeClassifier(labelCol="label", featuresCol="features")
```

```
model = dt.fit(trainingDF)
```

Call the fit  
method

# Transformers


- Transformer: A Transformer is an algorithm which can transform one DataFrame into another DataFrame.
  - E.g., a simple transformer could be used to convert a categorical string variable (such as 'Yes', 'No) into numeric data so it can be input into a model
- A Transformer implements a transform() method, in other words we can call its transform() method on a DataFrame to produce a new DataFrame with the desired transformations applied.
- Using the fit() method on an Estimator produces a model, which is a Transformer. We can then call the transform() method to generate predictions

```
predictions = model.transform(testDF)
```

# Generating Predictions

Using the `.transform()` method on the trained model generates a new DataFrame with the prediction columns appended. The prediction column is the class prediction and the probability column is a vector of probabilities that the example belongs to each class.

Columns appended



Temperature	Humidity	CO2	HumidityRatio	Occupancy	features	label	rawPrediction	probability	prediction
19.2	30.7	429.0	0.004222155	0	[19.2,30.7,429.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.245	31.65	435.0	0.004366035	0	[19.245,31.65,435.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.26	31.46333333	431.6666667	0.004344191	0	[19.26,31.46333333,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.29	26.79	469.0	0.003702057	0	[19.29,26.79,469.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.29	27.5	432.0	0.00380077	0	[19.29,27.5,432.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.29	30.63333333	441.0	0.004236777	0	[19.29,30.63333333,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.29	30.745	425.0	0.004252327	0	[19.29,30.745,425.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.34	30.6	430.0	0.004245423	0	[19.34,30.6,430.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.39	27.1	468.5	0.003768685	0	[19.39,27.1,468.5,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.39	27.2	460.0	0.003782676	0	[19.39,27.2,460.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.39	27.5	440.6666667	0.003824654	0	[19.39,27.5,440.6,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.39	31.1	436.0	0.00432882	0	[19.39,31.1,436.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.39	31.2	434.0	0.004342836	0	[19.39,31.2,434.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.39	31.29	435.0	0.004355451	0	[19.39,31.29,435.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.39	31.29	436.3333333	0.004355451	0	[19.39,31.29,436.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.5	27.2	452.0	0.00380881	0	[19.5,27.2,452.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.5	27.6	444.0	0.00386517	0	[19.5,27.6,444.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.5	27.79	449.0	0.003891944	0	[19.5,27.79,449.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0

# Model Evaluation

- So how do we know if we have a good model?
- We need to use a performance metric to judge how good our model is.
- A common metric is accuracy, which is simply a measure of what proportion of the predictions were correct (we will cover other metrics in MLDM)

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

- In MLlib we can use an Evaluator to calculate a performance metric, given a set of predictions and true class labels

# Evaluator

- Evaluator: An Evaluator allows us to see how a trained model performs according to a metric we choose
  - E.g., accuracy

```
# use evaluator to measure accuracy of predictions on test data

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")

accuracy = evaluator.evaluate(predictions)

print("Accuracy = %g " % (accuracy))
```

Select the  
evaluation  
metric

Use evaluate()  
method to  
calculate metric

# Pipelines

- In MLlib we can also use pipelines to chain together data preprocessing steps and model fitting. We call `fit()` on the training data which returns a `PipelineModel` and we can then use `transform()` to generate predictions for the test data.

```
from pyspark.ml import Pipeline

# Define pipeline

pipeline = Pipeline(stages=[preprocess, dt])

# Fit pipeline model

model = pipeline.fit(trainingDF)


# Generate predictions for test set

predictions = model.transform(testDF)


