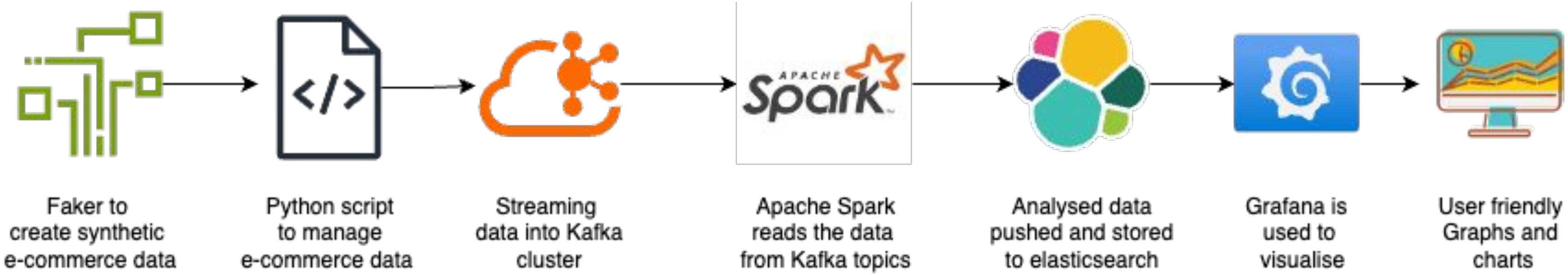
Introduction

In this project, we design and implement a robust data pipeline that integrates key technologies to achieve real-time analytics for an e-commerce platform :

- ➢ Kafka: For reliable and scalable data streaming.
- ➢ Spark: To handle large-scale data processing in real-time.
- ➢ Elasticsearch: For efficient data storage and retrieval.
- ➢ Grafana: To visualize and analyze the processed data.
- ➢ Python: As the scripting backbone for orchestrating these components.

System Design



Data Generation and Streaming

Data generation, a crucial component of the pipeline, where a Python script (data_generation.py) is called from the main.py file which simulates real-time e-commerce activity. This script leverages the Faker library to generate realistic yet synthetic schemas.

generate_customer()
generate_product()
generate_transaction()
generate_product_view()
generate_system_log()
generate_user_interaction()

```
# Function to send data to Kafka
def send_data():
    # Occasionally add new customers or products
    """
    Send multiple types of data (customer, product, transaction, etc.) to Kafka.

    Handles the generation and sending of various data types to their respective Kafka topics.
    """
    global customer_no
    global product_no
    if customer_no < 500:
        stream_customer_data('customer_data')
        customer_no += 1
    else:
        if random.random() < 0.1:
            stream_customer_data('customer_data')
            customer_no += 1
    if product_no < 100:
        stream_product_data('product_data')
        product_no += 1
    else:
        if random.random() < 0.1:
            stream_product_data('product_data')
            product_no += 1

    # stream_transaction_data('transaction_data')
    stream_product_view_data('product_view_data')
    stream_user_interaction_data('user_interaction_data')
    stream_system_log_data('system_log_data')
```

Kafka: The Data Streaming

```python
def send_message(topic, message):

    """
    Send a message to a specified Kafka topic.

    :param topic: The Kafka topic to send the message to
    :param message: The message to send (dictionary or JSON-serializable object)
    """

    try:
        print(f"Connecting to Kafka broker at: {kafka_broker}")
        producer = KafkaProducer(
            bootstrap_servers=[kafka_broker],
            value_serializer=lambda x: json.dumps(x).encode('utf-8')
        )
        print("Kafka producer initialized.")
    except KafkaError as e:
        print(f"Failed to initialize Kafka producer: {e}")
        raise
    try:
        producer.send(topic, value=message)
        producer.flush()
        print(f"Message sent to topic '{topic}': {message}")
    except KafkaError as e:
        print(f"Error sending message: {e}")
        raise
```

```python
create_topic('transaction_data', num_p    (variable) num_partitions: int
create_topic('product_view_data', num_
create_topic('user_interaction_data', num_partitions, replication_factor)
create_topic('system_log_data', num_partitions, replication_factor)
with ThreadPoolExecutor(max_workers=5) as executor:
    while True:
        executor.submit(send_data)
        time.sleep(random.uniform(1, 2))
```

Parallel Data Generation and Continuous Execution from main.py

The KafkaProducer is configured with a broker address and a serializer for the message values, converting them into a JSON-encoded UTF-8 string.

