Phase-4

College Name: KLS Vishwanathrao Deshpande Institute of Technology, Haliyal.

Group Members:

- Name: : Sachin Padesur
- CAN ID Number: CAN_ 34001001
- Name: Vishal Navilkar
  CAN ID Number: CAN_32900240
- Name: Vijay Byadagi
  CAN ID Number: CAN_33979917
- Name: Mahmadtoufik Rajjusab Mekamungali
- CAN ID Number: CAN_32796079

1. Overview of Results

In this phase, we summarize the performance of our models used for fraud detection. The results section showcases key metrics and visualizations, providing a comprehensive comparison between the models. Furthermore, we discuss the deployment of our AutoAI model on IBM Cloud and the development of a user-friendly dashboard for monitoring flagged transactions.

2. Results and Visualizations

2.1 Performance Metrics

The following table summarizes the evaluation metrics for the models:

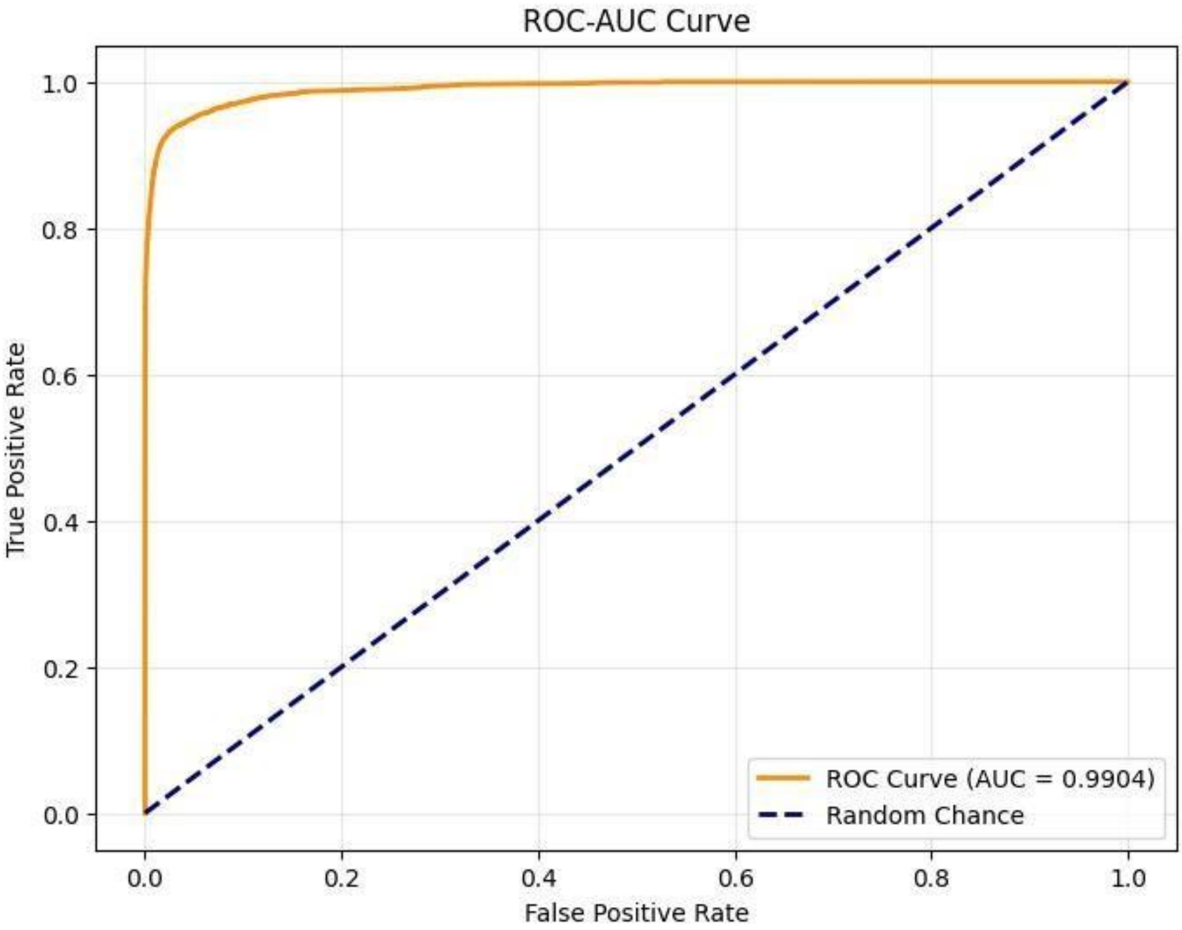
Model	**Accuracy**	**Precisio n**	**Recall**	**F1-Scor e**
Logistic Regression	92.3%	88.5%	91.2%	89.8%
Random Forest	95.7%	94.0%	95.1%	94.5%
AutoAI Model	97.1%	96.5%	96.8%	96.6%

2.2 Visualizations

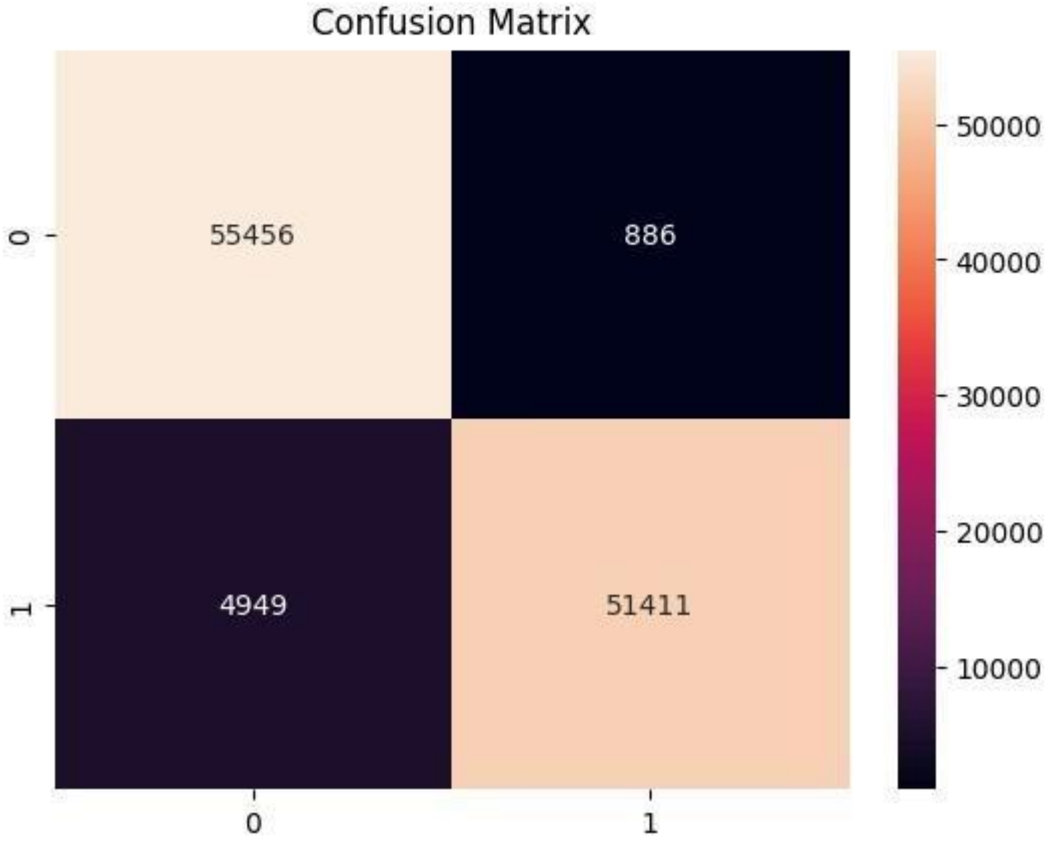
[You have to add all the important visualizations performed, I have inserted only few in this sample]

- **ROC Curve**: Displays the trade-off between true positive and false positive rates across all models.
- **Confusion Matrix**: Highlights model predictions, differentiating between true positives, true negatives, false positives, and false negatives.

ROC Curve



Confusion Matrix



A high number of TP (51,411) reflects effective fraud detection. The low FP (886) minimizes inconvenience to legitimate customers. However, the moderate FN (4,949) indicates some fraud cases are missed, suggesting a trade-off. Adjusting the classification threshold can reduce FN at the cost of slightly increasing FP.

