Experiment 7

Shashwat Shah
60004220126
TY BTECH COMPS B2

Aim: Implement RDD using Pyspark.

Theory: Apache spark is an open source distributed processing system used for big data workloads. It utilizes in-memory caching, and optimized query execution for fast analytic queries against data of any size.

Spark not only supports 'Map' and 'Reduce'. It provides developement API's in Java, Scala, Python and R. and supports code reuse across multiple workloads.

RDD is a core abstraction in spark which stands for resilent distributed dataset. It enables partition of large data into smaller data that fit's each machine. So that computational can be done parallely on multiple machines.

RDD supports two types of operations:

Transformations are operations (such as map, filter, join and so on) that are performed on an RDD and which yield a new RDD containing the result.

Actions are operations (such as reduce, count, first and so on). that return a value after running a computation on an RDD.

Conclusion: Thus we have implemented RDD using Pyspark.

Code & Output:

```
[ ]  from pyspark import SparkContext

     sc = SparkContext("local","RDD Example")

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

     squared_rdd = rdd.map(lambda x: x*x)
     filtered_rdd = squared_rdd.filter(lambda x: x > 10)

     result = filtered_rdd.collect()

     print(result)
```

```
[16, 25]
```

```python
from pyspark import SparkContext , SparkConf
import math
conf = SparkConf().setAppName("SquareRootNumbers").setMaster("local")
sc = SparkContext.getOrCreate(conf=conf)

numbers_rdd = sc.parallelize(range(1,21))

square_root_rdd = numbers_rdd.map(lambda x: math.sqrt(x))
square_roots= square_root_rdd.collect()
for square_root in square_roots:
  print(square_root)

sc.stop()
```

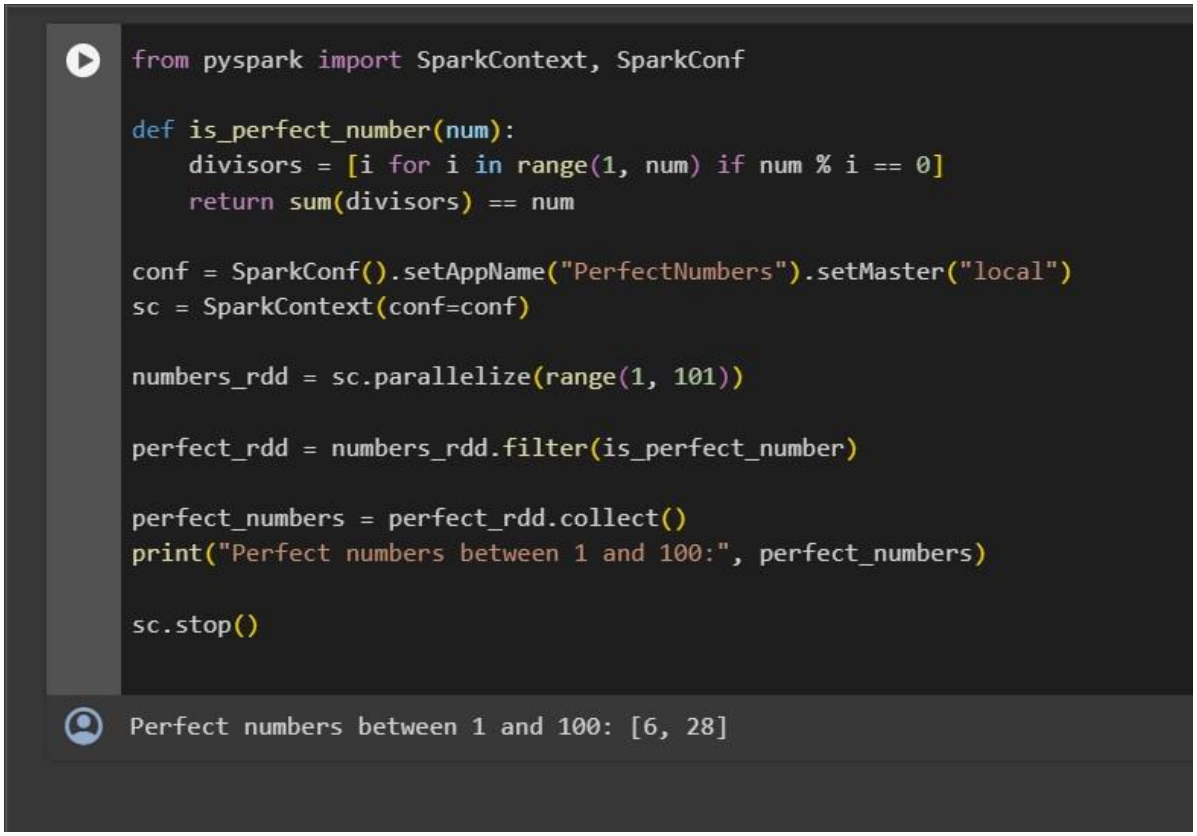
```
1.0
1.4142135623730951
1.7320508075688772
2.0
2.23606797749979
2.449489742783178
2.6457513110645907
2.8284271247461903
3.0
3.1622776601683795
3.3166247903554
3.4641016151377544
3.605551275463989
3.7416573867739413
3.872983346207417
4.0
4.123105625617661
4.242640687119285
4.358898943540674
4.47213595499958
```

RDD for Armstrong Numbers between 100 and 9999.

```python
from pyspark import SparkContext, SparkConf

def is_armstrong_number(num):
    order = len(str(num))
    temp = num
    sum = 0
    while temp > 0:
        digit = temp % 10
        sum += digit ** order
        temp //= 10
    return num == sum

conf = SparkConf().setAppName("ArmstrongNumbers").setMaster("local")
sc = SparkContext(conf=conf)

numbers_rdd = sc.parallelize(range(100, 9999))

armstrong_rdd = numbers_rdd.filter(is_armstrong_number)

armstrong_numbers = armstrong_rdd.collect()
print("Armstrong numbers between 100 and 9999:", armstrong_numbers)

sc.stop()
```

```
Armstrong numbers between 100 and 9999: [153, 370, 371, 407, 1634, 8208, 9474]
```

RDD for Perfect Numbers between 1 and 100



```
from pyspark import SparkContext, SparkConf

def is_perfect_number(num):
    divisors = [i for i in range(1, num) if num % i == 0]
    return sum(divisors) == num

conf = SparkConf().setAppName("PerfectNumbers").setMaster("local")
sc = SparkContext(conf=conf)

numbers_rdd = sc.parallelize(range(1, 101))

perfect_rdd = numbers_rdd.filter(is_perfect_number)

perfect_numbers = perfect_rdd.collect()
print("Perfect numbers between 1 and 100:", perfect_numbers)

sc.stop()
```

```
Perfect numbers between 1 and 100: [6, 28]
```