structuredDataUsingRDD

February 8, 2026

[23]:
```
from pyspark.sql import SparkSession

spark = (
    SparkSession.builder.master("local[1]")  # IMPORTANT: single core
    .appName("RDD Learning")
    .config("spark.hadoop.hadoop.native.io.disable", "true")
    .config("spark.python.worker.reuse", "false")
    .getOrCreate()
)
# for windows
# .config("spark.hadoop.hadoop.native.io.disable", "true") --> this is better␣
 ↪(to store files)

sc = spark.sparkContext
```

Handling Structured Data using RDD

[ ]:
```
# split based on delimiter
iris1 = sc.textFile("iris/iris_site.csv")

iris1_split = iris1.map(lambda var1: var1.split(","))
print(iris1_split.take(10))
```

```
[['5.1', '3.5', '1.4', '0.2', 'setosa'], ['4.9', '3.0', '1.4', '0.2', 'setosa'],
['4.7', '3.2', '1.3', '0.2', 'setosa'], ['4.6', '3.1', '1.5', '0.2', 'setosa'],
['5.0', '3.6', '1.4', '0.2', 'setosa'], ['5.4', '3.9', '1.7', '0.4', 'setosa'],
['4.6', '3.4', '1.4', '0.3', 'setosa'], ['5.0', '3.4', '1.5', '0.2', 'setosa'],
['4.4', '2.9', '1.4', '0.2', 'setosa'], ['4.9', '3.1', '1.5', '0.1', 'setosa']]
```

[ ]:
```
# chaining spark commands
# in spark multiple transformation and action can be combined by chaining
# them with dot operator
sc.textFile("iris/iris_site.csv").map(lambda var1: var1.split(",")).take(10)


# defining transformation(instead of lambda functions we can use normal defined␣
 ↪functions)
def fun1(var1):
```

```
    return var1.split(",")


sc.textFile("iris/iris_site.csv").map(fun1).take(10)
```

[ ]:
```
[['5.1', '3.5', '1.4', '0.2', 'setosa'],
 ['4.9', '3.0', '1.4', '0.2', 'setosa'],
 ['4.7', '3.2', '1.3', '0.2', 'setosa'],
 ['4.6', '3.1', '1.5', '0.2', 'setosa'],
 ['5.0', '3.6', '1.4', '0.2', 'setosa'],
 ['5.4', '3.9', '1.7', '0.4', 'setosa'],
 ['4.6', '3.4', '1.4', '0.3', 'setosa'],
 ['5.0', '3.4', '1.5', '0.2', 'setosa'],
 ['4.4', '2.9', '1.4', '0.2', 'setosa'],
 ['4.9', '3.1', '1.5', '0.1', 'setosa']]
```

[ ]:
```
# Extracting a particular column
iris1 = sc.textFile("iris/iris_site.csv")
iris1_split = iris1.map(lambda var1: var1.split(","))

iris1_mod = iris1_split.map(lambda col: col[0])  # only 1st column
print(iris1_mod.take(10))
```

```
['5.1', '4.9', '4.7', '4.6', '5.0', '5.4', '4.6', '5.0', '4.4', '4.9']
```

