Final Report
On
“Shopstream”

Submitted in Partial Full fillment of the Requirements
For the Award
Of
Degree of B.Tech
To

Uttarakhand Technical University, Dehradun

Under the Guidance of
Mrs. Akanksha Bisht
(Asst. Professor)

Submitted By

Naman Shrimali
(210410101075)

Department of Computer Science and Engineering
Shivalik College of Engineering
Shiniwala, P.O. Sherpur, Shimla Road, Dehradun
248197 Uttrakhand, India

Abstract

In the rapidly evolving world of e-commerce, personalized recommendations have become a cornerstone of enhancing user experience and driving business growth. This project, titled "Shopstream: A Scalable Data Pipelining Framework for Real-Time and Batch Processing in Personalized Retail Recommendations", presents a robust solution for delivering personalized product recommendations using state-of-the-art big data technologies.

The framework integrates real-time and batch data processing pipelines, leveraging tools such as Apache Kafka for real-time event streaming and Apache Spark for both real-time and batch data analytics. By analyzing live user interactions (e.g., product views, clicks, and purchases) alongside historical data, the system generates dynamic and contextually relevant recommendations.

A combination of machine learning models, including collaborative filtering, content-based filtering, and hybrid approaches, is employed to enhance recommendation accuracy. Real-time insights are processed through Spark Streaming, while batch processes periodically retrain models to incorporate long-term trends. The recommendations are stored in a scalable database and served through a user-friendly interface or APIs, making them accessible for various use cases.

Key features of the system include scalability to handle growing user traffic, adaptability for evolving user preferences, and integration with dashboards for real-time monitoring of performance metrics. By bridging the gap between real-time and historical data, Shopstream empowers businesses to deliver timely, accurate, and engaging shopping experiences.

This project not only demonstrates the integration of modern big data technologies but also highlights their potential to transform the e-commerce landscape, providing a scalable and efficient blueprint for real-time personalization in retail.

Introduction

In the rapidly evolving landscape of online retail, delivering personalized experiences has become a crucial differentiator for businesses. With consumers inundated by options, companies must leverage sophisticated technologies to stand out. This project, titled "Shopstream," focuses on developing a robust recommendation system that employs data pipelining techniques to provide personalized product suggestions in both real-time and batch processing scenarios.

Background: The evolution of recommendation systems is marked by a transition from simplistic rule-based models to complex machine learning algorithms capable of analyzing vast datasets. Early systems primarily relied on collaborative filtering methods, which recommended products based on user-item interactions. However, as the volume of data and user interactions has surged, the need for more advanced techniques has become apparent.

Recommendation Systems Today: Contemporary recommendation systems leverage both collaborative and content-based filtering methods, utilizing algorithms that analyze user behavior, purchase history, and even external data sources such as social media activity. These systems are not only crucial for enhancing user experience but also serve as a key driver of revenue growth in e-commerce. They can significantly increase conversion rates, with studies showing that personalized recommendations can lead to a substantial increase in sales.

Problem Statement: Despite the advancements in recommendation technologies, many existing systems face challenges in integrating real-time data with historical information. Traditional models often lag in providing timely updates, leading to outdated or irrelevant recommendations. This delay can frustrate users and lead to lost sales opportunities, as consumers may receive suggestions that do not align with their current interests or behaviors.

Purpose of the Project: The primary objective of Shopstream is to create a scalable data pipelining framework that enables both real-time and batch processing of user data for personalized recommendations. By incorporating technologies such as Apache Kafka for real time event streaming and Apache Spark for batch processing, Shopstream aims to deliver dynamic, contextually relevant recommendations based on the most current user interactions. This dual approach ensures that users receive suggestions tailored to their immediate browsing and purchasing activities while also benefiting from insights drawn from historical data.

Significance of the Study: The significance of Shopstream extends beyond its technical capabilities:

- Enhancing Customer Experience: In an era where consumer expectations are high, personalized recommendations can enhance user satisfaction and loyalty. By providing relevant product suggestions, businesses can create a more engaging shopping experience.
- Driving Business Performance: Effective recommendation systems can lead to higher conversion rates, increased average order values, and improved customer retention. By integrating real-time and batch processing, Shopstream can maximize the impact of recommendations on sales performance.
- Contributing to Data Science: The project also offers valuable insights into the integration of various data processing technologies, contributing to the broader field of data science. It showcases how businesses can harness the power of big data to drive informed decision-making.

Objectives

The Shopstream project aims to create a sophisticated recommendation system that leverages data pipelining techniques for both real-time and batch processing. The objectives outlined below provide a clear roadmap for the project's development and implementation:

1.Develop a Scalable Data Pipelining Framework:

• Goal: To design a robust architecture that integrates various data processing technologies, enabling seamless ingestion and processing of real-time and historical user data.

• Details: This involves utilizing Apache Kafka for real-time event streaming and Apache Spark for batch processing to ensure the system can handle large volumes of data efficiently.

2. Implement Real-Time Recommendation Updates:

• Goal: To provide instant, contextually relevant product recommendations based on live user interactions.

• Details: The system should dynamically adjust recommendations based on real-time user behaviors, such as product views, clicks, and purchases, ensuring users receive suggestions that reflect their current interests.

3. Enhance Batch Processing for Model Training:

• Goal: To perform in-depth analytics and retraining of recommendation models using historical data.

• Details: By employing Apache Spark for batch processing, the project will focus on training collaborative filtering models, matrix factorization techniques, and other advanced algorithms on extensive datasets to improve recommendation accuracy.

4. Integrate Multiple Recommendation Algorithms:

• Goal: To combine various recommendation techniques to enhance the personalization of suggestions.

• Details: The system will utilize collaborative filtering (user-based and item-based), content-based filtering, and hybrid models to ensure a diverse set of recommendations that cater to different user profiles and behaviors.

5. Create a User-Friendly Dashboard for Monitoring and Analytics:

• Goal: To develop a dashboard that visualizes key performance metrics of the recommendation system.

• Details: The dashboard will provide insights into user engagement, click-through rates, conversion rates, and overall system performance, allowing stakeholders to make data driven decisions for continuous improvement.

6. Conduct A/B Testing to Evaluate Effectiveness:

• Goal: To assess the performance of the recommendation system and its impact on user engagement and sales.

• Details: Implement A/B testing methodologies to compare the effectiveness of different recommendation algorithms and strategies, refining the system based on empirical data and user feedback.

7. Ensure Scalability and Performance Optimization:

• Goal: To design the system with scalability in mind, accommodating growing data volumes and user traffic.

• Details: This includes implementing microservices architecture, utilizing cloud technologies, and optimizing data processing pipelines to maintain high performance during peak loads.

8. Facilitate Continuous Learning and Model Improvement:

• Goal: To establish a framework for ongoing model updates and enhancements based on user behavior changes and emerging trends.

• Details: The system should support periodic retraining of models using both real-time and batch data to ensure recommendations remain relevant over time.

Proposed Solution:

The **Shopstream** project proposes a comprehensive, scalable, and efficient framework for real-time and batch processing to deliver personalized retail recommendations. The solution addresses the limitations of existing recommendation systems by combining modern big data technologies and advanced machine learning models to offer dynamic, contextually relevant suggestions to users. The core aspects of the proposed solution are outlined as follows:

1. Real-Time Data Ingestion and Processing

- **Challenge**: Many recommendation systems fail to incorporate real-time user interactions, leading to outdated or irrelevant recommendations.
- **Proposed Solution**:
  - Implement **Apache Kafka** to ingest real-time events such as product views, clicks, and purchases.
  - Use **Apache Spark Streaming** to process these events in real time, generating personalized recommendations instantly.
  - Real-time recommendations are stored in a low-latency database (e.g., MongoDB) for immediate serving.

2. Batch Processing for Historical Data

- **Challenge**: Relying solely on real-time data can lead to incomplete insights, while traditional batch models are too static.
- **Proposed Solution**:
  - Utilize **Apache Spark** to perform periodic batch processing on historical data stored in distributed storage (e.g., Hadoop HDFS).
  - Retrain machine learning models using collaborative filtering, content-based filtering, and hybrid approaches.
  - Update the recommendation logic to combine insights from real-time and historical data.

