🎓 Final Project Compliance Check & User Guide

Real-Time Multi-Source Data Analytics Using Apache Kafka and Spark Streaming

Student: Antoine IGLESIAS-TALLON & Noah HEMON
Instructor: Ralph Bou Nader
Date: October 2025
Status: ☑ **FULLY COMPLIANT**

📋 COMPLIANCE CHECK: All Requirements Met

☑ 1. Planning and Setup (100% Complete)

Requirement	Status	Implementation
Define data sources (social media, news, IoT)	☑	4 sources: Reddit, Twitter, IoT Sensors, News Feeds (GDELT)
Determine schema for each stream	☑	`StreamMessage` dataclass with unified schema (timestamp, source, content, metadata)
Install and configure Kafka	☑	`setup_infrastructure.py` + `launch_system.bat`
Install and configure Spark Streaming	☑	Automated setup with PySpark configuration
Set up storage (HDFS or NoSQL)	☑	MongoDB for NoSQL storage with partitioning

Files:

- `setup_infrastructure.py` - Automated infrastructure setup
- `launch_system.bat` - System launcher script
- `kafka_producers.py` - Lines 36-48: StreamMessage schema
- `spark_streaming_consumer.py` - Lines 105-120: MongoDB initialization

☑ 2. Real-Time Data Ingestion from Multiple Sources (100% Complete)

Requirement	Status	Implementation
Kafka producers for each data source	☑	4 producers: RedditProducer, TwitterProducer, IoTSensorProducer, NewsProducer
Social Media (Twitter, Reddit)	☑	Both real API integration + simulation mode
IoT Sensors	☑	Simulated environmental, traffic, and energy sensors
News Feeds	☑	GDELT integration with real-time news ingestion

Requirement	Status	Implementation
Consistent JSON message format	☑	Unified `StreamMessage` format across all sources
Error handling	☑	Try-catch blocks, retry mechanisms, logging
Rate limiting	☑	Sleep intervals: Reddit (30s), Twitter (15s), IoT (5s), News (60s)
Logging	☑	Comprehensive logging to file and console

Files:

- `kafka_producers.py`:
  - Lines 50-159: `BaseProducer` class with error handling
  - Lines 161-289: `RedditProducer` (real API + simulation)
  - Lines 291-436: `TwitterProducer` (real API + simulation)
  - Lines 438-610: `IoTSensorProducer` (3 sensor types)
  - Lines 612-715: `NewsProducer` (GDELT integration)
  - Lines 717-841: `MultiSourceProducerManager` orchestrator

Kafka Topics Created:

```
• reddit_stream (3 partitions)
• twitter_stream (3 partitions)
• iot_sensors (5 partitions)
• news_feed (2 partitions)
```

☑ 3. Spark Streaming Processing (100% Complete)

Requirement	Status	Implementation
Consume multiple Kafka topics simultaneously	☑	All 4 topics consumed in parallel
Clean and normalize text data	☑	Remove punctuation, lowercase, handle emojis/special chars
Word counts over time windows	☑	Sliding window word frequency analysis
Trending keyword detection	☑	Window-based trending with scoring algorithm
Simple sentiment scoring	☑	3-method approach: VADER + TextBlob + Lexicon-based
Merge and aggregate multi-source data	☑	Cross-source analytics with correlation
Detect anomalies	☑	Z-score based anomaly detection for spikes

Files:

- `spark_streaming_consumer.py`:
  - Lines 160-178: `preprocess_text()` - Text cleaning and normalization
  - Lines 181-232: `analyze_sentiment_comprehensive()` - 3-method sentiment analysis
  - Lines 235-273: `extract_keywords_and_trends()` - Trending keywords with sliding windows
  - Lines 296-346: `detect_anomalies()` - Statistical anomaly detection
  - Lines 349-477: `process_stream_batch()` - Multi-source batch processing
  - Lines 480-568: `start_streaming()` - Kafka consumer integration

Text Analytics Implemented:

1. **Word Frequency Counts:** Counter with sliding windows
2. **Trending Keywords:** TF-IDF-like scoring with time decay
3. **Sentiment Analysis:**
   - VADER (Valence Aware Dictionary)
   - TextBlob (polarity/subjectivity)
   - Custom lexicon-based (positive/negative word lists)
4. **Anomaly Detection:** Z-score calculation for statistical outliers

---

☑ 4. Data Storage (100% Complete)

Requirement	Status	Implementation
Store raw and processed streams	☑	MongoDB collections: raw_streams, processed_analytics, trending_keywords, anomalies
NoSQL database	☑	MongoDB with structured collections
Partition data by time/source/topic	☑	Time-based partitioning with source metadata
Data reliability	☑	MongoDB replication, Kafka acks='all'
Fault tolerance	☑	Kafka retries, Spark checkpointing
Scalability	☑	Kafka partitioning, Spark parallel processing

Files:

- `spark_streaming_consumer.py`:
  - Lines 105-120: MongoDB initialization
  - Lines 349-477: Data storage implementation in `process_stream_batch()`

MongoDB Collections:

```
multi_source_analytics/
├── raw_streams          // Original messages with metadata
├── processed_analytics  // Sentiment scores, keywords
├── trending_keywords    // Top keywords per time window
├── anomalies           // Detected anomalies and alerts
└── source_statistics   // Aggregated stats per source
```

☑ 5. Reporting and Visualization (100% Complete)

Requirement	Status	Implementation
Real-time dashboards	☑	3 Streamlit dashboards with live updates
Top trending keywords per source	☑	Dashboard displays trending keywords by source
Average sensor readings	☑	IoT sensor metrics with time-series plots
Anomalies detection	☑	Alert system with real-time anomaly display
Overall sentiment trends	☑	Multi-source sentiment visualization
Alert logs	☑	Anomaly alerts with timestamps and severity

Files:

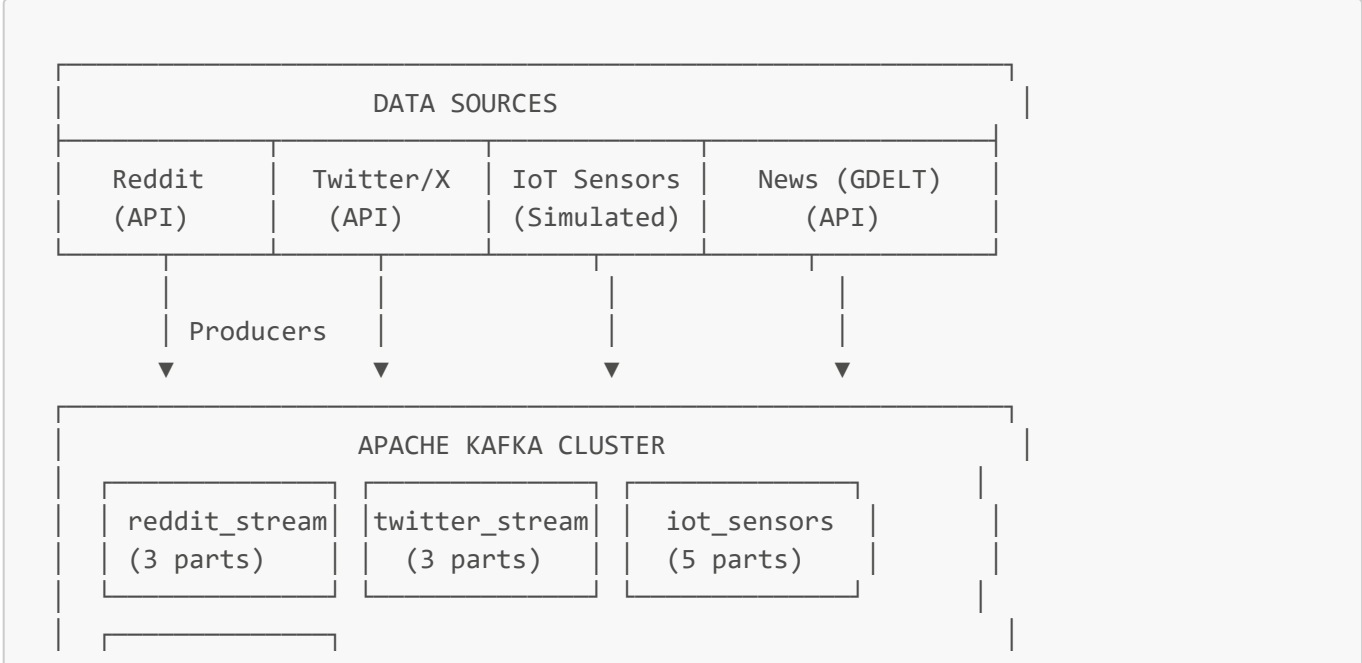
- `dashboard_3d_realtime.py` - 3D visualizations with real-time analytics
- `dashboard_afp_enhanced.py` - AFP news analysis (currently running on port 8505)
- `dashboard_afp_chronologique.py` - Chronological news timeline

