Pyspark

1. Catalog: upload csv files for the pyspark operation
2. Create cluster
3. Read = df=spark.read.format(‘csv’).option(‘inferschema’,True).option(‘header’,True).load(‘path’)

%md

### for giving title

details.fs.utils()- To find the folder path inside our catalog

df.display() – print

Creating data using ddl & StructType()

df.printschema()-print the structure

ddl – store the schema sql and store it in variable

df=spark.read.format(‘csv’).schema(schemaname).option(‘header’,True).load(‘path’)

import modules

from pyspark.sql.types import \*

from pyspark.sql.functions import \*

StructType([ StructField(‘col\_name’,StringType(),True-that means it has null values)])

df.select(col(‘column\_names’)).display() – **select the columns**

**To change the column name as alias**

df.select(col(‘column\_names’).alias(‘alias\_name’)).display()

**Filter the rows**

df.filter(col(‘col\_name’)== any condition).display()

**for more than one condition**

df.filter((col(‘col\_name’)== any condition) & (col(‘col\_name’)==any condition)).display()

**to check data in our table**

df.filter(col(‘col\_name’).isin(‘values,values’)).display()

df.filter(col(‘col\_name’).isNull()).display()-**fetch the null value records**

**To rename the column in our schema**

df.withColumnRenamed(‘current’,’new’).display()

**To create a new column if it does not have. if it is it modify the column**

df.withColumn(‘col\_name’,any\_operation).display()

**To replace the field values**

df.withColumn(‘col\_name’,regexp\_replace(col(),’old\_value’,’new\_value’)).display()

\-**for move to new line**

**For TypeCast the column:**

df.withColumn(‘col\_name’,col(‘col\_name’).cast(data\_type)).display()

**Sort the field**

df.sort(col().desc()or .asc()).display()

**For sort the more than two columns**

df.sort([‘col1’,’col2’],ascending=[0,0]).display()

ascending = 0 means descending

ascending =1 means ascending

**To limit the records**

df.limit(any\_number).display()

**To drop the field**

df.drop(col1,col2).display()

**To drop the duplicates**

df.dropDuplicates().display()

**To drop the duplicates for particular field**

df.drop\_duplicates(subset=[col\_name]).display()\

df.distinct().display()

**To combine two table with same field**

df1.union(df2).display()

**To combine two table with mismatch field in correct manner**

df1.unionByName(df2).display()

**create the dataframe**

data=[(),()]

schema=’id string’

df=spark.createDataFrame(data,schema)

**String function**

df.select(initcap(col\_name)).display()

df.select(lower(col\_name)).display()

**Date\_Fucntions**

df.withColumn(col\_name,curr\_date())

df.withColumn(col\_name,date\_add(date\_col,7))-**to add seven days similar for date\_sub()**

datediff(End,Start)-**difference bw two dates**

df.withColumn(col\_name,date\_format(date\_col,’dd-MM-yyyy’))-**To change the format**

**To dropnulls**

Df.dropna(‘all’)- **the record which all the null that will be removed**

Df.dropna(‘any’)- **the record which have any null value will be removed**

df.dropna(subset=[col\_name])-**it removes the null record by search the field**

**To fill Null values**

df.fillna(‘any\_value’).display()-**it fill the null value**

df.fillna(‘any\_value’,subset=[col\_name]).display()-**it fill the null value for the particular field.**

**Split:**

df.withColumn(col\_name,split(col\_name,delimeter like ‘ ‘ , )).display()

**Indexing:**

df.withColumn(col\_name,split(col\_name,delimeter like ‘ ‘ , )[1]).display()

**Explode:**

df.withColumn(col\_name,explode(col\_name)).display()

**for example in our field we have values like [val1,val2] this function generates 2 records for that 2 value.**

**Array\_contains:**

df.withColumn(col\_name,array\_contains(col\_name,search\_val)).display()

**it will return the new field with true or false**

**GroupBy:**

df.groupBy(col1,col2).agg(sum(col\_name).alias(),avg()).display()

**Collect\_list: It gives the result as list type to combine the response**

df.groupBy(col).agg(collect\_list(col\_name)).display()

**Pivot: that means give response for every field we have the field which has low,mid,high it creates the records like low,mid,high and give value**

df.groupBy(col).agg(pivot(col\_name)).agg(avg()).display()

**When-Otherwise:**

**It is similar case when in sql**

df.withColumn(col\_name,when(col(col\_name)==,’value’).otherwise(‘value’)).display()

**Joins:**

**Inner join**

df1.join(df2,df1[‘common\_field’]==df2[‘common\_field'],’inner’).display()

**Left Join**

df1.join(df2,df1[‘common\_field’]==df2[‘common\_field'],’left’).display()

**Right join**

df1.join(df2,df1[‘common\_field’]==df2[‘common\_field'],’right’).display()

**Anti Join**

**This will return the value that does not common or match in both table like df1 record only returns**

df1.join(df2,df1[‘common\_field’]==df2[‘common\_field'],’anti’).display()

**Windows Function:**

Row\_number()-**It assign unique value**

from pyspark.sql.window import window

df.withColumn(col\_name,row\_number().over(window.orderBy(col\_name))).display()

rank() – **it is used to assign rank if it ties the num skips for ex 1,1,1,1,5,5,5,5 like**

df.withColumn(col\_name,rank().over(window.orderBy(col\_name))).display()

dense\_rank() - **it is used to assign rank if it ties it doesn’t skips for ex 1,1,1,1,2,2,2 like**

df.withColumn(col\_name,dense\_rank().over(window.orderBy(col\_name))).display()

