1) Colab setup — install Java, Spark binary, PySpark, findspark

```
# Install Java
!apt-get install openjdk-11-jdk -y

# Install PySpark
!pip install pyspark
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  fonts-dejavu-core fonts-dejavu-extra libatk-wrapper-java
  libatk-wrapper-java-jni libxt-dev libxtst6 libxxf86dga1 openjdk-11-jre
  x11-utils
Suggested packages:
  libxt-doc openjdk-11-demo openjdk-11-source visualvm mesa-utils
The following NEW packages will be installed:
  fonts-dejavu-core fonts-dejavu-extra libatk-wrapper-java
  libatk-wrapper-java-jni libxt-dev libxtst6 libxxf86dga1 openjdk-11-jdk
  openjdk-11-jre x11-utils
0 upgraded, 10 newly installed, 0 to remove and 38 not upgraded.
Need to get 5,367 kB of archives.
After this operation, 15.2 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy/main amd64 fonts-dejavu-core all 2.37-2build1 [1,041 k
Get:2 http://archive.ubuntu.com/ubuntu jammy/main amd64 fonts-dejavu-extra all 2.37-2build1 [2,041
Get:3 http://archive.ubuntu.com/ubuntu jammy/main amd64 libxtst6 amd64 2:1.2.3-1build4 [13.4 kB]
Get:4 http://archive.ubuntu.com/ubuntu jammy/main amd64 libxxf86dga1 amd64 2:1.1.5-0ubuntu3 [12.6 k
Get:5 http://archive.ubuntu.com/ubuntu jammy/main amd64 x11-utils amd64 7.7+5build2 [206 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy/main amd64 libatk-wrapper-java all 0.38.0-5build1 [53.
Get:7 http://archive.ubuntu.com/ubuntu jammy/main amd64 libatk-wrapper-java-jni amd64 0.38.0-5build
Get:8 http://archive.ubuntu.com/ubuntu jammy/main amd64 libxt-dev amd64 1:1.2.1-1 [396 kB]
Get:9 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 openjdk-11-jre amd64 11.0.28+6-1ubu
Get:10 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 openjdk-11-jdk amd64 11.0.28+6-1ub
Fetched 5,367 kB in 3s (1,999 kB/s)
Selecting previously unselected package fonts-dejavu-core.
(Reading database ... 126675 files and directories currently installed.)
Preparing to unpack .../0-fonts-dejavu-core_2.37-2build1_all.deb ...
Unpacking fonts-dejavu-core (2.37-2build1) ...
Selecting previously unselected package fonts-dejavu-extra.
Preparing to unpack .../1-fonts-dejavu-extra_2.37-2build1_all.deb ...
Unpacking fonts-dejavu-extra (2.37-2build1) ...
Selecting previously unselected package libxtst6:amd64.
Preparing to unpack .../2-libxtst6_2%3a1.2.3-1build4_amd64.deb ...
Unpacking libxtst6:amd64 (2:1.2.3-1build4) ...
Selecting previously unselected package libxxf86dga1:amd64.
Preparing to unpack .../3-libxxf86dga1_2%3a1.1.5-0ubuntu3_amd64.deb ...
Unpacking libxxf86dga1:amd64 (2:1.1.5-0ubuntu3) ...
Selecting previously unselected package x11-utils.
Preparing to unpack .../4-x11-utils_7.7+5build2_amd64.deb ...
Unpacking x11-utils (7.7+5build2) ...
Selecting previously unselected package libatk-wrapper-java.
Preparing to unpack .../5-libatk-wrapper-java_0.38.0-5build1_all.deb ...
Unpacking libatk-wrapper-java (0.38.0-5build1) ...
Selecting previously unselected package libatk-wrapper-java-jni:amd64.
Preparing to unpack .../6-libatk-wrapper-java-jni_0.38.0-5build1_amd64.deb ...
Unpacking libatk-wrapper-java-jni:amd64 (0.38.0-5build1) ...
Selecting previously unselected package libxt-dev:amd64.
Preparing to unpack .../7-libxt-dev_1%3a1.2.1-1_amd64.deb ...
Unpacking libxt-dev:amd64 (1:1.2.1-1) ...
Selecting previously unselected package ope[illegible]k-11-jre:amd64.
```

```
Preparing to unpack .../8-openjdk-11-jre_11.0.28+6-1ubuntu1~22.04.1_amd64.deb ...
Unpacking openjdk-11-jre:amd64 (11.0.28+6-1ubuntu1~22.04.1) ...
Selecting previously unselected package openjdk-11-jdk:amd64.
Preparing to unpack .../9-openjdk-11-jdk_11.0.28+6-1ubuntu1~22.04.1_amd64.deb ...
```

```
# Download Spark (prebuilt for Hadoop 3)
!wget https://archive.apache.org/dist/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz

# Extract it
!tar xvf spark-3.5.0-bin-hadoop3.tgz
```

```
spark-3.5.0-bin-hadoop3/examples/src/main/resources/dir1/dir2/
spark-3.5.0-bin-hadoop3/examples/src/main/resources/dir1/dir2/file2.parquet
spark-3.5.0-bin-hadoop3/examples/src/main/resources/dir1/file3.json
spark-3.5.0-bin-hadoop3/examples/src/main/resources/kv1.txt
```

```
import os

os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.5.0-bin-hadoop3"
os.environ["PATH"] += ":/content/spark-3.5.0-bin-hadoop3/bin"
```

```
# 1
```

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("RDD_Practical") \
    .config("spark.sql.shuffle.partitions", "2") \
    .getOrCreate()

sc = spark.sparkContext
print("SparkContext initialized:", sc)
```

```
SparkContext initialized: <SparkContext master=local[*] appName=RDD_Practical>
```

```
# ------------------------------------------------------------
# 1. Parallelize (create RDD from a Python list)
# ------------------------------------------------------------
data = [1, 2, 3, 4, 5, 6]
rdd1 = sc.parallelize(data)
print("Parallelized RDD:", rdd1.collect())

# ------------------------------------------------------------
# 2. Read Text File into RDD
# ------------------------------------------------------------
# Create a sample text file
with open("sample_text.txt", "w") as f:
    f.write("Hello Spark\n")
    f.write("RDD Basics\n")
    f.write("PySpark in Colab\n")

text_rdd = sc.textFile("sample_text.txt")
print("Text File RDD:", text_rdd.collect())

# ------------------------------------------------------------
# 3. Read Multiple Text Files into RDD
# ------------------------------------------------------------
# Create more text files
with open("text1.txt", "w") as f:
    f.write("File One\nLine A\nLine B\n")
with open("text2.txt", "w") as f:
    f.write("File Two\nLine C\nLine D\n")

multi_text_rdd = sc.textFile("text*.txt")
print("Multiple Text Files RDD:", multi_text_rdd.collect())

# ------------------------------------------------------------
```

```
# 4. Read CSV File into RDD
# ------------------------------------------------------------
# Create a sample CSV file
with open("sample.csv", "w") as f:
    f.write("id,name,age\n")
    f.write("1,Alice,23\n")
    f.write("2,Bob,30\n")
    f.write("3,Charlie,28\n")

csv_rdd = sc.textFile("sample.csv")
csv_parsed = csv_rdd.map(lambda line: line.split(","))
print("CSV RDD:", csv_parsed.collect())

