PySpark

- RDDs are immutable, fault-tolerant, distributed collections of objects that can be operated on in parallel.

- Accumulators are variables used for aggregating information across the executors.

Read CSV file into df

```
file_path = "path/to/your/file.csv"

df = spark.read.csv(file_path, header=True, sep=",",
inferSchema=True)
```

OR

```
df = spark.read.format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load(file_path)

#JOIN two df

join_df = df1.join(df2, on='id', how='inner')
#OR
join_df = df1.join(df2, df1["id"] == df2["user_id"],
how='inner')

# Populate a new column 'new_column' with constant value
'xyz'
df = df.withColumn("new_column", lit("xyz"))

# Drop duplicates based on specific columns
df_no_duplicates_specific = df.dropDuplicates(["id"])
```

find out all the df in a particular sparkSession

```
from  spark.sql import DataFrame

for key, value in globals().items()
    if (type(v)==DataFrame):
        print(k)
```

What are the various types of Cluster Managers in PySpark?

Spark supports the following cluster managers:

● **Standalone-** a simple cluster manager that comes with Spark and makes setting up a cluster easier.
● **Apache Mesos**- Mesos is a cluster manager that can also run Hadoop MapReduce and PySpark applications.
● **Hadoop YARN**- It is the Hadoop 2 resource management.
● **Kubernetes**- an open-source framework for automating containerized application deployment, scaling, and administration.
● **local** – not exactly a cluster manager, but it's worth mentioning because we use "local" for master() to run Spark on our laptop/computer.

Word count problem

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

spark = SparkSession.builder.appname("app").getOrCreate()

file_path = “path/to/file”
df = spark.read.text(file_path)

df_words = df.flatMap(lambda x : x.split(“ “))
word_count = df_words.map(lambda x : (x.lower()))

# final_count = word_count.reduceByKey(lambda x, y : x+y)
# result = final_count.collect()

final_count =  word_count.countByValue()

print(final_count)

for a in result:
        print(a)
```

————————————————

Given an Input data set with name, age and city
if age > 18 add a new column that’s populated with ‘Y’ else ‘N’
Please solve this using apache spark

```
df = df.withColumn("is_adult", when(df.age > 18,
'Y').otherwise('N'))
df.show()
```

PySpark Window Functions:

Row_number()

```
From pyspark.sql.window import window
From pyspark.sql.functions import functions

#Row_number() OVER (PARTITION BY col1 ORDER BY col2)

windowSpec = window.paritionBy(col1).orderBY(col2)

df1 =  df.withColumn(“row”, row_number().over(windowSpec))
df1.show()
```

RANK() / DENSE_RANK()

```
windowSpec = window.partitionBy("col1").orderBy("col1")
df1 = df.withColumn(“rank”, rank().over(windowSpec))

from pyspark.sql.functions import dense_rank
df.withColumn("dense_rank",dense_rank().over(windowSpec)) \
    .show()
```

LAG() / Lead()

```
windowSpec = window.partitionBy("col1").orderBy("col1")
df.withColumn("lag", lag("column_name", 2).over(windowSpec))

# Aggregate functions examples
from pyspark.sql.functions import
col,avg,sum,min,max,row_number

windowSpecAgg  = Window.partitionBy("department")
df.withColumn("row",row_number().over(windowSpec)) \
  .withColumn("avg", avg(col("salary")).over(windowSpecAgg))
\
  .withColumn("sum", sum(col("salary")).over(windowSpecAgg))
\
  .withColumn("min", min(col("salary")).over(windowSpecAgg))
\
  .withColumn("max", max(col("salary")).over(windowSpecAgg))
\
  .where(col("row")==1).select("department","avg","sum","min
","max") \
  .show()
```

Merge two Dataframes

```
Df = df1.union(df2)
```

Find all duplicate emails

```
windowSpec = window.partitionBy(“email”)
Df1 = df.withColumn(“rn”, row_number().over(windowSpec))
Df2 = df1.filter(col(“rn”) > 1)
Df2.show()
```

```
ROW_NUMBER()        RANK()          DENSE_RANK()

A   1               A   1           A   1
A   2               A   1           A   1
B   1               B   3           B   2
C   1               C   4           C   3
C   2               C   4           C   3
D   1               D   6           D   4
```

Broadcast Join

```
df_joined = df1.join(broadcast(df2), df1.name == df2.name,
‘inner’)
```

Aggregation

Aggregation in PySpark involves summarizing data from a DataFrame by applying functions like sum, avg, max, min, etc., either on the entire DataFrame or grouped by one or more columns.

```
from pyspark.sql import functions as F

# Sample data data = [("Alice", 1),
                      ("Bob", 2),
                      ("Alice", 3),
                      ("Bob", 4),
                      ("Alice", 5)]

# Create a DataFrame
df = spark.createDataFrame(data, ["Name", "Value"])

# Aggregate without grouping
total_sum = df.agg(F.sum("Value").alias("TotalSum"),
                   F.avg("Value").alias("AverageValue"),
                   F.max("Value").alias("MaxValue"),
                   F.min("Value").alias("MinValue"))
```

```
total_sum.show()

#Output
+--------+----------------+--------+--------+
|TotalSum|    AverageValue|MaxValue|MinValue|
+--------+----------------+--------+--------+
|         15|             3.0|      5|         1|
+--------+----------------+--------+--------+
# Aggregate with grouping by "Name"
grouped_sum = df.groupBy("Name") \
                .agg(F.sum("Value").alias("TotalValue"),
                     F.avg("Value").alias("AverageValue"),
                     F.max("Value").alias("MaxValue"),
                     F.min("Value").alias("MinValue"))

grouped_sum.show()



+-----+----------+----------------+--------+--------+
| Name|TotalValue|    AverageValue|MaxValue|MinValue|
+-----+----------+----------------+--------+--------+
|  Bob|              6|               3.0|       4|      2|
|Alice|              9|               3.0|       5|      1|
+-----+----------+----------------+--------+--------+
```