BFSI-Experiential Learning Programme

Activity Report on

"Real-Time Transaction Monitoring Dashboard"

Under the Guidance of

Navya R

Nanda Kumar M

SL.No	Team Members	Role	Institution
1.	Sarvesh Raju Bavache	Team Lead	Atria Institute of Technology
2.	Shivani B	Business Analyst	Sai Vidya Institute of Technology
3.	Spoorthi C	Business Analyst	KNS Institute of Technology
4.	Prasad R Kakde	Planning and designing	BMS Institute of Technology
5.	Kavya R Naik	Planning and designing	HKBK College of Engineering

Table of Contents

List of Figures

REAL-TIME TRANSACTION MONITORING DASHBOARD

Executive Summary

Digital transformation has completely changed how banks operate today. In earlier days, most transactions used to take place manually or through limited digital systems. But now, with the rise of online payments, UPI, credit card transactions, and mobile banking, millions of transactions occur every minute.

With this massive increase, it has become essential for banks and financial institutions to monitor these transactions in real time not only to ensure smooth performance but also to detect frauds and failed payments as soon as they happen.

To understand and demonstrate this concept, our team developed a project called "Real-Time Transaction Monitoring Dashboard", which falls under the Payments and Digital Channels domain of BFSI.

Our project aims to simulate how real-time banking dashboards function inside large organizations. We created a system that shows transaction counts, failures, frauds, and processing times instantly on an interactive dashboard. The dataset we used was randomly generated using Python, but it closely imitates real bank transaction data.

The dashboard was built using Streamlit, a Python library that helps create web-based dashboards easily. Behind the scenes, we used libraries like Pandas, NumPy, Scikit-learn, Random Forest and XGBoost to process data and detect fraudulent patterns. The project was deployed locally and successfully visualized data in real time, reflecting how actual payment systems are monitored in financial institutions.

Objective of the Project

- To build a dashboard that can monitor bank transactions in real time.
- To predict potential fraud transactions using a machine learning model.
- To simulate a real-world transaction monitoring environment using synthetic data.
- To create a simple, interactive, and auto-refreshing dashboard using Streamlit.

Business Requirements

- Provide real-time visibility into transaction activity across digital channels.
- Detect transaction failures or delays quickly to safeguard customer experience.
- Identify suspicious or abnormal patterns indicative of fraud or system issues.
- Offer a monitoring tool that supports compliance with BFSI regulations.
- Simulate digital payment monitoring using synthetic data while ensuring realism.
- Provide an interface that reflects the way banks handle real-time anomaly detection.

Functional Requirements

- As a **Bank Operations Manager**, I want to view the total number of transactions happening in real time so that I can continuously monitor the payment system's performance and ensure that all digital channels are functioning without disruption.
- As a **Fraud Analyst**, I want the system to automatically detect suspicious or potentially fraudulent transactions using machine learning techniques so that I can take timely action and prevent any possible financial or reputational damage to the organization.
- As a **Data Analyst**, I want to track the average processing time of each transaction so that I can identify delays or system inefficiencies and report them to the technical team for optimization.
- As a **Compliance Officer**, I want to view summarized graphs and trends of transaction performance over time so that I can ensure the institution remains compliant with regulatory and operational guidelines.
- As a **Technical Support Engineer**, I want to have visibility into live logs and transaction activity updates so that I can identify any technical failures or system lags early and maintain continuous system availability.

Non-Functional Requirements

- The dashboard must refresh automatically every few seconds.
- The data processing should happen efficiently without delay.
- The interface should be clean and easy to understand.
- The solution must be reliable and stable during continuous execution.
- The design should align with BFSI standards, such as data privacy and auditability.

Acceptance Criteria

For the Bank Operations Manager

- The dashboard must display the total transaction count in real time.
- The count should update automatically without the need for manual refresh.
- The data displayed should always match the current dataset being processed.

For the Fraud Analyst

- The machine learning model (XGBoost) must flag suspicious or abnormal transactions.
- Each transaction marked as "fraudulent" must be displayed clearly in a separate section.
- The fraud detection output should update dynamically as new transactions are generated.
- The fraud count should always match the ML model's prediction results.

For the Data Analyst

- The system must calculate and display the average time of transactions.
- The processing time trend should be represented visually (e.g., line graph).
- The calculated average should update in real time as new data comes in.

For the Compliance Officer

- The dashboard should show transaction performance trends over a defined period.
- Graphs should display historical data as well as live data updates.
- Compliance metrics such as fraud rate should be clearly visible.
- The layout and data should support easy extraction for reports and audits.

For the Technical Support Engineer

- The system must log transaction activity and status changes in real time.
- The dashboard should display alerts or notifications for failures or anomalies.
- Technical staff should be able to verify that the dashboard remains responsive throughout operation.
- Any error or disconnection should be captured and logged locally for debugging.

Scope of the project

This project scope was limited to simulated real-time monitoring using Python. We didn't integrate real APIs, but the design supports it in the future. The project includes:

- Data generation
- Data processing

- Fraud detection using ML
- Dashboard visualization
- Local deployment

Dataflow diagram

Fig 1: Dataflow diagram

1. Start

The process begins when the system is initiated. This marks the start of the transaction monitoring cycle. It sets up all the required modules, including data generation, fraud model loading, and the dashboard interface.

2. Generate Transaction Data

In real banking systems, data comes from multiple payment gateways, but since our project uses simulated data, we generate synthetic transaction records using Python. Each transaction includes details such as:

- Transaction ID
- Amount
- Location
- Fraud Probability
- Processing Time
- Status

The data is generated continuously to mimic real-time transactions happening in a bank's digital payment channel.

3. Train Fraud Detection Model

Before we can predict frauds, we need a trained model. At this stage, the XGBoost Regressor Classifier is trained using the generated dataset. This model learns from historical transaction patterns to understand what type of behavior might indicate fraud.

For example:

A transaction is counted as FRAUD if:

1. Simulator marked it as fraud (`is_fraud = 1`), OR
2. ML model predicted fraud probability ≥ 75% (when simulator flag is 0)

By identifying such relationships, the model becomes capable of flagging suspicious transactions.

4. Predict Fraud on Live Stream

Once the model is trained, it starts analyzing live incoming data. For every new transaction generated, the system instantly applies the model to:

- Predict whether the transaction is normal or potentially fraudulent.
- Assign a risk score to each record.

This mimics real-world fraud detection systems used in banks that run predictive checks before approving transactions.

5. Update Dashboard

The dashboard, built using Streamlit, acts as the live visual layer of the project. Here, all metrics such as:

- Total transactions
- Successful vs. failed counts
- Fraud count
- Processing time trends are updated automatically in real time.

Whenever new data is processed, the dashboard refreshes and shows the latest information to the user, ensuring constant visibility into the system's performance.

6. Display Alerts / Fraud Trends

Alerting is one of the most critical stages. If the fraud detection model identifies a transaction as suspicious or risky, the dashboard immediately displays:

- A visual alert (like a red highlight or warning icon).

- The fraud trend graph, showing the pattern of risky transactions over time.

These alerts help banking or operations teams take quick action either by reviewing the transaction manually or flagging the user account temporarily.

7. End

The process completes one full cycle when all transactions for a given stream are analyzed. However, since the dashboard works continuously, the process repeats automatically making it a looping real-time system rather than a one-time workflow.

