Retail Sales Data Processing and Analysis using PySpark

Objective:

The goal of this project is to process and analyse retail sales data using PySpark and Spark SQL. Trainees will perform ETL (Extract, Transform, Load) operations, conduct exploratory data analysis (EDA), and generate insights from the data. The project will help trainees gain hands-on experience with PySpark, data cleaning, data transformation, and SQL queries. By following **medallion architecture – Bronze, Silver and Gold layer.**

Project Scope:

1. **Data Extraction**:

   Load three datasets (customers.csv, products.csv, and sales.csv) into PySpark Data Frames.

   Understand the structure and schema of each dataset.

   Code:

   Bronze - Raw Data Ingestion

   Data Extraction: Load three datasets (customers.csv, products.csv, and sales.csv) into PySpark Data Frames. Understand the structure and schema of each dataset.

07:34 AM (18s) 4

```
customers_csv = spark.read.format('csv').option('header',True).option('inferschema',True).load("/FileStore/tables/customers_dataset-1.csv")
sales_csv = spark.read.format('csv').option('header',True).option('inferschema',True).load("/FileStore/tables/sales_dataset-1.csv")
products_csv = spark.read.format('csv').option('header',True).option('inferschema',True).load("/FileStore/tables/products_dataset-1.csv")
```

(6) Spark Jobs

customers_csv: pyspark.sql.dataframe.DataFrame = [customer_id: integer, name: string ... 2 more fields]

products_csv: pyspark.sql.dataframe.DataFrame = [product_id: integer, product_name: string ... 2 more fields]

sales_csv: pyspark.sql.dataframe.DataFrame = [sale_id: integer, date: date ... 5 more fields]

07:35 AM (3s) 5

```
sales_csv.display()
customers_csv.display()
products_csv.display()
```

2 days ago (<1s) 6

```
customers_csv.printSchema()
sales_csv.printSchema()
products_csv.printSchema()
```

07:35 AM (34s) 7

```
# save data in delta by medallion architecture _ Bronze layer
customers_csv.write.format("delta").mode("overwrite").option("overwriteSchema", "true").save("/tmp/delta/
customers_bronze")
sales_csv.write.format("delta").mode("overwrite").option("overwriteSchema", "true").save("/tmp/delta/sales_bronze")
products_csv.write.format("delta").mode("overwrite").option("overwriteSchema", "true").save("/tmp/delta/
products_bronze")
```

(18) Spark Jobs

Results:

	product_id	product_name	category	price
1	201	Laptop	Electronics	1100
2	202	Tablet	Electronics	500
3	203	Phone	Electronics	710
4	204	Monitor	Accessories	300
5	205	Keyboard	Accessories	50
6	206	Mouse	Accessories	30
7	207	Headphones	Accessories	100
8	208	Smartwatch	Electronics	205
9	209	Camera	Electronics	800
10	210	Speaker	Accessories	120
11	211	Charger	Accessories	25
12	212	Hard Drive	Accessories	80
13	213	Flash Drive	Accessories	15
14	214	Printer	Accessories	150
15	215	Scanner	Accessories	130

	customer_id	name	age	city
1	100	Amery	22	Épernay
2	101	Tate	57	Liberia
3	102	Colton	58	İskenderun
4	103	Myles	null	Bogotá
5	104	Bree	41	Gisborne
6	105	Leroy	55	Stockholm
7	106	Aurora	53	Hopefield
8	107	Avye	53	Swellendam
9	108	Diana	24	Cartago
10	109	Octavius	44	Bevel
11	110	Lila	25	Secunda
12	111	Troy	46	Bollnäs
13	112	Geraldine	20	Jaén
14	113	Nerea	36	Borås
15	114	Zelenia	48	Jecheon

