**Pyspark:-**

PySpark is the Python API for **Apache Spark**, a **fast and general-purpose cluster computing system**. Spark provides **in-memory computing capabilities** to **speed up** the processing of **large datasets**. PySpark allows developers to write Spark applications using Python, providing an easy-to-use interface for distributed computing tasks.

*Here's an overview of some key functions and attributes commonly used for data manipulation in PySpark*

***Import pyspark***

* *from pyspark.sql import SparkSession*

***CreateSpark Session***

* *spark = SparkSession.builder.appName("Session01").getOrCreate()*

***DataFrame Operations****:*

* ***spark.read***
* ***createDataFrame():*** *Creates dataframe*
* ***select()****: Select a subset of columns.*
* ***withColumn()****: Adds a new column or replaces an existing one.*
* ***withColumnRenamed()****: Renames a column.*
* ***drop()****: Drops specified columns.*
* ***dropna():*** *drop rows containing missing or null values in a DataFrame.*
* ***na.drop()****: Drops rows with missing values.*
* ***fillna()****: Fills missing values in the DataFrame.*
* ***na.fill()****: Fills missing values in the DataFrame.*
* ***filter()****: Filters rows using a given condition.*
* ***groupBy()****: Groups the DataFrame using the specified columns.*
* ***orderBy()*** *or* ***sort()****: Orders the DataFrame based on the specified column(s).*
* ***agg()****: Performs aggregation on specified columns.*
* ***join()****: Joins two DataFrames based on a join expression.*
* ***distinct()****: Returns distinct rows in the DataFrame.*
* ***dropDuplicates()****: Drops duplicate rows from the DataFrame.*
* ***pivot()****: Pivots a DataFrame by rotating the values of a column into multiple columns.*
* ***explode()****: Explodes a column that contains arrays into multiple rows.*
* ***sample()****: Samples a fraction of rows from the DataFrame.*
* ***randomSplit()****: Splits the DataFrame into multiple DataFrames randomly.*
* ***union()****: Concatenates two DataFrames.*

***Column Functions****:*

* *Arithmetic operations:* ***+****,* ***-****,* ***\*****,* ***/****,* ***\*\**** *(power)*
* *Comparison operations:* ***==****,* ***!=****,* ***<****,* ***<=****,* ***>****,* ***>=***
* *Logical operations:* ***&*** *(and),* ***|*** *(or),* ***~*** *(not)*
* *Mathematical functions:* ***abs()****,* ***sqrt()****,* ***exp()****,* ***log()****,* ***sin()****,* ***cos()****,* ***tan()****, etc.*
* *Aggregate functions:* ***avg()****,* ***count()****,* ***sum()****,* ***min()****,* ***max()****, etc.*
* *String functions:* ***substring()****,* ***trim()****,* ***lower()****,* ***upper()****,* ***regexp\_replace()****,* ***split()****, etc.*
* *Date functions:* ***year()****,* ***month()****,* ***dayofmonth()****,* ***dayofweek()****,* ***dayofyear()****,* ***hour()****,* ***minute()****,* ***second()****, etc.*
* *User-defined functions (UDFs) using* ***udf()*** *or* ***pandas\_udf()****: Allows you to define custom functions.*

***DataFrame Attributes****:*

* ***columns****: Returns the column names of the DataFrame.*
* ***dtypes****: Returns the data types of the columns.*
* ***schema****: Returns the schema of the DataFrame.*
* ***count()****: Returns the number of rows in the DataFrame.*
* ***show()****: Displays the first n rows of the DataFrame.*
* ***printSchema()****: Prints the schema of the DataFrame.*
* ***describe()****: Computes basic statistics for numeric and string columns.*
* ***head()****: Returns the first n rows as a list.*
* ***collect()****: Returns all rows as a list.*
* ***take()****: Returns the first n rows as a list.*

*These are just some of the many functions and attributes available in PySpark for data manipulation. You can refer to the official PySpark documentation for more detailed information on each function and attribute.*

***Different types of joins in pyspark:***

*In PySpark, there are several types of joins that you can perform on DataFrames. Here's a list of the different types of joins available:*