# ------------------------------------------------------------
# 5. Create Empty RDD
# ------------------------------------------------------------
empty_rdd = sc.emptyRDD()
print("Empty RDD Count:", empty_rdd.count())

# ------------------------------------------------------------
# 6. RDD Actions
# ------------------------------------------------------------
print("Count:", rdd1.count())
print("First Element:", rdd1.first())
print("Sum:", rdd1.sum())
print("Take 3:", rdd1.take(3))

# ------------------------------------------------------------
# 7. Pair RDD Functions (key-value pairs)
# ------------------------------------------------------------
pair_rdd = rdd1.map(lambda x: (x, x*2))
print("Pair RDD:", pair_rdd.collect())

# Example: word count
word_rdd = sc.parallelize(["hello world", "hello spark", "spark rdd"])
word_pairs = word_rdd.flatMap(lambda line: line.split(" ")) \
                     .map(lambda word: (word, 1)) \
                     .reduceByKey(lambda a,b: a+b)
print("Word Count:", word_pairs.collect())

# ------------------------------------------------------------
# 8. Repartition and Coalesce
# ------------------------------------------------------------
print("Original Partitions:", rdd1.getNumPartitions())
rdd_repart = rdd1.repartition(4)
print("Repartitioned Partitions:", rdd_repart.getNumPartitions())
rdd_coalesce = rdd_repart.coalesce(2)
print("Coalesced Partitions:", rdd_coalesce.getNumPartitions())

# ------------------------------------------------------------
# 9. Shuffle Partitions (configure shuffle)
# ------------------------------------------------------------
spark.conf.set("spark.sql.shuffle.partitions", "5")
print("Shuffle Partitions set to:", spark.conf.get("spark.sql.shuffle.partitions"))

# ------------------------------------------------------------
# 10. Broadcast Variables
# ------------------------------------------------------------
broadcast_var = sc.broadcast([10,20,30])
print("Broadcast Value:", broadcast_var.value)
```

```
# ------------------------------------------------------------
# 11. Accumulator Variables
# ------------------------------------------------------------
accum = sc.accumulator(0)

def add_accum(x):
    global accum
    accum += x

rdd1.foreach(add_accum)
print("Accumulator Sum:", accum.value)

# ------------------------------------------------------------
# 12. Convert RDD to DataFrame
# ------------------------------------------------------------
df_from_rdd = csv_parsed.toDF(["id", "name", "age"])
df_from_rdd.show()
```

```
Parallelized RDD: [1, 2, 3, 4, 5, 6]
Text File RDD: ['Hello Spark', 'RDD Basics', 'PySpark in Colab']
Multiple Text Files RDD: ['File One', 'Line A', 'Line B', 'File Two', 'Line C', 'Line D']
CSV RDD: [['id', 'name', 'age'], ['1', 'Alice', '23'], ['2', 'Bob', '30'], ['3', 'Charlie', '28']]
Empty RDD Count: 0
Count: 6
First Element: 1
Sum: 21
Take 3: [1, 2, 3]
Pair RDD: [(1, 2), (2, 4), (3, 6), (4, 8), (5, 10), (6, 12)]
Word Count: [('hello', 2), ('world', 1), ('rdd', 1), ('spark', 2)]
Original Partitions: 2
Repartitioned Partitions: 4
Coalesced Partitions: 2
Shuffle Partitions set to: 5
Broadcast Value: [10, 20, 30]
Accumulator Sum: 21
+---+-------+---+
| id|   name|age|
+---+-------+---+
| id|   name|age|
|  1|  Alice| 23|
|  2|    Bob| 30|
|  3|Charlie| 28|
+---+-------+---+
```

```
#2
```

```
# Install Java & Spark (if using Google Colab)
!apt-get install openjdk-11-jdk-headless -qq > /dev/null
!wget -q https://downloads.apache.org/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz
!tar xf spark-3.5.0-bin-hadoop3.tgz
!pip install -q findspark

