## ****Case Study: Big Data Processing with PySpark****

**Objective:**  
This case study explores how PySpark can be used to perform large-scale data processing and analysis on a **dummy dataset**.

## ****Scenario:****

A **retail company** wants to analyze customer transactions to improve business performance. The dataset contains **1 million records** (dummy data). You need to perform **20 different PySpark tasks**.

### ****Generating Dummy Retail Dataset (****retail\_data.csv****)****

This script will create a dataset with the following columns:

* Transaction\_ID (Unique ID for each transaction)
* Customer\_ID (Unique customer identifier)
* Product\_Category (Random product category)
* Quantity (Random number of items purchased)
* Unit\_Price (Random unit price)
* Total\_Amount (Quantity × Unit Price)
* Payment\_Type (Random payment method)
* Order\_Date (Random order date in the past 2 years)
* City (Random customer city)
* Country (Random country)

### ****Python Script to Generate the Data****

import pandas as pd

import numpy as np

from faker import Faker

import random

# Initialize Faker for generating random names, cities, etc.

fake = Faker()

# Define constants

NUM\_RECORDS = 1000000 # 1 million records

PRODUCT\_CATEGORIES = ['Electronics', 'Clothing', 'Home & Kitchen', 'Sports', 'Toys', 'Books', 'Beauty']

PAYMENT\_METHODS = ['Credit Card', 'Debit Card', 'PayPal', 'Net Banking', 'Cash on Delivery']

CITIES = ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Phoenix', 'San Francisco', 'Seattle', 'Miami']

COUNTRIES = ['USA', 'Canada', 'UK', 'Germany', 'France', 'Australia']

# Generate random data

data = {

'Transaction\_ID': [fake.uuid4()[:8] for \_ in range(NUM\_RECORDS)], # Shortened UUID

'Customer\_ID': [fake.uuid4()[:12] for \_ in range(NUM\_RECORDS)], # Shortened UUID

'Product\_Category': [random.choice(PRODUCT\_CATEGORIES) for \_ in range(NUM\_RECORDS)],

'Quantity': np.random.randint(1, 10, NUM\_RECORDS),

'Unit\_Price': np.round(np.random.uniform(5, 500), 2), # Price between $5 and $500

'Payment\_Type': [random.choice(PAYMENT\_METHODS) for \_ in range(NUM\_RECORDS)],

'Order\_Date': [fake.date\_between(start\_date='-2y', end\_date='today') for \_ in range(NUM\_RECORDS)],

'City': [random.choice(CITIES) for \_ in range(NUM\_RECORDS)],

'Country': [random.choice(COUNTRIES) for \_ in range(NUM\_RECORDS)]

}

# Convert to Pandas DataFrame

df = pd.DataFrame(data)

# Calculate Total\_Amount

df['Total\_Amount'] = df['Quantity'] \* df['Unit\_Price']

# Save to CSV

df.to\_csv("retail\_data.csv", index=False)

print("Dummy dataset 'retail\_data.csv' generated successfully!")

## ****Setting Up PySpark****

from pyspark.sql import SparkSession

# Initialize Spark Session

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

# Load Dataset

df = spark.read.csv("retail\_data.csv", header=True, inferSchema=True)

## ****PySpark Tasks****

### ****1. Display the first 10 rows****

### ****2. Print the schema of the dataset****

### ****3. Get the total number of transactions****

### ****4. Find the distinct product categories****

### ****5. Filter transactions where Quantity > 10****

### ****6. Find the most popular payment type****

### ****7. Calculate total revenue****

### ****8. Find the top 5 cities with the highest sales****

### ****9. Find the average transaction value****

### ****10. Find the number of transactions per day****

### ****11. Extract year from the**** Order\_Date ****column****

### ****12. Find the total sales per year****

### ****13. Find the highest spending customer****

### ****14. Create a new column ‘Discounted\_Amount’ (Apply a 10% discount for transactions > $500)****

### ****15. Convert**** Order\_Date ****column to a proper date format****

### ****16. Find the top-selling product category****

### ****17. Find the customer with the highest number of orders****

### ****18. Find the total sales per country****

### ****19. Count how many customers used each payment method****

### ****20. Save the transformed data as a Parquet file****