Introduction to Data Science — Assignment 2

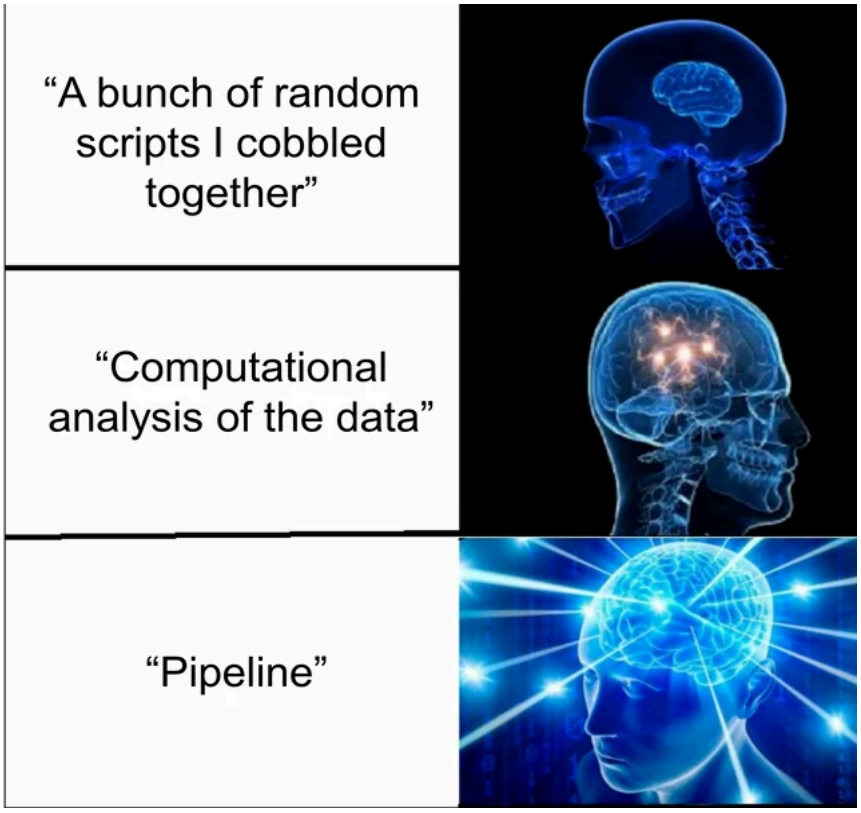
Instructors: **Dr. Bahrak, Dr. Yaghoobzadeh**

TA(s): Javad Razi, Mohammad Reza Nemati, Shahriar Attar, Mohammad Amin Yousefi

Deadline: Tuesday, Farvardin 28th, 11:59 PM

Introduction

This project gives you hands-on practice building a real-time payment data pipeline. **You must use Kafka for data intake and Spark for processing – these are the core tools we want you to learn.** For other parts (like databases or visualization), you can use different tools if you can explain why. For example: "We used Plotly instead of Matplotlib because it handles real-time updates better for our live transaction dashboard." Just give us a good technical reason. We've suggested specific tools to keep things manageable, but smart alternatives are okay if they make sense for your approach. Your final pipeline should process transactions, catch fraud, and deliver useful results.



What Is a Data Pipeline?

Requirements

PySpark

PySpark is the Python API for Apache Spark, a powerful open-source framework designed for large-scale data processing and analytics. It enables users to leverage Spark's distributed computing capabilities using Python, facilitating tasks such as data transformation, machine learning, and real-time stream processing. We recommend you install and use PySpark in a Linux environment since it's more convenient and user-friendly. Java installation is one of the mandatory things in installing Spark. Try the following command to verify the Java version:

```
java -version
```

In case you do not have Java installed on your system, then install Java before proceeding to the next step. We recommend you use OpenJDK for a more convenient installation.
Then, you need to install Spark; however, as of v2.2, installing PySpark will install Spark too. To install PySpark, you can use this link.

