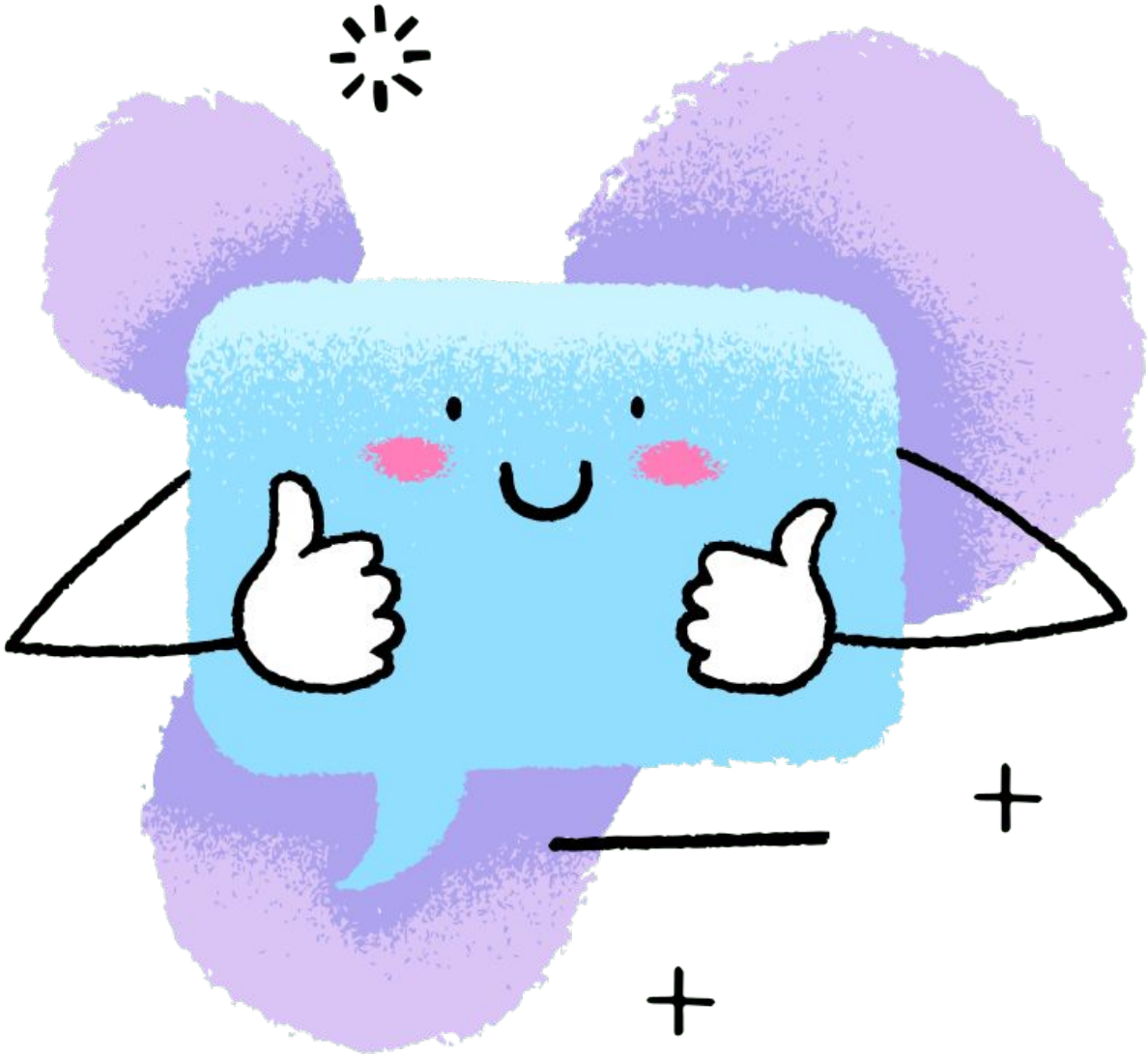
PySpark



GeekBrains

API. Пользовательские функции

Apache Spark

Что будет на уроке

1. Дополнение по API
2. Составные типы данных: Array, Map
3. User Defined Functions. Использование UDF в spark.sql
4. ML pipelines

Union

Нужно следить за порядком колонок

union(*other*) [source]

Return a new **DataFrame** containing union of rows in this and another frame.

This is equivalent to *UNION ALL* in SQL. To do a SQL-style set union (that does deduplication of elements), use this function followed by a distinct.

