# -\*- coding: utf-8 -\*-

"""PySpark

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1lJjl\_53lUs-N6yrta8lOpa20UKvjuEEi

#\*\*DAY 15\*\*

"""

pip install pyspark

import py4j as ps

import pyspark

from pyspark.sql import SparkSession

# the basics

sp= SparkSession.builder.master("local[1]") \

.appName("First.in") \

.getOrCreate()

print(sp)

# specifies where i want to create my local system and how many cores i need

# specifies the name of the file system we are creating

# creates or calls the system by that name

#creating an RDD

sc=sp.sparkContext

rdd1= sc.parallelize([1,2,3,4,5,6,7,8,9,20])

rdd2= rdd1.map(lambda x:(x,1))

#check lineage graph by toDebugstring

print(rdd1.toDebugString().decode())

print(rdd2.toDebugString().decode())

# shared variable= the stored data is present in various nodes, two types broadcast and acumulative variable

word=sc.broadcast(('bunny',1))#storing a tuple

# accessing the value

print(word.value)

#accumulator

acc=sc.accumulator(1)

acc.add(1)

acc.add(2)

print(acc.value)

acc.add(1)

print(acc.value)

#

#word.value.pop(0)

rdd2.collect()#to print the rdd

# lets create a fresh rdd to understand RDD more deeply

data=[1,2,3,4,5,6,7,8]

rdd= sp.sparkContext.parallelize(data)

#to collect no. of partitions

print(rdd.getNumPartitions())

d=[{'Name':'Dhananjay','Age':22,'Cost':1000},{'Name':'Vansh','Age':23,'Cost':20}]

rdd1= sp.sparkContext.parallelize(d)

rdd1.collect()

#to feed data from a external file

distFile = sc.textFile("/Notes.txt")

distFile.collect()

print(distFile.getNumPartitions())

#now there is another function called wholeTextFiles that gives us the key:value pair as path:data pair

df=sc.wholeTextFiles("/Notes.txt")

df.collect()

d=[{'Name':'Dhananjay','Age':22,'Cost':1000},{'Name':'Vansh','Age':23,'Cost':20}]

rdd2= sc.parallelize(d)

rdd2.collect()

#repartition

print(rdd2.getNumPartitions())

newrdd=rdd2.repartition(4)#creates a new rdd because RDDs are immutable so to repartition it has to be created anew

print("new partition count "+str(newrdd.getNumPartitions()))

# remember coalesce reduces the rdd back to original numpartitions

data1=[('bunny','panwar','M',19000.0),

('vansh','mishra','M',18000.0),

('sanya','singla','F',100000.0)]

columns1=['name','surname','gender','salary']

df=sp.createDataFrame(data=data1)

df.printSchema()

df2=rdd2.toDF()#change RDD to DF

df2.printSchema()

print(df2)

df2.collect()

"""#\*\*DAY 16\*\*

###Difference between Pandas and Pyspark is

### Pyspark has faster processing speeds as the data is in distributed systems while Pandas works on only one node so slower speeds

"""

pip install pyspark

import py4j as ps

import pyspark

from pyspark.sql import SparkSession

sp= SparkSession.builder.master("local[1]") \

.appName("Second.in") \

.getOrCreate()

print(sp)

sc=sp.sparkContext

data1=[('bunny','panwar','M',19000.0),

('vansh','mishra','M',18000.0),

('sanya','singla','F',100000.0)]

columns1=['name','surname','gender','salary']

df=sp.createDataFrame(data=data1,schema=columns1)

df.printSchema()

pandasDF=df.toPandas()#convert PySpark into Pandas Dataframe

print(pandasDF)

print(df)

df = sp.read.json("sample1.json")

print(df)

# to fix the error of corrupt record we can enable multiline reading so that is reads whole JSON files

df2 = sp.read.json("sample2.json",multiLine=True)

print(df2.printSchema())

"""### sp.read.csv returns a dataframe"""

df= sp.read.csv("Book3.csv")

df.write.parquet("zipcodes.parquet")

#writing CSV data into a parquet file

parDF1=sp.read.parquet("zipcodes.parquet")

print(parDF1.printSchema())

#reading data from a parquet file

print(df3)

# task save to avro

data=[(2012,8,"Batman",9.8),

(2012,8,"Hero",8.7),

(2012,7,"Robot",5.5),

(2011,7,"git",2.0)]

columns=["year","month","title","rating"]

df=sp.createDataFrame(data=data,schema=columns)