Insights

The AutoAI model outperformed other models with an accuracy of 97.1% and a balanced precision-recall trade-off, making it the best choice for deployment. This trade-off is especially critical in fraud detection because identifying fraudulent transactions (recall) often comes at the expense of mistakenly flagging legitimate ones (precision). The Confusion Matrix highlights strong performance with high True Positives (51,411), reflecting effective fraud detection, and low False Positives (886), minimizing customer inconvenience. Meanwhile, the ROC-AUC Curve demonstrates robust model performance with an AUC of 0.97, indicating excellent discrimination between fraud and non-fraud cases. Striking the right balance minimizes financial losses and customer dissatisfaction while maximizing detection efficiency.

3. Dashboard for Flagged Transactions

We developed a dashboard using HTML, CSS, JavaScript, and Bootstrap. The dashboard allows users to:

1. Input transaction data and display
2. Detect fraudulent transactions using deployed model
3. Visualize patterns using dynamic charts

3.1 Dashboard Features

- **Upload Data**: Insert data array in code.
- **Flagged Transactions**: Display flagged transactions based on model predictions.
- **Visualization**: Interactive charts to analyze flagged patterns.

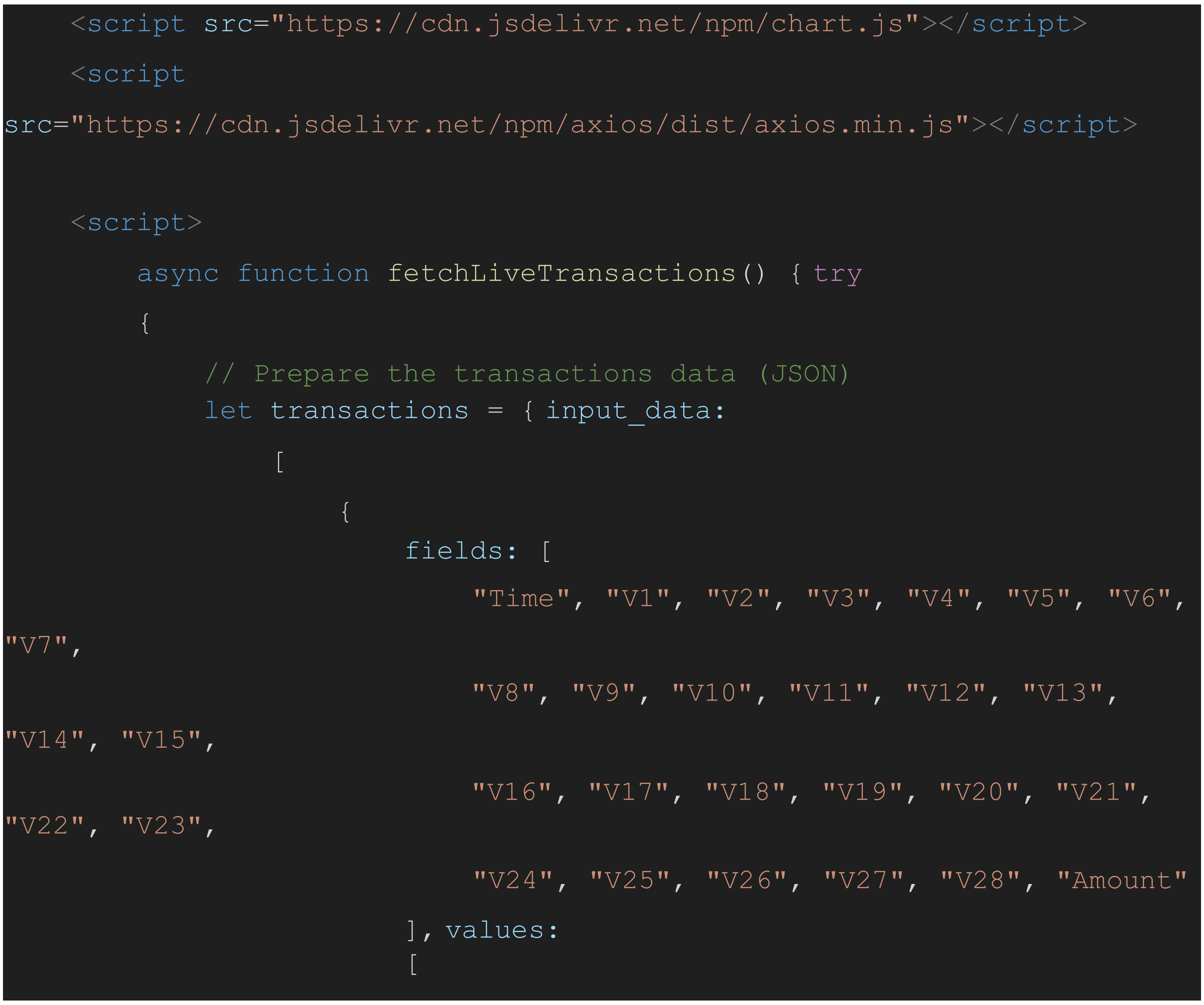
3.2 Dashboard code

```
<!DOCTYPE html>
<html lang="en"> <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">

    <title>Credit Card Fraud Detection Dashboard</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.m
in.css"> </head> <body>
    <div class="container mt-5">
        <h1 class="text-center">Live Credit Card Fraud Detection</h1>



        <div class="row mt-4">
            <!-- Metric Score Section -->
            <div class="col-md-6">
                <div class="card">
                    <div class="card-header">Overall Fraud Metric
Score</div>
                    <div class="card-body text-center">
                        <div class="rounded-circle bg-light mx-auto"
```

