Vivekanand Education Society's
Institute of Technology
Approved by AICTE & Affiliated to University of Mumbai

Artificial Intelligence and Data Science Department

BDA/Odd Sem 2023-24/Experiment 2b

Name : Manav Pahilwani	**Class/Roll No. : D16AD/ 37**	**Grade :**

Title of Experiment :
To implement the following programs using Pyspark:
1. Program to find no. of words starting specific letter (e.g., 'h' / 'a')
2. RDBMS operations:
    - Selection
    - Projection
    - Union
    - Aggregates and Grouping
    - Joins
    - Intersection

Theory:
1. Spark: Apache Spark (Software) – A Data Processing Framework: It is an open – source, distributed computing system that provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. It's designed to handle big data processing and analytics workloads. Spark supports various programming languages (Java, Scala, Python, R) and provides libraries for data processing, machine learning, graph processing and more.

Spark's key features include in – memory processing, which makes it much faster than traditional Map Reduce based systems for certain types of workloads, and a wide range of built – in libraries for machine learning (MLlib), graph processing (GraphX), SQL queries (Spark SQL) and stream processing (Spark Streaming).

Spark can be used for tasks like data transformations, batch processing, real – time stream processing, iterative machine learning algorithms and more. It's often employed in big data environments for handling large – scale data analytics tasks efficiently.

2. PySpark: PySpark is the Python library and interface for Apache Spark, which is an open-source, distributed data processing framework designed for big data analytics and large-scale data processing. PySpark allows Python developers to leverage the capabilities of Apache Spark, making it easier to work with big data processing tasks using the Python programming language.

The key features and aspects of PySpark are as follows:

Python API: PySpark provides a Python API that allows developers to interact with Spark using Python, which is a popular language for data analysis and machine learning. This means that Python developers can use familiar Python syntax and libraries to work with Spark.

Distributed Data Processing: PySpark retains the core distributed processing capabilities of Spark, enabling developers to distribute data and computation across clusters of machines for efficient and scalable data processing.

In-Memory Processing: Like Spark, PySpark supports in-memory data processing, which means it can cache intermediate data in memory to speed up computations and reduce the need for disk I/O.

Versatility: PySpark offers access to Spark's extensive ecosystem of libraries and tools, including Spark SQL for structured data processing, MLlib for machine learning, GraphX for graph processing, and more. This makes PySpark suitable for a wide range of data analytics tasks.

3. RDD: RDD stands for Resilient Distributed Dataset, and it is a fundamental data structure in Apache Spark, an open-source, distributed data processing framework. RDDs serve as the primary abstraction for data storage and processing in Spark. They are designed to handle large-scale data processing tasks efficiently in a distributed computing environment.

Program:

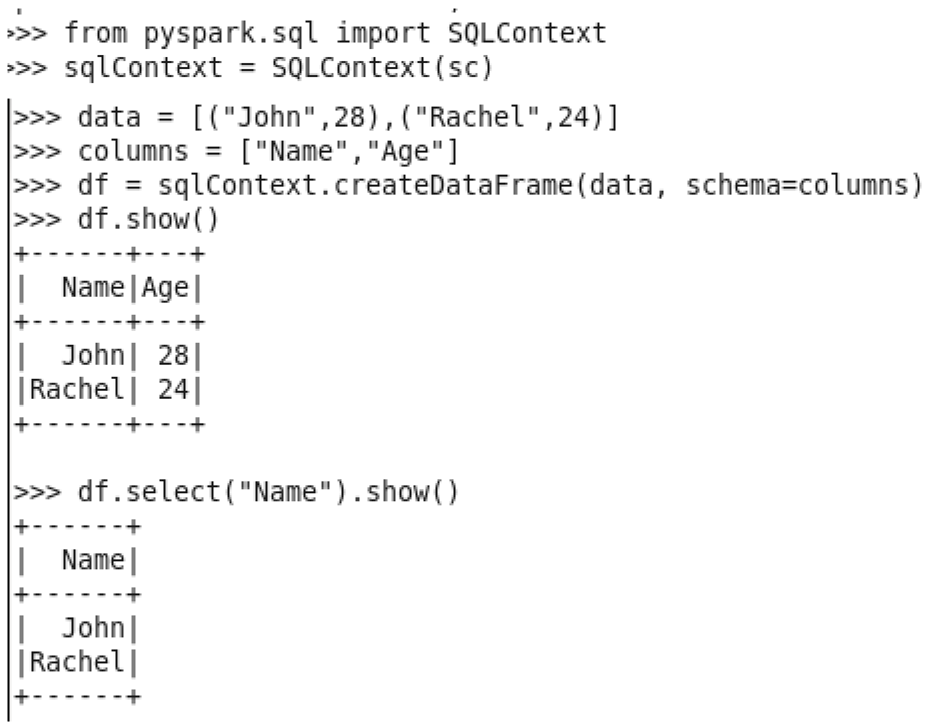
Count words starting with ‘h’:

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Python version 2.6.6 (r266:84292, Jul 23 2015 15:22:56)
SparkContext available as sc, HiveContext available as sqlContext.
>>> sc.appName
u'PySparkShell'
>>> from pyspark import SparkConf, SparkContext
>>> sc
<pyspark.context.SparkContext object at 0x2325c50>
>>> rdd = sc.textFile("file:/home/cloudera/Desktop/data.txt")
>>> rdd = rdd.flatMap(lambda line:line.split())
>>> rdd = rdd.filter(lambda word:word.startswith('h'))
>>> rdd = rdd.map(lambda word:(word,1))
>>> rdd.collect
<bound method PipelinedRDD.collect of PythonRDD[2] at RDD at PythonRDD.scala:43>
>>> rdd.collect()
23/10/15 09:37:03 WARN shortcircuit.DomainSocketFactory: The short-circuit local reads
 feature cannot be used because libhadoop cannot be loaded.
[(u'has', 1), (u'has', 1), (u'has', 1), (u'here,', 1), (u"here',", 1), (u'have', 1), (
u'humour', 1), (u'has', 1), (u'have', 1), (u'humour,', 1), (u'hidden', 1), (u'handful'
, 1), (u'humour,', 1)]
```

Word Count program:

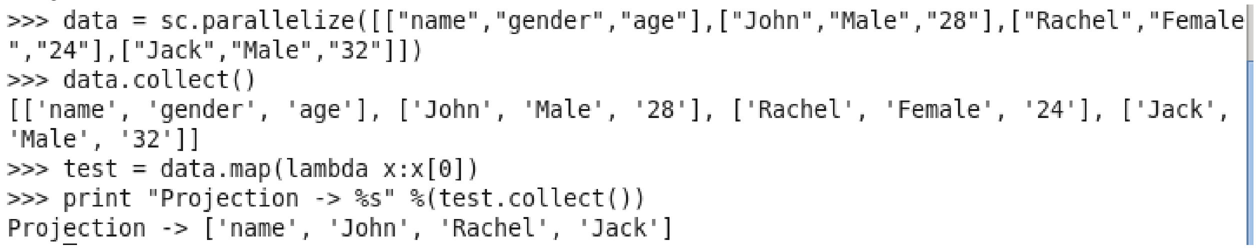
```
>>> rdd1 = sc.textFile("file:/home/cloudera/Desktop/data.txt")
>>> rdd1 = rdd1.flatMap(lambda line:line.split())
>>> rdd1 = rdd1.map(lambda word:(word,1))
>>> rdd1 = rdd1.reduceByKey(lambda a,b:a+b)
>>> rdd1.collect()
[(u'What', 1), (u'passages', 1), (u'words', 2), (u"don't", 1), (u'text', 2), (u'over',
 3), (u'random', 1), (u"ipsum'", 1), (u'years', 1), (u'discovered', 1), (u'Internet.',
 1), (u'including', 1), (u'looks', 1), (u"'lorem", 1), (u'Internet', 1), (u'1500s,', 1
), (u'still', 1), (u'its', 1), (u'roots', 1), (u'content', 2), (u'but', 2), (u'generat
or', 1), (u'also', 2), (u'Aldus', 1), (u'point', 1), (u'translation', 1), (u'editors',
 1), (u'desktop', 2), (u'to', 7), (u'only', 1), (u'going', 2), (u'1.10.32', 2), (u'1.1
0.33', 2), (u'Renaissance.', 1), (u'suffered', 1), (u'has', 4), (u'remaining', 1), (u'
PageMaker', 1), (u'Ipsum', 13), (u'centuries,', 1), (u'get', 1), (u'very', 1), (u'Many
```

Selection:



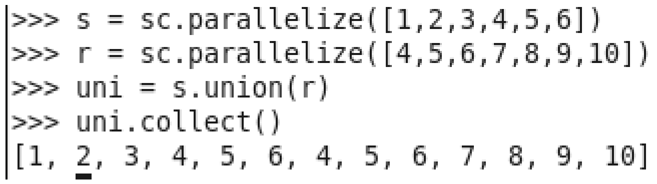
```
>>> from pyspark.sql import SQLContext
>>> sqlContext = SQLContext(sc)
>>> data = [("John",28),("Rachel",24)]
>>> columns = ["Name","Age"]
>>> df = sqlContext.createDataFrame(data, schema=columns)
>>> df.show()
+------+---+
|  Name|Age|
+------+---+
|  John| 28|
|Rachel| 24|
+------+---+

