

Experiment 1

Nupur Ghangarekar
D15C
Roll no 26

Aim: Behavioral Drift-Based Fraud Detection

Theory:

1. Dataset Source

The dataset “**Luxury Cosmetics Fraud Analysis 2025**” is a **synthetic transactional dataset** designed to simulate real-world fraud scenarios in the luxury retail and cosmetics industry.

It reflects realistic customer behavior, payment methods, device usage, store locations, and transaction characteristics typically observed in high-value retail environments.

Purpose of the dataset:

- To study fraud patterns in luxury retail transactions
- To analyze customer purchasing behavior
- To develop and evaluate fraud-detection models

Synthetic datasets are commonly used in fraud analytics to:

- Preserve customer privacy
- Enable controlled experimentation
- Simulate rare fraud events at scale

2. Dataset Description

Each row represents a **single transaction**, labeled as fraudulent or non-fraudulent.

Key Attributes

Feature	Description
Transaction_ID	Unique transaction identifier
Customer_ID	Unique customer identifier

Transaction_Date	Date of transaction
Transaction_Time	Time of transaction
Customer_Age	Age of customer (missing values imputed)
Customer_Loyalty_Tier	Bronze / Silver / Gold / Platinum
Location	City where purchase occurred
Store_ID	Store or channel identifier
Product_SKU	Product identifier
Product_Category	Cosmetic category
Purchase_Amount	Transaction value
Payment_Method	Credit Card, Gift Card, Mobile Payment
Device_Type	Desktop, Mobile, Tablet
IP_Address	Customer IP address
Footfall_Count	Store activity level
Fraud_Flag	Target variable (0 = Non-Fraud, 1 = Fraud)

Data Characteristics

- **Imbalanced dataset** (fraud cases are rare)
- Mixture of **numerical and categorical features**
- Time-based transactional data
- Presence of missing values (e.g., customer age)

3. Mathematical Formulation of the Algorithm

Fraud detection is modeled as a **binary classification problem**:

$$D = \{(x_i, y_i)\}_{i=1}^N$$

Random Forest Classifier

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the class that is the **majority vote** of all trees.

Each decision tree is trained on:

- A bootstrap sample of the dataset
- A random subset of features at each split

If there are KKK trees, the final prediction is:

$$\hat{y} = \text{mode}\{T_1(x), T_2(x), \dots, T_K(x)\}$$

4. Algorithm Limitations

Despite its robustness, Random Forest has several limitations:

- **Class imbalance sensitivity:**
When fraudulent cases are rare, the model may favor the majority class (non-fraud).
- **Reduced interpretability:**
Compared to single decision trees, Random Forests are harder to interpret.
- **Computational cost:**
Training multiple trees increases memory usage and computation time.
- **Bias toward dominant patterns:**
Rare fraud patterns may be overlooked without proper tuning or resampling.

5. Methodology / Workflow

The experimental workflow followed a systematic pipeline:

1. **Data Loading**
 - The dataset was loaded using the Pandas library.
2. **Data Preprocessing**
 - Converted transaction date to datetime format.
 - Missing values in Customer_Age were handled using median imputation.
3. **Exploratory Data Analysis**
 - Fraud distribution visualization
 - Analysis by payment method, device type, location, and loyalty tier

4. **Feature Selection**
 - Numerical features: Customer_Age, Purchase_Amount, Footfall_Count
5. **Train-Test Split**
 - 70% training data, 30% testing data
6. **Model Training**
 - Random Forest classifier trained on training data
7. **Model Evaluation**
 - Performance measured using classification metrics
8. **Visualization**
 - Boxplots, histograms, scatter plots for fraud analysis

Workflow Diagram :

Data Collection → Data Cleaning → Feature Selection →
Train/Test Split → Model Training → Evaluation → Analysis

6. Performance Analysis

The model performance was evaluated using a classification report.

Observed Results

- **Accuracy:** ~97%
- **Precision (Fraud):** 0.00
- **Recall (Fraud):** 0.00

Interpretation

- The high accuracy is misleading due to **class imbalance**.
- The model successfully predicts non-fraud transactions.
- Fraudulent transactions are not detected effectively.
- Indicates the need for imbalance-handling techniques such as:
 - Class weighting
 - Oversampling (SMOTE)
 - Threshold tuning

7. Hyperparameter Tuning

Initially, the Random Forest model was trained using default hyperparameters.

Key Hyperparameters

- n_estimators: Number of trees
- max_depth: Maximum depth of trees
- min_samples_split: Minimum samples required to split a node
- class_weight: Balances class distribution

Tuning Strategy

Hyperparameter tuning can be performed using **GridSearchCV**, testing combinations of parameters to improve fraud detection performance.