Also as standard in SQL, this function resolves columns by position (not by name).

New in version 2.0.

unionAll(*other*) [source]

Return a new **DataFrame** containing union of rows in this and another frame.

This is equivalent to *UNION ALL* in SQL. To do a SQL-style set union (that does deduplication of elements), use this function followed by a distinct.

Also as standard in SQL, this function resolves columns by position (not by name).

Note: Deprecated in 2.0, use union instead.

explode & flatten (spark 2.4)

Массив значений в строки

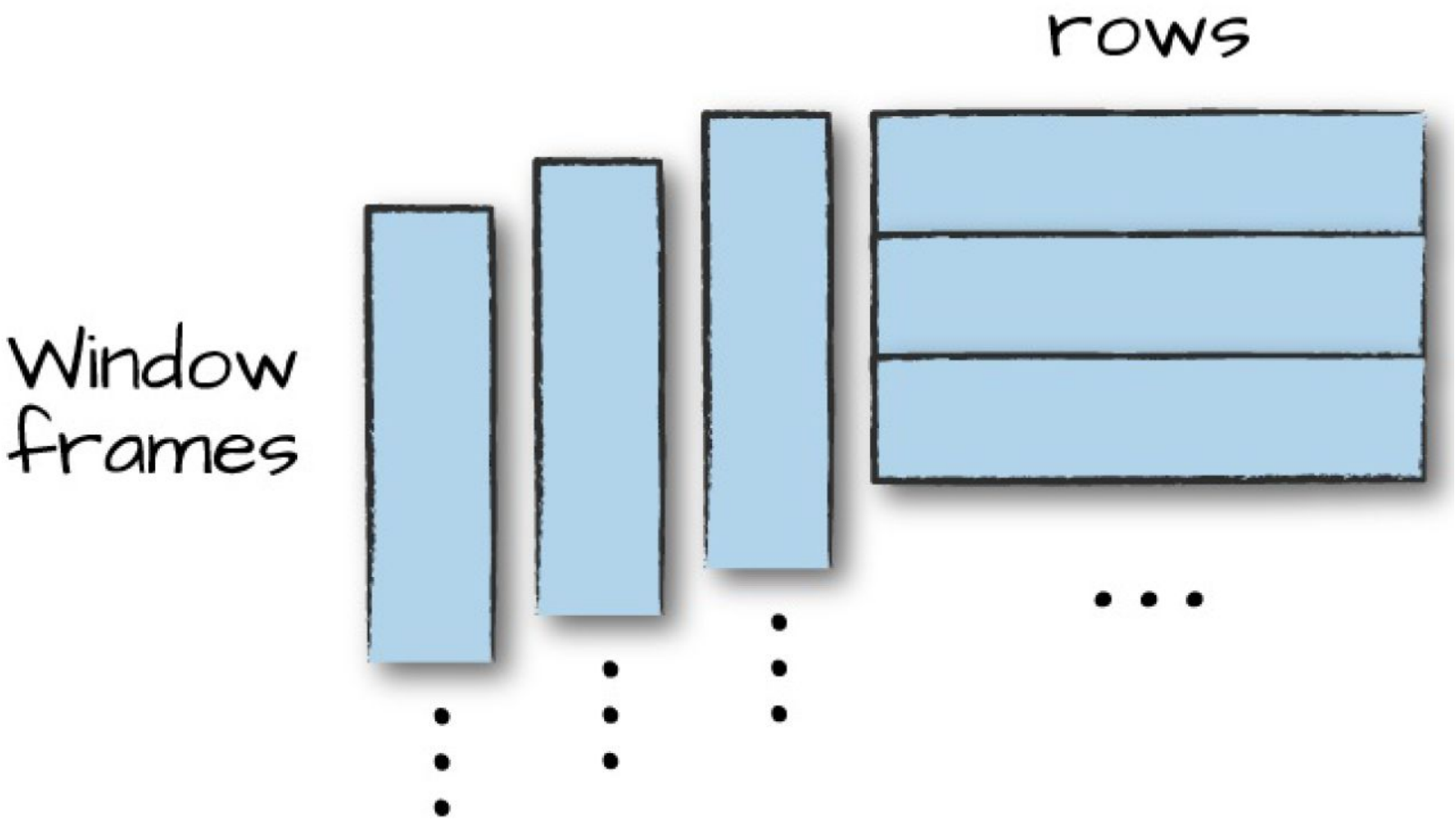
```
arrayArrayData = [
  ("James",[["Java","Scala","C++"],["Spark","Java"]]),
  ("Michael",[["Spark","Java","C++"],["Spark","Java"]]),
  ("Robert",[["CSharp","VB"],["Spark","Python"]])
]

df = spark.createDataFrame(data=arrayArrayData, schema = ['name','subjects'])

from pyspark.sql.functions import explode
df.select(df.name,explode(df.subjects)).show(truncate=False)

from pyspark.sql.functions import flatten
df.select(df.name,flatten(df.subjects)).show(truncate=False)
```