# Setup environment
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.5.0-bin-hadoop3"

import findspark
findspark.init()
```

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

# Create Spark session
spark = SparkSession.builder \
    .appName("Practical2_DataFrameOps") \
    .config("spark.sql.shuffle.partitions", "2") \
    .getOrCreate()
```

```
# 1. Create an empty DataFrame
empty_df = spark.createDataFrame([], StructType([]))
print("Empty DataFrame:")
empty_df.show()

# 2. Create an empty Dataset (in PySpark = typed DataFrame)
schema = StructType([
    StructField("id", IntegerType(), True),
    StructField("name", StringType(), True)
])
empty_ds = spark.createDataFrame([], schema)
print("Empty Dataset (typed DataFrame):")
empty_ds.show()

# Create sample DataFrame for next operations
data = [
    (1, "Alice", 2000, "HR"),
    (2, "Bob", 2500, "IT"),
    (3, "Cathy", 3000, "IT"),
    (4, "David", None, "Finance"),
    (5, "Eve", 2800, "Finance"),
    (6, "Frank", None, "HR")
]
columns = ["id", "name", "salary", "dept"]
df = spark.createDataFrame(data, columns)

# 3. Rename nested column (simulate nested structure)
nested_schema = StructType([
    StructField("id", IntegerType(), True),
    StructField("info", StructType([
        StructField("name", StringType(), True),
        StructField("salary", IntegerType(), True)
    ]))
])
nested_data = [(1, ("Alice", 2000)), (2, ("Bob", 2500))]
nested_df = spark.createDataFrame(nested_data, nested_schema)
renamed_df = nested_df.withColumnRenamed("info.name", "employee_name")
print("Rename nested column:")
renamed_df.show()

# 4. Add or Update a column
df = df.withColumn("bonus", col("salary") * 0.1)
print("Added Bonus column:")
df.show()

# 5. Drop a column
df_drop = df.drop("bonus")
print("After Dropping Bonus column:")
df_drop.show()
```

```
# 6. Add literal constant
df_lit = df.withColumn("Country", lit("India"))
print("Added constant column:")
df_lit.show()

# 7. Change column data type
df_cast = df.withColumn("salary", col("salary").cast("double"))
print("Salary converted to double:")
df_cast.show()

# 8. Pivot & Unpivot
sales_data = [
    ("Q1", "ProductA", 100),
    ("Q1", "ProductB", 150),
    ("Q2", "ProductA", 200),
    ("Q2", "ProductB", 250),
]
sales_df = spark.createDataFrame(sales_data, ["quarter", "product", "revenue"])

pivot_df = sales_df.groupBy("quarter").pivot("product").sum("revenue")
print("Pivot Example:")
pivot_df.show()

unpivot_df = pivot_df.selectExpr("quarter", "stack(2, 'ProductA', ProductA, 'ProductB', ProductB) as
print("Unpivot Example:")
unpivot_df.show()

# 9. Create DataFrame using StructType & StructField
custom_schema = StructType([
    StructField("emp_id", IntegerType(), True),
    StructField("emp_name", StringType(), True),
    StructField("emp_salary", DoubleType(), True)
])
custom_data = [(101, "John", 5000.0), (102, "Mike", 6000.0)]
custom_df = spark.createDataFrame(custom_data, custom_schema)
print("Custom schema DataFrame:")
custom_df.show()
```

```
+---+-----+------+-------+-----+
|  1|Alice|2000.0|     HR|200.0|
|  2|  Bob|2500.0|     IT|250.0|
|  3|Cathy|3000.0|     IT|300.0|
|  4|David|  NULL|Finance| NULL|
|  5|  Eve|2800.0|Finance|280.0|
|  6|Frank|  NULL|     HR| NULL|
+---+-----+------+-------+-----+

Pivot Example:
+-------+--------+--------+
|quarter|ProductA|ProductB|
+-------+--------+--------+
|     Q1|     100|     150|
|     Q2|     200|     250|
+-------+--------+--------+

Unpivot Example:
+-------+--------+-------+
|quarter| product|revenue|
+-------+--------+-------+
|     Q1|ProductA|    100|
|     Q1|ProductB|    150|
|     Q2|ProductA|    200|
|     Q2|ProductB|    250|
+-------+--------+-------+

Custom schema DataFrame:
+------+--------+----------+
|emp_id|emp_name|emp_salary|
+------+--------+----------+
|   101|    John|    5000.0|
|   102|    Mike|    6000.0|
+------+--------+----------+
```

```
# 1. Selecting the first row of each group
first_row = df.groupBy("dept").agg(first("name").alias("first_employee"))
print("First row of each group:")
first_row.show()

# 2. Sort DataFrame
df_sorted = df.orderBy(col("salary").desc_nulls_last())
print("Sorted DataFrame:")
df_sorted.show()

# 3. Union DataFrame
df_union = df.union(df)
print("Union DataFrame:")
df_union.show()

# 4. Drop rows with null values
df_dropna = df.na.drop()
print("Drop null rows:")
df_dropna.show()

# 5. Split single column into multiple
split_df = df.withColumn("name_split", split(col("name"), "a"))
print("Split name column:")
split_df.show()

# 6. Concatenate multiple columns
concat_df = df.withColumn("full_info", concat_ws("-", col("name"), col("dept")))
print("Concatenate columns:")
```

```
concat_df.show()

# 7. Replace null values
df_fill = df.na.fill({"salary": 0, "name": "Unknown"})
print("Fill nulls:")
df_fill.show()

# 8. Remove duplicate rows
df_nodup = df.dropDuplicates()
print("Removed duplicates:")
df_nodup.show()

# 9. Distinct on multiple selected columns
df_distinct = df.select("dept", "salary").distinct()
print("Distinct on dept & salary:")
df_distinct.show()

# 10. Spark UDF
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

def upper_case(name):
    return name.upper()

upper_udf = udf(upper_case, StringType())
df_udf = df.withColumn("name_upper", upper_udf(col("name")))
print("Using UDF:")
df_udf.show()
```

```
Distinct on dept & salary:
+-------+------+
|   dept|salary|
+-------+------+
|     IT|  3000|
|     HR|  2000|
|     IT|  2500|
|Finance|  2800|
|Finance|  NULL|
|     HR|  NULL|
+-------+------+

Using UDF:
+---+-----+------+-------+-----+----------+
| id| name|salary|   dept|bonus|name_upper|
+---+-----+------+-------+-----+----------+
|  1|Alice|  2000|     HR|200.0|     ALICE|
|  2|  Bob|  2500|     IT|250.0|       BOB|
|  3|Cathy|  3000|     IT|300.0|     CATHY|
|  4|David|  NULL|Finance| NULL|     DAVID|
|  5|  Eve|  2800|Finance|280.0|       EVE|
|  6|Frank|  NULL|     HR| NULL|     FRANK|
+---+-----+------+-------+-----+----------+
```

```
#3
```

```
# (Run this only once at the beginning of your notebook)
!apt-get install openjdk-11-jdk-headless -qq > /dev/null
!wget -q https://downloads.apache.org/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz
!tar xf spark-3.5.0-bin-hadoop3.tgz
!pip install -q findspark

import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.5.0-bin-hadoop3"

import findspark
findspark.init()

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

spark = SparkSession.builder.appName("Practical3_ArrayMap").getOrCreate()
```

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, explode, lit, array, struct, map_from_arrays, concat_ws, flat
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, ArrayType, MapType

spark = SparkSession.builder \
    .appName("Array_Map_Operations") \
    .config("spark.sql.shuffle.partitions", "2") \
    .getOrCreate()


data = [("Alice", 25), ("Bob", 30)]
df = spark.createDataFrame(data, ["name","age"])

# Add Array column
df = df.withColumn("scores", array(lit(85), lit(90), lit(95)))
```

```
df.show(truncate=False)
df = df.withColumn("subject_scores", map_from_arrays(array(lit("Math"), lit("Physics")),array(lit(85
df.show(truncate=False)

df_array_cols = df.select(col("name"), col("scores")[0].alias("score1"),col("scores")[1].alias("scor
df_array_cols.show()

df = df.withColumn("score_structs", array(struct(lit("Math").alias("subject"), lit(85).alias("score"
df.show(truncate=False)

df_explode = df.withColumn("score", explode(col("scores")))
df_explode.show()

df_explode_struct = df.withColumn("struct_item", explode(col("score_structs")))
df_explode_struct.select("name", "struct_item.subject", "struct_item.score").show()

df_map_array = df.withColumn("map_array", array(col("subject_scores")))
df_map_explode = df_map_array.withColumn("map_item", explode(col("map_array")))
df_map_explode.show(truncate=False)

data_nested = [("Alice", [[1,2],[3,4]]), ("Bob", [[5,6],[7,8]])]
df_nested = spark.createDataFrame(data_nested, ["name","nested_array"])
df_nested.show(truncate=False)

df_explode_nested = df_nested.withColumn("exploded_array", explode(col("nested_array")))
df_explode_nested.show(truncate=False)

df_flattened = df_nested.withColumn("flattened_array", flatten(col("nested_array")))
df_flattened.show(truncate=False)

df_array_string = spark.createDataFrame([("Alice", ["A","B","C"])], ["name","letters"])
df_array_string = df_array_string.withColumn("letters_str", concat_ws(",", col("letters")))
df_array_string.show()
```

```
+-----+----------------+
|name |nested_array    |
+-----+----------------+
|Alice|[[1, 2], [3, 4]]|
|Bob  |[[5, 6], [7, 8]]|
+-----+----------------+

+-----+----------------+--------------+
|name |nested_array    |exploded_array|
+-----+----------------+--------------+
|Alice|[[1, 2], [3, 4]]|[1, 2]        |
|Alice|[[1, 2], [3, 4]]|[3, 4]        |
|Bob  |[[5, 6], [7, 8]]|[5, 6]        |
|Bob  |[[5, 6], [7, 8]]|[7, 8]        |
+-----+----------------+--------------+

