Challenge 1: Write a PySpark function to remove duplicate rows from a DataFrame based on specific columns.

✅ Function: remove_duplicates_by_columns

```
from pyspark.sql import DataFrame

def remove_duplicates_by_columns(df: DataFrame, columns: list) -> DataFrame:
    """
    Removes duplicate rows based on specified columns.

    Parameters:
    df (DataFrame): The input PySpark DataFrame.
    columns (list): A list of column names to check for duplicates.

    Returns:
    DataFrame: A new DataFrame with duplicates removed based on specified cc
    """
    return df.dropDuplicates(columns)

```

🧪 Example Usage

```
from pyspark.sql import SparkSession

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

# Sample data
data = [
    ("Alice", "HR", 1000),
    ("Bob", "IT", 1200),
    ("Alice", "HR", 1000),  # duplicate
    ("Alice", "IT", 1000),  # not a duplicate
]

columns = ["name", "department", "salary"]
df = spark.createDataFrame(data, columns)

# Remove duplicates based on 'name' and 'department'
cleaned_df = remove_duplicates_by_columns(df, ["name", "department"])

cleaned_df.show()

```

🖨️ Output

```
+-----+----------+------+
| name|department|salary|
+-----+----------+------+
|Alice|        HR|  1000|
|  Bob|        IT|  1200|
|Alice|        IT|  1000|
+-----+----------+------+

```

✅ The row ("Alice", "HR", 1000) appeared twice, but only one instance is kept based on the ["name", "department"] columns.

Challenge 2: Create a PySpark pipeline to read a CSV file, filter out rows with null values, and write the result to a Parquet file.

✅ Objective:

📥 Read a CSV file.

🧹 Filter out rows with any null values.

💾 Write the cleaned data to a Parquet file.

```
from pyspark.sql import SparkSession

def csv_to_parquet_pipeline(input_csv_path: str, output_parquet_path: str):
    # Step 1: Create Spark session
    spark = SparkSession.builder \
        .appName("CSVtoParquetPipeline") \
        .getOrCreate()

    # Step 2: Read CSV file
    df = spark.read.option("header", "true").csv(input_csv_path)

    # Step 3: Filter out rows with any null values
    cleaned_df = df.dropna()

    # Step 4: Write to Parquet
    cleaned_df.write.mode("overwrite").parquet(output_parquet_path)

    print(f"✅ Pipeline completed. Parquet written to: {output_parquet_patl

    # Optional: Stop Spark session
    spark.stop()

```

🧪 Example Usage

```
csv_to_parquet_pipeline("/content/drive/MyDrive/pyspark csv data for practi
```

Challenge 3: Implement a window function to rank salespeople based on total sales by region.

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StringType, StructType, IntegerType, StructFi
from pyspark.sql.functions import col, sum, desc, rank
from pyspark.sql.window import Window
```

```
 5
 6 #Step:1 - Create spark session
 7 spark = SparkSession.builder.appName("rankSalesPeopleByRegion").getOrCreate
 8
 9 schema = StructType([
10                       StructField('salesperson', StringType(), True),
11                       StructField('region', StringType(), True),
12                       StructField('sale_amount', IntegerType(), True)
13                     ])
14
15 #Step:2 - Read CSV file & create a df
16 df = spark.read.option('header', 'true').csv("/content/drive/MyDrive/pyspar
17                                                schema=schema)
18 # sales_df = df.groupBy('salesperson', 'region').agg(sum('sale_amount')).al
19 # df.show()
20 # df.printSchema()
21
22 df = df.withColumn('sale_amount', col('sale_amount').cast('int'))
23
24 # Step:3 - create a new df
25 sales_df = df.groupBy('salesperson', 'region').agg(sum('sale_amount').alias
26 # sales_df.show()
27
28 #Step:4 - Apply a window function
29 window_spec = Window.partitionBy('region').orderBy(desc('total_sales'))
30
31 ranked_df = sales_df.withColumn('Rank', rank().over(window_spec))
32
33 ranked_df.show()
```

```
+-----------+------+-----------+
|salesperson|region|total_sales|
+-----------+------+-----------+
|       Ivan| South|       1350|
|      David| South|       1100|
|       Judy|  East|       1000|
|      Alice|  East|       1200|
|        Bob|  West|        800|
|        Eve|  West|       1500|
|    Charlie|  East|        950|
|      Frank|  East|        700|
|      Heidi|  West|        500|
|      Grace| South|       1250|
+-----------+------+-----------+