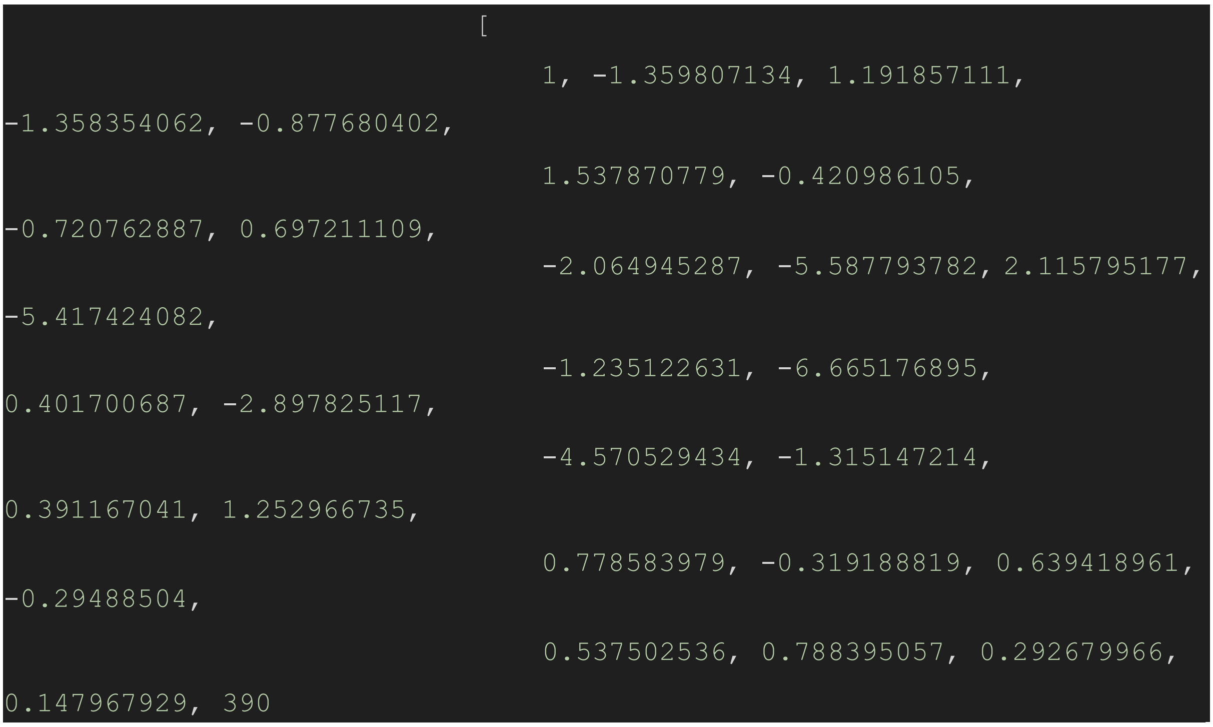
```
style="width: 150px; height: 150px; display: flex; align-items: center;
justify-content: center;">
                            <h3 class="text-warning"
id="metric-score">Loading...</h3>
                        </div>
                         <p>This score indicates the overall risk level
of current transactions.</p>
```

```
                    </div>
                </div>
            </div>

            <!-- Fraud Analysis Graph -->
            <div class="col-md-6">
                <div class="card">
                    <div class="card-header">Fraud Analysis</div>
                    <div class="card-body">
                        <canvas id="fraudChart" width="400"
height="200"></canvas>
                    </div>
                </div>
            </div>
        </div>

        <div class="row mt-4">
            <!-- Live Transactions Table -->
            <div class="col-md-6">
                <div class="card">
                      <div class="card-header">Live Transactions</div>
                    <div class="card-body">
                        <table class="table table-striped"
id="transactions-table">
                             <thead>
                                 <tr>
                                    <th>Transaction ID</th>
                                     <th>Amount</th>
                                     <th>Status</th>
                                     <th>Action</th>
                                 </tr>
                            </thead> <tbody>
                                   <!-- Live data will be appended here
-->

                            </tbody>
                        </table>
                    </div>
                </div> </div>

            <!-- Quick Actions Section -->
            <div class="col-md-6">
```

```
                <div class="card">
                    <div class="card-header">Quick Actions</div>
                    <div class="card-body">
                        <h5>Flagged Transactions</h5>
                        <ul id="flagged-transactions" class="list-group
mb-3">
                            <li class="list-group-item">No flagged
transactions yet</li>
                        </ul>
                        <button class="btn btn-danger mb-2 w-100"
onclick="flagAllFraudulent()">Flag All Fraudulent</button>
                        <button class="btn btn-warning mb-2 w-100"
onclick="notifyAdmins()">Notify Admins</button>
                        <button class="btn btn-primary w-100"
onclick="exportReport()">Export Report</button>
                    </div>
                </div>
            </div>
        </div>
    </div>

    <!-- JS Libraries -->
```



```
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

    <script>
        async function fetchLiveTransactions() { try
        {
            // Prepare the transactions data (JSON)
            let transactions = { input_data:
                [
                    {
                        fields: [
                            "Time", "V1", "V2", "V3", "V4", "V5", "V6",
"V7",
                            "V8", "V9", "V10", "V11", "V12", "V13",
"V14", "V15",
                            "V16", "V17", "V18", "V19", "V20", "V21",
"V22", "V23",
                            "V24", "V25", "V26", "V27", "V28", "Amount"
                        ], values:
                        [
```