[ ]:
```
# type casting
iris1 = sc.textFile("iris/iris_site.csv")
iris1_split = iris1.map(lambda var1: var1.split(","))

type_casting = iris1_split.map(lambda col: float(col[1]))
print(type_casting.collect())
```

```
[3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.4, 3.0, 3.0, 4.0, 4.4,
3.9, 3.5, 3.8, 3.8, 3.4, 3.7, 3.6, 3.3, 3.4, 3.0, 3.4, 3.5, 3.4, 3.2, 3.1, 3.4,
4.1, 4.2, 3.1, 3.2, 3.5, 3.6, 3.0, 3.4, 3.5, 2.3, 3.2, 3.5, 3.8, 3.0, 3.8, 3.2,
3.7, 3.3, 3.2, 3.2, 3.1, 2.3, 2.8, 2.8, 3.3, 2.4, 2.9, 2.7, 2.0, 3.0, 2.2, 2.9,
2.9, 3.1, 3.0, 2.7, 2.2, 2.5, 3.2, 2.8, 2.5, 2.8, 2.9, 3.0, 2.8, 3.0, 2.9, 2.6,
2.4, 2.4, 2.7, 2.7, 3.0, 3.4, 3.1, 2.3, 3.0, 2.5, 2.6, 3.0, 2.6, 2.3, 2.7, 3.0,
2.9, 2.9, 2.5, 2.8, 3.3, 2.7, 3.0, 2.9, 3.0, 3.0, 2.5, 2.9, 2.5, 3.6, 3.2, 2.7,
3.0, 2.5, 2.8, 3.2, 3.0, 3.8, 2.6, 2.2, 3.2, 2.8, 2.8, 2.7, 3.3, 3.2, 2.8, 3.0,
2.8, 3.0, 2.8, 3.8, 2.8, 2.8, 2.6, 3.0, 3.4, 3.1, 3.0, 3.1, 3.1, 3.1, 2.7, 3.2,
3.3, 3.0, 2.5, 3.0, 3.4, 3.0]
```

[ ]:
```
# for loop with rdd objects
# type_casting=iris1_split.map(lambda col:float(col[1]))
for var1 in type_casting.take(10):
    print(var1, type(var1))
```

```
3.5 <class 'float'>
```

```
3.0 <class 'float'>
3.2 <class 'float'>
3.1 <class 'float'>
3.6 <class 'float'>
3.9 <class 'float'>
3.4 <class 'float'>
3.4 <class 'float'>
2.9 <class 'float'>
3.1 <class 'float'>
```

[ ]:
```
# typecasting multiple columns
iris_split_mod = iris1_split.map(
    lambda var1: [
        float(var1[0]),
        float(var1[1]),
        float(var1[2]),
        float(var1[3]),
        var1[4],
    ]
)
print(iris_split_mod.take(10))
```

```
[[5.1, 3.5, 1.4, 0.2, 'setosa'], [4.9, 3.0, 1.4, 0.2, 'setosa'], [4.7, 3.2, 1.3,
0.2, 'setosa'], [4.6, 3.1, 1.5, 0.2, 'setosa'], [5.0, 3.6, 1.4, 0.2, 'setosa'],
[5.4, 3.9, 1.7, 0.4, 'setosa'], [4.6, 3.4, 1.4, 0.3, 'setosa'], [5.0, 3.4, 1.5,
0.2, 'setosa'], [4.4, 2.9, 1.4, 0.2, 'setosa'], [4.9, 3.1, 1.5, 0.1, 'setosa']]
```

[ ]:
```
# creating key value pairs in RDD
iris1_mod = iris1_split.map(
    lambda var1: (
        ("Sepal.Length", float(var1[0])),
        ("Sepal.Width", float(var1[1])),
        ("Petal.Length", float(var1[2])),
        ("Petal.Width", float(var1[3])),
    )
)  # by using this the entire key value collection is present inside a double␣
 ↪collection
# to remove this extra level we use flatMap function instead of map
print(iris1_mod.take(10))
```

```
[(('Sepal.Length', 5.1), ('Sepal.Width', 3.5), ('Petal.Length', 1.4),
('Petal.Width', 0.2)), (('Sepal.Length', 4.9), ('Sepal.Width', 3.0),
('Petal.Length', 1.4), ('Petal.Width', 0.2)), (('Sepal.Length', 4.7),
('Sepal.Width', 3.2), ('Petal.Length', 1.3), ('Petal.Width', 0.2)),
(('Sepal.Length', 4.6), ('Sepal.Width', 3.1), ('Petal.Length', 1.5),
('Petal.Width', 0.2)), (('Sepal.Length', 5.0), ('Sepal.Width', 3.6),
('Petal.Length', 1.4), ('Petal.Width', 0.2)), (('Sepal.Length', 5.4),
('Sepal.Width', 3.9), ('Petal.Length', 1.7), ('Petal.Width', 0.4)),
(('Sepal.Length', 4.6), ('Sepal.Width', 3.4), ('Petal.Length', 1.4),
```

```
('Petal.Width', 0.3)), (('Sepal.Length', 5.0), ('Sepal.Width', 3.4),
('Petal.Length', 1.5), ('Petal.Width', 0.2)), (('Sepal.Length', 4.4),
('Sepal.Width', 2.9), ('Petal.Length', 1.4), ('Petal.Width', 0.2)),
(('Sepal.Length', 4.9), ('Sepal.Width', 3.1), ('Petal.Length', 1.5),
('Petal.Width', 0.1))]
```