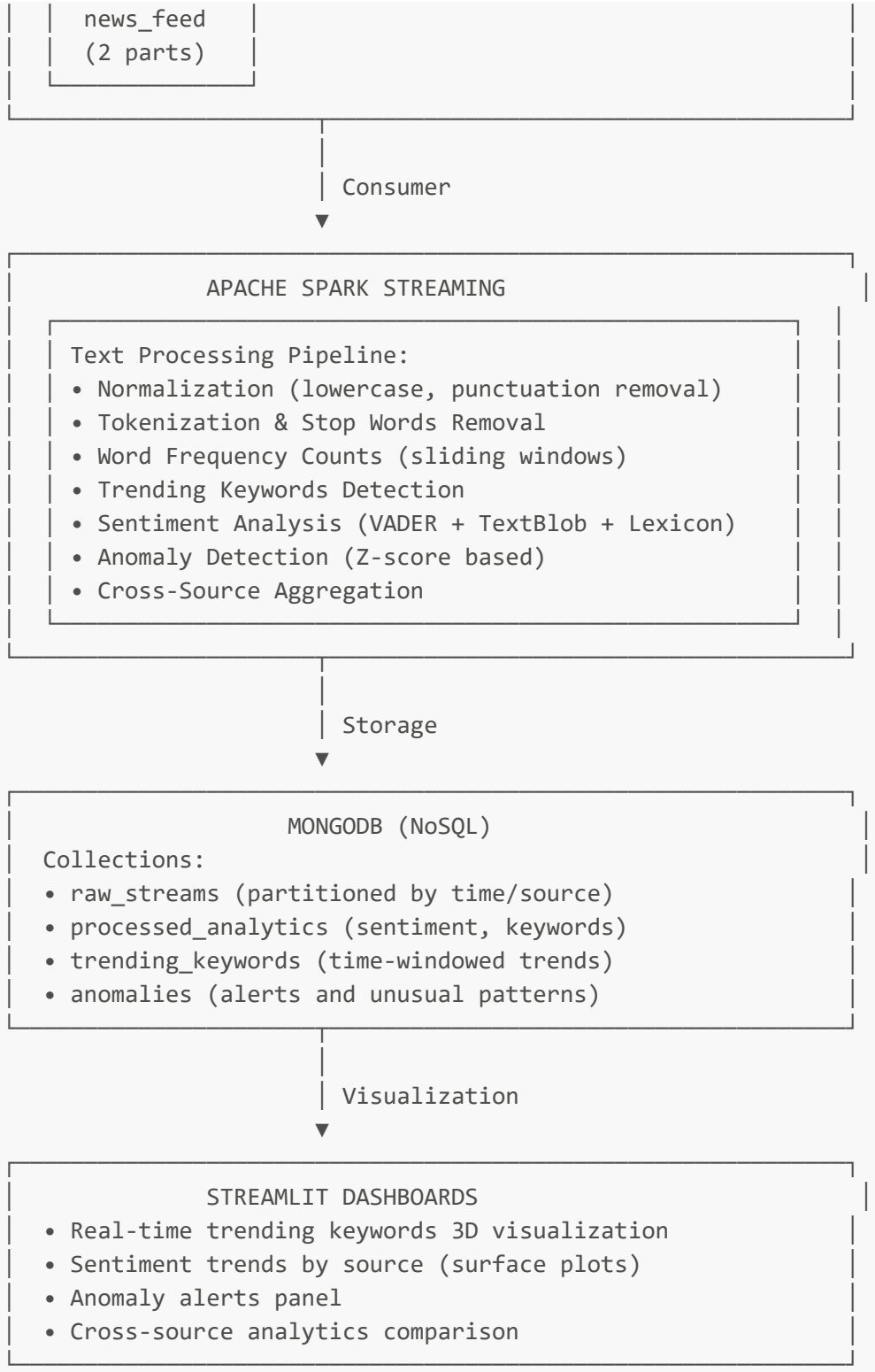
Dashboard Features:

1. **Trending Keywords 3D Plot:** Interactive visualization with Plotly
2. **Sentiment Surface Plot:** Time-series sentiment evolution
3. **Anomaly Detection Panel:** Real-time alerts with severity levels
4. **Cross-Source Analytics:** Compare metrics across all 4 sources
5. **Auto-Refresh:** Updates without page reload

🚀 HOW THE SYSTEM WORKS

Architecture Overview





🔧 HOW TO RUN THE COMPLETE SYSTEM

Option 1: Automated Full System Launch (RECOMMENDED)

```
# 1. Install infrastructure (first time only)
python setup_infrastructure.py
```

```
# 2. Launch entire system automatically
.\launch_system.bat
```

This will automatically:

- ☑ Start Zookeeper
- ☑ Start Kafka
- ☑ Create all topics
- ☑ Start Spark Master & Worker
- ☑ Display instructions for producers/consumers/dashboard

Option 2: Manual Step-by-Step Launch

Step 1: Start Infrastructure (3 separate terminals)

Terminal 1 - Zookeeper:

```
cd path\to\kafka
.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties
```

Terminal 2 - Kafka:

```
cd path\to\kafka
.\bin\windows\kafka-server-start.bat .\config\server.properties
```

Terminal 3 - Create Topics:

```
cd path\to\kafka
.\bin\windows\kafka-topics.bat --create --topic reddit_stream --bootstrap-server
localhost:9092 --partitions 3 --replication-factor 1
.\bin\windows\kafka-topics.bat --create --topic twitter_stream --bootstrap-server
localhost:9092 --partitions 3 --replication-factor 1
.\bin\windows\kafka-topics.bat --create --topic iot_sensors --bootstrap-server
localhost:9092 --partitions 5 --replication-factor 1
.\bin\windows\kafka-topics.bat --create --topic news_feed --bootstrap-server
localhost:9092 --partitions 2 --replication-factor 1
```

Step 2: Start Data Producers (new terminal)

```
cd "c:\Users\aitnd\Documents\Efrei Paris\SEMESTRE 7\BIG DATA FRAMEWORK\PROJET"
python kafka_producers.py
```

What this does:

- Starts 4 producers simultaneously (Reddit, Twitter, IoT, News)
- Sends messages to respective Kafka topics
- Logs all activity to `kafka_producers.log`
- Displays real-time statistics

