PYSPARK SCHEMA RECOVERY

StructType • StructField • Complex Types • Data Fixing

- You are given **corrupted datasets**
- Data comes from APIs / logs / CSVs / JSON-like sources
- Your job is to **fix the data using explicit schemas**
- Do **not** rely on schema inference
- Use:
  - `StructType`
  - `StructField`
  - Casting
  - Null handling
- Goal: **produce a clean, analyzable DataFrame**

DATASET 1 — USER PROFILE API (CORRUPTED TYPES)

Raw data (as received from API)

```
raw_users = [
    ("U001","Amit","29","Hyderabad","50000"),
    ("U002","Neha","Thirty Two","Delhi","62000"),
    ("U003","Ravi",None,"Bangalore","45k"),
    ("U004","Pooja","28","Mumbai",58000),
    ("U005",None,"31","Chennai","")
]
```

Problems intentionally introduced

- Age as string
- Non-numeric age values
- Salary mixed as int, string, shorthand (`45k`)
- Missing names
- Empty salary

Exercises

1. Design a **StructType schema** for this data
2. Load the data using the schema
3. Identify records that fail type conversion
4. Convert age to integer safely
5. Normalize salary into integer (handle `k`)
6. Replace missing names with `"UNKNOWN"`
7. Drop records where age cannot be recovered
8. Produce a final clean DataFrame

DATASET 2 — E-COMMERCE ORDERS (ARRAY CORRUPTION)

Raw data

```
raw_orders = [
    ("O001","U001","Laptop,Mobile,Tablet",75000),
    ("O002","U002",["Mobile","Tablet"],32000),
    ("O003","U003","Laptop",72000),
    ("O004","U004",None,25000),
    ("O005","U005","Laptop|Mobile",68000)
]
```

Problems intentionally introduced

- Items sometimes string, sometimes array

- Different delimiters (`,` and `|`)
- Single item as string
- Null items

Exercises

1. Define a schema with `ArrayType`
2. Normalize all item values into arrays
3. Handle multiple delimiters
4. Replace null items with empty arrays
5. Explode items into one row per item
6. Count frequency of each item
7. Identify orders with more than 2 items

DATASET 3 — DEVICE USAGE (MAP CORRUPTION)

Raw data

```
raw_devices = [
    ("U001",{"mobile":120,"laptop":300}),
    ("U002","mobile:200,tablet:100"),
    ("U003",{"desktop":"400","mobile":"150"}),
    ("U004",None),
    ("U005","laptop-250")
]
```

Problems intentionally introduced

- Map sometimes string
- Inconsistent delimiters
- Values as strings
- Missing maps

Exercises

1. Design a `MapType(StringType, IntegerType)` schema
2. Parse string maps into proper maps
3. Convert all usage values to integers
4. Handle malformed key-value pairs
5. Replace missing maps with empty maps
6. Extract mobile usage safely
7. Identify users with usage above a threshold

DATASET 4 — NESTED ADDRESS JSON (BROKEN STRUCTS)

Raw data

```
raw_profiles = [
    ("U001","Hyderabad,Telangana,500081"),
    ("U002",{"city":"Delhi","state":"Delhi","pincode":"110001"}),
    ("U003",("Bangalore","Karnataka",560001)),
    ("U004","Mumbai,MH"),
    ("U005",None)
]
```

Problems intentionally introduced

- Address sometimes string, map, tuple
- Missing fields
- Pincode as string
- Partial address

Exercises

1. Design a nested `StructType` for address
2. Normalize all address formats into struct
3. Extract city, state, pincode safely

4. Set default pincode when missing
5. Drop irrecoverable records
6. Flatten the struct into columns

DATASET 5 — TRANSACTION LOGS (MIXED DATES & NUMBERS)

Raw data

```
raw_transactions = [
    ("T001","2024-01-05","45000"),
    ("T002","05/01/2024",52000),
    ("T003","Jan 06 2024","Thirty Thousand"),
    ("T004",None,38000),
    ("T005","2024/01/07","42000")
]
```

Problems intentionally introduced

- Multiple date formats
- Amount as words
- Missing dates
- Inconsistent separators

Exercises

1. Design schema using `StructType`
2. Normalize all dates into `DateType`
3. Convert amount into integer
4. Identify unrecoverable records
5. Separate valid vs invalid transactions
6. Produce a clean transactions DataFrame