# Evaluate accuracy

accuracy = evaluator.evaluate(predictions)
print("Accuracy = %g " % (accuracy))
```

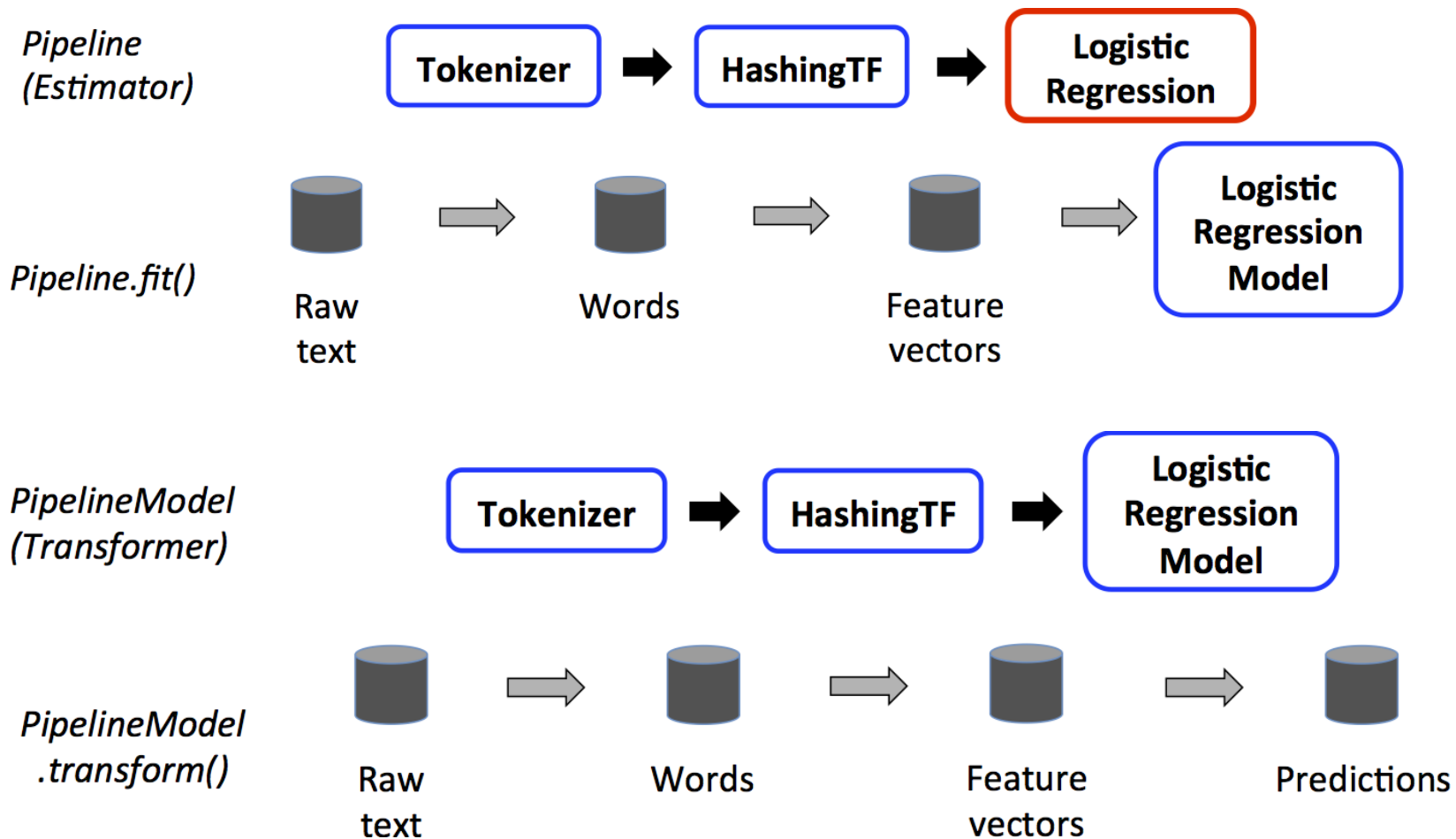
Define pipeline stages (based on instantiated Estimators earlier)



Use `fit()` to fit the pipeline and `transform()` to generate new predictions



# Pipelines



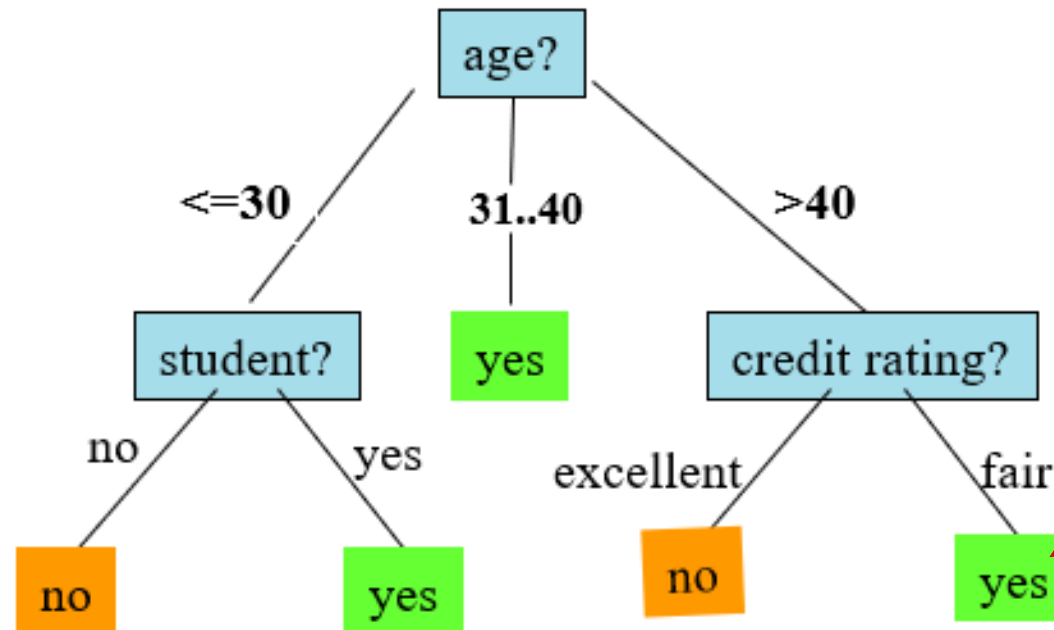
# Parameters vs Hyperparameters

- Parameters are the internal values that a machine learning model learns from the training data. They directly influence how the model makes predictions.
  - For a decision tree, parameters are values such as which values to split on, the thresholds used etc
- Hyperparameters are external settings that control how the model is trained. Unlike parameters, they are not learned from the data but are set manually before training begins.
  - For a decision tree and maximum depth, or the maximum number of splits before you reach a leaf node, is a hyperparameter



# Example Hyperparameters

maxDepth  
determines the  
maximum number  
of splits that you  
will need to go  
through to reach a  
leaf node



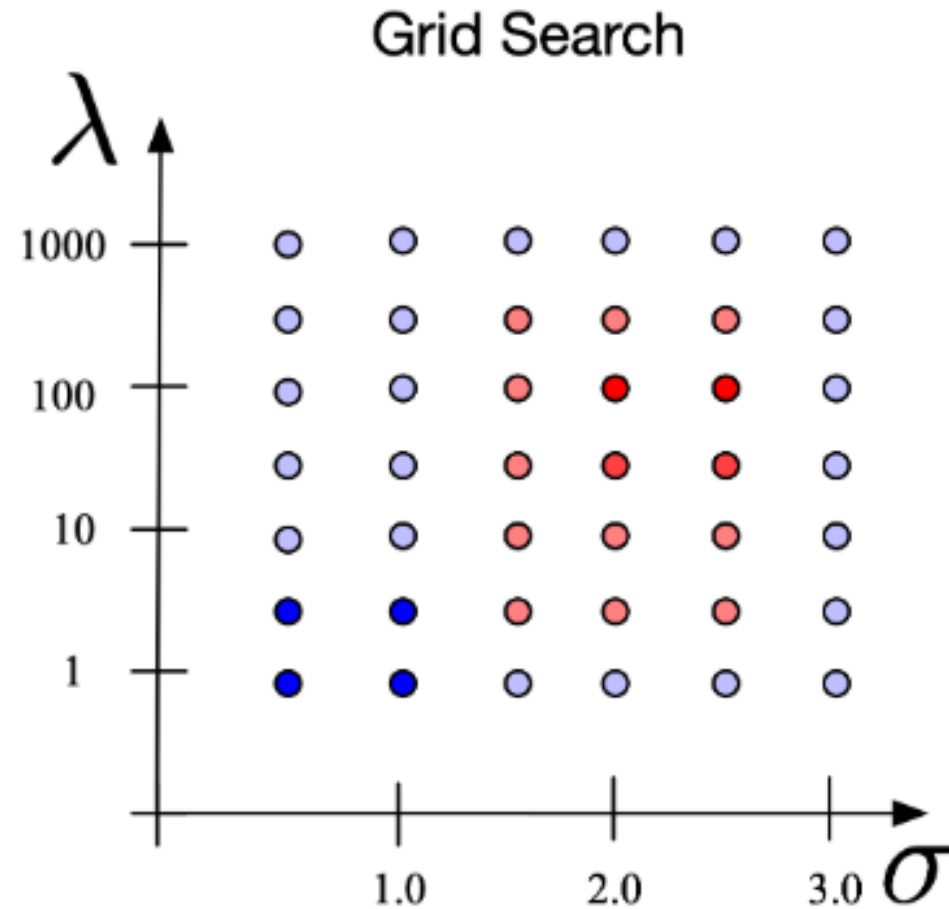
minInstancesPerNode  
– is the minimum  
number of training  
examples that can be  
left at any leaf node

# Hyperparameter Tuning

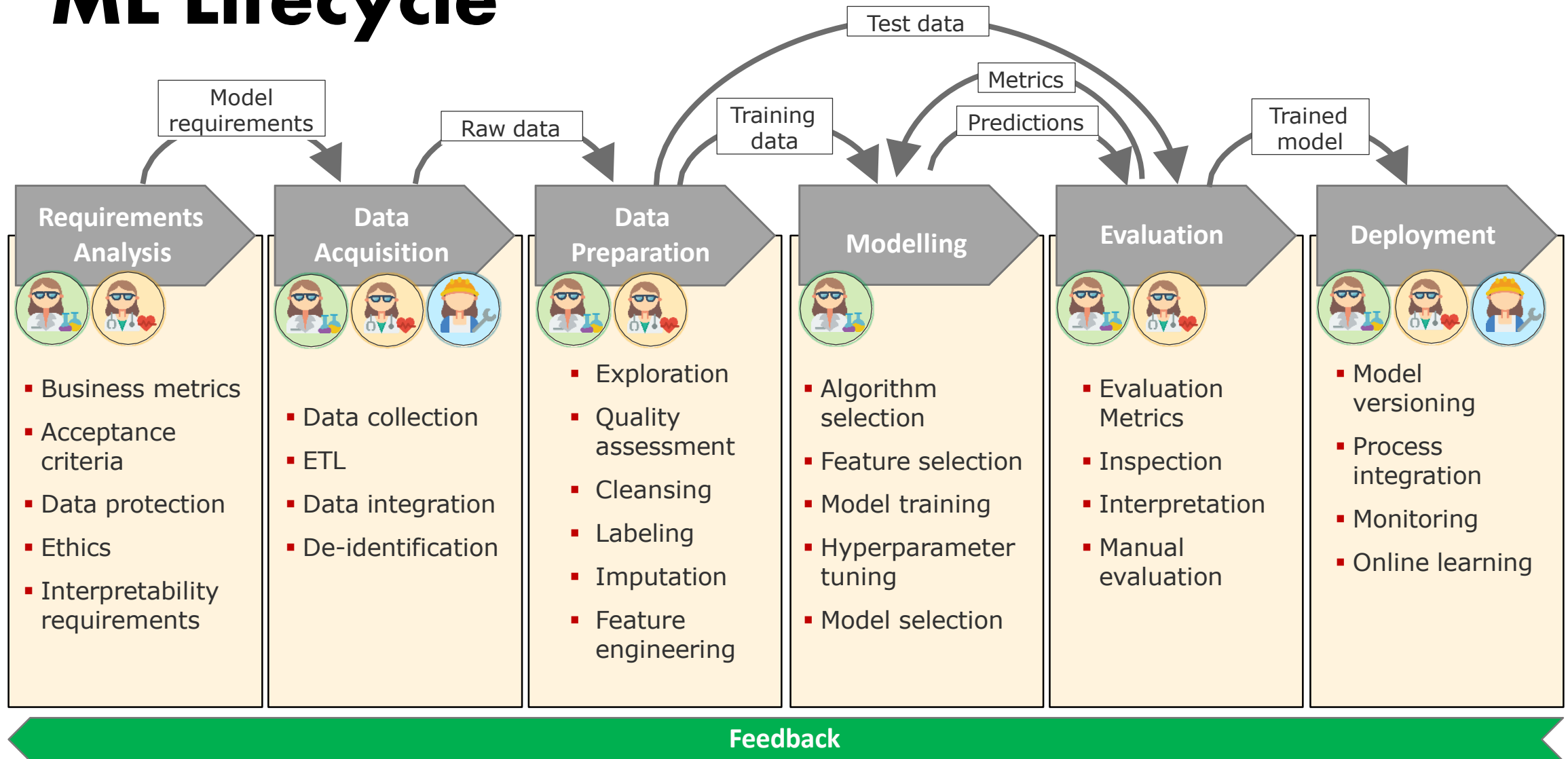
- Hyperparameter tuning is the process of selecting the optimal values for any hyperparameters to our model
- We can set hyperparameter values manually – e.g., based on default values which are known to work well
- However, we can also use more systematic ways of finding optimal values for hyperparameters experimentally by trying different combinations to see what works best

# Grid Search

- We can do this experimentally by trying different combinations of parameters to see which performs best – one approach to this is grid search
- Doing this is part of the modelling phase of the machine learning cycle
- Each machine learning algorithm has its own hyperparameters which we can tune (we discuss specific hyperparameters for each algorithm in MLDM)