Window functions



Window functions

WINDOW FUNCTIONS USAGE & SYNTAX	PYSPARK WINDOW FUNCTIONS DESCRIPTION
row_number(): Column	Returns a sequential number starting from 1 within a window partition
rank(): Column	Returns the rank of rows within a window partition, with gaps.
percent_rank(): Column	Returns the percentile rank of rows within a window partition.
dense_rank(): Column	Returns the rank of rows within a window partition without any gaps. Where as Rank() returns rank with gaps.
ntile(n: Int): Column	Returns the ntile id in a window partition
cume_dist(): Column	Returns the cumulative distribution of values within a window partition
lag(e: Column, offset: Int): Column lag(columnName: String, offset: Int): Column lag(columnName: String, offset: Int, defaultValue: Any): Column	returns the value that is `offset` rows before the current row, and `null` if there is less than `offset` rows before the current row.
lead(columnName: String, offset: Int): Column lead(columnName: String, offset: Int): Column lead(columnName: String, offset: Int, defaultValue: Any): Column	returns the value that is `offset` rows after the current row, and `null` if there is less than `offset` rows after the current row.

Booleans

Фильтрация по значению в колонке

```python
# in Python
from pyspark.sql.functions import col
df.where(col("InvoiceNo") != 536365)\
.select("InvoiceNo", "Description")\
.show(5, False)
```

```python
from pyspark.sql.functions import instr

priceFilter = col("UnitPrice") > 600
descripFilter = instr(df.Description, "POSTAGE") >= 1

df.where(df.StockCode.isin("DOT"))\
.where(priceFilter | descripFilter).show()
```

Регулярные выражения

```
1 from pyspark.sql.functions import regexp_replace
2 regex_string = "BLACK|WHITE|RED|GREEN|BLUE"
3 df.select(
4     regexp_replace(col("Description"), regex_string, "COLOR")
5         .alias("color_clean"),
6     col("Description")
7 ).show(2)
```

Nulls

первое не-null значение из списка столбцов

```
1 from pyspark.sql.functions import coalesce
2
3 df.select(coalesce(col("Description"), col("CustomerId")))\
4     .show()
```

