1. Introduction and Business Problem

Financial institutions face persistent challenges in identifying fraudulent transactions accurately and efficiently. As digital payment volumes continue to rise, fraud detection systems must evaluate increasingly large datasets in which true fraud represents only a small fraction of overall activity. This imbalance creates operational strain for fraud analysts, who must navigate high-volume queues where most alerts are legitimate. The manual review process is time-consuming, inconsistent, and difficult to scale, often resulting in missed fraud cases, unnecessary operational cost, and limited visibility into performance outcomes.

The central business problem addressed by this project is the lack of a unified, data-driven system that integrates fraud-risk modeling with structured reporting and financial evaluation. Existing workflows typically involve isolated tools that do not communicate well with one another: analysts receive alerts without clear prioritization, managers lack standardized metrics for evaluating accuracy and workload, and leadership has little insight into the financial implications of true positives, false positives, or missed fraud. These gaps hinder organizational decision-making and obscure opportunities to improve both operational efficiency and fraud prevention effectiveness.

This project aims to close these gaps through the development of a complete end-to-end fraud detection system that integrates machine learning, database design, and business intelligence reporting into a cohesive framework. Instead of building a predictive model in isolation, the system was designed to mirror real-world fraud operations by incorporating risk scoring, batch-level evaluation, financial calculations, and a dashboard interface for daily use. The solution provides a structured method for identifying high-risk transactions, quantifying financial outcomes, and supporting consistent review practices across the fraud team.

The need for such a system is particularly acute because fraud detection is not only a technical problem but also an operational one. The value of a model depends not only on its statistical performance but on how well its outputs integrate into investigative workflows. By addressing both the modeling and reporting layers, this project offers a practical solution that enhances analytical visibility and improves the overall fraud review process. The integrated design enables analysts to prioritize alerts more intelligently, helps managers understand trends and performance, and provides leadership with the financial insight needed to evaluate the impact of fraud prevention strategies.

Overall, this project demonstrates how combining predictive analytics with business intelligence tools can create a more transparent, scalable, and financially meaningful approach to fraud detection. The following sections outline the system's structure, data preparation, modeling methodology, architecture, and results, culminating in a robust reporting interface that aligns with real operational needs.

2. System Overview

This project delivers a comprehensive fraud detection system that integrates multiple components into a unified analytical pipeline. The system consists of three primary layers: the machine-learning modeling layer built in Python, the storage and integration layer implemented using SQLite, and the reporting layer developed in Power BI. Each component serves a distinct role but is intentionally designed to work cohesively to support end-to-end fraud detection and reporting.

At the core of the system is a Logistic Regression model trained on the highly imbalanced credit card fraud dataset. The model produces probability scores for each transaction, which are then converted into binary predictions and risk levels. In addition to generating classification outputs, the system calculates financial metrics associated with each prediction type, such as avoided fraud losses and review costs. These outputs ensure that the evaluation of the model accounts not only for accuracy but also for financial impact, reflecting how fraud detection systems are assessed in practice.

To organize and store the model's outputs, the system uses a SQLite database. Python exports several structured tables, including test predictions, batch-level financial summaries, and threshold evaluation metrics into SQLite. This database functions as the single source of informatio for downstream reporting and enables a consistent interface for Power BI. The database design follows clear, analytics-oriented schema principles, with surrogate keys, normalized numeric fields, and separate fact and dimension tables. This structure supports both transparency and scalability.

Power BI serves as the main interface for analysts and stakeholders. It connects directly to the SQLite database using an ODBC connector and uses DAX measures to recompute performance metrics such as recall, precision, and net savings. The dashboard presents information in an accessible and operationally relevant format, including key performance indicators, prediction-type breakdowns, risk-level distributions, and transaction-level investigative tools. Analysts can interact with the data through slicers and drill-down views, allowing for flexible review based on prediction type, risk level, and actual fraud outcomes.

The system's architecture emphasizes modularity: each layer can be updated independently without disrupting the others. This design mirrors enterprise-level fraud systems, where models, storage layers, and dashboards evolve separately but remain integrated through well-defined interfaces. The modular approach ensures that the system is maintainable, scalable, and adaptable to future enhancements such as different models, modified thresholds, or additional reporting pages.

In summary, the project delivers a complete fraud detection framework that integrates modeling, data management, and reporting into a coherent analytical system. The next section discusses the dataset and preprocessing steps that prepare the foundation for the model and reporting workflow.

3. Dataset Description and Preparation

The foundation of this system is the publicly available **Credit Card Fraud Detection dataset**, which contains **284,807** real credit card transactions collected over two days. This dataset is widely used in fraud research because it closely reflects key operational challenges in the financial industry: high-volume transactional data, behavioral features extracted through confidentiality-preserving transformations, and an extremely low fraud rate.

3.1 Dataset Structure

The dataset includes:

- **Time**: Seconds elapsed since the first recorded transaction in the dataset.
- **Amount**: Monetary value of the transaction.
- **V1–V28**: Principal Component Analysis (PCA) transformed features based on original transactional attributes.
- **Class**: Fraud label (1 = fraud, 0 = legitimate).

The PCA-transformed variables prevent direct reverse-engineering of customer-sensitive information, which makes the dataset suitable for open research while retaining important behavioral signal. These components capture nuanced transaction characteristics that can be used to identify patterns of fraud without exposing raw financial features.

The target variable is highly imbalanced:

- 492 fraud cases
- 284,315 legitimate cases

This results in a fraud rate of **0.17%**, which mirrors real-world fraud environments where many transactions are legitimate. This imbalance creates significant modeling challenges because standard classifiers tend to favor the majority class.