Architecture Diagram

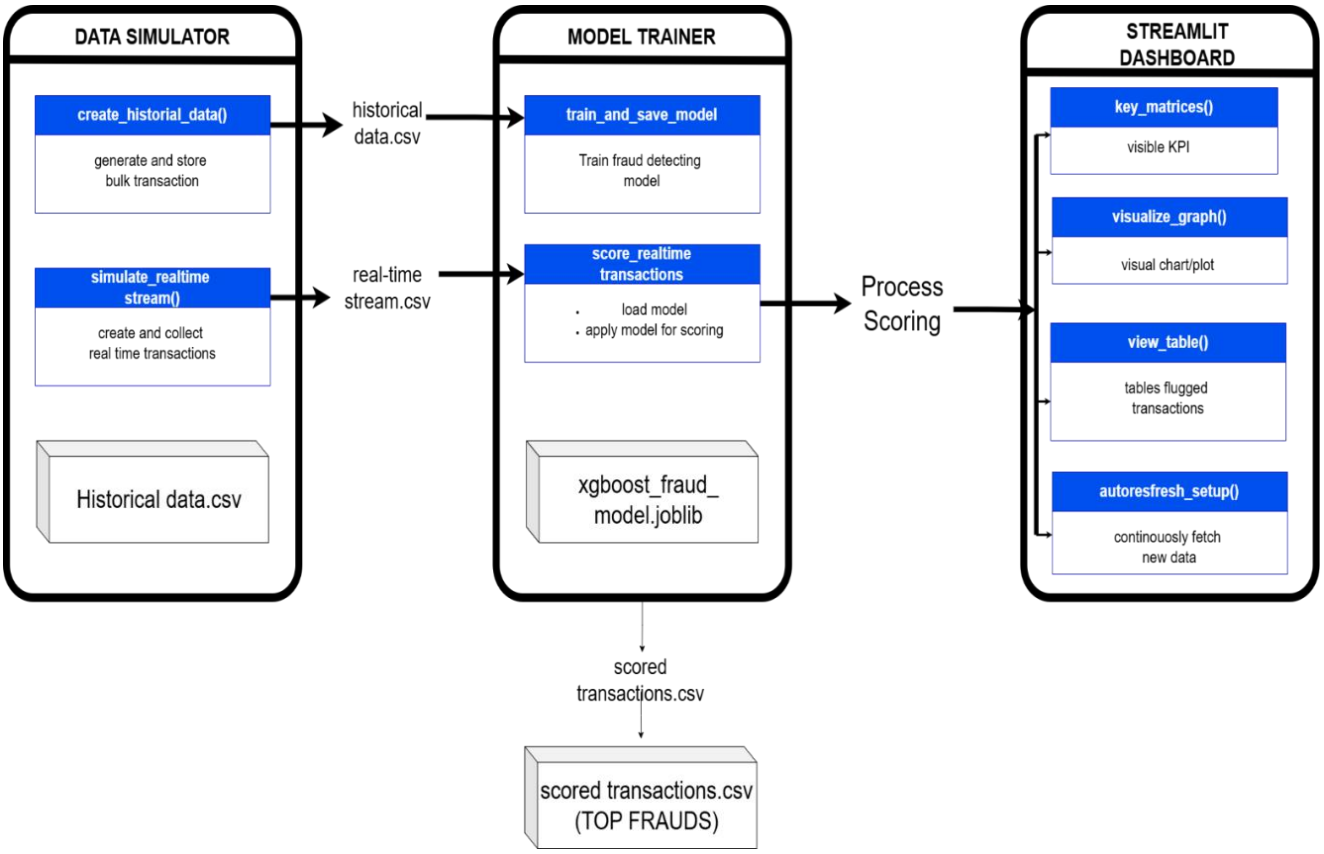


Fig 2: Architecture diagram

1. Overview

- Our project detects fraudulent transactions in real time.
- It has **three layers**:
    1. **Data Simulator Layer** – generates and stores data.
    2. **Processing Layer (AI Engine)** – trains model and scores live transactions.
    3. **Visualization Layer (Dashboard)** – displays real-time fraud alerts and analytics.

2. Data Simulator Layer

a. create_historical_data()

- Generates synthetic historical transaction data.
- Saves it as Historical data.csv.
- This file contains past transactions used to train the model.

Output: Historical data.csv

b. simulate_realtime_stream()

- Simulates a continuous live data feed of transactions.
- Mimics how banks receive transactions every few seconds.
- Provides the live stream for scoring by the AI model.

Purpose: To imitate real-time banking transaction flow.

3. Model trainer

a. train_and_save_model()

- Uses Historical data.csv to train an XGBoost-based ML model.
- Learns the difference between fraudulent and legitimate transactions.
- Saves the trained model as xgboost_fraud_model.joblib.

Output: xgboost_fraud_model.joblib

b. score_realtime_transactions()

- Reads incoming data from the live stream.
- Loads the trained ML model from xgboost_fraud_model.joblib.
- Predicts fraud probability for each transaction.
- Saves results in scored_transactions.csv containing scores and classifications.

Output: scored_transactions.csv

4. Streamlit Dashboard

a. key_matrices()

- Displays key performance metrics such as:
    - Total transactions processed
    - Fraud detection rate
    - Accuracy and precision

b. visualize_graph()

- Creates real-time graphs and charts.
- Shows fraud vs. normal trends, transaction volume, etc.

c. view_table()

- Displays a live data table of scored transactions.
- Lists top high-risk transactions (frauds).

d. autoresfresh_setup()

- Enables auto-refresh for real-time updates.
- Ensures dashboard visuals keep updating as new data arrives.

Dashboard Explanation

The dashboard we built is designed to give a clear, real-time view of how transactions are happening in the system. The idea is to mimic how actual banks monitor digital transactions every second. Below is a detailed explanation of each section of the dashboard based on the screenshots.

1. Dashboard Settings Panel

On the left side, we have a control panel that allows the user to adjust how the dashboard behaves.

Auto-refresh Interval

- This lets the user choose how often the dashboard should update automatically.
- For example, if it is set to 5 seconds, then every 5 seconds the dashboard pulls new data and refreshes all visuals.
- This simulates how banks keep dashboards auto-updated without manual effort.

Show Recent Fraudulent Transactions (Checkbox)

- When this box is checked, the dashboard displays the list of the most recent suspicious or fraudulent transactions.
- This is useful for fraud analysts who want to quickly see what kinds of transactions are being flagged in real time.

Tip Section

- This is a small guidance message telling the user the time taking to update.
- It makes the dashboard more user-friendly.

Overall, the settings panel improves interactivity and control, giving the user the flexibility to adjust how the system behaves.

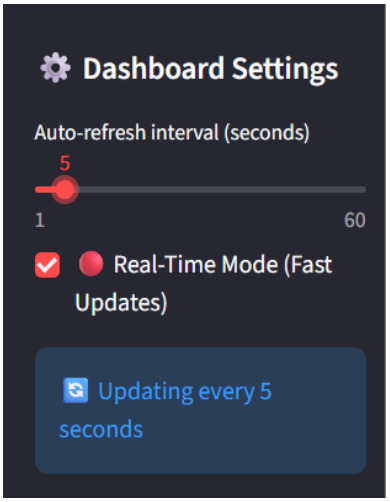


Fig 3: Dashboard setting

2. Key Metrics Section

This is one of the most important sections, because it shows a quick summary of what is happening.

The key metrics shown are:

Total Transactions

- This shows the number of transactions processed so far.
- Example: If it shows 200, it means 200 transactions have been generated and analyzed.

Fraudulent Transactions

- This tells how many transactions out of the total were identified as fraudulent.
- Example: 9 fraudulent transactions means that the ML model flagged 9 transactions as suspicious.

Fraud Rate (%)

- This is the percentage of transactions that are fraudulent.
- If 9 out of 200 are fraudulent, fraud rate = 4.5%.
- This is a very important KPI for banks to track system risk.

Average Amount (₹)

- This displays the average transaction amount.
- It helps see whether customers are transacting small or high values overall.

