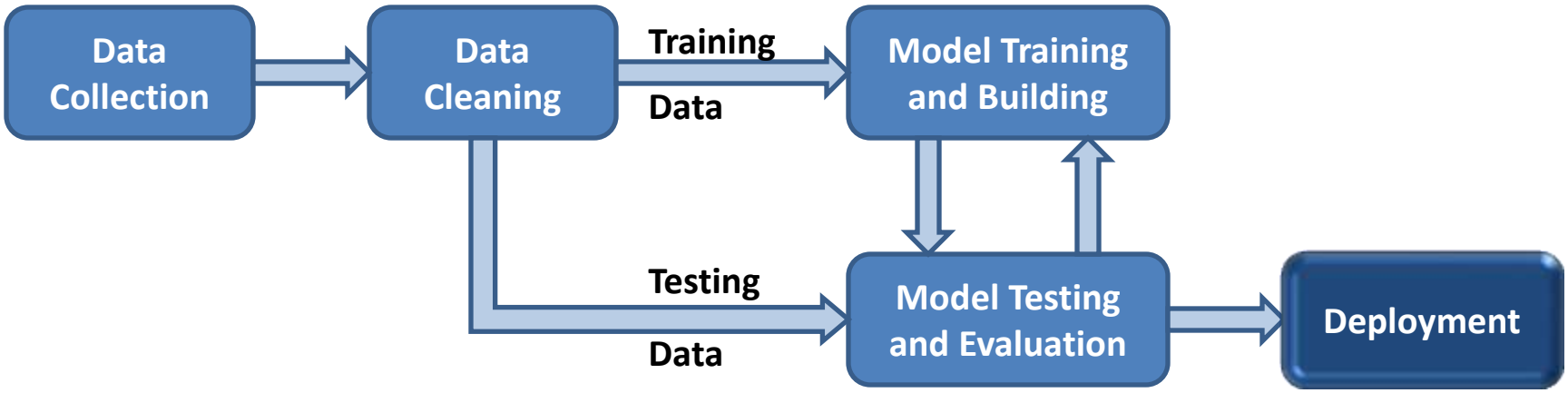
with

- Spark MLlib (Machine Learning library) is being developed with the goal of making machine learning easy and scalable.
- It makes available several utilities and common machine learning algorithms that include regression, classification, clustering, collaborative filtering.
- MLlib includes tools for operations such as:
  - Featurization: feature extraction, selection and transformation
  - Pipelines: APIs for constructing, evaluating, and tuning ML Pipelines
  - Persistence: saving & loading algorithms and models
- It is divided into two packages:
  - `spark.mllib` contains the original API built on top of RDDs.
  - `spark.ml` provides higher-level API built on top of DataFrames for constructing ML pipelines.
- As of Spark 2.0, the RDD-based APIs i.e. those in the `spark.mllib` package have entered maintenance mode.
- Going forward the primary Machine Learning API library for Spark is the DataFrame-based `spark.ml` package.

- To train and build the model, MLlib APIs of most of the supervised learning models, require historical data to be formatted into a dataframe with 2 columns – Features and Label.
- Though this is an additional task it is necessary and worthwhile because Spark MLlib can handle large scale dataframes which are distributed across the cluster nodes and processed & analysed in parallel.
- Also MLlib provides a number of APIs for all the operations necessary in the process of machine learning model building and deployment.
- Let us take a look at the steps of the process.



- **Data collection:** In a project if the data comes from different sources or different tables from the same source database then the data needs to be gathered and collated. If this needs sizeable effort then it can be done as a separate exercise.

- **Data cleaning:** Most often before formatting the data into the dataframe as required by the MLlib APIs we need to make sure that the data is clean. This involves the right data types are used for the columns. Converting some of categorical variables say income group from string type of *''low, 'medium'* and *'high'* to numeric type like 0, 1 and 2 and similar tasks. Spatk MLlib provides several APIs as part of Pipelines, Extracting, Transforming features for these tasks.

- Once the data is cleaned and transformed into dataframe it is split into two parts as training data and test data usually in 70:30 or 80:20 ratio.

- **Model Training and Building:** The training data is used for the learning part i.e. model training or model building. Here as well MLlib makes these operations easy by providing components like Estimator which work on large scale dataframes in distributed fashion under the hood. Any Spark MLlib machine learning API abstracts the concept of the learning algorithm that fits or trains on data and generates the Model. Code-wise, the Estimator, which is a machine learning object like decision tree classifier or linear regression implements the method `fit()`, which accepts the DataFrame and produces the Model.

