**Pyspark Assignment - 2**

**Name: Vikas Reddy Gorantla**

**Date: 22-11-2024**

**Manipulating, Droping, Sorting, Aggregations, Joining, GroupeBy DataFrames**

**1. groupBy:**

Groups the DataFrame rows based on one or more specified columns. This is the foundation for applying aggregate operations like sum, min, max, etc., on grouped data.

**2. sum:**

Calculates the total sum of a specified column for each group, which is helpful in aggregating numerical data such as salaries.

**3. min:**

Finds the smallest value in a specified column for each group. This is used to determine the lower limit or minimum value for grouped data.

**4. max:**

Finds the largest value in a specified column for each group. It is useful for determining the upper limit or maximum value in the grouped data.

**5. avg:**

Calculates the average (mean) value of a specified column for each group. It helps to understand the central tendency of grouped numerical data.

**6. mean:**

Similar to avg, it computes the arithmetic mean of a specified column for each group. It is an alternative function for calculating the average.

**7. count:**

Counts the number of rows in each group. This helps to determine the size or frequency of each group.

**8. pivot:**

Reshapes data by transforming unique values of a specified column into individual columns, enabling comparative analysis across categories.

**9. agg:**

Allows applying multiple aggregate functions to one or more columns in a single operation. It provides advanced aggregation capabilities beyond basic functions like sum or avg.

**10. na.drop():**

Removes rows with null or missing values from the DataFrame. It is essential for cleaning data and ensuring completeness during analysis.

**11. sort:**

Organizes the DataFrame rows based on one or more columns:

* + Default ascending order sorts from smallest to largest values.
  + Descending order (using desc()) arranges values from largest to smallest.

**Top of Form**

from pyspark.sql import SparkSession

# Initialize SparkSession

spark = SparkSession.builder \

    .appName("example") \

    .getOrCreate()

# Data

simpleData = [

    ("James", "Sales", "NY", 90000, 34, 10000),

    ("Michael", "Sales", "NY", 86000, 56, 20000),

    ("Robert", "Sales", "CA", 81000, 30, 23000),

    ("Maria", "Finance", "CA", 90000, 24, 23000),

    ("Raman", "Finance", "CA", 99000, 40, 24000),

    ("Scott", "Finance", "NY", 83000, 36, 19000),

    ("Jen", "Finance", "NY", 79000, 53, 15000),

    ("Jeff", "Marketing", "CA", 80000, 25, 18000),

    ("Kumar", "Marketing", "NY", 91000, 50, 21000)]

# Create DataFrame

schema = ["employee\_name", "department", "state", "salary", "age", "bonus"]

df = spark.createDataFrame(data=simpleData, schema=schema)

# Display schema

df.printSchema()

df.show()

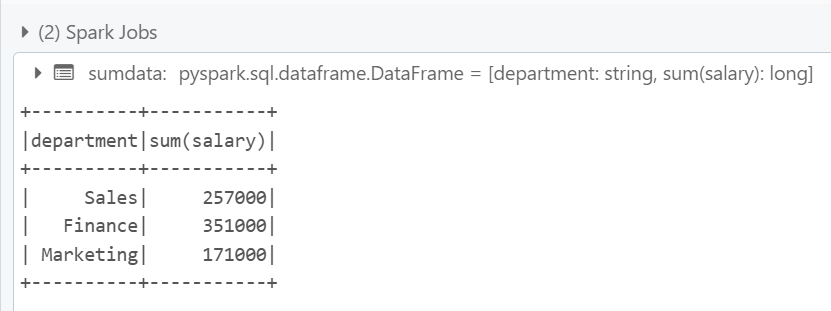
A screenshot of a computer

Description automatically generated

# groupby with sum of salaries

sumdata = df.groupBy("department").sum("salary")

sumdata.show()



df.groupBy("department").min("salary").show()

df.groupBy("department").max("salary").show()

df.groupBy("department").avg("salary").show()

df.groupBy("department").mean("salary").show()

df.groupBy("department").count().show()

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

#grouping by multiple columns

df.groupBy("employee\_name","department").min("salary").show()

df.groupBy("employee\_name","department").max("salary").show()

df.groupBy("employee\_name","department").avg("salary").show()

df.groupBy("employee\_name","department").mean("salary").show()

A screenshot of a data

Description automatically generateddf.groupBy("employee\_name","department").count().show()

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

#using pivot function

df.groupBy("department").sum("salary").show()

df.groupBy("department").pivot("employee\_name").sum("salary").show()

A screenshot of a computer

Description automatically generated

df.groupBy("department").agg(({"salary":"sum"})).show()

df.agg(({"salary":"sum"})).show()  # Without group using agg on salary colums

A screenshot of a computer

Description automatically generated

from pyspark.sql import SparkSession

# Initialize SparkSession

spark = SparkSession.builder \

.appName("example") \

.getOrCreate()

# Data

simpleData = [("James","Sales","NY",90000,34,10000),

("Michael","Sales","NY",86000,56,20000),

("Robert","Sales","CA",81000,None,23000),

("Maria","Finance","CA",90000,24,23000),

("Raman","Finance","CA",99000,40,None),

("Scott","Finance","NY",None,36,44000),

("Jen","Finance","NY",55000,53,15000),

("Jeff",None,"CA",80000,25,18000),

("null","Marketing","NY",91000,50,21000)]

# Create DataFrame

schema = ["employee\_name","department","state","salary","age","bonus"]

dfa = spark.createDataFrame(data=simpleData, schema = schema)

dfa.show()

A screenshot of a computer

Description automatically generated

# Initialize SparkSession

spark = SparkSession.builder \

.appName("example") \

.getOrCreate()

