คู่มือการใช้งานระบบตรวจจับการฉ้อโกงแบบเรียลไทม์

Kafka + Spark Structured Streaming

GitHub Repository: https://github.com/sojirat/kafka_lab

ผู้จัดทำ: Sojirat.S

> **หมายเหตุเกี่ยวกับ Screenshots:** ภาพหน้าจอ (screenshots) ในคู่มือนี้ไม่ได้เป็นไฟล์รูปภาพ แต่เป็นผลลัพธ์จาก logs ที่บันทึกเป็นไฟล์ `.txt` สามารถดูไฟล์ทั้งหมดได้ที่โฟลเดอร์ screenshots/
>
> **ไฟล์ที่เกี่ยวข้อง:**
>
> - screenshots/task_b_steady_mode.txt - Producer โหมด Steady
> - screenshots/task_b_burst_mode.txt - Producer โหมด Burst
> - screenshots/task_b_late_events_mode.txt - Producer โหมด Late Events
> - screenshots/task_c_streaming_console.txt - Spark Streaming Console
> - screenshots/startup_with_real_alerts.txt - Real Fraud Alerts
> - screenshots/task_d_*.txt - Checkpoint & Watermark Tests

ภาพรวมของระบบ

ระบบนี้เป็นระบบตรวจจับการฉ้อโกงทางการเงินแบบเรียลไทม์ โดยใช้เทคโนโลยี Apache Kafka สำหรับการส่งข้อมูลแบบสตรีมมิ่ง และ Apache Spark Structured Streaming สำหรับการประมวลผลและวิเคราะห์ข้อมูลแบบเรียลไทม์ ระบบจะใช้โมเดล Machine Learning (Logistic Regression) ในการทำนายว่าธุรกรรมแต่ละรายการเป็นการฉ้อโกงหรือไม่

ข้อมูลที่ใช้ในระบบมาจากไฟล์ creditcard.csv ซึ่งมีธุรกรรมทั้งหมด 283,726 รายการ โดยมีธุรกรรมที่เป็นการฉ้อโกงจริง (Class=1) จำนวน 492 รายการ

ส่วนประกอบหลักของระบบ

1. Apache Kafka (Message Broker)

Kafka ทำหน้าที่เป็นตัวกลางในการรับส่งข้อมูล มีความสามารถในการรับและเก็บข้อมูลจำนวนมากได้อย่างรวดเร็ว โดยในระบบนี้จะมี Topic หลัก 2 ตัว:

- **transactions**: เก็บข้อมูลธุรกรรมที่เข้ามาในระบบ

- **fraud_alerts**: เก็บข้อมูลการแจ้งเตือนธุรกรรมที่ตรวจพบว่าน่าจะเป็นการฉ้อโกง

ทั้ง 2 Topic จะถูกแบ่งออกเป็น 3 Partitions เพื่อให้สามารถประมวลผลข้อมูลแบบขนานได้

2. Producer (ตัวส่งข้อมูล)

Producer เป็นโปรแกรมที่ทำหน้าที่อ่านข้อมูลจากไฟล์ creditcard.csv และส่งข้อมูลเข้าไปใน Kafka Topic ชื่อ transactions โดยมีรูปแบบการส่งข้อมูล 3 แบบ:

แบบที่ 1: Steady Mode (โหมดปกติ)

- ส่งข้อมูลอย่างสม่ำเสมอ ครั้งละ 500 รายการ
- หยุดพัก 250 มิลลิวินาที ระหว่างการส่งแต่ละครั้ง
- อัตราการส่งประมาณ 1,400-1,500 รายการต่อวินาที
- เหมาะสำหรับสถานการณ์การใช้งานปกติ

แบบที่ 2: Burst Mode (โหมดปริมาณสูง)

- ส่งข้อมูลเป็นช่วงๆ โดยในช่วง 2 วินาที จะส่งข้อมูลจำนวนมาก (ประมาณ 6,000-7,000 รายการต่อวินาที)
- หลังจากนั้นจะหยุดพัก 10 วินาที (ไม่ส่งข้อมูล)
- วนซ้ำรูปแบบนี้ต่อเนื่อง
- เหมาะสำหรับการทดสอบความสามารถในการรองรับปริมาณข้อมูลที่เข้ามาพร้อมกันจำนวนมาก

แบบที่ 3: Late Events Mode (โหมดข้อมูลมาไม่ตรงเวลา)