- **Model Testing and Evaluation:** MLlib provides an abstraction Transformer which includes not only feature transformers that formats dataframes as per requirements but also the models. The model created in the previous step is a transformer which takes an input dataframe, reads the feature vector column, applies the model to perform prediction for each feature vector and outputs a new transformed dataframe which contains the additional column of predictions.
- For evaluation all standard evaluation metrics are available in SparkMLlib for the respective machine learning algorithms used.

Spark MLlib

Algorithms

- Regression
- Classification
- Clustering
- Collaborative Filtering

Featurization

- Feature Extraction
- Feature Selection
- Transformation
- Dimensionality Reduction

Utilities

- Linear Algebra
- Statistics
- Data Handling

Pipeline

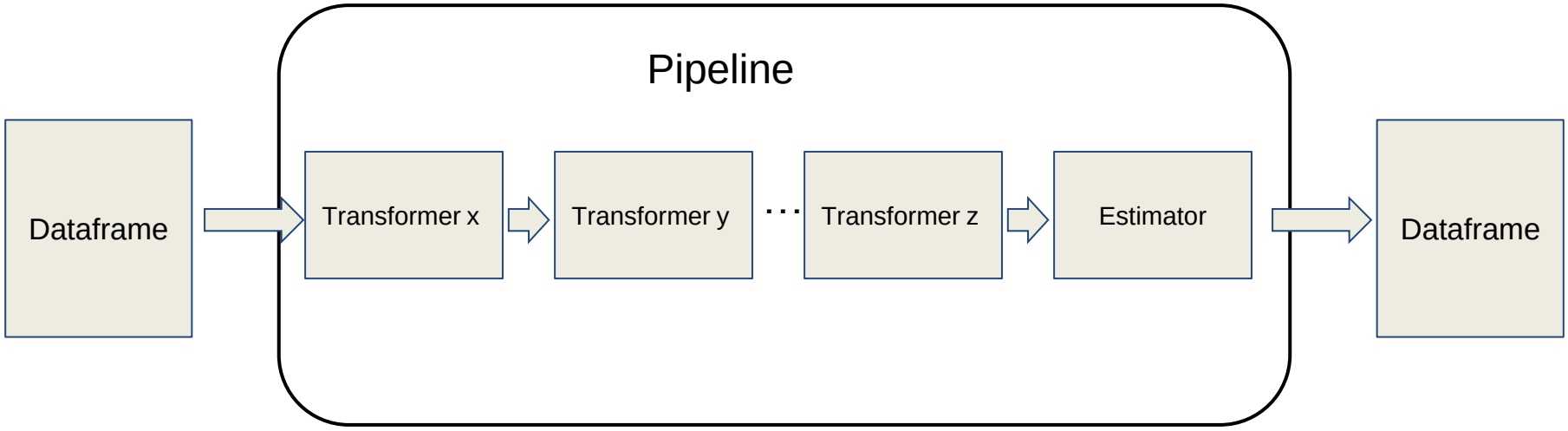
- Pipeline Construction
- Model Tuning
- Model Persistence

Spark ML Pipeline

- In machine learning, there are a lot of transformation steps that are performed to pre-process the data
- We repeat the same steps while making prediction
- You may often get confused about this transformations while working on huge projects
- To avoid this, pipelines were introduced that hold every step that is performed to fit the data on a model

Spark ML Pipeline

- The Pipeline API in Spark chains multiple Transformers and Estimator specifying a ML workflow
- It is a high-level API for MLlib that lives under the spark.ml package



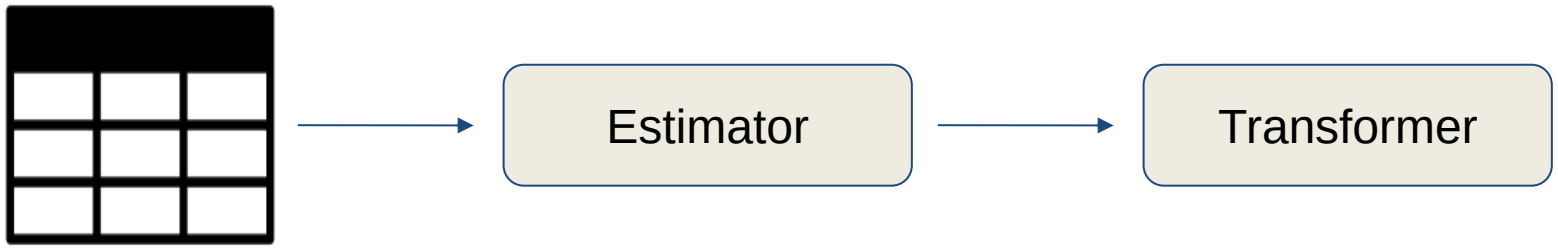
Transformers

- A ***Transformer*** takes a dataset as input and produces an augmented dataset as output
- It basically *transforms* one DataFrame into another DataFrame