Real-Time Data Processing and Analysis

```
customerDF = (spark.readStream
              .format("kafka")
              .option("kafka.bootstrap.servers", kafka_bootstrap_servers)
              .option("subscribe", 'customer_data')
              .option("startingOffsets", "earliest")  # Start from the earl
              .load()
              .selectExpr("CAST(value AS STRING)")
              .select(from_json("value", customerSchema).alias("data"))
              .select("data.*")
              .withColumn("processingTime", current_timestamp())
              .withWatermark("last_login", "2 hours")
             )
customerDF = customerDF.withColumn("@timestamp", col("processingTime"))
```

The script reads streaming data from Kafka, ensuring real-time ingestion of customer transactions, product details, and consumer interactions.

We leverage windowing and aggregation functions in Spark to perform complex, time-based analytics on the incoming data streams. To enhance the value of consumer reviews, we incorporated Natural Language Processing (NLP) by utilizing a custom User-Defined Function (UDF) in PySpark. (spark_processing.py)

```
salesVelocityDF.writeStream \
    .outputMode("complete") \
    .format("memory") \
    .queryName("sales_velocity") \
    .option("checkpointLocation", "/tmp/sales_velocity_checkpoint") \
    .start()

spark.sql("SELECT * FROM sales_velocity").show()

# Join with Product Data to include category
productWithThresholdDF = productDF.join(
    spark.sql("SELECT * FROM sales_velocity"),
    "product_id",
    "left_outer"
).withColumn(
    "threshold", col("average_daily_sales") * 2
).fillna({"threshold": 10})  # Default threshold

# Filter for Low Stock and Include Category
lowStockDF = productWithThresholdDF.filter(
    col("stock_quantity") < col("threshold")
).select(
    "product_id", "category", "name", "stock_quantity", "threshold", "processingTime"
).withColumn(
    "alert", lit("Low stock level detected")
)
```

Writing Data to Elasticsearch

```
# Wait for any of the streams to finish
customerAnalysisDF.writeStream \
    .outputMode("update") \
    .format("org.elasticsearch.spark.sql") \
    .option("checkpointLocation", "/tmp/spark/checkpoints/customeranalysis_analysis") \
    .option("es.nodes", "localhost") \
    .option("es.port", "9200") \
    .option("es.resource", "customeranalysis_index") \
    .option("es.mapping.id", "unique_id") \
    .start()
productAnalysisDF.writeStream \
    .outputMode("update") \
    .format("org.elasticsearch.spark.sql") \
    .option("checkpointLocation", "/tmp/spark/checkpoints/productanalysis_analysis") \
    .option("es.nodes", "localhost") \
    .option("es.port", "9200") \
    .option("es.resource", "productanalysis_index") \
    .option("es.mapping.id", "unique_id") \
    .start()
```

The script uses writeStream with foreachBatch to process each batch of streaming data. This ensures that as new data arrives, it is processed in batches and efficiently transferred to Elasticsearch.