Total Volume (₹)

- Represents the cumulative value of all transactions processed.
- This metric is important for monitoring overall business impact.

These metrics help the operations team instantly understand the situation without going through detailed graphs.

A high-risk alert message is displayed when fraud levels cross a defined threshold.

- This alert is intentionally highlighted to ensure immediate attention.
- The alert also indicates that multiple transactions have very high fraud probability, meaning they are more likely to be fraudulent based on the model's analysis.
- Such alerts guide fraud analysts and operations teams to prioritize investigation and corrective actions.
- Overall, this section acts as an early-warning mechanism, helping banks detect risks quickly and respond before issues escalate.

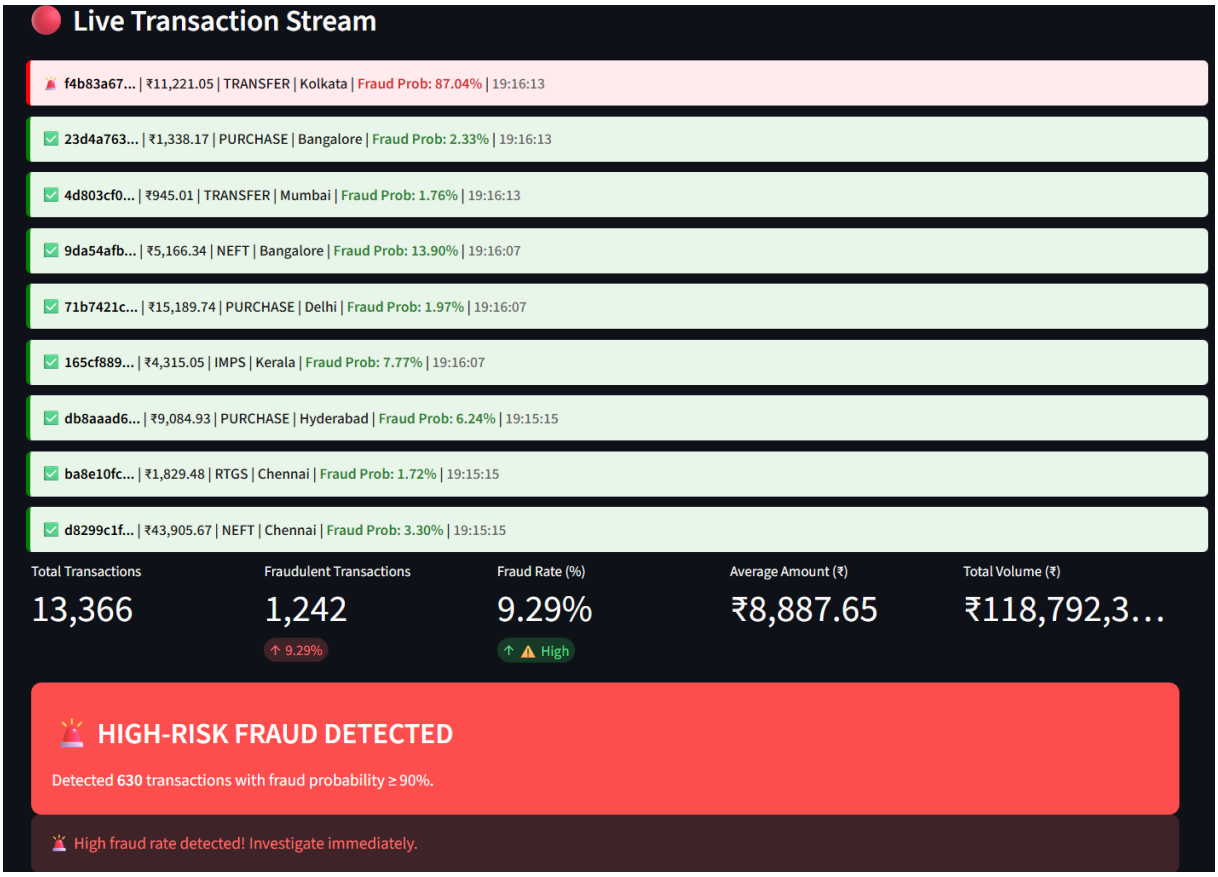


Fig 4: Key Metrics

3. Fraud Rate Gauge Chart

This is a semi-circular Fraud Risk Gauge that visually represents the overall fraud rate in the system. Instead of reading detailed numbers, users can quickly understand the risk level by looking at the position of the pointer.

- Green Zone (0–20%) → Low Risk
  Transactions are mostly normal, and the system is operating within safe limits.
- Yellow Zone (20–40%) → Moderate Risk
  Some suspicious activity is detected, and closer monitoring may be required.
- Red Zone (40–100%) → High Risk

A large number of transactions are potentially fraudulent, indicating an urgent need for investigation.

- At the beginning, the fraud risk gauge showed a very low fraud rate of 1.6%, which falls in the green zone. This indicated that most transactions were normal, the system was stable, and there was no immediate fraud concern.
- As transactions continued to flow into the system over the next 2 hours, the model kept analyzing each transaction in real time and updating the fraud predictions continuously.
- After 2 hours of live transactions, the fraud rate increased significantly to 9.3%, moving the gauge into the red zone. This shift clearly shows a rise in suspicious transaction activity.
- This comparison demonstrates how fraud patterns can change over time. The dashboard helps banks quickly notice such risk escalation and take immediate preventive actions, such as blocking transactions or alerting fraud teams.

In this dashboard, the fraud rate is 9.3, and the pointer clearly lies in the red zone, indicating high fraud risk. This suggests that the system is detecting a large volume of suspicious transactions and requires immediate attention and investigation.

Such gauge visuals are widely used in bank monitoring rooms because they provide instant, intuitive insights, allowing teams to respond quickly to potential fraud threats.

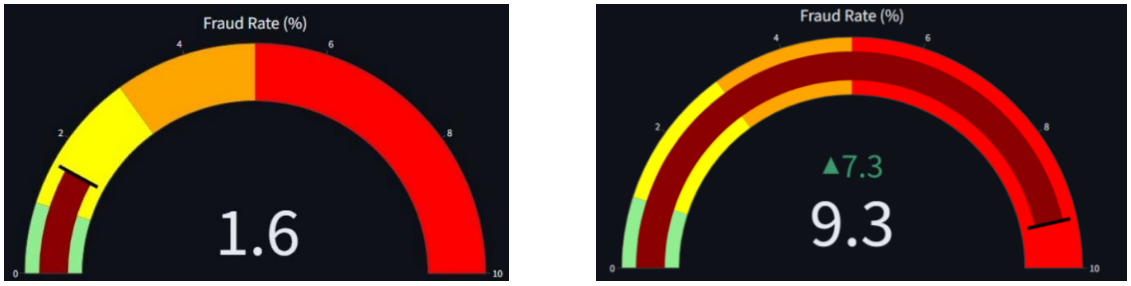


Fig 5: Fraud rate gauge chart

4. Transaction Volume Over Time

This chart represents the real-time transaction activity over the last 30 minutes, showing how transaction volume changes minute by minute.

- Each point on the graph represents the total number of transactions processed in a particular minute.
- The x-axis displays the timeline with timestamps, indicating when the transactions occurred.
- The y-axis shows the transaction count at each point in time.
- The blue line represents total transactions, while the red line represents fraudulent

transactions detected during the same period.

From the chart, we can observe that the overall transaction volume remains relatively steady, with minor fluctuations. This indicates that the transaction system is functioning normally and handling a consistent load. The fraudulent transactions remain much lower than total transactions, which is expected in a healthy payment system.