#df.write.format("avro").save("person.avro")

df.write.mode("overwrite").orc("data.orc")

orc=sp.read.orc("data.orc")

print(orc)

'''

file\_location='userdata1\_orc'

file\_type="orc"

infer\_schema="false"

first\_row\_is\_header="false"

delimiter=","

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

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

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

.load(file\_location)

df.show()

'''

from pyspark import SparkContext

spark=SparkSession.builder.master("local[2]")\

.appName("Try.in")\

.getOrCreate()

sc=spark.sparkContext

data=[("bunny",9000,"M",22),

("Vansh",8000,"M",22),

("Mohan",1000,"M",23)]

columns=["Name","salary","gender","age"]

df=sp.createDataFrame(schema=columns,data=data)

df.printSchema()

from pyspark.sql.types import StructType,StructField,StringType,IntegerType

#through structype we can establish the data types of columns for us

table=StructType([\

StructField("Name",StringType(),True),\

StructField("Salary",IntegerType(),True),\

StructField("Gender",StringType(),True),\

StructField("Age",IntegerType(),True),\

])

df=sp.createDataFrame(data=data,schema=table)

df.show()

#struct type narrates the schema and their datatypes which can be important for us in the long run

df.printSchema()

# we can observe that the datatypes are exactly what we set

# reading data from a csv file and making a DF out of it

data= sp.read.csv("Book3.csv")

table=StructType([\

StructField("ID",IntegerType(),True),\

StructField("File Name",StringType(),True),\

StructField("Status",IntegerType(),True)\

])

df=sp.createDataFrame(data=data,schema=table)

df.show()

"""#\*\*DAY 17\*\*"""

pip install pyspark

import py4j as ps

import pyspark

from pyspark.sql import SparkSession

from pyspark import SparkContext

sp=SparkSession.builder.master("local[1]")\

.appName("Third.in")\

.getOrCreate()

print(sp)

from pyspark import SparkContext

#

from pyspark.sql.types import StructType,StructField,StringType,IntegerType

#Nested StructType

schema= StructType([\

StructField("Full Name",StructType([

StructField("First Name",StringType(),True),\

#StructField("Middle Name",StringType,True),

StructField("Last Name",StringType(),True),\

])),\

StructField("Skill",StringType(),True),\

StructField("Date",StringType(),True),\

])

data=[

(("Amit","Pathak"), "Python", "02-07-2021"),

(("Shikhar","Mishra"), "Soft skills","07-10-2021"),

(("Shivani", "Suvarna"), "Accounting","20-08-2021"),

(("Pooja", "Jain"), "Data Science","02-12-2021"),

]

df=sp.createDataFrame(data=data,schema=schema)

"""####Arraytype"""

df.show()

from pyspark.sql.types import ArrayType

#another problem

data1=[("James,,Smith",["Java","Scala","C++"],["Spark","Java"],"OH","CA"),

("Michael,,Rose,",["Spark","Java","C++"],["Spark","Java"],"NY","NJ"),

("Robert,,William",["Csharp","VB"],["Spark","Python"],"UT","NV")]

schema=StructType([\

StructField("Full Name",StringType()),\

StructField("Languages",ArrayType(StringType()),True),

StructField("Languages at work",ArrayType(StringType()),True),

StructField("State",StringType(),True),

StructField("Prev State",StringType(),True),

])

df2=sp.createDataFrame(data=data1,schema=schema)

df2.show()

"""### To import files from cloud"""

#GDRIVE

url="https://drive.google.com/file/d/19pQiFdorIvSa21o4hd23\_wk-\_thkEJwV/view?usp=sharing"

import pandas as pd

df=sp.createDataFrame(pd.read\_csv(url))

df.show()

#From AZURE blob storage

import pandas as pd

#storing the SAS URL from Azure

source = 'https://dhananjaymk4.blob.core.windows.net/con1/Worksheet.csv?sp=r&st=2023-11-29T05:15:47Z&se=2023-11-29T13:15:47Z&sv=2022-11-02&sr=b&sig=Pij2%2FI5j9%2FlrJPXTuJ7Vr%2FEop6rtiLTcEH6zPCrvtV8%3D'

df = pd.read\_csv(source)

print(df)

spark\_df = sp.createDataFrame(df)

spark\_df.show()

#print(sp.read.csv(source))