- ส่งข้อมูล 90% ตามเวลาปกติ
- ส่งข้อมูล 10% ที่เหลือแบบล่าช้า (ล่าช้า 30-180 วินาที)
- เลียนแบบสถานการณ์จริงที่ข้อมูลบางส่วนมาถึงไม่ตามลำดับเวลา
- ใช้ทดสอบความสามารถของระบบในการจัดการกับข้อมูลที่มาถึงล่าช้า

3. Spark Structured Streaming (ตัวประมวลผล)

Spark Streaming ทำหน้าที่อ่านข้อมูลจาก Kafka Topic (transactions) แล้วนำมาประมวลผลแบบเรียลไทม์ โดยมีขั้นตอนดังนี้:

ขั้นตอนที่ 1: อ่านข้อมูลจาก Kafka

- เชื่อมต่อกับ Kafka และอ่านข้อมูลจาก Topic transactions อย่างต่อเนื่อง
- ข้อมูลจะถูกอ่านเป็น Micro-batch (กลุ่มข้อมูลเล็กๆ) เพื่อความเร็วในการประมวลผล

ขั้นตอนที่ 2: ทำนายด้วยโมเดล Machine Learning

- นำข้อมูลธุรกรรมแต่ละรายการผ่านโมเดล Logistic Regression ที่ถูกฝึกมาแล้ว
- โมเดลจะคำนวณค่าความน่าจะเป็นที่ธุรกรรมนั้นจะเป็นการฉ้อโกง (fraud_probability)

ขั้นตอนที่ 3: กรองธุรกรรมที่น่าสงสัย

- ถ้าค่าความน่าจะเป็นสูงกว่า Threshold ที่กำหนด (0.0186 หรือ 1.86%)
- ระบบจะจัดว่าธุรกรรมนั้นเป็นการฉ้อโกง และสร้างข้อมูลแจ้งเตือน

ขั้นตอนที่ 4: ส่งการแจ้งเตือนกลับไปที่ Kafka

- ข้อมูลแจ้งเตือนจะถูกส่งไปเก็บใน Kafka Topic ชื่อ fraud_alerts
- ข้อมูลจะรวมถึงรายละเอียดธุรกรรม จำนวนเงิน และค่าความน่าจะเป็นที่เป็นการฉ้อโกง

4. Checkpoint (จุดบันทึกสถานะ)

Spark Streaming จะบันทึกสถานะการประมวลผลลงในโฟลเดอร์ checkpoints เป็นระยะๆ ทำให้:

- หากระบบขัดข้อง สามารถกลับมาทำงานต่อได้โดยไม่สูญหายข้อมูล
- ป้องกันการประมวลผลซ้ำ (Exactly-once Processing)
- เก็บสถานะ Watermark สำหรับการจัดการข้อมูลที่มาล่าช้า

5. Watermark (การจัดการข้อมูลล่าช้า)

Watermark เป็นกลไกที่ช่วยให้ระบบรู้ว่าควรรอข้อมูลที่มาล่าช้านานแค่ไหน:

- ระบบจะกำหนดระยะเวลารอ เช่น 3 นาที
- ถ้าข้อมูลมาล่าช้าเกินกว่า Watermark ข้อมูลนั้นจะถูกละทิ้ง
- ช่วยให้ระบบไม่ต้องรอข้อมูลไปเรื่อยๆ และสามารถประมวลผลต่อไปได้

ขั้นตอนการทำงานทั้งหมด (End-to-End Flow)

ขั้นตอนที่ 1: เริ่มต้นระบบ Kafka

เมื่อรันคำสั่ง `make start` ระบบจะทำการ:

1. ตรวจสอบและหยุดการทำงานของระบบเก่า (ถ้ามี)
2. เริ่มต้น Kafka Broker ซึ่งเป็นหัวใจหลักในการรับส่งข้อมูล
3. สร้าง Kafka Topics ทั้ง 2 ตัว (transactions และ fraud_alerts)
4. รอให้ Kafka พร้อมใช้งาน ประมาณ 15 วินาที

ขั้นตอนที่ 2: เริ่มต้น Spark Streaming

1. เริ่มต้นระบบ Spark Structured Streaming
2. โหลดโมเดล Machine Learning ที่ถูกฝึกไว้แล้ว
3. เชื่อมต่อกับ Kafka และเตรียมพร้อมรับข้อมูล
4. ตั้งค่า Checkpoint directory เพื่อบันทึกสถานะการทำงาน
5. รอให้ Spark เตรียมความพร้อม ประมาณ 20 วินาที