Estimators

- An ***Estimator*** fit on the input data that produces a model

- For eg., logistic regression is an Estimator that trains on a dataset with labels and features and produces a logistic regression model



- The model acts as a Transformer that transforms the input dataset

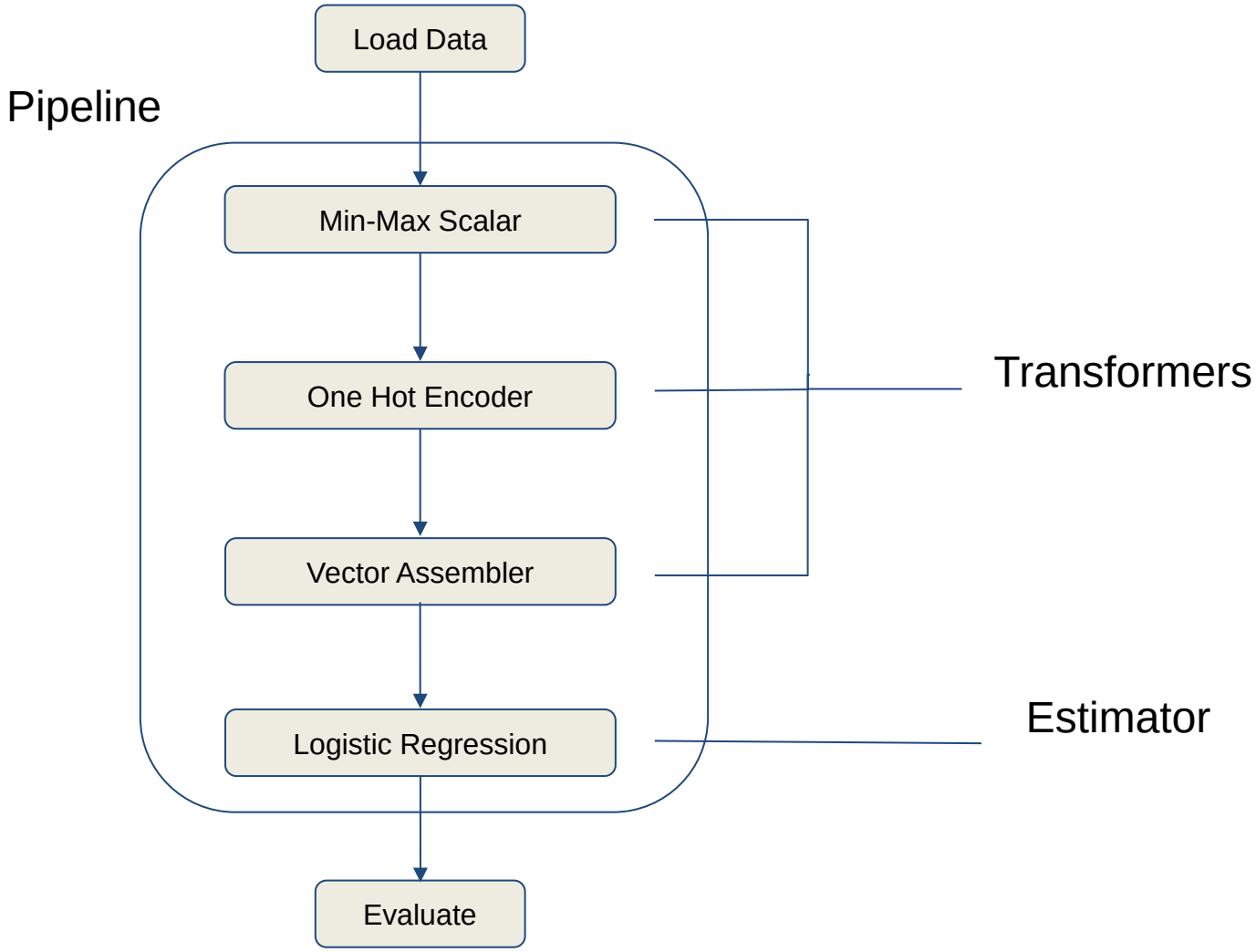
  For eg., logistic regression model can later be used to make predictions which technically adds prediction columns (Transformation) in the dataset