3.2 Data Validation and Initial Checks

Before conducting any modeling steps, the dataset was validated to ensure:

- No missing values
- All numeric columns
- No duplicate rows
- No formatting inconsistencies

Because the dataset contains only numeric variables, preprocessing largely focused on scaling, stratification, and imbalance mitigation rather than data type conversion or imputation.

3.3 Transaction Identifier Creation

One of the limitations of the original dataset is the absence of a natural primary key. To ensure consistency across all components of the pipeline, Python, SQLite, and Power BI, a unique identifier was created:

```
df["Transaction_ID"] = np.arange(len(df))
```

This variable serves as a stable surrogate key, enabling:

- Tracking predictions across systems
- Joining tables within SQLite
- Linking model outputs to financial calculations
- Displaying transaction-level detail in Power BI

This step enhances traceability and mirrors best practices in data warehousing environments.

3.4 Feature Scaling Strategy

Due to the nature of the PCA features and the Amount variable, the dataset contains a mixture of differently scaled values. To avoid distorting model performance, two feature matrices were created:

- **Scaled features (StandardScaler)** for algorithms requiring normalization such as Logistic Regression.
- **Unscaled features** for tree-based models such as Random Forest, XGBoost, and LightGBM.

Maintaining parallel feature sets allowed each model to be trained under conditions that aligned with its assumptions, improving the fairness of comparisons and ensuring stability during training.

3.5 Train–Test Split

A stratified 70/30 split was applied to preserve the original ratio of fraud to non-fraud cases in both sets. The training set was used for model fitting and oversampling, while the test set remained untouched for unbiased evaluation.

3.6 Addressing Class Imbalance with SMOTE

Fraud detection requires models to identify rare events. Without intervention, a classifier could achieve 99.8% accuracy simply by predicting “not fraud” for every transaction. To ensure the model had sufficient exposure to minority-class examples, the training set underwent Synthetic Minority Oversampling Technique (SMOTE):

```
X_train_res, y_train_res = smote.fit_resample(X_train_scaled, y_train)
```

SMOTE generates new synthetic fraud examples by interpolating between existing minority-class points. This increases representation of fraudulent behavior without altering the test set distribution. The decision to apply SMOTE exclusively to the training set preserves a realistic evaluation environment and ensures that test results generalize to real-world conditions.

3.7 Operational Batch Creation

To approximate real production workflows, transactions in the test set were grouped into operational batches. These batches represent sequential windows of activity and allow for:

- Batch-level financial evaluation
- Identification of periods where fraud is concentrated
- Stable performance tracking across time-based segments

Batch labeling became an important component in both Python's financial evaluation tables and Power BI's filtering capabilities.

4. User Design and Functional Requirements

The design of this fraud detection system was shaped by the needs of its core users and the workflows that typically exist within fraud operations teams. Although the technical model and database form the foundation of the system, the success of any fraud-scoring tool ultimately depends on whether analysts, supervisors, and data teams can use it easily and confidently. For this reason, the system was deliberately developed with an emphasis on clarity, interpretability, and operational alignment rather than purely algorithmic complexity.

4.1 Primary User Groups

The system is intended to support three main user groups, each with distinct responsibilities and expectations.

Fraud Analysts.
Analysts are responsible for reviewing large volumes of transactions to determine which ones require further investigation. Their daily work is time-sensitive and often high-pressure, especially during periods of increased fraud activity. To support this group, the system needed to provide a clear way to identify high-risk cases, surface the most relevant information about each transaction, and minimize unnecessary steps in the review process. Analysts benefit from having probability scores, risk levels, and prediction types available in an organized interface so they can move quickly from broad trends to individual decisions.

Fraud Team Leads and Supervisors.
Supervisors oversee operational performance and need to balance efficiency with fraud-loss prevention. Their focus is typically on understanding how the model behaves overall rather than investigating individual transactions. They require access to summary indicators such as total alerts, actual fraud counts, recall, and estimated financial savings. These metrics help determine

whether the review process is meeting organizational goals and whether staffing or threshold adjustments might be necessary. The dashboard therefore needed to include transparent, easy-to-interpret KPIs.

Data Science and Analytics Teams.
This group ensures long-term reliability and correctness of the model. They monitor data quality, confirm that predictions remain stable, and evaluate how model performance evolves over time. They also rely heavily on transaction-level details to diagnose unexpected patterns. Fields such as prediction type (TP, FP, TN, FN), risk tier, probability, and batch assignment give them the traceability required to validate model assumptions and monitor drift. The system therefore incorporates consistent field definitions and maintains a data structure that supports analytical evaluation beyond standard reporting.

4.2 Functional Requirements

To meet the needs of these user groups, the system had to deliver a set of key capabilities that would make the model's outputs accessible, interpretable, and actionable.

Model Output Requirements.
The core requirement was to generate clear, interpretable model outputs that directly support decision-making. Each transaction needed a binary prediction indicating whether it should be flagged as potential fraud, along with a calibrated probability score that communicates uncertainty. To make the results more intuitive for analysts, the system also assigns each transaction a risk level, high, medium, or low based on probability thresholds. Finally, every transaction is labeled according to its prediction type (true positive, false positive, true negative, or false negative), which is essential for performance evaluation in both Python and Power BI.

Operational Requirements.
Since analysts regularly sift through thousands of transactions, the system needed features that simplify prioritization and streamline case review. This includes searchable and sortable transaction tables, the ability to filter by risk level or prediction type, and direct access to the probability score influencing each alert. Additionally, transactions should be grouped into operational batches so analysts and managers can monitor behavior across segments of activity. These design choices reinforce real-world workflows in fraud teams, where context, volume, and timing all influence review strategies.

Reporting Requirements.
To ensure supervisors and analytics teams can interpret the model's performance at a glance, the system must provide high-level KPIs such as total transactions, actual fraud count, predicted frauds, recall, avoided fraud loss, and net savings. Visual summaries including distribution charts, risk-level breakdowns, and confusion-matrix-based outcome charts help translate model behavior into insights quickly. The reporting layer also needs to support filtering by key dimensions such as actual class, prediction type, and risk level. This structured approach allows analysts to move from overall results to transaction details in a logical, intuitive sequence.

4.3 Non-Functional Requirements

In addition to functional goals, the system had to satisfy several non-functional requirements to ensure that it would be practical and sustainable for long-term use.

Usability.
The dashboard interface must be intuitive even for users with limited technical backgrounds. Visuals should be consistent in style, labels must be clear, and interactions such as slicers and filters should behave predictably. The goal is to help analysts access insights quickly without requiring extensive training.

Performance.
Because fraud analysis is time-sensitive, the system must load quickly and handle large datasets efficiently. Power BI visuals and Python pipelines should not introduce delays that interrupt workflow. SQLite was chosen specifically to enable fast querying and lightweight storage without requiring enterprise infrastructure.

Scalability.
The system needed to allow for model updates without forcing structural changes to the dashboard or database. This ensures the design can adapt to new fraud patterns, alternate model architectures, or additional features without extensive rework.

Traceability.
Every prediction must be easily traceable back to the original transaction, model score, and financial impact. Creating a unique Transaction_ID and maintaining consistent naming conventions across Python, SQLite, and Power BI ensures that every component of the system can be audited and validated.

Maintainability.
Both the SQLite schema and the DAX measures in Power BI were designed to be clean, structured, and extendable. Analysts and data teams should be able to introduce new metrics, adjust thresholds, or incorporate additional datasets without having to rebuild the workflow from scratch.

5. Technical Specifications and System Architecture

This section describes the full technical foundation of the fraud detection system, outlining how the modeling pipeline, database infrastructure, and reporting environment work together as a unified analytical workflow. The goal of this architecture is to produce a fraud-scoring system that is not only accurate but also transparent, maintainable, and aligned with real operational needs. For that reason, the system follows a modular design: each layer performs a distinct role, yet all layers interface cleanly through consistent schemas and identifiers. This structure reflects industry practice, where model development, data engineering, and business intelligence must remain interconnected but independently manageable.

5.1 Architectural Overview

The system consists of three major layers:

1. **Python Modeling Layer** – performs all data processing, feature engineering, model training, prediction generation, threshold evaluation, and financial calculation.
2. **SQLite Storage Layer** – stores the standardized outputs from Python in a structured, queryable format for downstream reporting.
3. **Power BI Reporting Layer** – transforms the stored predictions and financial metrics into an interactive dashboard that supports analyst workflows.

Although simple in appearance, this three-layer architecture mirrors how modern fraud detection systems are deployed:

Computation and model training occurs in python, results are later store in a relational database accessible to other tools and end users interact with a visual interface build on these outputs.

The intentional decoupling of layers ensures that the model can be retrained, the database can be updated, or the dashboard refreshed without breaking the workflow. This greatly increases long-term scalability.

5.2 Python Modeling Layer

Python serves as the computational core of the system. Nearly all analytical processes ranging from preprocessing to model evaluation is executed here. The following subsections describe each component in detail.

5.2.1 Data Loading and Identifier Assignment

The Credit Card Fraud dataset contains 284,807 transactions with 30 anonymized PCA-based features, along with the transaction amount and a binary target variable (“Class”). To ensure full traceability throughout the system, the first technical step involves assigning each row a **Transaction_ID**, which acts as a surrogate key:

```
df["Transaction_ID"] = np.arange(1, len(df) + 1)
```

This identifier becomes the backbone of the entire architecture. It:

- allows Python outputs to be joined cleanly with SQLite tables,
- ensures Power BI can link predictions back to individual transactions, and

- supports debugging, auditing, and transparency which are important requirements in fraud analytics.

Although simple, this design choice guarantees that every stage of the workflow references the same unique entity.

5.2.2Train–Test Split Under Severe Class Imbalance

Fraud datasets are inherently imbalanced, and this one is no exception: only **0.17%** of transactions are fraudulent. This extreme imbalance must be handled carefully, as naïve classification approaches tend to produce overly optimistic accuracy while failing to detect actual fraud.

To maintain a realistic evaluation environment, the system uses a stratified train–test split as stratification preserves the fraud distribution in both splits and the test set remains untouched by resampling.

This approach ensures that the modeling pipeline does not artificially inflate performance by oversampling fraud in the test set.

5.2.3 Feature Preparation: Scaled and Unscaled Inputs

Because the PCA-based features vary significantly in magnitude, and because different algorithms have different sensitivities to scaling, the pipeline creates two parallel feature matrices:

Scaled Feature Matrix

Used by algorithms that assume normalized input, including:

- Logistic Regression
- Gradient Boosting

Unscaled Feature Matrix

Used by algorithms that are robust to feature magnitude, including:

- Random Forest
- XGBoost
- LightGBM

Maintaining both matrices ensures each model is trained in conditions that reflect its underlying mathematical assumptions. This results in more valid comparisons between algorithms and prevents preprocessing from skewing results.

5.2.4 Addressing Class Imbalance with SMOTE

Fraud detection models perform poorly when minority-class examples are scarce, because the model has too few fraud cases from which to learn meaningful patterns. To mitigate this, the system applies SMOTE (Synthetic Minority Oversampling Technique) exclusively to the training set:

```
X_train_res, y_train_res = smote.fit_resample(X_train_scaled, y_train)
```

SMOTE artificially generates new minority-class samples by interpolating existing ones, increasing the number of fraudulent observations and allowing the model to learn a broader representation of fraud behaviors.

The test set is never resampled, as modifying the fraud proportion in evaluation would misrepresent real-world performance.

5.2.5 Model Selection Framework and Cross-Validation

The system evaluates five different models:

- Logistic Regression
- Random Forest
- Gradient Boosting
- XGBoost
- LightGBM

Each algorithm is trained and evaluated using **5-fold stratified cross-validation**, providing performance metrics including:

- precision,
- recall,
- F1 score, and
- ROC-AUC.

Cross-validation ensures that model performance is not dependent on a single train–test split. Instead, it provides a robust estimate across multiple randomized partitions. This is critical for fraud detection, where model stability is as important as raw predictive accuracy.

Although ensemble models occasionally performed competitively, Logistic Regression consistently demonstrated strong recall, reliable behavior across folds, and interpretable probability outputs. Its simplicity and consistency made it the most suitable candidate for deployment in a business-facing analytical system.

5.2.6 Prediction Generation, Threshold Optimization, and Financial Calculations

After selecting the champion model, Python generates probability scores for each test transaction. Rather than default to a 0.5 threshold, the pipeline evaluates performance across thresholds from 0.1 to 0.9. This allows the system to explore the trade-off between identifying more fraud (higher recall) and minimizing unnecessary reviews (lower false positives).

Threshold performance is evaluated not only through precision and recall but also through financial impact, including:

- **Fraud Loss Avoided** (true positives),
- **False Positive Cost** (review overhead),
- **Missed Fraud Loss** (false negatives), and
- **Net Savings**.

This financial framing acknowledges that fraud detection systems are evaluated not only by accuracy but by economic outcomes. By evaluating thresholds under financial as well as statistical criteria, the system reflects realistic organizational priorities.

5.2.7 Batch-Level Analysis for Operational Alignment

In real fraud review environments, analysts work with incoming transactions in time-segmented windows. Fraud also tends to cluster within specific periods. To reflect this operational reality, the test data is grouped into batches of 5,000 transactions.

Python computes batch-level summaries including:

- number of fraud cases,
- number of alerts,
- breakdown of prediction types,
- total fraud amount,
- avoided fraud loss, and
- net savings.

Batch-level monitoring offers operational insight into how the model behaves over time, helping analysts and supervisors identify whether performance trends fluctuate under certain transaction conditions or periods.

5.2.8 Exporting Outputs to SQLite

After generating predictions, probability scores, prediction types, risk levels, and financial metrics, Python exports all outputs to a SQLite database. Two key tables are produced:

model_predictions

A transaction-level table containing:

- Transaction_ID
- Actual_Class
- Prediction
- Probability
- Risk_Level
- Prediction_Type
- Amount
- Batch_ID

financial_metrics

A financial table containing:

- Fraud_Loss_Avoided
- False_Positive_Cost
- Net_Savings
- Batch_ID
- and other financial aggregations

These tables form the sole data source for Power BI, which ensures consistency between the modeling and reporting layers.

5.3 SQLite Storage Layer

SQLite functions as the central integration point between Python and Power BI. It is lightweight, serverless, and ideal for analytical workflows where fast access and minimal overhead are required. Because it does not rely on a dedicated database server, SQLite is also portable and easy to refresh , qualities that make it suitable for iterative model development.

The database schema follows an analytics-friendly structure:

- **Fact table:** model_predictions
- **Financial fact table:** financial_metrics
- **Shared dimension:** Transaction_ID

This schema is simple, durable, and easy to extend if additional datasets or new model versions are introduced in the future.

5.4 Power BI Reporting Layer

Power BI acts as the user-facing interface for all stakeholders. It reads directly from the SQLite database using an ODBC connection, meaning no manual import steps are required once the system is configured.

A critical design decision was to prepare a streamlined SQLite database containing only the Logistic Regression predictions for Power BI. The original development database contained outputs from all five models, which caused inaccurate KPI calculations when Power BI recomputed confusion matrix metrics. By providing Power BI with only the champion model's outputs, the dashboard produces consistent and transparent performance metrics that accurately reflect the deployed model.

In addition to tabular data, Power BI uses DAX measures to compute performance indicators such as recall, precision, F1 score, fraud avoided, and net savings. These measures allow metrics to update dynamically when users apply filters, making the dashboard interactive and aligned with real investigative workflows.

5.5 DAX Measures Implemented in the Dashboard

The system includes a comprehensive set of DAX measures supporting performance evaluation, financial reporting, and analyst investigation. These measures include:

- **Core count measures** (Total Transactions, Actual Frauds, Predicted Frauds)
- **Confusion matrix measures** (TP, FP, TN, FN)
- **Performance metrics** (Precision, Recall, F1 Score)
- **Financial metrics** (Fraud Cost Avoided, False Positive Cost, Net Savings)
- **Analytical enhancers** (Average Probability, High Risk Count)

These measures form the backbone of the dashboard's KPI cards, distribution charts, and investigative table. Because they are calculated directly from the SQLite tables, they ensure transparency for analysts and supervisors who need to understand how the model behaves under different operational filters.

