**Day 19 Assignment - Vamsi Viswanadham**

**Date: 26/12/2023**

**Spark in Jupyter Lab**

To use Spark in Jupyter Lab, you typically start by installing PySpark, the Python API for Spark.

This can be done using pip:

**pip install pyspark**

**pip install jupyterlab**

Once installed, you can start a Jupyter Lab session and import PySpark to begin working with Spark. The key component of Spark in Python is the SparkContext, which acts as the entry point for Spark functionality.

**Resilient Distributed Datasets (RDD)**

RDD stands for Resilient Distributed Dataset (RDD). RDDs are immutable, distributed collections of objects, which can be created from a file in the Hadoop-supported file system or from an existing Python list. They are fault-tolerant and capable of parallel operations.

**Creating an RDD**

RDDs can be created in several ways, here's an example of creating one from a Python list:

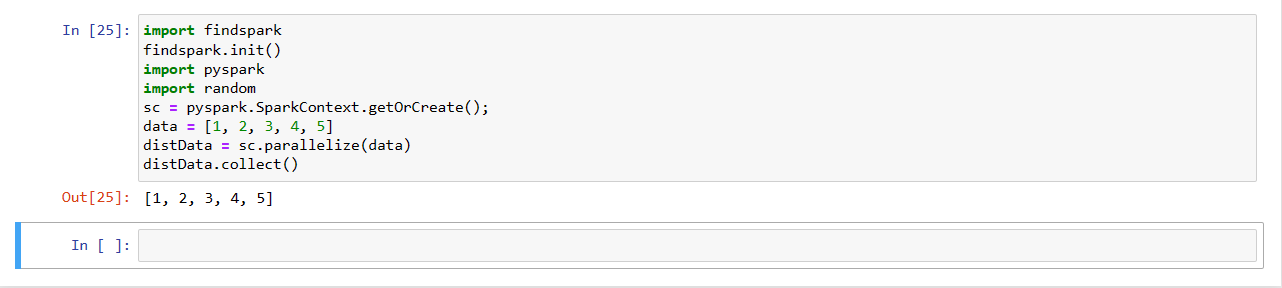
**from pyspark import SparkContext**

**sc = SparkContext()**

**data = [1, 2, 3, 4, 5]**

**distData = sc.parallelize(data)**

**Output:**

****

**Basic RDD Operations**

RDDs support two types of operations: transformations and actions.

1. Transformations create a new RDD from an existing one, such as map and filter.
2. Actions return a value after running a computation on the RDD, such as reduce and collect.

Examples:

**Map Transformation:**

rdd = sc.parallelize([1, 2, 3, 4, 5])

rdd = rdd.map(lambda x: x \* x)

Output:



**Filter Transformation:**

rdd = sc.parallelize([1, 2, 3, 4, 5])

filtered = rdd.filter(lambda x: x % 2 == 0)

Output:

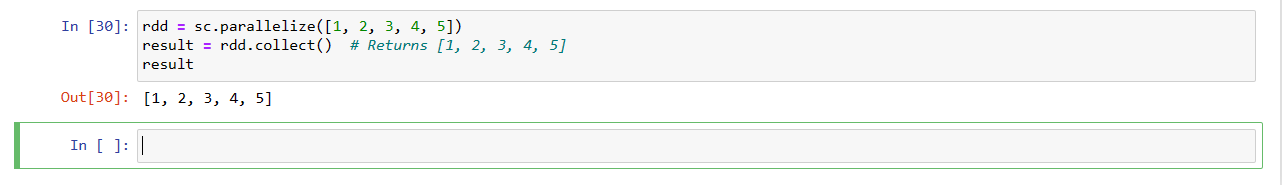


**Collect Action:**

rdd = sc.parallelize([1, 2, 3, 4, 5])

result = rdd.collect() # Returns [1, 2, 3, 4, 5]

Output:



**Reduce Action:**

rdd = sc.parallelize([1, 2, 3, 4, 5])

sum = rdd.reduce(lambda x, y: x + y) # Returns 15

Output:



**PySpark ClassRoom Example:**

import findspark

findspark.init()

import pyspark

import random

sc = pyspark.SparkContext(appName="Pi")

num\_samples = 100000000

def inside(p):

x, y = random.random(), random.random()

return x\*x + y\*y < 1

count = sc.parallelize(range(0, num\_samples)).filter(inside).count()

pi = 4 \* count / num\_samples

print(pi)

sc.stop()

Output:

