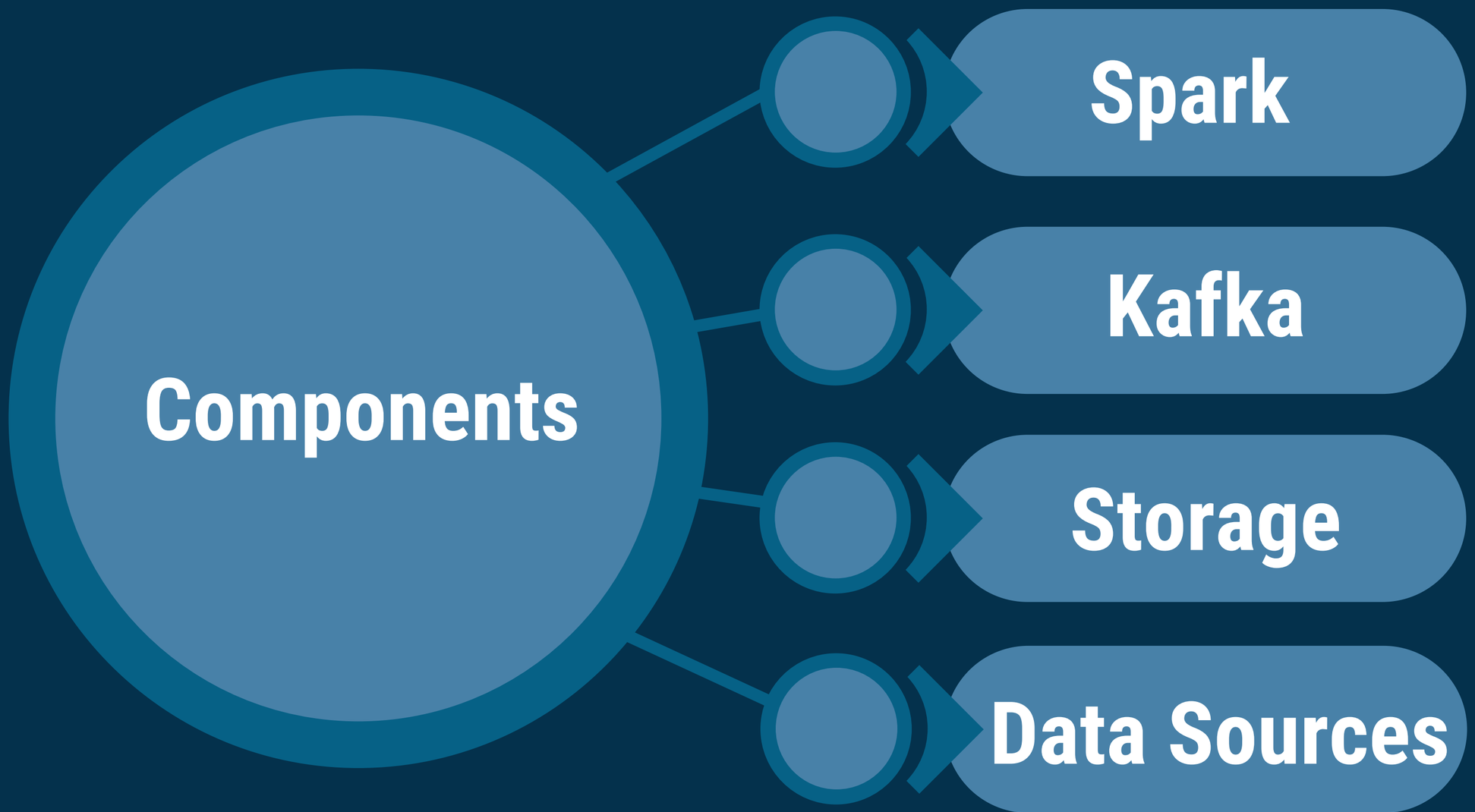
Real-Time E-commerce Data Pipeline with Spark ETL

Table Of Content

Introduction

The ShopEase platform generates substantial data from transactions, user interactions, and inventory updates. This project implements an ETL pipeline using Apache Spark to extract, transform, and load data for real-time analytics and reporting. The goal is to process data efficiently and deliver actionable insights that support the decision-making process for both historical and real-time operations.

Project Architecture



Project Pipeline

DATA SOURCE

1-Historical data (CSV, JSON)
2-Streaming data (Kafka user activity logs)

DATA INGESTION

1- Batch Data (Load CSV, JSON)
2- Real-Time Data Simulate and ingest streaming data from a Kafka

DATA PROCESSING

1- Batch processing
2- Real-Time Streaming Processing

DATA STORAGE

1- Parquet format for batch-processed
2-in-memory data store for real-time metrics.