5.6 Summary of Technical Architecture

The fraud detection system is built on a modular, maintainable architecture where Python handles computational modeling, SQLite stores structured outputs, and Power BI provides interactive reporting.

This structure ensures traceability, operational realism, and extensibility. It reflects how analytics pipelines are deployed in industry settings and provides a foundation that can scale with increased data volume, model complexity, or additional reporting needs.

6. Model Results and Evaluation

The evaluation stage of this project focused on understanding how well the model performs in a realistic fraud-review environment. Instead of only looking at accuracy or one single metric, the analysis combined cross-validation results, test-set behavior, threshold tuning, and financial impact. This allowed the system to be evaluated not just as a machine-learning model but as a practical tool that analysts could rely on.

6.1 Cross-Validation Summary and Champion Model Selection

All five models; Logistic Regression, Random Forest, Gradient Boosting, XGBoost, and LightGBM were trained using stratified cross-validation. This was necessary because the dataset is imbalanced and the goal was to check stability across folds, not just one split.

Across these experiments, Logistic Regression consistently showed the most reliable recall. Even though some tree-based models had higher precision, their recall varied more across folds. In fraud detection, missing cases is significantly more harmful than flagging extra transactions, so recall mattered more for this project.

A simplified comparison is shown below:

Model	**Precision**	**Recall**	**F1 Score**	**ROC-AUC**
Logistic Regression	0.07	**0.918**	0.13	**0.982**
XGBoost	0.887	0.831	0.858	0.981
LightGBM	0.909	0.813	0.859	0.975
Random Forest	0.947	0.770	0.849	0.956
Gradient Boosting	0.788	0.519	0.605	0.672

Even though Logistic Regression has noticeably lower precision, it produced the most consistent recall and the smoothest probability distributions. These qualities made it more suitable for a real analyst workflow, where transparency and consistency matter just as much as performance. For these reasons, Logistic Regression was selected as the champion model.

6.2 Behavior on the Test Set

Once the final model was trained using the SMOTE-enhanced training set, it was evaluated on the untouched test set. Each transaction received:

- a fraud probability score,
- a binary prediction,

A clear separation was visible in the score distribution. Fraud cases tended to appear toward the higher end of the probability scale, while legitimate transactions were more spread out. This is expected in imbalanced datasets but also confirms that the model learned patterns that meaningfully distinguish fraudulent behavior.

6.3 Threshold Tuning

The model was evaluated across thresholds from 0.1 to 0.9 to understand how performance changes when we adjust how “strict” the system is with alerts.

General trends included:

- Lower thresholds: more alerts but lower precision
- Higher thresholds: fewer alerts but more stable recall
- Extremely high thresholds :risk of missing fraud, but Logistic Regression still captured all fraud in this test set even at 0.9

The threshold of 0.90 performed best overall. It balanced recall, F1 score, and financial outcomes while keeping the number of false positives manageable for downstream review.

6.4 Confusion Matrix Outcomes

Using the selected threshold, the model produced the following behaviors:

- Zero false negatives (FN = 0), meaning no fraud was missed
- All fraud cases were detected (TP = all fraud)
- A high number of false positives, which is expected given the cost structure and the emphasis on recall
- Most legitimate transactions correctly classified as non-fraud (TN)

In fraud detection, this trade-off is very normal. False positives increase workload, but the cost of missing a true fraud case is much higher.

6.5 Batch-Level Performance

To make the analysis closer to real operations, the test set was grouped into 5,000-transaction batches. Fraud rarely appears evenly across time, so batch-level analysis helps determine how stable the model is when real transaction volumes fluctuate.

A summary of the first three batches is shown below:

Batch	Fraud Cases	TP	FP	Fraud Avoided	FP Cost	Net Savings
1	26	26	4,795	$5,011	$1,655	**$3,355**
2	6	6	4,862	$452	$2,391	-$1,939
3	17	17	4,798	$2,320	$2,702	-$382

Overall patterns:

- Batches with meaningful fraud activity (like Batch 1) produce positive net savings.
- Batches with almost no fraud naturally result in higher review costs.
- This kind of variability is normal and reflects real-world patterns where fraud “clusters” instead of appearing randomly.

This reinforces why organizations track model performance at both the global level and the batch level.

6.6 Risk Levels and Prediction Types

Along with predictions, the DAX measure assigns:

- Risk Levels (High, Medium, Low)
- Prediction Types (TP, FP, TN, FN)

These labels help different user groups interact with the model: Analysts can quickly filter high-risk cases.Supervisors can understand how much workload comes from false positives.Data teams can track how the model is behaving over time.

These fields also power most of the slicers and visuals in the dashboard.

6.7 Alignment Between Python and Power BI Results

A discrepancy appeared when metrics were calculated inside Power BI. The first SQLite database contained predictions for all five models, and when Power BI recalculated confusion-matrix values using DAX, the results were inconsistent because multiple models' predictions were mixed.

To fix this, a new SQLite database was created containing only:

- Logistic Regression predictions
- Probability scores
- Risk levels
- Prediction types
- Financial metrics

Once Power BI connected to this cleaned-up dataset, all dashboard KPIs aligned perfectly with the champion model's behavior.

Fraud teams design real systems where the modeling environment contains multiple model versions, and the reporting environment only reflects the deployed version.

6.8 Summary of Evaluation

Overall, the model performed well for the intended use case. The key strengths include:

- Strong and consistent recall
- No missed fraud cases on the test set
- Good financial performance in fraud-heavy batches
- Clear probability outputs and risk tiers
- Smooth integration into the reporting system

Even though false positives were high, this is typical in fraud detection when the goal is to avoid missed fraud. The model is interpretable, practical, and aligned with how analysts review cases in real operational environments.