"""##### REST API vs SOAP API

\* In Rest API we can work with JSON and XML files

\* Whereas in SOAP API we can work with only XML files

"""

# via API requests

import requests

source = 'https://dhananjaymk4.blob.core.windows.net/con1/Worksheet.csv?sp=r&st=2023-11-29T05:15:47Z&se=2023-11-29T13:15:47Z&sv=2022-11-02&sr=b&sig=Pij2%2FI5j9%2FlrJPXTuJ7Vr%2FEop6rtiLTcEH6zPCrvtV8%3D'

response= requests.get(source)

with open("sample.csv","wb") as file:

file.write(response.content)

df=sp.read.csv("sample.csv",header=True,inferSchema=True)

df.show()

"""###Select statement for DF and other statements for various operations"""

#Explode Functionality

from pyspark.sql.functions import explode

data1=[("James,,Smith",["Java","Scala","C++"],["Spark","Java"],"OH","CA"),

("Michael,,Rose,",["Spark","Java","C++"],["Spark","Java"],"NY","NJ"),

("Robert,,William",["Csharp","VB"],["Spark","Python"],"UT","NV")]

schema=StructType([\

StructField("Name",StringType()),\

StructField("Languages",ArrayType(StringType()),True),

StructField("Languages at work",ArrayType(StringType()),True),

StructField("State",StringType(),True),

StructField("Prev\_State",StringType(),True),

])

df3=sp.createDataFrame(schema=schema,data=data1)

df3.select(df3.Name,explode(df3.Languages)).show()

# to store firstnames

name=[]

for j in range(0,3):

name.append(data1[j][0].split(",")[0])

print(name)

# there is a split function for pyspark too

from pyspark.sql.functions import split

df3.select(split(df3.Name,",").alias("Name")).show()

#to combine two columns and show them together

from pyspark.sql.functions import concat

df3.select(concat(df3.State,df3.Prev\_State).alias("Full\_State")).show()

import pyspark.sql.functions

#Run a loop in the dataframe to get the first names

for i in range(0,3):

print(df3.collect()[i][0])

"""### Therefore to traverse through a dataframe

####Syntax: dataframe.collect()[row\_index][column\_index]

where, row\_index is the row number and column\_index is the column number

"""

# Check a column for a certain string and print

# Pyspark uses contains and array\_contains for different purposes

# while contains compares values for colummns ,array\_contains on the other side compares column element values and variables and if the element contains it or not

from pyspark.sql.functions import contains

df3.select(contains(df3.Name,df3.State)).show()

# Check a column for a certain string and print

from pyspark.sql.functions import array\_contains

df3.select(array\_contains(df3.Languages,"Java")).show()

#creating a new df for

data2=[('James',{'hair':'black'}),('Michael',{'hair':'brown'}),('Robert',{'hair':'red'}),('Washington',{'hair':'grey'}),('Jefferson',{'hair':'brown'})]

from pyspark.sql.types import StringType,MapType,StructType,StructField

schema1=StructType([\

StructField("Name",StringType(),True),\

StructField("properties",MapType(StringType(),StringType()),True),\

])

df4=sp.createDataFrame(schema=schema1,data=data2)

df4.collect()

from pyspark.sql.functions import map\_values,map\_keys

df4.select(df4.Name,map\_values(df4.properties)).show()

df4.select(df4.Name,map\_keys(df4.properties)).show()

df4.select(df4.Name,(df4.properties['hair']).alias("Hair\_Colour")).show()

"""###Maps are usually used for transformations and its better to use map in RDDs and then convert them back into DF for our uses

"""

#ndf=df.rdd.map(lambda x:)

"""###Row Object"""

#rows are an important part of the PySpark work , they are like tuples which can be retrieved and also access elements with their name like dictionary key value pair

from pyspark.sql import Row

r1=Row("Bunny",8,4,1)

print(r1[1])

#now to demo how to use column names to retrieve values

r2=Row(name="Vansh",age=22)

print(r2.age)

#we can create a custom class using Row

Human=Row('name','age')

h1=Human("Bunny",22)

h2=Human("Dhananjay",22)

print(h1.name)

#RDD using row object

r=Row('name','age','DOB')

data=[r("Dhananjay",22,841),r("Vansh",22,420),r("Sanya",22,1551)]

rdd= sp.sparkContext.parallelize(data)

rdd.collect()