Tests

```
[2024-12-06T15:24:18,145][INFO ][o.e.h.AbstractHttpServerTransport] [Anjalis-MacBook-Air.local] publish_add
ress {10.0.0.37:9200}, bound_addresses {[::]:9200}
[2024-12-06T15:24:18,171][INFO ][o.e.n.Node               ] [Anjalis-MacBook-Air.local] started {Anjalis-Ma
cBook-Air.local}{r_xg_H62RkWr5HQpmvecyQ}{ARAk1ACrRQm5J7kwF2ioSg}{Anjalis-MacBook-Air.local}{127.0.0.1}{127.
0.0.1:9300}{cdfhilmrstw}{8.16.1}{7000099-8518000}{ml.machine_memory=8589934592, transform.config_version=10
.0.0, xpack.installed=true, ml.config_version=12.0.0, ml.max_jvm_size=4294967296, ml.allocated_processors_d
ouble=8.0, ml.allocated_processors=8}
[2024-12-06T15:24:18,834][INFO ][o.e.l.ClusterStateLicenseService] [Anjalis-MacBook-Air.local] license [4f5
81cbd-c8c1-4758-99d3-86694d23dd89] mode [basic] - valid
[2024-12-06T15:24:18,838][INFO ][o.e.c.f.AbstractFileWatchingService] [Anjalis-MacBook-Air.local] starting
file watcher ...
[2024-12-06T15:24:18,843][INFO ][o.e.c.f.AbstractFileWatchingService] [Anjalis-MacBook-Air.local] file sett
ings service up and running [tid=88]
[2024-12-06T15:24:18,845][INFO ][o.e.g.GatewayService     ] [Anjalis-MacBook-Air.local] recovered [3] indic
es into cluster_state
[2024-12-06T15:24:18,846][INFO ][o.e.r.s.FileSettingsService] [Anjalis-MacBook-Air.local] setting file [/Us
ers/anjali./downloads/elasticsearch-8.16.1/config/operator/settings.json] not found, initializing [file_set
tings] as empty
[2024-12-06T15:24:19,325][INFO ][o.e.h.n.s.HealthNodeTaskExecutor] [Anjalis-MacBook-Air.local] Node [{Anjal