>>> df.select("Name").show()
+------+
|  Name|
+------+
|  John|
|Rachel|
+------+
```

Projection:



```
>>> data = sc.parallelize([["name","gender","age"],["John","Male","28"],["Rachel","Female
","24"],["Jack","Male","32"]])
>>> data.collect()
[['name', 'gender', 'age'], ['John', 'Male', '28'], ['Rachel', 'Female', '24'], ['Jack',
'Male', '32']]
>>> test = data.map(lambda x:x[0])
>>> print "Projection -> %s" %(test.collect())
Projection -> ['name', 'John', 'Rachel', 'Jack']
```

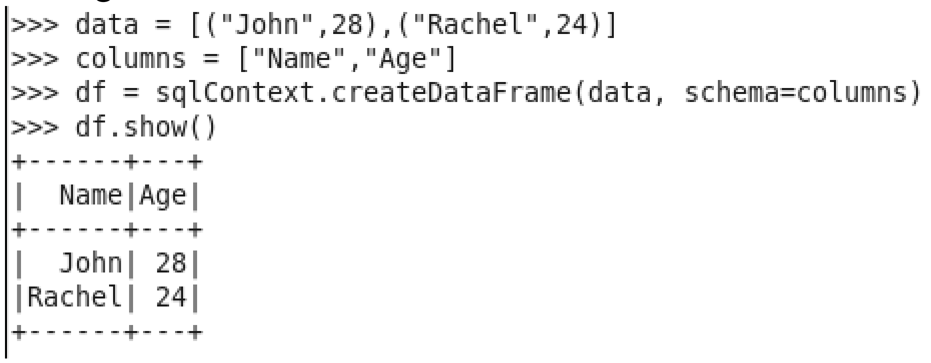
Union:



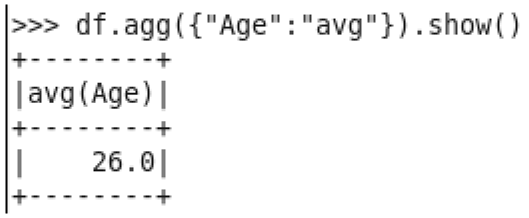
```
>>> s = sc.parallelize([1,2,3,4,5,6])
>>> r = sc.parallelize([4,5,6,7,8,9,10])
>>> uni = s.union(r)
>>> uni.collect()
[1, 2, 3, 4, 5, 6, 4, 5, 6, 7, 8, 9, 10]
```

Aggregation and Grouping:

- Average

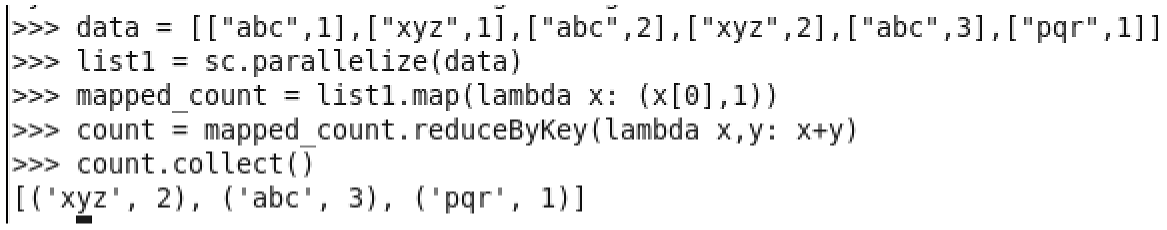