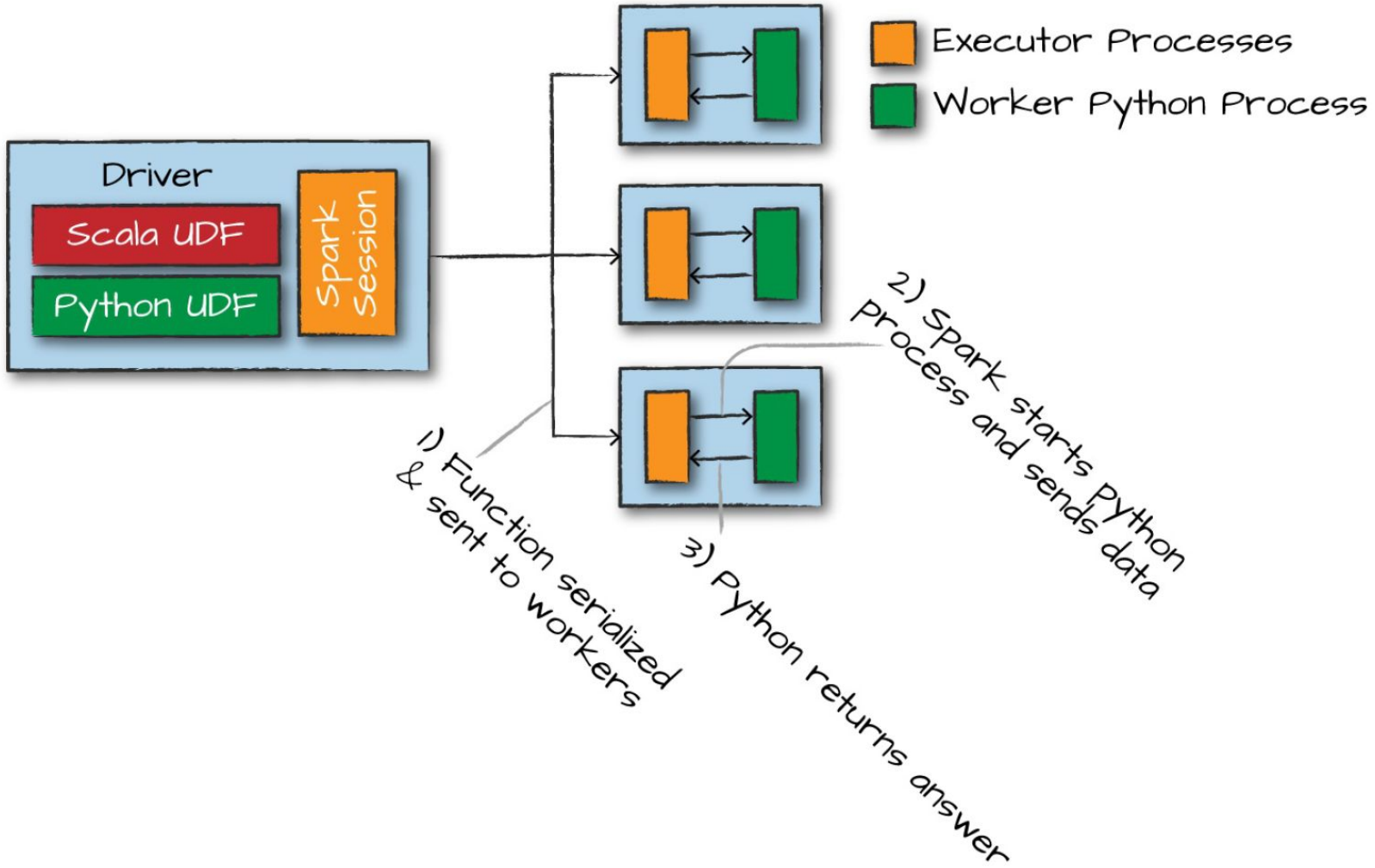
удаление строк, содержащих null

```
1 df.na.drop()
2 df.na.drop("any") # drop if ANY column is null
3
4 df.na.drop("all") # drop if ALL columns are null
5
6 df.na.drop("all", subset=["StockCode", "InvoiceNo"])
```

Nulls
заполнение пустых значений

```
1 df.na.fill("it was null value")
2
3 # specify values with dict
4 fill_cols_vals = {"StockCode": 5, "Description" : "No Value"}
5 df.na.fill(fill_cols_vals)
```