#DF using Row object

df=sp.createDataFrame(schema=r,data=data)

df.show()

#create a list of new column names and pass it to change the name of all the columns in one go

col=["Name","Age","DateOfBirth"]

test=sp.createDataFrame(data=data).toDF(\*col)

test.show()

#a new DF to store certain columns of the DF

dfshow=test.select(test.Name,test.Age)

dfshow.show()

#concept of COL

from pyspark.sql.functions import col

df.select(col("name"),col("age")).show()

import requests

url='http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'

response= requests.get(url)

with open("IRIS.csv","wb") as file:

file.write(response.content)

df=sp.read.csv("IRIS.csv",header=True,inferSchema=True)

df.show()

"""###Get a file without downloading it using \*\*SparkFiles\*\*"""

from pyspark import SparkFiles

sp.sparkContext.addFile(url)

df=sp.read.csv(SparkFiles.get("iris.data"), header=True)

print(type(df))

df.show()

"""#\*\*DAY 18\*\*"""

pip install pyspark

import pyspark

from pyspark import SparkFiles

from pyspark.sql import SparkSession

from pyspark import SparkContext

sp=SparkSession.builder.master("local[1]")\

.appName("Fourth.in")\

.getOrCreate()

print(sp)

"""###SparkFiles"""

#SparkFiles are an important feature as they upload the files in Spark environment and accessible by all the

#nodes in the system. The files can be of any type

#Task 1

from pyspark.sql import Row

r=Row('id','address','state')

data=[r(1,"St. Francis Ave","CA")

,r(2,"St. Francis Rd","DE")

,r(3,"Main Rd","NY")]

df=sp.createDataFrame(data=data,schema=r)

df.show()

from pyspark.sql.functions import regexp\_replace

# how to replace values in DataFrame Columns

df.withColumn('address1',regexp\_replace('address','Rd','Road')).show()

#withColumn can do things 1)Add a new column 2)update the value

df.withColumn('address1',regexp\_replace('address','Rd','Road')).show()# added a new column with the same values as

#lit and filter

df.filter((df.id==3)).show()

df.filter((df.state!="NY")).show()

#df.filter(df.state.isin(li)).show()

df.filter(df.address.like("%A","E%")).show()

'''values=df.select(df.address).show()

li=[for value in values]

print(li)'''

df.select(df.columns[:2])

from pyspark.sql.types import StructType,StructField,StringType,IntegerType

#TASK

data = [("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)

]

col=["Name","Dept","State","Salary","Age","Bonus"]

df=sp.createDataFrame(data=data).toDF(\*col)

df.show()

from pyspark.sql.functions import sum

li=df.groupBy("Dept").sum('Salary').collect()

print(li)

name=["Dept","total salary"]

df2=sp.createDataFrame(data=li).toDF(\*name)

df2.show()

#problem with GroupBy is that it only shows 2 columns

#from pyspark.sql.functions import join

data3=df.join(df2,[df.Dept==df2.Dept],'outer').show()

data3=df.join(df2,[df.Dept==df2.Dept],'outer').collect()

#A functionality called Window is used to "partition" the rows into categories as per requirement

from pyspark.sql.window import Window

part=0

df.withColumn('sumsalary',sum("Salary")).over(Window.partitionBy("Dept")).show()

#more than 3 lakh

#salary dept wise whose salary than 80k

cols=["Name","Dept","State","Salary","Age","Bonus","departement","total"]

df3=sp.createDataFrame(data=data3).toDF(\*cols)

df3.show()

df3.groupBy('Dept').agg(sum('Salary')).where(sum("Salary")>300000).show()

df3.where(col('Salary')>80000).groupBy('Dept').agg(sum('Salary')).show()

print(data3)

"""###Dataset:Iris

#### Find the greatest difference between sepal and petal length from the dataset and return the row which has the greatest difference

"""

from pyspark import SparkFiles

url='http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'

sp.sparkContext.addFile(url)

df=sp.read.csv(SparkFiles.get("iris.data"), header=True)

print(type(df))

df.show()

col=["sepal\_length","sepal\_width","petal\_length","petal\_width","class"]

df2=sp.createDataFrame(df.collect(),col)

df2.show()

from pyspark.sql.functions import try\_subtract

df2.withColumn("Diff",try\_subtract(df2.sepal\_length,df2.petal\_length)).show()#added a column with a value

dt=df2.withColumn("Diff",try\_subtract(df2.sepal\_length,df2.petal\_length)).collect()

df3=sp.createDataFrame(dt)

df3.show()

df3.select(df3.sepal\_length,df3.petal\_length,df3.Diff).show()

print(df3.agg({"Diff": "max"}).collect()[0][0])

df3.groupby('class').max("Diff").show()