**Cumulative sum:**

df.withColumn(col\_name,sum(col\_name).over(window.orderBy(col\_name))).display() – **it produces the full output does not give the cum sum**

df.withColumn(col\_name,rank().over(window.orderBy(col\_name).rowsBetween(window.unboundedPreceding,window.currentRow))).display()

unboundedPreceeding-starts at the first row

unboundedFollowing-starts at the last row

**User defined function:**

**Create function using def:**

**Then assign it to variable**

Var\_name=udf(func\_name)

df.withColumn(col\_name,var\_name(col\_name))

**Data Writing:**

df.write.format(‘csv’).mode(‘append’).option(‘path’,path).save()

then in mode we also use overwrite , error , ignore

**parquet: The meta data will be stored**

**Delta: The meta data will not be stored**

df.write.format(‘parquet’).mode(any\_mode).saveAsTable(Table\_name).

**spark sql**

**by create the view we can perform sql**

df.createTempView(‘my\_view’)-**but it needs to be performed.**

**then choose sql and perform sql operation**

**Another method**

**We use**

Var\_name = spark.sql(“sql query”)

Var\_name.display()

**Spark – Read File**

From pyspark import SparkConf,SparkContext

Var\_name = SparkConf().setAppName(‘Read File’)

Sc=SparkContext.getOrCreate(conf = Var\_name)

Text=sc.textFile(‘path’)

Text.collect() – print the file content.

.map() – is used to map values in a list,It will create new RDD

.flatMap() – is used to return result in one list

.filter() – is used to filter the data in the list

.distinct() – is used to get the distinct elements

.groupByKey()- it should be in (key,value) in our word so we should combine the result like (key,[val1,val2,val3]) it combines the value result in form of group .mapvalues(list)-is used to give the value otherwise the reference only will be printed.

.reduceByKey(lambda x,y:x+y) – it should be in also (key,value) in our word this function combine the values in one instead of group.It performs any mathametical operations for values.Reduction the values.

.count() – it returns the count of values.

.countByValue() – it returns the count for each variable

.saveAsTextFile() – it save the results it partitioned our output.

.getNumPartitions()- give the number of partition.

.repartition()- is used to change the number of partition

.coalesce() – is used to decrease the number of partition.

String Functions:

.startswith() – to check the string with starting

.endswith() – to check the string at ending

.contais() – is used to return the matching string value.

.like() – it is similar to sql function which is for pattern matching.

.where() – is used to filter the records.

.cache() – if we perform some transformations it will simply save the transformation and when again the action is called it doesn’t start from transformation.It starts from cache() it has all transformation part.It is best for optimization.

To convert df to RDD simply var\_name = df.rdd

Pyspark – Json:

df = spark.read.format('json').option('inferSchema','true').option('multiline','true').option('header', 'true').load('/datasets/orders.json')

Explode the fields in Json:

Df1=df.select(explode(df.datasets).alias(‘name’)).

Datasets=array.

We need to convert every nested elements under root.

Df1=df1.select(‘orders.\*’)

If it still have nested elements then we can use explode function.

We can access the nested elements using . operator Ex:shipmentdetails.city

So we need to separate the nested elements using with column functions and then drop the shipments that the main heading.

For Handling Json file in Pyspark first we need to find the schema of the json elements   
then we need to explode the elements and then separate elements using withColumn function.

concat\_ws stands for **"concatenate with separator"**. It's a PySpark function used to **join multiple string values** into a single string, using a specified **separator**.

**🔧 Syntax:**

**concat\_ws(separator, column1, column2, ...)**

**✅ Example:**

**concat\_ws(",", collect\_set("amount"))**

If you have a list of amounts like [1000, 1200] and you use:

It will return:

"1000,1200"

Write a PySpark program to count the occurrences of each word in a given text file. The solution must utilize RDD transformations and actions for processing, and then convert the final RDD into a DataFrame for output. Sort DataFrame by count in descending order.

**Hint**

To convert an RDD to a DataFrame, use the spark.createDataFrame() method. Example:

df = spark.createDataFrame(rdd, schema=["word", "count"])

**Input**

The input data will be provided as a plain text file located at /datasets/notes.txt. Each line contains a sentence or phrase. Example input text:

hello world

hello PySpark

PySpark is fun

hello world again

from pyspark import SparkConf,SparkContext

from pyspark.sql import SparkSession

from pyspark.sql.types import \*

from pyspark.sql.functions import \*

from pyspark.sql.window import Window

Var\_name = SparkConf().setAppName('Text')

sc=SparkContext.getOrCreate(conf = Var\_name)

spark = SparkSession(sc)

text=sc.textFile('/datasets/notes.txt')

rdd=text.flatMap(lambda x:x.split(' '))

rdd1=rdd.map(lambda x:(x,1))

df = spark.createDataFrame(rdd1, schema=["word","count"])

df1=df.groupBy('word').agg(sum('count').alias("total\_count")).orderBy(col('total\_count').desc())

display(df1)

to\_date() function is used to convert string into date datatype.

date\_format()-is used to change the format.