7. Power BI Reporting Layer and Analytical Workflow

Power BI serves as the primary interface for analysts to interact with the predictions, financial metrics, and behavioral patterns generated by the fraud detection model. While Python handles model development and evaluation, and SQLite provides the storage layer, Power BI is responsible for transforming these outputs into a structured, intuitive reporting experience. The dashboard was designed to align with how fraud analysts naturally review cases, beginning with high-level summaries and progressively narrowing into transaction-level detail.

A dedicated SQLite database was created for reporting, containing only the Logistic Regression outputs. This decision was essential because the original modeling database included predictions from all trained models, which caused Power BI to recalculate inconsistent metrics. The final reporting database contains two tables, **model_predictions** and **financial_metrics** that serve as the foundation of all visuals, measures, and slicers in the dashboard.

7.1 Dataset Connection and Table Structure

Power BI connects to SQLite through an ODBC driver. Once connected, the two core tables provide the structured data needed for reporting.

model_predictions includes:
Transaction_ID, Actual_Class, Prediction (0/1), Probability Score, Risk Level, Prediction Type (TP, FP, TN, FN), Amount, and Batch_ID.

This table supplies nearly all fields used for visuals and DAX calculations.

financial_metrics includes:
Fraud_Loss_Avoided, False_Positive_Cost, Net_Savings, False_Positives, Frauds_Detected, and Batch-Level Totals.

These fields support the financial KPI reporting and allow leadership to assess operational cost patterns.

7.2 DAX Measures Used in the Dashboard

To ensure transparency and consistency, all performance and financial metrics displayed in the dashboard are computed using DAX. These measures dynamically update when analysts apply filters, such as selecting a batch, risk level, or prediction type which ensures that every KPI reflects the exact subset of data being reviewed.

Core Count Measures (model_predictions)

Total Transactions

```
Total Transactions =
COUNTROWS(model_predictions)
```

Actual Frauds

```
Actual Frauds =
CALCULATE(
    COUNTROWS(model_predictions),
    model_predictions[Actual_Class] = 1
)
```

Predicted Frauds

```
Predicted Frauds =
CALCULATE(
    COUNTROWS(model_predictions),
    model_predictions[Prediction] = 1
)
```

These measures give analysts a quick understanding of dataset volume, fraud prevalence, and the system's alert rate.

Confusion Matrix Measures

True Positives (TP)

```
True Positives =
CALCULATE(
    COUNTROWS(model_predictions),
    model_predictions[Actual_Class] = 1,
    model_predictions[Prediction] = 1
)
```

False Positives (FP)

```
False Positives =
CALCULATE(
    COUNTROWS(model_predictions),
    model_predictions[Actual_Class] = 0,
    model_predictions[Prediction] = 1
)
```

True Negatives (TN)

```
True Negatives =
CALCULATE(
    COUNTROWS(model_predictions),
    model_predictions[Actual_Class] = 0,
    model_predictions[Prediction] = 0
)
```

False Negatives (FN)

```
False Negatives =
CALCULATE(
    COUNTROWS(model_predictions),
    model_predictions[Actual_Class] = 1,
    model_predictions[Prediction] = 0
)
```

These measures form the backbone of model evaluation inside Power BI. They support recall, precision, and prediction-type breakdowns while allowing analysts to examine performance at different operational levels, such as specific batches or risk tiers.

Performance Metrics

Precision

```
Precision =
DIVIDE(
    [True Positives],
    [True Positives] + [False Positives]
)
```

Recall

```
Recall =
DIVIDE(
    [True Positives],
    [True Positives] + [False Negatives]
)
```

F1 Score

```
F1 Score =
2 * DIVIDE(
    [Precision] * [Recall],
    [Precision] + [Recall]
)
```

Only recall is displayed in the dashboard KPIs, but all three measures were implemented so the system can be extended in the future. These metrics mirror standard fraud-model evaluation practices and help validate that the deployed model behaves consistently over time.

Financial Measures (financial_metrics)

Fraud Cost Avoided

```
Fraud Cost Avoided =
SUM(financial_metrics[Fraud_Loss_Avoided])
```

False Positive Cost

```
False Positive Cost =
SUM(financial_metrics[False_Positive_Cost])
```

Total Net Savings

```
Total Net Savings =
SUM(financial_metrics[Net_Savings])
```

These measures allow analysts and supervisors to quantify the monetary impact of the model. They reflect how much loss was prevented, how much review cost was generated, and whether the model produced a net financial benefit within selected batches or across the entire dataset.

Prediction Type Column
This is not a DAX measure but a calculated column built in Power Query or SQLite. It assigns each transaction one of four labels:

- True Positive
- False Positive
- False Negative
- True Negative

This field is essential for filtering and for the outcome breakdown visuals.

7.3 Slicers and Interactive Filtering

The dashboard includes several slicers that allow users to isolate patterns or investigate specific subsets of the data. The available slicers include:

- **Risk Level** (High, Medium, Low)
- **Prediction Type** (TP, FP, TN, FN)
- **Actual Class** (Fraud / Non-Fraud)

Together, these slicers make the dashboard highly interactive. Analysts can narrow results to high-risk cases, supervisors can evaluate workload by filtering for false positives, and data teams can validate the model's behavior within specific batches. The slicers also enable drill-down workflows where users can explore how performance changes across different segments.