User-Defined Functions



https://docs.databricks.com/spark/latest/spark-sql/udf-python.html

Performance concerns with UDFs

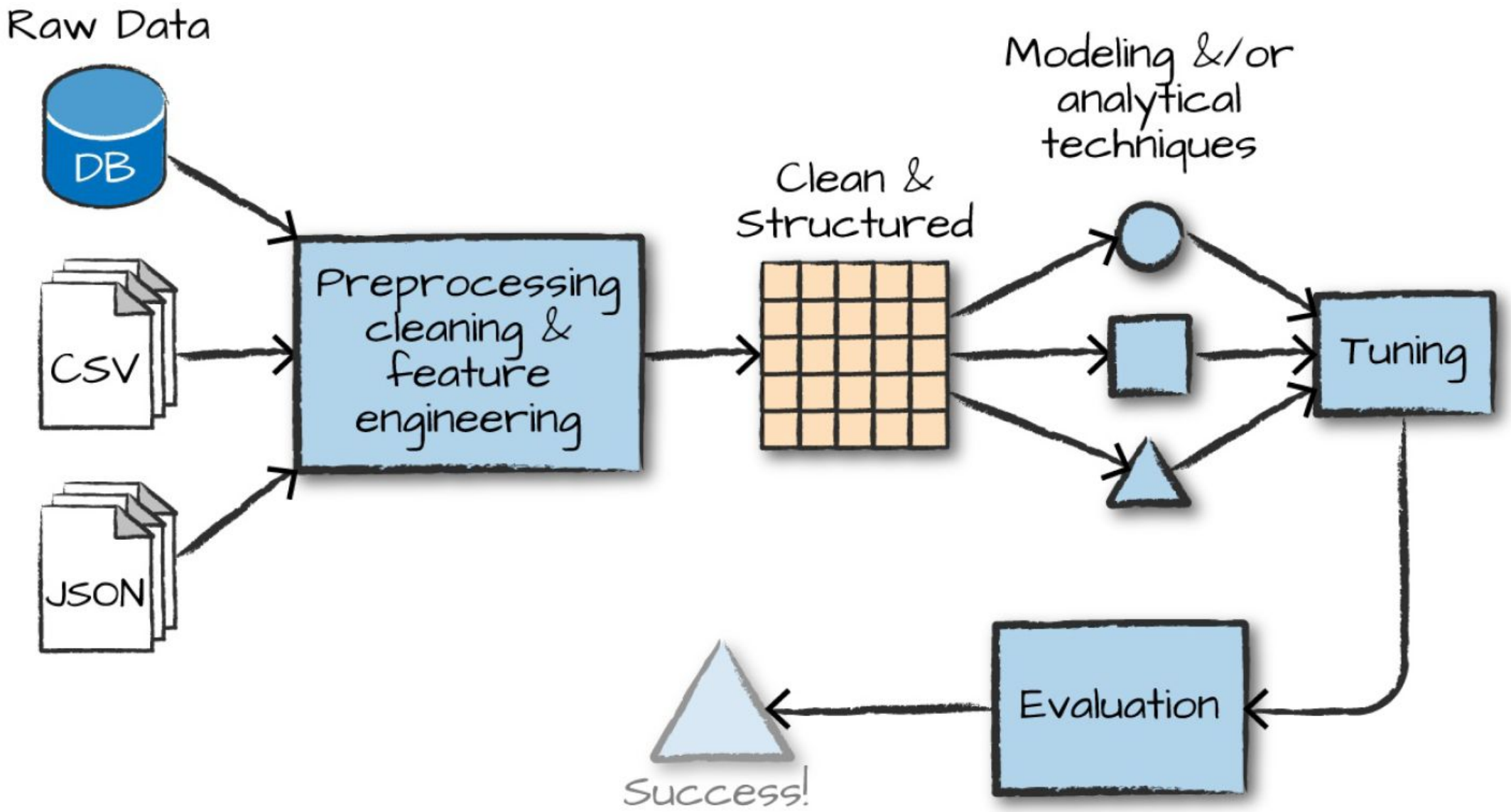
- UDFs are black-box to Spark optimizations.
- UDFs block many spark optimizations like
  - WholeStageCodegen
  - Null Optimizations
  - Predicate Pushdown
  - More optimizations from Catalyst Optimizer
- String Handling within UDFs
  - UTF-8 to UTF-16 conversion. Spark maintains string in UTF-8 encoding versus Java runtime encodes in UTF-16.
  - Any String input to UDF requires UTF-8 to UTF-16 conversion.
  - Conversely, a String output requires a UTF-16 to UTF-8 conversion.