# ML Lifecycle



# Model Deployment

- The point of building a machine learning is so we can use it to make predictions on data 'in the wild' in future!
- We have discussed how to train a machine learning model. But how do we now go on to manage and deploy the models we have trained?
  - Batch ML: Train models periodically on historical data.
  - Streaming ML: Train models in real-time on incoming data (e.g., fraud detection).

Use Case	Batch Prediction	Streaming
Fraud detection	No	Yes
Predicting customer churn	Yes	No
Product recommendation	Yes	Yes
Predictive maintenance	Yes	Yes

# Model Deployment

- The point of building a machine learning is so we can use it to make predictions on data 'in the wild' in future!
- We have discussed how to train a machine learning model. But how do we now go on to manage and deploy the models we have trained?
  - Batch ML: Train models periodically on historical data.
  - Streaming ML: Train models in real-time on incoming data (e.g., fraud detection).

	Throughput	Latency	Example application
Batch	High	High (hours to days)	Customer churn prediction
Streaming	Medium	Medium (seconds to minutes)	Dynamic pricing
Real-time	Low	Low (milliseconds)	Online ad bidding

# ML Workflow Challenges

Getting to the point of deploying a model to a production system can be challenging:

- **It's difficult to keep track of experiments.** When you are just working with files on your laptop, or with an interactive notebook, how do you tell which data, code and parameters went into getting a particular result?
- **It's difficult to reproduce code.** Even if you have meticulously tracked the code versions and parameters, you need to capture the whole environment (for example, library dependencies) to get the same result again. This is especially challenging if you want another data scientist to use your code, or if you want to run the same code at scale on another platform (for example, in the cloud)

<https://mlflow.org/docs/latest/concepts.html>