Apache Kafka

Apache Kafka is an open-source distributed event-streaming platform developed by the Apache Software Foundation.
It is designed to handle high-throughput, low-latency data feeds for real-time analytics and monitoring. For using Apache Kafka you also need Java so make sure of that first. To install Apache Kafka, you can use this link. Kafka requires ZooKeeper to manage and coordinate the cluster. To start ZooKeeper, use the following command:

```
bin/zookeeper-server-start.sh config/zookeeper.properties
```

In a new terminal, start the Kafka broker (server):

```
bin/kafka-server-start.sh config/server.properties
```

To interact with Kafka using Python, there are two primary client libraries available:

1. **kafka-python**: A pure Python client for Apache Kafka. It's designed to function much like the official Java client, with a Pythonic interface.
2. **confluent-kafka-python**: A Python wrapper around `librdkafka`, a high-performance C library. This client is known for its reliability and performance.

You can find more information about these two in this link. You are allowed to use which of them you like as you see fit. You can also follow the instructions here to set up Apache Kafka.
If you want to connect to the server via `kafka-python`, you can check this link. For `confluent-kafka-python`, check this link. You also need to use Kafdrop to view Kafka topics and monitor other things.

MongoDB

First, it's essential to understand the core differences between NoSQL and SQL Databases:

- **SQL Databases**: Utilize a structured, table-based format with predefined schemas, ensuring data integrity and consistency.
- **NoSQL Databases**: Offer flexible schema designs, accommodating various data types such as document, key-value, wide-column, and graph formats. This flexibility is advantageous for handling large volumes of unstructured or semi-structured data.

For more information, you can check out this link.

For this project, we utilize MongoDB. It is a highly scalable, distributed NoSQL database designed to handle large volumes of data across many servers without a single point of failure. It offers high availability and fault tolerance, making it suitable for applications requiring robust performance and reliability. For installation instructions, please refer to the official MongoDB documentation.
If you need a GUI, MongoDB Compass is a powerful GUI for MongoDB, designed to provide users with a visual environment to interact with their data. It allows for querying, aggregating, and analyzing data without requiring knowledge of MongoDB's query language. For installation and documentation, you can refer to this link. For connecting to MongoDB from Python, you can use PyMongo.

P.S. For better visualization of data frames in Jupyter Notebooks, since they are rendered to HTML, you can add this block of code at the beginning of your notebook:

```
from IPython.display import display, HTML
display(HTML('<style>pre { white-space: pre !important; }</style>'))
```

Main Task

The Scenario

You are 'Darooghe', a payment service provider offering payment services via Online Payment Gateway, POS (Point of Sale) Device, Mobile Application, and NFC (Contactless) Payment. Your pipeline will process transaction events, calculate commissions, detect fraudulent activities, and provide business intelligence insights.

Environment Setup (5 pts)

We provide a Transaction Generator, A service generating synthetic payment transaction events. You can change parts of it as you see fit there is no restriction.

Configurable Parameters:

Parameter	Description	Default	Valid Range
EVENT_RATE	Average events per minute	100	10-1000
PEAK_FACTOR	Multiplier for peak hours	2.5	1.0-5.0
FRAUD_RATE	Percentage of fraudulent transactions	0.02	0.0-0.1
DECLINED_RATE	Percentage of declined transactions	0.05	0.0-0.2
MERCHANT_COUNT	Number of unique merchants	50	10-500
CUSTOMER_COUNT	Number of unique customers	1000	100-10000

The transaction generator produces events following a Non-homogeneous Poisson Process with daily and weekly patterns.