Ссылки

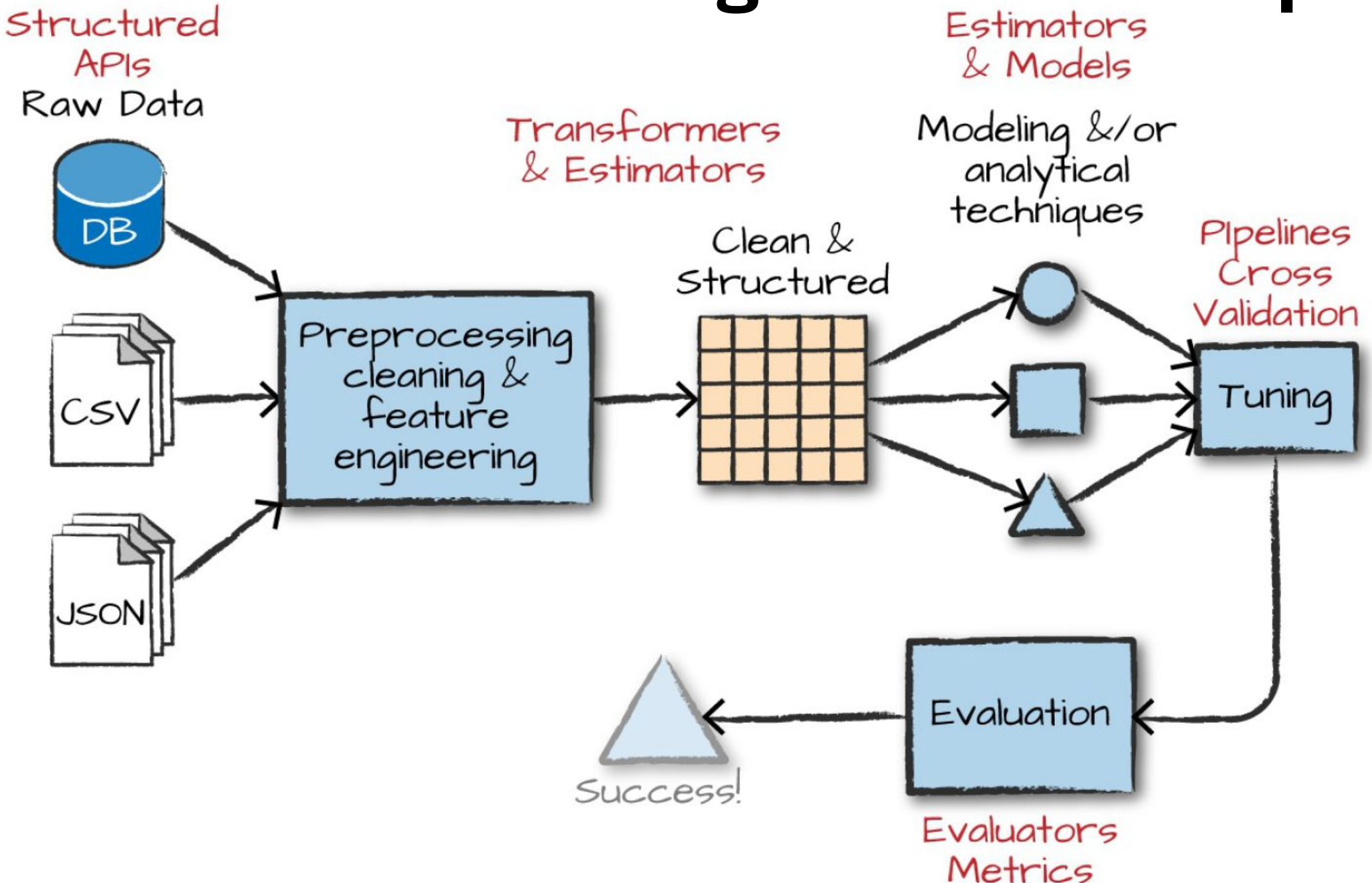
Introducing Window Functions in Spark SQL - The Databricks Blog
https://www.youtube.com/watch?v=2QNk-bcjN-I