# ML Workflow Challenges (Cont.)

Getting to the point of deploying a model to a production system can be challenging:

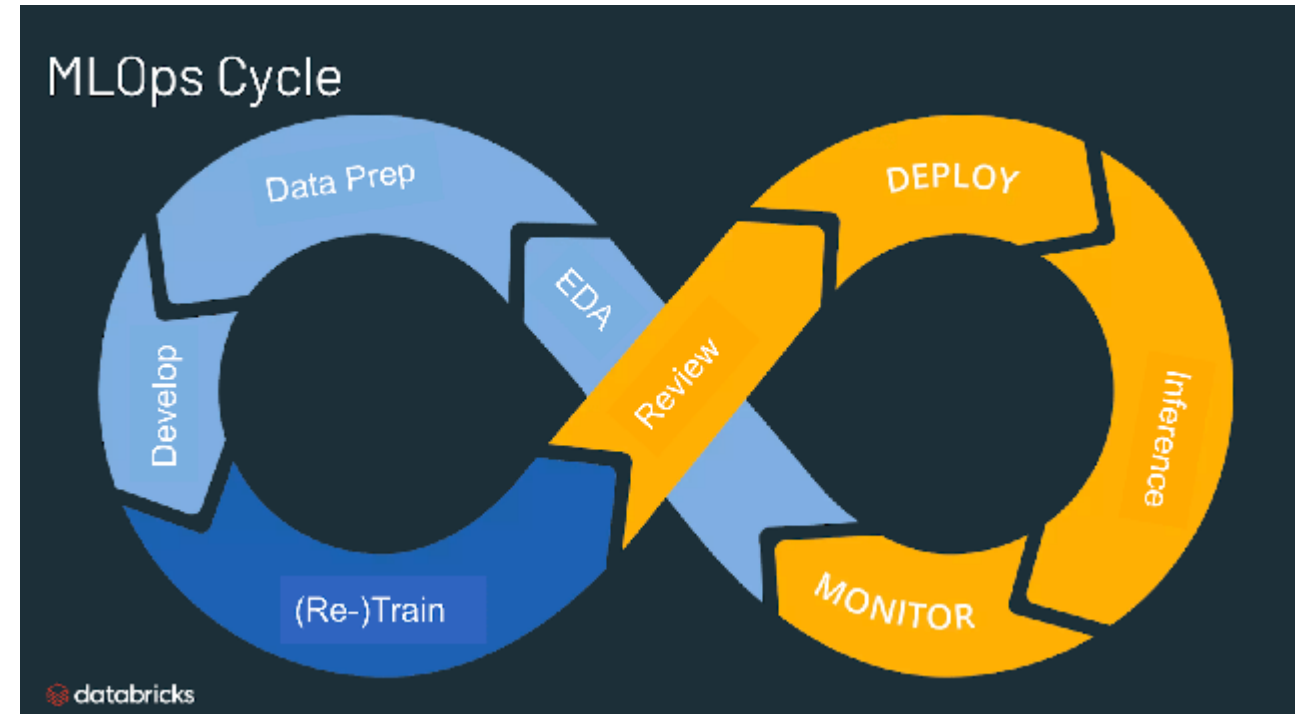
- **There's no standard way to package and deploy models.** Every data science team comes up with its own approach for each ML library that it uses, and the link between a model and the code and parameters that produced it is often lost.
- **There's no central store to manage models (their versions and stage transitions).** A data science team creates many models. In absence of a central place to collaborate and manage model lifecycle, data science teams face challenges in how they manage models stages: from development to staging, and finally, to archiving or production, with respective versions, annotations, and history.

<https://mlflow.org/docs/latest/concepts.html>



# MLflow & MLOps

- MLOps is “a set of practises that focuses on streamlining the process of taking machine learning models to production, and then maintaining and monitoring them. MLOps is a collaborative function, often comprising data scientists, devops engineers, and IT.”  
[Databricks](#)
- MLflow is a tool which supports an MLOps approach



# MLflow

- MLflow is an open-source platform to manage the ML lifecycle, including experimentation, reproducibility, deployment, and a central model registry



## mlflow Tracking

Record and query experiments:  
code, data, config,  
and results

## mlflow Projects

Package data science  
code in a format to  
reproduce runs on  
any platform

## mlflow Models

Deploy machine  
learning models in  
diverse serving  
environments

## mlflow Registry

Store, annotate,  
discover, and  
manage models in  
a central repository

# MLflow

- MLflow has integrations with many of the most widely used machine learning libraries - so it's relevant if you are using Scikit Learn, Tensorflow, etc
- We will be introducing MLflow in today's workshop, and using it to track our training runs



# MLflow – Experiment Tracking

enchanted-hare-886

⋮ [Reproduce Run](#) [Register model](#)