+-----+----------------+---------------+
|name |nested_array    |flattened_array|
+-----+----------------+---------------+
|Alice|[[1, 2], [3, 4]]|[1, 2, 3, 4]   |
|Bob  |[[5, 6], [7, 8]]|[5, 6, 7, 8]   |
+-----+----------------+---------------+

+-----+---------+-----------+
| name|  letters|letters_str|
+-----+---------+-----------+
|Alice|[A, B, C]|      A,B,C|
+-----+---------+-----------+
```

```
# # Sample DataFrame
# data = [
#     (1, "Alice", [100, 200, 300], {"dept": "HR", "loc": "Mumbai"}),
#     (2, "Bob", [400, 500], {"dept": "IT", "loc": "Delhi"}),
#     (3, "Cathy", [600], {"dept": "Finance", "loc": "Pune"})
# ]
# schema = StructType([
#     StructField("id", IntegerType(), True),
#     StructField("name", StringType(), True),
#     StructField("scores", ArrayType(IntegerType()), True),
#     StructField("info", MapType(StringType(), StringType()), True)
# ])
# df = spark.createDataFrame(data, schema)
# print("Original DataFrame:")
# df.show(truncate=False)

# # 1. Create an Array (ArrayType) column
# df_array = df.withColumn("extra_scores", array(lit(50), lit(60), lit(70)))
# print("Array column added:")
# df_array.show(truncate=False)

# # 2. Create a Map (MapType) column
# df_map = df.withColumn("extra_info", create_map(lit("gender"), lit("F"), lit("status"), lit("activ
# print("Map column added:")
# df_map.show(truncate=False)

# # 3. Convert Array to Columns
# df_array_to_cols = df.withColumn("first_score", col("scores")[0]) \
#                      .withColumn("second_score", col("scores")[1])
# print("Array to columns:")
# df_array_to_cols.show(truncate=False)

# # 4. Create an Array of Struct column
```

```
# df_struct_array = df.withColumn("struct_array",
#     array(struct(col("id"), col("name"))))
# print("Array of struct column:")
# df_struct_array.show(truncate=False)

# # 5. Explode Array column
# df_explode = df.withColumn("score_exploded", explode(col("scores")))
# print("Exploded array column:")
# df_explode.show(truncate=False)

# # 6. Explode Map column
# df_explode_map = df.withColumn("map_exploded", explode(col("info")))
# print("Exploded map column:")
# df_explode_map.show(truncate=False)

# # 7. Explode Array of Structs
# data_struct = [
#     (1, [("HR", 2000), ("IT", 3000)]),
#     (2, [("Finance", 4000)])
# ]
# schema_struct = StructType([
#     StructField("id", IntegerType(), True),
#     StructField("dept_data", ArrayType(StructType([
#         StructField("dept", StringType(), True),
#         StructField("salary", IntegerType(), True)
#     ])), True)
# ])
# df_struct = spark.createDataFrame(data_struct, schema_struct)
# df_struct_explode = df_struct.withColumn("exploded_struct", explode(col("dept_data")))
# print("Exploded Array of Structs:")
# df_struct_explode.show(truncate=False)

# # 8. Explode Array of Maps
# data_map = [
#     (1, [{"a": "x", "b": "y"}, {"c": "z"}]),
#     (2, [{"d": "p"}])
# ]
# schema_map = StructType([
#     StructField("id", IntegerType(), True),
#     StructField("map_array", ArrayType(MapType(StringType(), StringType())), True)
# ])
# df_map_array = spark.createDataFrame(data_map, schema_map)
# df_map_array_explode = df_map_array.withColumn("exploded_map", explode(col("map_array")))
# print("Exploded Array of Maps:")
# df_map_array_explode.show(truncate=False)

# # 9. Create a DataFrame with nested Array
# nested_data = [(1, [["A", "B"], ["C", "D"]])]
# nested_schema = StructType([
#     StructField("id", IntegerType(), True),
#     StructField("nested_array", ArrayType(ArrayType(StringType())), True)
# ])
# nested_df = spark.createDataFrame(nested_data, nested_schema)
# print("Nested Array DataFrame:")
# nested_df.show(truncate=False)

# # 10. Explode nested Array
# nested_explode = nested_df.withColumn("exploded_nested", explode(col("nested_array")))
# print("Exploded nested Array:")
# nested_explode.show(truncate=False)
```

```
# # 11. Flatten nested Array to single Array
# flattened_df = nested_df.withColumn("flat_array", flatten(col("nested_array")))
# print("Flattened Array:")
# flattened_df.show(truncate=False)

# # 12. Convert array of String to String column
# df_str_array = df.withColumn("scores_str", concat_ws(",", col("scores")))
# print("Array of Ints to String column:")
# df_str_array.show(truncate=False)
```

Next steps: Explain error

```
Original DataFrame:
+---+-----+--------------+-----------------------------+
|id |name |scores        |info                         |
+---+-----+--------------+-----------------------------+
|1  |Alice|[100, 200, 300]|{loc -> Mumbai, dept -> HR}  |
|2  |Bob  |[400, 500]    |{loc -> Delhi, dept -> IT}   |
|3  |Cathy|[600]         |{loc -> Pune, dept -> Finance}|
+---+-----+--------------+-----------------------------+

Array column added:
+---+-----+--------------+-----------------------------+------------+
|id |name |scores        |info                         |extra_scores|
```

#4

```
|2  |Bob  |[400, 500]    |{loc -> Delhi, dept -> IT}   |[50, 60, 70]|
```

```
# Install & setup Spark (only if not done earlier)
!apt-get install openjdk-11-jdk-headless -qq > /dev/null
!wget -q https://downloads.apache.org/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz
!tar xf spark-3.5.0-bin-hadoop3.tgz
!pip install -q findspark

import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.5.0-bin-hadoop3"

import findspark
findspark.init()

from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.window import Window
from pyspark.sql.types import *

spark = SparkSession.builder.appName("Practical4_Aggregates").getOrCreate()
```

```
|id |name |scores        |info                         |struct_array|
```

```
# Sample data
data = [
    ("Alice", "HR", 3000),
    ("Bob", "HR", 4000),
    ("Cathy", "IT", 5000),
    ("David", "IT", 6000),
    ("Eva", "Finance", 7000),
    ("Frank", "Finance", 8000),
    ("George", "HR", 3000)
]
schema = ["name", "dept", "salary"]
df = spark.createDataFrame(data, schema)
print("Original DataFrame:")
df.show()

# 1. Group rows in DataFrame (sum, avg, max, min)
grouped_df = df.groupBy("dept").agg(
    count("*").alias("employee_count"),
    avg("salary").alias("avg_salary"),
    sum("salary").alias("total_salary"),
    max("salary").alias("max_salary"),
    min("salary").alias("min_salary")
)
print("Grouped DataFrame with aggregates:")
grouped_df.show()
```

```
# 2. Get Count distinct on DataFrame
distinct_count = df.select(countDistinct("dept").alias("distinct_departments"))
print("Distinct count of departments:")
distinct_count.show()

# 3. Add row number to DataFrame (Window function)
windowSpec = Window.partitionBy("dept").orderBy(col("salary").desc())
df_with_rownum = df.withColumn("row_number", row_number().over(windowSpec))
print("DataFrame with row numbers:")
df_with_rownum.show()