	sale_id	date	customer_id	product_id	store_type	quantity	price
1	1	2024-01-10	101	201	Online	2	50
2	2	2024-01-12	102	202	Physical	3	30
3	3	2024-01-15	103	203	Online	1	20
4	4	2024-02-10	101	201	Physical	4	50
5	5	2024-02-12	104	204	Online	0	60
6	6	2024-02-15	102	202	Physical	-1	30
7	7	2024-02-18	105	205	Online	2	70
8	8	2024-02-20	106	206	Physical	5	30
9	9	2024-03-01	107	207	Online	3	120
10	10	2024-03-03	108	208	Physical	1	200
11	11	2024-03-10	109	209	Online	4	800
12	12	2024-03-15	110	210	Physical	2	25
13	13	2024-04-01	111	211	Online	3	15
14	14	2024-04-05	112	212	Physical	1	150
15	15	2024-04-10	113	213	Online	0	130

2. **Data Transformation:**

- Handle missing values in the customers.csv dataset (e.g., fill nulls in city with "Unknown" and age with the average age).
- Drop rows with negative quantity or price in the sales.csv dataset.
- Join the three datasets (customers, products, and sales) into a single Data Frame using customer_id and product_id.
- Enrich the data by calculating a new column total_revenue (quantity * price) and deriving a sale_month column from the date.

Code:

07:36 AM (1s) 9 Python

```python
avg_age = customers_csv.select(avg(col("age").cast("double"))).first()[0]
customers_csv = customers_csv.withColumn("city", when(col("city").isNull(), "Unknown").otherwise(col("city"))).
withColumn("age", when(col("age").isNull(), avg_age).otherwise(col("age")))
```

(2) Spark Jobs

customers_csv: pyspark.sql.dataframe.DataFrame = [customer_id: integer, name: string ... 2 more fields]

07:36 AM (1s) 10

```python
sales_csv = sales_csv.filter((col("quantity") > 0) & (col("price") > 0)).withColumnRenamed('price','price sold')
sales_csv.display()
```

07:36 AM (3s) 11

```python
joined_df = sales_csv \
    .join(customers_csv, on="customer_id", how="inner") \
    .join(products_csv, on="product_id", how="inner") \
    .withColumn("total_revenue", col("quantity") * col("price sold")) \
    .withColumn("sale_month", month(col("date")))
joined_df.display()
```

12

```python
# Saving data in Delta - Silver Layer
joined_df.write \
    .format("delta") \
    .mode("overwrite") \
    .option("overwriteSchema", "true") \
    .save("/tmp/delta/joinedretailsilver")
```

Result:

	sale_id	date	customer_id	product_id	store_type	quantity	price sold
1	1	2024-01-10	101	201	Online	2	50
2	2	2024-01-12	102	202	Physical	3	30
3	3	2024-01-15	103	203	Online	1	20
4	4	2024-02-10	101	201	Physical	4	50
5	7	2024-02-18	105	205	Online	2	70
6	8	2024-02-20	106	206	Physical	5	30
7	9	2024-03-01	107	207	Online	3	120
8	10	2024-03-03	108	208	Physical	1	200
9	11	2024-03-10	109	209	Online	4	800
10	12	2024-03-15	110	210	Physical	2	25
11	13	2024-04-01	111	211	Online	3	15
12	14	2024-04-05	112	212	Physical	1	150
13	17	2024-04-15	115	215	Online	2	60
14	18	2024-05-01	116	216	Physical	6	500

	product_id	customer_id	sale_id	date	store_type	quantity	price sold
1	201	101	1	2024-01-10	Online	2	50
2	202	102	2	2024-01-12	Physical	3	30
3	203	103	3	2024-01-15	Online	1	20
4	201	101	4	2024-02-10	Physical	4	50
5	205	105	7	2024-02-18	Online	2	70
6	206	106	8	2024-02-20	Physical	5	30
7	207	107	9	2024-03-01	Online	3	120
8	208	108	10	2024-03-03	Physical	1	200
9	209	109	11	2024-03-10	Online	4	800
10	210	110	12	2024-03-15	Physical	2	25
11	211	111	13	2024-04-01	Online	3	15
12	212	112	14	2024-04-05	Physical	1	150
13	215	115	17	2024-04-15	Online	2	60
14							

