Artificial Intelligence and Machine Learning

Online Payments Fraud Detection using Machine Learning

Introduction

**Project Title : ** Online Fraud Detection Using Python

**Team ID : ** LTVIP2026TMIDS64588

**Team Size : ** 4

Team member : -

**Team Leader : ** Maddala Jahnavi

**Team member : ** Leela Vasanth

**Team member : ** Pallagani Nagalakshmi

**Team member : ** Tumma Deepak

Summary

This document is a complete project write-up for an "Online Payments Fraud Detection" system using machine learning. It covers the problem statement, scenarios, technical architecture, dataset and preprocessing steps, exploratory data analysis (EDA), model building and comparison, evaluation, model saving, Flask-based deployment, project structure, sample code snippets, and illustrative images.

Table of contents

---

1. Project overview

Online Payments Fraud Detection uses supervised machine learning to identify fraudulent payment transactions. The model learns from historical labeled transactions (fraud vs. not fraud) and predicts whether new transactions are suspicious. The final model is wrapped in a Flask app for real-time predictions via a web UI or API.

2. Scenarios

- **Scenario 1 — Real-time Fraud Monitoring**: Monitor transactions as they occur. Features like amount, device info, geo-location, transaction type and historic user behavior are scored and high-risk transactions are flagged.
- **Scenario 2 — Fraudulent Account Detection**: Detect accounts that behave abnormally (sudden high spend, many failed logins, rapid transfers) and quarantine or block them.
- **Scenario 3 — Adaptive Fraud Prevention**: Continuously retrain or fine-tune models as new examples arrive to adapt to new fraud tactics.

3. Objectives

- Learn core ML workflow: EDA, preprocessing, modeling, evaluation.
- Build and compare multiple classification models.
- Tune the best-performing model with hyperparameter search.
- Save the model and deploy via Flask.

4. Technical architecture & environment

Languages & tools: Python, pandas, scikit-learn, xgboost, matplotlib, seaborn, Flask, pickle.

Quick environment setup (run in Anaconda Prompt or a terminal):

```
pip install numpy pandas scikit-learn matplotlib scipy pickle-mixin seaborn
flask xgboost
```

High-level architecture (diagram)

Description: Data → Preprocessing → Model training → Model.pkl → Flask API → Web UI / Monitoring Dashboard.

5. Project structure (recommended)

```
fraud_detection_project/
├─ data/
│  └─ transactions.csv
├─ training/
│  └─ train_model.py
├─ training_ibm/   # optional
├─ app.py
├─ model.pkl
├─ requirements.txt
├─ templates/
│  ├─ home.html
│  ├─ predict.html
│  └─ submit.html
├─ static/
│  └─ (css, js, screenshots)
└─ README.md
```

6. Data collection

Collect a labeled dataset containing transaction features and a binary label `isFraud` (0 = genuine, 1 = fraud). Common columns used in tutorials/examples:

- `step` (time-step)
- `type` (TRANSFER, CASH_OUT, PAYMENT, etc.)
- `amount`
- `nameOrig`, `oldbalanceOrg`, `newbalanceOrig`
- `nameDest`, `oldbalanceDest`, `newbalanceDest`
- `isFraud` (target)

Place the dataset file in `data/transactions.csv`.

7. Data exploration & visualization (EDA)

Perform univariate, bivariate and descriptive analysis.

Univariate examples

- Histograms for `amount`, `oldbalanceOrg`, `newbalanceDest`.
- Boxplots to see outliers for `amount` and balances.
- Countplots for categorical features: `type`, `isFraud`.

Sample plots (placeholders below). Replace with real charts after running the notebook.

Bivariate examples

- `isFraud` vs `amount` (boxplot or violinplot)
- `isFraud` vs `type` (countplot with hue)
- Jointplots: `newbalanceDest` vs `isFraud`

Descriptive statistics

Use `df.describe()` and `df.info()` to understand types, missing values, means, std, min, max and percentiles.

8. Data pre-processing

Typical steps:

- Drop unnecessary identifiers: `nameOrig`, `nameDest`.
- Check and handle missing values: `df.isnull().sum()`.
- Detect and treat outliers (log-transform `amount`, winsorize, or clip).
- Encode categorical features: `type` → OneHot or LabelEncoder depending on the algorithm.
- Scale numeric features when required (StandardScaler or RobustScaler for outliers).
- Split X / y and create train/test sets with `train_test_split`.