# 4. Select the first row of each group (highest salary in each dept)
first_row_df = df_with_rownum.filter(col("row_number") == 1).drop("row_number")
print("First row of each group (Top salary per dept):")
first_row_df.show()
```

```
Original DataFrame:
+------+-------+------+
|  name|   dept|salary|
+------+-------+------+
| Alice|     HR|  3000|
|   Bob|     HR|  4000|
| Cathy|     IT|  5000|
| David|     IT|  6000|
|   Eva|Finance|  7000|
| Frank|Finance|  8000|
|George|     HR|  3000|
+------+-------+------+

Grouped DataFrame with aggregates:
+-------+--------------+------------------+------------+----------+----------+
|   dept|employee_count|        avg_salary|total_salary|max_salary|min_salary|
+-------+--------------+------------------+------------+----------+----------+
|     HR|             3|3333.3333333333335|       10000|      4000|      3000|
|     IT|             2|            5500.0|       11000|      6000|      5000|
|Finance|             2|            7500.0|       15000|      8000|      7000|
+-------+--------------+------------------+------------+----------+----------+

Distinct count of departments:
+--------------------+
|distinct_departments|
+--------------------+
|                   3|
+--------------------+

DataFrame with row numbers:
+------+-------+------+----------+
|  name|   dept|salary|row_number|
+------+-------+------+----------+
| Frank|Finance|  8000|         1|
|   Eva|Finance|  7000|         2|
|   Bob|     HR|  4000|         1|
| Alice|     HR|  3000|         2|
|George|     HR|  3000|         3|
| David|     IT|  6000|         1|
| Cathy|     IT|  5000|         2|
+------+-------+------+----------+

First row of each group (Top salary per dept):
+-----+-------+------+
| name|   dept|salary|
+-----+-------+------+
|Frank|Finance|  8000|
```

```
|  Bob|     HR|  4000|
|David|     IT|  6000|
+-----+-------+------+
```

```
#5
```

```
!apt-get install openjdk-11-jdk-headless -qq > /dev/null
!wget -q https://downloads.apache.org/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz
!tar xf spark-3.5.0-bin-hadoop3.tgz
!pip install -q findspark

import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.5.0-bin-hadoop3"

import findspark
findspark.init()

from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.sql.window import Window

spark = SparkSession.builder.appName("Practical5_Joins_SQL").getOrCreate()
```

```
# Sample DataFrames
emp_data = [
    (1, "Alice", 101, 3000),
    (2, "Bob", 102, 4000),
    (3, "Cathy", 101, 5000),
    (4, "David", 103, 6000),
    (5, "Eva", 104, 7000)
]
dept_data = [
    (101, "HR"),
    (102, "IT"),
    (103, "Finance")
]

emp_schema = ["emp_id", "name", "dept_id", "salary"]
dept_schema = ["dept_id", "dept_name"]

emp_df = spark.createDataFrame(emp_data, emp_schema)
dept_df = spark.createDataFrame(dept_data, dept_schema)

print("Employees:")
emp_df.show()
print("Departments:")
dept_df.show()

# 1. Spark SQL Join (register as temp view and run SQL)
emp_df.createOrReplaceTempView("employees")
dept_df.createOrReplaceTempView("departments")

sql_join = spark.sql("""
SELECT e.emp_id, e.name, d.dept_name, e.salary
FROM employees e
```

```
JOIN departments d ON e.dept_id = d.dept_id
""")
print("SQL Join:")
sql_join.show()

# 2. Join multiple DataFrames
extra_data = [(101, "Mumbai"), (102, "Delhi"), (103, "Pune"), (104, "Chennai")]
loc_df = spark.createDataFrame(extra_data, ["dept_id", "location"])

multi_join = emp_df.join(dept_df, "dept_id").join(loc_df, "dept_id")
print("Join multiple DataFrames:")
multi_join.show()

# 3. Inner join two tables/DataFrames
inner_join = emp_df.join(dept_df, emp_df.dept_id == dept_df.dept_id, "inner")
print("Inner Join:")
inner_join.show()

# 4. Self Join
self_join = emp_df.alias("a").join(emp_df.alias("b"), col("a.dept_id") == col("b.dept_id"))
print("Self Join (employees in same dept):")
self_join.select("a.name", "b.name", "a.dept_id").show()

# 5. Join tables on multiple columns
multi_col_join = emp_df.join(dept_df, (emp_df.dept_id == dept_df.dept_id) & (emp_df.salary > 4000))
print("Join on multiple columns:")
multi_col_join.show()

# 6. Convert case class to a schema (in PySpark we use StructType)
schema = StructType([
    StructField("id", IntegerType(), True),
    StructField("name", StringType(), True),
    StructField("age", IntegerType(), True)
])
df_schema = spark.createDataFrame([(1, "Alex", 30), (2, "Bella", 25)], schema)
print("DataFrame with StructType Schema:")
df_schema.show()

# 7. Create array of struct column
array_struct_df = emp_df.withColumn("array_struct", array(struct("emp_id", "name")))
print("Array of struct column:")
array_struct_df.show(truncate=False)

# 8. Flatten nested column
nested_data = [(1, ("Alice", ("HR", 3000)))]
nested_schema = StructType([
    StructField("id", IntegerType()),
    StructField("emp", StructType([
        StructField("name", StringType()),
        StructField("details", StructType([
            StructField("dept", StringType()),
            StructField("salary", IntegerType())
        ]))
    ]))
])
nested_df = spark.createDataFrame(nested_data, nested_schema)
flat_df = nested_df.select("id", col("emp.name").alias("emp_name"),
                           col("emp.details.dept").alias("department"),
                           col("emp.details.salary").alias("salary"))
print("Flatten nested column:")
```

```
flat_df.show()
```

```
Inner Join:
+------+-----+-------+------+-------+---------+
|emp_id| name|dept_id|salary|dept_id|dept_name|
+------+-----+-------+------+-------+---------+
|     3|Cathy|    101|  5000|    101|       HR|
|     1|Alice|    101|  3000|    101|       HR|
|     2|  Bob|    102|  4000|    102|       IT|
|     4|David|    103|  6000|    103|  Finance|
+------+-----+-------+------+-------+---------+

Self Join (employees in same dept):
+-----+-----+-------+
| name| name|dept_id|
+-----+-----+-------+
|Alice|Cathy|    101|
|Alice|Alice|    101|
|  Bob|  Bob|    102|
|  Eva|  Eva|    104|
|Cathy|Cathy|    101|
|Cathy|Alice|    101|
|David|David|    103|
+-----+-----+-------+

Join on multiple columns:
+------+-----+-------+------+-------+---------+
|emp_id| name|dept_id|salary|dept_id|dept_name|
+------+-----+-------+------+-------+---------+
|     3|Cathy|    101|  5000|    101|       HR|
|     4|David|    103|  6000|    103|  Finance|
+------+-----+-------+------+-------+---------+

DataFrame with StructType Schema:
+---+-----+---+
| id| name|age|
+---+-----+---+
|  1| Alex| 30|
|  2|Bella| 25|
+---+-----+---+