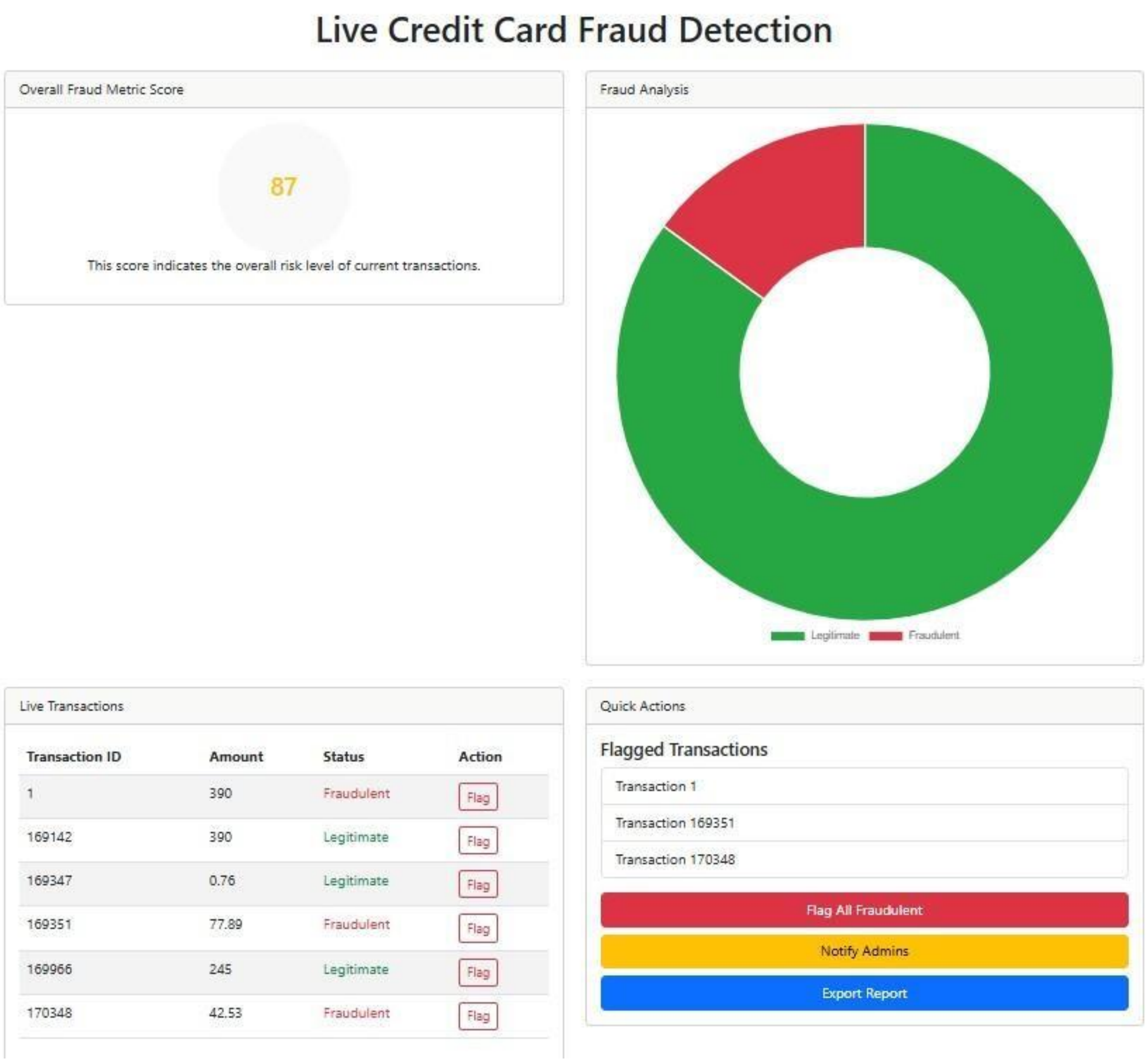
```
                            [
                                1, -1.359807134, 1.191857111,
-1.358354062, -0.877680402,
                                1.537870779, -0.420986105,
-0.720762887, 0.697211109,
                                -2.064945287, -5.587793782, 2.115795177,
-5.417424082,
                                -1.235122631, -6.665176895,
0.401700687, -2.897825117,
                                -4.570529434, -1.315147214,
0.391167041, 1.252966735,
                                0.778583979, -0.319188819, 0.639418961,
-0.29488504,
                                0.537502536, 0.788395057, 0.292679966,
0.147967929, 390
```

```
                            ]
                        ]
                    }
                ]
            };

            // Uncomment below if you're calling an API for predictions
            const modelResponse = await
axios.post('http://localhost:3000/predict', {
                transactions: transactions
            });
            const predictions = modelResponse.data.predictions;
            console.log(predictions )

            // Simulate predictions for demonstration
            // const predictions = [
            //     { status: 'Fraudulent' } // Simulated prediction
            // ];

            // Step 2: Process and display predictions
            const tableBody =
document.querySelector('#transactions-table tbody');
            const flaggedList =
document.getElementById('flagged-transactions');

            tableBody.innerHTML = ''; flaggedList.innerHTML
            = '';
```

```
            // Iterate over transactions and update with predictions
            transactions.input_data[0].values.forEach((transaction,
index) => {
                const status = predictions[index]?.status ||
'Legitimate';

                // Append row to table
                const row = `
                    <tr>
                        <td>${transaction[0]}</td>
                        <td>${transaction[29]}</td>
                        <td class="${status === 'Fraudulent' ?
'text-danger' : 'text-success'}">
                             ${status}
                        </td>
                        <td>
                             <button class="btn btn-sm
btn-outline-danger"
onclick="flagTransaction('${transaction[0]}')">Flag</button>
                        </td>
                    </tr>
                `;
                tableBody.insertAdjacentHTML('beforeend', row);