Snippet — cleaning and split

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler

# load
df = pd.read_csv('data/transactions.csv')

# drop ids
DF = df.drop(columns=['nameOrig','nameDest'], errors='ignore')

# encode
le = LabelEncoder()
DF['type_enc'] = le.fit_transform(DF['type'].astype(str))

# features / target
X = DF.drop(columns=['isFraud','type'])
```

```
y = DF['isFraud']

# split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)
```

9. Model building (algorithms tried)

Try multiple classifiers, evaluate and select the best. Examples used in this project:

- RandomForestClassifier
- DecisionTreeClassifier
- ExtraTreesClassifier
- SVC (Support Vector Classifier)
- XGBoostClassifier

Snippet — train RandomForest and evaluate

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

pred = rf.predict(X_test)
print(classification_report(y_test, pred))
print('Accuracy', accuracy_score(y_test, pred))
```

Note on class imbalance: Fraud datasets are typically heavily imbalanced. Consider techniques like:

- Class weighting (`class_weight='balanced'`)
- Resampling: SMOTE, RandomUnderSampler/OverSampler
- Threshold tuning on predicted probabilities

10. Model comparison & evaluation

Evaluate models via confusion matrix, precision, recall, F1-score, ROC-AUC, and PR-AUC (precision-recall area). Because fraud detection values recall/precision trade-offs differently, use metrics that emphasize detection of the minority class (fraud) — e.g., recall and precision for the `isFraud` class.

Insert model comparison chart here after running experiments.

11. Model saving

Save the final selected model to disk with pickle for Flask integration.

```
import pickle

best_model = rf  # or whichever you selected
with open('model.pkl', 'wb') as f:
    pickle.dump(best_model, f)
```

12. Flask application: UI + server

Create a simple Flask app to load `model.pkl` and expose a UI and/or API endpoint for predictions.

Snippet — `app.py`

```
from flask import Flask, render_template, request
import pickle
import numpy as np

app = Flask(__name__)

# load model
with open('model.pkl','rb') as f:
    model = pickle.load(f)

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/predict', methods=['GET'])
def predict_page():
    return render_template('predict.html')

@app.route('/submit', methods=['POST'])
def submit():
    # retrieve form fields and convert to float/int as necessary
    # example: amount = float(request.form['amount'])
    data = [
        float(request.form.get('step', 0)),
        float(request.form.get('amount', 0)),
        # add or rearrange features exactly as used in training
    ]
    arr = np.array(data).reshape(1,-1)
    pred = model.predict(arr)
    label = 'Fraud' if pred[0]==1 else 'Legit'
    return render_template('submit.html', prediction=label)

if __name__ == '__main__':
    app.run(debug=True)
```

13. Example HTML templates

Place these files in `templates/`.

home.html (simplified)

```
<!doctype html>
<html>
  <head><title>Fraud Detector</title></head>
  <body>
    <h1>Online Payments Fraud Detection</h1>
    <a href="/predict">Predict transaction</a>
  </body>
</html>
```

predict.html (simplified)

```
<!doctype html>
<html>
  <body>
    <h2>Enter transaction details</h2>
    <form action="/submit" method="post">
      Step: <input name="step" type="number" step="1"/><br/>
      Amount: <input name="amount" type="number" step="0.01"/><br/>
      <!-- add other inputs as required -->
      <button type="submit">Submit</button>
    </form>
  </body>
</html>
```

submit.html

```
<!doctype html>
<html>
  <body>
    <h2>Prediction result</h2>
    <p>Result: {{ prediction }}</p>
    <a href="/">Back home</a>
  </body>
</html>
```

Screenshot placeholders*:*

14. How to run the project

1. Create a (virtual) environment and install requirements.
2. Put your dataset in `data/transactions.csv`.
3. Run training script: `python training/train_model.py` (this should create `model.pkl`).
4. Start Flask: `python app.py`.
5. Open `http://127.0.0.1:5000` in your browser.

15. Deliverables & screenshots

- `model.pkl` — saved trained model
- `app.py` — Flask server
- `templates/` — three HTML files
- `README.md` — project README with run instructions
- Example EDA and model charts (generated during training)

Add screenshots from the app and charts to the `static/` folder.

16. Next steps & improvements

- Add feature-engineering: user-level aggregates (average spend, frequency), time-window statistics.
- Build streaming scoring (Kafka or serverless functions) for true real-time scoring.
- Add a monitoring dashboard (Grafana/Power BI) for model drift and alerting.
- Implement explainability (SHAP) to show contributors to each fraud score.
- Deploy as container (Docker) and add CI/CD for retraining pipeline.

