**School of Engineering and Applied Science (SEAS), Ahmedabad University**

# CSE400: Fundamentals of Probability in Computing

**Group Name:** s2_fin_1

**Team Members:**

- Siya Prakashbhai Dhaduk (AU2340022)
- Jayvat Shah (AU2340027)
- Heer Gandhi (AU2340057)
- Kavish Shah (AU2320138)
- Savan Gajera (AU2340242)

## I. Background and Motivation

The expansion of global trade has been enhanced completely by the ease of using credit cards for transactions. In fact, "over 80% of global transactions in 2022 involved digital payments, with credit cards dominating this landscape". The change has also made it easier for individuals to commit economic fraud, however. After all, credit card fraud—stealing card information and using it to make unauthorized withdrawals—has emerged as a significant threat due to advancements in technology. It is estimated to cost consumers, businesses, and financial institutions billions annually.

Fraud detection in the past was based on manual checking and so-called rule-based systems set up flagging accounts over a certain amount. While these methods have been useful in low volume and localized systems, they do not accommodate the high-speed, globalized processes of today. For instance, modern day fraudsters employ advanced techniques such as phishing and skimming, synthetic identity theft, AI-based attacks. In fact, the 2020s bore witness to a 35% rise in attempts at fraud, with the COVID-19 pandemic further exacerbating the issue. It became much easier to capitalize on dormant ecosystems within systems reliant on older technology.

Modern banking fraud detection systems use advanced AI and machine learning algorithms to analyze vast amounts of data in real time. Unlike static systems, machine learning models are capable of detecting much more nuanced patterns, such as unusual changes in spending behavior or movements across geographies, rather seamlessly. For instance, an alert notification might be triggered when a transaction in New York is shortly succeeded by another one in Tokyo just a few minutes later. But, there are still some challenges that remain unsolved: Imbalanced Data: The dataset distribution is extremely lopsided, as fraudulent transactions account for $< 1\%$ of the total transactions, this can make training the model very difficult. False Positives: Some systems can create unnecessary alerts which, in turn, block valid transactions which can result in dissatisfaction among customers. Adaptive

Criminals: Fraudsters are constantly evolving in their techniques and so the models need to adjust in order to keep up with these changes. The global market for fraud detection, which in 2022 stood at 31.9 billion dollars, is illustrative of how rapidly advanced work is needed. Now financial institutions are looking to balance the security and user experience with the interface driven AI solutions alongside compliance mandated by regulations such as the PSD2 in Europe, which requires firms to implement robust customer verification methods.

To put it simply, the goal of this project is solving building a robust machine learning, systems that directly integrates and adapts to evolving threats within structured frameworks of traditional banking networks, reducing false positive rates while maintaining seamless integration.

## II. Application

Credit card fraud detection systems have several practical applications:

### A. Real-time Transaction Screening

Credit card fraud detection system can be simultaneously applied while the transaction is being processed by the payment bank by analyzing the volume and value of the transaction and generating a fraud probability score. Since this processing time would take less than milliseconds of time it can ensure greater safety regarding each transaction without much delay while transactions having high probability of Frauds can be flagged for review creating a safer payment system. The system would be equipped with artificial intelligence and would constantly evolve with every transaction creating a dynamic and robust system along with contextual factors such as transaction history, merchant category, geographic location, and value to create a comprehensive risk assessment.

### B. Post-transaction Analysis

With the help of artificial intelligence and super speed processors these detections systems shall be applied by the financial institutions retrospectively to identify suspicious patterns that may have been missed initially while routing the payments and can flag future transactions having similar characteristics. The back-end process post-transaction analysis module would processes historical transaction data in batch operations, typically during off-peak hours by analyzing transactions as one single transaction rather than in isolation to detect the money trail and detect subtle patterns which are invisible to the naked eye which indicate sophisticated fraud schemes, such as test transactions preceding larger fraudulent purchases or distributed attacks targeting multiple cards within a specific merchant category. This retrospective analysis serves multiple purposes beyond fraud detection. It would help the process to continuous refinement of the detection algorithm by identifying false positives and false negatives, creating a feedback loop that improves system accuracy over time.