	age	city	product_name	category	price	total_revenue	sale_month
1	57	Liberia	Laptop	Electronics	1100	100	1
2	58	İskenderun	Tablet	Electronics	500	90	1
3	36842105263158	Bogotá	Phone	Electronics	710	20	1
4	57	Liberia	Laptop	Electronics	1100	200	2
5	55	Stockholm	Keyboard	Accessories	50	140	2
6	53	Hopefield	Mouse	Accessories	30	150	2
7	53	Swellendam	Headphones	Accessories	100	360	3
8	24	Cartago	Smartwatch	Electronics	205	200	3
9	44	Bevel	Camera	Electronics	800	3200	3
10	25	Secunda	Speaker	Accessories	120	50	3
11	46	Bollnäs	Charger	Accessories	25	45	4
12	20	Jaén	Hard Drive	Accessories	80	150	4
13	48	Patarrá	Scanner	Accessories	130	120	4

3. Data Filtering:

- Filter the dataset to keep only sales with quantity > 1 and total_revenue > 50.

4. **Data Loading:**

- Save the transformed data as a Parquet file for further analysis.

5. **Exploratory Data Analysis (EDA):**

- Perform basic insights such as calculating total revenue, identifying top products by revenue, and analysing total sales by store type.
- Conduct customer analysis to find the top 10 customers contributing the most to revenue and calculate the average age of customers by product category.
- Analyse monthly sales trends and revenue contribution by product category over time.

Code:

07:41 AM (<1s) 14

```
joined_final = joined_df.filter((col("quantity") > 1) & (col("total_revenue") > 50))
```

joined_final: pyspark.sql.dataframe.DataFrame = [product_id: integer, customer_id: integer ... 13 more fields]

07:41 AM (8s) 15

```
monthly_sales = joined_final.groupBy("sale_month").agg(sum("total_revenue").alias("monthly_revenue")).orderBy
('sale_month')
top_products = joined_final.groupBy("product_name").agg(sum("total_revenue").alias("revenue")).orderBy(desc
("revenue"))
top_customers = joined_final.groupBy("customer_id").agg(sum("total_revenue").alias("revenue")).orderBy(desc
("revenue")).limit(10)
store_type_sales = joined_final.groupBy('store_type').agg(sum('total_revenue').alias('total_sales'))
monthly_sales.display()
top_products.display()
top_customers.display()
store_type_sales.display()
```

07:41 AM (2s) 16 Python

```
joined_final.groupBy('category').agg(avg('age').alias('avg_age')).orderBy('avg_age', ascending=False).display()
```

(4) Spark Jobs

07:41 AM (2s) 17 Python

```
category_over_time = joined_final.groupBy('sale_month', 'category').agg(sum('total_revenue').alias('total_revenue')).
orderBy('sale_month', 'category')

category_over_time.display()
```

07:45 AM (21s) 27

```
# save data in delta - Gold layer
store_type_sales.write.format("delta").mode("overwrite").option("overwriteSchema", "true").save("/tmp/delta/
store_sales_gold")
top_customers.write.format("delta").mode("overwrite").option("overwriteSchema", "true").save("/tmp/delta/
top_customers_gold")
monthly_sales.write.format("delta").mode("overwrite").option("overwriteSchema", "true").save("/tmp/delta/
monthly_sales_gold")
category_over_time.write.format("delta").mode("overwrite").option("overwriteSchema", "true").save("/tmp/delta/
overtime_gold")
```

07:42 AM (3s) 25

```
joined_final.write.mode("overwrite").parquet("/FileStore/Table/Retail_sales_data")
```

