SPARK PRACTICE:-BASIC

## Create spark session

→ # from pyspark.sql import SparkSession

# spark=SparkSession.builder.appName('firsttry').getOrCreate()

# spark

1) Read a dataset – basic way

→ spark.read.csv('C:/spark practice/csv/cs.csv')

2) Read dataset with column name.

→ spark.read.option('header', 'true').csv('C:/spark practice/csv/cs.csv')

3) Save dataset in a variable

→ df_pyspark=spark.read.option('header', 'true').csv('C:/spark practice/csv/cs.csv')

#NOTE:- Instead of 'df_pyspark' you can use any variable name such as: df, abc, z, xyz, etc

4) Check schema

→ df_pyspark.printSchema()

#NOTE: It shows all the data type 'string' as default, to show orignal we have to set 'inferSchema=True' in the query as below in 5)

5) Check schema with orignal datatype

→ df_pyspark=spark.read.option('header', 'true').csv('C:/spark practice/csv/cs.csv', inferSchema=True)

→ df_pyspark.printSchema()

6) Alternative way to set header and inferSchema

→ df_pyspark=spark.read.csv('C:/spark practice/csv/cs.csv', header=True, inferSchema=True)

→ df_pyspark.show()

7) Check type of variable

→ type(df_pyspark)

8) Show column name

→ df_pyspark.columns

9) List top 2 data

→ df_pyspark.head(2)

10) List data of specific column 'name' in table form.

→ df_pyspark.select('name').show()

11) List data of multiple columns in table form.

→ df_pyspark.select(['name', 'age']).show()

12) Check datatype

→ df_pyspark.dtypes

13) Describe the dataframe

→ df_pyspark.describe().show()

14) Add column in dataframe (only for display, doesn't change the original dataset)

→ df_pyspark.withColumn('age after 5 year', df_pyspark[' age']+5)

15) Add column in dataframe and show (only display)

→ df_pyspark.withColumn('age after 5 year', df_pyspark[' age']+5).show()

#NOTE: here it doesn't require to run 'df_pyspark.show()' as .show() is already mentioned in code.

16) Add column in orignal dataset

→ df_pyspark=df_pyspark.withColumn('age after 5 year', df_pyspark[' age']+5)

→ df.pyspark.show()

17) Delete column

→ df_pyspark=df_pyspark.drop('age after 5 year')

18) Rename column

→ df_pyspark=df_pyspark.withColumnRenamed('name','names')

19) Delete row having null value

→ df_pyspark=df_pyspark.na.drop()

20) Delete row if all values are null

→df_pyspark=df_pyspark.na.drop(how="all")

21) Delete row if 3 column are null (#totsl number of column in 10)

→ df_pyspark=df_pyspark.na.drop(how="any",thresh=7)

22) Delete record if there is null value in specific column

→ df_pyspark=df_pyspark.na.drop(how="any",subset=['email'])

→ df=df.na.drop(how='any',subset=['age','gender','address'])

23) Fill 'not available' in null value ---only works for string data--

→ df_pyspark=df_pyspark.na.fill('not available')

#NOTE: if integer then fill(0)

24) Fill 'not available' in null value of specific column

→ df_pyspark=df_pyspark.na.fill('not available', 'address')

25) Fill 'not available' in null value of multiple column

→ df_pyspark=df_pyspark.na.fill('not available', ['address','email'])

26) Change datatype of "salary" column to double

→ df_pyspark=df_pyspark.withColumn("salary", col("salary").cast("double"))

#note: for this, col should be imported from pyspark.sql.functions

26) #imputer# fill the missing value on the basis of mean or median

```
→
from pyspark.ml.feature import Imputer
imputer=Imputer(
inputCols=['age'],
outputCols=["{}_imputed".format(c) for c in['age']]
).setStrategy("mean")

→ imputer.fit(df_test).transform(df_test).show()
```

27) #FILTER# list the records of employee of age above 30

```
→ df_test.filter("age>40")
```

--show with specific columns--

```
→ df_test.filer("age>40").select(['name','age']).show()
```

---alternatives---

```
→ df_test.filter((df_test['age']>35) & (df_test['gender']=='m')).show()
```

#NOTE: in most of the case I found the use of "" is similar to '' . for OR operation use |

#NOTE for inverse operation (not operation), we can use ~

```
→ df_test.filter(~(df_test['age']>35) & (df_test['gender']=='m')).show()
```

28) #GROUP BY: list maximum age of employee in each department

```
→ df_test.groupBy('department').max().show()
```

29) List the group by salary

```
→ df_test.groupBy('department').sum('salary').show()
```

30) Count the number of employee in each department

```
→ df_test.groupBy('department').count().show()
```

31) Find the total salary spend (use of aggregate function)

→ df_test.agg({'salary':'sum'}).show()

32) List average salary of male and female

→ df_test.groupby('gender').max().show()

33) Filter

→ df_test.filter("salary>30000").show()

→ df_test.filter("salary>30000").select(['name','department','salary']).show()

#34) Export csv file

→ df_test.write.csv('C:/spark practice/Output', header=True, mode='overwrite')

35) Gather the dataframe in single partition

→ df_test=df_test.coalesce(1)

##NOTE: Using panda, we can read data in the following way:

# Read a csv file.

--> # pip install pyspark

# import pyspark

# import pandas as pd

# pd.read_csv('C:/spark practice/csv/cs.csv')