7.4 Dashboard Layout and Analyst Workflow

The dashboard layout follows a logical top-down structure, reflecting how a fraud analyst typically approaches their work.

1. KPI Section
The top section displays core metrics such as total transactions, actual fraud count, predicted fraud alerts, recall, fraud cost avoided, and net savings. These KPIs provide immediate situational awareness and help analysts understand whether the model performed as expected.

2. Model Behavior Visuals
The middle of the dashboard includes:

- A eswdrfchart showing the breakdown of predicted fraud versus non-fraud
- A donut chart showing high, medium, and low risk categories
- A prediction-type distribution visual (TP, FP, TN, FN)

These visuals help users interpret how the model is distributing alerts, whether risk levels are concentrated in certain ranges, and how often the model produces correct or incorrect predictions.

3. Transaction-Level Table
The lower section contains the transaction table, which is the main workspace for investigation. Analysts can sort by probability, filter by risk level or prediction type, and identify the exact transactions requiring attention. Because all DAX measures dynamically update based on sliced data, the KPIs can reflect filtered subsets, allowing analysts to trace how individual transactions contribute to performance and cost.

This structured layout mirrors a real operational workflow: analysts start with the overall picture, narrow down areas of interest, and finally inspect transactions one by one.

7.5 Significance of the Dashboard Structure

The dashboard is not just a visual interface; it is designed to support real operational decision-making. Fraud review requires fast interpretation, prioritization, and clarity. The combination of summary KPIs, risk-distribution visuals, confusion-type breakdowns, and detailed tables ensures that analysts can move seamlessly from high-level insights to transaction-level review without losing context.

The slicers enhance this process by giving users full control over the analysis. Whether the goal is to validate model behavior, focus on high-risk alerts, or understand how costs accumulate in specific batches, the interface adapts to different use cases. This flexibility makes the dashboard suitable for analysts, team leads, and data science staff.

8. System Testing, Validation, and Quality Assurance

After the modeling pipeline and reporting system were fully developed, the final step involved validating that the entire workflow functioned correctly across Python, SQLite, and Power BI. Because this project relied on multiple interconnected components, testing focused not only on technical accuracy within each tool but also on ensuring that the end-to-end pipeline behaved consistently. The goal was to confirm that the model's predictions, financial metrics, and dashboard visuals aligned with real operational expectations and supported the intended analyst workflow.

A key part of this validation process involved checking data integrity in SQLite. The exported tables were reviewed to ensure that all required fields were present, that Transaction_ID values remained unique, and that the batch structure used during financial evaluation was preserved correctly. This step was essential because any inconsistencies at the database level would have directly affected DAX calculations inside Power BI. In addition, the model_predictions table was examined to confirm that predicted labels, probability scores, and risk tiers matched the values produced in Python.

Another important stage involved reconciling metrics calculated in Python with the values displayed in Power BI. Since Power BI recalculates recall, precision, and other metrics using DAX, the dashboard needed to reflect the same logic used during Python evaluation. During this process, a discrepancy was identified: the original database contained predictions from all five trained models. When Power BI recomputed metrics, these mixed predictions led to inflated or unstable values. This issue was resolved by creating a simplified second database containing only the Logistic Regression output. After switching the dashboard to this new database, all KPIs, confusion-matrix results, and financial summaries aligned with Python's evaluation, confirming that the reporting layer reflected the true behavior of the deployed model.

The system was also tested for threshold consistency and edge cases. The model had originally been evaluated in Python across thresholds from 0.10 to 0.90, and these transitions were manually verified to ensure correct behavior. This included checking that prediction types updated accurately and that no unexpected anomalies appeared in the threshold-based outputs. The financial calculations were validated as well by manually reviewing a subset of transactions to confirm that avoided fraud, false positive costs, and net savings were computed correctly.

Finally, the Power BI dashboard itself was tested to ensure that all slicers, filters, and visuals responded immediately and correctly to user interactions. Filtering by risk level, prediction type, or batch ID consistently updated the KPIs, bar charts, and transaction-level tables. This type of interactive testing was important because the dashboard is designed to support investigative workflows, where analysts often need to isolate specific cases or explore subsets of data. The ability to drill down into individual transactions while maintaining accurate recalculated metrics confirmed that the reporting interface behaved reliably.

Overall, the testing and validation process demonstrated that the system is stable, accurate, and aligned with practical fraud-review needs. Each component—modeling, data storage, and reporting—performed as expected both individually and as part of a larger pipeline. This provided confidence that the system could be used in a real operational environment and could be extended in the future without requiring major structural changes.

9. Limitations and Opportunities for Future Enhancements

While the system developed in this project provides a functional and integrated fraud-detection workflow, several limitations emerged throughout the process. These limitations are not weaknesses of the approach but rather natural constraints of working with a public dataset, a simplified architecture, and a model designed for instructional purposes. Recognizing these constraints not only provides transparency but also helps identify areas where future versions of the system could be expanded or refined.

One of the most significant limitations is the nature of the dataset itself. The Credit Card Fraud dataset contains only numerical PCA-transformed features, which limits the types of fraud patterns the model can learn. In real operational environments, analysts also rely on categorical attributes such as merchant category, transaction channel, device information, or customer behavior history. Because these variables were not available, the model may not reflect the full complexity of actual fraud behavior. Future enhancements could incorporate additional features or be adapted to real proprietary datasets that include richer context.