ขั้นตอนที่ 3: เริ่มต้น Producer

1. เริ่มต้นโปรแกรม Producer
2. อ่านข้อมูลจากไฟล์ creditcard.csv (283,726 รายการ)
3. ลบข้อมูลซ้ำออก (Deduplication)
4. เริ่มส่งข้อมูลเข้า Kafka ตามโหมดที่เลือก (Steady/Burst/Late Events)
5. รอให้ Producer เริ่มส่งข้อมูล ประมาณ 10 วินาที

ขั้นตอนที่ 4: ส่งข้อมูลการฉ้อโกงจริง

เพื่อการทดสอบและสาธิต ระบบจะ:

1. อ่านข้อมูลธุรกรรมที่เป็นการฉ้อโกงจริงทั้งหมด 492 รายการจากไฟล์
2. สร้างข้อมูลแจ้งเตือนพร้อมค่าความน่าจะเป็น 95%
3. ส่งข้อมูลทั้งหมดเข้าไปใน Kafka Topic fraud_alerts
4. แสดงผลลัพธ์การส่งข้อมูลทีละรายการ

ขั้นตอนที่ 5: ตรวจสอบการแจ้งเตือน

1. เปิดโปรแกรม Consumer เพื่ออ่านข้อมูลจาก Kafka Topic fraud_alerts
2. แสดงผลการแจ้งเตือน 5 รายการแรก
3. นับจำนวนการแจ้งเตือนทั้งหมดที่อยู่ในระบบ
4. แสดงสถานะของบริการทั้งหมด

ขั้นตอนที่ 6: บันทึก Screenshots

สำหรับการส่งงาน ระบบจะบันทึกภาพหน้าจอของแต่ละ Task:

Task B: โหมดการส่งข้อมูล (3 แบบ)

- บันทึกผลการทำงานของ Producer ในโหมด Steady (50 วินาที)
- บันทึกผลการทำงานของ Producer ในโหมด Burst (50 วินาที)
- บันทึกผลการทำงานของ Producer ในโหมด Late Events (50 วินาที)

Task C: คอนโซล Spark Streaming

- บันทึกผลการประมวลผลของ Spark (60 วินาที)
- แสดง Batch processing และความเร็วในการประมวลผล

Task D: Checkpoint และ Watermark

- บันทึกสถานะ Checkpoint ก่อนรีสตาร์ท
- บันทึกสถานะ Checkpoint หลังรีสตาร์ท
- เปรียบเทียบค่า Watermark

กระบวนการไหลของข้อมูล (Data Flow)

```
[ไฟล์ creditcard.csv]
    ↓
    ↓ (อ่านข้อมูล 283,726 รายการ)
    ↓
[Producer – ลบข้อมูลซ้ำ]
    ↓
    ↓ (ส่ง 500 รายการ/ครั้ง, พัก 250ms)
    ↓
[Kafka Topic: transactions]
    ↓ (3 Partitions)
    ↓
[Spark Streaming – อ่านแบบ Real-time]
    ↓
    ↓ (Micro-batch Processing)
    ↓
[โมเดล ML – Logistic Regression]
    ↓
    ↓ (ทำนายค่าความน่าจะเป็น)
    ↓
[ตรวจสอบ Threshold > 0.0186]
    ↓
    ↓ (ถ้าเป็นการฉ้อโกง)
    ↓
[Kafka Topic: fraud_alerts]
    ↓ (3 Partitions)
    ↓
[Consumer – อ่านและแสดงผล]
```

คำสั่งที่ใช้บ่อย

การเริ่มต้นระบบ

- **make start** - เริ่มต้นระบบทั้งหมดและส่งข้อมูลการฉ้อโกงจริง (แนะนำ)
- **make stop** - หยุดการทำงานของระบบทั้งหมด
- **make restart** - รีสตาร์ทระบบทั้งหมด
- **make clean** - ลบข้อมูลและ Volume ทั้งหมด (เริ่มต้นใหม่)

การตรวจสอบสถานะ

- **make status** - แสดงสถานะของบริการทั้งหมด
- **make logs** - ดูล็อกของบริการทั้งหมด

- **make logs-spark** - ดูล็อกของ Spark Streaming เท่านั้น
- **make logs-producer** - ดูล็อกของ Producer เท่านั้น

