NOTE CONTENTS :-

- Filter Operation
- &, |, ==, ~

- Filter operations are very important for data preprocessing techniques.
- When we want to retrieve some data based upon some conditions (boolean), we use filter operations.

This is the sample table, on which we will perform filter op :-

Name	age	Experience	Salary
Krish	31	10	30000
Sudhanshu	30	8	25000
Sunny	29	4	20000
Paul	24	3	20000
Harsha	21	1	15000
Shubham	23	2	18000

Filter Operations :-

- **filter()** method is used to filter records acc. to criteria. It does the same work as of WHERE clause in SQL.
- As the argument of filter(), we pass a string,

which contains the condition for filtration.

Ex:-

To fetch the people with Salary less than equal to 20000:-

must exactly match with column name

```
df_spark.filter("Salary<=20000").show()
```

filter condition with relational operator passed as a string (within quotes)

⇒ Alternatively:- we can also get same result, by writing the code as below:-

```
df_spark.filter(df_spark["Salary"]<=20000).show()
```

we can refer to column like this

In this case, the filter condition will not be as a string, as we are accessing the column using dataframe name, in the argument.

⇒ We can combine it with select() method, to display only the specific columns.
Say, we want to only view "Name" & "Experience" of those, who have salaries less than equal to 20000:-

```
df_spark.filter("Salary<20000").select(["Name","Experience"]).show()
```

⇒ Conditions can be combined using logical operators:-
i) & (AND) , ii) | (OR) , iii) ~ (NOT) , iv) ==

Example:-

logical operator

```
df_pyspark.filter((df_pyspark['Salary']<=20000) &
                  (df_pyspark['Salary']>=15000)).show()
```

Individual conditions must be within parenthesis, else will be an error

- NOT (~) operator can be used to reverse a condition's truth value, as follows:-

```
df_spark.filter(~(df_spark["Salary"]<=20000)).show()
```

All the records with Salary greater than 20000 will be fetched.