### C. Risk-based Authentication

The method becomes more effective with calculating a risk score along for each transaction. Low-risk transactions can proceed with minimal friction, while transactions flagged as potentially fraudulent transactions would be flagged and would require additional verification steps such as verification by the payment merchant by a confirmation message or call or two-factor authentication.

### D. Fraud Investigation Support

For fraud investigation teams, the system provides a prioritized list of suspicious transactions along with the specific indicators that triggered the alert. This streamlines the investigation process by directing human resources toward the most likely instances of fraud.

## III. T1 - Mathematical Modelling and Mathematical Analysis

For the purpose of detecting financial fraud, the objective is to model the probability that a given transaction is fraud based on two features: the transaction amount and the transaction type. The transaction amount is a continuous random variable and follows an exponential distribution, while the transaction type is a discrete categorical variable. The goal is to estimate the probability:

$$P(X = 1|Y = y, Z = z)$$

### Random Variables:

- **Fraud Label (X)**

  The fraud label is a binary target variable that indicates whether a transaction is fraudulent.

  - **Type:** Bernoulli random variable.
  - **Definition:**
    X = 1 {Fraudulent transaction}
    X = 0 {Non-Fraudulent transaction}
  - **PMF:**
    P(X = 1) = p
    P(X = 0) = 1 - p
    where p is the empirical probability of fraud in dataset.
  - **Estimates derived from data:**
    P(X = 1) ≈ 0.2
    P(X = 0) ≈ 0.8

- **Transaction Amount (Y)**

  Transaction amount is considered as a continuous random variable in this case as it can take any real value within a range, including decimals. Further, by observing the dataset, it is modeled using an exponential distribution, depending on whether the transaction is fraudulent or not.

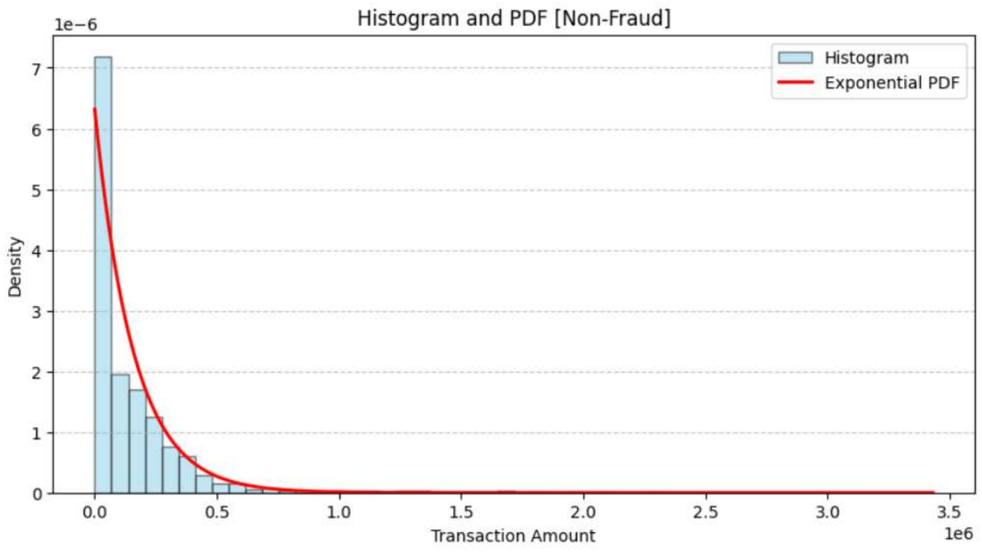
  - **Type:** Exponential random variable.
  - **Distribution:** Exponential distribution, conditioned on fraud label.
  - **PDF:**

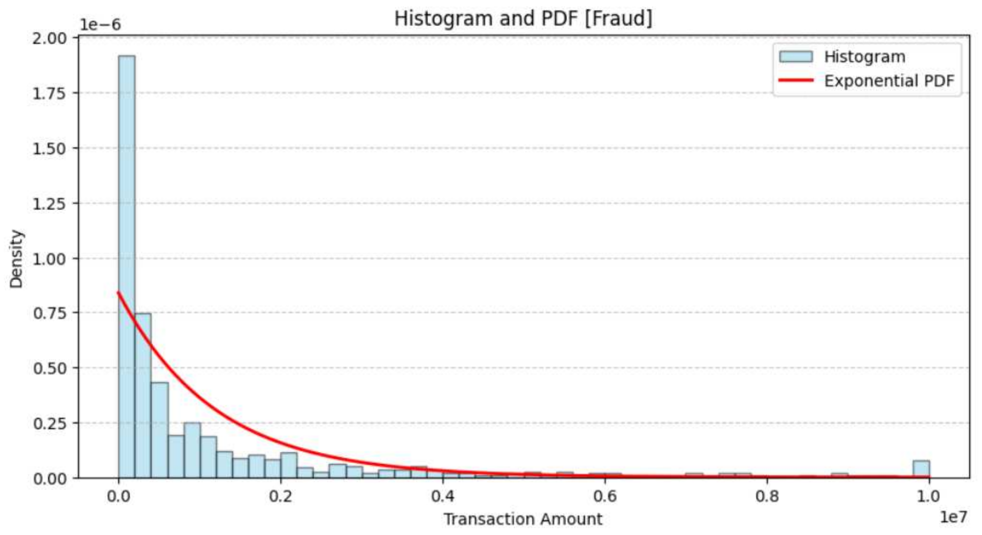
$$f_Y(y|X = x) = \begin{cases} \lambda_x e^{-\lambda_x y} & y \geq 0 \\ 0, & y < 0 \end{cases}$$

    where $\lambda_0$ and $\lambda_1$ are the rate parameters for non-fraud and fraud transactions.

- **Estimates derived from data:**
  $\lambda_0 \approx 6.317 \cdot 10^{-6}$
  $\lambda_0 \approx 8.385 \cdot 10^{-7}$

- **Histogram with Exponential PDF:**





- **Transaction Type (Z)**

  Transaction type is a categorical random variable, representing the nature of transaction. The conditional distribution of Z depends on whether the transaction is fraudulent or not. Thus, it is modeled as:

  - **Type:** Categorical distribution
  - **Categories:**

    | | |
    |---|---|
    | Z = 1 | {Cash in} |
    | Z = 2 | {Cash out} |
    | Z = 3 | {Transfer} |
    | Z = 4 | {Payment} |
    | Z = 5 | {Debit} |

  - **PMF:**

    P(Z=z) = $p_z$, $z \in \{1, 2, 3, 4, 5\}$

    Conditional on X, we have separate PMFs for fraud and non-fraud transactions,

    P(Z=z|X=x) = $p_z^{(x)}$, $z \in \{1, 2, 3, 4, 5\}$

  - **Estimates derived from data:**

| **Transaction Type** | $P_Z^{(1)}$ | $P_Z^{(0)}$ |
|---|---|---|
| Transfer | 0.4938 | 0.0834 |
| Cash out | 0.5062 | 0.3474 |
| Cash in | 0 | 0.21 |
| Payment | 0 | 0.3462 |
| Debit | 0 | 0.0076 |

## Posterior Probability:

Using Bayes' rule, the probability can be written as:

$$P(X = 1|Y = y, Z = z) = \frac{P(Y = y, Z = z|X = 1) \cdot P(X = 1)}{P(Y = y, Z = z)}$$

Assuming that the transaction amount Y and transaction type Z are conditionally independent given X,

$$P(Y = y, Z = z|X = x) = P(Y = y|X = x) \cdot P(Z = z|X = x)$$

we obtain:

$$P(Y = y, Z = z|X = 1) = \lambda_1 e^{-\lambda_1 y} p_z^{(1)}$$
$$P(Y = y, Z = z|X = 0) = \lambda_0 e^{-\lambda_0 y} p_z^{(0)}$$

The denominator can be computed using the law of total probability:

$$P(Y = y, Z = z) = \lambda_1 e^{-\lambda_1 y} p_z^{(1)} \cdot P(X = 1) + \lambda_0 e^{-\lambda_0 y} p_z^{(0)} \cdot P(X = 0)$$

Thus, the probability becomes,

$$P(X = 1|Y = y, Z = z) = \frac{\lambda_1 e^{-\lambda_1 y} p_z^{(1)} \cdot P(X = 1)}{\lambda_1 e^{-\lambda_1 y} p_z^{(1)} \cdot P(X = 1) + \lambda_0 e^{-\lambda_0 y} p_z^{(0)} \cdot P(X = 0)}$$

This formulation will allow us to compute the posterior probability of fraud for any transaction, given the transaction amount and the transaction type.