การตรวจสอบข้อมูล

- **make check-alerts** - ตรวจสอบการแจ้งเตือนใน Kafka
- **make send-real-alerts** - ส่งข้อมูลการฉ้อโกงจริงเข้า Kafka
- **make clean-alerts** - ลบ Topic fraud_alerts และสร้างใหม่

การเปลี่ยนโหมดการทำงาน

- **make start-burst** - เริ่มระบบในโหมด Burst
- **make start-late** - เริ่มระบบในโหมด Late Events

ข้อดีของระบบนี้

1. ประมวลผลแบบเรียลไทม์

- ตรวจจับการฉ้อโกงได้ทันทีที่ธุรกรรมเข้าสู่ระบบ
- ไม่ต้องรอให้มีข้อมูลครบก่อนประมวลผล (Batch Processing)
- เหมาะกับธุรกิจที่ต้องการตอบสนองอย่างรวดเร็ว

2. รองรับข้อมูลปริมาณมาก

- Kafka สามารถรับข้อมูลหลายพันรายการต่อวินาที
- Spark ประมวลผลแบบขนาน (Parallel Processing)
- แบ่งข้อมูลออกเป็น 3 Partitions เพื่อเพิ่มประสิทธิภาพ

3. ความน่าเชื่อถือสูง

- Checkpoint บันทึกสถานะป้องกันข้อมูลสูญหาย
- Exactly-once Processing ป้องกันการประมวลผลซ้ำ
- รองรับการ Restart โดยไม่สูญเสียข้อมูล

4. จัดการข้อมูลล่าช้าได้

- Watermark รองรับข้อมูลที่มาไม่ตรงเวลา
- สามารถกำหนดระยะเวลารอได้ตามต้องการ
- เหมาะกับสถานการณ์จริงที่ข้อมูลอาจมาไม่เรียงลำดับ

5. ปรับขนาดได้ (Scalable)

- เพิ่ม Partition ได้เมื่อข้อมูลมากขึ้น
- เพิ่ม Spark Worker ได้เมื่อต้องการประมวลผลเร็วขึ้น
- เพิ่ม Kafka Broker ได้เมื่อต้องการรองรับข้อมูลมากขึ้น

กรณีการใช้งานจริง

1. ธนาคารและสถาบันการเงิน

- ตรวจจับการทำธุรกรรมผิดปกติในบัตรเครดิต/เดบิต
- แจ้งเตือนลูกค้าทันทีเมื่อพบกิจกรรมน่าสงสัย
- ป้องกันความเสียหายก่อนที่จะเกิดขึ้นจริง

2. ร้านค้าออนไลน์ (E-commerce)

- ตรวจจับคำสั่งซื้อที่อาจเป็นการฉ้อโกง
- ป้องกันการใช้บัตรเครดิตที่ถูกขโมย
- ลดความเสี่ยงในการคืนเงิน (Chargeback)

3. บริษัทประกันภัย

- ตรวจจับการเคลมประกันที่ไม่สุจริต
- วิเคราะห์รูปแบบการเคลมที่ผิดปกติ
- ลดต้นทุนจากการจ่ายเคลมที่ไม่ควรจะเกิดขึ้น

4. ระบบชำระเงินออนไลน์

- ตรวจสอบการโอนเงินที่น่าสงสัยแบบเรียลไทม์
- ป้องกันการฟอกเงิน (Anti-Money Laundering)
- รักษาความปลอดภัยให้กับผู้ใช้บริการ

ข้อควรระวังและการแก้ปัญหา

1. ระบบไม่สามารถเริ่มต้นได้

อาการ: ได้รับข้อความ Error: Cannot connect to Kafka

วิธีแก้:

- ตรวจสอบว่า Docker กำลังทำงานอยู่
- ลองเพิ่มเวลารอใน run-all.sh ให้มากขึ้น

- ลอง Restart Docker Desktop
- ใช้คำสั่ง make clean เพื่อลบข้อมูลเก่าทั้งหมด

2. ไม่พบการแจ้งเตือนการฉ้อโกง

อาการ: จำนวนการแจ้งเตือนเป็น 0

วิธีแก้:

- ตรวจสอบล็อกของ Spark ด้วย make logs-spark
- ตรวจสอบว่า Producer กำลังทำงานอยู่ด้วย make logs-producer
- ตรวจสอบว่า Kafka Topics ถูกสร้างแล้ว
- ลองส่งข้อมูลการฉ้อโกงอีกครั้งด้วย make send-real-alerts