[14]:
```
iris1_mod = iris1_split.flatMap(
    lambda var1: (
        ("Sepal.Length", float(var1[0])),
        ("Sepal.Width", float(var1[1])),
        ("Petal.Length", float(var1[2])),
        ("Petal.Width", float(var1[3])),
    )
)
print(iris1_mod.take(10))
```

```
[('Sepal.Length', 5.1), ('Sepal.Width', 3.5), ('Petal.Length', 1.4),
('Petal.Width', 0.2), ('Sepal.Length', 4.9), ('Sepal.Width', 3.0),
('Petal.Length', 1.4), ('Petal.Width', 0.2), ('Sepal.Length', 4.7),
('Sepal.Width', 3.2)]
```

0.1 Sorting RDD

sorted based on a particular column using sortBy function

sortByKey –> sorting rdd based on key

[ ]:
```
iris1 = sc.textFile("iris/iris_site.csv")
iris1_split = iris1.map(lambda var1: var1.split(","))  # this is common for all␣
 ↪cells
```

[ ]:
```
# simple sort
iris1_mod = iris1_split.map(
    lambda var1: (
        float(var1[0]),
        float(var1[1]),
        float(var1[2]),
        float(var1[3]),
        var1[4],
    )
)  # type convert all string columns

iris1_mod.sortBy(lambda x: x[0]).take(10)
```

[ ]:
```
[(4.3, 3.0, 1.1, 0.1, 'setosa'),
 (4.4, 2.9, 1.4, 0.2, 'setosa'),
 (4.4, 3.0, 1.3, 0.2, 'setosa'),
 (4.4, 3.2, 1.3, 0.2, 'setosa'),
 (4.5, 2.3, 1.3, 0.3, 'setosa'),
```

```
 (4.6, 3.1, 1.5, 0.2, 'setosa'),
 (4.6, 3.4, 1.4, 0.3, 'setosa'),
 (4.6, 3.6, 1.0, 0.2, 'setosa'),
 (4.6, 3.2, 1.4, 0.2, 'setosa'),
 (4.7, 3.2, 1.3, 0.2, 'setosa')]
```

[ ]:
```
# sorting RDD based on key
iris1_mod = iris1_split.map(
    lambda var1: (var1[4], [var1[0], float(var1[1]), float(var1[2]),␣
 ↪float(var1[3])])
)

iris1_mod.sortByKey(ascending=True, keyfunc=lambda k: k).take(
    10
)  # defaultly its sorts based on key only, so lambda is unnecessary
```

[ ]:
```
[('setosa', ['5.1', 3.5, 1.4, 0.2]),
 ('setosa', ['4.9', 3.0, 1.4, 0.2]),
 ('setosa', ['4.7', 3.2, 1.3, 0.2]),
 ('setosa', ['4.6', 3.1, 1.5, 0.2]),
 ('setosa', ['5.0', 3.6, 1.4, 0.2]),
 ('setosa', ['5.4', 3.9, 1.7, 0.4]),
 ('setosa', ['4.6', 3.4, 1.4, 0.3]),
 ('setosa', ['5.0', 3.4, 1.5, 0.2]),
 ('setosa', ['4.4', 2.9, 1.4, 0.2]),
 ('setosa', ['4.9', 3.1, 1.5, 0.1])]
```