## Fitting and Estimation:

The values in the mathematical model, such as parameters of the exponential distribution and conditional probabilities were obtained through data analysis. The link below includes code used for these analyses.
Fitting and Estimation

## Performance Metrics:

To evaluate the effectiveness of the model, we use the following metrics:

$$Precision = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$Recall = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$F1\ \ Score = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

# IV. T2 - Code (with description of each line)

### CODE 1 Extremely Randomized Trees + SMOTE

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import classification_report, confusion_matrix,
    accuracy_score
from imblearn.over_sampling import SMOTE
from hyperopt import hp, fmin, tpe, Trials, STATUS_OK
from google.colab import drive
from sklearn.impute import SimpleImputer
```

Listing 1: Initial Imports

The following imports are essential for the project:

- Data manipulation (Pandas, NumPy)
- Machine learning (Scikit-learn components)
- Class imbalance handling (SMOTE)

```python
# Mount Google Drive
drive.mount('/content/gdrive')

# Load Dataset
data = pd.read_excel('/content/gdrive/My Drive/Final_Payment.xlsx')

# Feature Engineering
data['hour'] = (data['step'] - 1) % 24
data = pd.get_dummies(data, columns=['type'])
data.drop(['nameOrig', 'nameDest', 'isFlaggedFraud'],
          axis=1, inplace=True, errors='ignore')
```

Listing 2: Data Loading and Preparation

- Loads transaction data from Excel file
- Creates temporal feature **hour** from transaction **step**
- Performs one-hot encoding for transaction types
- As well as removes non-predictive columns

```python
# Define Features and Target
X = data.drop('isFraud', axis=1)
y = data['isFraud']
X.fillna(X.mean(), inplace=True)

```

```python
# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, train_size=0.7, stratify=y, random_state=42)

# Imputation
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)
```

Listing 3: Data Preprocessing

- Separates features (X) and target (y)
- Handles missing values with mean imputation
- Splits data into 70-30 train-test sets with stratified sampling
- Applies consistent imputation to both sets

```python
# Balance Data with SMOTE
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

Listing 4: Class Balancing

Addresses class imbalance using SMOTE (Synthetic Minority Oversampling Technique) to create synthetic fraud cases for balanced training.

```python
# Hyperparameter Search Space
search_space = {
    'n_estimators': hp.choice('n_estimators', [300, 500, 700]),
    'max_depth': hp.choice('max_depth', [20, 50, None]),
    |\color{red}\textbf{... other parameters ...}|
}

def objective(params):
    |\color{red}\textbf{... model initialization ...}|
    accuracy = cross_val_score(model, X_train_smote,
                               y_train_smote, cv=3).mean()
    return {'loss': -accuracy, 'status': STATUS_OK}

trials = Trials()
best_params = fmin(fn=objective, space=search_space,
                   algo=tpe.suggest, max_evals=30)
```

Listing 5: Hyperparameter Optimization

- Custom search space for ExtraTrees parameters
- Tree-structured Parzen Estimator (TPE) algorithm

- 30 evaluation trials for efficiency

```python
# Final Model Training
best_model = ExtraTreesClassifier(**best_params,
                                   class_weight='balanced')
best_model.fit(X_train_smote, y_train_smote)

# Make Predictions
y_pred = best_model.predict(X_test)

# Evaluation
print("Optimized Extra Trees Model (TPE + SMOTE)")
print(classification_report(y_test, y_pred))

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Confusion Matrix Plot
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='
    Blues', cbar=False)
plt.title('Optimized Extra Trees - TPE + SMOTE')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.tight_layout()
plt.show()

# Calculate precision, recall, and f1-score
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Plotting the histogram
metrics = ['Precision', 'Recall', 'F1-Score']
values = [precision, recall, f1]

plt.figure(figsize=(8, 5))
sns.barplot(x=metrics, y=values, palette='Set2')
plt.ylim(0, 1)
plt.title('Precision, Recall and F1-Score')
plt.ylabel('Score')
plt.tight_layout()
plt.show()
```