This visualization helps in understanding:

- Transaction load patterns over time
- Sudden spikes or drops that may indicate abnormal user activity or system issues
- Increase in fraudulent transactions, even when total volume appears stable

Banks and financial institutions use such real-time timelines to monitor system health, ensure capacity readiness, and quickly identify unusual activity without needing to analyze raw transaction logs.

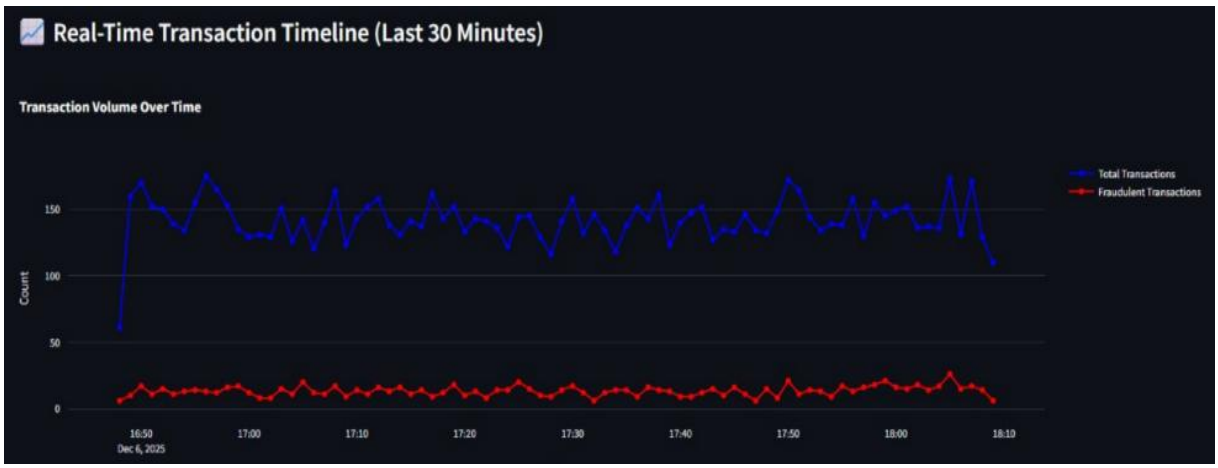


Fig 6: Transaction volume over time graph

5. Fraud Analysis dashboard

The Fraud Analysis Dashboard provides a deeper analytical view of fraudulent transactions by breaking them down across different dimensions such as transaction amount, transaction type, risk score, and geographic location. This section helps banks and monitoring teams understand *where*, *how*, and *to what extent* fraud is occurring.

a. Fraud Amount Distribution

This chart shows the distribution of fraudulent transaction amounts.

- The x-axis represents the transaction amount in rupees.
- The y-axis represents the number of fraudulent transactions within each amount range.
- Most fraudulent transactions are concentrated in the lower amount ranges, indicating that fraudsters often attempt multiple small-value transactions to avoid immediate detection.

- As the transaction amount increases, the number of fraudulent transactions decreases, but high-value frauds still exist, which can cause significant financial impact.

At the bottom, the dashboard also displays:

- Total fraud amount, which gives a cumulative financial loss.
- Average and maximum fraud amount, helping the bank understand typical fraud size versus extreme cases.

This analysis helps banks prioritize both high-frequency small frauds and low-frequency high-value frauds.

b. Fraud by Transaction Type

This bar chart represents the number of fraudulent transactions across different transaction types such as Withdrawal, RTGS, Purchase, IMPS, UPI, NEFT, Deposit, and Transfer.

- Each bar shows how frequently fraud occurs for a particular transaction type.
- The color intensity reflects the fraud rate, making it easy to identify higher-risk channels.
- Digital payment modes such as UPI, IMPS, and RTGS show relatively higher fraud counts, highlighting their exposure due to high usage and real-time processing.

This visualization helps banks:

- Identify high-risk transaction channels
- Apply stricter controls or limits on specific payment modes
- Improve channel-specific fraud prevention strategies

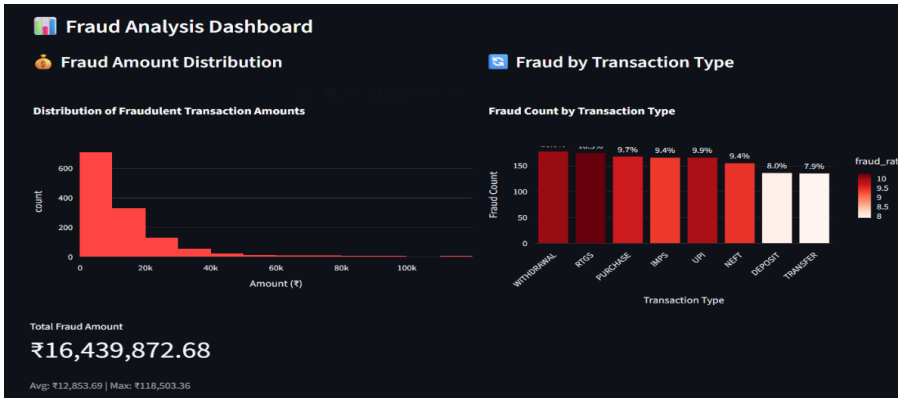


Fig 7: Fraud amount distribution and fraud by transaction type dashboard

c. Fraud Risk Score Distribution

This chart displays the distribution of fraud probability scores generated by the machine learning model.

- The x-axis shows fraud probability values ranging from 0 to 1.
- The y-axis represents the number of transactions falling into each probability range.

- A threshold line (0.5) is marked to separate normal transactions from suspicious ones.
- Transactions above this threshold are considered potential frauds and are flagged for further action.

The summary below the chart categorizes transactions into:

- Low Risk (<50%)
- Medium Risk (50–90%)
- High Risk (>90%)

This allows banks to prioritize alerts, ensuring that highly suspicious transactions are investigated first instead of treating all alerts equally.

d. Top Fraud Locations

This chart shows the geographical distribution of fraudulent transactions.

- The x-axis lists different cities or locations.
- The y-axis shows the fraud count per location.
- The color scale represents the total fraud amount, helping identify regions with higher financial impact.
- One location is highlighted as having the highest fraud count and total fraud value, making it easier to focus investigation efforts.

This analysis supports:

- Region-based monitoring
- Deployment of location-specific fraud rules
- Better coordination with regional fraud investigation teams

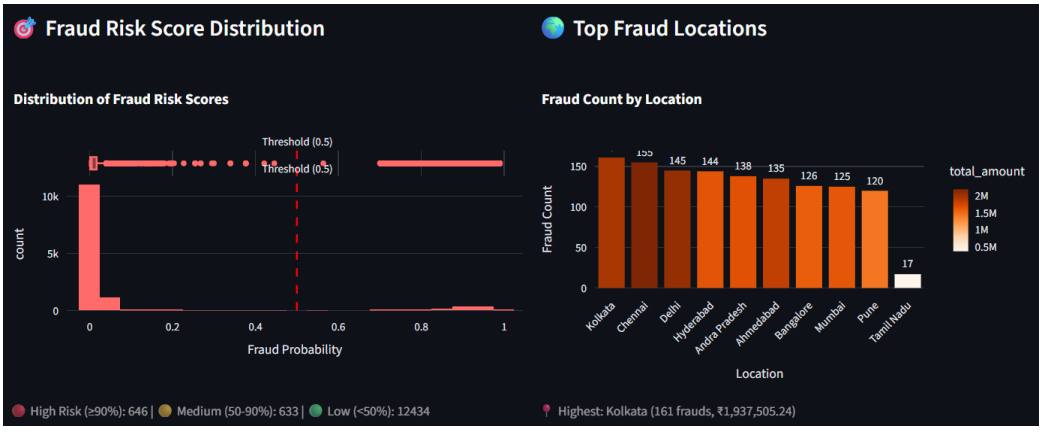


Fig 8: Fraud Risk Score Distribution and Top Fraud Locations dashboard

6. Fraudulent Transactions Trend

This chart illustrates how fraudulent activity varies across different hours of the day, helping identify time periods when fraud risk is higher.