Data Ingestion

Batch Data Ingestion

Data Ingestion - Load Large Datasets

```
from pyspark.sql import SparkSession

# Initialize SparkSession
spark = SparkSession.builder \
    .appName("ShopEase Data Ingestion") \
    .getOrCreate()

# Load large transactions data from CSV
transactions_df = spark.read.option("header", True) \
                            .csv("/user/student/spark_project/large_transactions.csv")

# Load large inventory data from JSON
inventory_df = spark.read.option("multiline", True) \
                         .json("/user/student/spark_project/large_inventory.json")

# Load large customer feedback data from CSV
feedback_df = spark.read.option("header", True) \
                        .csv("/user/student/spark_project/large_customer_feedback.csv")

# Display a sample of each dataset
transactions_df.show(5)
inventory_df.show(5)
feedback_df.show(5)
```

```
|transaction_id|user_id|product_id|quantity|amount|transaction_date|
+--------------+-------+----------+--------+------+----------------+
|             1|   5388|      9152|       9|305.67|      2023-01-07|
|             2|  31656|      5763|       1|184.31|      2023-05-16|
|             3|  36725|      4733|       4|  7.66|      2023-01-20|
|             4|  22713|      4207|       1|411.73|      2023-04-19|
|             5|  74067|      1914|       5|499.21|      2023-08-17|
+--------------+-------+----------+--------+------+----------------+
only showing top 5 rows

Inventory Table:
+------+----------+------------+-----------+
| price|product_id|product_name|stock_level|
+------+----------+------------+-----------+
| 91.89|         1|   Product_1|         61|
|  9.75|         2|   Product_2|        553|
|182.89|         3|   Product_3|        328|
|148.38|         4|   Product_4|        347|
| 78.17|         5|   Product_5|        180|
+------+----------+------------+-----------+
only showing top 5 rows

Customer Feedback Table:
+-------+----------+------+--------------------+-----------+
|user_id|product_id|rating|              review|review_date|
+-------+----------+------+--------------------+-----------+
|   9320|      1971|     4|Review text for p...| 2023-01-11|
|  64245|      3664|     4|Review text for p...| 2023-01-23|
|  37341|      7419|     4|Review text for p...| 2023-08-14|
```

Data Ingestion

Real-Time Data Ingestion

```
# !pip install kafka-python

from kafka import KafkaProducer
import json
import time
import random

# Define Kafka producer
producer = KafkaProducer(bootstrap_servers='localhost:9092', value_serializer=lambda v: json.dump

# Simulating user activity data
user_actions = ["login", "view", "purchase", "logout"]

def generate_user_activity():
    return {
        "user_id": random.randint(1, 1000),
        "action": random.choice(user_actions),
        "timestamp": int(time.time())
    }

# Stream data to Kafka
while True:
    activity = generate_user_activity()
    producer.send('user-activity', activity)
    print(f"Produced: {activity}")
    time.sleep(1)  # Delay to simulate real-time streaming
```

```
Produced: {'user_id': 103, 'action': 'login', 'timestamp': 1725623778}
Produced: {'user_id': 127, 'action': 'view', 'timestamp': 1725623779}
Produced: {'user_id': 253, 'action': 'logout', 'timestamp': 1725623780}
Produced: {'user_id': 265, 'action': 'purchase', 'timestamp': 1725623781}
Produced: {'user_id': 825, 'action': 'login', 'timestamp': 1725623782}
Produced: {'user_id': 872, 'action': 'view', 'timestamp': 1725623783}
Produced: {'user_id': 723, 'action': 'logout', 'timestamp': 1725623784}
Produced: {'user_id': 708, 'action': 'login', 'timestamp': 1725623785}
Produced: {'user_id': 137, 'action': 'purchase', 'timestamp': 1725623786}
Produced: {'user_id': 993, 'action': 'logout', 'timestamp': 1725623787}
Produced: {'user_id': 577, 'action': 'logout', 'timestamp': 1725623788}
Produced: {'user_id': 69, 'action': 'view', 'timestamp': 1725623789}
Produced: {'user_id': 387, 'action': 'purchase', 'timestamp': 1725623790}
Produced: {'user_id': 39, 'action': 'view', 'timestamp': 1725623791}
Produced: {'user_id': 490, 'action': 'login', 'timestamp': 1725623792}
Produced: {'user_id': 789, 'action': 'logout', 'timestamp': 1725623793}
Produced: {'user_id': 499, 'action': 'view', 'timestamp': 1725623794}
Produced: {'user_id': 935, 'action': 'logout', 'timestamp': 1725623795}
Produced: {'user_id': 23, 'action': 'login', 'timestamp': 1725623796}
Produced: {'user_id': 99, 'action': 'login', 'timestamp': 1725623797}
```