3. Scalable and Robust Architecture

- **Challenge**: Increasing user traffic and data volume can overwhelm traditional recommendation systems.
- **Proposed Solution**:
  - Design a microservices-based architecture with independent components for ingestion, processing, and serving.
  - Ensure scalability using distributed systems like Kafka (for streaming) and Spark (for parallel processing).
  - Deploy on cloud platforms (e.g., AWS, GCP, or Azure) to scale resources dynamically based on demand.

4. Advanced Machine Learning Models

- **Challenge**: Traditional recommendation models struggle with the "cold start problem" and lack adaptability.
- **Proposed Solution**:
  - Use **Collaborative Filtering** for personalized recommendations based on user-item interactions.

        - Leverage **Content-Based Filtering** to recommend similar products based on their attributes.
        - Implement **Hybrid Models** to address cold start issues and enhance accuracy.
        - Continuously retrain models using batch-processed historical data to capture evolving user preferences.

5. Integration with Analytics and Monitoring

- **Challenge**: Businesses lack real-time insights into system performance and user engagement.
- **Proposed Solution**:
    - Develop a user-friendly dashboard using tools like **Power BI** or **Tableau** to visualize metrics such as:
        - Popular products.
        - Real-time recommendation performance.
        - Conversion rates and engagement metrics.
    - Use **Prometheus** and **Grafana** for system monitoring to ensure reliability and identify bottlenecks.

6. Serving Recommendations

- **Challenge**: Delivering recommendations in a timely and user-accessible manner.
- **Proposed Solution**:
    - Store recommendations in a NoSQL database (e.g., MongoDB) for fast retrieval.
    - Provide RESTful APIs for integration with web or mobile applications.
    - Design a front-end interface to display recommendations dynamically, tailored to user behavior.

Key Benefits of the Proposed Solution

1. **Real-Time Personalization**:
    - Delivers contextually relevant recommendations immediately based on user actions.
2. **Improved Accuracy**:
    - Combines real-time and historical insights for better predictive performance.
3. **Scalability**:
    - Handles growing user traffic and data volume efficiently.
4. **Business Insights**:
    - Offers detailed analytics for better decision-making.
5. **User Satisfaction**:
    - Enhances shopping experiences by providing tailored product suggestions.

Rationale:

The rationale for developing the Shopstream project stems from the critical need for enhanced personalization in online retail and the challenges faced by existing recommendation systems. As consumer behavior evolves, businesses must adapt to meet rising expectations for tailored shopping experiences. This section outlines the motivations behind Shopstream, including market trends, technological advancements, and the limitations of current systems.

1. Market Trends in Online Retail: The online retail landscape has witnessed significant growth, with consumers increasingly turning to e-commerce for their shopping needs. According to industry reports, online sales have surged, especially in the wake of global events such as the COVID-19 pandemic, which accelerated the shift to digital shopping. As competition intensifies, retailers are recognizing the importance of personalized customer experiences as a key driver of success.

• Changing Consumer Expectations: Modern consumers expect personalized interactions that resonate with their preferences and needs. They seek tailored product suggestions, targeted promotions, and seamless user experiences. Failure to meet these expectations can result in abandoned carts and lost sales.

• Increased Data Availability: The volume of data generated by user interactions has exploded, creating opportunities for businesses to leverage this information for more effective marketing strategies. However, capitalizing on this data requires advanced systems capable of real-time analysis and insights.

2. Limitations of Existing Recommendation Systems: While many retailers have implemented recommendation systems, significant gaps remain in their effectiveness:

• Static Recommendations: Many traditional systems primarily rely on historical data, leading to static recommendations that do not account for real-time user behavior. For instance, a user may receive suggestions based on past purchases, but these recommendations may not reflect their current interests or browsing patterns.

• Integration Challenges: Existing systems often struggle to integrate real-time data streams with batch processing, leading to delays in updating recommendations. This can frustrate users, who may find that suggested products do not align with their immediate preferences.

• Algorithmic Limitations: Common recommendation algorithms, such as collaborative filtering, may fail to provide relevant suggestions for new users or items (the "cold start" problem). This limitation further underscores the need for more sophisticated and adaptable recommendation frameworks.

3. Technological Advancements: Recent advancements in data processing technologies present an opportunity to address the limitations of existing systems:

• Data Pipelining: The emergence of data pipelining frameworks, such as Apache Kafka and Apache Spark, allows for the efficient handling of large volumes of real-time and batch data. These technologies enable businesses to ingest, process, and analyze data streams seamlessly, paving the way for more dynamic recommendation systems.

• Machine Learning Techniques: The evolution of machine learning has opened new avenues for improving recommendation accuracy. Techniques such as deep learning and hybrid models can analyze complex patterns in user behavior and preferences, leading to more relevant and personalized product suggestions.

4. Strategic Importance for Retailers: The strategic implications of implementing an advanced recommendation system like Shopstream are profound:

• Competitive Advantage: By adopting a scalable and efficient recommendation framework, retailers can differentiate themselves in a crowded market. Personalized experiences foster customer loyalty and retention, which are essential for long-term success.

• Increased Revenue: Studies have shown that effective recommendation systems can significantly boost conversion rates and average order values. By providing timely and relevant product suggestions, retailers can capitalize on upselling and cross-selling opportunities.

• Data-Driven Decision Making: The insights gained from real-time and batch data processing can inform broader marketing strategies and product offerings. Retailers can better understand customer preferences and trends, enabling them to tailor their inventory and promotions accordingly.

Methodology

1. Research and Data Collection:

   • Identify and catalogue user interaction data sources, including clickstreams, purchase histories, and product views.

   • Compile a comprehensive database that organizes this data for easy access and analysis.

2. System Architecture Design:

   • Design the overall architecture using Apache Kafka for real-time data ingestion and Apache Spark for batch processing.

   • Create a data flow diagram to illustrate how data will move through the system from ingestion to processing and storage.

3. Data Ingestion Setup:

   • Implement Apache Kafka to capture real-time user events, such as clicks and purchases.

   • Develop ETL processes to clean and transform historical data for batch processing in Spark.

4. Implementation of Recommendation Algorithms:

   • Integrate collaborative filtering techniques (user-based and item-based) to generate personalized recommendations.

   • Develop content-based filtering algorithms that utilize product attributes for improved suggestion accuracy.

5. Real-Time Processing Integration:

   • Utilize Spark Streaming to process incoming data from Kafka and update recommendations in real-time.

   • Implement session-based personalization logic that adjusts recommendations based on current user behavior.

6. Dashboard Development:

   • Create a user-friendly dashboard using Looker or Power BI to visualize key metrics like click-through rates and user engagement.

   • Integrate real-time data from MongoDB for immediate insights into recommendation performance.

7. Testing and Quality Assurance:

   • Conduct unit tests and integration tests to validate the functionality of individual components.

• Implement A/B testing to evaluate the effectiveness of different recommendation strategies and gather user feedback.

8. Deployment and Scaling:

• Deploy the application using Docker containers to ensure consistent environments across development and production.

• Utilize Kubernetes for orchestration and scaling of microservices to handle increased traffic and data loads.

9. Continuous Improvement:

• Establish a periodic retraining process for recommendation models to incorporate new user data and trends.

• Regularly review performance metrics and user feedback to make iterative improvements to the system.

Feasibility Study:

The "AI Scout" project demonstrates high technical feasibility, leveraging modern web development technologies like HTML5, CSS3, JavaScript, Django, and Flask. These tools enable the creation of a user-friendly interface and robust backend infrastructure to support features such as tool categorization and user authentication. Logistically, the project requires access to reliable internet connectivity and hosting services for deployment and maintenance.

Collaboration tools such as Git and project management platforms ensure effective coordination among team members. Adequate hardware resources, including computers with sufficient processing power, support development and testing activities. Financially, the project can minimize upfront costs by utilizing open-source technologies. Potential revenue streams, such as advertising or premium features, may sustain long-term operations.

However, legal and ethical considerations, including data protection regulations and intellectual property rights, require careful attention to ensure compliance and mitigate risks. Overall, the "AI Scout" project demonstrates strong feasibility, poised to deliver valuable benefits to developers and researchers in the AI community.