Spark ML Component Flow

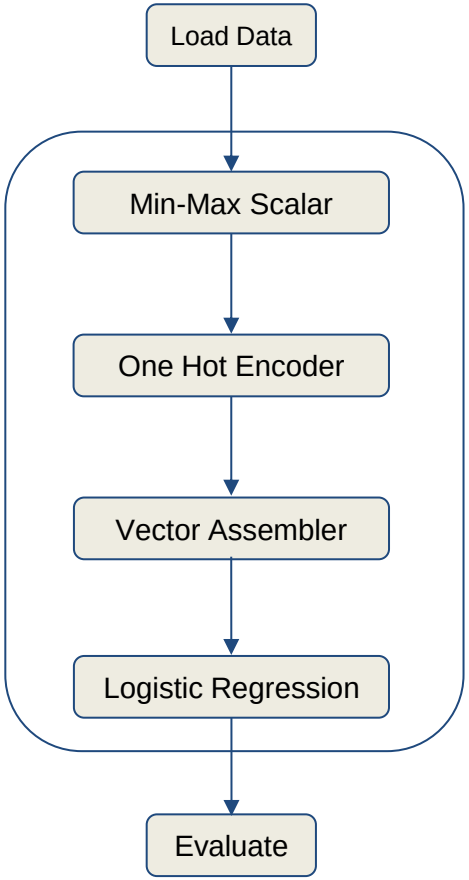
Spark ML Component Flow

- **Pipeline API** chains Transformers and Estimator each as a stage to specifying ML workflow
- These stages are run in order
- The input DataFrame is transformed as it passes through each stage
- **Evaluator** then evaluates the model performance

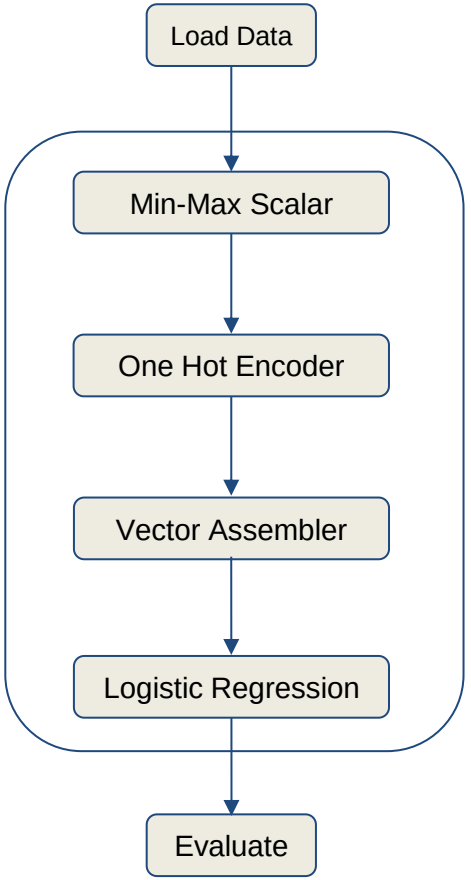
Spark ML Component Flow



Spark ML Component Flow



Pipeline of machine learning components



Reusing the pipeline on new data

Spark ML Component Flow

- Spark ML algorithms (estimators) expects all features to be contained within a single column in the form of Vector

- It is one of the important spark ml data types that you need to understand before we take a look at different feature transformers

Please Note

Dense and Sparse Vectors are vector representation of data

Spark ML Data Types - Local Vector

- Dense Vectors:
  - By definition, dense means closely compacted
  - Dense Vector is a vector representation that contains many values or values that are not zeros (very few zero values)
  - It is basically an array of values
  - For eg. A vector (3.0, 5.0, 8.0, 0.0) can be represented in dense format as [3.0, 5.0, 8.0, 0.0]