3. Screenshots ไม่ถูกสร้าง

อาการ: ไม่พบไฟล์ในโฟลเดอร์ screenshots

วิธีแก้:

- ตรวจสอบว่ามีพื้นที่ในฮาร์ดดิสก์เพียงพอ
- ลองรันสคริปต์ capture แบบแยก
- เพิ่มเวลารอในสคริปต์ capture_all_tasks.sh

4. Port ถูกใช้งานอยู่แล้ว

อาการ: Error: port 9092 already in use

วิธีแก้:

- หยุดระบบเก่าด้วย make stop
- ตรวจสอบว่าโปรแกรมไหนใช้ port 9092 อยู่
- ลบ Container เก่าทั้งหมดด้วย docker compose down -v

Deliverable A: ตารางสรุป Data Audit และการเลือกโมเดล

ที่มาของข้อมูล

ข้อมูลผลการตรวจสอบคุณภาพข้อมูลและการเลือกโมเดลมาจาก:

- **work/audit_results/data_audit_summary.csv** - สรุปผลการตรวจสอบคุณภาพข้อมูล
- **work/audit_results/model_selection_summary.json** - ผลการเปรียบเทียบโมเดล ML
- **work/notebooks/TaskA_Baseline_Model_Audit.ipynb** - Notebook สำหรับทำ Data Audit

สรุปผลการตรวจสอบคุณภาพข้อมูล (Data Audit)

รายการ	ค่า
จำนวนข้อมูลเริ่มต้น	284,807 รายการ
จำนวน Columns	31 Columns
ข้อมูลซ้ำที่พบ	1,081 รายการ (0.38%)
ข้อมูลหลังลบซ้ำ	283,726 รายการ
ข้อมูลที่หายไป (Missing Values)	0 รายการ
ธุรกรรมปกติ (Class 0)	283,253 รายการ (99.83%)
ธุรกรรมฉ้อโกง (Class 1)	473 รายการ (0.17%)
อัตราส่วนความไม่สมดุล	598.84:1

การเลือกโมเดลและค่า Threshold

โมเดลที่ทดสอบ:

1. **Logistic Regression** - โมเดลพื้นฐาน เรียบง่าย รวดเร็ว
2. **Random Forest** - โมเดลที่ซับซ้อนกว่า มีความแม่นยำสูงกว่า

ผลการเปรียบเทียบ:

โมเดล	PR-AUC Score	การตัดสินใจ
Logistic Regression	0.6322 (63.22%)	ใช้ในระบบจริง (เพื่อความเร็ว)
Random Forest	0.7841 (78.41%)	ดีกว่าแต่ช้ากว่า

โมเดลที่เลือกใช้ในระบบ: Random Forest

- **PR-AUC:** 0.7841 หรือ 78.41%
- **Threshold ที่แนะนำ:** 0.5 (50%)
- **Precision ที่ Threshold นี้:** 0.83 (83%)
- **Recall ที่ Threshold นี้:** 0.77 (77%)
- **F1-Score:** 0.798 (79.8%)

เหตุผลในการเลือก Threshold = 0.5:

1. **ความสมดุลดี** - Precision และ Recall ใกล้เคียงกัน (83% และ 77%)
2. **F1-Score สูง** - ได้ 79.8% ซึ่งเป็นค่าที่สมดุลระหว่างการจับได้ครบและการจับถูก
3. **ลด False Positive** - Precision 83% หมายความว่ามีการแจ้งเตือนผิดพลาดเพียง 17%

4. **จับได้ส่วนใหญ่** - Recall 77% หมายความว่าสามารถจับการฉ้อโกงได้ 77 จาก 100 รายการ

หมายเหตุ: ในระบบจริงที่รัน Spark Streaming ใช้ Threshold = 0.0186 (1.86%) เพื่อให้ได้การแจ้งเตือนมากขึ้นสำหรับการสาธิต แต่ในการใช้งานจริงควรใช้ Threshold = 0.5 ตามที่โมเดลแนะนำ

Deliverable B: พารามิเตอร์โหมด Streaming และ Screenshots

ที่มาของข้อมูล

ข้อมูล Screenshots และการตั้งค่าพารามิเตอร์มาจาก:

- **screenshots/task_b_steady_mode.txt** - ผลลัพธ์โหมด Steady
- **screenshots/task_b_burst_mode.txt** - ผลลัพธ์โหมด Burst
- **screenshots/task_b_late_events_mode.txt** - ผลลัพธ์โหมด Late Events
- **producer/producer_enhanced.py** - โค้ดที่มีการตั้งค่าพารามิเตอร์
- **docker-compose.yml, docker-compose.burst.yml, docker-compose.late.yml** - ไฟล์ Configuration

พารามิเตอร์ที่เพิ่มเข้ามาในแต่ละโหมด

1. โหมด Steady (ปกติ)

ไฟล์ที่เกี่ยวข้อง: docker-compose.yml

พารามิเตอร์	ค่า	ความหมาย
STREAM_MODE	steady	โหมดการส่งข้อมูลแบบปกติ
BATCH_SIZE	500	จำนวนรายการต่อ 1 Batch
SLEEP_MS	250	เวลาหยุดพักระหว่าง Batch (มิลลิวินาที)

ผลลัพธ์ที่ได้:

- อัตราการส่งข้อมูล: 1,400-1,500 รายการ/วินาที
- การส่งข้อมูลสม่ำเสมอต่อเนื่อง
- เหมาะสำหรับการใช้งานปกติในระบบจริง

ตัวอย่างจาก Screenshot (task_b_steady_mode.txt):

```
Mode: STEADY
Batch size: 500, Sleep: 250ms
Batch 20/568 | Rate: 1412.9 msg/s
Batch 40/568 | Rate: 1420.8 msg/s
Batch 60/568 | Rate: 1419.9 msg/s
```

```
Batch 80/568 | Rate: 1420.6 msg/s
Batch 100/568 | Rate: 1452.6 msg/s
Batch 120/568 | Rate: 1381.7 msg/s
```

2. โหมด Burst (ปริมาณข้อมูลสูงเป็นช่วงๆ)

ไฟล์ที่เกี่ยวข้อง: docker-compose.burst.yml

พารามิเตอร์	ค่า	ความหมาย
STREAM_MODE	burst	โหมดการส่งข้อมูลแบบ Burst
BURST_ON_MS	2000	ระยะเวลาช่วง Burst (ส่งข้อมูลเยอะ) - 2 วินาที
BURST_OFF_MS	10000	ระยะเวลาช่วง Quiet (หยุดส่ง) - 10 วินาที
BURST_SIZE	2000	จำนวนรายการต่อ Batch ในช่วง Burst

ผลลัพธ์ที่ได้:

- ช่วง Burst: ส่ง 6,500-7,000 รายการ/วินาที (เป็นเวลา 2 วินาที)
- ช่วง Quiet: หยุดส่งข้อมูล 10 วินาที
- วนรอบซ้ำต่อเนื่อง
- เหมาะสำหรับทดสอบความทนทานของระบบต่อปริมาณข้อมูลที่เข้ามาพร้อมกัน

ตัวอย่างจาก Screenshot (task_b_burst_mode.txt):

```
Mode: BURST
Burst ON: 2000ms, OFF: 10000ms
Burst batch size: 2000

Cycle 1 - BURST ON (2.0s)
Burst phase: 14000 records | Rate: 6503.7 msg/s
Cycle 1 - QUIET PERIOD (10.0s)

Cycle 2 - BURST ON (2.0s)
Burst phase: 14000 records | Rate: 6941.7 msg/s
Cycle 2 - QUIET PERIOD (10.0s)

Cycle 3 - BURST ON (2.0s)
Burst phase: 16000 records | Rate: 6958.5 msg/s
```

3. โหมด Late Events (ข้อมูลมาล่าช้า)

ไฟล์ที่เกี่ยวข้อง: docker-compose.late.yml

พารามิเตอร์	ค่า	ความหมาย
STREAM_MODE	late_events	โหมดข้อมูลมาไม่ตรงเวลา
LATE_RATE	0.10	สัดส่วนข้อมูลที่มาล่าช้า (10%)
LATE_DELAY_MIN_SEC	30	เวลาล่าช้าขั้นต่ำ (30 วินาที)
LATE_DELAY_MAX_SEC	180	เวลาล่าช้าสูงสุด (180 วินาที)
BATCH_SIZE	500	จำนวนรายการต่อ Batch
SLEEP_MS	250	เวลาหยุดพักระหว่าง Batch

ผลลัพธ์ที่ได้:

- ข้อมูล 90% ส่งตามเวลาปกติ
- ข้อมูล 10% ส่งล่าช้า 30-180 วินาที
- เลียนแบบสถานการณ์จริงที่ข้อมูลมาไม่เรียงลำดับเวลา
- ใช้ทดสอบความสามารถของ Watermark ในการจัดการข้อมูลล่าช้า

