9|2|24

Creating dataframes (back side continuation)

* Parquet
* orc
* avro

} are file formats.

(ii) from External file

(d) spark.read.format()

```
spark.read.format("text").load("output".txt")
```

⇓

Converts textfile into dataframe

⇓

when it runs on pyspark notebook it stores as RDD.

```
from pyspark.sql import SparkSession
spark = SparkSession..builder.getOrCreate()
df = spark.read.format(" text").load(" output.txt")
df.selectExpr("split(value, '') as Text_Data").show
                                              (4, False)
```

Adding columns to dataframe

1) Adding column with constant value

lit(value) ⇒ literal value

```
df.withColumn("column_name", lit(value))
```

Here, value is constant value.

import lit :- from pyspark.sql.functions import lit

Ex:- df.withColumn(" Salary", lit (30000))

2) Add column based on another column of df

i.e., existing column

(i) withColumn() method

```
df.withColumn("Column_name", df.existing_column)
```

Ex :- df. withColumn("Salary", df. Salary + 20000)
New Salary

(ii) using concat_ws
↳ from pyspark.sql.functions import concat_ws

* Concat two existing columns

df. withColumn("col_name", concat_ws("Separator", "existing_col1", "existing_column2")). show()

Ex:- df. withColumn("Details", concat_ws("-", "Company", "Sal

3) Add column when not exists in dataframe

if 'column_name' not in df. columns:
df. withColumn("column name", lit(value))

Ex:- if 'Age' not in df. columns:
df. withColumn("Age", lit(30))

Group by and aggregate function :-

groups the identical data.

groupBy :- groups the identical data.

df. groupBy("Departments"). sum("Salary"). show()

(or)

collect the identical data into groups on Dataframe and perform count, sum, min, max and avg functions on grouped data.

1) sum() :- df. groupBy("Departments"). sum("Salary"). show()

2) min()

3) max()

4) avg()

5) mean()

6) count() - df.groupBy("Departments").count().show(

7) agg() → used when we want to perform multiple aggregations at a time.

Ex:- df.groupBy('Departments'). agg(sum(df.Salary) .alias("Salary_sum"), min(df.Salary).alias("min_Salary")).show()

8) pivot()

* rotate the data from one column into multiple columns (transpose row to column).

* It is an aggregation where one of the grouping column values is transposed into individuals columns with distinct data.

Ex:- df.groupBy("Departments").pivot("Name"). sum("salary").show()

orderBy() and sort():-

Sort() → Sort a dataframe using one or more columns

* by default sorts in ascending order.

1) Sort based on single column

df.sort("Salary").show()

2)

2) Sorted using desc()

```
df.sort(df.Salary.desc()).show()
```

3) sort based on first column and then by second column.

```
df.sort("Salary", "Name").show()
```

If salary values are same, it will sort based on name.

orderBy() => alternative to sort

Replace sort() with orderBy() in above examples.

Missing values

1) dropping rows based on null values

```
df.na.drop().show()
```

2) If all values in rows are null then drop otherwi

default is any

```
df.na.drop(how = "all").show()
```

3) thresh = 2 (atleast 2 non null values should be present)

```
df.na.drop(how = "any", thresh = 2).show()
```

4) only in the particular, null values will be deleted

using subset

```
df.na.drop(how="any", subset=["salary"]).
                                          show()
```

Joins

Combine two dataframes

1) inner   2) left outer   3) right outer   4) Left anti

5) left semi   6) cross   7) self join

1) Inner join :-

join records when key columns are matched, and dropped when they are not matched.

2) outer join :-

Returns all rows from both datasets, where join join expression doesn't match it returns null values

3) left join | left outer join :-

returns all rows from left dataset irrespective of match found on right dataset, when join doesn't match it assigns null for that record.

4) Right " | Right " " :-

viceversa of left join.

5) left semi join :-

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

6) left anti join :-

returns only columns from left dataset for non-matched records.

1) Inner join

empDF . join (deptDF , empDF . emp_dept_id == deptDF . dept_id,

"inner") . show()

2) Full outer join

empDF . join (deptDF, empDF . emp_dept_id == deptDF . dept_id

"outer") . show()

(or) "fullouter" (or) "full"

~~left~~ 3) left outer join "left" (or) "leftouter"

4) Right outer join "right" (or) "rightouter"

5) left semi join "leftsemi"

6) left anti join "leftanti"