Spark ML Data Types - Local Vector

- Dense Vectors PySpark

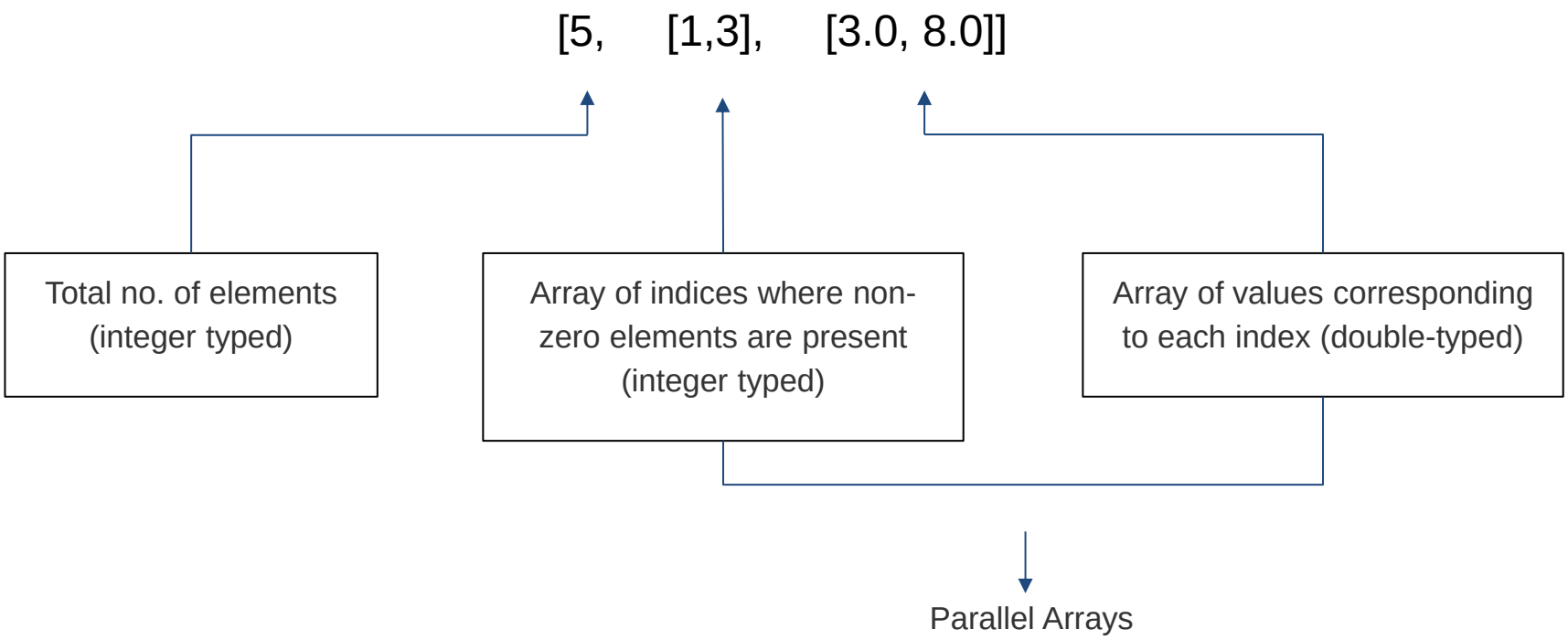
```
1  from pyspark.ml.linalg import Vectors
2  # Create a dense vector representation for (3.0, 5.0, 8.0, 0.0)
3  denseVector = Vectors.dense(3.0, 5.0, 8.0, 0.0)
4  denseVector
```

```
Out[1]: DenseVector([3.0, 5.0, 8.0, 0.0])
```

Spark ML Data Types - Local Vector

- Sparse Vectors:
  - By definition, sparse means thinly dispersed or scattered
  - If a vector has a majority of its elements as zero, it can be represented as sparse vector
  - It stores the size of the vector, an array of indices, and an array of values corresponding to those indices
  - For ex. A vector (0.0, 3.0, 0.0, 8.0, 0.0) can be represented in sparse format as [5, [1,3], [3.0, 8.0]]

Spark ML Data Types - Local Vector



- A sparse vector is used for storing non-zero entries for saving space

Spark ML Data Types - Local Vector

- Sparse Vector PySpark

```
1  from pyspark.ml.linalg import Vectors
2  # Create a sparse vector representation of (0.0, 3.0, 0.0, 8.0, 0.0)
3  sparseVector = Vectors.sparse(5, [1,3], [3.0, 8.0])
4  sparseVector
```

```
Out[2]: SparseVector(5, {1: 3.0, 3: 8.0})
```

Spark ML Transformers

ML Feature Transformers

- Feature building is a super important step for model building
- Some of the common feature transformer that we use for model building are:
  - Binarizer
  - Bucketizer
  - StringIndexer
  - IndexToString
  - OneHotEncoder
  - VectorAssembler
  - VectorIndexer
  - StandardScaler
  - MinMaxScaler
- Most Transforms are under *org.apache.spark.ml.feature* package

Model Persistence

- In real-life scenarios, you will be producing ML model and hands it over to the development team for deploying in a production environment
- This becomes easier with model persistence
- Model persistence means saving your model to a disk for later use without the need to retrain your model

Model Persistence

- We use model.save(‘path’) to save our model at the desired location
- It might happen that you wish to retrain your model and save it at the same the place
- In those cases, use model.write().overwrite().save(‘path’) to save your retrained model at the same place

Model Persistence

- You can then load the model and perform predictions
- Use PipelineModel module from pyspark.ml package to load the persisted pipeline model
- The loaded model can then be used for perform prediction on test data