[ ]:
```
# Union Multiple RDD's
SL = iris1_split.map(lambda var1: ["Sepal.Length", float(var1[0])])
SW = iris1_split.map(lambda var1: ["Sepal.Width", float(var1[1])])
PL = iris1_split.map(lambda var1: ["Petal.Length", float(var1[2])])
PW = iris1_split.map(lambda var1: ["Petal.Width", float(var1[3])])
CV = iris1_split.map(lambda var1: ["Species", var1[4]])

print(SL.take(10))
print(SW.take(10))
print(PL.take(10))
print(PW.take(10))

union_data = sc.union([SL, SW, PL, PW])
print(union_data.take(10))
```

```
[['Sepal.Length', 5.1], ['Sepal.Length', 4.9], ['Sepal.Length', 4.7],
['Sepal.Length', 4.6], ['Sepal.Length', 5.0], ['Sepal.Length', 5.4],
['Sepal.Length', 4.6], ['Sepal.Length', 5.0], ['Sepal.Length', 4.4],
['Sepal.Length', 4.9]]
[['Sepal.Width', 3.5], ['Sepal.Width', 3.0], ['Sepal.Width', 3.2],
['Sepal.Width', 3.1], ['Sepal.Width', 3.6], ['Sepal.Width', 3.9],
```

```
['Sepal.Width', 3.4], ['Sepal.Width', 3.4], ['Sepal.Width', 2.9],
['Sepal.Width', 3.1]]
[['Petal.Length', 1.4], ['Petal.Length', 1.4], ['Petal.Length', 1.3],
['Petal.Length', 1.5], ['Petal.Length', 1.4], ['Petal.Length', 1.7],
['Petal.Length', 1.4], ['Petal.Length', 1.5], ['Petal.Length', 1.4],
['Petal.Length', 1.5]]
[['Petal.Width', 0.2], ['Petal.Width', 0.2], ['Petal.Width', 0.2],
['Petal.Width', 0.2], ['Petal.Width', 0.2], ['Petal.Width', 0.4],
['Petal.Width', 0.3], ['Petal.Width', 0.2], ['Petal.Width', 0.2],
['Petal.Width', 0.1]]
[['Sepal.Length', 5.1], ['Sepal.Length', 4.9], ['Sepal.Length', 4.7],
['Sepal.Length', 4.6], ['Sepal.Length', 5.0], ['Sepal.Length', 5.4],
['Sepal.Length', 4.6], ['Sepal.Length', 5.0], ['Sepal.Length', 4.4],
['Sepal.Length', 4.9]]
```

[ ]:
```python
# Intersection (used to find the common elements of two RDD's)
r1 = sc.parallelize(["a", "b", "c", "d", "e"])
r2 = sc.parallelize(["d", "e", "f", "g"])
# print(r1.intersection(r2).collect()) works in python 3.10 or 3.9 version
```

Joins

[ ]:
```python
# Joins
#  Two rdds containing data in the form of key-value pair can be joined
# by using join function
location1 = sc.parallelize(
    [("employee1", [3, 1, 2, 5, 4]), ("employee2", [2, 4, 1, 2, 2])]
)
location2 = sc.parallelize(
    [("employee2", [2, 1, 1, 1, 2]), ("employee1", [4, 5, 2, 1, 1])]
)

join1 = location1.join(location2)
print(join1.collect())
```

```
[('employee1', ([3, 1, 2, 5, 4], [4, 5, 2, 1, 1])), ('employee2', ([2, 4, 1, 2,
2], [2, 1, 1, 1, 2]))]
```

[ ]:
```python
# extract column values of second RDD
print(join1.map(lambda var1: [var1[0], var1[1][1]]).collect())
```

```
[['employee1', [4, 5, 2, 1, 1]], ['employee2', [2, 1, 1, 1, 2]]]
```