Data Processing

Using RDDs

Data Cleaning and Transformation with RDDs

```python
from pyspark.sql import Row
import hashlib

# Convert transactions DataFrame to RDD
transactions_rdd = transactions_df.rdd

# Filter out corrupted records (e.g., missing transaction_id or amount)
# Assuming 'transaction_id' and 'amount' are the required fields
cleaned_rdd = transactions_rdd.filter(lambda row: row['transaction_id'] is not None or row['amount'] is not

# Function to Anonymize user IDs using Hashing
def anonymize(record):
    # Assuming 'user_id' is the field we want to anonymize
    hashed_user_id = hashlib.sha256(record['user_id'].encode()).hexdigest()
    # Create a new Row object with the hashed user_id
    new_record = record.asDict()  # Convert Row to dict
    new_record['user_id'] = hashed_user_id  # Replace the user_id with the hashed value
    return Row(**new_record)  # Return a new Row object with the updated user_id

# Apply the anonymization function to the RDD
anonymized_rdd = cleaned_rdd.map(anonymize)

# Convert back to DataFrame
cleaned_transactions_df = anonymized_rdd.toDF()

# Display cleaned and anonymized data
print("Cleaned and Anonymized Transactions Data:")
cleaned_transactions_df.show(5)

cleaned_transactions_df.count()
```

```
Cleaned and Anonymized Transactions Data:
+--------------+--------------------+----------+--------+------+---------
|transaction_id|             user_id|product_id|quantity|amount|transacti
+--------------+--------------------+----------+--------+------+---------
|             1|b797d397acb6b79e3...|      9152|       9|305.67|      202
|             2|3911f43de537d6765...|      5763|       1|184.31|      202
|             3|a0127a90205f9ca56...|      4733|       4|  7.66|      202
|             4|3ec95a5c41362c498...|      4207|       1|411.73|      202
|             5|90ee0ca050e4b8892...|      1914|       5|499.21|      202
+--------------+--------------------+----------+--------+------+---------
only showing top 5 rows

1000000
```

Data Processing

Using DataFrames

DataFrame Operations for Cleaning and Transformatic

```
from pyspark.sql.functions import col, lower, trim

# Step 3: DataFrame Operations for Cleaning and Transformation

# Clean inventory data by handling missing values and normalizing text
cleaned_inventory_df = inventory_df.dropna(subset=["stock_level"]) \
                                   .withColumn("product_name", lower(trim(col("

# Display the cleaned inventory data (first 5 rows)
print("Cleaned Inventory Data:")
cleaned_inventory_df.show(5)

# Assuming you have a column in both DataFrames for joining (e.g., 'product_id'
# Perform a join operation to combine user activity logs with cleaned transacti
# For this example, we'll assume joining on 'product_id' from inventory and 'pr
joined_df = cleaned_inventory_df.join(transactions_df, "product_id", "inner")

# Display the joined DataFrame (first 5 rows)
print("Joined Data:")
joined_df.show(5)
```

```
leaned Inventory Data:
+------+----------+------------+-----------+
 price|product_id|product_name|stock_level|
+------+----------+------------+-----------+
 91.89|         1|   product_1|         61|
  9.75|         2|   product_2|        553|
182.89|         3|   product_3|        328|
148.38|         4|   product_4|        347|
 78.17|         5|   product_5|        180|
+------+----------+------------+-----------+
nly showing top 5 rows

oined Data:
+----------+-----+------------+-----------+--------------+-------+--------+------+------
|product_id|price|product_name|stock_level|transaction_id|user_id|quantity|amount|transa
+----------+-----+------------+-----------+--------------+-------+--------+------+------
      9152|238.7|product_9152|        871|             1|   5388|       9|305.67|
      5763|91.52|product_5763|        252|             2|  31656|       1|184.31|
      4733|33.22|product_4733|        235|             3|  36725|       4|  7.66|
      4207| 6.67|product_4207|        928|             4|  22713|       1|411.73|
      1914|29.71|product_1914|          4|             5|  74067|       5|499.21|
+----------+-----+------------+-----------+--------------+-------+--------+------+------
nly showing top 5 rows
```

Data Processing

Spark SQL

Create Temporary Views for SQL Queries

```
]: cleaned_transactions_df.createOrReplaceTempView("transactions")
   cleaned_inventory_df.createOrReplaceTempView("inventory")
   joined_df.createOrReplaceTempView("joined_data")
```

