31:40

NOTE CONTENTS :-

- Dropping columns
- Dropping Rows
- Various parameters in Dropping function
- Handling missing values by Mean, Median & Mode

Dropping Columns :-

- drop() method is used to drop columns.

```
df_spark.drop('Name')
```

- In the table used in previous example, suppose a column named 'Salary' is added, of 'int' type. This table will be used in following examples...

Dropping Rows wrt Null Values :-

- Here, we will see, how to drop records, that has any null value in them :-

For temp viewing :-

```
df_spark.na.drop().show()
```

function to drop null value containing records.

storing for later use :-

```
df2_spark = df_spark.na.drop()
```

- na.drop() method is used for dropping 'null' value containing records.

- The 'na.drop()' method takes 3 keyword arguments (optional), that can be used to customise which records should be dropped. The 3 keyword arguments are :-
  i) how , ii) thresh , iii) subset.

i) how :

For this keyword argument, there are 2 values possible :-

a) how = "any" : If this is set, then only those rows will be dropped, which has atleast 1 'null' value. [It is default value]

```
df_spark.na.drop(how = "any")
```

	Name	Age	Exp
(I)	Sunny	null	6
(II)	Sudhan	null	null
(III)	null	null	null

All the columns will be dropped in "any" option.

b) how = "all" : If this is set, then only those records will be dropped, which has all the values as "null"

```
df_spark.na.drop(how = "all")
```

	Name	Age	Exp
(I)	Sunny	null	6
(II)	Sudhan	null	null
(III)	null	null	null

Only row (III) will be dropped as all the values in it are "null"

38:05

ii) thresh :

⇒ In thresh, we pass an integer, say (n).
⇒ When set, it will keep all the records, that contains at-least (n) non-null values.
Records with lesser the (n) non-null values (true data values) will get deleted.

```
df_spark.na.drop(thresh=2)
```

	Name	age	Exp.	Salary
(I)	Mahesh	null	null	40000
(II)	null	34	10	38000
(III)	null	36	null	null

(I) has 2 non-null values
(II) has 3 non-null values
(III) has 1 non-null value.
As thresh is set to 2 above, thus, only record (III) will be deleted, as it has 1 (<2) non-null/true value.

iii) Subset :

40:20

⇒ In this, we pass a list of column name(s).
⇒ The records containing NaN/null values in the column(s) (passed as list), will get deleted.

`df_spark.na.drop(subset=['Experienced'])`

	Name	Age	Experience
①	Sunny	null	6
②	Sudhan	null	null
③	null	null	null

Row ② & ③ will be deleted, as in those rows, Experience column contains null value.

Filling the Missing Values :-

⇒ For filling missing/null values, `na.fill()` method is used, of the SparkSession class.

⇒ Returns the modified dataframe.

Syntax :-

`df_spark.na.fill(value, subset=None)`

compulsory param — optional param.

Parameters :-

i) value : The data (of any type – int, string, ...) passed here, will be used / replaced at all the places where null values appear.

ii) subset : Takes a single column name as string, or multiple column names as list of strings.

subset is

⇒ When passed, null values of only those columns will be replaced.

By default, null values of all rows are replaced

Ex: `df_spark.na.fill("Test",['Experience','age'])`

Name	Age	Experience
Sunny	null	6
Sudhan	null	null
null	null	null

→

Name	Age	Experience
Sunny	Test	6
Sudhan	Test	Test
null	Test	Test

⇒ Filling Missing Values based on Mean, Median, Mode of column

⇒ for filling missing values with mean/median values, we make use of Imputer Function

⇒ In PySpark, we make an Imputer function suitable for our dataframe, as follow:—

```
from pyspark.ml.feature import Imputer

imputer = Imputer(
    inputCols=['age', 'Experience', 'Salary'],
    outputCols=["{}_imputed".format(c) for c in ['age', 'Experience', 'Salary']]
    ).setStrategy("mean")
```

name of the columns in which we want to fill the null values with mean, median, mode

on what basis we want to fill the null values: 'mean', 'median', or 'mode'

those inputCols columns only should be written

• Now, we will add these imputation columns to our dataframe, using the imputation function imputer (that we made) as follow :-

```
imputer.fit(df_spark).transform(df_spark).show()
```

- imputer: the imputation function that we made
- df_spark: dataframe, to which we are applying the mean
- show(): used to display the rest. We can store the result for further usage

⇒ The resulting dataframe will have both the initial null-value containing columns, and the final output columns.
We can set/change the names of output columns, in the above function.