[Overview](#) [Model metrics](#) [System metrics](#) [Evaluation results](#) [Preview](#) [Artifacts](#)

Description [✎](#)

No description

Details

Created at	Feb 25, 2025, 07:48 PM
Created by	stuartnathan2015@gmail.com
Experiment ID	2732982277298854 <a href="#">🔗</a>
Status	✅ Finished
Run ID	ec8c2d499ead45a992819975d561e557 <a href="#">🔗</a>
Duration	1.2min
Datasets used	—
Tags	<code>estimator_class: pyspark.ml.pipeline.Pipeline</code> <code>estimator_name: Pipeline</code> <code>spar... : path=dbfs:/FileStore/tables/Occupancy_D...</code> <a href="#">✎</a>
Source	<a href="#">📄</a> <a href="#">Occupancy Detection V2</a>
Logged models	<a href="#">🔗</a> spark
Registered models	—

Parameters (23)

<input type="text" value="Search parameters"/>	
Parameter	Value
DecisionTreeClassifier.cacheNodeIds	False
DecisionTreeClassifier.checkpointInterval	10

Metrics (1)

<input type="text" value="Search metrics"/>	
Metric	Latest
<a href="#">accuracy_testDF</a>	0.8639455782312925

# MLflow – Experiment Tracking

Overview

Model metrics

System metrics

Evaluation results

Preview

Artifacts

▼ model

▶ sparkml

MLmodel

conda.yaml

python\_env.yaml

requirements.txt

estimator\_info.json

metric\_info.json

model

Path: dbfs:/databricks/mlflow-tracking/2732982277298854/ec8c2d499ead45a992819975d561e557/artifacts/model

The code snippets below demonstrate how to make predictions using the logged model. You can also [register it to the model registry](#) to version control and deploy as a REST endpoint for *real time serving*.

Model schema

Input and output schema for your model. [Learn more](#)

Name	Type
Inputs (4)	
Temperature (required)	double
HumidityRatio (required)	double
CO2 (required)	double
Humidity (required)	double
Outputs (0)	

No schema. See [MLflow docs](#) for how to include input and output schema with your model.

Validate the model before deployment

Run the following code to validate model inference works on the example input data and logged model dependencies, prior to deploying it to a serving endpoint