Array of struct column:
+------+-----+-------+------+------------+
|emp_id|name |dept_id|salary|array_struct|
+------+-----+-------+------+------------+
|1     |Alice|101    |3000  |[{1, Alice}]|
|2     |Bob  |102    |4000  |[{2, Bob}]  |
|3     |Cathy|101    |5000  |[{3, Cathy}]|
|4     |David|103    |6000  |[{4, David}]|
|5     |Eva  |104    |7000  |[{5, Eva}]  |
+------+-----+-------+------+------------+

Flatten nested column:
+---+--------+----------+------+
| id|emp_name|department|salary|
+---+--------+----------+------+
|  1|   Alice|        HR|  3000|
+---+--------+----------+------+
```

```
#6
```

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lit, when, udf
from pyspark.sql.types import StringType

spark = SparkSession.builder.appName("SparkSQL_Practical8").getOrCreate()
data = [(1,"Alice",23), (2,"Bob",30), (3,"Charlie",25), (4,"David",30)]
df = spark.createDataFrame(data, ["id","name","age"])
df.show()
df.filter(df.age > 25).show()
df.where(col("name") == "Alice").show()

df2 = df.withColumn("age_plus_5", col("age") + 5)
df2.show()
df3 = df.withColumnRenamed("name","full_name")
df3.show()
df4 = df.drop("age")
df4.show()
df.distinct().show()
df.groupBy("age").count().show()
dept = [(1,"HR"), (2,"IT"), (3,"Finance")]
df_dept = spark.createDataFrame(dept, ["id","dept"])

df_join = df.join(df_dept, on="id", how="inner")
df_join.show()
```

```
+---+-----+---+
| id| name|age|
+---+-----+---+
|  1|Alice| 23|
+---+-----+---+

+---+-------+---+----------+
| id|   name|age|age_plus_5|
+---+-------+---+----------+
|  1|  Alice| 23|        28|
|  2|    Bob| 30|        35|
|  3|Charlie| 25|        30|
|  4|  David| 30|        35|
+---+-------+---+----------+

+---+---------+---+
| id|full_name|age|
+---+---------+---+
|  1|    Alice| 23|
|  2|      Bob| 30|
|  3|  Charlie| 25|
```

```
|  2|    Bob| 30|
|  3|Charlie| 25|
|  4|  David| 30|
+---+-------+---+

+---+-----+
|age|count|
+---+-----+
| 23|    1|
| 30|    2|
| 25|    1|
+---+-----+

+---+-------+---+-------+
| id|   name|age|   dept|
+---+-------+---+-------+
|  1|  Alice| 23|     HR|
|  2|    Bob| 30|     IT|
|  3|Charlie| 25|Finance|
+---+-------+---+-------+
```

```
rdd = df.rdd

mapped = rdd.map(lambda x: (x[1], x[2]+1)).collect()
print("map:", mapped)

mapped_part = rdd.mapPartitions(lambda part: [(x[1], x[2]*2) for x in part]).collect()
print("mapPartitions:", mapped_part)
print("foreach output:")
df.rdd.foreach(lambda x: print(x))

print("foreachPartition output:")
df.rdd.foreachPartition(lambda part: [print("Partition:", list(part))])
```

```
map: [('Alice', 24), ('Bob', 31), ('Charlie', 26), ('David', 31)]
mapPartitions: [('Alice', 46), ('Bob', 60), ('Charlie', 50), ('David', 60)]
foreach output:
foreachPartition output:
```

```
sales = [("Alice","Jan",200),
         ("Alice","Feb",250),
         ("Bob","Jan",300),
         ("Bob","Feb",100)]

df_sales = spark.createDataFrame(sales, ["name","month","amount"])
pivoted = df_sales.groupBy("name").pivot("month").sum("amount")
pivoted.show()
df_a = spark.createDataFrame([(5,"Eve",28)], ["id","name","age"])
df_union = df.union(df_a)
df_union.show()
collected = df.collect()
print("collect():", collected)
df_cached = df.cache()
print("Cached count:", df_cached.count())

from pyspark import StorageLevel
df_persisted = df.persist(StorageLevel.MEMORY_AND_DISK)
print("Persisted count:", df_persisted.count())
def greet(name):
    return "Hello " + name

greet_udf = udf(greet, StringType())
```

```
df_udf = df.withColumn("greeting", greet_udf(col("name")))
df_udf.show()
```

```
+-----+---+---+
| name|Feb|Jan|
+-----+---+---+
|Alice|250|200|
|  Bob|100|300|
+-----+---+---+

+---+-------+---+
| id|   name|age|
+---+-------+---+
|  1|  Alice| 23|
|  2|    Bob| 30|
|  3|Charlie| 25|
|  4|  David| 30|
|  5|    Eve| 28|
+---+-------+---+

collect(): [Row(id=1, name='Alice', age=23), Row(id=2, name='Bob', age=30), Row(id=3, name='Charlie',
Cached count: 4
Persisted count: 4
+---+-------+---+-------------+
| id|   name|age|     greeting|
+---+-------+---+-------------+
|  1|  Alice| 23|  Hello Alice|
|  2|    Bob| 30|    Hello Bob|
|  3|Charlie| 25|Hello Charlie|
|  4|  David| 30|  Hello David|
+---+-------+---+-------------+
```

```
# # Run only once per session
# !apt-get install openjdk-11-jdk-headless -qq > /dev/null
# !wget -q https://downloads.apache.org/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz
# !tar xf spark-3.5.0-bin-hadoop3.tgz
# !pip install -q findspark
# !pip install -q pyspark
# !pip install -q spark-xml
# !pip install -q fastavro

# import os
# os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
# os.environ["SPARK_HOME"] = "/content/spark-3.5.0-bin-hadoop3"

# import findspark
# findspark.init()

# from pyspark.sql import SparkSession
# from pyspark.sql.functions import *
# from pyspark.sql.types import *

# spark = SparkSession.builder.appName("Practical7_DataSourceAPI").getOrCreate()
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/tmp/ipython-input-3189427153.py in <cell line: 0>()
      3 get_ipython().system('wget -q https://downloads.apache.org/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz')
      4 get_ipython().system('tar xf spark-3.5.0-bin-hadoop3.tgz')
----> 5 get_ipython().system('pip install -q findspark')
      6 get_ipython().system('pip install -q pyspark')
      7 get_ipython().system('pip install -q spark-xml')

                                 16 frames
/usr/lib/python3.12/pathlib.py in __init__(self, *args)
    356         return (self.__class__, self.parts)
    357
--> 358     def __init__(self, *args):
    359         paths = []
    360         for arg in args:

KeyboardInterrupt:
```

```
# import shutil
# import os

# # Create a temp folder for files
# os.makedirs("/content/data_source_demo", exist_ok=True)

# # Sample data
# data = [(1, "Alice", 3000), (2, "Bob", 4000), (3, "Cathy", 5000)]
# columns = ["id", "name", "salary"]
# df = spark.createDataFrame(data, columns)

# # ===== 7a: JSON, CSV, Parquet, XML, Avro =====
# # Write CSV
# df.write.mode("overwrite").csv("/content/data_source_demo/sample_csv", header=True)
# # Read CSV
# csv_df = spark.read.option("header",True).csv("/content/data_source_demo/sample_csv")
# print("CSV Read:")
# csv_df.show()

# # Write JSON
# df.write.mode("overwrite").json("/content/data_source_demo/sample_json")
# # Read JSON
# json_df = spark.read.json("/content/data_source_demo/sample_json")
# print("JSON Read:")
# json_df.show()

# # Write Parquet
# df.write.mode("overwrite").parquet("/content/data_source_demo/sample_parquet")
# # Read Parquet
# parquet_df = spark.read.parquet("/content/data_source_demo/sample_parquet")
# print("Parquet Read:")
# parquet_df.show()

# # Write XML (requires spark-xml)
# df.write.mode("overwrite").format("xml").option("rootTag","employees").option("rowTag","employee").
# # Read XML
# xml_df = spark.read.format("xml").option("rootTag","employees").option("rowTag","employee").load("/
# print("XML Read:")
# xml_df.show()

# # Write Avro
```

```
# # Write Avro
# df.write.mode("overwrite").format("avro").save("/content/data_source_demo/sample_avro")
# # Read Avro
# avro_df = spark.read.format("avro").load("/content/data_source_demo/sample_avro")
# print("Avro Read:")
# avro_df.show()

# # ===== 7b: ORC, Binary Files =====
# # Write ORC
# df.write.mode("overwrite").orc("/content/data_source_demo/sample_orc")
# # Read ORC
# orc_df = spark.read.orc("/content/data_source_demo/sample_orc")
# print("ORC Read:")
# orc_df.show()

# # Write binary file (save as text in binary)
# df.write.mode("overwrite").text("/content/data_source_demo/sample_text")
# # Read binary file
# binary_df = spark.read.text("/content/data_source_demo/sample_text")
# print("Binary/Text Read:")
# binary_df.show(truncate=False)

# # ===== 7c: File Conversions (CSV <-> JSON <-> Parquet <-> Avro <-> Text) =====
# # CSV -> JSON
# csv_to_json_df = spark.read.option("header",True).csv("/content/data_source_demo/sample_csv")
# csv_to_json_df.write.mode("overwrite").json("/content/data_source_demo/csv_to_json")
# # JSON -> Parquet
# json_to_parquet_df = spark.read.json("/content/data_source_demo/csv_to_json")
# json_to_parquet_df.write.mode("overwrite").parquet("/content/data_source_demo/json_to_parquet")
# # Parquet -> Avro
# parquet_to_avro_df = spark.read.parquet("/content/data_source_demo/json_to_parquet")
# parquet_to_avro_df.write.mode("overwrite").format("avro").save("/content/data_source_demo/parquet_t
# # Avro -> Text
# avro_to_text_df = spark.read.format("avro").load("/content/data_source_demo/parquet_to_avro")
# avro_to_text_df.write.mode("overwrite").text("/content/data_source_demo/avro_to_text")
# print("File conversions completed")
```

```
#7
```