- The x-axis represents the hour of the day during which transactions were processed.
- The left y-axis (bars) shows the number of fraudulent transactions detected in each hour.
- The right y-axis (line) represents the fraud rate percentage, indicating the proportion of fraud relative to total transactions during that hour.
- The red bars indicate fraud count, while the yellow line shows the trend of fraud rate.

From the chart, we can observe that:

- The highest number of fraudulent transactions occurs around 17:00 hours, suggesting a peak fraud activity period.
- Even when the fraud count reduces in later hours, the fraud rate percentage increases, especially around 19:00–20:00 hours, indicating that fewer transactions are happening but a larger proportion of them are fraudulent.
- Early hours show comparatively lower fraud counts and rates, reflecting more stable transaction behavior.

This analysis is particularly useful for banks because:

- It helps identify high-risk time windows during the day.
- Banks can deploy additional monitoring or stricter validation rules during peak fraud hours.
- Fraud investigation teams can be better staffed and prepared during critical periods.

Overall, this hourly trend analysis enables proactive fraud prevention by aligning monitoring efforts with time-based risk patterns rather than treating all hours equally.

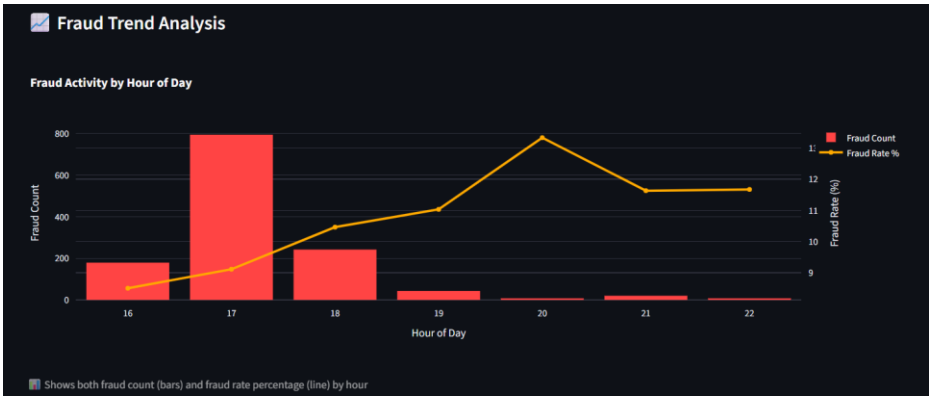


Fig 9: Fraudulent transactions trend graph

7. Transactions Review

a. Manual Fraud Verification

This screen is used by an analyst to review suspicious transactions before confirming fraud.

This diagram shows:

- A list of flagged transactions that look suspicious based on system analysis.

- Each row is one transaction that needs human review.

The columns are:

- **Transaction ID** – Unique ID to track the transaction.
- **Timestamp** – When the transaction happened.
- **Sender Account / Receiver Account** – Who sent money and who received it.
- **Amount** – How much money was involved.
- **Fraud Probability** – System's confidence that this transaction is fraudulent.
- **Transaction Type** – Method used (UPI, IMPS, NEFT, Withdrawal, Purchase, etc.).
- **Location** – City where the transaction occurred.
- **Checkbox (Not Fraud)** – Analyst can mark a transaction as *not fraud* if it looks legitimate.

Here:

- The system does not auto-block transactions.
- A human reviewer checks each transaction manually.
- Only transactions confirmed as fraud move to the next table.
- The green message at the bottom confirms:
    - 50 transactions marked as FRAUD
    - None marked as "Not Fraud" yet

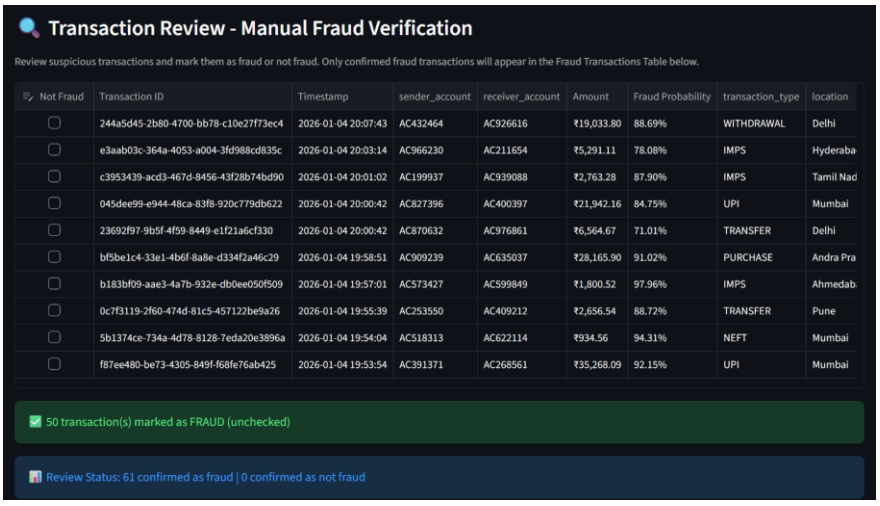


Fig 10: Manual Fraud Verification Table

b. Confirmed Fraud Transactions

This screen shows the final list of fraud cases after human confirmation.

This diagram shows:

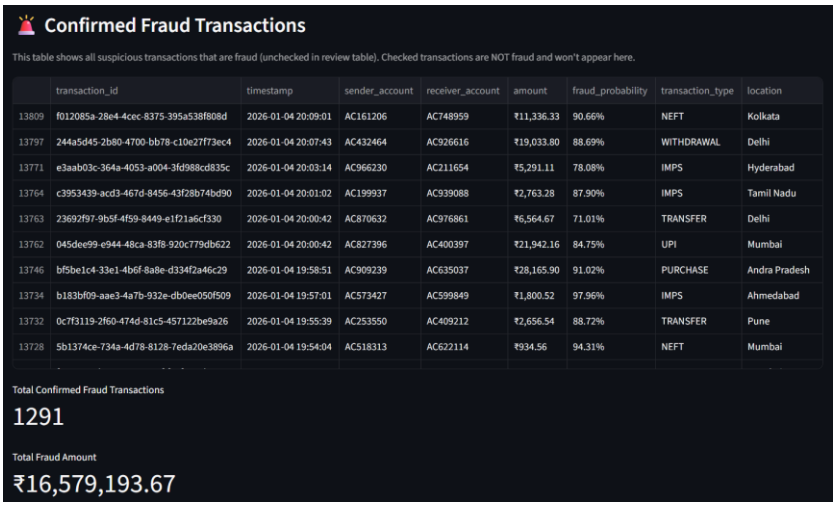
- Only transactions that were verified and confirmed as fraud.
- Clean, filtered data - no false positives here.

Key insights from this table:

- Shows the same transaction details, but only for fraud cases.
- Transactions marked as “Not Fraud” do not appear here.
- Helps teams focus on real financial threats.

