PYSPARK CODING CHALLENGE

Apply Transformations & actions in pyspark

Spark Session

```
from pyspark.sql import SparkSession

# Start a Spark session

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

# Read employee and department CSV files

emp_df = spark.read.csv(r"C:\Users\harci\Downloads\employees.csv",
header=True, inferSchema=True)

dept_df = spark.read.csv(r"C:\Users\harci\Downloads\departments.csv",
header=True, inferSchema=True)

emp_df.show()

dept_df.show()
```

```
+---+-------+-------+------+
| id|   name|   dept|salary|
+---+-------+-------+------+
|  1|  Alice|     HR|  3000|
|  2|    Bob|     IT|  4000|
|  3|Charlie|     IT|  4500|
|  4|  David|Finance|  5000|
|  5|    Eva|     HR|  3500|
+---+-------+-------+------+

+-------+---------+
|   dept| location|
+-------+---------+
|     HR|  Chennai|
|     IT|Bangalore|
|Finance|   Mumbai|
+-------+---------+
```

TRANSFORMATIONS

1. Filter

Used to filter rows based on condition.

Employees with salary > 4000

filtered_df = emp_df.filter(emp_df.salary > 4000)

filtered_df.show()

```
# Employees with salary > 4000
filtered_df = emp_df.filter(emp_df.salary > 4000)
filtered_df.show()
```

```
+---+-------+-------+------+
| id|   name|   dept|salary|
+---+-------+-------+------+
|  3|Charlie|     IT|  4500|
|  4|  David|Finance|  5000|
+---+-------+-------+------+
```

2. Join

Used to combine two DataFrames based on a key column.

Joining employees and department on dept

joined_df = emp_df.join(dept_df, on="dept", how="inner")

joined_df.show()

```
# Joining employees and department on dept
joined_df = emp_df.join(dept_df, on="dept", how="inner")
joined_df.show()
```

```
+-------+---+-------+------+---------+
|   dept| id|   name|salary| location|
+-------+---+-------+------+---------+
|     HR|  1|  Alice|  3000|  Chennai|
|     IT|  2|    Bob|  4000|Bangalore|
|     IT|  3|Charlie|  4500|Bangalore|
|Finance|  4|  David|  5000|   Mumbai|
|     HR|  5|    Eva|  3500|  Chennai|
+-------+---+-------+------+---------+
```

3. Simple Aggregations (with select or agg)

Use sum, avg, count, etc. without grouping.

Total salary of all employees

from pyspark.sql.functions import avg, max, sum

agg_df = emp_df.agg(sum("salary").alias("total_salary"))

agg_df.show()

```
from pyspark.sql.functions import avg, max, sum

# Total salary of all employees
agg_df = emp_df.agg(sum("salary").alias("total_salary"))
agg_df.show()
```

```
+------------+
|total_salary|
+------------+
|       20000|
+------------+
```

4. GroupBy

Used to group rows and perform aggregations.

Average salary per department

grouped_df = emp_df.groupBy("dept").agg(avg("salary").alias("avg_salary"))

grouped_df.show()

```
# Average salary per department
grouped_df = emp_df.groupBy("dept").agg(avg("salary").alias("avg_salary"))
grouped_df.show()
```

```
+-------+----------+
|   dept|avg_salary|
+-------+----------+
|     HR|    3250.0|
|Finance|    5000.0|
|     IT|    4250.0|
+-------+----------+
```

5. Window Functions

Perform operations like ranking or running totals over a window (partition).

```
from pyspark.sql.window import Window

from pyspark.sql.functions import row_number

# Rank employees by salary within each department

window_spec = Window.partitionBy("dept").orderBy(emp_df.salary.desc())

ranked_df = emp_df.withColumn("rank",
row_number().over(window_spec))

ranked_df.show()
```

```
+---+-------+-------+------+----+
| id|   name|   dept|salary|rank|
+---+-------+-------+------+----+
|  4|  David|Finance|  5000|   1|
|  5|    Eva|     HR|  3500|   1|
|  1|  Alice|     HR|  3000|   2|
|  3|Charlie|     IT|  4500|   1|
|  2|    Bob|     IT|  4000|   2|
+---+-------+-------+------+----+
```

6. Select

Used to select or compute new columns.

```
# Select name and salary only

selected_df = emp_df.select("name", "salary")

selected_df.show()
```

```
+-------+------+
|   name|salary|
+-------+------+
|  Alice|  3000|
|    Bob|  4000|
|Charlie|  4500|
|  David|  5000|
|    Eva|  3500|
+-------+------+
```

7. WithColumn

Used to add or modify a column.

Add 10% bonus column

bonus_df = emp_df.withColumn("bonus", emp_df.salary * 0.10)

bonus_df.show()

```
+---+-------+-------+------+-----+
| id|   name|   dept|salary|bonus|
+---+-------+-------+------+-----+
|  1|  Alice|     HR|  3000|300.0|
|  2|    Bob|     IT|  4000|400.0|
|  3|Charlie|     IT|  4500|450.0|
|  4|  David|Finance|  5000|500.0|
|  5|    Eva|     HR|  3500|350.0|
+---+-------+-------+------+-----+
```

8. OrderBy

Used to sort the DataFrame.

