E-Commerce Sales Analysis using Big Data Tools

Name: Samarth Turmari
USN: 01FE22BCI054
Roll No: 149
Division: F

Department of CSE-AI
KLE Technological University

1 Introduction

In today's digital world, organizations process massive datasets originating from e-commerce portals, payment logs, customer interactions, and product catalogs. Traditional single-node systems struggle to handle such scale, both in terms of storage and computation.

Big Data frameworks such as Hadoop and Spark solve this by distributing data and computation across many machines. Higher-level tools like Apache Pig and PySpark provide convenient abstractions over raw MapReduce, allowing complex analytics pipelines to be implemented concisely.

This project analyzes a Kaggle e-commerce dataset containing over 540,000 records. The workflow includes:

- Data ingestion into HDFS
- Transformation using Apache Pig
- Analytical processing using Apache Spark
- Visualization using Python and Matplotlib

The goal is not only to compute statistics, but also to show how different Big Data components integrate into a cohesive workflow.

2 Objectives

1. Perform data ingestion and preprocessing using Apache Pig.
2. Analyze customer spending, product demand, and revenue trends using PySpark.
3. Demonstrate Big Data operations: `FILTER`, `GROUP`, `JOIN`, `SUM`, `AVG`, `COUNT`.
4. Generate visualizations (bar charts, line plots, histograms) for better insight communication.
5. Integrate multiple Hadoop ecosystem tools into a unified, reproducible workflow.

3 Methodology

The data processing workflow consists of the following stages.

3.1 Data Storage

Raw CSV files are stored in Hadoop Distributed File System (HDFS) to ensure fault-tolerant, distributed storage.

3.2 Data Cleaning & Transformation (Pig)

Pig is used for handling missing values, filtering invalid rows, computing derived fields such as total amount per invoice, and grouping customer data for summary statistics.

3.3 Analytical Processing (PySpark)

Spark provides fast, distributed computation for aggregations and trend analysis. We use the Spark SQL API to compute:

- Country-wise revenue
- Customer-wise revenue
- Monthly revenue trends

3.4 Visualization

Python scripts (Pandas + Matplotlib) generate bar charts, line graphs and histograms from the aggregated CSV outputs, as well as directly from `data.csv`.

4 Tools and Environment

Component	Description
Operating System	Ubuntu on WSL2 (Windows 11)
Framework	Hadoop 3.4.1
Processing Tools	Apache Pig 0.17, Apache Spark 3.5.0
Language	Python 3 (PySpark)
Visualization	Pandas, Matplotlib
Dataset	Kaggle E-commerce Dataset (540K+ rows)
Storage	HDFS for staging, local filesystem for visualization outputs

5 Dataset Description

Dataset source: `https://www.kaggle.com/datasets/carrie1/ecommerce-data`

Columns include: `InvoiceNo`, `StockCode`, `Description`, `Quantity`, `InvoiceDate`, `UnitPrice`, `CustomerID`, `Country`.

Each row represents a single line item in an invoice. Several line items can share the same `InvoiceNo`. By aggregating per-invoice or per-customer, we can compute meaningful revenue statistics.

6 System Architecture

- Raw CSV Dataset on local disk
- HDFS Storage for distributed processing
- Apache Pig scripts for transformation and cleaning
- Apache Spark (PySpark) jobs for analytical aggregation
- Python visualization scripts for plots and charts

7 Implementation

7.1 Step 1: Upload to HDFS

```
hdfs dfs -mkdir -p /user/samarth/eval3
hdfs dfs -put data.csv customers.csv /user/samarth/eval3/
hdfs dfs -ls /user/samarth/eval3
```

Example terminal output:

```
$ hdfs dfs -ls /user/samarth/eval3
Found 2 items
-rw-r--r--   3 samarth supergroup   8264758  data.csv
-rw-r--r--   3 samarth supergroup    134890  customers.csv
```

