
Module 1: Setup & SparkSession Initialization

Tasks:

- Install and configure PySpark in your local system or Colab.
- Initialize Spark with:

```
spark = SparkSession.builder \
    .appName("BotCampus PySpark Practice") \
    .master("local[*]") \
    .getOrCreate()
```

- Create a DataFrame from:

```
data = [
    ("Anjali", "Bangalore", 24),
    ("Ravi", "Hyderabad", 28),
    ("Kavya", "Delhi", 22),
    ("Meena", "Chennai", 25),
    ("Arjun", "Mumbai", 30)
]
columns = ["name", "city", "age"]
```

- Show schema, explain data types, and convert to RDD.
- Print `.collect()` and `df.rdd.map()` output.

Module 2: RDDs & Transformations

Scenario: You received app feedback from users in free-text.

```
feedback = spark.sparkContext.parallelize([
    "Ravi from Bangalore loved the delivery",
    "Meena from Hyderabad had a late order",
    "Ajay from Pune liked the service",
    "Anjali from Delhi faced UI issues",
    "Rohit from Mumbai gave positive feedback"
])
```

Tasks:

- Split each line into words (`flatMap`).
- Remove stop words (`from`, `the`, etc.).
- Count each word frequency using `reduceByKey`.
- Find top 3 most frequent non-stop words.

Module 3: DataFrames & Transformation (With Joins)

DataFrames:

```
students = [
    ("Amit", "10-A", 89),
    ("Kavya", "10-B", 92),
    ("Anjali", "10-A", 78),
    ("Rohit", "10-B", 85),
    ("Sneha", "10-C", 80)
]
columns = ["name", "section", "marks"]

attendance = [
    ("Amit", 24),
    ("Kavya", 22),
    ("Anjali", 20),
    ("Rohit", 25),
    ("Sneha", 19)
]
columns2 = ["name", "days_present"]
```

Tasks:

- Join both DataFrames on `name`.
- Create a new column: `attendance_rate = days_present / 25`.
- Grade students using `when`:
  - A: >90, B: 80–90, C: <80.
- Filter students with good grades but poor attendance (<80%).

📁 Module 4: Ingest CSV & JSON, Save to Parquet

Tasks:

1. Ingest CSV:

   ```
   emp_id,name,dept,city,salary
   101,Anil,IT,Bangalore,80000
   102,Kiran,HR,Mumbai,65000
   103,Deepa,Finance,Chennai,72000
   ```

2. Ingest JSON:

   ```
   {
     "id": 201,
     "name": "Nandini",
     "contact": {
       "email": "nandi@example.com",
       "city": "Hyderabad"
     },
     "skills": ["Python", "Spark", "SQL"]
   }
   ```

Tasks:

- Read both formats into DataFrames.

- Flatten nested JSON using `select`, `col`, `alias`, `explode`.
- Save both as Parquet files partitioned by city.

Module 5: Spark SQL with Temp Views

Tasks:

- Register the `students` DataFrame as `students_view`.
- Write and run the following queries:

```
-- a) Average marks per section
-- b) Top scorer in each section
-- c) Count of students in each grade category
-- d) Students with marks above class average
-- e) Attendance-adjusted performance
```

Module 6: Partitioned Data & Incremental Loading

Step 1: Full Load

```
students_df.write.partitionBy("section").parquet("output/students/")
```

Step 2: Incremental Load

```
incremental = [("Tejas", "10-A", 91)]
df_inc = spark.createDataFrame(incremental, ["name", "section", "marks"])
df_inc.write.mode("append").partitionBy("section").parquet("output/students/")
```

Tasks:

- List files in `output/students/` using Python.
- Read only partition `10-A` and list students.
- Compare before/after counts for section `10-A`.

Module 7: ETL Pipeline – End to End

Given Raw Data (CSV):

```
emp_id,name,dept,salary,bonus
1,Arjun,IT,75000,5000
2,Kavya,HR,62000,
3,Sneha,Finance,68000,4000
4,Ramesh,Sales,58000,
```

Tasks:

- Load CSV with inferred schema.

- Fill null bonuses with `2000`.

- Create `total_ctc = salary + bonus`.

- Filter employees with `total_ctc > 65000`.

- Save result in:

- JSON format.
- Parquet format partitioned by department.