Appendix: quick checklists

Pre-training checklist

- ☐ Verify dataset labels
- ☐ Handle imbalanced classes
- ☐ Encode categorical features
- ☐ Feature scaling if needed

Deployment checklist

- ☐ Model saved as `model.pkl`
- ☐ Flask app loads the model
- ☐ UI inputs match training feature order
- ☐ Error handling for bad input types

This document was generated to provide a ready-to-use project guide. Replace the placeholder images with real screenshots and charts produced while running the notebooks/training scripts.

Screenshots or Demo

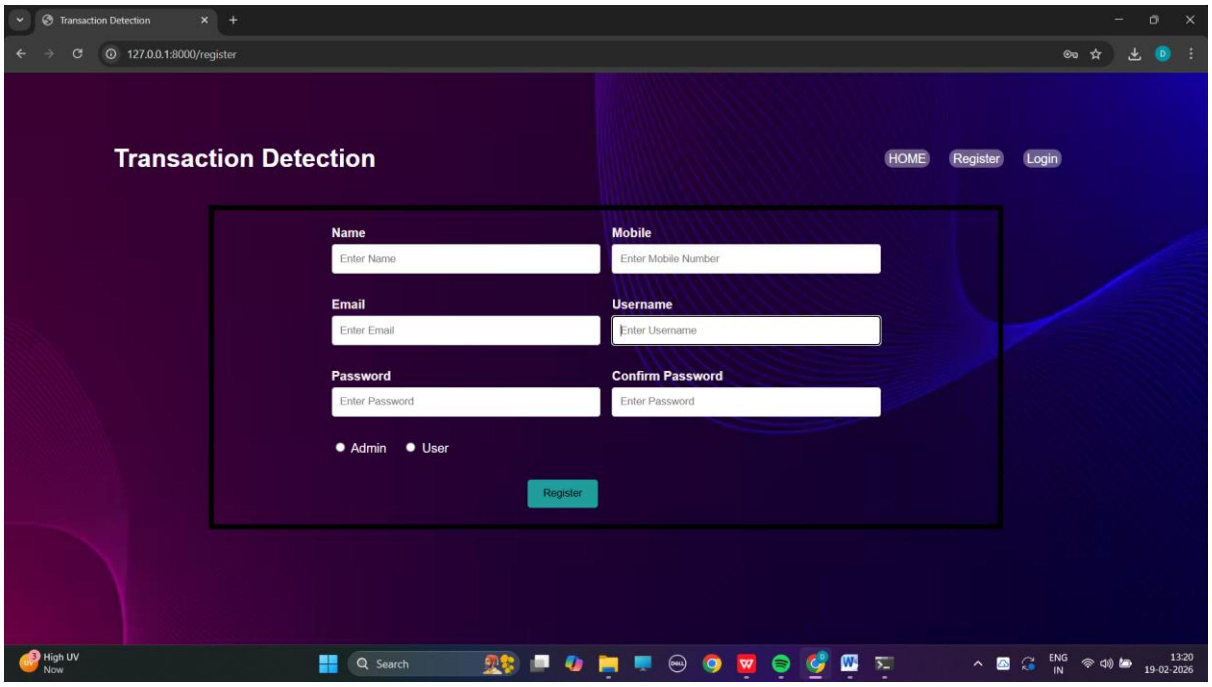
To provide a clear understanding of the Online Payments Fraud Detection, below are screenshots showcasing the key

features of the platform:

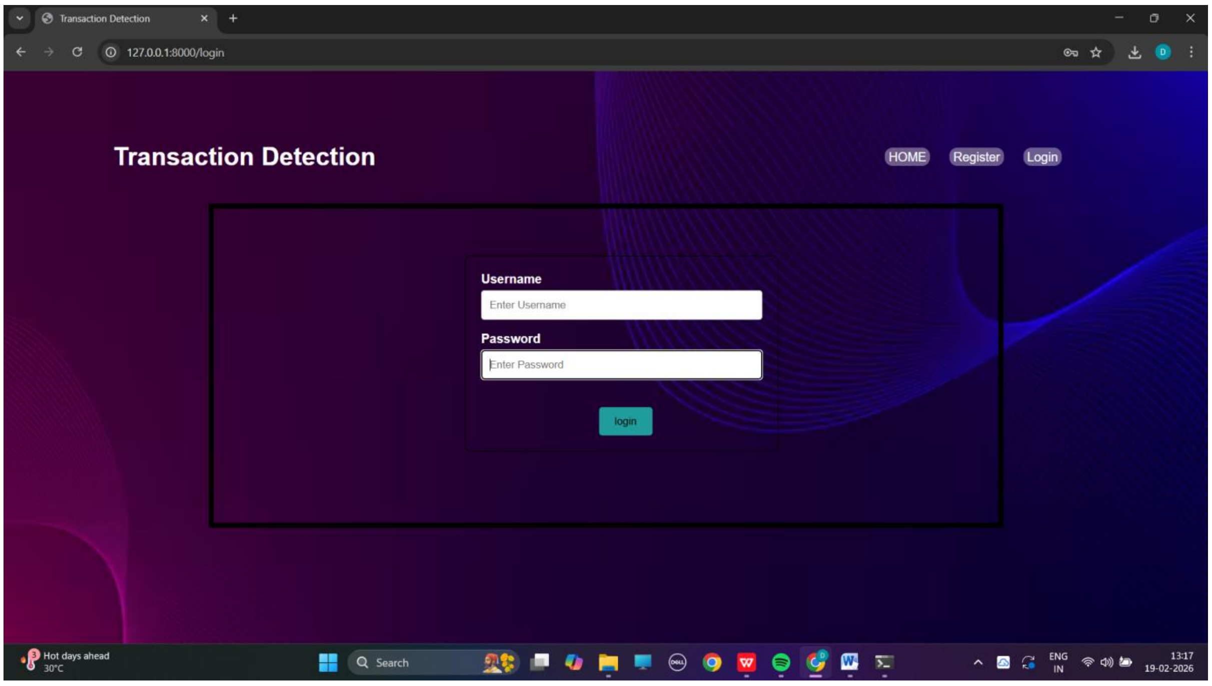
● Login and Registration Pages

● Transactions Detection Dashboard with Search Filters

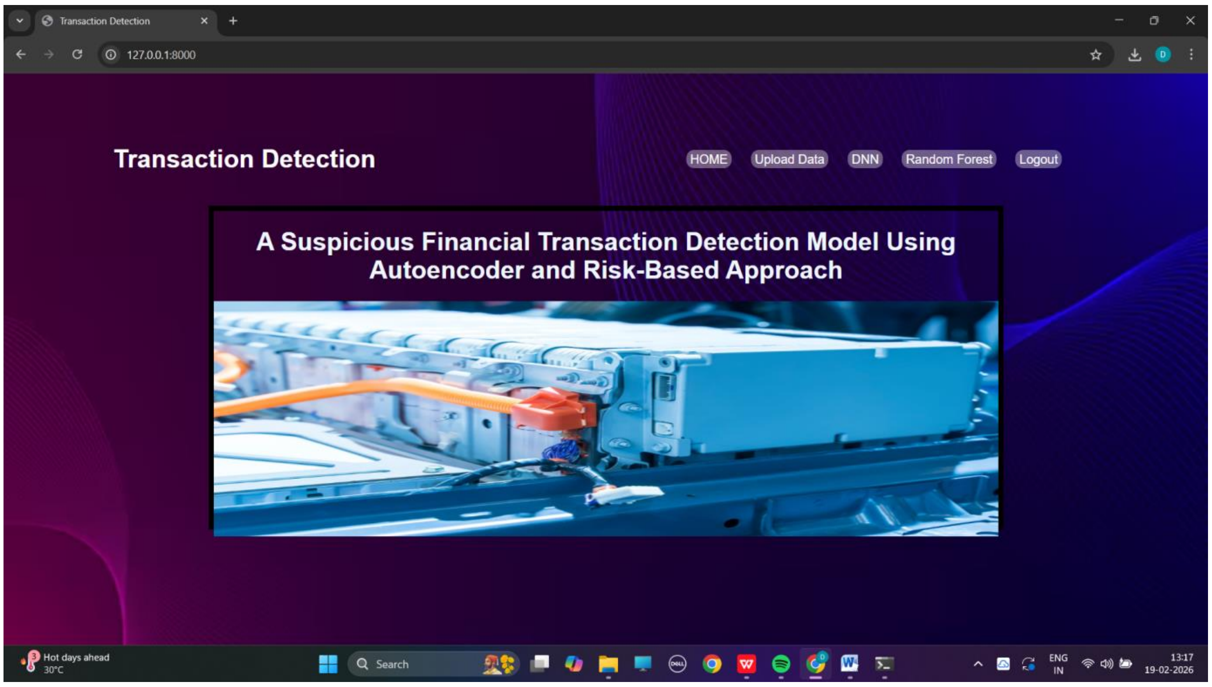
● **Registration Page**



● **Login page**



● Transactions Detection Dashboard with Search Filters



Demo Video Link : -

https://drive.google.com/file/d/1ztw9ow8UT0c10DX-j_6YeUrjttzgA-nr/view?usp=drivesdk

Github Link : -

https://github.com/deepak4535/Online-Payments-Fraud-Detection