#dfsimple.groupby("Department").max('Salary').show()

"""#### to get the greatest length between petal or sepal length"""

from pyspark.sql.functions import greatest

df3.select(greatest(df3.sepal\_length,df3.petal\_length)).show()

"""##Integrating SQL and PySpark

####capability to write sql queries in dataframes by creating tables out of them

"""

df3.createOrReplaceTempView("Table")#creating a table out of DF

sp.sql('select \* from Table where Diff=4.6').show()

sp.sql('select Diff from Table order by Diff DESC ').show()

#sp.sql('select distinct(class,Diff) from Table order by Diff DESC').show()

"""#####Cannot create tables in Spark.sql as it is a sql analytics tool rather than a RDBMS so it cant store a table

##\*\*DAY 19\*\*

"""

pip install pyspark

import pyspark

from pyspark.sql import SparkSession

sp=SparkSession.builder.master("local[1]")\

.appName("Fifth.in")\

.getOrCreate()

data=[("bunny",22,"TC",2000),("vansh",22,"TC",2000)]

col=["Name","Age","Dept","salary"]

df=sp.createDataFrame(data=data).toDF(\*col)

df.show()

df.withColumnRenamed("salary","Monthly\_Salary").show()

#TO rename one column

df.withColumnRenamed("salary","Monthly\_Salary")\

.withColumnRenamed("Age","Persons\_Age")\

.withColumnRenamed("Dept","Department").show()

#Multiline renaming

data=[("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)]

col=["Name","Dept","State","Salary","Age","Bonus"]

df2=sp.createDataFrame(data=data).toDF(\*col)

from pyspark.sql.functions import when

df2.show()

df2.withColumn("Increment",when(df2.Salary>=90000,"yes").otherwise("no")).show()

df2.createOrReplaceTempView("Table")

sp.sql("select \* from Table").show()

query='''select \* ,

case

when Salary>=90000 then 'yes'

when salary<90000 then 'no'

end as Increment

from Table '''

sp.sql(query).show()

"""##Moving into data cleaning"""

#Removing duplicates

df3=df2.dropDuplicates(['Dept'])

df2.show()

df3.show()

print(df2.count())

print(df3.count())

(df2.distinct()).show()

#shows all the rows because all the rows are distinct

"""##CASE STUDY: Housing Dataset"""

df=sp.read.csv("Housing Dataset.csv",inferSchema=True,header=True)

df.show()

from pyspark.sql.functions import min

df.agg(min('population')).show()

df.createOrReplaceTempView("House")

query="""

select longitude,latitude,population from House where population=3

"""

sp.sql(query).show()

query="""

select population,latitude,longitude from House where population=(select min(population) from House)

"""

sp.sql(query).show()

query='''

select distinct(population) from House order by population asc limit 1

'''

sp.sql(query).show()

"""##Sort and order in pyspark"""

data=[("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)]

col=["Name","Dept","State","Salary","Age","Bonus"]

df4=sp.createDataFrame(data=data).toDF(\*col)

df4.sort(["Salary"],ascending = [True]).show()

df4.union(df2).show()

#to rename columns

df.show()

df.withColumnRenamed("longitude","Longitude").withColumnRenamed("latitude","Latitude").show()

df4.unionByName(df,allowMissingColumns=True).show()

#Join

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) \

]

ecol=["Id","name","supid","year","EMPID","gender","salary"]

edf=sp.createDataFrame(data=emp).toDF(\*ecol)

dept = [("Finance",10), \

("Marketing",20), \

("Sales",30), \

("IT",40) \

]

dcol=["Department","EMPID"]

ddf=sp.createDataFrame(data=dept).toDF(\*dcol)

edf.show()

ddf.show()

# change the EMPID in ddf to string using withColumn

#generally used for creating a new column out of an existing one but this was it recreates the same column with different datatype

ddf.withColumn("EMPID",ddf.EMPID.cast('string'))

ddf.show()

edf.join(ddf,(edf.EMPID==ddf.EMPID),'outer').select(['name','year',edf.EMPID,'salary','gender','Department']).show()