## reduceByKey()

02:30 PM (2m) 18

```
1 orders_rdd = spark.sparkContext.textFile("/FileStore/tables/orders_1gb.csv")
2 mapped_rdd = orders_rdd.map(lambda x: (x.split(",")[3],1))
3 results = mapped_rdd.reduceByKey(lambda x,y : x+y)
4 results.collect()
```

▸ (1) Spark Jobs

```
[('ON_HOLD', 1424250),
 ('CANCELED', 535500),
 ('PAYMENT_REVIEW', 273375),
 ('CLOSED', 2833500),
 ('PENDING_PAYMENT', 5636250),
 ('PENDING', 2853750),
 ('SUSPECTED_FRAUD', 584250),
 ('COMPLETE', 8587125),
 ('PROCESSING', 3103125)]
```

## countByValue()

02:39 PM (1m) 19

```
1 orders_rdd = spark.sparkContext.textFile("/FileStore/tables/orders_1gb.csv")
2 mapped_rdd = orders_rdd.map(lambda x: (x.split(",")[3]))
3 mapped_rdd.countByValue()
```

▸ (1) Spark Jobs

```
defaultdict(int,
            {'CLOSED': 2833500,
             'PENDING_PAYMENT': 5636250,
             'COMPLETE': 8587125,
             'PROCESSING': 3103125,
             'PAYMENT_REVIEW': 273375,
             'PENDING': 2853750,
             'ON_HOLD': 1424250,
             'CANCELED': 535500,
             'SUSPECTED_FRAUD': 584250})
```

✅ **When to Use reduceByKey() :**

- You're working with **key-value RDDs** (e.g., (key, value)).
- You need to **aggregate values by key** using a custom function (e.g., sum, max, min, average).
- You're dealing with **large datasets**, and you want the computation to remain **distributed** (not collected to the driver).
- You plan to **chain more operations** on the result (since the output is still an RDD).

✅ **When to Use countByValue() :**

- You have an RDD of **single values**, not key-value pairs.
- You just want a **quick count of each unique item**.
- The dataset is **small enough** to safely bring results **to the driver as a Python dictionary**.
- You don’t need further distributed processing on the result.