(3) Spark Jobs

07:43 AM (5s) 26

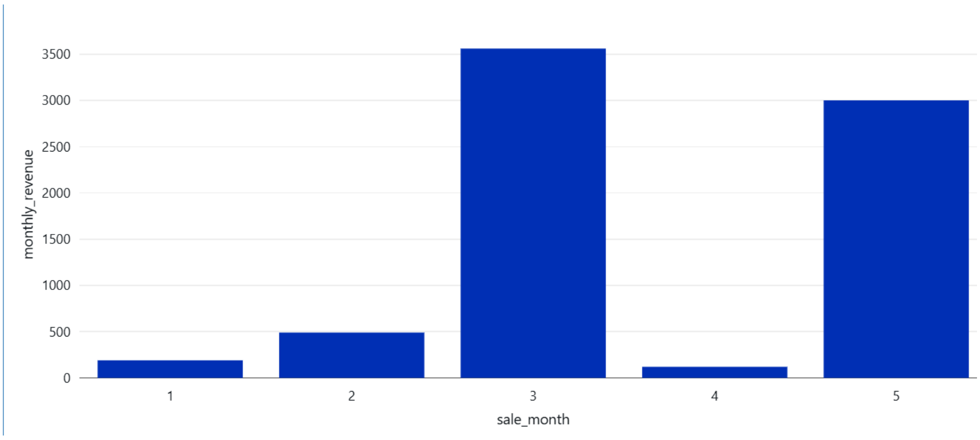
```
df_parquet = spark.read.parquet("/FileStore/Table/Retail_sales_data")
df_parquet.display()
```

Result:

1. Monthly sales:

Table | Visualization 1 | +

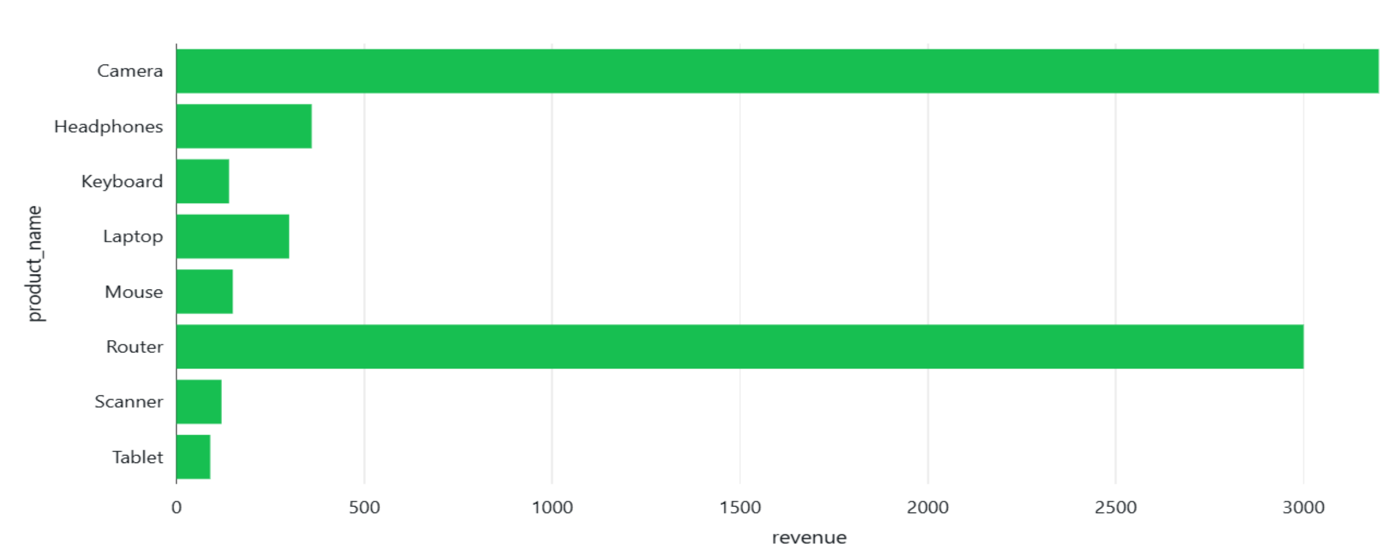
	sale_month	monthly_revenue
1	1	190
2	2	490
3	3	3560
4	4	120
5	5	3000