Another limitation relates to the extreme imbalance of the dataset. Even though SMOTE oversampling improved the model's ability to learn from the minority class, oversampling can introduce synthetic patterns that differ from real fraud cases. While Logistic Regression performed consistently during testing, recall values may not remain as stable when applied to completely new datasets. A future version of the system could explore alternative imbalance-handling techniques, such as anomaly detection, cost-sensitive learning, or ensemble methods specifically optimized for rare-event behavior.

From an architectural perspective, the system uses SQLite as the storage layer, which is suitable for prototyping but not fully optimized for enterprise environments. A production deployment would require a more robust database such as SQL Server, Snowflake, or a cloud-managed platform. These systems support user authentication, faster query performance, and higher storage scalability. Moving to one of these databases would also enable scheduled refreshes and automated retraining workflows.

The Power BI dashboard, although functional, represents only the first phase of reporting. The current version is intentionally simple, with one main page focused on high-level KPIs and transaction-level drill-down. Additional dashboard pages could expand the analytical capabilities of the system. Potential enhancements include threshold tuning visuals, drift detection, longitudinal performance monitoring, feature importance summaries, or analyst workload analytics. Incorporating these views would help supervisors and data teams evaluate the model more holistically and identify when updates may be necessary.

A related limitation comes from the fact that the model was evaluated only on a static test set. In real fraud operations, model performance changes over time as fraud patterns evolve. A future iteration of this system could include a monitoring pipeline that automatically compares current performance with historical baselines. Such drift monitoring would help ensure that the model remains reliable and does not degrade in accuracy or recall as new fraud patterns emerge.

Finally, although the project incorporated financial evaluation, the cost model was necessarily simplified. In practice, organizations may incur additional operational costs such as case investigation time, customer contact effort, or manual verification processes. The cost of missed fraud may also vary depending on account type or customer profile. Expanding the financial evaluation framework to include multiple cost parameters would provide a richer understanding of the model's business impact.

Despite these limitations, the system provides a strong foundation for future development. Each component, modeling, database design, and reporting can be expanded without restructuring the entire pipeline. The modular workflow makes it straightforward to integrate new models, additional features, or more advanced analytics. As a result, this project not only demonstrates the feasibility of building an end-to-end fraud detection system but also highlights clear opportunities for continued improvement and operational scaling.

10. Conclusion

This project set out to design and implement a complete fraud detection reporting system that integrates machine learning, database engineering, and business intelligence. Although each component, model development in Python, data storage in SQLite, and dashboard creation in Power BI could have been built independently, the goal was to demonstrate how these tools work together to support operational decision-making. By connecting the technical workflow to the daily tasks of fraud analysts, the system aligns statistical modeling with real business needs, which is a central objective of applied analytics.

The project began by clearly defining the business context and the challenges associated with fraud detection. With fraud representing less than 0.2% of all transactions, the workload on analysts can be overwhelming, and traditional reporting tools often lack structure for prioritization. This motivated the development of a system that does more than predict fraud, it also helps analysts understand risk, allocate time efficiently, and quantify the financial impact of their work. The dataset provided a realistic environment for testing these ideas because of its imbalance, its similarity to real transaction data, and its size, which mimics the volume analysts encounter in practice.

Model development focused on evaluating multiple algorithms and comparing their ability to detect fraudulent transactions under extreme imbalance. Logistic Regression emerged as the champion model due to its interpretable output, stable recall, and consistent performance across cross-validation folds. While some ensemble models achieved higher precision, the priority for this use case was minimizing missed fraud. The final model achieved strong recall, consistent scoring behavior, and reliable probability outputs. These qualities made it suitable for an operational environment where transparency and predictability matter as much as statistical accuracy.

A critical part of the project involved not just evaluating predictions but also interpreting their financial consequences. By computing avoided fraud losses, review costs, and batch-level net savings, the evaluation framework moved beyond traditional metrics like F1 score and ROC-AUC. This financial analysis demonstrated how model outcomes translate into operational impact. Even though net savings varied across batches which is expected given fraud's irregular nature the model reliably identified all fraud cases in the test set, preventing significant losses. These financial insights provided the foundation for executive-level reporting and reinforce the model's operational relevance.

Power BI served as the reporting layer, transforming raw model outputs into a structured and intuitive interface for analysts. The dashboard integrates KPIs, prediction distributions, risk-level summaries, and a transaction-level drill-down table. By combining flexible slicers with dynamic DAX measures, the dashboard allows analysts to explore model behavior interactively and supports daily fraud review workflows. The decision to create a dedicated reporting database with only Logistic Regression outputs helped ensure that the dashboard reflected a single, consistent model and prevented confusion arising from earlier experimental outputs.

As the system was tested end-to-end, it became clear that the modular architecture allowed each component to be validated independently while still supporting a cohesive workflow. The checks performed across Python, SQLite, and Power BI helped ensure that predictions remained accurate, financial metrics were computed correctly, and the dashboard responded smoothly to user interaction. These validation steps reinforce the system's practicality and readiness for operational deployment.

Although the project successfully delivered a fully functional fraud detection and reporting system, it also highlighted several opportunities for enhancement. Future iterations could incorporate richer datasets, adopt more advanced imbalance-handling techniques, introduce continuous monitoring to detect model drift, or expand the dashboard to include additional analytical views. The modular design of the current system ensures that these enhancements can be integrated without significant restructuring.

Overall, this project demonstrates how data science and business intelligence can be combined to address real-world challenges in fraud detection. By focusing on interpretability, usability, and financial impact, the system provides analysts with a valuable tool that not only identifies potential fraud but also enhances decision-making. The end-to-end workflow spanning modeling, storage, and reporting illustrates a practical approach to building analytics systems that are both technically rigorous and operationally meaningful.