Spark ML

The machine learning workflow



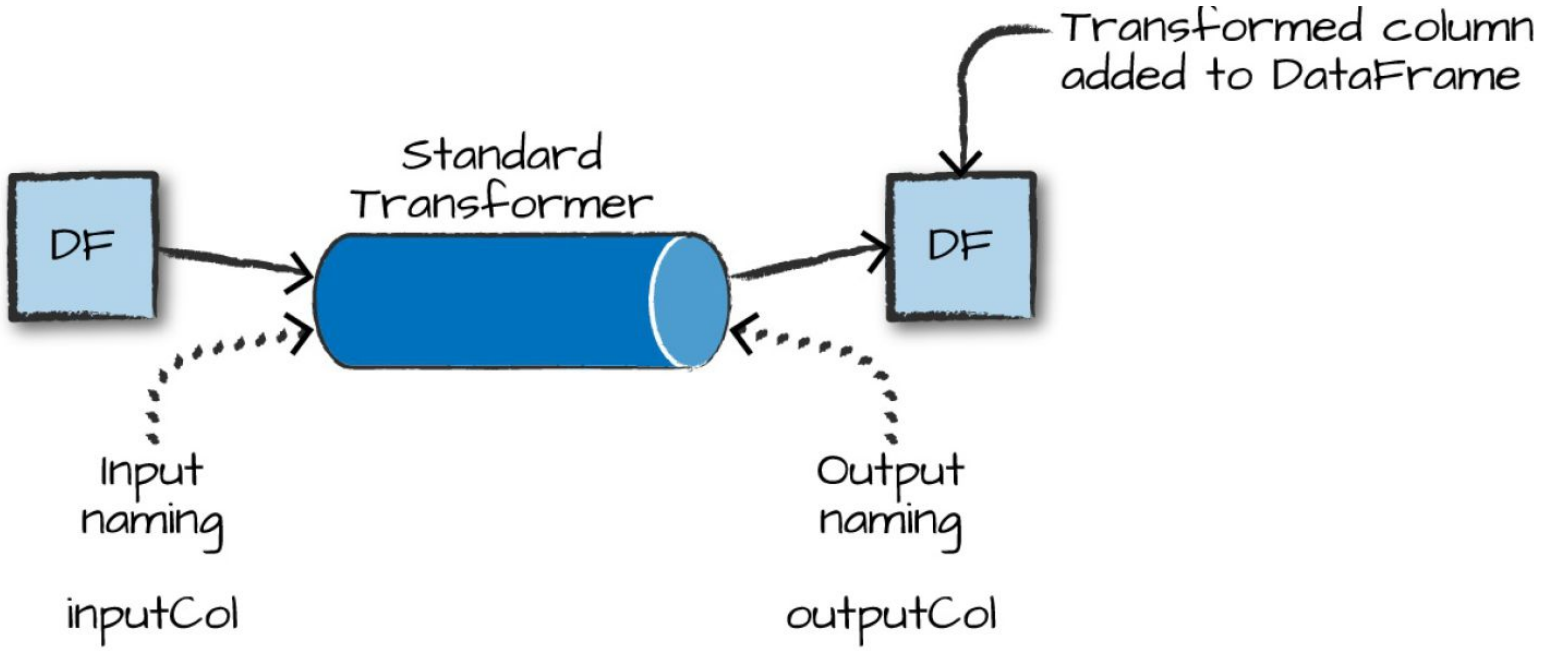
The machine learning workflow in Spark



Pipeline components
Transformer and Estimator

A **Transformer** can transform one DataFrame into another DataFrame.

An **Estimator** can be fit on a DataFrame to produce a Transformer



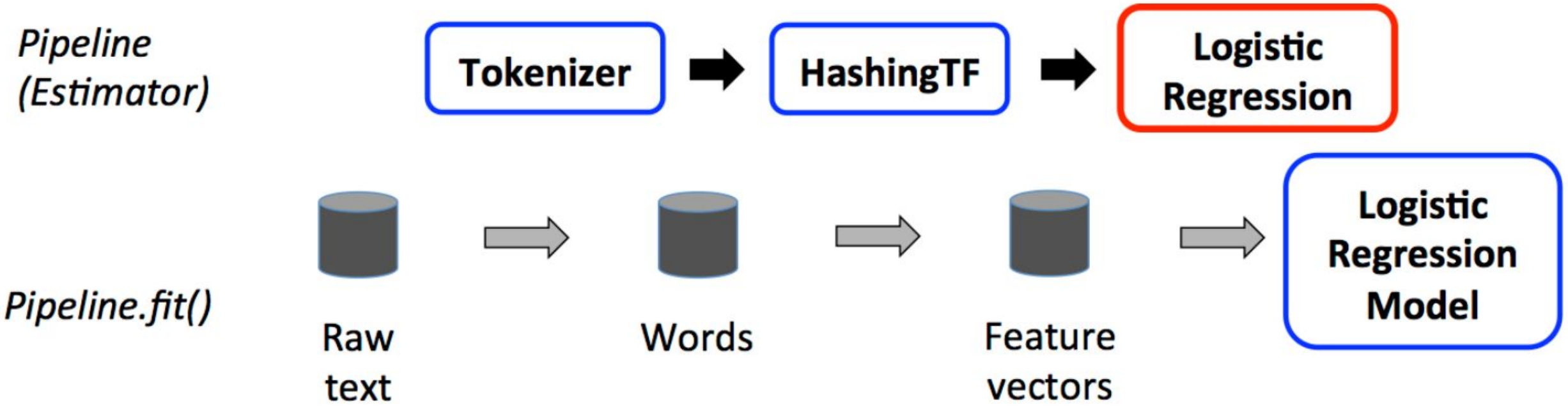
Low-level data types

Vector

```
1 from pyspark.ml.linalg import Vectors
2 denseVec = Vectors.dense(1.0, 2.0, 3.0)
3 size = 3
4 idx = [1, 2] # locations of non-zero elements in vector
5 values = [2.0, 3.0]
6 sparseVec = Vectors.sparse(size, idx, values)
```