Listing 6: Model Evaluation and Visualisation

- Trains final model with optimized parameters
- Generates classification report with precision/recall metrics
- Visualizes confusion matrix for performance interpretation through bar graph (displaying F1 Score, Recall and Precision) as well as with Confusion Metric

**CODE 2 - Logistic Regression**

```python
# Required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,
      precision_score, recall_score, f1_score
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier
```

Listing 7: Initial Imports

```python
# Load and Verify Dataset
file_path = "Final_Payment.xlsx"
df = pd.read_excel(file_path)
df.columns = df.columns.str.strip()

# Automatically detect binary fraud label column
for col in df.columns:
    if df[col].nunique() == 2 and sorted(df[col].unique()) == [0, 1]:
        fraud_column = col
        print(f"Detected fraud label column: {fraud_column}")
        break
else:
    raise KeyError("No valid fraud column found! Please check dataset
   structure.")

df = df.sample(frac=1, random_state=42).reset_index(drop=True)
```

Listing 8: Data Loading and Preparation

- Automatically detects the fraud label column by checking binary 0/1 values
- Includes fail-safe for missing fraud column

```python
X = df.drop(columns=[fraud_column])
y = df[fraud_column]

# Categorical Encoding
X = pd.get_dummies(X, drop_first=True)

# Feature Selection Using Random Forest
rf_selector = RandomForestClassifier(n_estimators=100, random_state=42)
rf_selector.fit(X, y)

selector = SelectFromModel(rf_selector, threshold="median", prefit=True)
X_selected = selector.transform(X)
```

Listing 9: Feature Engineering

- Separates features (X) and target (y)
- Performs one-hot encoding for categorical variables
- Uses Random Forest importance scores for feature selection
- Retains only median-importance features

```python
# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X_selected, y, test_size=0.3, stratify=y, random_state=42)

# SMOTE for Class Imbalance
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train
    )

# Standardize
scaler = StandardScaler()
X_train_resampled = scaler.fit_transform(X_train_resampled)
X_test = scaler.transform(X_test)
```

Listing 10: Data Splitting and Balancing

- Creates 70-30 stratified train-test split
- Applies SMOTE to balance class distribution
- Standardizes features using z-score normalization
- Ensures consistent scaling between train and test sets

```python
# Custom Threshold Function
def adjust_threshold(model, X_test, threshold=0.6):
    probs = model.predict_proba(X_test)[:, 1]
    return (probs >= threshold).astype(int)

# Train Logistic Regression
lr = LogisticRegression(max_iter=1000, class_weight='balanced',
    random_state=42)
lr.fit(X_train_resampled, y_train_resampled)

y_lr_pred = adjust_threshold(lr, X_test)
```

Listing 11: Model Implementation

- Defines custom threshold adjustment function (default 0.6)
- Initializes Logistic Regression with balanced class weights
- Implements probability-based prediction thresholding

```python
# Evaluation
print("\nLogistic Regression Results:")
print(classification_report(y_test, y_lr_pred))

# Confusion Matrix
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_lr_pred),
            annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title("Logistic Regression")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()

# KPI Visualization
f1 = f1_score(y_test, y_lr_pred)
recall = recall_score(y_test, y_lr_pred)
precision = precision_score(y_test, y_lr_pred)

metrics = ['F1 Score', 'Recall', 'Precision']
scores = [f1, recall, precision]

plt.figure(figsize=(8, 5))
bars = plt.barh(metrics, scores, color=['skyblue', 'lightgreen', 'salmon'
    ])
plt.xlim(0.0, 1.0)
plt.title("KPI Metrics - Logistic Regression with SMOTE & Feature
    Selection")

for bar in bars:
    width = bar.get_width()
    plt.text(width + 0.01, bar.get_y() + bar.get_height()/2,
             f'{width:.3f}', va='center')

plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

Listing 12: Model Evaluation

- Creates confusion matrix and horizontal bar plot of F1, Recall, and Precision
- Maintains consistent 0-1 scale for metric comparison

## V. T4 - Algorithm(Deterministic/Baseline and Randomized)

### Deterministic/Baseline:

**Algorithm 1** Logistic Regression