```
>>> data = [("John",28),("Rachel",24)]
>>> columns = ["Name","Age"]
>>> df = sqlContext.createDataFrame(data, schema=columns)
>>> df.show()
+------+---+
|  Name|Age|
+------+---+
|  John| 28|
|Rachel| 24|
+------+---+
```



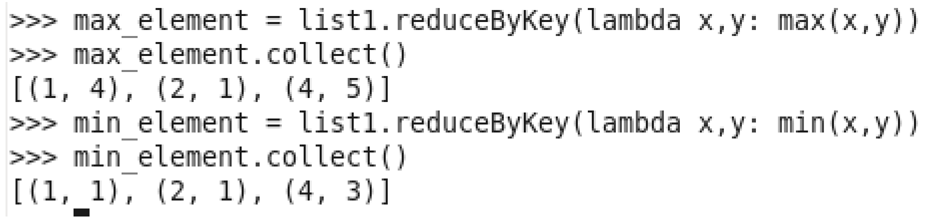
```
>>> df.agg({"Age":"avg"}).show()
+--------+
|avg(Age)|
+--------+
|    26.0|
+--------+
```

- Count



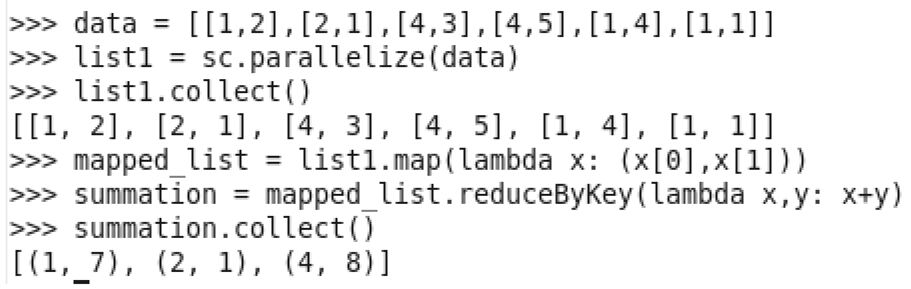
```
>>> data = [["abc",1],["xyz",1],["abc",2],["xyz",2],["abc",3],["pqr",1]]
>>> list1 = sc.parallelize(data)
>>> mapped_count = list1.map(lambda x: (x[0],1))
>>> count = mapped_count.reduceByKey(lambda x,y: x+y)
>>> count.collect()
[('xyz', 2), ('abc', 3), ('pqr', 1)]
```

- Max and min element



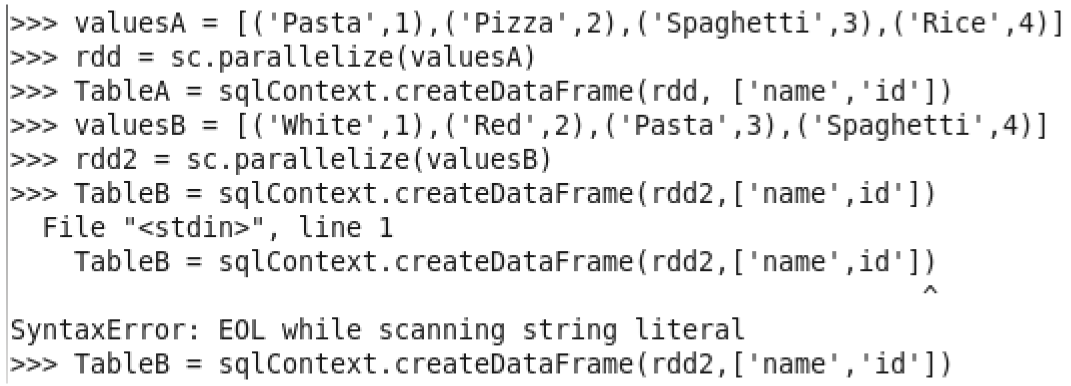
```
>>> max_element = list1.reduceByKey(lambda x,y: max(x,y))
>>> max_element.collect()
[(1, 4), (2, 1), (4, 5)]
>>> min_element = list1.reduceByKey(lambda x,y: min(x,y))
>>> min_element.collect()
[(1, 1), (2, 1), (4, 3)]
```

- Sum

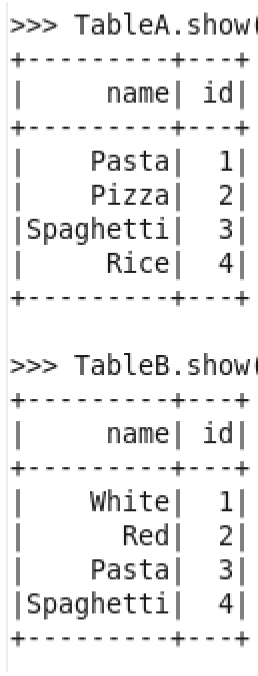