This is the final fraud record used for:

- Reporting
- Audits
- Recovery actions
- Law enforcement or compliance



Confirmed Fraud Transactions

This table shows all suspicious transactions that are fraud (unchecked in review table). Checked transactions are NOT fraud and won't appear here.

	transaction_id	timestamp	sender_account	receiver_account	amount	fraud_probability	transaction_type	location
13809	f012085a-28e4-4cec-8375-395a538f808d	2026-01-04 20:09:01	AC161206	AC748959	₹11,336.33	90.66%	NEFT	Kolkata
13797	244a5d45-2b80-4700-bb78-c10e27f73ec4	2026-01-04 20:07:43	AC432464	AC926616	₹19,033.80	88.69%	WITHDRAWAL	Delhi
13771	e3aab03c-364a-4053-a004-3fd988cd835c	2026-01-04 20:03:14	AC966230	AC211654	₹5,291.11	78.08%	IMPS	Hyderabad
13764	c3953439-acd3-467d-8456-43f28b74bd90	2026-01-04 20:01:02	AC199937	AC939088	₹2,763.28	87.90%	IMPS	Tamil Nadu
13763	23692f97-9b5f-4f59-8449-e1f21a6cf330	2026-01-04 20:00:42	AC870632	AC976861	₹6,564.67	71.01%	TRANSFER	Delhi
13762	045dee99-e944-48ca-83f8-920c779db622	2026-01-04 20:00:42	AC827396	AC400397	₹21,942.16	84.75%	UPI	Mumbai
13746	bf5be1c4-33e1-4b6f-8a8e-d334f2a46c29	2026-01-04 19:58:51	AC909239	AC635037	₹28,165.90	91.02%	PURCHASE	Andra Pradesh
13734	b183bf09-aae3-4a7b-932e-db0ee050f509	2026-01-04 19:57:01	AC573427	AC599849	₹1,800.52	97.96%	IMPS	Ahmedabad
13732	0c7f3119-2f60-474d-81c5-457122be9a26	2026-01-04 19:55:39	AC253550	AC409212	₹2,656.54	88.72%	TRANSFER	Pune
13728	5b1374ce-734a-4d78-8128-7eda20e3896a	2026-01-04 19:54:04	AC518313	AC622114	₹934.56	94.31%	NEFT	Mumbai

Total Confirmed Fraud Transactions

1291

Total Fraud Amount

₹16,579,193.67

Fig 11: Confirmed Fraud Transactions

Snapshots of Machine Learning Models

```python
def generate_transaction(force_fraud=False):
    now = datetime.now()
    transaction_type = random.choice(transaction_types)
    location = random.choice(locations)
    fraud_rate = get_dynamic_fraud_rate(transaction_type, location)

    # REALISTIC FRAUD RATE - 4% chance of fraud
    if force_fraud or random.random() < 0.04:
        is_fraud = True
        # High-risk fraud transactions with high probability
        fraud_prob = round(random.uniform(0.85, 0.99), 4)
        # Fraud transactions often have unusual amounts
        amount = round(np.random.exponential(scale=15000) + 1000, 2)  # Higher amounts for fraud
    else:
        is_fraud = random.random() < fraud_rate
        fraud_prob = round(random.uniform(0.7, 0.99), 4) if is_fraud else round(random.uniform(0.01, 0.3), 4)
        amount = round(np.random.exponential(scale=8000) + 500, 2)

    return {
        "transaction_id": str(uuid.uuid4()),
        "timestamp": now.strftime("%Y-%m-%d %H:%M:%S"),
        "processed_time": now.strftime("%Y-%m-%d %H:%M:%S"),
        "sender_account": f"AC{random.randint(100000, 999999)}",
        "receiver_account": f"AC{random.randint(100000, 999999)}",
        "amount": amount,
        "transaction_type": transaction_type,
        "location": location,
        "is_fraud": int(is_fraud),
        "fraud_probability": fraud_prob
    }
```

Fig 12: generate_transaction() function

This function creates one synthetic banking transaction that closely mimics how real transactions look in actual banking systems. It generates natural-looking transaction amounts using a realistic

statistical distribution and ensures that only a small percentage of transactions (around 3–4%) are marked as fraud, which reflects real-world banking behavior. Fraudulent transactions are deliberately designed to stand out, they usually involve unusually high amounts, riskier transaction types like transfers or withdrawals, and locations known for higher fraud activity. In contrast, normal transactions represent everyday banking actions such as routine purchases, deposits, or small transfers. Each transaction is also enriched with a timestamp, unique sender and receiver account IDs, and a fraud probability score indicating how suspicious it appears. Overall, the function outputs a clean, well-structured transaction record that is ideal for use in dashboards, simulations, and fraud detection or machine learning models.

```python
def load_realtime_data():
    """Load and process real-time transaction data"""
    if not os.path.exists(REALTIME_FILE):
        return pd.DataFrame()

    try:
        # Read CSV
        df_rt = pd.read_csv(REALTIME_FILE, low_memory=False)
        if df_rt.empty:
            return pd.DataFrame()

        # Convert transaction_id to string
        if 'transaction_id' in df_rt.columns:
            df_rt['transaction_id'] = df_rt['transaction_id'].astype(str)
        else:
            return pd.DataFrame()

        # Handle numeric columns
        if 'amount' in df_rt.columns:
            df_rt['amount'] = pd.to_numeric(df_rt['amount'], errors='coerce')
        if 'fraud_probability' in df_rt.columns:
            df_rt['fraud_probability'] = pd.to_numeric(df_rt['fraud_probability'], errors='coerce')

        # Handle fraud_prediction - check if it exists, otherwise use is_fraud
        if 'fraud_prediction' in df_rt.columns:
            df_rt['fraud_prediction'] = pd.to_numeric(df_rt['fraud_prediction'], errors='coerce').fillna(0).astype(int)
        elif 'is_fraud' in df_rt.columns:
            df_rt['fraud_prediction'] = pd.to_numeric(df_rt['is_fraud'], errors='coerce').fillna(0).astype(int)
        else:
            df_rt['fraud_prediction'] = 0
        # Handle timestamps - specify format to avoid warnings
        if 'timestamp' in df_rt.columns:
            df_rt['timestamp'] = pd.to_datetime(df_rt['timestamp'], format='%Y-%m-%d %H:%M:%S', errors='coerce')
        if 'processed_time' in df_rt.columns:
            df_rt['processed_time'] = pd.to_datetime(df_rt['processed_time'], format='%Y-%m-%d %H:%M:%S', errors='coerce')
        else:
            if 'timestamp' in df_rt.columns:
                df_rt['processed_time'] = df_rt['timestamp']
            else:
                df_rt['processed_time'] = pd.Timestamp.now()

        # Extract transaction_type and location if not present
        if 'transaction_type' not in df_rt.columns:
            df_rt['transaction_type'] = df_rt.apply(extract_transaction_type, axis=1)
        if 'location' not in df_rt.columns:
            df_rt['location'] = df_rt.apply(extract_location, axis=1)

        # Remove duplicates
        df_rt.drop_duplicates(subset="transaction_id", inplace=True)
        df_rt["source"] = "Real-Time"

        # Sort by processed_time, handling NaT values
        if 'processed_time' in df_rt.columns:
            df_rt = df_rt.sort_values(by="processed_time", ascending=False, na_position='last')

        return df_rt
    except Exception as e:
        # Don't show error in function - let caller handle it
        return pd.DataFrame()
```