Facilities required:

To successfully develop and implement the Shopstream project, several facilities and resources are necessary. These include technical infrastructure, tools, and human resources, as outlined below:

1. Technical Infrastructure:
    - o Cloud Services or On-Premises Servers:
        - ▪ Access to scalable cloud infrastructure (e.g., AWS, Google Cloud, or Azure) to host applications, databases, and data processing tools.
        - ▪ Alternatively, high-performance on-premises servers capable of handling data storage and processing needs.
2. Data Storage Solutions:
    - o Database Management Systems:
    - ▪ PostgreSQL or MySQL for structured data storage, such as user profiles and transactional data.
    - ▪ MongoDB for storing real-time user interaction data and dynamic recommendations.
    - ▪ Apache Hadoop for batch processing and storage of large datasets.
3. Development Tools:
    - o Programming Languages:
    - ▪ Python for implementing machine learning algorithms and data processing tasks.
    - ▪ Java or Scala for developing applications with Apache Kafka and Spark.
    - o Integrated Development Environment (IDE):
    - ▪ Access to IDEs such as PyCharm or Visual Studio Code for code development and debugging.
4. Data Processing Frameworks:
    - o Apache Kafka:
    - ▪ Set up for real-time data streaming and event processing.
    - o Apache Spark:
    - ▪ Installed for batch processing and analytics, enabling efficient handling of large datasets.

5. Machine Learning Libraries:

o Scikit-Learn and TensorFlow:

▪ Libraries required for developing and training machine learning models for recommendations.

6. Dashboard and Visualization Tools:

o Business Intelligence Software:

▪ Tools like Looker or Power BI for creating dashboards that visualize performance metrics and insights.

7. Testing and Quality Assurance Tools:

o Automated Testing Frameworks:

▪ Tools for conducting unit tests, integration tests, and performance testing to ensure the system's reliability.

8. Miscellaneous:

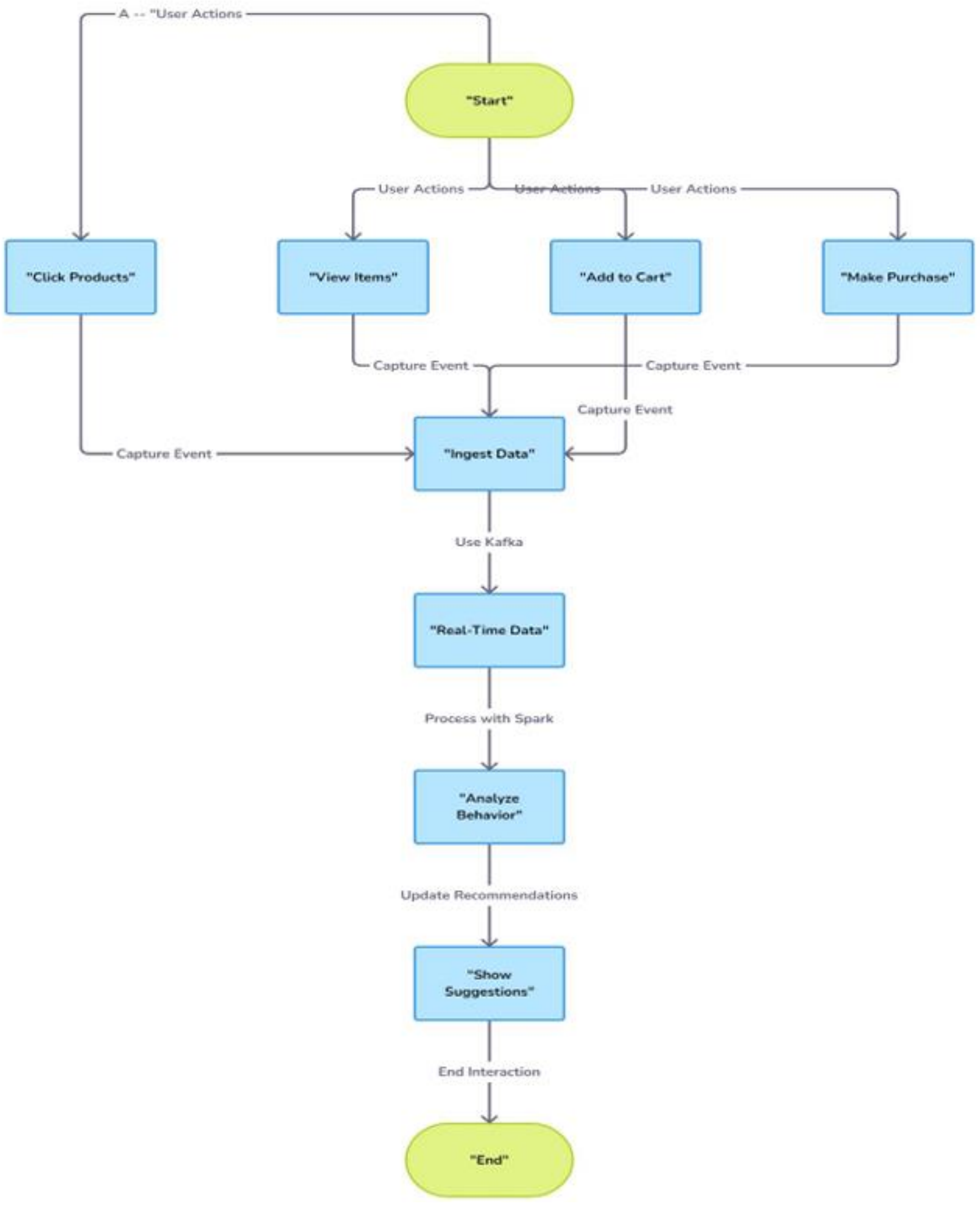
o Version Control Systems:

▪ Tools like Git for managing code changes and collaboration among team members.

o Communication Tools:

▪ Platforms like Slack or Microsoft Teams for team collaboration and updates throughout the project lifecycle.

Flowchart:


A -- "User Actions
"Start"
User Actions
~~User Actions~~
User Actions
"Click Products"
"View Items"
"Add to Cart"
"Make Purchase"
Capture Event
Capture Event
Capture Event
Capture Event
"Ingest Data"
Use Kafka
"Real-Time Data"
Process with Spark
"Analyze Behavior"
Update Recommendations
"Show Suggestions"
End Interaction
"End"

<u>Outcomes:</u>

The Shopstream project aims to achieve quantifiable impacts that enhance the online retail experience. The expected outcomes are outlined below, with measurable targets for each:

1. Personalized User Experience:
    - o Dynamic Recommendations: Aim for a 30% increase in user engagement metrics (time spent on site and pages viewed) through personalized recommendations.
    - o Increased Session Duration: Target an average session duration increase from 5 minutes to 7 minutes per user.
2. Improved Conversion Rates:
    - o Higher Click-Through Rates (CTR): Achieve a CTR increase of 15% for recommended products.
    - o Sales Growth: Project a 20% increase in sales attributed to personalized recommendations within the first quarter post-implementation.
3. Data-Driven Insights:
    - o Analytics Dashboard: Provide real-time analytics that can generate weekly reports, improving decision-making speed by 50%.
    - o User Segmentation: Establish at least 5 distinct customer segments based on purchasing behavior for targeted marketing efforts.
4. Scalability and Adaptability:
    - o Architecture Efficiency: Ensure the system can handle a 100% increase in user traffic without performance degradation.
    - o Continuous Model Learning: Implement retraining processes that update models monthly, improving recommendation accuracy by at least 10% with each iteration.
5. Enhanced Operational Efficiency:
    - o Automated Recommendations: Reduce the time to generate and update recommendations by 40%, allowing for faster response to user behavior changes.
    - o Cost Optimization: Expect a 15% reduction in operational costs through optimized resource usage in data processing.
6. Testing and Validation:
    - o Rigorous Testing Success Rate: Achieve a testing success rate of 95% for functionality, performance, and security metrics.

o A/B Testing Insights: Gather actionable insights that lead to a 10% increase in user satisfaction scores.

7. User Satisfaction and Loyalty:

    o Customer Satisfaction Score (CSAT): Target a CSAT increase from 75% to 85% within six months post-launch.

    o Repeat Purchase Rate: Aim for a 25% increase in repeat purchases driven by enhanced user experience.

8. Documentation and Knowledge Transfer:

    o Comprehensive Documentation: Create detailed documentation that facilitates onboarding new team members, reducing training time by 30%.

    o Skill Development: Ensure at least 3 team members gain advanced skills in data processing and machine learning.