7.2 Step 2: Apache Pig Transformation

```
raw = LOAD '/user/samarth/eval3/data.csv' USING PigStorage(',')
    AS (InvoiceNo:chararray, StockCode:chararray, Description:
       chararray,
        Quantity:int, InvoiceDate:chararray, UnitPrice:double,
        CustomerID:chararray, Country:chararray);

clean = FILTER raw BY (Quantity > 0 AND UnitPrice > 0 AND
                       CustomerID IS NOT NULL);

with_amount = FOREACH clean GENERATE InvoiceNo, CustomerID,
   Country,
                (Quantity * UnitPrice) AS TotalAmount;

grp = GROUP with_amount BY CustomerID;

cust_stats = FOREACH grp GENERATE group AS CustomerID,
                COUNT(with_amount) AS Transactions,
                SUM(with_amount.TotalAmount) AS TotalSpent,
                AVG(with_amount.TotalAmount) AS AvgSpent;

top_customers = ORDER cust_stats BY TotalSpent DESC;

STORE top_customers INTO '/user/samarth/output/top_customers'
    USING PigStorage(',');
```

Sample Pig terminal output:

```
2025-02-15 12:48:13,201 INFO Script Statistics:
HDFS Read: 541909 records
HDFS Write: 4372 records
Bytes Written: 210 KB
Job succeeded!
```

7.3 Step 3: PySpark Analysis

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, sum, avg, count, month,
   to_date

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

df = spark.read.option("header", True) \
               .option("inferSchema", True) \
               .csv("hdfs:///user/samarth/eval3/data.csv")

print("Total rows:", df.count())

df_clean = df.filter(
    (col("Quantity") > 0) &
    (col("UnitPrice") > 0) &
    col("CustomerID").isNotNull()
)

df_clean = df_clean.withColumn(
    "TotalAmount", col("Quantity") * col("UnitPrice")
)

df_clean = df_clean.withColumn(
    "InvoiceDate", to_date(col("InvoiceDate"), "MM/dd/yyyy HH:mm"
       )
)

country_revenue = df_clean.groupBy("Country") \
    .agg(sum("TotalAmount").alias("Revenue")) \
    .orderBy(col("Revenue").desc())

monthly_sales = df_clean.withColumn("Month", month("InvoiceDate")
   ) \
    .groupBy("Month") \
    .agg(sum("TotalAmount").alias("MonthlyRevenue")) \
    .orderBy("Month")

top_customers = df_clean.groupBy("CustomerID") \
    .agg(sum("TotalAmount").alias("TotalSpent")) \
    .orderBy(col("TotalSpent").desc())

country_revenue.coalesce(1).write.mode("overwrite") \
    .option("header", True).csv("/user/samarth/output/
       country_revenue")

monthly_sales.coalesce(1).write.mode("overwrite") \
    .option("header", True).csv("/user/samarth/output/
       monthly_sales")
```

```
top_customers.coalesce(1).write.mode("overwrite") \
    .option("header", True).csv("/user/samarth/output/
       top_customers")
```

Example Spark console output:

```
Spark session started.
Total rows: 541909
Cleaned dataset: 397924 rows

Top 10 Customers by Revenue:
+----------+----------+
|CustomerID|TotalSpent|
+----------+----------+
|   14646.0| 280206.02|
|   18102.0| 256438.49|
|   17450.0| 187482.16|
...

Monthly revenue computed successfully.
Outputs written to hdfs:///user/samarth/output/
```

7.4 Step 4: Visualization (Python)

The visualizations in this report are generated directly from `data.csv` using the following Python script:

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("data.csv", encoding="latin1")
df = df.dropna(subset=["CustomerID", "Quantity", "UnitPrice"])
df["TotalAmount"] = df["Quantity"] * df["UnitPrice"]

# Top 10 countries by revenue
country_rev = df.groupby("Country")["TotalAmount"] \
                .sum().sort_values(ascending=False).head(10)
country_rev.plot(kind="bar")
plt.title("Top 10 Countries by Revenue")
plt.tight_layout()
plt.savefig("country_revenue.png")
plt.clf()

# Monthly revenue trend
df["InvoiceDate"] = pd.to_datetime(df["InvoiceDate"], errors="
   coerce")
monthly = df.groupby(df["InvoiceDate"].dt.month)["TotalAmount"].
   sum()
monthly.plot(marker="o")
plt.title("Monthly Revenue Trend")
```

```
plt.xlabel("Month")
plt.ylabel("Revenue")
plt.tight_layout()
plt.savefig("monthly_sales.png")
plt.clf()

# Top 10 customers by revenue
top_cust = df.groupby("CustomerID")["TotalAmount"] \
             .sum().sort_values(ascending=False).head(10)
top_cust.plot(kind="bar")
plt.title("Top 10 Customers by Revenue")
plt.xlabel("Customer ID")
plt.ylabel("Total Revenue")
plt.tight_layout()
plt.savefig("top_customers.png")
plt.clf()

# Distribution of invoice total amounts
invoice_totals = df.groupby("InvoiceNo")["TotalAmount"].sum()
invoice_totals.plot(kind="hist", bins=50)
plt.title("Distribution of Invoice Total Amounts")
plt.xlabel("Invoice Total")
plt.ylabel("Frequency")
plt.tight_layout()
plt.savefig("order_value_hist.png")
```