Top 10 Most Purchased Products in the Last Month

In [13]:
```
top_products_df = spark.sql("""
    SELECT product_id, COUNT(*) AS purchase_count
    FROM transactions
    WHERE transaction_date >= '2023-12-01' and transaction_date <= '2023-12-30'
    GROUP BY product_id
    ORDER BY purchase_count DESC
    LIMIT 10
""")
print("Top 10 Most Purchased Products in the Last Month:")
top_products_df.show()
```

```
Top 10 Most Purchased Products in the Last Month:
[Stage 33:=======================================>                 (4 + 2) / 6]
+----------+--------------+
|product_id|purchase_count|
+----------+--------------+
|       629|            22|
|      4590|            21|
|      7350|            20|
|      8334|            20|
|       127|            20|
|      4870|            19|
|      1733|            19|
|      3673|            19|
|      8220|            19|
|      6795|            19|
```

Data Processing

Spark SQL

Query: Monthly revenue trends

[15]:
```
monthly_revenue_df = spark.sql("""
    SELECT
        YEAR(transaction_date) AS year,
        MONTH(transaction_date) AS month,
        SUM(amount) AS total_revenue
    FROM transactions
    GROUP BY year, month
    ORDER BY year, month
""")
print("Monthly Revenue Trends:")
monthly_revenue_df.show()
```

```
Monthly Revenue Trends:
[Stage 38:====================>                                    (2 + 4) / 6]
+----+-----+--------------------+
|year|month|       total_revenue|
+----+-----+--------------------+
|2023|    1| 2.144845023000001E7|
|2023|    2| 1.949427259999998E7|
|2023|    3|        2.15441875E7|
|2023|    4| 2.081824269000004E7|
|2023|    5|2.1434179750000037E7|
|2023|    6|2.0885249160000004E7|
|2023|    7|2.1363113009999946E7|
|2023|    8|2.1311435569999993E7|
|2023|    9| 2.066753037999999E7|
|2023|   10|2.1576082419999994E7|
|2023|   11| 2.090389989999999E7|
|2023|   12| 2.084169785999999E7|
```

Data Processing

Spark SQL

Query: Inventory turnover rates

In [16]:
```
turnover_rate_df = spark.sql("""
    SELECT
        i.product_id,
        i.product_name,
        SUM(t.amount) AS total_sold,
        AVG(i.stock_level) AS avg_stock_level,
        (SUM(t.amount) / AVG(i.stock_level)) AS turnover_rate
    FROM inventory i
    JOIN transactions t ON i.product_id = t.product_id
    GROUP BY i.product_id, i.product_name
    ORDER BY turnover_rate DESC
""")
print("Inventory Turnover Rates:")
turnover_rate_df.show()
```

```
Inventory Turnover Rates:
[Stage 41:=====================================>                   (4 + 2) / 6]
+----------+------------+------------------+---------------+------------------+
|product_id|product_name|        total_sold|avg_stock_level|     turnover_rate|
+----------+------------+------------------+---------------+------------------+
|      7567|product_7567|30000.179999999997|            1.0|30000.179999999997|
|      2871|product_2871|29058.729999999996|            1.0|29058.729999999996|
|      9501|product_9501|          28807.18|            1.0|          28807.18|
|      4373|product_4373|27230.920000000002|            1.0|27230.920000000002|
|      7759|product_7759|26989.679999999997|            1.0|26989.679999999997|
|      1284|product_1284|          26472.62|            1.0|          26472.62|
|      3135|product_3135|24224.690000000002|            1.0|24224.690000000002|
|      1467|product_1467|24153.539999999997|            1.0|24153.539999999997|
|        40|  product_40|          22390.04|            1.0|          22390.04|
|      2821|product_2821|          21368.41|            1.0|          21368.41|
|      6771|product_6771|20911.020000000004|            1.0|20911.020000000004|
|      9021|product_9021|20415.170000000002|            1.0|20415.170000000002|
|      9503|product_9503|          26934.28|            2.0|          13467.14|
```

Data Processing

Real-Time Streaming Processing

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col, approx_count_distinct, expr, window, to_timestamp
from pyspark.sql.types import StructType, StringType, IntegerType

# Initialize Spark session with Kafka package
spark = SparkSession.builder \
    .appName("KafkaPySparkStream") \
    .master("local[*]") \
    .config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10_2.12:3.1.1") \
    .getOrCreate()

# Define schema for user activity data
schema = StructType() \
    .add("user_id", IntegerType()) \
    .add("action", StringType()) \
    .add("timestamp", IntegerType())  # The timestamp is initially an integer (epoch time)