Sort employees by salary descending

sorted_df = emp_df.orderBy(emp_df.salary.desc())

sorted_df.show()

```
+---+-------+-------+------+
| id|   name|   dept|salary|
+---+-------+-------+------+
|  4|  David|Finance|  5000|
|  3|Charlie|     IT|  4500|
|  2|    Bob|     IT|  4000|
|  5|    Eva|     HR|  3500|
|  1|  Alice|     HR|  3000|
+---+-------+-------+------+
```

9. Distinct

Removes duplicates.

Distinct departments

distinct_df = emp_df.select("dept").distinct()

distinct_df.show()

```
+-------+
|   dept|
+-------+
|     HR|
|Finance|
|     IT|
+-------+
```

10. Drop

Removes a column from the DataFrame.

Drop dept column

dropped_df = emp_df.drop("dept")

dropped_df.show()

```
+---+-------+------+
| id|   name|salary|
+---+-------+------+
|  1|  Alice|  3000|
|  2|    Bob|  4000|
|  3|Charlie|  4500|
|  4|  David|  5000|
|  5|    Eva|  3500|
+---+-------+------+
```

ACTIONS

1. show()

Displays the DataFrame contents in a tabular format.

emp_df.show()

```
+---+-------+-------+------+
| id|   name|   dept|salary|
+---+-------+-------+------+
|  1|  Alice|     HR|  3000|
|  2|    Bob|     IT|  4000|
|  3|Charlie|     IT|  4500|
|  4|  David|Finance|  5000|
|  5|    Eva|     HR|  3500|
+---+-------+-------+------+
```

2. collect()

Returns all rows as a list of Row objects (moves data to driver!).

```
rows = emp_df.collect()

for row in rows:

 print(row.name, row.salary)
```

```
Alice 3000
Bob 4000
Charlie 4500
David 5000
Eva 3500
```

3. count()

Returns the number of rows in the DataFrame.

total_rows = emp_df.count()

print("Total rows:", total_rows)

```
# COUNT

total_rows = emp_df.count()
print("Total rows:", total_rows)
```

```
Total rows: 5
```

4. first() / head()

Returns the first row.

first_row = emp_df.first()

print(first_row)

```
# FIRST

first_row = emp_df.first()
print(first_row)
```

```
Row(id=1, name='Alice', dept='HR', salary=3000)
```

5. take(n)

Returns the first n rows as a list.

top_2 = emp_df.take(2)

for row in top_2:

print(row)

```
# TAKE(N)

top_2 = emp_df.take(2)
for row in top_2:
    print(row)
```

```
Row(id=1, name='Alice', dept='HR', salary=3000)
Row(id=2, name='Bob', dept='IT', salary=4000)
```

6. foreach()

Executes a function for each row (used for side effects only, not for returning values).

for row in emp_df.collect():

print(row.name, row.salary)

```
# FOREACH

for row in emp_df.collect():
    print(row.name, row.salary)
```

```
Alice 3000
Bob 4000
Charlie 4500
David 5000
Eva 3500
```

7. agg()

Triggers aggregation and returns result.

from pyspark.sql.functions import avg

emp_df.agg(avg("salary")).show()

```
# AGG
from pyspark.sql.functions import avg
emp_df.agg(avg("salary")).show()
```

```
+-----------+
|avg(salary)|
+-----------+
|     4000.0|
+-----------+
```

8. toPandas()

Converts the DataFrame to a Pandas DataFrame (moves data to driver).

pandas_df = emp_df.toPandas()

print(pandas_df.head())

```
# TO PANDAS
pandas_df = emp_df.toPandas()
print(pandas_df.head())
```

	id	name	dept	salary
0	1	Alice	HR	3000
1	2	Bob	IT	4000
2	3	Charlie	IT	4500
3	4	David	Finance	5000
4	5	Eva	HR	3500

9. describe() + show()

Returns summary statistics of numeric columns.

emp_df.describe().show()

```
+-------+------------------+-----+-------+-----------------+
|summary|                id| name|   dept|           salary|
+-------+------------------+-----+-------+-----------------+
|  count|                 5|    5|      5|                5|
|   mean|               3.0| NULL|   NULL|           4000.0|
| stddev|1.5811388300841898| NULL|   NULL|790.5694150420949|
|    min|                 1|Alice|Finance|             3000|
|    max|                 5|  Eva|     IT|             5000|
+-------+------------------+-----+-------+-----------------+
```

10. Schema ()

defining each column's name, data type, and nullable status.

emp_df.printSchema()

```
# schema

emp_df.printSchema()
```

```
root
 |-- id: integer (nullable = true)
 |-- name: string (nullable = true)
 |-- dept: string (nullable = true)
 |-- salary: integer (nullable = true)
```