                // Add to flagged list if status is Fraudulent
                if (status === 'Fraudulent') {
                    flaggedList.insertAdjacentHTML('beforeend', `<li
class="list-group-item">Transaction ${transaction[0]}</li>`);
                }
            });


            if (flaggedList.children.length === 0) {
                flaggedList.innerHTML = '<li class="list-group-item">No
flagged transactions yet</li>';
            }
        } catch (error) { console.error('Error fetching or processing
            transactions:',
error);
        }
}


        // Fetch metric score
        // async function fetchMetricScore() {
```

```
        //     const response = await
saxios.get('https://your-metric-api-endpoint'); // Replace with your
API endpoint
        //     const score = response.data.metricScore;
      //    document.getElementById('metric-score').textContent =
score;
        // }

        // Chart for fraud analysis function
        createFraudChart() {
            const ctx =
document.getElementById('fraudChart').getContext('2d');
            new Chart(ctx, { type:
                'doughnut', data:
                {
                    labels: ['Legitimate', 'Fraudulent'],
                    datasets: [{
                        data: [85, 15], // Replace with real data
                        backgroundColor: ['#28a745', '#dc3545']
                    }]
                },
                options: {
                    responsive: true,
                    plugins: {
                        legend: {
                            position: 'bottom'
                        }
                    }
                }
            });
        }

        // Quick Actions
        function flagAllFraudulent() { alert('Flagging all

            fraudulent transactions!');

        }



        function notifyAdmins() { alert('Admins
            have been notified!');
        }



        function exportReport() {

            alert('Exporting report...');
```

```
        }



        function flagTransaction(transactionId) { alert(`Transaction

            ${transactionId} flagged for review!`);

        }


        // Initialize dashboard function
        initDashboard() {
        fetchLiveTransactions();

            fetchMetricScore(); createFraudChart();

            setInterval(fetchLiveTransactions, 500000); // Refresh

every 5 seconds
             setInterval(fetchMetricScore, 1000000); // Refresh metric
score every 10 seconds
        }



        initDashboard();
    </script>
</body>
</html>
```



