---

1. PySpark Setup & Initialization

Exercise 1.1 – Setup Spark:

- Initialize SparkSession with:

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

Exercise 1.2 – Load starter data:

```
data = [("Ananya", "Bangalore", 24),
        ("Ravi", "Hyderabad", 28),
        ("Kavya", "Delhi", 22),
        ("Meena", "Chennai", 25)]

columns = ["name", "city", "age"]
df = spark.createDataFrame(data, columns)
df.show()
```

---

2. RDDs & Transformations

Exercise 2.1 – Create RDD from feedback:

```
feedback = spark.sparkContext.parallelize([
    "Ravi from Bangalore loved the mobile app",
    "Meena from Delhi reported poor response time",
    "Ajay from Pune liked the delivery speed",
    "Ananya from Hyderabad had an issue with UI",
    "Rohit from Mumbai gave positive feedback"
])
```

- **Tasks**:
  - Count total number of words.
  - Find top 3 most common words.
  - Remove stop words (`from` , `with` , `the` , etc.).
  - Create a dictionary of word → count.

---

3. DataFrames – Transformations

Exercise 3.1 – Create `exam_scores` DataFrame:

```
scores = [
    ("Ravi", "Math", 88),
    ("Ananya", "Science", 92),
    ("Kavya", "English", 79),
    ("Ravi", "English", 67),
    ("Neha", "Math", 94),
    ("Meena", "Science", 85)
]
columns = ["name", "subject", "score"]
df_scores = spark.createDataFrame(scores, columns)
```

Tasks:

- Add grade column (`>=90` → A, `80-89` → B, `70-79` → C, else D).
- Group by subject, find average score.
- Use `when` and `otherwise` to classify subject difficulty (`Math/Science` = Difficult).
- Rank students per subject using Window function.
- Apply UDF to format names (e.g., make all uppercase).

---

4. Ingest CSV & JSON – Save to Parquet

Dataset 1: CSV file: `students.csv`

```
id,name,department,city,salary
1,Amit,IT,Bangalore,78000
2,Kavya,HR,Chennai,62000
3,Arjun,Finance,Hyderabad,55000
```

Dataset 2: JSON file `employee_nested.json`

```
[
  {
    "id": 101,
    "name": "Sneha",
    "address": {
      "city": "Mumbai",
      "pincode": 400001
    },
    "skills": ["Python", "Spark"]
  }
]
```

Tasks:

- Load both datasets into PySpark.
- Print schema and infer nested structure.
- Flatten the JSON (use `explode` , `select` , `alias`).
- Convert both to Parquet and write to `/tmp/output` .

---

5. Spark SQL – Temp Views & Queries

Exercise 5.1 Create view from exam scores and run:

```
-- a) Top scorer per subject
-- b) Count of students per grade
-- c) Students with multiple subjects
-- d) Subjects with average score above 85
```

Exercise 5.2 Create another DataFrame `attendance(name, days_present)` and:

- Join with scores
- Calculate attendance-adjusted grade:

  > *If days_present < 20 → downgrade grade by one level*

---

6. Partitioned Load (Full + Incremental)

Initial Load:

```
df_scores.write.partitionBy("subject").parquet("/tmp/scores/")
```

Incremental Load:

```
incremental = [("Meena", "Math", 93)]
df_inc = spark.createDataFrame(incremental, columns)
df_inc.write.mode("append").partitionBy("subject").parquet("/tmp/scores/")
```

Task:

- List all folders inside `/tmp/scores/`
- Read only `Math` partition and display all entries.

---

7. ETL: Clean, Transform, Load

Raw CSV:

```
emp_id,name,dept,salary,bonus
1,Arjun,IT,78000,5000
2,Kavya,HR,62000,
3,Sneha,Finance,55000,3000
```

Tasks:

- Load data with header.
- Fill missing bonus with 2000.
- Calculate `total_ctc = salary + bonus`.
- Filter where total_ctc > 60,000.
- Save final DataFrame to Parquet and JSON.