Impact of Tuning

- Improves recall for fraud class
- Reduces bias toward non-fraud transactions
- Enhances model generalization

Code:

```
from google.colab import files

uploaded = files.upload()

import pandas as pd

df = pd.read_csv("luxury_cosmetics_fraud_analysis_2025.csv")

df.head()
```

	Transaction_ID	Customer_ID	Transaction_Date	Transaction_Time	Customer_Age	Customer_Loyalty_Tier	Location	Store_ID	F
0	702bdd9b-9c93-41e3-9db- a849b2422080	119dca0b-8554-4b2d-9bec- e964eaf6af97	2025-07-27	04:04:15	56.0	Silver	San Francisco	FLAGSHIP-LA	
1	2664c346-36bc-4acf- bc2b-8b0fdf46abc5	299df086-26c4-4708- b6d7-fcaeceb14637	2025-03-14	20:23:23	46.0	Platinum	Zurich	BOUTIQUE-SHANGHAI	
2	29ad1278-70ce-421f- 8d81-23816b39f4ac	dfa3d24d-b935-49a5- aa1d-7d57a44d8773	2025-02-20	12:36:02	32.0	Silver	Milan	POPUP-TOKYO	
3	07dc4894-e0eb-48f1- 99a7-1942b1973d9b	7a67e184-9369-49ee- aaec-18f5b51b230f	2025-04-25	19:09:43	60.0	Bronze	London	BOUTIQUE-NYC	
4	ae407054-5543-429c- 918a-cdcc42ea9782	cf14730a-8f5a-453d- b527-39a278852b27	2025-04-17	14:23:23	NaN	Platinum	Miami	BOUTIQUE-NYC	

```
import matplotlib.pyplot as plt

import seaborn as sns
```

```

sns.set(style="whitegrid")

df["Transaction_Date"] = pd.to_datetime(df["Transaction_Date"])

df["Customer_Age"].fillna(df["Customer_Age"].median(), inplace=True)
df["Customer_Age"].fillna(df["Customer_Age"].median(), inplace=True)

sns.countplot(x="Fraud_Flag", data=df)

plt.title("Fraud vs Non-Fraud Transactions")

plt.show()

sns.countplot(x="Payment_Method", hue="Fraud_Flag", data=df)

plt.xticks(rotation=45)

plt.title("Fraud by Payment Method")

plt.show()

sns.countplot(x="Device_Type", hue="Fraud_Flag", data=df)

plt.title("Fraud by Device Type")

plt.show()

sns.boxplot(x="Fraud_Flag", y="Purchase_Amount", data=df)

plt.title("Purchase Amount vs Fraud")

```

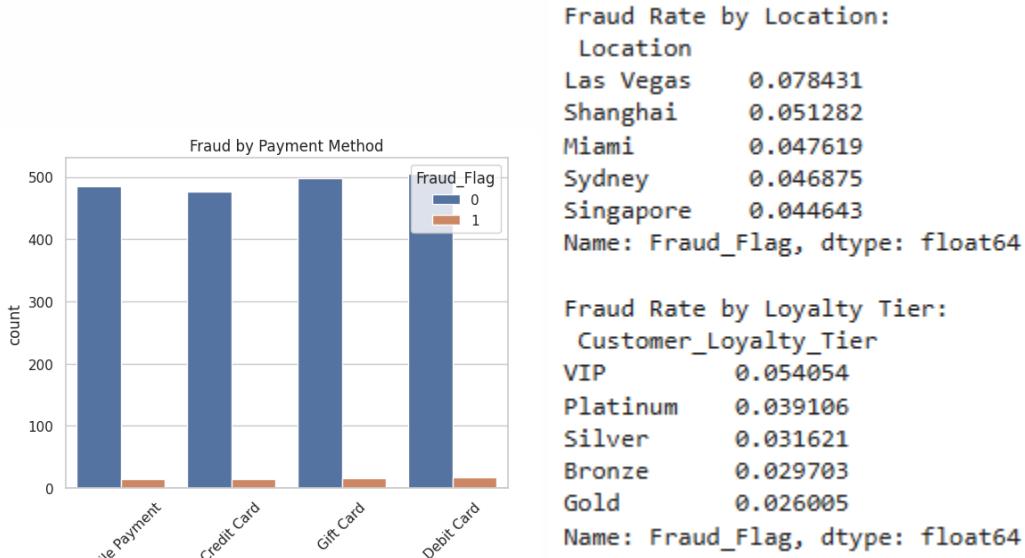
Fraud vs Non-Fraud Transactions

Fraud_Flag	count
0	~2000
1	~50

```

plt.show()

```



```

fraud_by_location =
df.groupby("Location") ["Fraud_Flag"].mean().sort_values(ascending=False)

print("Fraud Rate by Location:\n", fraud_by_location.head())

fraud_by_loyalty =
df.groupby("Customer_Loyalty_Tier") ["Fraud_Flag"].mean().sort_values(ascending=False)

print("\nFraud Rate by Loyalty Tier:\n", fraud_by_loyalty)

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report

X = df[["Customer_Age", "Purchase_Amount", "Footfall_Count"]]

y = df["Fraud_Flag"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

model = RandomForestClassifier()