Implementation and Code for the Shopstream:

- Model Code:

```
import pandas as pd
aisles = pd.read_csv("aisles.csv")
departments = pd.read_csv("departments.csv")
orders = pd.read_csv("orders.csv")
products = pd.read_csv("products.csv")
order_products_prior = pd.read_csv("order_products__prior.csv")
order_products_train = pd.read_csv("order_products__train.csv")



print("Aisles:\n", aisles.head())
print("Departments:\n", departments.head())
print("Orders:\n", orders.head())
print("Products:\n", products.head())
print("Order Products Prior:\n", order_products_prior.head())
print("Order Products Train:\n", order_products_train.head())



import matplotlib.pyplot as plt
import seaborn as sns

# Check for missing values
for df, name in zip([aisles, departments, orders, products, order_products_prior, order_products_train],
                    ["Aisles", "Departments", "Orders", "Products", "Order Products Prior", "Order Products Train"]):
    print(f"{name} Missing Values:\n", df.isnull().sum())

# Top 10 most popular products
popular_products = order_products_prior.groupby("product_id")["add_to_cart_order"].count().nlargest(10)
popular_products = popular_products.reset_index().merge(products, on="product_id",
how="left")


plt.figure(figsize=(10, 5))

sns.barplot(data=popular_products, x="product_name", y="add_to_cart_order")

plt.xticks(rotation=45)

plt.title("Top 10 Popular Products")

plt.show()
```

Feature Engineering :-

```
# Merge products with aisles and departments

products = products.merge(aisles, on="aisle_id", how="left").merge(departments,
on="department_id", how="left")


# Combine orders and prior order products

order_details = order_products_prior.merge(products, on="product_id",
how="left").merge(orders, on="order_id", how="left")


# Save the combined dataset

order_details.to_csv("combined_order_details.csv", index=False)

print("Combined Dataset Shape:", order_details.shape)


# User-level features

user_features = orders.groupby("user_id").agg({

   "order_id": "count",  # Total orders

   "days_since_prior_order": "mean"  # Average reorder time

}).rename(columns={

   "order_id": "total_orders",

   "days_since_prior_order": "avg_days_between_orders"

}).reset_index()


# Product-level features

product_features = order_products_prior.groupby("product_id").agg({

   "order_id": "count",  # Total purchases
```

```
    "reordered": "sum"    # Total reorders
}).rename(columns={
    "order_id": "purchase_count",
    "reordered": "total_reorders"
}).reset_index()

print("User Features:\n", user_features.head())
print("Product Features:\n", product_features.head())

# Generate labels
train = order_products_train.merge(orders, on="order_id", how="left").merge(products,
on="product_id", how="left")
train["reordered"] = train["reordered"].fillna(0)

print("Training Data Shape:", train.shape)
print(train.head())

# Ensure proper merging of products with aisles and departments
products = products.merge(aisles, on="aisle_id", how="left").merge(departments,
on="department_id", how="left")

# Merge training data with products
train = train.merge(products, on="product_id", how="left")

# Merge training data with user and product features
train_data = train.merge(user_features, on="user_id",
how="left").merge(product_features, on="product_id", how="left")

# Adjust the drop statement dynamically
drop_columns = ["reordered", "user_id"]
if "product_name" in train_data.columns:
    drop_columns.append("product_name")
```

```
# Drop unnecessary columns
X = train_data.drop(drop_columns, axis=1)
y = train_data["reordered"]

# Check shapes
print("Feature Matrix Shape:", X.shape)
print("Target Variable Shape:", y.shape)
```

Model Preparations:

```
# Generate labels
train = order_products_train.merge(orders, on="order_id", how="left").merge(products,
on="product_id", how="left")
train["reordered"] = train["reordered"].fillna(0)

print("Training Data Shape:", train.shape)
print(train.head())
```

Model Training:

```
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import LabelEncoder
import pandas as pd

# If X is already a pandas DataFrame, proceed to impute missing values and handle
categorical encoding
if isinstance(X, pd.DataFrame):
    # Label encoding for categorical features in X
    label_encoder = LabelEncoder()
    for column in X.select_dtypes(include=['object']).columns:
        X[column] = label_encoder.fit_transform(X[column])
```

```
# Imputer to replace NaN with the mean for numerical columns (or mode for categorical)
imputer = SimpleImputer(strategy='mean')  # 'mean', 'median', or 'most_frequent'

# Apply imputer to handle missing data in 'X' and 'y'
X_imputed = imputer.fit_transform(X)

# If y has missing values, impute them similarly
y_imputed = imputer.fit_transform(y.values.reshape(-1, 1)).ravel()  # Impute if needed

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y_imputed, test_size=0.2, random_state=42)

# Train a Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predictions and evaluation
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

REAL TIME PROCESSING :

- Kafka Producer :

```
from kafka import KafkaProducer
import json
import time

# Create a Kafka producer
producer = KafkaProducer(
    bootstrap_servers='localhost:9092',
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

# Sample user interaction data
events = [
    {"user_id": 1, "product_id": 101, "event_type": "view"},
    {"user_id": 2, "product_id": 102, "event_type": "purchase"},
    {"user_id": 1, "product_id": 103, "event_type": "view"}
]

# Send events to Kafka
for event in events:
    producer.send('user_interactions', value=event)
    print(f"Sent: {event}")
    time.sleep(1)  # Simulate real-time events

producer.close()
```

- Kafka Spark Consumer:

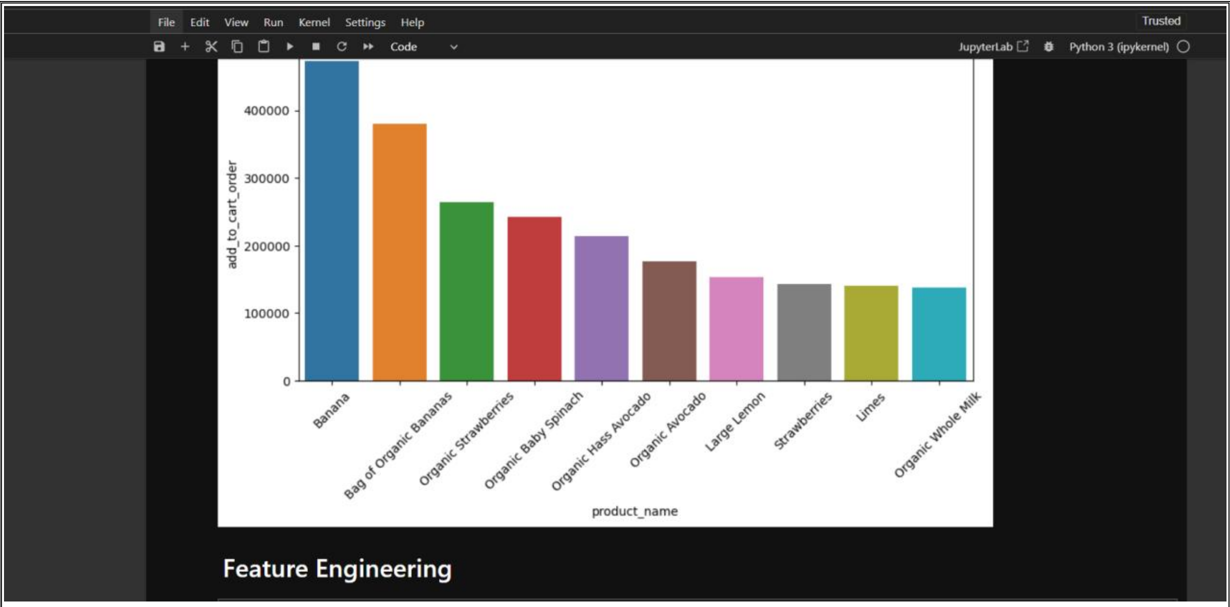
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col
from pyspark.sql.types import StructType, StructField, StringType, IntegerType

# Start a Spark session
spark = SparkSession.builder \
    .appName("RealTimeRecommendation") \
    .getOrCreate()

# Define schema for Kafka messages
schema = StructType([
    StructField("user_id", IntegerType(), True),
    StructField("product_id", IntegerType(), True),
    StructField("event_type", StringType(), True)
])
```

```
# Read messages from Kafka
df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "user_interactions") \
    .load()

# Parse Kafka messages
parsed_df = df.selectExpr("CAST(value AS STRING)") \
    .select(from_json(col("value"), schema).alias("data")) \
    .select("data.*")

# Perform real-time processing (e.g., filtering purchase events)
processed_df = parsed_df.filter(col("event_type") == "purchase")

# Write the processed data back to a Kafka topic
processed_df.writeStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("topic", "real_time_recommendations") \
    .start() \
    .awaitTermination()
```