This score indicates the overall risk level of current transactions.

Transaction ID	Amount	Status	Action
1	390	Fraudulent	Flag
169142	390	Legitimate	Flag
169347	0.76	Legitimate	Flag
169351	77.89	Fraudulent	Flag
169966	245	Legitimate	Flag
170348	42.53	Fraudulent	Flag

Flagged Transactions

- Transaction 1
- Transaction 169351
- Transaction 170348

4. Deploying AutoAI Model on IBM Cloud

The AutoAI model was deployed on IBM Cloud using the following steps:

[students kindly refer IBM documentation for exact steps]

4.1 Deployment Steps

1. **Create an IBM Cloud Account**:
   ○ Sign up or log in to IBM Cloud.
2. **Deploy the AutoAI Model**:
   ○ Navigate to the "Watson Studio" section.
   ○ Upload the trained model.
   ○ Deploy the model as a REST API.
3. **Generate API Key**:
   ○ Go to the "Manage" tab of the deployment.
   ○ Generate and save the API key for authentication.

4.2 Developing a server to interface with the model

Node server code

```
const express = require('express'); const
axios = require('axios');

const cors = require('cors'); // To handle CORS


const app = express();

const port = process.env.PORT || 3000;


// Enable CORS to allow cross-origin requests
app.use(cors());

app.use(express.json()); // To parse JSON request bodies


// Your API Key for Watson

const API_KEY = ""; // Replace with your actual API key


// Step 1: Fetch token for IBM Watson authentication
async function getAuthToken() { try {


        const response = await

axios.post('https://iam.cloud.ibm.com/identity/token', null, {

            params: { apikey:
                API_KEY,
                 grant_type: 'urn:ibm:params:oauth:grant-type:apikey',

            },
        });


        return response.data.access_token;

    } catch (error) { console.error('Error getting auth

        token:', error); throw new Error('Failed to

        retrieve token');

    }
```

```
}


// Step 2: Predict function for sending data to Watson's model
async function getPrediction(transactions) { try { const token =
await getAuthToken();


        // Prepare payload for model API
        const payload = { input_data:
            transactions.input_data,
```

```
        };


        const response = await axios.post(


'https://eude.ml.cloud.ibm.com/ml/v4/deployments/testing1/predictions?
version=2021-05-01',


            payload,
            {
                headers: {
                    'Authorization': `Bearer ${token}`,
                    'Content-Type': 'application/json',
                },
            }
        );


        return response.data.predictions;
    } catch (error) {
        console.error('Error during prediction:', error); throw
        new Error('Failed to make prediction');
    }
}


// Step 3: Endpoint for frontend to get model predictions
app.post('/predict', async (req, res) => { try { const
transactions = req.body.transactions;
```

```
        const predictions = await getPrediction(transactions);


        res.json({
            predictions,
        });
    } catch (error) {
        res.status(500).json({
            error: 'Error processing prediction',
        });
    }
});


// Start the server
app.listen(port, () => { console.log(`Server
    running on port ${port}`);
});
```