```

```

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))

      precision    recall  f1-score   support

          0       0.97     1.00     0.98      620
          1       0.00     0.00     0.00       20

  accuracy                           0.97      640
 macro avg       0.48     0.50     0.49      640
weighted avg       0.94     0.97     0.95      640

```

```

fraud_counts = df["Fraud_Flag"].value_counts()

fraud_counts.plot.pie(autopct='%1.1f%%', labels=["Non-Fraud", "Fraud"],
colors=["skyblue", "salmon"])

plt.title("Fraud Proportion")

plt.ylabel("")

plt.show()

```

Fraud Proportion

Category	Percentage
Non-Fraud	96.9%
Fraud	3.1%

```

plt.figure(figsize=(8,5))

sns.histplot(df[df["Fraud_Flag"]==1]["Purchase_Amount"], color="red", kde=True,
label="Fraud")

```

```

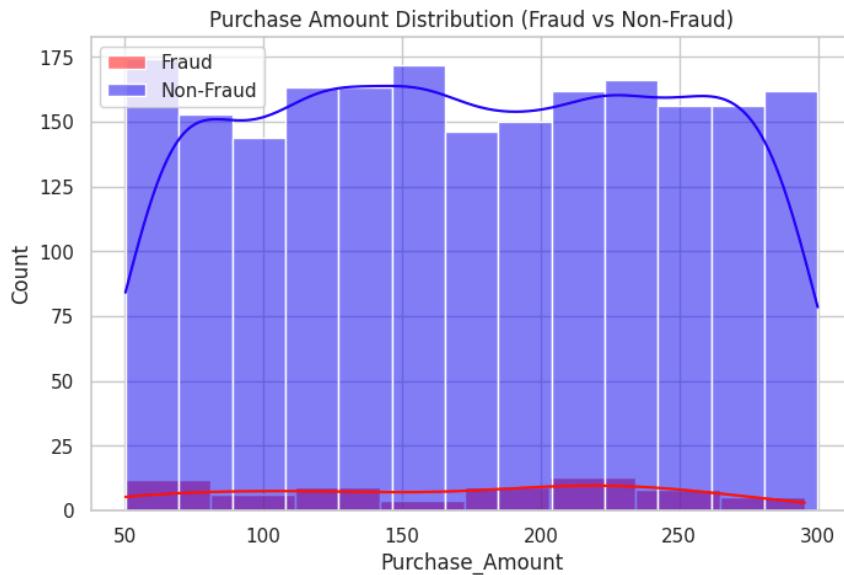
sns.histplot(df[df["Fraud_Flag"]==0]["Purchase_Amount"], color="blue",
kde=True, label="Non-Fraud")

plt.legend()

plt.title("Purchase Amount Distribution (Fraud vs Non-Fraud)")

plt.show()

```



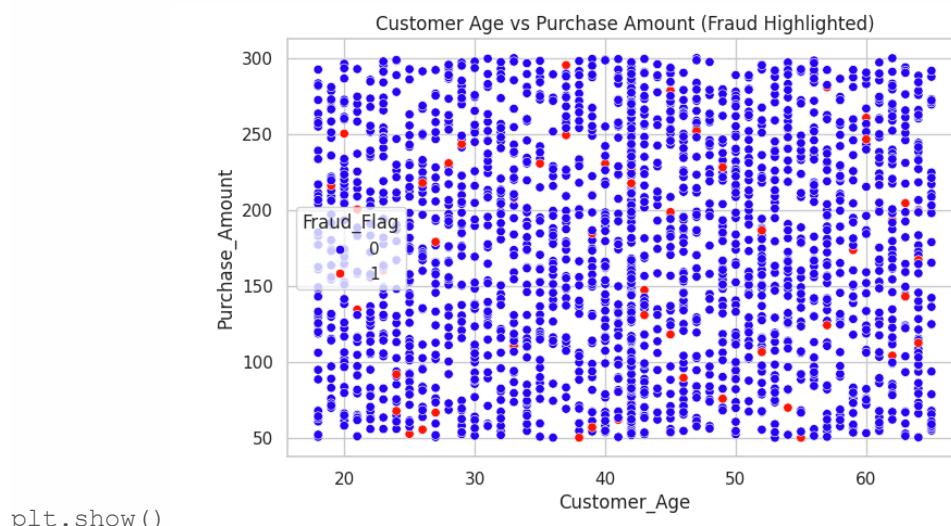
```

plt.figure(figsize=(8,5))

sns.scatterplot(x="Customer_Age", y="Purchase_Amount", hue="Fraud_Flag",
data=df, palette={0:"blue",1:"red"})

plt.title("Customer Age vs Purchase Amount (Fraud Highlighted)")

```



Conclusion

The experiment demonstrates that while Random Forest performs well on majority-class prediction, handling data imbalance is crucial for effective fraud detection. Proper preprocessing, feature engineering, and hyperparameter tuning significantly impact model performance.

Google Colab link: [co DMBI_EXP_Nupur](#)