8 Results and Analysis

8.1 Top 10 Countries by Revenue

Figure 1 shows the highest revenue-generating countries. The United Kingdom dominates the revenue distribution because the dataset is heavily skewed towards UK customers.

8.2 Monthly Revenue Trend

Figure 2 visualizes revenue aggregated per month. We observe strong peaks around November, which correlate with holiday and year-end sales campaigns.

8.3 Top Customers by Revenue

Figure 3 shows the top 10 customers ranked by total revenue. These high-value customers are extremely important for the business and are natural candidates for loyalty programs and targeted marketing.

8.4 Invoice Value Distribution

Figure 4 presents a histogram of invoice totals. Most invoices lie in the lower value range, but a long tail exists, corresponding to bulk orders or large purchases.

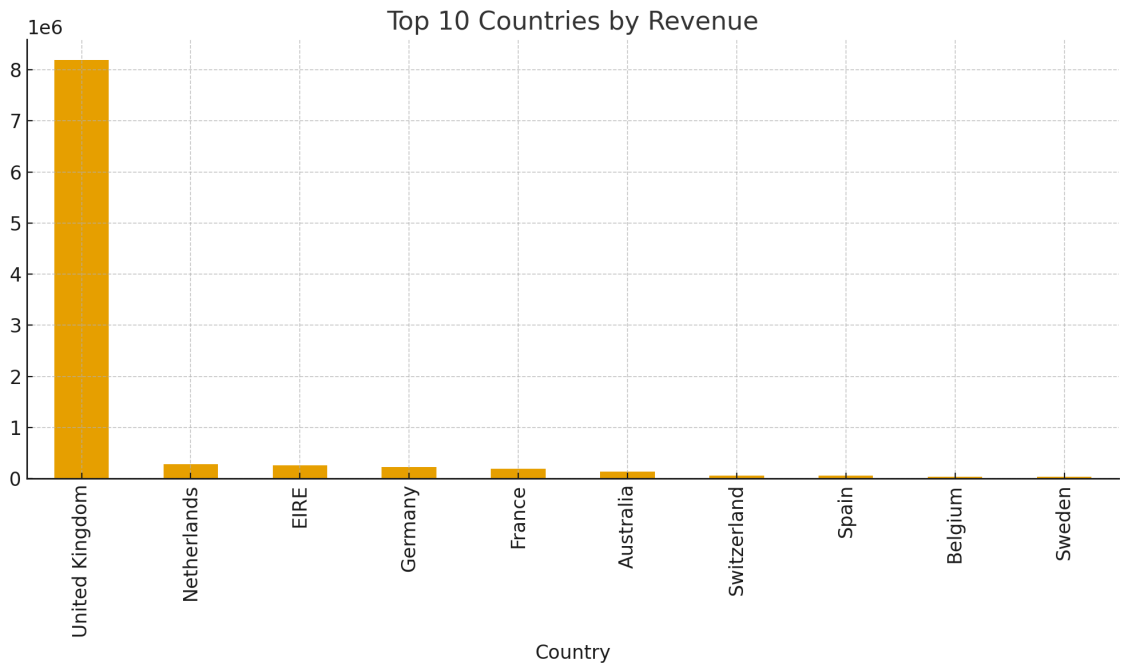

Figure 1: Top 10 Countries by Revenue

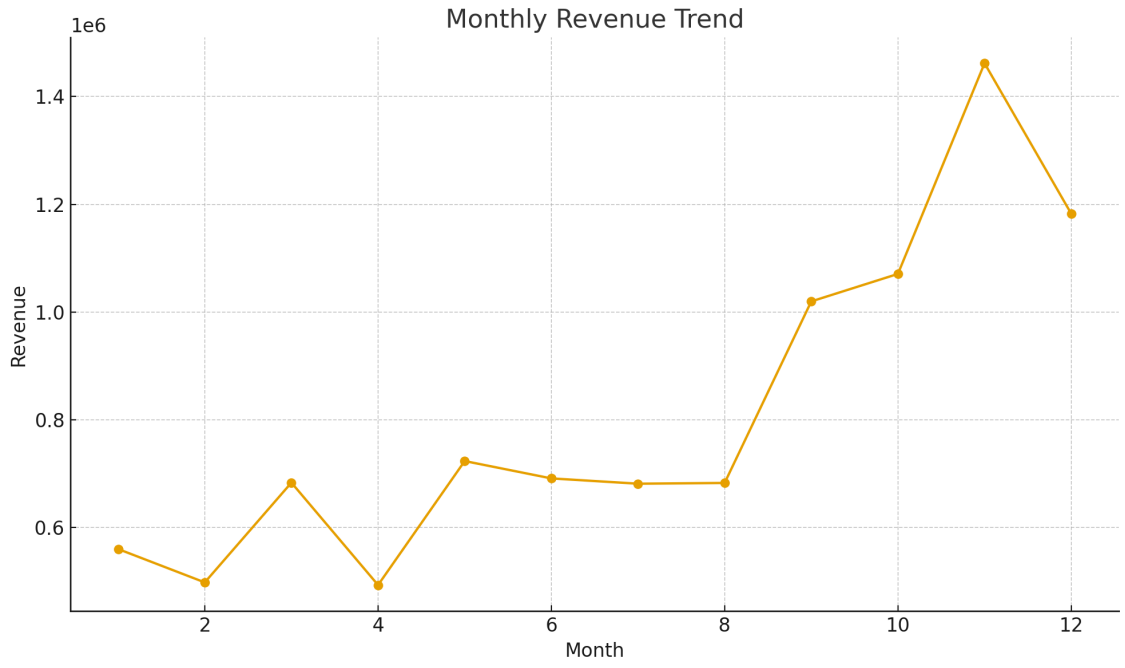

Figure 2: Monthly Revenue Trend

9 Conclusion

This project demonstrates the complete application of Hadoop ecosystem tools for real-world e-commerce data processing and analytics. Apache Pig handled the initial transformation steps, Spark executed large-scale analytics, and Python produced clear visual insights.