is-MacBook-Air.local}{r_xg_H62RkWr5HQpmvecyQ}] is selected as the current health node.
[2024-12-06T15:24:19,329][INFO ][o.e.c.r.a.AllocationService] [Anjalis-MacBook-Air.local] current.health="G
REEN" message="Cluster health status changed from [RED] to [GREEN] (reason: [shards started [[.ds-ilm-histo
ry-7-2024.12.05-000001][0], [.ds-.logs-deprecation.elasticsearch-default-2024.12.05-000001][0], [.security-
7][0]]])." previous.health="RED" reason="shards started [[.ds-ilm-history-7-2024.12.05-000001][0], [.ds-.lo
gs-deprecation.elasticsearch-default-2024.12.05-000001][0], [.security-7][0]]"
```

```
(base) anjali.@Anjalis-MacBook-Air elasticsearch-8.16.1 % curl http://localhost:9200
{
  "name" : "Anjalis-MacBook-Air.local",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "Ke5tlZmQQz-qVJ-VWFFKdA",
  "version" : {
    "number" : "8.16.1",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "ffe992aa682c1968b5df375b5095b3a21f122bf3",
    "build_date" : "2024-11-19T16:00:31.793213192Z",
    "build_snapshot" : false,
    "lucene_version" : "9.12.0",
    "minimum_wire_compatibility_version" : "7.17.0",
    "minimum_index_compatibility_version" : "7.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

Ping Test:

- Validated ElasticSearch functionality by successfully pinging the server.
- Confirmed the server's readiness and ability to respond to queries.

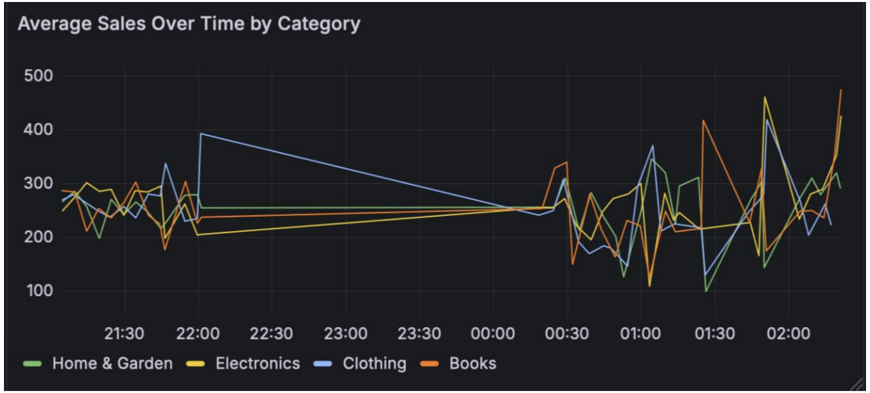
```
(proj532) anjali.@Anjalis-MacBook-Air bin % /opt/homebrew/opt/kafka/bin/kafka-consol