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://archive.apache.org/dist/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz
!tar xf spark-3.5.0-bin-hadoop3.tgz
!pip install -q findspark pyspark==3.5.0 lxml fastavro avro-python3
!wget -q https://repo1.maven.org/maven2/com/databricks/spark-xml_2.12/0.16.0/spark-xml_2.12-0.16.0.j

import os, findspark
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.5.0-bin-hadoop3"
findspark.init()

from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").appName("SparkDataSourceAPI").getOrCreate()
```

```
# Create JSON text file
with open("sample.json", "w") as f:
    f.write('{"id":1,"name":"Abhishek"}\n')
    f.write('{"id":2,"name":"Ashwin"}\n')
```

```
df_json = spark.read.json("sample.json")
df_json.show()
# Save sample DataFrame to CSV
df = spark.createDataFrame([(1,"A"),(2,"B")],["id","name"])
df.write.mode("overwrite").csv("sample_csv", header=True)

# Read CSV
df_csv = spark.read.csv("sample_csv", header=True, inferSchema=True)
df_csv.show()
df.write.mode("overwrite").json("sample_json")
df2 = spark.read.json("sample_json")
df2.show()
df.write.mode("overwrite").parquet("sample_parquet")
df_parquet = spark.read.parquet("sample_parquet")
df_parquet.show()

df_xml = spark.createDataFrame([(1,"Alex"),(2,"John")],["id","name"])
df_xml.write.mode("overwrite").format("com.databricks.spark.xml").option("rootTag","people").option(

# Read XML
df_read_xml = spark.read.format("com.databricks.spark.xml").option("rowTag","person").load("sample_x
df_read_xml.show()
#read avro
df.write.mode("overwrite").format("avro").save("sample_avro")
df_avro = spark.read.format("avro").load("sample_avro")
df_avro.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipython-input-4292878313.py in <cell line: 0>()
      4     f.write('{"id":2,"name":"Ashwin"}\n')
      -
----> 6 df_json = spark.read.json("sample.json")
      7 df_json.show()
      8 # Save sample DataFrame to CSV

NameError: name 'spark' is not defined
```

Next steps: Explain error

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://archive.apache.org/dist/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz
!tar xf spark-3.5.0-bin-hadoop3.tgz
!pip install -q findspark pyspark==3.5.0 lxml fastavro avro-python3

import os, findspark
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.5.0-bin-hadoop3"
findspark.init()

from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").appName("SparkDataSourceAPI").getOrCreate()
```

```
# Create JSON text file
with open("sample.json", "w") as f:
    f.write('{"id":1,"name":"Abhishek"}\n')
```

```
    f.write('{"id":2,"name":"Ashwin"}\n')

df_json = spark.read.json("sample.json")
df_json.show()
# Save sample DataFrame to CSV
df = spark.createDataFrame([(1,"A"),(2,"B")],["id","name"])
df.write.mode("overwrite").csv("sample_csv", header=True)

# Read CSV
df_csv = spark.read.csv("sample_csv", header=True, inferSchema=True)
df_csv.show()
df.write.mode("overwrite").json("sample_json")
df2 = spark.read.json("sample_json")
df2.show()
df.write.mode("overwrite").parquet("sample_parquet")
df_parquet = spark.read.parquet("sample_parquet")
df_parquet.show()
!wget -q https://repo1.maven.org/maven2/com/databricks/spark-xml_2.12/0.16.0/spark-xml_2.12-0.16.0.j

df_xml = spark.createDataFrame([(1,"Alex"),(2,"John")],["id","name"])
df_xml.write.mode("overwrite").format("com.databricks.spark.xml").option("rootTag","people").option(

# Read XML
df_read_xml = spark.read.format("com.databricks.spark.xml").option("rowTag","person").load("sample_x
df_read_xml.show()
df.write.mode("overwrite").format("avro").save("sample_avro")
df_avro = spark.read.format("avro").load("sample_avro")
df_avro.show()
```