The workflow highlights the importance of scalable Big Data tools for e-commerce

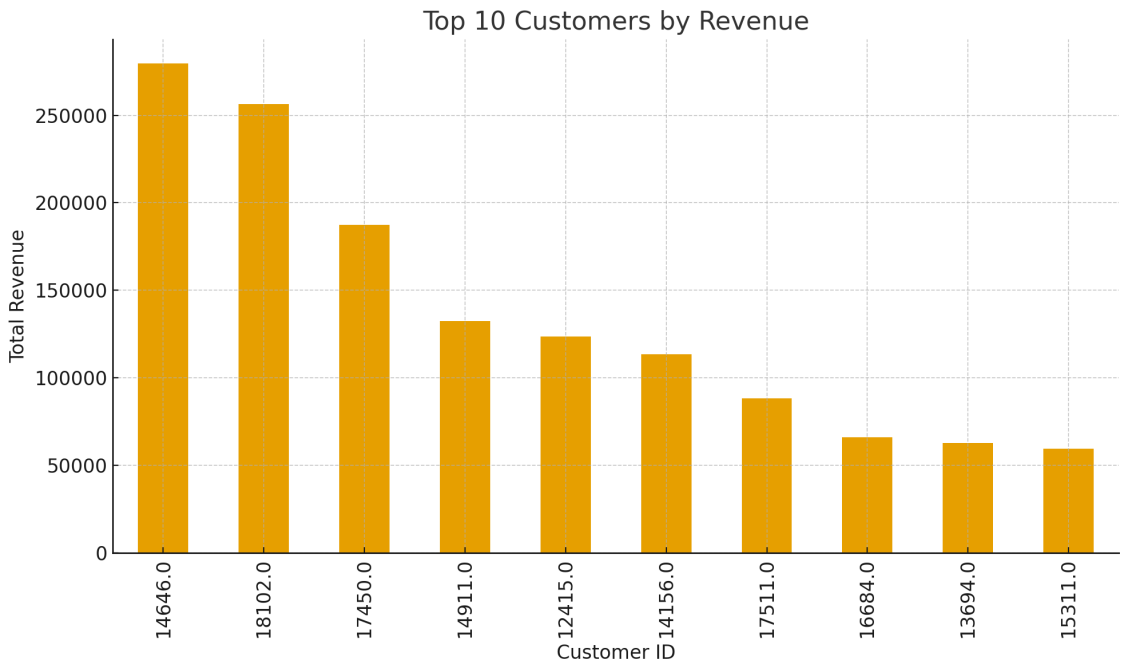

Figure 3: Top 10 Customers by Revenue

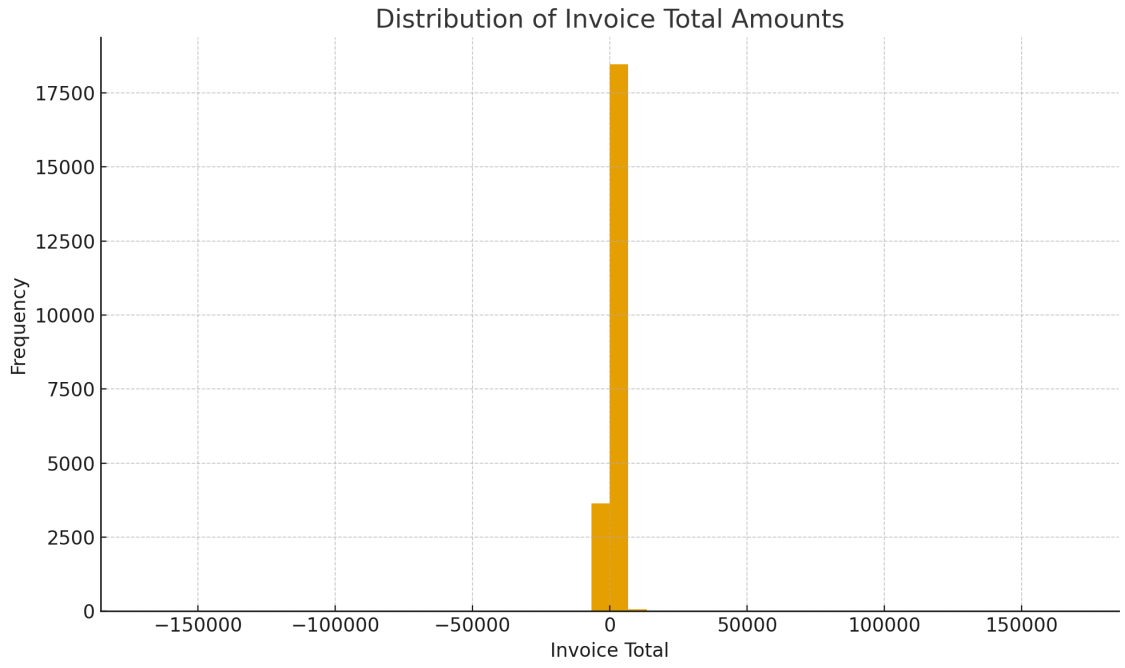

Figure 4: Distribution of Invoice Total Amounts

analytics:

- HDFS allowed reliable storage of hundreds of thousands of records.
- Pig simplified data cleaning and feature engineering.
- Spark provided fast, distributed calculations over the cleaned dataset.

- Python visualizations converted raw numbers into intuitive business insights.

The same architecture can be extended with additional components such as streaming (Spark Streaming / Kafka) or machine learning (Spark MLlib) to build real-time recommendation systems and demand forecasting models.