Fig 13: load_realtime_data() function

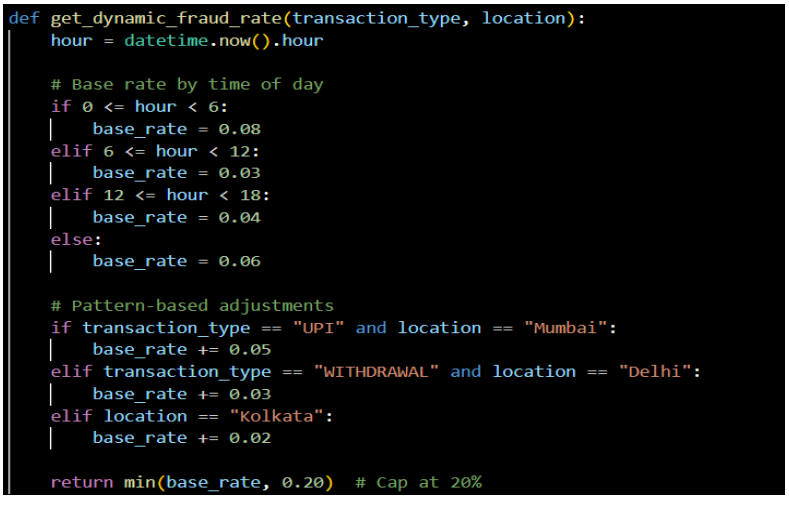
This function loads and prepares real-time transaction data in a safe and reliable way. It first checks whether the real-time data file exists and contains records, returning an empty dataset if it doesn't. The function then cleans and standardizes important fields by converting transaction

IDs to strings, amounts and probabilities to numeric values, and timestamps to a consistent date-time format. It also standardizes fraud information, ensuring there is a single fraud indicator by using fraud_prediction when available or falling back to is_fraud if needed. Missing transaction details such as transaction type or location are automatically derived, duplicates are removed, and each record is labeled as “Real-Time” for clear identification. Overall, it returns a clean, consistent, and up-to-date dataset ready for real-time analysis or fraud monitoring.

```python
def load_historical_data():
    """Load historical data"""
    if not os.path.exists(HISTORICAL_FILE):
        return pd.DataFrame()

    try:
        df_hist = pd.read_csv(HISTORICAL_FILE, low_memory=False)
        if df_hist.empty:
            return pd.DataFrame()

        df_hist['amount'] = pd.to_numeric(df_hist['amount'], errors='coerce')
        # Specify format to avoid warnings and improve performance
        df_hist['timestamp'] = pd.to_datetime(df_hist['timestamp'], format='%Y-%m-%d %H:%M:%S', errors='coerce')
        if 'processed_time' in df_hist.columns:
            df_hist['processed_time'] = pd.to_datetime(df_hist['processed_time'], format='%Y-%m-%d %H:%M:%S', errors='coerce')
        else:
            df_hist['processed_time'] = df_hist['timestamp']

        # Handle fraud prediction
        if 'fraud_prediction' in df_hist.columns:
            df_hist["fraud_prediction"] = pd.to_numeric(df_hist["fraud_prediction"], errors='coerce').fillna(0).astype(int)
        elif 'is_fraud' in df_hist.columns:
            df_hist["fraud_prediction"] = pd.to_numeric(df_hist["is_fraud"], errors='coerce').fillna(0).astype(int)
        else:
            df_hist["fraud_prediction"] = 0

        df_hist["source"] = "Historical"
        return df_hist
    except Exception as e:
        return pd.DataFrame()
```

Fig 14: load_historical_data() function

This function loads and prepares historical transaction data in a safe and reliable way. It first checks if the file exists and contains data, returning an empty dataset if it doesn’t. The function then cleans key fields by converting amounts to numeric values and timestamps to a consistent date-time format. It also standardizes fraud information, ensuring there is a single fraud indicator regardless of whether the source uses is_fraud or fraud_prediction. Each record is labeled as “Historical” for easy identification. Overall, it returns a clean, consistent dataset ready to be combined with new transactions for analysis or fraud monitoring.



```python
def get_dynamic_fraud_rate(transaction_type, location):
    hour = datetime.now().hour

    # Base rate by time of day
    if 0 <= hour < 6:
        base_rate = 0.08
    elif 6 <= hour < 12:
        base_rate = 0.03
    elif 12 <= hour < 18:
        base_rate = 0.04
    else:
        base_rate = 0.06

    # Pattern-based adjustments
    if transaction_type == "UPI" and location == "Mumbai":
        base_rate += 0.05
    elif transaction_type == "WITHDRAWAL" and location == "Delhi":
        base_rate += 0.03
    elif location == "Kolkata":
        base_rate += 0.02

    return min(base_rate, 0.20)  # Cap at 20%
```

Fig 15: get_dynamic_fraud_rate() function

This function computes a fraud risk score based on the time of day, transaction type, and location. Risk is higher during late-night hours and increases further for known risky patterns like UPI in Mumbai, withdrawals in Delhi, or transactions from Kolkata. To keep results realistic, the risk is capped at 20% and returned as a single score that can be used for fraud detection and monitoring.

```
def score_new_transactions():
    if not os.path.exists(INPUT_FILE):
        print("⚠️ Input file not found.")
        return

    df_stream = pd.read_csv(INPUT_FILE)
    if df_stream.empty:
        print("⏳ No transactions found.")
        return

    df_stream["transaction_id"] = df_stream["transaction_id"].astype(str)

    # Debug: Check for fraud flags in input
    if 'is_fraud' in df_stream.columns:
        fraud_count_input = df_stream['is_fraud'].sum()
        print(f"🔍 Input file has {fraud_count_input} fraud transactions (is_fraud column)")
    elif 'fraud_prediction' in df_stream.columns:
        fraud_count_input = df_stream['fraud_prediction'].sum()
        print(f"🔍 Input file has {fraud_count_input} fraud transactions (fraud_prediction column)")

    if os.path.exists(OUTPUT_FILE):
        df_scored = pd.read_csv(OUTPUT_FILE)
        if "transaction_id" in df_scored.columns:
            df_scored["transaction_id"] = df_scored["transaction_id"].astype(str)
            scored_ids = set(df_scored["transaction_id"])
        else:
            print("⚠️ 'transaction_id' column missing. Resetting scored file.")
            df_scored = pd.DataFrame()
            scored_ids = set()
    else:
        df_scored = pd.DataFrame()
        scored_ids = set()

    df_new = df_stream[~df_stream["transaction_id"].isin(scored_ids)].copy()
    if df_new.empty:
        print("⏳ No new transactions to score.")
        return

    print(f"🆕 New transactions to score: {len(df_new)}")

    # Debug: Check for fraud flags in new transactions
    if 'is_fraud' in df_new.columns:
        fraud_count_new = df_new['is_fraud'].sum()
        print(f"🔍 New transactions include {fraud_count_new} fraud transactions (is_fraud column)")
    elif 'fraud_prediction' in df_new.columns:
        fraud_count_new = df_new['fraud_prediction'].sum()
        print(f"🔍 New transactions include {fraud_count_new} fraud transactions (fraud_prediction column)")

    required_cols = ["transaction_id", "timestamp", "sender_account", "receiver_account", "amount", "transaction_type", "location"]
    # Don't drop rows based on is_fraud or fraud_probability - they're optional
    df_new = df_new.dropna(subset=required_cols)

    # Preserve simulator fraud flags if they exist
    simulator_fraud_backup = None
    simulator_prob_backup = None
    if 'is_fraud' in df_new.columns:
        simulator_fraud_backup = df_new['is_fraud'].copy()
    elif 'fraud_prediction' in df_new.columns:
        simulator_fraud_backup = df_new['fraud_prediction'].copy()
    if 'fraud_probability' in df_new.columns:
```

```
    simulator_prob_backup = df_new['fraud_probability'].copy()

df_new['amount'] = pd.to_numeric(df_new['amount'], errors='coerce')
before = len(df_new)
df_new = df_new.dropna(subset=['amount'])
df_new = df_new[df_new['amount'] < 1e6]
after = len(df_new)
print(f"🧹 Dropped {before - after} rows with invalid or extreme amount values.")

if df_new.empty:
    print("⚠️ No valid transactions to score after cleaning.")
    return

df_new['timestamp'] = pd.to_datetime(df_new['timestamp'], errors='coerce')
df_new['hour'] = df_new['timestamp'].dt.hour
df_new['day_of_week'] = df_new['timestamp'].dt.dayofweek
df_new['amount_log'] = np.log1p(df_new['amount'])

df_new["transaction_type_raw"] = df_new["transaction_type"]
df_new["location_raw"] = df_new["location"]

df_model = pd.get_dummies(df_new, columns=['transaction_type', 'location'], drop_first=True)

model_features = model.get_booster().feature_names
for col in model_features:
    if col not in df_model.columns:
        df_model[col] = 0
df_model = df_model[model_features]
```

Fig 16: score_new_transactions() function

This function scores new banking transactions in a production-like fraud monitoring flow. It checks for valid input, avoids reprocessing already scored transactions, and keeps only new data. The function then cleans and validates records, removes invalid amounts, and preserves any existing fraud labels. It also creates key fraud-detection features (time, day, amount scale) and converts transaction details into a model-ready format. Overall, it turns raw transaction data into a clean, structured dataset ready for fraud prediction, dashboards, and alerts.