1. ***Inner Join****: Returns only the rows that have matching values in both DataFrames.*
2. ***Left Outer Join (Left Join)****: Returns all the rows from the left DataFrame and the matched rows from the right DataFrame. If there's no match, it returns* ***null*** *for the right DataFrame columns.*
3. ***Right Outer Join (Right Join)****: Returns all the rows from the right DataFrame and the matched rows from the left DataFrame. If there's no match, it returns* ***null*** *for the left DataFrame columns.*
4. ***Full Outer Join (Full Join)****: Returns all the rows from both DataFrames, combining the results of both left and right outer joins.*
5. ***Left Semi Join****: Returns all the rows from the left DataFrame where there's a match with the right DataFrame. It doesn't include any columns from the right DataFrame.*
6. ***Left Anti Join****: Returns all the rows from the left DataFrame where there's no match with the right DataFrame. It doesn't include any columns from the right DataFrame.*
7. ***Cross Join (Cartesian Join)****: Returns the Cartesian product of the two DataFrames, resulting in all possible combinations of rows.*

*These are the main types of joins available in PySpark, each serving different purposes and useful in different scenarios. Depending on your use case, you can choose the appropriate type of join to manipulate and analyze your data efficiently.*

# Import pyspark

from pyspark.sql import SparkSession

# CreateSpark Session

spark = SparkSession.builder.appName("Session01").getOrCreate()

# Read CSV file into a DataFrame

filePath = "/FileStore/tables/file.csv"

df = spark.read.csv(filePath, header=True, inferSchema=True)

# Show the first few rows of the DataFrame

df.show()

#Print only 1st 4 Rows

df.head(4)

# Know the DataType

type(df)

pyspark.sql.dataframe.DataFrame

# Know the DataType

df.dtypes

[('Airline', 'string'),

('Date\_of\_Journey', 'string'),

('Dep\_Time', 'timestamp'),

('Arrival\_Time', 'timestamp'),

('Dept\_Hour', 'int'),

('Dept\_Minute', 'int'),

('Arrival\_Hour', 'int'),

('Arrival\_Minute', 'int')]

# printSchema

df.printSchema()

|-- Airline: string (nullable = true)

|-- Date\_of\_Journey : string (nullable = true)

|-- Dep\_Time : timestamp (nullable = true)

|-- Arrival\_Time: timestamp (nullable = true)

|-- Dept\_Hour : integer (nullable = true)

|-- Dept\_Minute: integer (nullable = true)

|-- Arrival\_Hour: integer (nullable = true)

|-- Arrival\_Minute: integer (nullable = true)

#print columns

df.columns

# print column/columns

df.select("Airline").show()

df.select(["Airline","Date\_of\_Journey"]).show()

# Describe dataframe

df.describe().show()

# Add New Columns

df1 = df.withColumn("Arrival\_MinuteNew",df['Arrival\_Minute']+10)

# Drop A Column/Columns

df.drop('Arrival\_MinuteNew').show()

df.drop('Arrival\_Time','Dept\_Minute').show()

# Rename the column

df.withColumnRenamed('Airline',"Flight").show()

# Drop Na

df.na.drop(how='any',thresh=1).show()

# drop na from specific column

df.na.drop(subset=['Dept\_Hour']).show()

# fill na with specific value

df.na.fill(10).show()

# Fill na with some value

from pyspark.sql.functions import mean

meanVlaue = df.select(mean(df['Dept\_Hour'])).collect()[0][0]

df.na.fill(meanVlaue).show()

# Fliter operation

df.filter(df["Dept\_Hour"]<=9).show()

# Filter and print only 'Dep\_Time','Arrival\_Time'

df.filter(df["Dept\_Hour"]<=9).select('Dep\_Time','Arrival\_Time').show()

# filter with two conditions

df.filter((df['Dept\_Hour']<=12) | (df['Arrival\_Hour']<=5)).show()

## Group By and Aggregate Function

df.groupBy('Airline').count().show()  
df.groupBy('Airline').sum().show()

df.groupBy('Airline').mean().show()

# Create a list of dictionaries containing sample data

data = [{"Name": "John", "Age": 25},

        {"Name": "Alice", "Age": 30},

        {"Name": "Bob", "Age": 35}]

# Create DataFrame using the list of dictionaries

df = spark.createDataFrame(data)