Assessment Report

on

“Credit Card Fraud Detection”

submitted as partial fulfillment for the award of

BACHELOR OF TECHNOLOGY DEGREE

SESSION 2024-25

in

CSE(AI)

By

Name : **Group 13**

Samveg Jain (202401100300214)

Nitish Sablok (202401100300164)

Sanjay Saraswat (202401100300215)

Paras Jain (202401100300167)

Prahlad Kumar (202401100300171)

Section: C

Under the supervision of

"Mr. Mayank Lakhotia Sir"

KIET Group of Institutions, Ghaziabad

May, 2025

1. Introduction

Credit card fraud is a growing threat in the digital economy, demanding smart detection systems. This project uses unsupervised learning to identify fraudulent transactions without relying on labeled data, making it effective in detecting evolving fraud patterns in real-world scenarios. We use a real-world credit card transaction dataset containing both normal and fraudulent entries. Techniques like Isolation Forest and Autoencoders are applied to detect anomalies based on transaction behavior.

2. Problem Statement

To detect potentially fraudulent credit card transactions using financial data and transaction patterns. This anomaly detection system will help financial institutions mitigate risk by identifying suspicious activity without relying on labeled fraud cases.

3. Objectives

- To develop an unsupervised anomaly detection system for identifying fraudulent credit card transactions.
- To analyze transaction data and detect deviations from normal spending behavior.
- To apply unsupervised learning techniques such as Isolation Forest, DBSCAN, and Autoencoders.
- To evaluate the model's performance using appropriate metrics like precision, recall, F1-score, and ROC-AUC.

---

4. Methodology

- **Data Collection**
    - The user uploads a CSV file containing the credit card transaction dataset.
- **Data Preprocessing**
    - Missing values are handled using mean for numerical and mode for categorical features.
    - One-hot encoding is applied to convert categorical variables into numerical format.

- Features are standardized using StandardScaler to ensure uniform scale across the dataset.

- **Model Building**
- The dataset is split into training and testing sets to validate performance.
- A Logistic Regression classifier is trained on the processed training data to detect fraud.
- **Model Evaluation**
- The model's performance is evaluated using accuracy, precision, recall, and F1-score.
- A confusion matrix is generated and visualized using a heatmap to better understand prediction outcomes.

---

5. Data Processing

- Handled missing values:
  - **Mean imputation** for numerical features.
  - **Mode imputation** for categorical features.
- Converted categorical variables using **one-hot encoding** to make them suitable for model training.

- Applied **feature scaling** using **StandardScaler** to normalize the feature values and improve model performance.

---

6. Model Implementation

Logistic Regression is used for its simplicity, interpretability, and effectiveness in binary classification problems. After preprocessing, the dataset is split into training and testing sets. The Logistic Regression model is trained on the training set and then used to predict fraudulent transactions in the test set.

---

7. Evaluation Metrics

- **Accuracy**: Measures the overall correctness of the model's predictions.
- **Precision**: Proportion of transactions predicted as fraud that are actually fraudulent.
- **Recall**: Proportion of actual fraudulent transactions correctly identified by the model.

- **F1 Score**: Harmonic mean of precision and recall.

- **Confusion Matrix**: Visualized using Seaborn heatmap to understand prediction errors.

---

8. Result

- The model showed reasonable performance in detecting fraudulent transactions on the test set.
- The confusion matrix heatmap helped visualize the balance between true positives and false negatives.
- Precision and recall metrics highlighted how well the model identified actual frauds while minimizing false alarms.

---

9. Conclusion

The project successfully implemented an unsupervised anomaly detection system to identify fraudulent credit card transactions. Using Logistic Regression and proper preprocessing, the model achieved reasonable performance in detecting fraud and can aid financial institutions in reducing risk and preventing losses.