[ ]:
```python
# full example
DimEmployee = sc.textFile("AdventureWorks/AdventureWorks_RDD/DimEmployee.csv")
FactResellerSales = sc.textFile(
    "AdventureWorks/AdventureWorks_RDD/FactResellerSales.csv"
)
```

```
# split data
DimEmployee = DimEmployee.map(lambda var1: var1.split(","))
FactResellerSales = FactResellerSales.map(lambda var1: var1.split(","))

# convert to key-value pairs
DimEmployee = DimEmployee.map(lambda var1: [var1[0], [var1[1], var1[2],␣
 ↪var1[3]]])
FactResellerSales = FactResellerSales.map(
    lambda var1: [
        var1[5],
        [
            var1[0],
            var1[1],
            var1[2],
            var1[3],
            var1[4],
            var1[5],
            var1[6],
            var1[7],
            float(var1[8]),
            float(var1[9]),
            float(var1[10]),
            float(var1[11]),
            float(var1[12]),
            float(var1[13]),
            float(var1[14]),
            float(var1[15]),
        ],
    ]
)

# perform join
RF = FactResellerSales.join(DimEmployee)
RF.take(10)
print(RF.map(lambda var1: (var1[0], var1[1][0])).take(4))
```

```
[('282', ['339', '20070401', '20070413', '20070408', '523', '282', '1', '3',
2.0, 469.794, 0.0, 0.0, 486.7066, 973.4132, 939.588, 23.4897]), ('282', ['213',
'20060801', '20060813', '20060808', '403', '282', '1', '4', 2.0, 20.1865, 0.0,
0.0, 13.8782, 27.7564, 40.373, 1.0093]), ('282', ['586', '20070701', '20070713',
'20070708', '205', '282', '13', '4', 13.0, 334.0575, 0.15, 651.4121, 461.4448,
5998.7824, 3691.3354, 92.2834]), ('282', ['459', '20060801', '20060813',
'20060808', '79', '282', '1', '3', 5.0, 53.994, 0.0, 0.0, 37.1209, 185.6045,
269.97, 6.7493])]
```

[ ]:
```
# Loop through values
iris1 = sc.textFile("iris/iris_site.csv")
```

```
iris1_split = iris1.map(lambda var1: var1.split(","))
iris1_mod = iris1_split.flatMap(
    lambda var1: (
        ("Sepal.Length", float(var1[0])),
        ("Sepal.Width", float(var1[1])),
        ("Petal.Length", float(var1[2])),
        ("Petal.Width", float(var1[3])),
    )
)

def fun(x): print(x)

iris1_mod.foreach(fun)
# print(iris1_mod.take(10))
for data in iris1_mod.take(10):
    print(data)
```

```
[('Sepal.Length', 5.1), ('Sepal.Width', 3.5), ('Petal.Length', 1.4),
('Petal.Width', 0.2), ('Sepal.Length', 4.9), ('Sepal.Width', 3.0),
('Petal.Length', 1.4), ('Petal.Width', 0.2), ('Sepal.Length', 4.7),
('Sepal.Width', 3.2)]
('Sepal.Length', 5.1)
('Sepal.Width', 3.5)
('Petal.Length', 1.4)
('Petal.Width', 0.2)
('Sepal.Length', 4.9)
('Sepal.Width', 3.0)
('Petal.Length', 1.4)
('Petal.Width', 0.2)
('Sepal.Length', 4.7)
('Sepal.Width', 3.2)
```

[33]:
```
# filter data based on  a condition
filter1=iris1_split.filter(lambda x:float(x[0])>7)
print(filter1.take(10))
```

```
[['7.1', '3.0', '5.9', '2.1', 'virginica'], ['7.6', '3.0', '6.6', '2.1',
'virginica'], ['7.3', '2.9', '6.3', '1.8', 'virginica'], ['7.2', '3.6', '6.1',
'2.5', 'virginica'], ['7.7', '3.8', '6.7', '2.2', 'virginica'], ['7.7', '2.6',
'6.9', '2.3', 'virginica'], ['7.7', '2.8', '6.7', '2.0', 'virginica'], ['7.2',
'3.2', '6.0', '1.8', 'virginica'], ['7.2', '3.0', '5.8', '1.6', 'virginica'],
['7.4', '2.8', '6.1', '1.9', 'virginica']]
```