+-----------+------+-----------+----+
|salesperson|region|total_sales|Rank|
+-----------+------+-----------+----+
|      Alice|  East|       1200|   1|
|       Judy|  East|       1000|   2|
|    Charlie|  East|        950|   3|
|      Frank|  East|        700|   4|
|       Ivan| South|       1350|   1|
|      Grace| South|       1250|   2|
|      David| South|       1100|   3|
|        Eve|  West|       1500|   1|
|        Bob|  West|        800|   2|
|      Heidi|  West|        500|   3|
+-----------+------+-----------+----+
```

Challenge 4: Write a PySpark SQL query to calculate the average salary by department, including only employees with more than 3 years of experience.

🎯 Task:

Write a PySpark SQL query to:

Filter employees with more than 3 years of experience

Group by department

Calculate the average salary

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructField, StructType, StringType, IntegerT
from pyspark.sql.functions import avg

# Step-1: create a spark session
spark = SparkSession.builder.appName("AvgSal_Deptwise").getOrCreate()

schema = StructType([
    StructField('employee_id', IntegerType(), True),
    StructField('name', StringType(), True),
    StructField('department', StringType(), True),
    StructField('salary', IntegerType(), True),
    StructField('experience', IntegerType(), True)
])

# Step-2 : Read csv
df = spark.read.option('header', 'true').csv('/content/drive/MyDrive/pyspar
                                                    schema=schema)
# df.show()
# df.printSchema()

# Step-3 : Filter employees with more than 3 years of experience
df = df.filter(df['experience'] > 3)
# df.show()

# Step-4 : Calculate the average salary department-wise
df = df.groupBy('department').agg(avg('salary').alias('avg_sal_deptwise'))
df.show()

```

```
+----------+----------------+
|department|avg_sal_deptwise|
+----------+----------------+
|        HR|         61000.0|
|   Finance|         90000.0|
|        IT|         80500.0|
+----------+----------------+
```

Challenge 5: Implement a PySpark function to split a large DataFrame into smaller DataFrames based on a specific column value.

```
from pyspark.sql import SparkSession
```

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Split_LargeDF_to_SmallerDFs").getOrCre

schema = StructType([
    StructField('employee_id', IntegerType(), True),
    StructField('name', StringType(), True),
    StructField('region', StringType(), True),
    StructField('department', StringType(), True),
    StructField('salary', IntegerType(), True)
])

# Step-1: Read CSV
df = spark.read.option('header', 'true').csv('/content/drive/MyDrive/pyspark
                                             , schema = schema)
df.show()

# Step-2: Let's split the data on Region, let's store unique regions in regi
regions = [row['region'] for row in df.select('region').distinct().collect()

# Step-3: Create a dictionary of data frames for each region
dfs_by_region = {region:df.filter(df.region == region) for region in regions

dfs_by_region
print(dfs_by_region)

dfs_by_region['East'].show()
dfs_by_region['West'].show()
dfs_by_region['North'].show()
dfs_by_region['South'].show()
```

```
+-----------+-------+------+----------+------+
|employee_id|   name|region|department|salary|
+-----------+-------+------+----------+------+
|          1|  Alice|  East|        HR| 60000|
|          2|    Bob|  West|        IT| 75000|
|          3|Charlie| North|        IT| 82000|
|          4|  David|  East|   Finance| 90000|
|          5|    Eve|  West|        HR| 58000|
|          6|  Frank| South|        IT| 79000|
|          7|  Grace|  East|   Finance| 91000|
|          8|  Heidi| North|        HR| 62000|
|          9|   Ivan| South|        IT| 70000|
|         10|   Judy|  West|   Finance| 88000|
+-----------+-------+------+----------+------+

{'South': DataFrame[employee_id: int, name: string, region: string, departm
+-----------+-----+------+----------+------+
|employee_id| name|region|department|salary|
+-----------+-----+------+----------+------+
|          1|Alice|  East|        HR| 60000|
|          4|David|  East|   Finance| 90000|
|          7|Grace|  East|   Finance| 91000|
+-----------+-----+------+----------+------+

+-----------+----+------+----------+------+
|employee_id|name|region|department|salary|
+-----------+----+------+----------+------+
|          2| Bob|  West|        IT| 75000|
|          5| Eve|  West|        HR| 58000|
|         10|Judy|  West|   Finance| 88000|
+-----------+----+------+----------+------+

+-----------+-------+------+----------+------+
|employee_id|   name|region|department|salary|
+-----------+-------+------+----------+------+
|          3|Charlie| North|        IT| 82000|
|          8|  Heidi| North|        HR| 62000|
+-----------+-------+------+----------+------+

+-----------+-----+------+----------+------+
|employee_id| name|region|department|salary|
+-----------+-----+------+----------+------+
|          6|Frank| South|        IT| 79000|
|          9| Ivan| South|        IT| 70000|
+-----------+-----+------+----------+------+
```