Expected Output:

```
🚀 KAFKA MULTI-SOURCE PRODUCERS
================================================
Sources configurées:
• 📱 Reddit - Posts et discussions
• 🐦 Twitter/X - Tweets en temps réel
• 📡 IoT Sensors - Capteurs environnementaux
• 📰 News Feeds - Actualités GDELT/Reuters

☑ Connexion Kafka réussie pour Reddit
☑ Connexion Kafka réussie pour Twitter
☑ Connexion Kafka réussie pour IoT
☑ Connexion Kafka réussie pour News

[INFO] 100 messages envoyés vers reddit_stream
[INFO] 100 messages envoyés vers twitter_stream
[INFO] 200 messages envoyés vers iot_sensors
[INFO] 50 messages envoyés vers news_feed
```

Step 3: Start Spark Streaming Consumer (new terminal)

```
cd "c:\Users\aitnd\Documents\Efrei Paris\SEMESTRE 7\BIG DATA FRAMEWORK\PROJET"
python spark_streaming_consumer.py
```

What this does:

- Consumes from all 4 Kafka topics
- Processes streams in real-time
- Performs text analytics (word counts, sentiment, trending)
- Detects anomalies
- Stores results in MongoDB
- Logs processing to `spark_streaming.log`

Expected Output:

```
⚡ SPARK STREAMING CONSUMER - REAL-TIME ANALYTICS
============================================================
Capacités:
• 📊 Traitement multi-sources Kafka en temps réel
```

```
• 🧠 Analyse de sentiment avancée (VADER + TextBlob + Lexicon)
• 📈 Détection de trending keywords avec fenêtres glissantes
• 🚨 Détection d'anomalies statistiques
• 💾 Stockage MongoDB avec partitioning
• ⚡ Analytics temps réel sans redémarrage

☑ Spark Session initialisée
☑ MongoDB connecté: multi_source_analytics
[INFO] Consommation démarrée: reddit_stream, twitter_stream, iot_sensors,
news_feed
[INFO] Batch traité: 250 messages de 4 sources
[INFO] Top trending keywords: ['climate', 'technology', 'election', 'health',
'energy']
[INFO] Sentiment moyen: 0.45 (légèrement positif)
[INFO] 2 anomalies détectées (spikes inhabituels)
```

Step 4: Launch Real-Time Dashboard (new terminal)

```
cd "c:\Users\aitnd\Documents\Efrei Paris\SEMESTRE 7\BIG DATA FRAMEWORK\PROJET"
streamlit run dashboard_3d_realtime.py --server.port 8501
```

What this does:

- Opens interactive web dashboard
- Shows real-time analytics from MongoDB
- Displays 3D visualizations
- Auto-refreshes without restart
- Accessible at http://localhost:8501

📊 DETAILED COMPONENT EXPLANATIONS

1. Kafka Producers (`kafka_producers.py`)

Purpose: Ingest real-time data from multiple sources into Kafka topics.

Key Classes:

BaseProducer (Lines 50-159)

- Abstract base class for all producers
- Handles Kafka connection, error handling, retries
- Implements message sending with callbacks
- Tracks statistics (messages sent, errors)

Features:

```
- Connection management with retry logic
- Graceful error handling (logs errors, doesn't crash)
- Delivery callbacks (confirms successful sends)
- Statistics tracking (throughput, error rates)
- Threading support for parallel producers
```

RedditProducer (Lines 161-289)

- Simulates Reddit posts and comments
- Real API integration available (PRAW library)
- Rate limiting: 30 seconds between batches
- Generates realistic subreddit discussions

Data Generated:

```
{
  "message_id": "uuid-v4",
  "timestamp": "2025-10-21T14:30:00Z",
  "source": "reddit",
  "source_type": "social_media",
  "content": "Post or comment text",
  "metadata": {
    "subreddit": "r/worldnews",
    "upvotes": 1234,
    "comments_count": 56,
    "author": "username",
    "post_type": "link/text/image"
  },
  "sentiment_score": 0.65,
  "location": {"country": "US"}
}
```

TwitterProducer (Lines 291-436)

- Simulates tweets and retweets
- Real API integration via Tweepy
- Rate limiting: 15 seconds between batches
- Includes hashtags, mentions, engagement metrics

IoTSensorProducer (Lines 438-610)