# Read stream from Kafka
kafka_df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "user-activity") \
    .load()

# Parse Kafka stream and convert to structured format
parsed_df = kafka_df.selectExpr("CAST(value AS STRING)") \
    .select(from_json(col("value"), schema).alias("data")) \
    .select("data.*")

# Convert the integer timestamp (epoch) to TimestampType
parsed_df = parsed_df.withColumn("timestamp", to_timestamp(col("timestamp")))

# Perform transformations (e.g., calculate active users per minute using approx_count_distinct)
active_users_df = parsed_df \
    .withWatermark("timestamp", "1 minute") \
    .groupBy(window(col("timestamp"), "1 minute"), "action") \
    .agg(approx_count_distinct("user_id").alias("active_users"))

# Show active users on the console in real-time
query = active_users_df.writeStream \
    .outputMode("update") \
    .format("console") \
    .start()

query.awaitTermination()
```

Data Processing

Real-Time Streaming Processing

```
Batch: 1
-------------------------------------------
+--------------------+--------+------------+
|              window|  action|active_users|
+--------------------+--------+------------+
|{2024-09-06 20:22...|purchase|           4|
|{2024-09-06 20:22...|   login|           4|
|{2024-09-06 20:22...|    view|           3|
|{2024-09-06 20:21...|    view|           3|
|{2024-09-06 20:22...|  logout|           4|
|{2024-09-06 20:21...|  logout|           1|
+--------------------+--------+------------+

-------------------------------------------
Batch: 2
-------------------------------------------
+--------------------+--------+------------+
|              window|  action|active_users|
+--------------------+--------+------------+
|{2024-09-06 20:22...|purchase|           6|
|{2024-09-06 20:22...|   login|           6|
|{2024-09-06 20:22...|    view|           8|
|{2024-09-06 20:22...|  logout|           9|
+--------------------+--------+------------+

-------------------------------------------
Batch: 3
-------------------------------------------
+--------------------+--------+------------+
|              window|  action|active_users|
+--------------------+--------+------------+
|{2024-09-06 20:22...|purchase|           9|
|{2024-09-06 20:22...|    view|          11|
|{2024-09-06 20:22...|  logout|          12|
```

Store Data

Store Data into HDFS

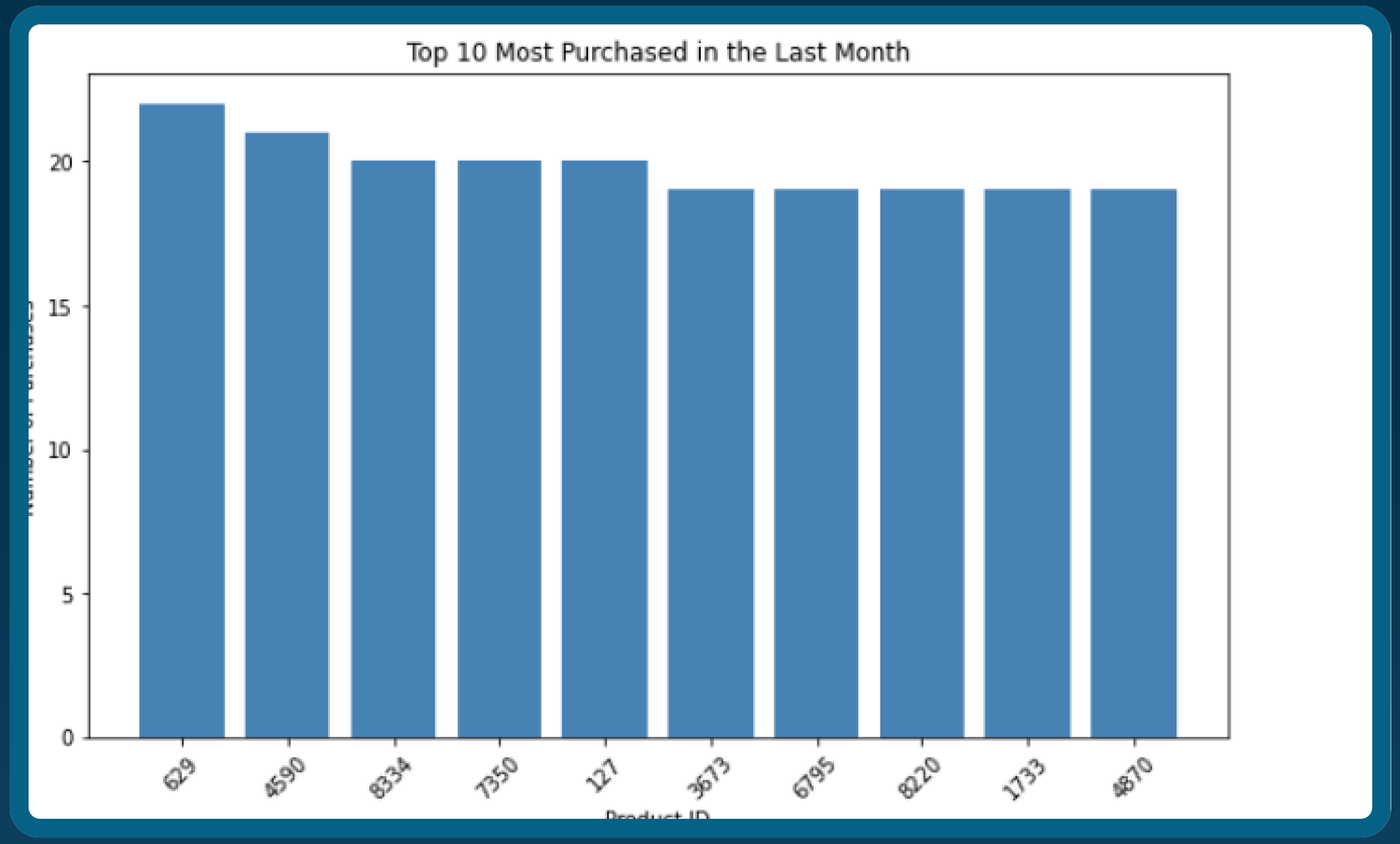
In [8]:

```
output_path_hdfs = "hdfs:///user/student/spark_project/batch_data"

# Save the cleaned transactions data to HDFS in Parquet format
cleaned_transactions_df.write \
    .mode("overwrite") \
    .parquet(f"{output_path_hdfs}/transactions_parquet")

# Save the cleaned inventory data to HDFS
cleaned_inventory_df.write \
    .mode("overwrite") \
    .parquet(f"{output_path_hdfs}/inventory_parquet")

# Save the joined data to HDFS
joined_df.write \
    .mode("overwrite") \
    .parquet(f"{output_path_hdfs}/joined_data_parquet")
```

```
student@192 spark_project]$ hdfs dfs -ls /user/student/spark_project/batch_data/transactions_parquet
ound 7 items
rw-r--r--   1 student student          0 2024-09-06 21:06 /user/student/spark_project/batch_data/transactions_parquet/_SUCCESS
rw-r--r--   1 student student   14434807 2024-09-06 21:06 /user/student/spark_project/batch_data/transactions_parquet/part-00000-14d59d2e-4f32-44be-99d4-138e8f15f273-c000.snappy.parquet
rw-r--r--   1 student student   14146900 2024-09-06 21:06 /user/student/spark_project/batch_data/transactions_parquet/part-00001-14d59d2e-4f32-44be-99d4-138e8f15f273-c000.snappy.parquet
rw-r--r--   1 student student   14182695 2024-09-06 21:06 /user/student/spark_project/batch_data/transactions_parquet/part-00002-14d59d2e-4f32-44be-99d4-138e8f15f273-c000.snappy.parquet
rw-r--r--   1 student student   14217397 2024-09-06 21:06 /user/student/spark_project/batch_data/transactions_parquet/part-00003-14d59d2e-4f32-44be-99d4-138e8f15f273-c000.snappy.parquet
rw-r--r--   1 student student   14188465 2024-09-06 21:06 /user/student/spark_project/batch_data/transactions_parquet/part-00004-14d59d2e-4f32-44be-99d4-138e8f15f273-c000.snappy.parquet
rw-r--r--   1 student student    5924733 2024-09-06 21:06 /user/student/spark_project/batch_data/transactions_parquet/part-00005-14d59d2e-4f32-44be-99d4-138e8f15f273-c000.snappy.parquet
student@192 spark_project]$ hdfs dfs -ls /user/student/spark_project/batch_data/inventory_parquet