# Data

simpleData = [("James","Sales","NY",90000,34,10000),

("Michael","Sales","NY",86000,56,20000),

("Robert","Sales","CA",81000,None,23000),

("Maria","Finance","CA",90000,24,23000),

("Raman","Finance","CA",99000,40,None),

("Scott","Finance","NY",None,36,44000),

("Jen","Finance","NY",55000,53,15000),

("Jeff","null","CA",80000,25,18000),

("null","Marketing","NY",91000,50,21000)]

# Create DataFrame

schema = ["employee\_name","department","state","salary","age","bonus"]

dfa = spark.createDataFrame(data=simpleData, schema = schema)

dfa.na.drop().show()

A screenshot of a computer

Description automatically generated

df.sort('bonus').show()

df.sort(df['salary'].desc()).show()

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Joins in pyspark**

**Inner Join:**

Join records when key columns are matched, and dropped when they are not matched

**Outer join:**

Returns all rows from both datasets, where Join expression doesn't match it returns null or respective columns

**Left Join/ Left outer join:**

Returns all rows from left dataset regardless of match found on right dataset, when Join doesn't match it assigns null for that record

**Right Join/ Right outer join:**

Returns all rows from Right dataset regardless of match found on left dataset, when Join doesn't match - it assigns null for that record

**Left Semi Join:**

Returns columns from the only left dataset for the matched records in the right dataset on join expression

**Left Anti Join:**

Returns only columns from left dataset for non-matched records

from pyspark.sql import SparkSession

# Initialize SparkSession

spark = SparkSession.builder \

.appName("example") \

.getOrCreate()

# Data

emp = [(1,"Smith",-1,"2018","10","M",3000),

       (2, "Rose",1 , "2010", "20","M", 4000),

       (3,"Williams",1,"2010","10","M",1000),

       (4, "Jones",2 ,"2005","10","F",2000),

       (5,"Brown",2,"2010","40","",-1),

       (6, "Brown", 2, "2010","50","",-1)]

empColumns = ["emp\_id","name","superior\_emp\_id","year\_joined", "emp\_dept\_id","gender","salary"]

empDF = spark.createDataFrame(data=emp, schema = empColumns)

empDF.printSchema()

empDF.show()

dept = [("Finance",10),("Marketing",20),("Sales",30),("IT",40)]

deptColumns = ["dept\_name","dept\_id"]

deptDF = spark.createDataFrame(data=dept, schema = deptColumns)

deptDF.printSchema()

deptDF.show()

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

#Inner join

empDF.join(deptDF,empDF.emp\_dept\_id == deptDF.dept\_id, "inner").show()

A screenshot of a computer code

Description automatically generated

#Outer join

empDF.join(deptDF,empDF.emp\_dept\_id == deptDF.dept\_id, "outer").show()

A screenshot of a computer code

Description automatically generated

#Full join

empDF.join(deptDF,empDF.emp\_dept\_id == deptDF.dept\_id, "full").show()

A screenshot of a computer code

Description automatically generated

#Left join

empDF.join(deptDF,empDF.emp\_dept\_id == deptDF.dept\_id, "left").show()

#Left outer join

empDF.join(deptDF,empDF.emp\_dept\_id == deptDF.dept\_id, "leftouter").show()

A screenshot of a computer code

Description automatically generated

#Right join

empDF.join(deptDF,empDF.emp\_dept\_id == deptDF.dept\_id, "right").show()

#Right outer join

empDF.join(deptDF,empDF.emp\_dept\_id == deptDF.dept\_id, "rightouter").show()

A screenshot of a computer

Description automatically generated

#Left semi join

empDF.join(deptDF,empDF.emp\_dept\_id == deptDF.dept\_id, "leftsemi").show()

#Left anti join

empDF.join(deptDF,empDF.emp\_dept\_id == deptDF.dept\_id, "leftanti").show()

A screenshot of a computer code

Description automatically generated

**Spark SQL**

# File location and type

file\_location = "/FileStore/tables/simple\_zipcodes.csv"

file\_type = "csv"

# CSV options

infer\_schema = "false"

first\_row\_is\_header = "false"

delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.

df = spark.read.format(file\_type) \

  .option("inferSchema", infer\_schema) \

  .option("header", first\_row\_is\_header) \

  .option("sep", delimiter) \

  .load(file\_location)

display(df)

df.createOrReplaceTempView("tempdata")

A screenshot of a computer

Description automatically generated

spark.sql("select \* from tempdata").show()

df.select("\_c0","\_c1").show(5)

A screenshot of a computer

Description automatically generated

spark.sql("""SELECT \* From tempdata WHERE \_c4='AZ'""").show(5)

A white background with black text

Description automatically generated

# File location and type

file\_location = "/FileStore/tables/simple\_zipcodes.csv"

file\_type = "csv"

# CSV options

infer\_schema = "false"

first\_row\_is\_header = "true"

delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.

df = spark.read.format(file\_type) \

  .option("inferSchema", infer\_schema) \

  .option("header", first\_row\_is\_header) \

  .option("sep", delimiter) \

  .load(file\_location)

display(df)

df.createOrReplaceTempView("customer")

A screenshot of a computer

Description automatically generated

spark.sql("""SELECT \* From customer WHERE state='PR'""").show(5)

A close-up of a ticket

Description automatically generated

spark.sql("""select \* FROM customer WHERE state in ('PR','AZ','FL')order by state """).show(10)

A screenshot of a computer

Description automatically generated

spark.sql("""SELECT state,count(\*) as count FROM customer GROUP BY state""").show()

A screenshot of a computer

Description automatically generated