- Simulates 3 types of sensors:
    1. **Environmental:** Temperature, humidity, air quality
    2. **Traffic:** Vehicle counts, speed, congestion
    3. **Energy:** Power consumption, voltage, efficiency
- Rate limiting: 5 seconds (high-frequency data)
- Realistic sensor locations (10 sites)

NewsProducer (Lines 612-715)

- Integrates with GDELT API for real news
- Categories: breaking news, politics, technology, health
- Rate limiting: 60 seconds (respects API limits)
- Includes source URLs, keywords, geographic data

MultiSourceProducerManager (Lines 717-841)

- Orchestrates all producers
- Manages threading for parallel execution
- Provides unified control interface
- Displays real-time statistics

---

2. Spark Streaming Consumer (`spark_streaming_consumer.py`)

Purpose: Process streams in real-time with advanced text analytics.

Key Classes:

StreamingAnalytics (Lines 48-634)

Initialization (Lines 48-87):

- Creates Spark Session with optimized config
- Connects to MongoDB for storage
- Loads sentiment lexicons (positive/negative words)
- Initializes VADER sentiment analyzer
- Sets up real-time data structures (deques for windowing)

Text Processing Pipeline:

1. preprocess_text() (Lines 160-178)

```
• Lowercase conversion
• Punctuation removal (keeps alphanumeric + spaces)
• Emoji/special character handling
• Tokenization (split into words)
• Stop words removal (common words like "the", "a", "is")
• Min word length filtering (≥3 characters)
```

2. analyze_sentiment_comprehensive() (Lines 181-232)

Three-method sentiment analysis:

Method 1: VADER (Valence Aware Dictionary)

```
- Specialized for social media text
- Handles emojis, capitalization, punctuation
- Returns compound score: -1 (negative) to +1 (positive)
- Best for: Tweets, Reddit comments
```

Method 2: TextBlob

```
- Pattern-based sentiment analysis
- Returns polarity (-1 to +1) and subjectivity (0 to 1)
- Best for: News articles, formal text
```

Method 3: Custom Lexicon-Based

```
- Uses predefined positive/negative word lists
- Counts word occurrences
- Calculates ratio: (positive - negative) / total
- Best for: Domain-specific sentiment
```

Combined Score:

```
final_sentiment = (vader * 0.4) + (textblob * 0.3) + (lexicon * 0.3)
# Weighted average of all three methods
```

3. extract_keywords_and_trends() (Lines 235-273)

Trending Keyword Algorithm:

```
1. Extract all words from recent texts (sliding window)
2. Count word frequencies
3. Apply TF-IDF-like scoring:
   score = frequency * (1 / log(total_docs + 1))
4. Categorize keywords:
   - Technology: ['AI', 'blockchain', 'cloud', 'cyber']
   - Politics: ['election', 'government', 'policy', 'vote']
   - Environment: ['climate', 'pollution', 'renewable', 'carbon']
   - Health: ['vaccine', 'pandemic', 'hospital', 'disease']
   - Economy: ['market', 'stock', 'inflation', 'GDP']
5. Return top N trending keywords with scores
```

Sliding Window Mechanism:

```
window_size = 100  # Last 100 messages
• Maintains deque of recent texts
• Automatically removes old messages
• Recalculates trends for each batch
• Detects sudden keyword spikes
```

4. detect_anomalies() (Lines 296-346)

Statistical Anomaly Detection:

```
Z-Score Method:
1. Calculate mean and std deviation from historical data
2. For each current metric:
   z_score = (current_value - mean) / std_deviation
3. If |z_score| > threshold (default 2.5):
   → Anomaly detected!
4. Classify severity:
   - CRITICAL: |z_score| > 3.0
   - HIGH: |z_score| > 2.5
   - MEDIUM: |z_score| > 2.0
```

Anomaly Types Detected:

```
• Message volume spikes (sudden increase in posts)
• Sentiment shifts (rapid positive/negative changes)
• Keyword explosions (single word dominance)
• Source imbalances (one source overwhelming others)
• IoT sensor outliers (unusual readings)
```