```
import mlflow

model_uri = 'runs:/ec8c2d499ead45a992819975d561e557/model'

# Replace INPUT_EXAMPLE with your own input example to the model
# A valid input example is a data instance suitable for pyfunc prediction
input_data = INPUT_EXAMPLE

# Verify the model with the provided input data using the logged dependencies.
# For more details, refer to:
# https://mlflow.org/docs/latest/models.html#validate-models-before-deployment
mlflow.models.predict(
    model_uri=model_uri,
    input_data=input_data,
    env_manager="uv",
)
```

Make Predictions

Predict on a Pandas DataFrame:

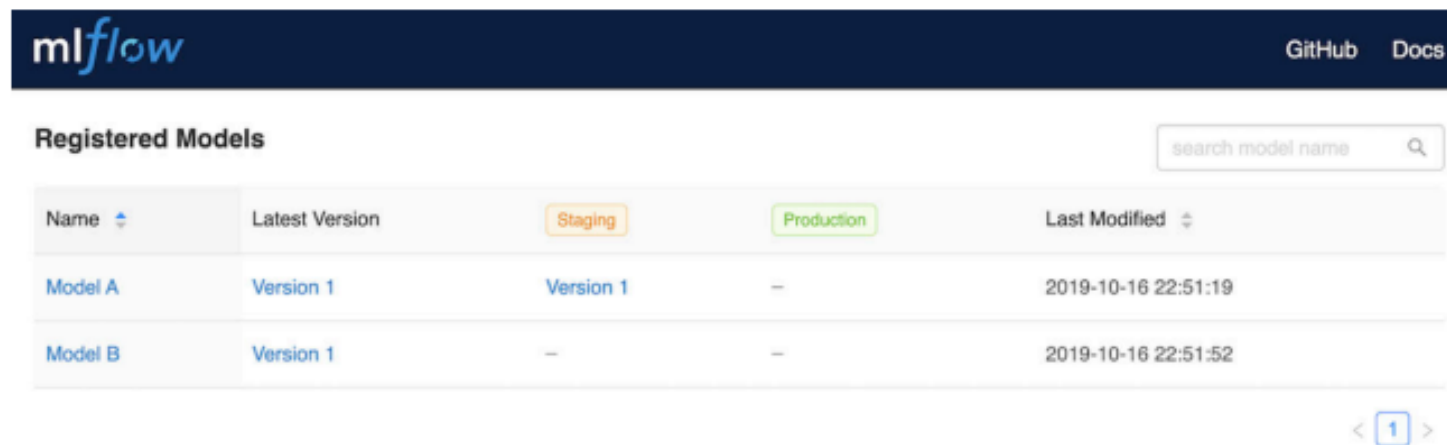
```
import mlflow
logged_model = 'runs:/ec8c2d499ead45a992819975d561e557/model'