2. Top products:

Table | Visualization 1 | +

	product_name	revenue
1	Camera	3200
2	Router	3000
3	Headphones	360
4	Laptop	300
5	Mouse	150
6	Keyboard	140
7	Scanner	120
8	Tablet	90



3. Top customers:

	customer_id	revenue
1	109	3200
2	116	3000
3	107	360
4	101	300
5	106	150
6	105	140
7	115	120
8	102	90

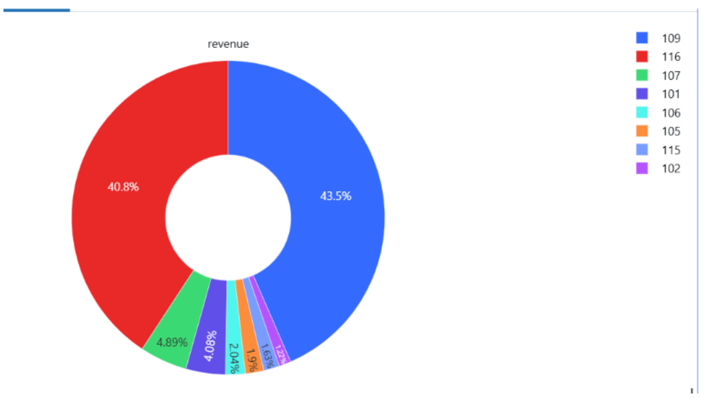


Table Visualization 1

	store_type	total_sales
1	Online	3920
2	Physical	3440

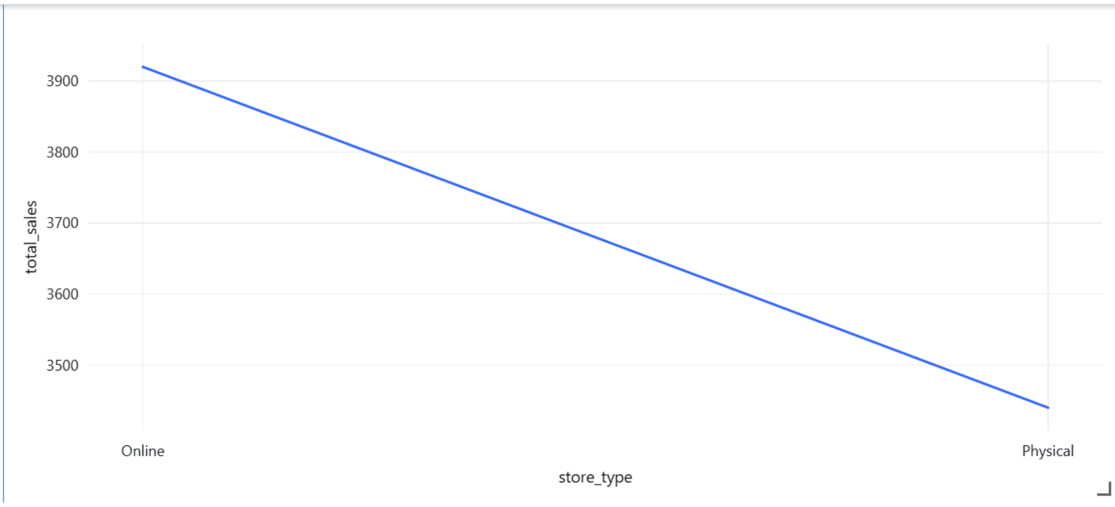


Table | Visualization 1 | +

	sale_month	category	total_revenue
1	1	Electronics	190
2	2	Accessories	290
3	2	Electronics	200
4	3	Accessories	360
5	3	Electronics	3200
6	4	Accessories	120
7	5	Electronics	3000