---

10. References

Scikit-learn documentation

Pandas Documentation

Seaborn Visualization library

Kaggle

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```python
# loading the dataset to a Pandas DataFrame
credit_card_data = pd.read_csv('/content/fraud_detection.zip')
```

```python
# first 5 rows of the dataset
credit_card_data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153

5 rows × 31 columns

```python
credit_card_data.tail()
```

```python
credit_card_data.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.068472
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.108821
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415

5 rows × 31 columns

```python
# dataset informations
credit_card_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
```

```python
# checking the number of missing values in each column
credit_card_data.isnull().sum()
```

	0
Time	0
V1	0
V2	0
V3	0
V4	0
V5	0
V6	0
V7	0
V8	0
V9	0
V10	0
V11	0
V12	0
V13	0
V14	0
V15	0

```
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

V18	0
V19	0
V20	0
V21	0
V22	0
V23	0
V24	0
V25	0
V26	0
V27	0
V28	0
Amount	0
Class	0

dtype: int64

```
[ ] # distribution of legit transactions & fraudulent transactions
credit_card_data['Class'].value_counts()
```

	count
Class	

```
fraud.Amount.describe()
```

	Amount
count	492.000000
mean	122.211321
std	256.683288
min	0.000000
25%	1.000000
50%	9.250000
75%	105.890000
max	2125.870000

dtype: float64

```
[ ] # compare the values for both transactions
credit_card_data.groupby('Class').mean()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V2
Class																		
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	...	-0.000644	-0.001235	-0.000024	0.000070	0.000182	-0.000072	-0.00008
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319	0.713588	0.014049	-0.040308	-0.105130	0.041449	0.05164

2 rows × 30 columns

```
legit_sample = legit.sample(n=492)
```

```
[ ] new_dataset = pd.concat([legit_sample, fraud], axis=0)
```

```
[ ] new_dataset.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27
36427	38532.0	-0.173327	0.444243	1.539241	-1.010397	0.170250	-0.106297	0.460180	-0.043621	-0.087733	...	-0.120110	-0.288660	-0.209572	-0.482383	-0.334417	0.872304	-0.194216
155177	104712.0	2.008267	0.218204	-1.539950	0.583119	0.240766	-1.675223	0.596099	-0.542519	1.223328	...	0.052506	0.467095	0.053894	0.511407	0.281465	-0.260088	-0.087998
8604	11609.0	-1.064190	-0.265377	1.848667	0.884056	0.366953	-0.676092	0.788136	-0.429524	0.613823	...	0.000336	0.179660	0.213751	0.680089	0.179273	1.327778	-0.255719
35699	38225.0	1.130079	0.048359	1.229691	1.211499	-0.753646	0.058038	-0.620792	0.181190	0.493411	...	0.147314	0.501852	0.016096	0.076881	0.270336	-0.317203	0.080379
105660	69633.0	1.122470	0.283391	0.582275	1.215417	-0.477178	-1.034619	0.255212	-0.285688	-0.148617	...	0.008507	0.002761	-0.033266	0.739762	0.521197	-0.467988	0.021776

5 rows × 31 columns

```
[ ] new_dataset.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.064945	...	0.778584	-0.319189	0.639419	-0.294885	0.537503	0.788395	0.292680	0
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.127396	...	0.370612	0.028234	-0.145640	-0.081049	0.521875	0.739467	0.389152	0
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.652250	...	0.751826	0.834108	0.190944	0.032070	-0.739695	0.471111	0.385107	0

```python
# separating the data for analysis
legit = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.Class == 1]
```

```python
print(legit.shape)
print(fraud.shape)
```

```
(284315, 31)
(492, 31)
```

```python
# statistical measures of the data
legit.Amount.describe()
```

	Amount
count	284315.000000
mean	88.291022
std	250.105092
min	0.000000
25%	5.650000
50%	22.000000
75%	77.050000
max	25691.160000

dtype: float64

280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.652250	...	0.751826	0.834108	0.190944	0.032070	-0.739695	0.471111	0.385107	0
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.632333	...	0.583276	-0.269209	-0.456108	-0.183659	-0.328168	0.606116	0.884876	-0
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.577829	...	-0.164350	-0.295135	-0.072173	-0.450261	0.313267	-0.289617	0.002988	-0

5 rows × 31 columns

```python
new_dataset['Class'].value_counts()
```

Class	count
0	492
1	492

dtype: int64

```python
new_dataset.groupby('Class').mean()
```

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26
0	95595.831301	-0.016763	0.107376	0.038756	-0.085949	0.001322	-0.070722	0.049763	0.039551	-0.027342	...	0.003665	0.021244	-0.001462	-0.018483	0.011574	-0.025505	0.020810
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319	0.713588	0.014049	-0.040308	-0.105130	0.041449	0.051648

2 rows × 30 columns