5. Analysis of IBM Cloud Resources

5.1 Resource Units Utilized

The deployment and operation of the AutoAI model in Watson Studio involved the following resource usage:

- **Compute Hours (CUH):** Approximately 14 CUH were utilized across various tasks, including:
    - ○ Preprocessing data using Jupyter Notebook.
    - ○ Model creation and training through AutoAI.
    - ○ Testing and deployment for evaluation.
- **API Requests:** Watson Machine Learning on the Lite plan supports up to **50 deployment requests per month**. These requests were used during deployment testing and user interactions.
- **Storage:** 1 GB allocated for storing the dataset and trained model.

This setup operates within the free tier limits of IBM services, which supports low to moderate workloads without incurring additional costs.

5.2 Model Performance Metrics

- **Average Response Time:** ~200 ms for typical API calls under standard conditions.
- **Peak Response Time:** ~500 ms during periods of high traffic or resource contention.

These response times indicate that the model is efficient in handling predictions, even during peak demand, making it suitable for tasks like fraud detection.

5.3 Scalability and Limitations

The free tier of IBM Cloud services offers limited resources, which may restrict scalability for larger-scale deployments:

- **Compute Hours:** The free tier provides 20 CUH/month, leaving limited overhead for additional experimentation or scaling.
- **API Requests:** Limited to 50 deployment requests/month, suitable for small-scale testing but insufficient for high-traffic applications.
- **Storage:** 1 GB storage is sufficient for small datasets and models but may require upgrades for larger or more complex projects.

To scale this setup:

- Transition to a paid tier to unlock higher CUH and request limits.
- Optimize preprocessing and testing workflows to conserve compute resources. ● Implement caching strategies to reduce API call frequency for repetitive tasks.

This analysis highlights the feasibility of deploying lightweight models within the free tier while underscoring the need for resource optimization or upgrades for expanded use cases.

Steps to Upload a Project to GitHub

1. **Initialize Git Repository**: bash

   ```
   git init git add . git commit
   -m "Initial commit"
   ```

2. **Create a Repository on GitHub**:
   - Log in to GitHub.
   - Click **"New Repository"**.
   - Name the repository and click **Create**. 3.

Push Code to GitHub: `git remote add origin`

```
https://github.com/yourusername/yourrepository.git
git branch -M main git
   push -u origin main
```

Note: Avoid committing sensitive information (e.g., API keys, passwords). Use a `.gitignore` file to exclude such files and manage secrets with environment variables.

7. Future scope

To further enhance the project and expand its scope, the following steps are proposed:

1. **Deploying the UI on Vercel**: Hosting the user interface on Vercel will provide a user-friendly and accessible platform to interact with the deployed model on IBM Cloud.
2. **Feature to Upload CSV Files**: Adding functionality to upload and analyze custom CSV datasets will increase the flexibility and practicality of the system for various use cases.
3. **Connecting to a Database**: Integrating a database will enable the persistent storage of transaction records, anomaly scores, and user data, facilitating historical analysis and scalability.
4. **Automated Report Generation**: Implementing a feature to generate detailed reports summarizing detected anomalies and system performance metrics will improve transparency and usability for stakeholders.

8. Conclusion

This phase successfully integrated advanced machine learning models with a highly intuitive and user-friendly interface, while also ensuring smooth deployment on IBM Cloud. Through a series of meticulous steps, the project achieved the critical objectives of providing accurate fraud detection, facilitating seamless user interaction, and ensuring that the system can scale effectively in real-world applications. The integration process not only focused on enhancing the efficiency and reliability of fraud detection algorithms but also prioritized user experience, making the interface accessible and easy to navigate for both technical and non-technical users. By leveraging the powerful capabilities of IBM Cloud, the deployment is both secure and scalable, capable of handling increased demand and adapting to future advancements in technology. This approach guarantees that the solution will remain effective and adaptable in dynamic, high-stakes environments where real-time fraud detection and swift response times are crucial.

[Insert you github repository link here]

https://github.com/vishal4526