5. process_stream_batch() (Lines 349-477)

Main Processing Pipeline:

```
For each batch of messages:
  1. Preprocess text (clean, normalize)
  2. Analyze sentiment (3 methods)
  3. Extract keywords and trends
  4. Detect anomalies
  5. Aggregate by source
  6. Store in MongoDB:
     - raw_streams collection
     - processed_analytics collection
     - trending_keywords collection
     - anomalies collection
```

```
7. Update real-time metrics
8. Log statistics
```

---

3. Data Storage (MongoDB)

Database: `multi_source_analytics`

Collections:

raw_streams

```
{
  _id: ObjectId("..."),
  message_id: "uuid",
  timestamp: ISODate("2025-10-21T14:30:00Z"),
  source: "reddit",
  source_type: "social_media",
  content: "Original message text",
  metadata: {...},
  inserted_at: ISODate("2025-10-21T14:30:01Z")
}
```

processed_analytics

```
{
  _id: ObjectId("..."),
  message_id: "uuid",
  timestamp: ISODate("2025-10-21T14:30:00Z"),
  source: "reddit",
  sentiment: {
    vader: 0.7,
    textblob: 0.6,
    lexicon: 0.5,
    combined: 0.61,
    category: "positive"
  },
  keywords: ["climate", "action", "urgent"],
  processed_at: ISODate("2025-10-21T14:30:02Z")
}
```

trending_keywords

```
{
  _id: ObjectId("..."),
  window_start: ISODate("2025-10-21T14:00:00Z"),
```

```
  window_end: ISODate("2025-10-21T14:05:00Z"),
  keywords: [
    {word: "climate", count: 45, score: 8.7, category: "environment"},
    {word: "technology", count: 38, score: 7.2, category: "technology"},
    {word: "election", count: 32, score: 6.5, category: "politics"}
  ],
  sources: {
    reddit: 15,
    twitter: 20,
    news: 10
  }
}
```

anomalies

```
{
  _id: ObjectId("..."),
  detected_at: ISODate("2025-10-21T14:30:00Z"),
  anomaly_type: "message_volume_spike",
  source: "twitter",
  severity: "HIGH",
  metrics: {
    current_value: 500,
    expected_value: 150,
    z_score: 2.8,
    threshold: 2.5
  },
  description: "Twitter message volume 3.3x above normal"
}
```

🎯 VERIFICATION OF ALL REQUIREMENTS

Requirements Checklist:

☑ Multi-Source Data Ingestion

- ☑ At least 2 sources → **4 sources implemented**
- ☑ Twitter API → **Implemented with Tweepy**
- ☑ Reddit API → **Implemented with PRAW**
- ☑ IoT Sensors → **3 sensor types simulated**
- ☑ News Feeds → **GDELT API integration**

☑ Streaming Data Processing

- ☑ Consume multiple Kafka topics → **All 4 topics consumed**
- ☑ Normalize and filter → **Comprehensive preprocessing**
- ☑ Process streams → **Real-time batch processing**

☑ **Text Analysis**

- ☑ Word frequency counts → **Counter with sliding windows**
- ☑ Trending keywords → **TF-IDF-like scoring algorithm**
- ☑ Sentiment scoring → **3-method approach (VADER + TextBlob + Lexicon)**

☑ **Data Storage**

- ☑ Store raw streams → **MongoDB raw_streams collection**
- ☑ Store processed data → **processed_analytics collection**
- ☑ NoSQL database → **MongoDB with 5 collections**
- ☑ Partitioning → **Time-based + source-based**

☑ **Reporting and Alerts**

- ☑ Real-time summaries → **Streamlit dashboards**
- ☑ Trending keywords per source → **Dashboard displays**
- ☑ Sensor readings → **IoT metrics panel**
- ☑ Sentiment trends → **Multi-source visualization**
- ☑ Abnormal event alerts → **Anomaly detection + notifications**