e-consumer --topic test-topic --from-beginning --bootstrap-server localhost:9092
Hey
how are you?
holla
Hi This is team Proj532
Hi This is team Proj532
```

Consumer

Producer

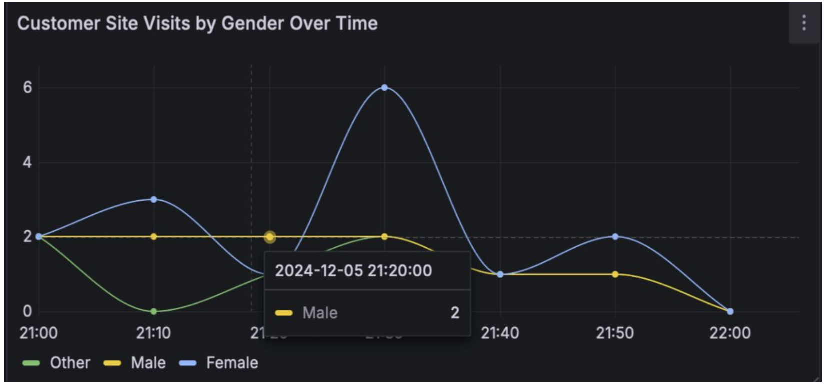
```
-producer --topic test-topic --bootstrap-server localhost:9092
>Hi This is team Proj532
>^C%
(base) anjali.@Anjalis-MacBook-Air ~ % /opt/homebrew/opt/kafka/bin/kafka-console
-producer --topic test-topic --bootstrap-server localhost:9092
>Hi This is team Proj532
>
```

Experimental Results

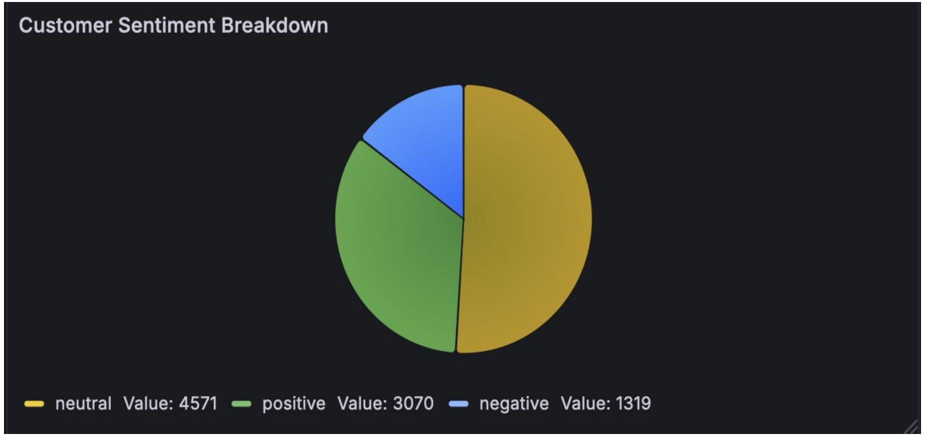


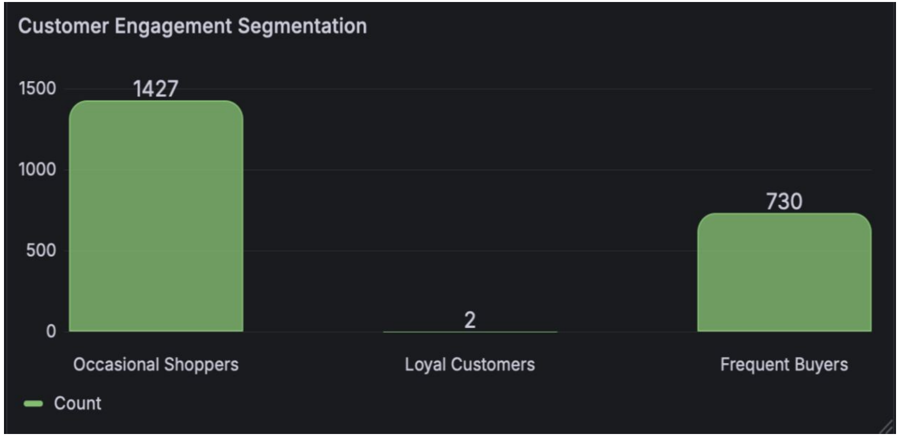
Average sales performance of different product categories across a defined time period. Each categories showing how sales fluctuate throughout the observed time frame.

Gender categories include Male, Female, and Other, with engagement measured through site visits.
Analyze gender-wise preferences for product categories based on browsing patterns.

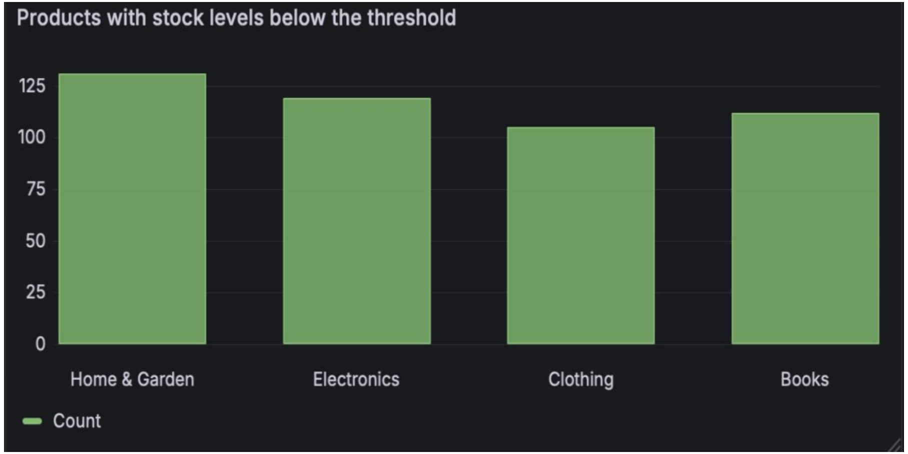


Results: Grafana Output

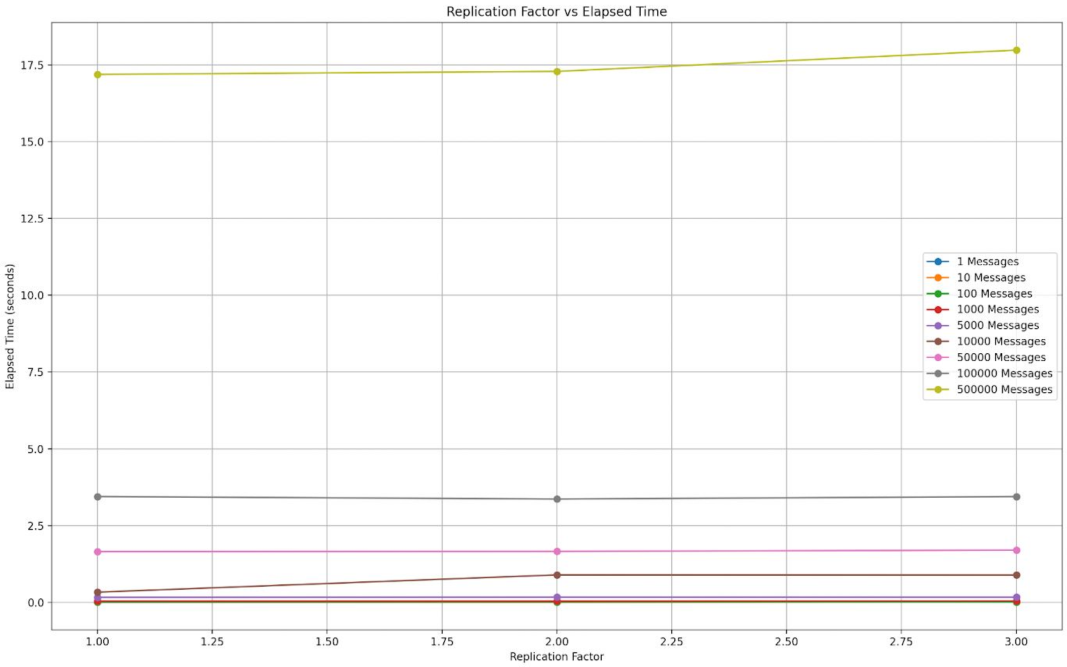




customer_id	total_interactions	wishlist_additions	reviews	ratings	@timestamp
fbd89a20-fbe1-4e4...	36	13	10	13	2024-12-06 02:26:...
db19c50a-b8eb-438...	36	12	11	13	2024-12-06 02:26:...
f07e8009-0816-4fb...	35	14	13	8	2024-12-06 02:26:...
22d061db-44b4-452...	35	12	6	17	2024-12-06 02:26:...
63d55cd1-3d79-4f1...	33	13	7	13	2024-12-06 02:26:...
f966b49e-da20-4a4...	33	12	10	11	2024-12-06 02:26:...
{"customer_id":"8...	33	14	10	9	2024-12-06 02:26:...