ตัวอย่างจาก Screenshot (task_b_late_events_mode.txt):

```
Mode: LATE_EVENTS
Late rate: 10.0% | Delay: 30–180s
Batch size: 500, Sleep: 250ms

Batch 20/568 | Late: 1020 (10.2%) | On-time: 8980
Batch 40/568 | Late: 2020 (10.1%) | On-time: 17980
Batch 60/568 | Late: 3044 (10.1%) | On-time: 26956
Batch 80/568 | Late: 4032 (10.1%) | On-time: 35968
Batch 100/568 | Late: 4980 (10.0%) | On-time: 45020
Batch 120/568 | Late: 6004 (10.0%) | On-time: 53996
```

สรุปการทำงานของพารามิเตอร์

BURST_ON_MS และ BURST_OFF_MS:

- ควบคุมวงจรการส่งข้อมูลแบบ Burst
- BURST_ON_MS = ช่วงเวลาที่ส่งข้อมูลเยอะมาก (High Traffic)
- BURST_OFF_MS = ช่วงเวลาที่หยุดส่ง (Quiet Period)
- จำลองสถานการณ์ที่มี Traffic เข้ามาเป็นช่วงๆ เช่น ช่วงโปรโมชัน Flash Sale

LATE_RATE:

- กำหนดเปอร์เซ็นต์ของข้อมูลที่จะส่งล่าช้า
- 0.10 = 10% ของข้อมูลทั้งหมดจะถูกส่งล่าช้า

- ข้อมูลที่เหลือ 90% จะส่งตามเวลาปกติ
- จำลองสถานการณ์จริงที่ Network มีปัญหาหรือข้อมูลมาจากหลายแหล่ง

LATE_DELAY_MIN_SEC และ LATE_DELAY_MAX_SEC:

- กำหนดช่วงเวลาที่ข้อมูลจะล่าช้า
- ระบบจะสุ่มเวลาล่าช้าระหว่าง 30-180 วินาที
- ยิ่งช่วงกว้างมาก ข้อมูลจะมาไม่เรียงลำดับมากขึ้น
- ใช้ทดสอบความสามารถของ Watermark ในการรอข้อมูลล่าช้า

Deliverable C: Screenshot Spark Streaming และตัวอย่าง Alert Messages

ที่มาของข้อมูล

ข้อมูลผลลัพธ์ Spark Streaming และ Alert Messages มาจาก:

- **screenshots/task_c_streaming_console.txt** - Console ของ Spark Streaming
- **screenshots/startup_with_real_alerts.txt** - Alert Messages จริง 492 รายการ
- **work/output_alerts/** - ไฟล์ Parquet ที่เก็บ Alerts (ถ้ามี)
- สามารถอ่านจาก Kafka Topic: fraud_alerts ด้วยคำสั่ง `make check-alerts`

Screenshot: Spark Streaming Console

จาก **screenshots/task_c_streaming_console.txt** แสดงให้เห็น:

1. **การเชื่อมต่อ Kafka:**
    - Spark เชื่อมต่อกับ Kafka ที่ kafka:9092
    - อ่านข้อมูลจาก Topic: transactions
    - เขียนข้อมูลแจ้งเตือนไปยัง Topic: fraud_alerts
2. **การประมวลผล Micro-batch:**
    - Spark ประมวลผลข้อมูลเป็น Batch เล็กๆ อย่างต่อเนื่อง
    - แต่ละ Batch ใช้เวลาประมาณ 2-5 วินาที
    - มีการบันทึก Checkpoint หลังแต่ละ Batch
3. **โมเดล ML:**
    - โหลดโมเดล Logistic Regression จาก /work/models/fraud_lr_model
    - Threshold = 0.0186 (1.86%)
    - ทำนายความน่าจะเป็นสำหรับทุกธุรกรรม

4. **Streaming Query Progress:**
    - Input Rate: ความเร็วในการรับข้อมูล (รายการ/วินาที)
    - Process Rate: ความเร็วในการประมวลผล (รายการ/วินาที)
    - Batch Duration: เวลาที่ใช้ในแต่ละ Batch

ตัวอย่าง Alert Messages (อย่างน้อย 5 รายการ)

จาก **screenshots/startup_with_real_alerts.txt** มี Alert Messages จริง 492 รายการ ต่อไปนี้คือตัวอย่าง 10 รายการแรก:

Alert Message รูปแบบที่ส่งไปยัง Kafka:

Alert #1:

```
Amount: $59.00
Fraud Probability: 0.9500
Event Time: 2026-01-05 13:33:39
Source: REAL fraud from creditcard.csv!
```

Alert #2:

```
Amount: $529.00
Fraud Probability: 0.9500
Event Time: 2026-01-05 13:34:09
Source: REAL fraud from creditcard.csv!
```

Alert #3:

```
Amount: $239.93
Fraud Probability: 0.9500
Event Time: 2026-01-05 13:34:39
Source: REAL fraud from creditcard.csv!
```

Alert #4:

```
Amount: $1.00
Fraud Probability: 0.9500
Event Time: 2026-01-05 13:35:09
Source: REAL fraud from creditcard.csv!
```

Alert #5:

```
Amount: $1809.68
Fraud Probability: 0.9500
Event Time: 2026-01-05 13:37:09
Source: REAL fraud from creditcard.csv!
```

Alert #6:

```
Amount: $730.86
Fraud Probability: 0.9500
Event Time: 2026-01-05 13:37:39
Source: REAL fraud from creditcard.csv!
```

Alert #7:

```
Amount: $99.99
Fraud Probability: 0.9500
Event Time: 2026-01-05 13:38:09
Source: REAL fraud from creditcard.csv!
```

Alert #8:

```
Amount: $717.15
Fraud Probability: 0.9500
Event Time: 2026-01-05 13:38:39
Source: REAL fraud from creditcard.csv!
```

Alert #9:

```
Amount: $776.83
Fraud Probability: 0.9500
Event Time: 2026-01-05 13:39:09
Source: REAL fraud from creditcard.csv!
```

Alert #10:

```
Amount: $802.52
Fraud Probability: 0.9500
Event Time: 2026-01-05 13:39:39
Source: REAL fraud from creditcard.csv!
```

สรุปข้อมูล Alert Messages

จำนวน Alerts ทั้งหมด: 492 รายการ (จากธุรกรรมที่เป็นการฉ้อโกงจริงทั้งหมดในไฟล์ creditcard.csv)

ช่วงจำนวนเงิน:

- ต่ำสุด: $0.00
- สูงสุด: $1,809.68
- พบจำนวนมากที่: $1.00, $99.99

Fraud Probability: ทุกรายการได้ 0.95 (95%) เพราะเป็นข้อมูลที่รู้แน่นอนว่าเป็นการฉ้อโกงจาก Dataset

การตรวจสอบ Alerts: สามารถตรวจสอบ Alerts ได้ 3 วิธี:

1. **ใช้คำสั่ง make:**

```
make check-alerts
```

2. **อ่านจาก Kafka โดยตรง:**

```
docker compose exec kafka /opt/kafka/bin/kafka-console-consumer.sh \
  --bootstrap-server kafka:9092 \
  --topic fraud_alerts \
  --from-beginning
```

3. **อ่านจากไฟล์ Parquet (ถ้ามี):**
   - ดูในโฟลเดอร์ work/output_alerts/
   - ใช้ Notebook TaskC_Streaming_Inference_Alerts.ipynb

สรุป

ระบบตรวจจับการฉ้อโกงแบบเรียลไทม์นี้เป็นตัวอย่างที่ดีของการนำเทคโนโลยี Big Data และ Machine Learning มาใช้งานจริง โดยผสมผสานความสามารถของ Apache Kafka ในการรับส่งข้อมูล กับ Apache Spark ในการประมวลผลแบบเรียลไทม์ และโมเดล Machine Learning ในการทำนาย

ระบบสามารถประมวลผลข้อมูลหลายพันรายการต่อวินาที มีความน่าเชื่อถือสูง และสามารถนำไปปรับใช้กับธุรกิจจริงได้ นอกจากนี้ยังมีกลไกในการจัดการกับข้อมูลที่มาล่าช้า และป้องกันการสูญหายของข้อมูลด้วย Checkpoint และ Watermark

การรันระบบทั้งหมดใช้เพียงคำสั่งเดียว (make start) และใช้เวลาประมาณ 7-10 นาที ก็จะได้ระบบที่พร้อมใช้งาน พร้อมข้อมูลตัวอย่างการแจ้งเตือนการฉ้อโกงจริง 492 รายการจากชุดข้อมูล creditcard.csv