```
>>> data = [[1,2],[2,1],[4,3],[4,5],[1,4],[1,1]]
>>> list1 = sc.parallelize(data)
>>> list1.collect()
[[1, 2], [2, 1], [4, 3], [4, 5], [1, 4], [1, 1]]
>>> mapped_list = list1.map(lambda x: (x[0],x[1]))
>>> summation = mapped_list.reduceByKey(lambda x,y: x+y)
>>> summation.collect()
[(1, 7), (2, 1), (4, 8)]
```

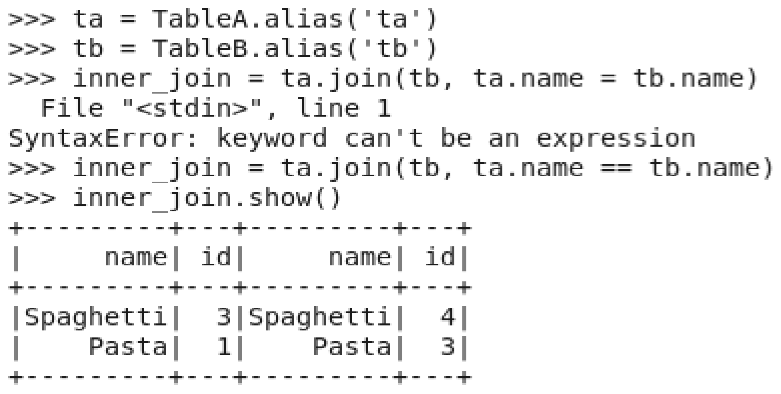
Join



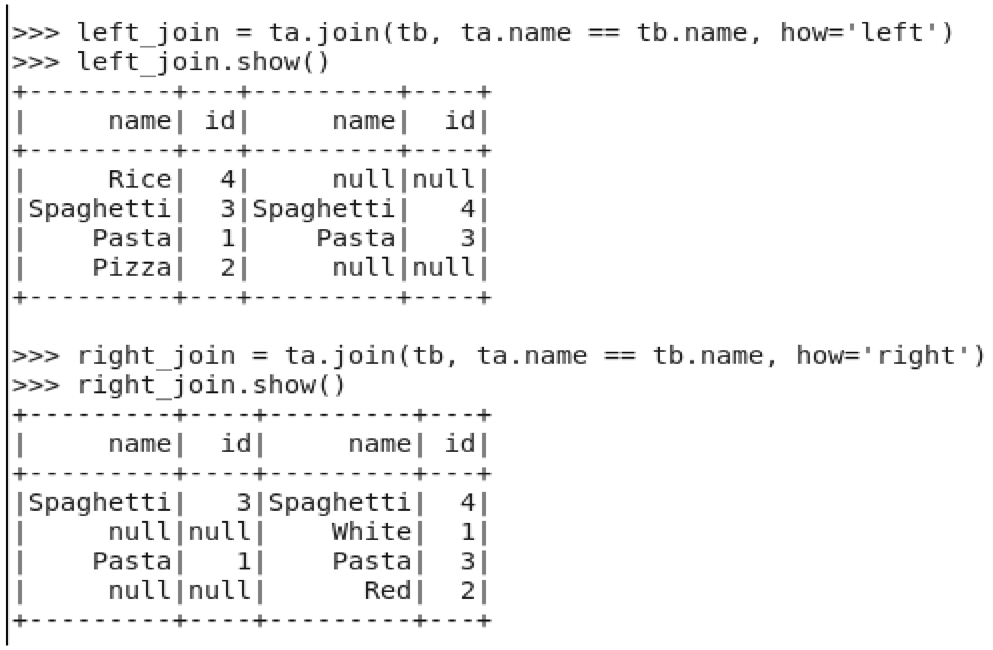
```
>>> valuesA = [('Pasta',1),('Pizza',2),('Spaghetti',3),('Rice',4)]
>>> rdd = sc.parallelize(valuesA)
>>> TableA = sqlContext.createDataFrame(rdd, ['name','id'])
>>> valuesB = [('White',1),('Red',2),('Pasta',3),('Spaghetti',4)]
>>> rdd2 = sc.parallelize(valuesB)
>>> TableB = sqlContext.createDataFrame(rdd2,['name',id'])
  File "<stdin>", line 1
    TableB = sqlContext.createDataFrame(rdd2,['name',id'])
                                                          ^
SyntaxError: EOL while scanning string literal
>>> TableB = sqlContext.createDataFrame(rdd2,['name','id'])
```