c6f4dfcb-1970-44f...	33	7	10	16	2024-12-06 02:26:...
ba996c04-f07b-405...	33	8	17	8	2024-12-06 02:26:...
e889f6bc-f89a-440...	33	8	12	13	2024-12-06 02:26:...
91f4ac58-883e-43f...	32	7	11	14	2024-12-06 02:26:...
1cc4be58-4e6a-415...	32	13	14	5	2024-12-06 02:26:...
5558472d-c15f-411...	31	15	6	10	2024-12-06 02:26:...
4d6c382a-b383-4e0...	31	10	9	12	2024-12-06 02:26:...
ddfb9843-9ba4-4ad...	31	12	10	9	2024-12-06 02:26:...
be9d1e6d-0044-4e2...	31	16	11	4	2024-12-06 02:26:...
0a83d3d3-ee95-4c3...	31	12	8	11	2024-12-06 02:26:...
f1779f68-5fe0-436...	31	8	10	13	2024-12-06 02:26:...
1283f7cf-1a59-4a0...	30	11	12	7	2024-12-06 02:26:...
6c93c1c7-60de-437...	30	4	17	9	2024-12-06 02:26:...



Kafka Analytics - Replication Factor



- Impact of **replication factor in kafka brokers** with increasing number of messages
- As the number of messages (transaction_data in our system) increase in the system, the time taken increases with the replication factor.
- This is expected cause as there are more message more time is required to replicate them across the brokers.

Kafka Analytics - 12 Partitions and varied brokers

- Kafka partitions are the unit of parallelism. As the number of brokers increases, partitions can be spread across more brokers

- For our test setup, 12 consumer threads are fetching data from 12 partitions of the transaction_data topic.

- The expectation is that adding more brokers reduces contention and enhances parallelism.

- We have tested the throughput for 1 broker, 2 broker and 3 brokers

1 broker:-