A Pipeline

blue - transformations, red - estimator



Logistic Regression

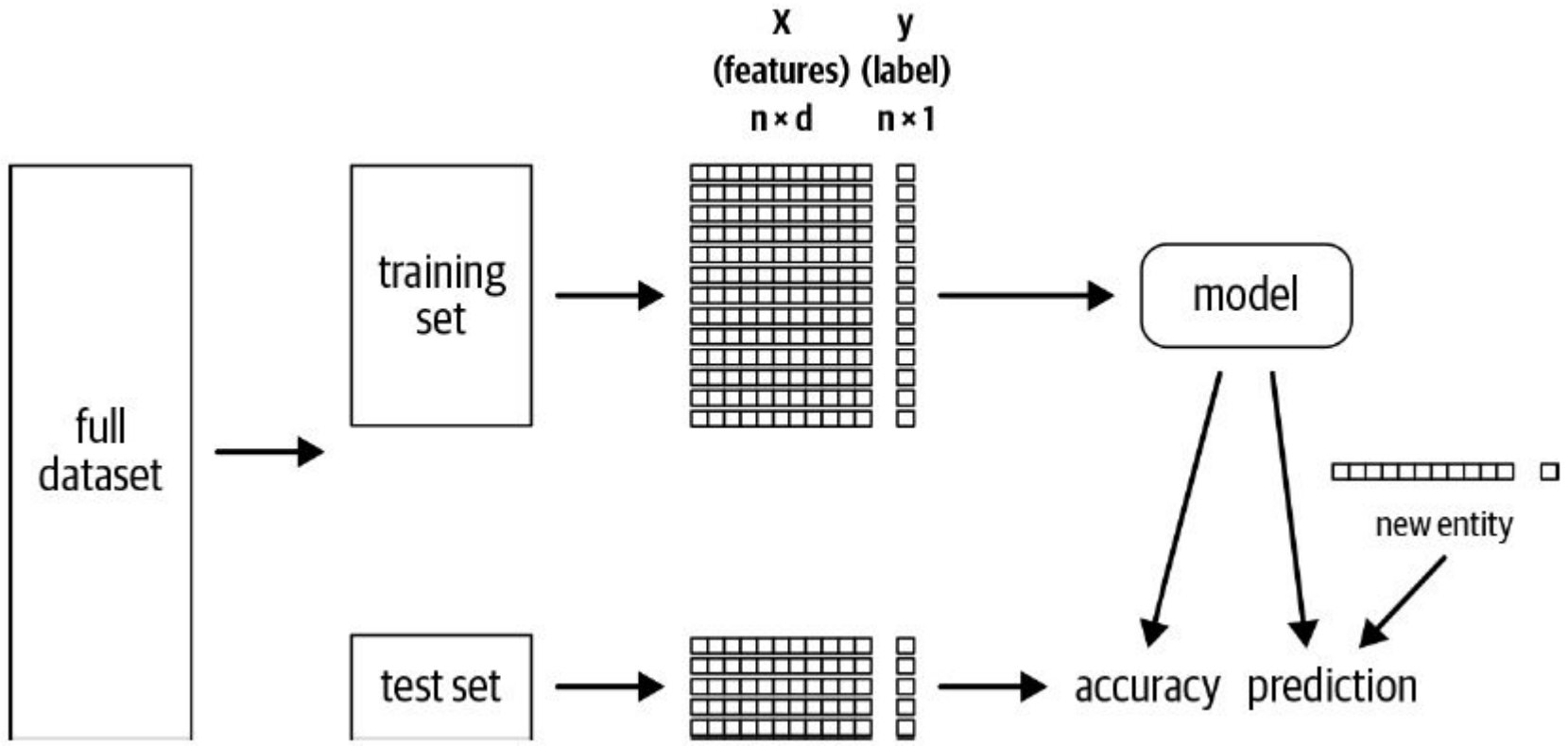
```
from pyspark.mllib.linalg import Vectors
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.param import Param, Params

# Prepare training data from a list of (label, features) tuples.
training = sqlContext.createDataFrame([
    (1.0, Vectors.dense([0.0, 1.1, 0.1])),
    (0.0, Vectors.dense([2.0, 1.0, -1.0])),
    (0.0, Vectors.dense([2.0, 1.3, 1.0])),
    (1.0, Vectors.dense([0.0, 1.2, -0.5]))], ["label", "features"])

# Create a LogisticRegression instance. This instance is an Estimator.
lr = LogisticRegression(maxIter=10, regParam=0.01)
# Print out the parameters, documentation, and any default values.
print "LogisticRegression parameters:\n" + lr.explainParams() + "\n"

# Learn a LogisticRegression model. This uses the parameters stored in
  lr.
model1 = lr.fit(training)
```

Creating Training and Test Data Sets



Ссылки

https://spark.apache.org/docs/latest/ml-pipeline.html

Спасибо!

Каждый день вы становитесь лучше :)