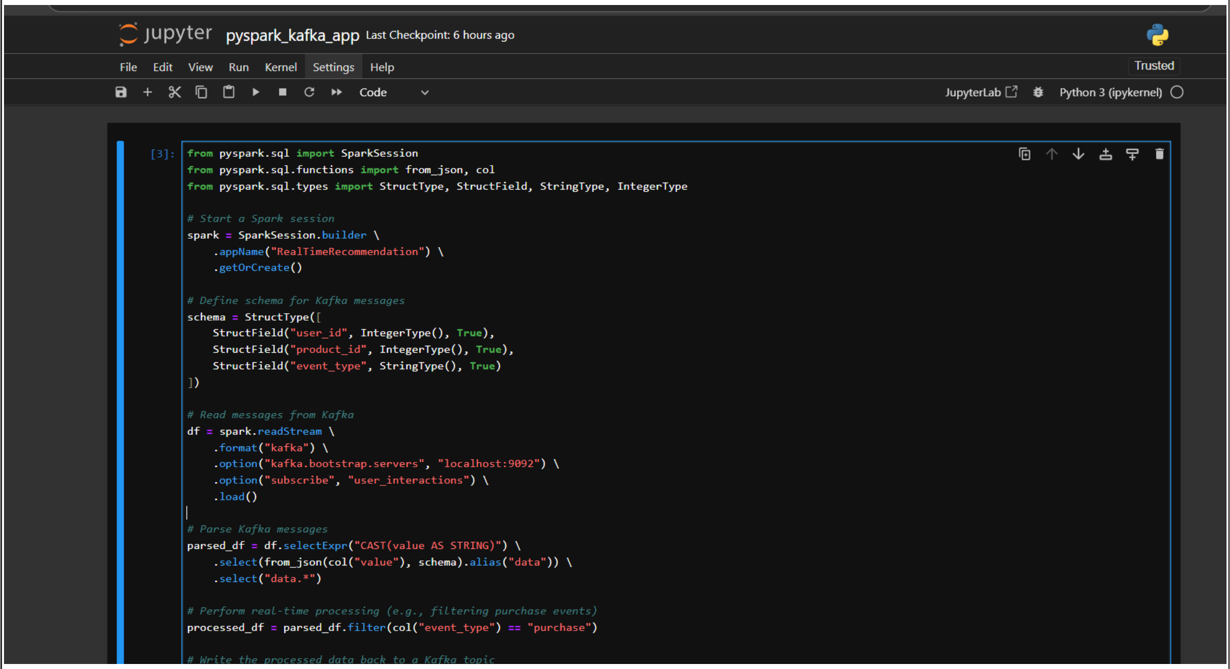
BATCH PROCESSING CODE:

```
from pyspark.sql import SparkSession
from pyspark.ml.recommendation import ALS
from pyspark.sql.functions import col
from pyspark.ml.evaluation import RegressionEvaluator

# Initialize Spark session
spark = SparkSession.builder \
    .master("local[*]") \
    .appName("ALS Recommendation") \
    .getOrCreate()

# Load the CSV files
aisles = spark.read.csv("aisles.csv", header=True, inferSchema=True)
departments = spark.read.csv("departments.csv", header=True, inferSchema=True)
orders = spark.read.csv("orders.csv", header=True, inferSchema=True)
products = spark.read.csv("products.csv", header=True, inferSchema=True)
order_products_prior = spark.read.csv("order_products__prior.csv", header=True,
inferSchema=True)

# Show data preview (optional)
print("Aisles")
aisles.show(5)
print("Departments")
departments.show(5)
print("Orders")
orders.show(5)
print("Products")
products.show(5)
print("order_products_prior")
order_products_prior.show(5)

# Merge the datasets based on common columns (such as product_id, order_id, etc.)
# Join order_products_prior with products to get product details
order_products_prior = order_products_prior.join(products, "product_id")

# Optionally, join with aisles or departments if needed (for product categorization info)
order_products_prior = order_products_prior.join(aisles, "aisle_id", how="left") \
                                           .join(departments, "department_id", how="left")

# Now we have a dataset that contains user_id, product_id, and the rating (order data)
# If the dataset does not contain ratings directly, we can use the order information to create
implicit feedback

# Example of creating implicit feedback (1 for each product purchase)
order_products_prior = order_products_prior.withColumn("rating", col("reordered"))

# Join with the orders data to get user_id
order_products_prior = order_products_prior.join(orders, "order_id")

# Filter necessary columns
data = order_products_prior.select("user_id", "product_id", "rating")

# Show the merged data preview
print("Filterd Data")
data.show(5)

# Split the data into training and test sets
(training_data, test_data) = data.randomSplit([0.8, 0.2])

# Initialize the ALS model
als = ALS(maxIter=10, regParam=0.1, userCol="user_id", itemCol="product_id",
ratingCol="rating", coldStartStrategy="drop")

# Fit the ALS model
model = als.fit(training_data)

# Generate predictions for the test data
predictions = model.transform(test_data)

# Show the predictions (optional)
print("Predictions")
predictions.show(5)
```

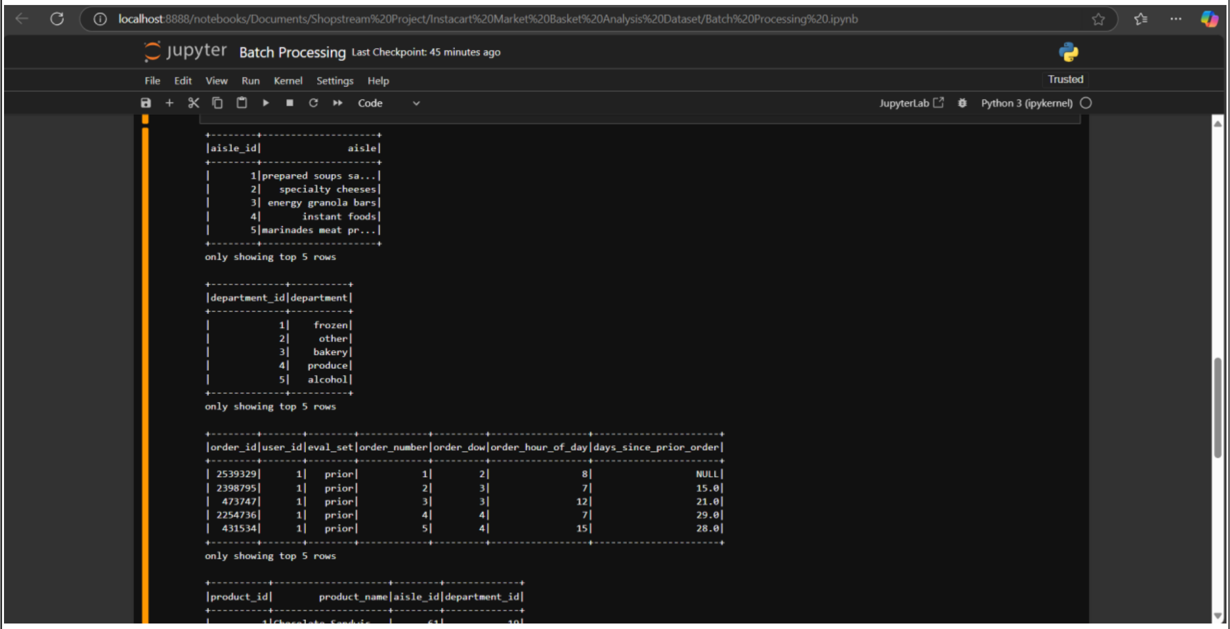
```
# Evaluate the model (using RMSE as the metric)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
predictionCol="prediction")
rmse = evaluator.evaluate(predictions)


# Print RMSE value
print(f"Root Mean Squared Error (RMSE) = {rmse}")


rom pyspark.sql.functions import col


# Prepare the data
# Here we are assuming that `order_products_prior` has columns `order_id`, `product_id`, and
`reordered` (which acts as the rating)
ratings = order_products_prior.select(col("user_id"), col("product_id"),
col("reordered").cast("float"))


# ALS requires column names to be `user`, `item` and `rating`


ratings = ratings.withColumnRenamed("user_id", "user").withColumnRenamed("product_id",
"item")


all_user_recs = model.recommendForAllUsers(10)


all_user_recs.show(truncate=False)
```