```
=== Average Throughput ===
Average Throughput over 10 runs: 385.22 records/second
24/12/05 18:26:50 INFO ShutdownHookManager: Shutdown hook called
24/12/05 18:26:50 INFO ShutdownHookManager: Deleting directory /pr
```

2 brokers:-

```
=== Average Throughput ===
Average Throughput over 10 runs: 406.82 records/second
24/12/05 18:30:54 INFO ShutdownHookManager: Shutdown hook called
24/12/05 18:30:54 INFO ShutdownHookManager: Deleting directory /pri
```

3 brokers:-

```
=== Average Throughput ===
Average Throughput over 10 runs: 408.58 records/second
24/12/05 18:28:32 INFO ShutdownHookManager: Shutdown hook called
24/12/05 18:28:32 INFO ShutdownHookManager: Deleting directory /p
```

PySpark Analytics - Performance of the system across different pyspark cores

Setup: Evaluated transaction data processing with varying cores (1, 2, 4, 8) and memory (512MB to 4GB).

Observations:

1 Core, 512MB: Longest elapsed time (41.82s) due to sequential processing.
2 Cores, 1GB: Significant improvement (24.03s) from parallelism and increased memory.
4 Cores, 2GB: Similar performance (24.54s)
8 Cores, 4GB: Best performance (21.82s)

Key Insight: Increasing cores and memory improves performance initially but hits bottlenecks as parallelism saturates.

Optimal configuration for this workload is 8 cores and 4GB memory, balancing performance and resource usage.

PySpark Analytics - Performance of the system across different pyspark cores

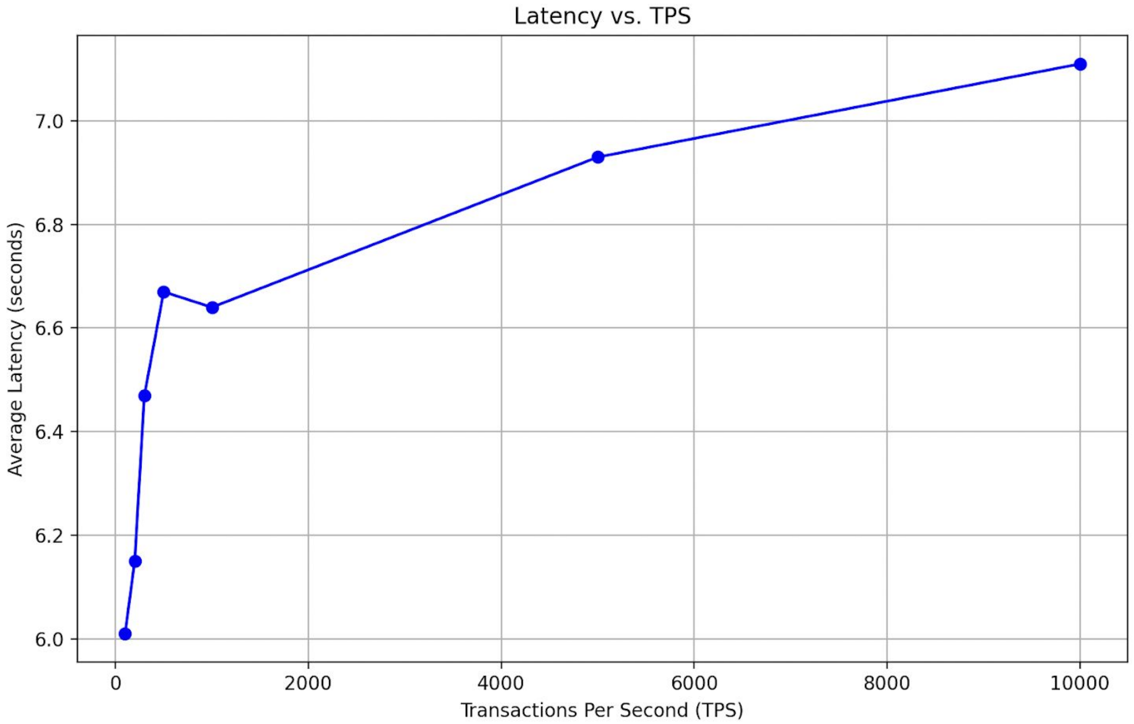
```
Performance Evaluation Results:
Cores: 1, Memory: 512m, Elapsed Time: 41.82 seconds
Cores: 2, Memory: 1g, Elapsed Time: 24.03 seconds
Cores: 4, Memory: 2g, Elapsed Time: 24.54 seconds
Cores: 8, Memory: 4g, Elapsed Time: 21.82 seconds
24/12/05 20:00:49 INFO ShutdownHookManager: Shutdown hook called
24/12/05 20:00:49 INFO ShutdownHookManager: Deleting directory /p
24/12/05 20:00:49 INFO ShutdownHookManager: Deleting directory /p
24/12/05 20:00:49 INFO ShutdownHookManager: Deleting directory /p
24/12/05 20:00:49 INFO ShutdownHookManager: Deleting directory /p
24/12/05 20:00:49 INFO ShutdownHookManager: Deleting directory /p
24/12/05 20:00:49 INFO ShutdownHookManager: Deleting directory /p
(venv) (base) Admin@MacBook-Air project %
```

End to End Latency across different loads (peak sale periods vs low sale periods)

Producer streams transaction_data at varying TPS (transactions per second) levels (100 (low sale periods), 200, 300, 500, 1000, 5000, 10000 (peak sale periods)) to Kafka. Every configuration runs for 60 seconds

Spark Streaming job processes all the transactions for every TPS level

Latency is calculated from the ingestion time (time when the transaction record is added to the stream) to the processing time (time when the spark stream reads the record and analyzes on it).



As the load increases and transaction_data, latency rises in a controlled manner, demonstrating the system’s ability to handle higher loads while maintaining predictable performance trends.

Goals

Evaluate the performance of the application with different numbers of PySpark cores and memory settings. Analyze how these changes affect processing time, throughput, and latency of the application.	Done
How does the system scale as we increase the number of Kafka brokers ? At what point does adding resources no longer improve performance, and why?	Done
What is the system’s response time from data ingestion to insight generation, and how does it vary across different load levels (normal operation vs peak sales periods)?	Done
What is the distribution of active customers by their last login time, and how does it vary across different genders?	Done
What is the average price of products in each category, and how does it compare to overall market trends?	Done
How does sales performance vary by product over different time windows, and are there peak periods?	Done
What are the top-selling products by quantity, and which products are struggling to sell?	Done

Possible Improvements and Future Scope

- Could have achieved more Analysis goals, e.g. If time permits, how can we analyze the product purchase patterns of similar customers and generate personalized product recommendations using collaborative filtering or content-based filtering techniques?
-
-