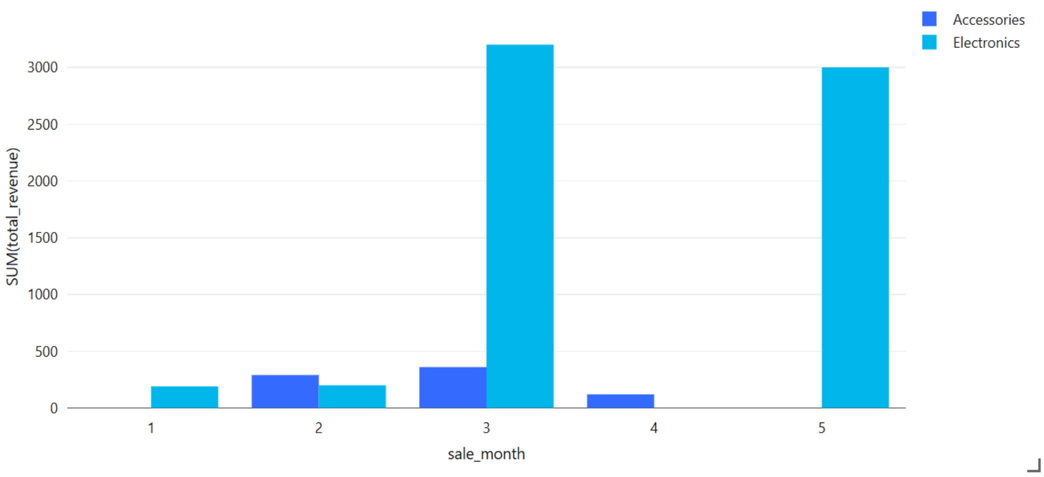


Table | +

	category	avg_age
1	Accessories	52.25
2	Electronics	48.8

6. **SQL Queries**:

- Use Spark SQL to identify products that contributed at least 10% of the total revenue.
- Identify cities with more than 100 unique customers (if any).

Code, Result:

07:41 AM (<1s) 19

```
joined_final.createOrReplaceTempView("sales_csv")
```

07:41 AM (2s) 20

```
total_revenue = spark.sql("SELECT SUM(total_revenue) AS total_revenue FROM sales_csv").collect()[0]['total_revenue']

total_revenue
```

(4) Spark Jobs

```
Out[18]: 7360
```

07:41 AM (2s) 21 Python

```
query = f"""
    select product_name,
            sum(total_revenue) as total_revenue_prod,
            (total_revenue_prod / {total_revenue}) * 100 as revenue_percentage
    from sales_csv
    group by product_name
    having revenue_percentage >= 10
"""

spark.sql(query).display()
```

(4) Spark Jobs

Table

	product_name	total_revenue_prod	revenue_percentage
1	Router	3000	40.76086956521739
2	Camera	3200	43.47826086956522

2 days ago (1s) 22

```
query = f"""
    select product_name,
            sum(total_revenue) as total_revenue_prod,
            (total_revenue_prod / {total_revenue}) * 100 as revenue_percentage
    from sales_csv
    group by product_name
"""

spark.sql(query).display()
```

(4) Spark Jobs

Table

	product_name	total_revenue_prod	revenue_percentage
1	Scanner	120	1.6304347826086956
2	Router	3000	40.76086956521739
3	Laptop	300	4.076086956521739
4	Mouse	150	2.0380434782608696
5	Camera	3200	43.47826086956522
6	Tablet	90	1.2228260869565217
7	Keyboard	140	1.9021739130434785
8	Headphones	360	4.891304347826087

07:42 AM (2s) 23

```
spark.sql("select city, count(distinct customer_id) as unique_customers from sales_csv group by city having
unique_customers > 100").display()
```

(5) Spark Jobs

Table

	city	unique_customers

Conclusion:

This project was a great opportunity to apply PySpark and Spark SQL in a real-world retail context. By following the Medallion Architecture, I gained practical experience in designing a structured data pipeline — from raw data ingestion to enriched insights ready for business analysis.

It deepened my understanding of:

- Data cleaning and transformation techniques
- Building modular ETL workflows
- Leveraging Delta Lake for scalable and reliable data storage
- Applying SQL for business-driven analytics