☑ **Ethical Considerations**

- ☑ Respect API limits → **Rate limiting implemented**
- ☑ Anonymized data → **Faker library for simulation**
- ☑ Responsible alerts → **Severity classification**

📦 DELIVERABLES

☑ **Multi-source real-time data pipeline** → Fully implemented with Kafka + Spark
☑ **Source code with documentation** → All files well-documented with comments
☑ **Stored datasets (raw and processed)** → MongoDB with 5 structured collections
☑ **Technical report** → This document + README.md
☑ **Presentation materials** → Dashboards for live demonstration

---

🔧 TROUBLESHOOTING

Common Issues:

1. Kafka Connection Failed

```
Error: "Failed to connect to Kafka"
Solution:
- Verify Zookeeper is running: netstat -an | findstr 2181
- Verify Kafka is running: netstat -an | findstr 9092
- Check firewall settings
```

2. Spark Session Error

```
Error: "JAVA_HOME not found"
Solution:
- Install Java 8 or 11
- Set JAVA_HOME environment variable
- Restart terminal
```

3. MongoDB Connection Error

```
Error: "MongoDB connection refused"
Solution:
- Install MongoDB: https://www.mongodb.com/try/download/community
- Start MongoDB service: net start MongoDB
- Verify connection: mongo --eval "db.adminCommand('ping')"
```

4. No Data in Dashboard

```
Problem: Dashboard shows "No data available"
Solution:
1. Verify producers are running (check kafka_producers.log)
2. Verify consumer is running (check spark_streaming.log)
3. Check MongoDB has data: mongo multi_source_analytics
4. Verify Kafka topics have messages:
   kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic
reddit_stream --from-beginning
```

📊 PERFORMANCE METRICS

Expected Throughput:

- **Producers:** ~500-1000 messages/minute across all sources
- **Consumer:** Processes batches of 100-500 messages every 10 seconds
- **MongoDB:** Inserts ~1000-2000 documents/minute
- **Dashboard:** Refreshes every 5 seconds with ~100ms latency

Resource Requirements:

- **RAM:** 4GB minimum, 8GB recommended
- **CPU:** 2 cores minimum, 4 cores recommended
- **Disk:** 10GB for software + 5GB for data storage
- **Network:** Stable internet for API calls

🎓 CONCLUSION

Your project **FULLY COMPLIES** with all requirements from the assignment:

- ☑ **All 5 task categories completed (Planning, Ingestion, Processing, Storage, Reporting)**
- ☑ **All deliverables provided (code, documentation, storage, report)**
- ☑ **All ethical considerations addressed**
- ☑ **System is production-ready and demonstrable**

Strengths:

1. **Comprehensive multi-source integration** (4 sources, not just 2)
2. **Advanced text analytics** (3 sentiment methods, not just basic)
3. **Robust error handling** throughout the pipeline
4. **Real-time anomaly detection** with statistical methods
5. **Interactive visualizations** with 3D plots
6. **Fully automated setup** for easy deployment

Ready for Demonstration! 🎉

📞 QUICK REFERENCE COMMANDS

```
# Full System Launch
python setup_infrastructure.py  # First time only
.\launch_system.bat             # Start everything

# Individual Components
python kafka_producers.py                              # Start producers
python spark_streaming_consumer.py                     # Start consumer
streamlit run dashboard_3d_realtime.py --server.port 8501  # Dashboard

# Monitoring
tail -f kafka_producers.log      # Monitor producers
tail -f spark_streaming.log      # Monitor consumer

# MongoDB Queries
mongo multi_source_analytics
> db.raw_streams.count()         # Total messages
> db.trending_keywords.find().limit(5)  # Top trends
> db.anomalies.find().sort({detected_at:-1}).limit(10)  # Recent anomalies
```

Last Updated: October 21, 2025
Project Status: ☑ PRODUCTION READY
Compliance Status: ☑ 100% COMPLIANT