# Load model as a PyFuncModel.
loaded_model = mlflow.pyfunc.load_model(logged_model)

# Predict on a Pandas DataFrame.
import pandas as pd
loaded_model.predict(pd.DataFrame(data))
```

# MLflow – Model Registry

- You can add a model to the MLflow Model Registry, a central repository for managing ML models.
- It tracks model versions and supports deployment.
- Spark ML models can be loaded from the registry for inference, supporting both batch and streaming workflows.
- Spark ML models cannot be directly deployed as a REST API but can be exported as a PyFunc model and deployed using MLflow Serving.



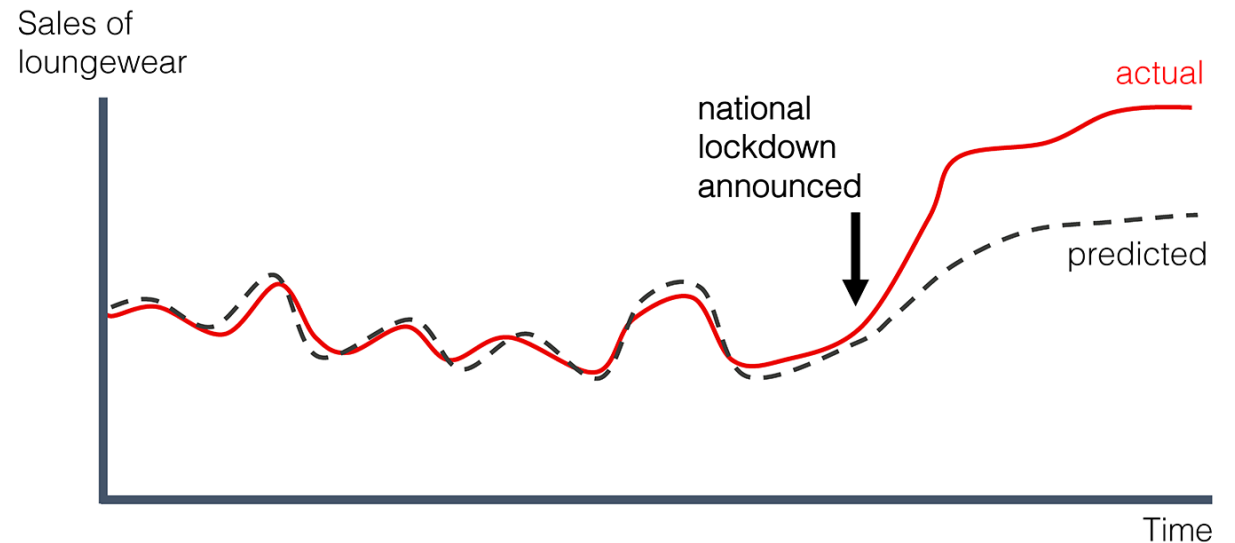
The screenshot shows the MLflow Model Registry web interface. At the top is a dark blue header with the 'mlflow' logo on the left and 'GitHub' and 'Docs' links on the right. Below the header, the title 'Registered Models' is followed by a search bar labeled 'search model name'. The main content is a table with the following columns: 'Name', 'Latest Version', 'Staging', 'Production', and 'Last Modified'. There are two rows of data: 'Model A' and 'Model B'. 'Model A' has 'Version 1' in the Latest Version column, 'Version 1' in the Staging column, and a dash in the Production column. 'Model B' has 'Version 1' in the Latest Version column, a dash in the Staging column, and a dash in the Production column. The last modified times are '2019-10-16 22:51:19' for Model A and '2019-10-16 22:51:52' for Model B. At the bottom right of the table, there is a pagination control showing '< 1 >'.

Name	Latest Version	Staging	Production	Last Modified
Model A	Version 1	Version 1	–	2019-10-16 22:51:19
Model B	Version 1	–	–	2019-10-16 22:51:52

# ML Workflow Challenge

Once a model is deployed, the challenges don't end there:

- You also need to plan for ongoing model monitoring and maintenance (e.g. Concept Drift is where the relationship between features and labels change over time, meaning model performance will degrade)



<https://www.evidentlyai.com/blog/machine-learning-monitoring-data-and-concept-drift>