Feature Engineering

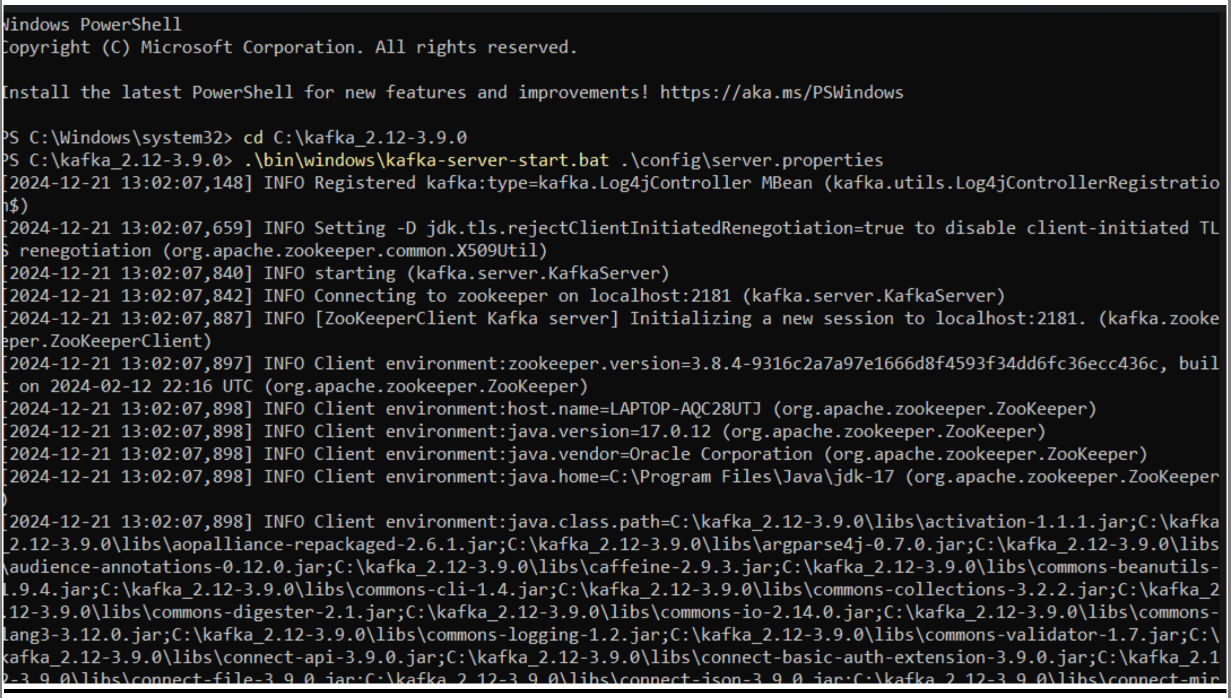


```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col
from pyspark.sql.types import StructType, StructField, StringType, IntegerType

# Start a Spark session
spark = SparkSession.builder \
    .appName("RealTimeRecommendation") \
    .getOrCreate()

# Define schema for Kafka messages
schema = StructType([
    StructField("user_id", IntegerType(), True),
    StructField("product_id", IntegerType(), True),
    StructField("event_type", StringType(), True)
])

# Read messages from Kafka
df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "user_interactions") \
    .load()

# Parse Kafka messages
parsed_df = df.selectExpr("CAST(value AS STRING)") \
    .select(from_json(col("value"), schema).alias("data")) \
    .select("data.*")

# Perform real-time processing (e.g., filtering purchase events)
processed_df = parsed_df.filter(col("event_type") == "purchase")

# Write the processed data back to a Kafka topic
```

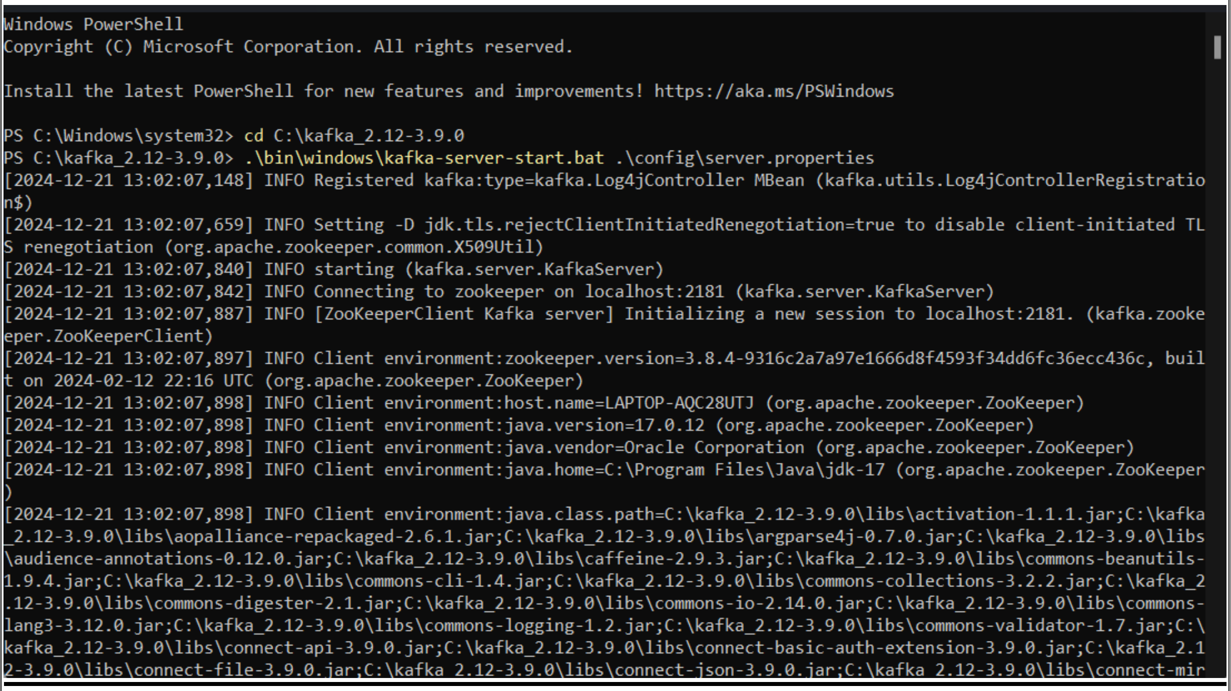

localhost:8888/notebooks/Documents/Shopstream%20Project/Instacart%20Market%20Basket%20Analysis%20Dataset/Batch%20Processing%20.ipynb
jupyter
Batch Processing
Last Checkpoint: 45 minutes ago
File
Edit
View
Run
Kernel
Settings
Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

```
+--------+--------------------+
|aisle_id|               aisle|
+--------+--------------------+
|       1|prepared soups sa...|
|       2|   specialty cheeses|
|       3| energy granola bars|
|       4|       instant foods|
|       5|marinades meat pr...|
+--------+--------------------+
only showing top 5 rows

+-------------+----------+
|department_id|department|
+-------------+----------+
|            1|    frozen|
|            2|     other|
|            3|    bakery|
|            4|   produce|
|            5|   alcohol|
+-------------+----------+
only showing top 5 rows

+--------+-------+--------+------------+---------+-----------------+----------------------+
|order_id|user_id|eval_set|order_number|order_dow|order_hour_of_day|days_since_prior_order|
+--------+-------+--------+------------+---------+-----------------+----------------------+
| 2539329|      1|   prior|           1|        2|                8|                  NULL|
| 2398795|      1|   prior|           2|        3|                7|                  15.0|
|  473747|      1|   prior|           3|        3|               12|                  21.0|
| 2254736|      1|   prior|           4|        4|                7|                  29.0|
|  431534|      1|   prior|           5|        4|               15|                  28.0|
+--------+-------+--------+------------+---------+-----------------+----------------------+
only showing top 5 rows

+----------+--------------------+--------+-------------+
|product_id|        product_name|aisle_id|department_id|
+----------+--------------------+--------+-------------+
```