Next steps: Explain error

```
+---+--------+
| id|    name|
+---+--------+
|  1|Abhishek|
|  2|  Ashwin|
+---+--------+

+---+----+
| id|name|
+---+----+
|  1|   A|
|  2|   B|
+---+----+

+---+----+
| id|name|
+---+----+
|  2|   B|
```

```
# Example template (will not run in Colab)
df_hbase = spark.read.format("org.apache.hadoop.hbase.spark") \
    .option("hbase.table","my_table") \
    .option("hbase.config.resources","/etc/hbase/conf/hbase-site.xml") \
    .load()

df_hbase.show()

# Writing back
df.write.format("org.apache.hadoop.hbase.spark") \
    .option("hbase.table","my_table") \
    .save()
df.write.mode("overwrite").orc("sample_orc")
df_orc = spark.read.orc("sample_orc")
df_orc.show()
# Create binary file (image substitute)
with open("sample.bin", "wb") as f:
    f.write(b"HelloSpark")

df_bin = spark.read.format("binaryFile").load("sample.bin")
df_bin.show(truncate=False)
```

```
--> 326                 raise Py4JJavaError(
```

```
# Run only once per session
!apt-get install openjdk-11-jdk-headless -qq > /dev/null
!wget -q https://downloads.apache.org/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz
!tar xf spark-3.5.0-bin-hadoop3.tgz
!pip install -q findspark
!pip install -q pyspark

import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.5.0-bin-hadoop3"

import findspark
findspark.init()

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

spark = SparkSession.builder.appName("Practical8_Streaming").getOrCreate()
```

```
# 1️⃣ Streaming JSON from Directory
import shutil, os
os.makedirs("/content/stream_json_dir", exist_ok=True)

# Simulate streaming by adding JSON files
sample_data = [{"name":"Alice","salary":3000},{"name":"Bob","salary":4000}]
import json
with open("/content/stream_json_dir/sample1.json","w") as f:
    json.dump(sample_data,f)

# Read streaming JSON
json_stream_df = spark.readStream.schema(StructType([
    StructField("name", StringType(), True),
    StructField("salary", IntegerType(), True)
])).json("/content/stream_json_dir")

# Output to console
query = json_stream_df.writeStream.outputMode("append").format("console").start()
# Allow some time to simulate streaming
import time; time.sleep(5)
query.stop()

# 2️⃣ Streaming from TCP Socket
# Start TCP server in terminal: `nc -lk 9999` (cannot be done directly in Colab)
# Simulate TCP streaming (if running locally)
# tcp_stream_df = spark.readStream.format("socket").option("host","localhost").option("port",9999).l
# tcp_stream_df.writeStream.outputMode("append").format("console").start().awaitTermination()
```

```
# 1️⃣ Reading Kafka messages (JSON format)
kafka_json_df = spark.read.format("kafka")\
    .option("kafka.bootstrap.servers","localhost:9092")\
    .option("subscribe","json_topic")\
    .load()

# Convert value (bytes) to string & parse JSON
json_parsed_df = kafka_json_df.selectExpr("CAST(value AS STRING) as json_str")\
    .select(from_json(col("json_str"), StructType([
        StructField("name", StringType()),
```

```
        StructField("salary", IntegerType())
    ])).alias("data")).select("data.*")

# 2️⃣ Kafka messages in Avro format
# Requires from_avro/to_avro functions & schema registry
# from pyspark.sql.avro.functions import from_avro, to_avro
# avro_schema = '{"type":"record","name":"employee","fields":[{"name":"name","type":"string"},{"name
# avro_df = kafka_df.select(from_avro(col("value"), avro_schema).alias("data")).select("data.*")

# 3️⃣ Batch Processing from Kafka Data Source
# kafka_batch_df = spark.read.format("kafka")\
#     .option("kafka.bootstrap.servers","localhost:9092")\
#     .option("subscribe","json_topic")\
#     .load()
# kafka_batch_df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)").show()
```

```
---------------------------------------------------------------------------
AnalysisException                         Traceback (most recent call last)
/tmp/ipython-input-3464024954.py in <cell line: 0>()
      3     .option("kafka.bootstrap.servers","localhost:9092")\
      4     .option("subscribe","json_topic")\
----> 5     .load()
      6
      7 # Convert value (bytes) to string & parse JSON

 2 frames
/usr/local/lib/python3.12/dist-packages/pyspark/errors/exceptions/captured.py in deco(*a, **kw)
    183                 # Hide where the exception came from that shows a non-Pythonic
    184                 # JVM exception message.
--> 185                 raise converted from None
    186             else:
    187                 raise

AnalysisException: Failed to find data source: kafka. Please deploy the application as per the
deployment section of Structured Streaming + Kafka Integration Guide.
```

Next steps: Explain error

Start coding or generate with AI.

```
# Run only once per session
!apt-get install openjdk-11-jdk-headless -qq > /dev/null
!wget -q https://downloads.apache.org/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz
!tar xf spark-3.5.0-bin-hadoop3.tgz
!pip install -q findspark
!pip install -q pyspark

import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.5.0-bin-hadoop3"

import findspark
findspark.init()

from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler, StringIndexer
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
```

```
from pyspark.ml.param import Param, Params

spark = SparkSession.builder.appName("Practical9_MLlib").getOrCreate()
```

```
# Sample data for regression
data = [
    (1, 10, 100),
    (2, 20, 200),
    (3, 30, 300),
    (4, 40, 400),
    (5, 50, 500)
]
columns = ["id", "feature1", "label"]

df = spark.createDataFrame(data, columns)
print("Original DataFrame:")
df.show()

# ===== 1️⃣ Feature Assembling =====
# VectorAssembler is a Transformer (transforms columns to feature vector)
assembler = VectorAssembler(inputCols=["feature1"], outputCol="features")
df_features = assembler.transform(df)
print("After VectorAssembler (Transformer):")
df_features.show()

# ===== 2️⃣ Estimator: LinearRegression =====
# Estimator: Fit model on training data
lr = LinearRegression(featuresCol="features", labelCol="label")
lr_model = lr.fit(df_features)  # Estimator.fit() returns a Transformer (Model)
print("Linear Regression Model coefficients:", lr_model.coefficients)
print("Linear Regression Model intercept:", lr_model.intercept)

# Predict using the model (Transformer)
predictions = lr_model.transform(df_features)
print("Predictions using Transformer (Model):")
predictions.select("id","feature1","label","prediction").show()

# ===== 3️⃣ Using Param =====
# Check default params of LinearRegression
print("LinearRegression parameters:")
print(lr.extractParamMap())

# Set params using Param
lr_new = LinearRegression(featuresCol="features", labelCol="label", maxIter=50, regParam=0.1)
print("New LinearRegression parameters with maxIter and regParam set:")
print(lr_new.extractParamMap())

# ===== 4️⃣ Pipeline Example =====
# Pipeline: sequence of Estimator/Transformer
indexer = StringIndexer(inputCol="label", outputCol="label_index")  # Transformer
pipeline = Pipeline(stages=[assembler, lr])
pipeline_model = pipeline.fit(df)
pred_pipeline = pipeline_model.transform(df)
print("Pipeline Predictions:")
pred_pipeline.show()
```

```
Start coding or generate with AI.
```