**Regular Expressions**

* **Definition:** Patterns used to match character combinations in strings.
* **Usage in PySpark:** Data cleaning and transformation.
* **Example in PySpark:**

from pyspark.sql.functions import regexp\_replace

df = df.withColumn('cleaned\_column', regexp\_replace('column\_name', r'\d+', ''))

* **Explanation:** This code removes all digits from 'column\_name'.

**Caching and Data Persistence**

* **Definition:** Storing intermediate results to speed up subsequent operations.
* **PySpark Caching:** Use .cache() or .persist() to keep data in memory.
* **Example:**

df = spark.read.csv('data.csv')

df.cache() # Data is now cached

df.show() # Accessing cached data

* **Explanation:** cache() keeps the DataFrame in memory, speeding up repeated access.

**Shared Variables**

* **Definition:** Variables used in multiple tasks across the cluster.
* **Types:**
  + **Broadcast Variables:** Efficiently share large read-only data across tasks.
  + **Accumulators:** Used to accumulate values (like counters).
* **Example of Broadcast Variable:**

broadcast\_var = spark.sparkContext.broadcast([1, 2, 3])

def func(x):

return x in broadcast\_var.value

df\_filtered = df.filter(df['column'].rdd.map(func))

* **Explanation:** broadcast\_var is shared across all nodes.

**Developing Self-contained PySpark Application, Packages, and Files**

* **Definition:** Creating modular and reusable PySpark applications.
* **Structure:**
  + **Main Application File:** The entry point of the PySpark job.
  + **Modules/Packages:** Separate code into modules for better organization.
  + **Configuration Files:** Manage configurations outside of code.
* **Example:**
  + **Directory Structure:**

my\_pyspark\_app/

├── main.py

├── utils.py

└── config.json

* + **main.py:**

from pyspark.sql import SparkSession

from utils import process\_data

spark = SparkSession.builder.appName('MyApp').getOrCreate()

df = spark.read.json('data.json')

processed\_df = process\_data(df)

processed\_df.show()

**Disadvantages of RDDs**

* **Overview:** Resilient Distributed Datasets (RDDs) are a fundamental abstraction in Spark but have limitations.
* **Disadvantages:**
  + **Lack of Optimization:** No optimizations like Catalyst optimizer and Tungsten execution engine used in DataFrames/Datasets.
  + **Verbosity:** Requires more boilerplate code compared to DataFrames.
  + **Performance:** DataFrames and Datasets often provide better performance due to optimizations.
* **Example Comparison:**
  + **RDD Operation:**

rdd = spark.sparkContext.textFile('data.txt')

words = rdd.flatMap(lambda line: line.split(' '))

word\_counts = words.map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b)

* + **DataFrame Operation:**

df = spark.read.text('data.txt')

words = df.selectExpr("explode(split(value, ' ')) as word")

word\_counts = words.groupBy('word').count()