```
>>> TableA.show()
+---------+---+
|     name| id|
+---------+---+
|    Pasta|  1|
|    Pizza|  2|
|Spaghetti|  3|
|     Rice|  4|
+---------+---+

>>> TableB.show()
+---------+---+
|     name| id|
+---------+---+
|    White|  1|
|      Red|  2|
|    Pasta|  3|
|Spaghetti|  4|
+---------+---+
```

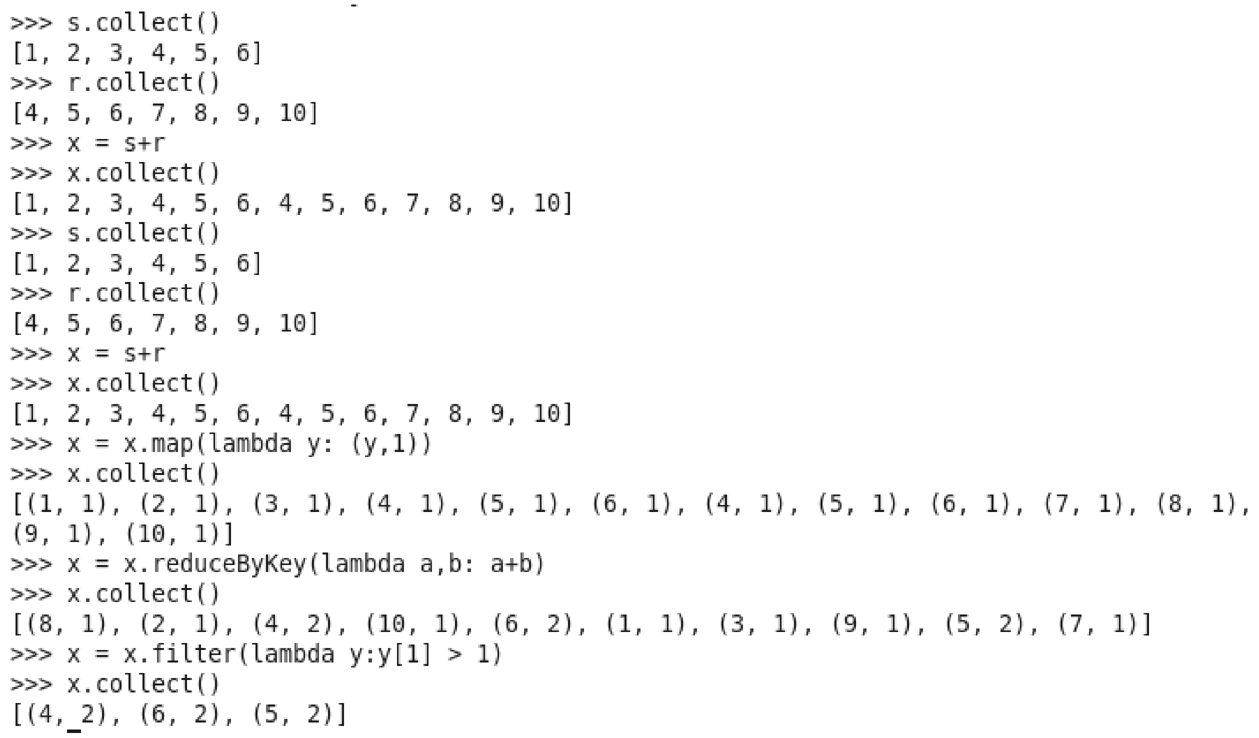