Functional Significance in BFSI

This module implementation aligns with BFSI operational needs, where fraud prevention, transparency, and compliance are critical.

Through real-time visualization, the system allows institutions to:

- Detect anomalies within seconds of occurrence.
- Support fraud teams with actionable insights.
- Maintain regulatory compliance by continuously tracking and auditing digital transactions.

The dashboard bridges the gap between data analytics and user interpretation, offering both a technical and business-focused view of transaction monitoring.

Future Enhancements

1. Integrate live API data from payment systems.
2. Host dashboard on a cloud platform like AWS.

3. Add user authentication for secure access.
4. Enable email/SMS alerts for detected frauds.
5. Add more AI-based anomaly detection features.

Conclusion

Our project "Real-Time Transaction Monitoring Dashboard" successfully demonstrates how digital banking operations can be visualized and monitored efficiently. By simulating transaction data, processing it, and displaying insights on a dashboard, we were able to show the real-time behavior of a payment system in a safe and controlled environment.

The project also shows the potential of combining data analytics and machine learning in the BFSI domain. With minor improvements and API integration, this system could easily be extended into a real-world model used by digital banking platforms for live transaction monitoring, compliance reporting, and fraud prevention.

Overall, this project gave us strong exposure to technical, analytical, and business perspectives in the BFSI domain and taught us how powerful data visualization can be in making banking systems transparent and intelligent.

References

[1] M. S. Islam, M. Y. Ahmad, I. Zerine, Y. A. Biswas and M. Islam, "Real-Time Data Stream Analytics and Artificial Intelligence for Enhanced Fraud Detection and Transaction Monitoring in Banking Security," American Journal of Innovation in Science and Engineering, vol. 4, no. 3, 2025.

[2] M. Lu, Z. Han, S. X. Rao, Z. Zhang, Y. Zhao, Y. Shan, R. Raghunathan, C. Zhang and J. Jiang, "BRIGHT – Graph Neural Networks in Real-Time Fraud Detection," arXiv preprint arXiv:2205.13084, 2022.

[3] T. Awosika, R. Shukla and B. Pranggono, "Transparency and Privacy: The Role of Explainable AI and Federated Learning in Financial Fraud Detection," arXiv preprint arXiv:2312.13334, 2023.

[4] A. Alwadain, R. F. Ali and A. Muneer, "Estimating Financial Fraud through Transaction-Level Features and Machine Learning," Mathematics, MDPI, vol. 11, no. 5, 2023.

[5] S. Islam, G. Raj Gupta, A. Chakraborty, et al., “Detecting Fraudulent Transactions for Different Patterns in Financial Networks Using Layer Weighted GCN,” Human-Centric Intelligent Systems, Springer, 2025.

[6] M. Pudi, “Real-Time Fraud Detection: A Banking Industry Case Study,” International Journal of Computer Engineering & Technology (IJCET), 2025.

[7] M. I. Ayub, B. Bhattacharjee, P. Akter, et al., “Deep Learning for Real-Time Fraud Detection: Enhancing Credit Card Security in Banking Systems,” The American Journal of Engineering and Technology, 2025.

[8] R. Tehseen, H. Shahid, A. Mustaqeem, M. F. Khan, U. Omer and R. Javaid, “A Framework for Fraud Detection in Banking Transactions Using Machine Learning and Federated Learning,” International Journal of Innovations in Science & Technology (IJIST), 2025.

[9] V. P. Varshini, A. Vishnu Reddy, G. Akhilesh Reddy and K. Radha, “Fraud Detection in Banking Transactions Using Machine Learning,” in Proc. 4th International Conference on Recent Innovations in Computer Engineering & Information Technology (ICRICEIT-2024), 2024.

[10] T. Awad, M. Hammad, A. El-Said and A. El-Fergany, “Real-Time Fraud Detection on Financial Transactions Using Adaptive Ensemble Learning,” Journal of Big Data, Springer, 2024.

[11] L. Zhang, Y. Li and J. Zhao, “An Online Machine Learning Framework for Real-Time Transaction Fraud Detection,” IEEE Access, vol. 11, 2023.

[12] J. Y. Kim, J. H. Lee and S. W. Kim, “Graph-Based Financial Fraud Detection Considering Temporal Transaction Patterns,” in Proc. IEEE International Conference on Data Mining (ICDM), 2023.

[13] K.-Y. Yap, S. Chakraborty and J.-K. Peh, “Real-Time Credit Card Fraud Detection Using Federated Learning and Gradient Boosting,” IEEE Transactions on Big Data, 2025.

[14] J. Huang, Y. Cai, Z. Xu and Z. Wu, “A Real-Time Stream Processing Approach for Bank Fraud Detection at Scale,” in Proc. IEEE Big Data Conference, 2023.

[15] Y. Yao, B. Shen, X. Wu and Y. Zhang, “Applying Online Learning Model for Real-Time

Transaction Fraud Detection in Fintech," IEEE Transactions on Knowledge and Data Engineering, 2024.

[16] R. K. Gupta, A. Verma and S. Singh, "Real-Time Log-Level Transaction Monitoring in Financial Systems: Challenges and Solutions," International Journal of Innovative Research in Computer Science & Technology, 2025.

[17] S. Murthy, R. Rathod and M. K. Bhakta, "Fraud Transaction Prediction in Digital Payments: A Real-Time Big Data Framework," IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), 2024.

[18] P. G. Arora and V. Kumar, "Behavioral Analysis Based Real-Time Fraud Detection System in Banking using Hybrid Learning," SN Computer Science, 2025.

[19] Z. Huang, Y. Che and T. Gao, "Real-Time Financial Fraud Detection Based on Streaming Rules and Adaptive Thresholds," Journal of Computer Science & Technology, 2024.

[20] F. P. Silva, E. Castillo, "Data-Driven Real-Time Fraud Detection Using Sliding Window and Concept Drift Techniques," Information Sciences, Elsevier, 2023.

[21] L. Yang, S. Wang and J. Chen, "Real-Time Anomaly Detection System for FinTech Payments using Ensemble Learning and Stream Processing," IEEE Access, vol. 12, 2024.

[22] D. Sharma, A. K. Tiwari and S. K. Malhotra, "Machine Learning Based Fraud Detection for Real-Time Payment Systems," Proc. 6th International Conference on Machine Learning and Data Science (MLDS-6), 2024.

[23] B. Bhushan and S. Kulkarni, "Real-Time Financial Fraud Detection using Streaming Big Data Architecture and Big-ML," International Journal of Emerging Technologies and Innovative Research (JETIR), 2023