ound 2 items
rw-r--r--   1 student student          0 2024-09-06 21:06 /user/student/spark_project/batch_data/inventory_parquet/_SUCCESS
rw-r--r--   1 student student     156715 2024-09-06 21:06 /user/student/spark_project/batch_data/inventory_parquet/part-00000-2a17a017-c712-4214-b8ae-bab52090b9f5-c000.snappy.parquet
student@192 spark_project]$
student@192 spark_project]$ hdfs dfs -ls /user/student/spark_project/batch_data/joined_data_parquet
ound 7 items
rw-r--r--   1 student student          0 2024-09-06 21:06 /user/student/spark_project/batch_data/joined_data_parquet/_SUCCESS
rw-r--r--   1 student student    3974039 2024-09-06 21:06 /user/student/spark_project/batch_data/joined_data_parquet/part-00000-e2982aef-aa76-4ceb-bb59-6ddd0d9bf983-c000.snappy.parquet
rw-r--r--   1 student student    3921009 2024-09-06 21:06 /user/student/spark_project/batch_data/joined_data_parquet/part-00001-e2982aef-aa76-4ceb-bb59-6ddd0d9bf983-c000.snappy.parquet
rw-r--r--   1 student student    3923245 2024-09-06 21:06 /user/student/spark_project/batch_data/joined_data_parquet/part-00002-e2982aef-aa76-4ceb-bb59-6ddd0d9bf983-c000.snappy.parquet
rw-r--r--   1 student student    3923784 2024-09-06 21:06 /user/student/spark_project/batch_data/joined_data_parquet/part-00003-e2982aef-aa76-4ceb-bb59-6ddd0d9bf983-c000.snappy.parquet
rw-r--r--   1 student student    3921699 2024-09-06 21:06 /user/student/spark_project/batch_data/joined_data_parquet/part-00004-e2982aef-aa76-4ceb-bb59-6ddd0d9bf983-c000.snappy.parquet
```

Results and Reporting

Top Products Bar Chart

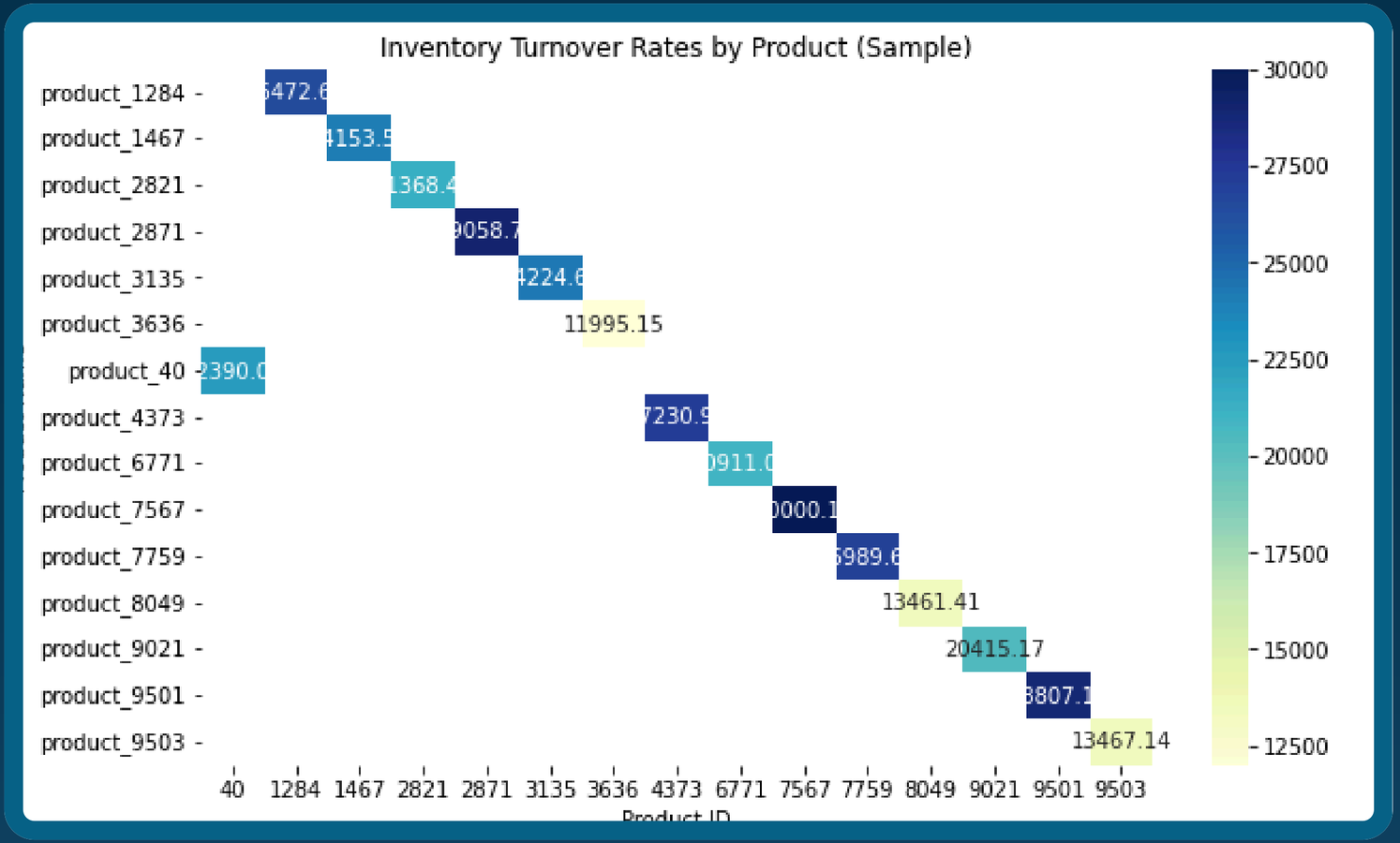


Results and Reporting

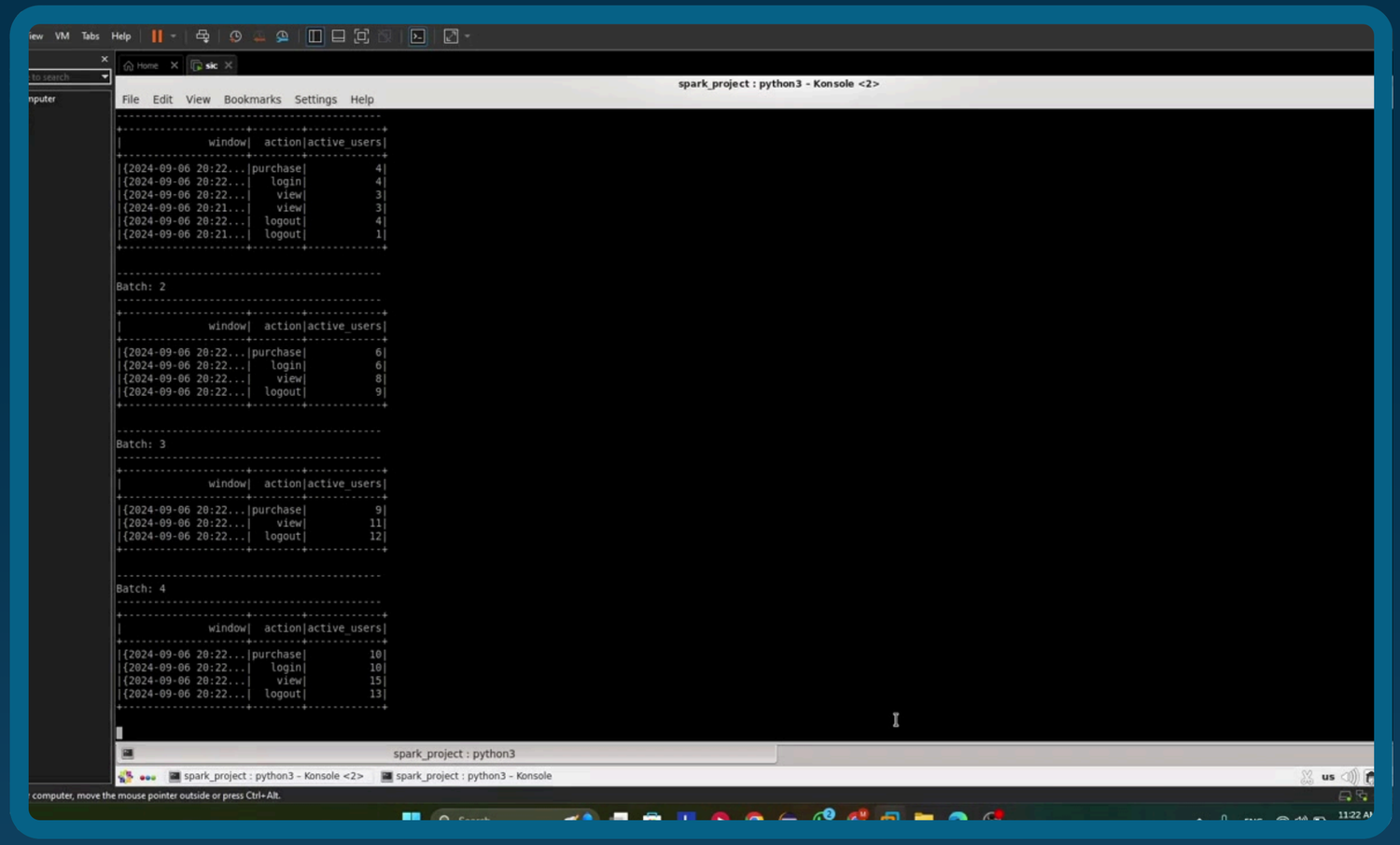
Revenue Trend Line Chart

Results and Reporting

Inventory Turnover Heatmap



Streaming Results



Thank You