```
[ ] X = new_dataset.drop(columns='Class', axis=1)
    Y = new_dataset['Class']
```

```
[ ] print(X)
```

```
              Time        V1        V2        V3        V4        V5        V6  \
36427      38532.0 -0.173327  0.444243  1.539241 -1.010397  0.170250 -0.106297
155177    104712.0  2.008267  0.218204 -1.539950  0.583119  0.240766 -1.675223
8604       11609.0 -1.064190 -0.265377  1.848667  0.884056  0.366953 -0.676092
35699      38225.0  1.130079  0.048359  1.229691  1.211499 -0.753646  0.058038
105660     69633.0  1.122470  0.283391  0.582275  1.215417 -0.477178 -1.034619
...            ...       ...       ...       ...       ...       ...       ...
279863    169142.0 -1.927883  1.125653 -4.518331  1.749293 -1.566487 -2.010494
280143    169347.0  1.378559  1.289381 -5.004247  1.411850  0.442581 -1.326536
280149    169351.0 -0.676143  1.126366 -2.213700  0.468308 -1.120541 -0.003346
281144    169966.0 -3.113832  0.585864 -5.399730  1.817092 -0.840618 -2.943548
281674    170348.0  1.991976  0.158476 -2.583441  0.408670  1.151147 -0.096695

                V7        V8        V9  ...       V20       V21       V22  \
36427     0.460180 -0.043621 -0.087733  ...  0.036115 -0.120110 -0.288660
155177    0.596099 -0.542519  1.223328  ... -0.301631  0.052506  0.467095
8604      0.788136 -0.429524  0.613823  ...  0.800291  0.000336  0.179660
35699    -0.620792  0.181190  0.493411  ... -0.146961  0.147314  0.501852
105660    0.255212 -0.285688 -0.148617  ...  0.017733  0.008507  0.002761
...            ...       ...       ...  ...       ...       ...       ...
279863   -0.882850  0.697211 -2.064945  ...  1.252967  0.778584 -0.319189
280143   -1.413170  0.248525 -1.127396  ...  0.226138  0.370612  0.028234
280149   -2.234739  1.210158 -0.652250  ...  0.247968  0.751826  0.834108
281144   -2.208002  1.058733 -1.632333  ...  0.306271  0.583276 -0.269209
281674    0.223050 -0.068384  0.577829  ... -0.017652 -0.164350 -0.295135

               V23       V24       V25       V26       V27       V28  Amount
36427    -0.209572 -0.482383 -0.334417  0.872304 -0.194216 -0.193849    3.84
```

```
[ ] print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy score on Test Data :  0.949238578680203
```

```
[ ] import matplotlib.pyplot as plt
    import pandas as pd
    from sklearn.ensemble import IsolationForest

    # Load dataset

    X = data[["Amount", "Time"]]  # Selecting relevant features

    # Train Isolation Forest model
    model = IsolationForest(contamination=0.02, random_state=42)
    model.fit(X)

    # Predict anomalies (-1 represents fraud)
    data["fraud_prediction"] = model.predict(X)
    data["is_fraudulent"] = data["fraud_prediction"].apply(lambda x: True if x == -1 else False)

    # Plot transactions with fraud highlighted
    plt.figure(figsize=(10, 6))
    plt.scatter(data["Time"], data["Amount"], c=data["is_fraudulent"], cmap="coolwarm", edgecolors="k")
    plt.xlabel("Transaction Time")
    plt.ylabel("Transaction Amount")
    plt.title("Fraudulent Transaction Detection")
    plt.colorbar(label="Fraudulent (1) vs Non-Fraudulent (0)")
    plt.show()
```

```
[ ] print(X.shape, X_train.shape, X_test.shape)
```

```
(984, 30) (787, 30) (197, 30)
```

```
[ ] model = LogisticRegression()
```

```
[ ] # training the Logistic Regression Model with Training Data

    model = LogisticRegression(solver='liblinear')
    model.fit(X_train, Y_train)
```

```
LogisticRegression
LogisticRegression(solver='liblinear')
```

```
[ ] # accuracy on training data
    X_train_prediction = model.predict(X_train)
    training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

```
[ ] print('Accuracy on Training data : ', training_data_accuracy)
```

```
Accuracy on Training data :  0.9504447268106735
```

```
[ ] # accuracy on test data
    X_test_prediction = model.predict(X_test)
```