Data Description

The payment transaction data contains the following fields:

Field	Description
transaction_id	Unique identifier for each transaction (UUID format)
timestamp	Time when a transaction occurred (ISO format)
customer_id	Customer who made the payment
merchant_id	Merchant who received the payment
merchant_category	One of ['retail', 'food_service', 'entertainment', 'transportation', 'government']
payment_method	One of ['online', 'pos', 'mobile', 'nfc']
amount	Transaction amount in IRR
location	Geolocation data (latitude, longitude)
device_info	Information about the device used (for online/mobile transactions)
status	Transaction status ['approved', 'declined', 'pending']
commission_type	One of ['flat', 'progressive', 'tiered']
commission_amount	Amount of commission collected by Darooghe (in IRR)
vat_amount	Value-added tax amount
total_amount	Total amount paid by customer (amount + vat_amount + commission_amount)
customer_type	One of ['individual', 'CIP[1]', 'business']
risk_level	Risk level assigned by Darooghe's risk engine (1-5)
failure_reason	Reason for failed transactions ['cancelled, 'insufficient_funds', 'system_error', 'fraud_prevented', null]

[1] CIP: Commercially Important Person (Fancy name to refer to the customers keeping Darooghe in the business)

Example Transaction:


```
{
  "transaction_id": "89a7e4c1-fb88-4bd9-8c47-0ad28aef77eb",
  "timestamp": "2025-03-23T14:35:22.145Z",
  "customer_id": "cust_29847",
  "merchant_id": "merch_1578",
  "merchant_category": "food_service",
  "payment_method": "mobile",
  "amount": 850000,
  "location": {"lat": 35.7219, "lng": 51.3347},
  "device_info": {"os": "Android", "app_version": "2.4.1",
"device_model": "Samsung Galaxy S25"},
  "status": "approved",
  "commission_type": "progressive",
  "commission_amount": 17000,
  "vat_amount": 127500,
  "total_amount": 977500,
  "customer_type": "individual",
  "risk_level": 3,
  "failure_reason": null
}
```


Data Ingestion Layer (20 pts)

Kafka Consumer Implementation

I. Implement a Kafka consumer to connect to the Kafka server and read from the `darooghe.transactions` topic.

II. Implement proper deserialization and data validation. Validate each transaction against these basic business rules below. Write all transactions that have been determined as invalid, into the topic `darooghe.error_logs`. Include the error code, with relevant data, and, obviously, the ID of the transaction in the messages you write into `darooghe.error_logs` topic.

Rule	Validation Logic	Error Code	Insight Potential
Amount Consistency	`total_amount == amount + vat_amount + commission_amount`	`ERR_AMOUNT`	Detects misconfigured merchant pricing engines
Time Warping	`timestamp` not in future AND not more than a day older than Kafka ingestion time	`ERR_TIME`	Surfaces clock drift in POS devices (completely possible!)
Device Mismatch	`if (payment_method=="mobile") => device_info.os ∈ ["iOS","Android"]`	`ERR_DEVICE`	Reveals potentially spoofed device headers

III. Display one message.

Schema Management

I. Create appropriate data structures for transaction events.
II. Implement type conversion and validation.

Batch Processing Layer (25 pts)

In this part of the project, the focus is on processing and analyzing **existing data** that has already been generated. This part should only be implemented using PySpark. You should store all the insights and analysis done in this part and then later visualize it.

Commission Analysis Batch Job

I. Develop batch processing jobs that query aggregated data to generate reports on commission efficiency by merchant category. For example, calculate total commissions, average commissions per transaction, and commission-to-transaction ratios.
II. Identify optimal commission structures, use historical commission data to analyze trends, and simulate different commission models. The job should identify optimal structures by comparing historical performance and profitability across various merchant categories.

Transaction Pattern Analysis

I. Discover and report on temporal patterns in transaction data
II. Identify peak transaction times based on historical data.
III. Segment customers based on spending frequency and patterns.
IV. Compare transaction behavior across different merchant categories.
V. Identify when most transactions happen (e.g., morning, evening).
VI. Notice if people are spending more or less over time.

Data Storage Implementation

I. Load the data into MongoDB. Use a proper partitioning strategy—by date, merchant, or another logical key—to ensure efficient querying and scalability.
II. Implement data retention policy (e.g. keep the last 24 hours of detailed data).
III. Using MongoDB queries, create aggregated historical datasets for longer-term analysis. These datasets should summarize key insights, such as:
    - **Summarized transaction data**: Group transactions by merchant, customer segment, or time period (e.g., daily, weekly, monthly).
    - **Commission reports**: Aggregate total commissions earned per merchant category over time.