# Deployment & Drift Detection

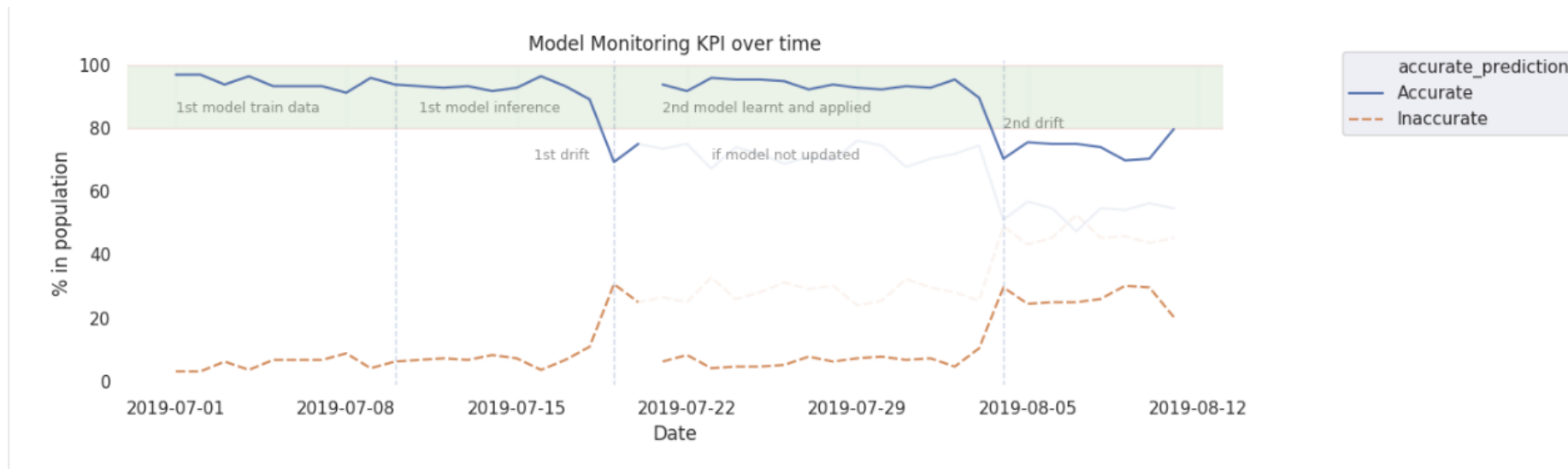
- A common way to detect model drift is to monitor the quality of predictions.
- An ideal ML model training exercise would start with loading data from sources, followed by feature engineering, model tuning and selection using Databricks, while having all experiment runs and produced models tracked in MLflow.
- In the deployment phase, models are loaded from MLflow Model Registry at runtime to make predictions. You can log model performance metrics as well as predictions back to storage (e.g., Delta Lake) for use in downstream systems and performance monitoring.

<https://www.databricks.com/blog/2019/09/18/productionizing-machine-learning-from-deployment-to-drift-detection.html>



# ML Workflow Challenge

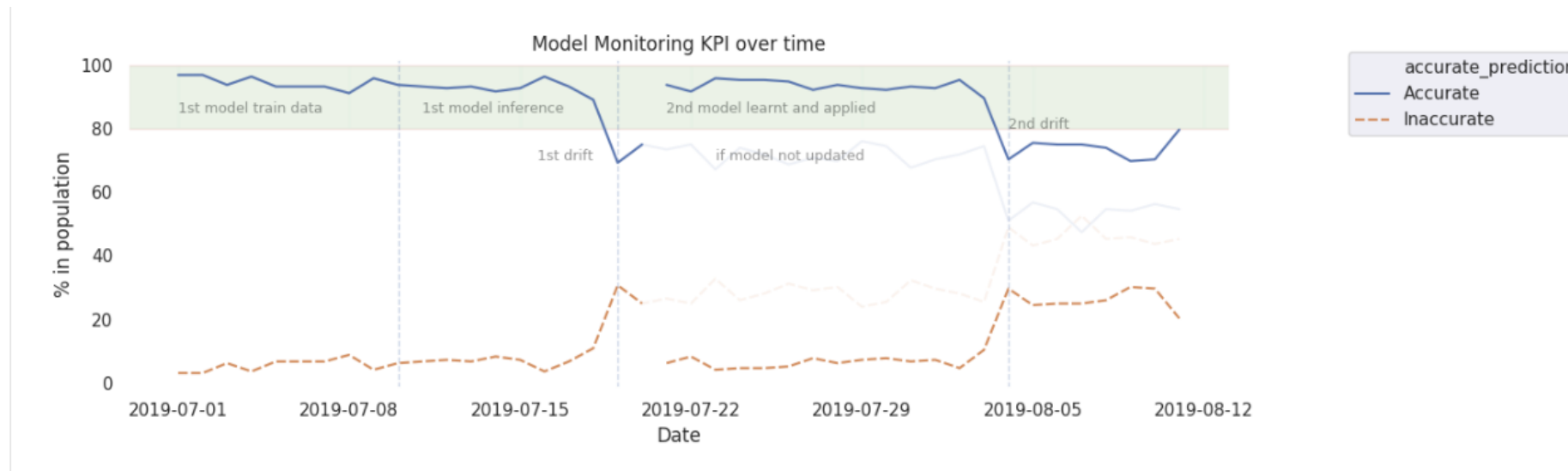
- By having training data, performance metrics, and predictions logged in one place you can ensure accurate monitoring.
- During supervised training, you use features and labels from your training data to evaluate the quality of the model. Once a model is deployed, you can log and monitor two types of data: model performance metrics and model quality metrics.



<https://www.databricks.com/blog/2019/09/18/productionizing-machine-learning-from-deployment-to-drift-detection.html>

# ML Workflow Challenge

- Model quality metrics depend on the actual labels. Once the labels are logged, you can compare predicted and actual labels to compute quality metrics and detect drift in the predictive quality of the model.



<https://www.databricks.com/blog/2019/09/18/productionizing-machine-learning-from-deployment-to-drift-detection.html>

# Learning Outcomes

By the end of this lecture and workshop, you will be able to:

1. Discuss how supervised machine learning can be used to solve problems in real-world scenarios
2. Apply classification algorithms to a dataset using the MLlib library, interpret the result and evaluate model performance
3. Explain how tools such as MLflow can support the machine learning lifecycle