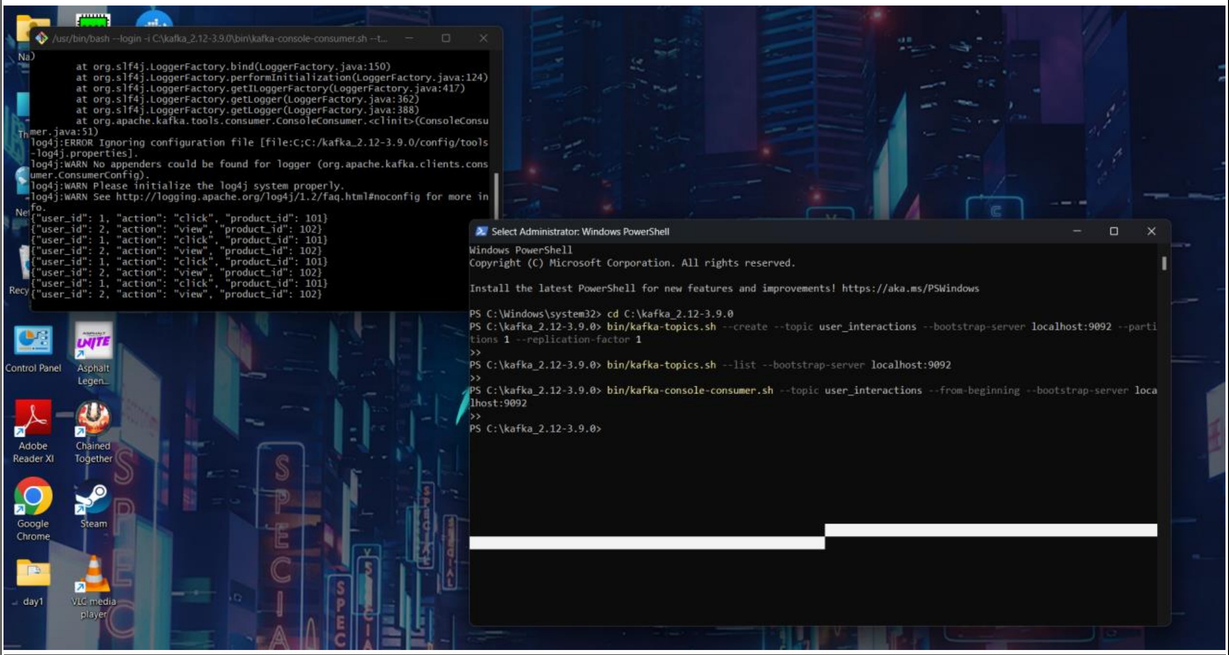
Shopstream in Action:



```
lindows PowerShell
:opyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Windows\system32> cd C:\kafka_2.12-3.9.0
PS C:\kafka_2.12-3.9.0> .\bin\windows\kafka-server-start.bat .\config\server.properties
[2024-12-21 13:02:07,148] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistratio
n$)
[2024-12-21 13:02:07,659] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TL
S renegotiation (org.apache.zookeeper.common.X509Util)
[2024-12-21 13:02:07,840] INFO starting (kafka.server.KafkaServer)
[2024-12-21 13:02:07,842] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2024-12-21 13:02:07,887] INFO [ZooKeeperClient Kafka server] Initializing a new session to localhost:2181. (kafka.zooke
eper.ZooKeeperClient)
[2024-12-21 13:02:07,897] INFO Client environment:zookeeper.version=3.8.4-9316c2a7a97e1666d8f4593f34dd6fc36ecc436c, buil
t on 2024-02-12 22:16 UTC (org.apache.zookeeper.ZooKeeper)
[2024-12-21 13:02:07,898] INFO Client environment:host.name=LAPTOP-AQC28UTJ (org.apache.zookeeper.ZooKeeper)
[2024-12-21 13:02:07,898] INFO Client environment:java.version=17.0.12 (org.apache.zookeeper.ZooKeeper)
[2024-12-21 13:02:07,898] INFO Client environment:java.vendor=Oracle Corporation (org.apache.zookeeper.ZooKeeper)
[2024-12-21 13:02:07,898] INFO Client environment:java.home=C:\Program Files\Java\jdk-17 (org.apache.zookeeper.ZooKeeper
)
[2024-12-21 13:02:07,898] INFO Client environment:java.class.path=C:\kafka_2.12-3.9.0\libs\activation-1.1.1.jar;C:\kafka
_2.12-3.9.0\libs\aopalliance-repackaged-2.6.1.jar;C:\kafka_2.12-3.9.0\libs\argparse4j-0.7.0.jar;C:\kafka_2.12-3.9.0\libs
\audience-annotations-0.12.0.jar;C:\kafka_2.12-3.9.0\libs\caffeine-2.9.3.jar;C:\kafka_2.12-3.9.0\libs\commons-beanutils-
1.9.4.jar;C:\kafka_2.12-3.9.0\libs\commons-cli-1.4.jar;C:\kafka_2.12-3.9.0\libs\commons-collections-3.2.2.jar;C:\kafka_2
.12-3.9.0\libs\commons-digester-2.1.jar;C:\kafka_2.12-3.9.0\libs\commons-io-2.14.0.jar;C:\kafka_2.12-3.9.0\libs\commons-
lang3-3.12.0.jar;C:\kafka_2.12-3.9.0\libs\commons-logging-1.2.jar;C:\kafka_2.12-3.9.0\libs\commons-validator-1.7.jar;C:\
kafka_2.12-3.9.0\libs\connect-api-3.9.0.jar;C:\kafka_2.12-3.9.0\libs\connect-basic-auth-extension-3.9.0.jar;C:\kafka_2.1
2-3.9.0\libs\connect-file-3.9.0.jar;C:\kafka_2.12-3.9.0\libs\connect-json-3.9.0.jar;C:\kafka_2.12-3.9.0\libs\connect-mir
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Windows\system32> cd C:\kafka_2.12-3.9.0
PS C:\kafka_2.12-3.9.0> .\bin\windows\kafka-server-start.bat .\config\server.properties
[2024-12-21 13:02:07,148] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistratio
n$)
[2024-12-21 13:02:07,659] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TL
S renegotiation (org.apache.zookeeper.common.X509Util)
[2024-12-21 13:02:07,840] INFO starting (kafka.server.KafkaServer)
[2024-12-21 13:02:07,842] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2024-12-21 13:02:07,887] INFO [ZooKeeperClient Kafka server] Initializing a new session to localhost:2181. (kafka.zooke
eper.ZooKeeperClient)
[2024-12-21 13:02:07,897] INFO Client environment:zookeeper.version=3.8.4-9316c2a7a97e1666d8f4593f34dd6fc36ecc436c, buil
t on 2024-02-12 22:16 UTC (org.apache.zookeeper.ZooKeeper)
[2024-12-21 13:02:07,898] INFO Client environment:host.name=LAPTOP-AQC28UTJ (org.apache.zookeeper.ZooKeeper)
[2024-12-21 13:02:07,898] INFO Client environment:java.version=17.0.12 (org.apache.zookeeper.ZooKeeper)
[2024-12-21 13:02:07,898] INFO Client environment:java.vendor=Oracle Corporation (org.apache.zookeeper.ZooKeeper)
[2024-12-21 13:02:07,898] INFO Client environment:java.home=C:\Program Files\Java\jdk-17 (org.apache.zookeeper.ZooKeeper
)
[2024-12-21 13:02:07,898] INFO Client environment:java.class.path=C:\kafka_2.12-3.9.0\libs\activation-1.1.1.jar;C:\kafka
_2.12-3.9.0\libs\aopalliance-repackaged-2.6.1.jar;C:\kafka_2.12-3.9.0\libs\argparse4j-0.7.0.jar;C:\kafka_2.12-3.9.0\libs
\audience-annotations-0.12.0.jar;C:\kafka_2.12-3.9.0\libs\caffeine-2.9.3.jar;C:\kafka_2.12-3.9.0\libs\commons-beanutils-
1.9.4.jar;C:\kafka_2.12-3.9.0\libs\commons-cli-1.4.jar;C:\kafka_2.12-3.9.0\libs\commons-collections-3.2.2.jar;C:\kafka_2
.12-3.9.0\libs\commons-digester-2.1.jar;C:\kafka_2.12-3.9.0\libs\commons-io-2.14.0.jar;C:\kafka_2.12-3.9.0\libs\commons-
lang3-3.12.0.jar;C:\kafka_2.12-3.9.0\libs\commons-logging-1.2.jar;C:\kafka_2.12-3.9.0\libs\commons-validator-1.7.jar;C:\
kafka_2.12-3.9.0\libs\connect-api-3.9.0.jar;C:\kafka_2.12-3.9.0\libs\connect-basic-auth-extension-3.9.0.jar;C:\kafka_2.1
2-3.9.0\libs\connect-file-3.9.0.jar;C:\kafka_2.12-3.9.0\libs\connect-json-3.9.0.jar;C:\kafka_2.12-3.9.0\libs\connect-mir
```



```
/usr/bin/bash --login -i C:\kafka_2.12-3.9.0\bin\kafka-console-consumer.sh --t...
        at org.slf4j.LoggerFactory.bind(LoggerFactory.java:150)
        at org.slf4j.LoggerFactory.performInitialization(LoggerFactory.java:124)
        at org.slf4j.LoggerFactory.getILoggerFactory(LoggerFactory.java:417)
        at org.slf4j.LoggerFactory.getLogger(LoggerFactory.java:362)
        at org.slf4j.LoggerFactory.getLogger(LoggerFactory.java:388)
        at org.apache.kafka.tools.consumer.ConsoleConsumer.<clinit>(ConsoleConsu
mer.java:51)
log4j:ERROR Ignoring configuration file [file:C;C:/kafka_2.12-3.9.0/config/tools
-log4j.properties].
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.cons
umer.ConsumerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more in
fo.
{"user_id": 1, "action": "click", "product_id": 101}
{"user_id": 2, "action": "view", "product_id": 102}
{"user_id": 1, "action": "click", "product_id": 101}
{"user_id": 2, "action": "view", "product_id": 102}
{"user_id": 1, "action": "click", "product_id": 101}
{"user_id": 2, "action": "view", "product_id": 102}
{"user_id": 1, "action": "click", "product_id": 101}
{"user_id": 2, "action": "view", "product_id": 102}
```

```
Select Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Windows\system32> cd C:\kafka_2.12-3.9.0
PS C:\kafka_2.12-3.9.0> bin/kafka-topics.sh --create --topic user_interactions --bootstrap-server localhost:9092 --parti
tions 1 --replication-factor 1
>>
PS C:\kafka_2.12-3.9.0> bin/kafka-topics.sh --list --bootstrap-server localhost:9092
>>
PS C:\kafka_2.12-3.9.0> bin/kafka-console-consumer.sh --topic user_interactions --from-beginning --bootstrap-server loca
lhost:9092
>>
PS C:\kafka_2.12-3.9.0>
```