```
>>> ta = TableA.alias('ta')
>>> tb = TableB.alias('tb')
>>> inner_join = ta.join(tb, ta.name = tb.name)
  File "<stdin>", line 1
SyntaxError: keyword can't be an expression
>>> inner_join = ta.join(tb, ta.name == tb.name)
>>> inner_join.show()
+---------+---+---------+---+
|     name| id|     name| id|
+---------+---+---------+---+
|Spaghetti|  3|Spaghetti|  4|
|    Pasta|  1|    Pasta|  3|
+---------+---+---------+---+
```



```
>>> left_join = ta.join(tb, ta.name == tb.name, how='left')
>>> left_join.show()
+---------+---+---------+----+
|     name| id|     name|  id|
+---------+---+---------+----+
|     Rice|  4|     null|null|
|Spaghetti|  3|Spaghetti|   4|
|    Pasta|  1|    Pasta|   3|
|    Pizza|  2|     null|null|
+---------+---+---------+----+

>>> right_join = ta.join(tb, ta.name == tb.name, how='right')
>>> right_join.show()
+---------+----+---------+---+
|     name|  id|     name| id|
+---------+----+---------+---+
|Spaghetti|   3|Spaghetti|  4|
|     null|null|    White|  1|
|    Pasta|   1|    Pasta|  3|
|     null|null|      Red|  2|
+---------+----+---------+---+
```

Intersection



```
>>> s.collect()
[1, 2, 3, 4, 5, 6]
>>> r.collect()
[4, 5, 6, 7, 8, 9, 10]
>>> x = s+r
>>> x.collect()
[1, 2, 3, 4, 5, 6, 4, 5, 6, 7, 8, 9, 10]
>>> s.collect()
[1, 2, 3, 4, 5, 6]
>>> r.collect()
[4, 5, 6, 7, 8, 9, 10]
>>> x = s+r
>>> x.collect()
[1, 2, 3, 4, 5, 6, 4, 5, 6, 7, 8, 9, 10]
>>> x = x.map(lambda y: (y,1))
>>> x.collect()
[(1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1),
(9, 1), (10, 1)]
>>> x = x.reduceByKey(lambda a,b: a+b)
>>> x.collect()
[(8, 1), (2, 1), (4, 2), (10, 1), (6, 2), (1, 1), (3, 1), (9, 1), (5, 2), (7, 1)]
>>> x = x.filter(lambda y:y[1] > 1)
>>> x.collect()
[(4, 2), (6, 2), (5, 2)]
```

Results and Discussions :

PySpark programs cater to text analysis and data manipulation needs effectively. They mirror essential RDBMS actions and harness PySpark's distributed computing. Real – world applications span sentiment analysis to integration, with scalability and optimization as important considerations.