Real-Time Processing Layer (25 pts)

In this part of the project, the focus is on processing **incoming data** in real-time as it arrives, rather than working with pre-existing or historical data in the dataset. The goal is to analyze the stream of new transactions from Kafka continuously.

Spark Streaming Application

I. Implement a Spark Streaming application that connects to Kafka Consumer.
II. Process data in small time intervals (micro-batches), using time-based windows to aggregate and analyze data over specific periods (e.g., 1 minute), with the window sliding at regular intervals (e.g., every 20 seconds), to ensure timely and efficient data processing in real time. Use the processing to gain some informative insights. Write detected insights to an (or multiple) arbitrary topic (topics).
III. Implement a checkpoint mechanism for fault tolerance.

Fraud Detection System

I. Implement three fraud detection rules:

A. Velocity check: More than 5 transactions from the same customer in 2 minutes
B. Geographical impossibility: Transactions from locations >50 km apart within 5 minutes.
C. Amount anomaly: Transaction amount >1000% of customer's average (find customer's average from pre-existing data).

II. Write detected fraud events to the `darooghe.fraud_alerts` topic.

Real-Time Commission Analytics

I. Calculate and write to an (or multiple) arbitrary topic (topics), real-time metrics for commissions:
   A. Total commission by type per minute.
   B. Commission ratio (commission/transaction amount) by merchant category.
   C. Highest commission-generating merchants in 5-minute windows.

Visualization (25 pts)

You can use any visualization libraries in Python to extract these insights (and more!) from the data. Visualizations should clearly communicate trends and anomalies, with concise labeling and legends.

Key Visualizations:

I. **Transactions Volume:** Display time-series charts that show real-time and historical transaction volumes, helping to identify trends and seasonal peaks.
II. **Merchant Analysis:** Show the top 5 merchants based on the number of transactions.
III. **User Activity:** Display metrics such as the number of transactions per user, frequency of activity, and growth trends, offering insights into user engagement.

Bonus Tasks (20 pts total)

Advanced Fraud Patterns (5 pts)

Temporal Analysis

I. Identify transactions occurring outside the merchant's local business hours. (3 pts)
II. Identify which part of the day, each merchant category is most active. (1 pts)
III. Identify merchants whose business faces sudden transaction spikes. (1 pts)

Commission Audit System (5 pts)

Dynamic Pricing Simulator

I. Create PySpark UDF to recommend optimal commission type per transaction. (3 pts)
II. Validate against historical profitability data. (2 pts)

Pipeline Optimization (10 pts)

Resource Monitoring

I. Implement Prometheus metrics for Kafka consumer lag, Spark executor CPU/MEM, and JVM garbage collection time.

Notes

- Upload your work as a zip file in this format on the website: DS_CA2_[Std number].zip. If the project is done in a group, include all of the group members' student numbers in the name.
- Only one member must upload the work if the project is done in a group.
- We will run your code during the project delivery, so make sure your results are reproducible.[2]
- There are no explicit theoretical questions in the assignment, but please note that you will be asked questions about concepts and tools you used in this project. You are expected to understand the key functionalities of the main tools used in this assignment. (Apache Kafka, Spark, and core concepts behind building a real-time data analytics pipeline) As an example, you should be comfortable with explaining what was the point of using Kafka in this project, how things would differ if we used alternative solutions (e.g. storing events in a database, reading from it), and other fundamental concepts you are expected to gain a decent level of grasp on them.

[2] Your code will be executed as-is in a fresh environment (e.g., a clean Docker container or VM) to verify that it runs without errors, and your outputs (e.g. visualizations and dataframes) are same (or effectively the same) as the result you've uploaded. Use practices like randomizing with seeds, to make the result more reproducible, but we also understand that due to the design of the project, reproducing exact same output on every section is not possible.