Future Scope:

1. **Enhanced Real-Time Processing**:
   - Incorporate additional data streams (e.g., social media interactions, real-time reviews) into the recommendation engine to provide more accurate and context-aware suggestions.
2. **Integration with Deep Learning Models**:
   - Replace or complement the collaborative filtering approach with advanced deep learning methods such as neural collaborative filtering (NCF) or autoencoders to improve recommendation quality.
3. **Personalization Through Demographics**:
   - Integrate demographic data (e.g., age, gender, location) to enhance personalized recommendations, tailoring suggestions based on user-specific attributes.
4. **Scalability to Larger Datasets**:
   - Implement distributed storage systems like AWS S3 or Google BigQuery for handling massive datasets, allowing the system to scale to millions of users and products.
5. **Feedback Loop for Continuous Improvement**:
   - Introduce a feedback loop where user interactions with recommendations (e.g., clicks or purchases) are analyzed to continuously refine the model.
6. **Cross-Platform Integration**:
   - Expand the system to support multiple platforms, such as mobile apps, web browsers, and chatbots, ensuring consistent recommendations across user touchpoints.
7. **Hybrid Recommendations**:
   - Develop a hybrid model that combines collaborative filtering, content-based filtering, and knowledge-based systems to address cold-start issues and improve overall performance.
8. **Explainable Recommendations**:
   - Include explainability features that provide users with reasons for specific recommendations, enhancing transparency and user trust.
9. **Multi-Language Support**:
   - Add support for multi-language product names and descriptions to cater to a global user base, improving accessibility and inclusivity.
10. **Deployment to Cloud Platforms**:
    - Migrate the system to cloud services like AWS, GCP, or Azure for increased reliability, automatic scaling, and cost-effectiveness in production environments.

Limitations:

1. **Cold-Start Problem**:
   - The collaborative filtering approach used in this project struggles with the cold-start problem for new users and products, as it relies heavily on historical interaction data.
2. **Data Quality Dependency**:
   - The accuracy of recommendations is highly dependent on the quality and completeness of the input data. Missing or noisy data can degrade performance.
3. **Resource Constraints**:
   - Running PySpark jobs and managing Kafka streams on limited local hardware can lead to slower processing times and memory issues.
4. **Lack of User Feedback Integration**:
   - The system currently lacks a mechanism to capture explicit user feedback (e.g., likes or dislikes) to further refine recommendations.
5. **Scalability Challenges**:
   - While the system is designed for scalability, the current implementation may require significant adjustments to handle very large datasets or high user traffic in a distributed production environment.
6. **Limited Real-Time Capabilities**:
   - The real-time recommendation system processes user interactions in near real-time but might experience latency under heavy loads.
7. **Bias in Recommendations**:
   - The model may unintentionally reinforce popularity bias, favoring products that are already frequently interacted with, potentially limiting diversity in recommendations.
8. **Explainability Gap**:
   - The system does not provide insights into why certain recommendations are made, which could hinder user trust and engagement.
9. **Single-Source Data**:
   - The system primarily relies on interaction data from the Instacart dataset. Incorporating multi-source data (e.g., user reviews or external trends) could improve recommendation accuracy.
10. **Manual Parameter Tuning**:
    - The current ALS model requires manual tuning of hyperparameters (e.g., rank, regularization), which can be time-consuming and suboptimal.

References:

• Books: Pattern Recognition and Machine Learning by Christopher M. Bishop. This book provides a foundational understanding of machine learning algorithms, which will be critical for implementing the recommendation system. Data Science for Business by Foster Provost and Tom Fawcett. This text explains how to leverage data analytics in business, providing insights into the application of recommendation systems.

• Research Papers: Collaborative Filtering for Implicit Feedback Datasets by Yifan Hu, Yehuda Koren, and Chris Volinsky. This paper discusses collaborative filtering techniques that can be applied to implicit feedback data, relevant for user interactions in retail. A Survey of Recommendation Systems by Paul Resnick and Hal R. Varian. This survey provides an overview of various recommendation algorithms and their applications in e commerce.

• Online Resources: Apache Kafka Documentation: Apache Kafka – Official documentation for setting up and using Kafka for real-time data processing. Apache Spark Documentation: Apache Spark – Comprehensive guide to using Spark for batch and streaming data processing.

• Webinars and Tutorials: Coursera Course: Machine Learning Specialization by Andrew Ng. This course covers key machine learning concepts that will help in developing recommendation algorithms. YouTube Tutorials on Spark Streaming: Various channels provide step-by-step guides on implementing Spark Streaming with real-time data.

• Tools and Frameworks: Scikit-Learn Documentation: Scikit-Learn – Documentation for the machine learning library used for model implementation. TensorFlow Documentation: TensorFlow – Resource for utilizing deep learning techniques in the recommendation system.

• Industry Reports: The Future of Retail: Trends and Predictions by McKinsey & Company. This report discusses emerging trends in the retail sector, providing context for the implementation of recommendation systems. E-commerce Trends 2023 by Statista. A report highlighting current trends in online shopping, relevant for understanding the market landscape.

Conclusion

The Shopstream project successfully demonstrates a scalable and efficient framework for delivering personalized retail recommendations by integrating real-time and batch processing techniques. Leveraging cutting-edge big data tools like Apache Kafka and Apache Spark, alongside advanced machine learning models, the system addresses the challenges faced by traditional recommendation systems, including outdated suggestions, scalability limitations, and the cold start problem.

By combining real-time data streaming with historical insights, the proposed solution provides dynamic, contextually relevant recommendations that enhance user experience and drive business growth. The system's modular architecture ensures adaptability to evolving user behaviors and scalability to accommodate increasing data volumes, making it a robust solution for modern e-commerce platforms.

In addition to personalized recommendations, the project provides actionable business insights through analytics and monitoring tools, enabling better decision-making and operational efficiency. The successful implementation of Shopstream underscores the transformative potential of integrating big data technologies and machine learning to revolutionize user engagement and improve business performance in the competitive online retail landscape.

This framework lays the foundation for future innovations, such as integrating advanced deep learning models, exploring new data sources like social media, and further enhancing the real-time capabilities of recommendation systems, ensuring businesses stay ahead in the rapidly evolving world of e-commerce.