```
pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.5.1.tar.gz (317.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 317.0/317.0 MB 4.5 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.1-py2.py3-none-any.whl size=317488491 sha256=b7f86abed7c046c16f0c605ba617fa3359624
  Stored in directory: /root/.cache/pip/wheels/80/1d/60/2c256ed38dddce2fdd93be545214a63e02fbd8d74fb0b7f3a6
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.1
```

```
from pyspark import SparkContext

def matrix_vector_multiplication(matrix, vector):
    sc = SparkContext("local", "MatrixVectorMultiplication")
    matrix_rdd = sc.parallelize(matrix)
    vector_rdd = sc.parallelize(vector)

    result = matrix_rdd.flatMap(lambda row: [(row[0], row[1][i]*vector[i]) for i in range(len(vector))]) \
                       .reduceByKey(lambda x, y: x + y) \
                       .sortByKey() \
                       .map(lambda x: x[1]) \
                       .collect()

    sc.stop()
    return result

# Example usage
matrix = [(1, [1, 2, 3]), (2, [4, 5, 6]), (3, [7, 8, 9])]
vector = [1, 2, 3]
result = matrix_vector_multiplication(matrix, vector)
print(result)
```

```
[14, 32, 50]
```

```
from pyspark import SparkContext
import numpy as np

def aggregate_operations(data):
    sc = SparkContext("local", "AggregateOperations")
    data_rdd = sc.parallelize(data)

    mean = data_rdd.mean()
    total_sum = data_rdd.sum()
    std_dev = np.sqrt(data_rdd.map(lambda x: (x - mean)**2).sum() / data_rdd.count())

    sc.stop()
    return mean, total_sum, std_dev

# Example usage
data = [1, 2, 3, 4, 5]
mean, total_sum, std_dev = aggregate_operations(data)
print("Mean:", mean)
print("Sum:", total_sum)
print("Standard Deviation:", std_dev)
```

```
Mean: 3.0
Sum: 15
Standard Deviation: 1.4142135623730951
```

```
from pyspark import SparkContext

def sort_data(data):
    # Check if a SparkContext already exists
    sc = SparkContext.getOrCreate()

    # Create RDD from data
    data_rdd = sc.parallelize(data)

    # Sort the RDD
    sorted_data = data_rdd.sortBy(lambda x: x)

    # Stop SparkContext (if created in this function)
    if not SparkContext._active_spark_context._jsc:
        sc.stop()
```

```
        sc.stop()

    # Collect and return sorted data
    return sorted_data.collect()

# Example usage
data = [3, 1, 5, 2, 4]
sorted_data = sort_data(data)
print(sorted_data)
```

```
[1, 2, 3, 4, 5]
```

```
from pyspark import SparkContext

def search_data(data, target):
    # Check if a SparkContext already exists
    sc = SparkContext.getOrCreate()

    # Create RDD from data
    data_rdd = sc.parallelize(data)

    # Check if target exists in data
    found = data_rdd.filter(lambda x: x == target).count() > 0

    # Stop SparkContext (if created in this function)
    if not SparkContext._active_spark_context._jsc:
        sc.stop()

    return found

# Example usage
data = [1, 2, 3, 4, 5]
target = 3
result = search_data(data, target)
print("Target found:", result)
```

```
Target found: True
```

```
from pyspark import SparkContext

def map_side_join(data1, data2):
    # Check if a SparkContext already exists
    sc = SparkContext.getOrCreate()

    data1_rdd = sc.parallelize(data1)
    data2_rdd = sc.parallelize(data2)

    joined_data = data1_rdd.map(lambda x: (x[0], (x[1],))) \
                           .join(data2_rdd.map(lambda x: (x[0], (x[1],))))

    return joined_data.collect()

def reduce_side_join(data1, data2):
    # Check if a SparkContext already exists
    sc = SparkContext.getOrCreate()

    data1_rdd = sc.parallelize(data1)
    data2_rdd = sc.parallelize(data2)

    joined_data = data1_rdd.join(data2_rdd)

    return joined_data.collect()

# Example usage
data1 = [(1, 'A'), (2, 'B'), (3, 'C')]
data2 = [(1, 'X'), (2, 'Y'), (4, 'Z')]
map_side_result = map_side_join(data1, data2)
reduce_side_result = reduce_side_join(data1, data2)

print("Map Side Join Result:", map_side_result)
print("Reduce Side Join Result:", reduce_side_result)
```

```
Map Side Join Result: [(1, (('A',), ('X',))), (2, (('B',), ('Y',)))]
Reduce Side Join Result: [(1, ('A', 'X')), (2, ('B', 'Y'))]
```