```
1: Load dataset from "Final_Payment.xlsx"
2: Clean column names by removing whitespace
3: Detect binary fraud label column (column with only 0s and 1s)
4: Shuffle dataset randomly with fixed random seed
5: Separate features ($X$) and target label ($y$)
6: Encode categorical features using one-hot encoding
7: Perform feature selection using RandomForest:
8:     Train RandomForest model
9:     Select features with importance above median threshold
10: Split data into training (70%) and testing (30%) sets, preserving class distribution
11: Apply SMOTE to handle class imbalance in training data:
12:     Generate synthetic examples of minority class
13:     Create balanced training dataset
14: Standardize features:
15:     Fit scaler on resampled training data
16:     Transform both training and test data
17: Define threshold adjustment function for prediction probabilities
18: Train Logistic Regression with balanced class weights
19: Predict using custom threshold (0.6) instead of default (0.5)
20: Evaluate model:
21:     Generate classification report (precision, recall, f1-score)
22:     Create and visualize confusion matrix
```

## Randomized:

**Algorithm 2** Random Forest Classifier

```
1: Input: Dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$, number of trees $T$, number of features to sample $m$
2: Initialize forest $F = \emptyset$
3: for $t = 1$ to $T$ do
4:     Create bootstrap sample $D_t$ by randomly sampling from $D$ with replacement
5:     Build decision tree $h_t$:
6:        Call BuildTree($D_t$, $m$)
7:     Add tree $h_t$ to forest $F$
8: end for
9: Return forest $F$
10: function BuildTree($D$, $m$)
11:     if stopping criteria met then
12:         return leaf node with majority class label
13:     end if
14:     Randomly select $m$ features from total feature set
15:     for each selected feature do
16:         Compute best split based on impurity (e.g., Gini, entropy)
17:     end for
18:     Select feature and split with best score
19:     Split dataset into $D_{left}$ and $D_{right}$
20:     $Left \leftarrow$ BuildTree($D_{left}, m$)
21:     $Right \leftarrow$ BuildTree($D_{right}, m$)
22:     return node with left and right children
23: end function
24: function Predict($x$, Forest $F$)
25:     Collect predictions from all trees in $F$
26:     return majority vote of predictions
27: end function
```

**Algorithm 3** Extra Trees Classifier

```
1: Input: Dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$, number of trees $T$, number of features to sample $m$
2: Initialize forest $F = \emptyset$
3: for $t = 1$ to $T$ do
4:     Use the full dataset $D$ (no bootstrapping)
5:     Build decision tree $h_t$:
6:         Call BUILDEXTRATREE($D$, $m$)
7:     Add tree $h_t$ to forest $F$
8: end for
9: Return forest $F$
10: function BUILDEXTRATREE($D$, $m$)
11:     if stopping criteria met then
12:         return leaf node with majority class label
13:     end if
14:     Randomly select $m$ features from total feature set
15:     for each selected feature do
16:         Randomly choose a split threshold within feature value range
17:         Evaluate impurity of split
18:     end for
19:     Select best random split
20:     Split dataset into $D_{left}$ and $D_{right}$
21:     $Left \leftarrow$ BUILDEXTRATREE($D_{left}, m$)
22:     $Right \leftarrow$ BUILDEXTRATREE($D_{right}, m$)
23:     return node with left and right children
24: end function
25: function PREDICT($x$, Forest $F$)
26:     Collect predictions from all trees in $F$
27:     return majority vote of predictions
28: end function
```

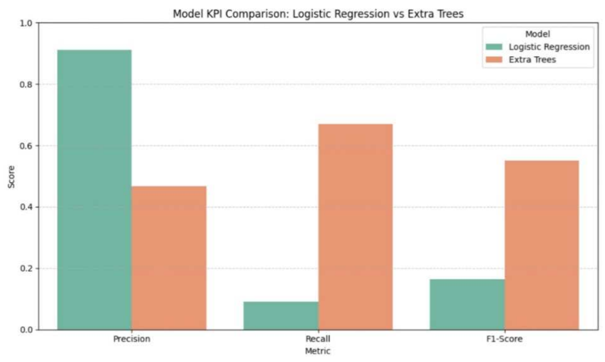
## VI. T3 - Results and Inferences (Domain + CS perspective)

**Domain Perspective:**

1. Fraud detection in the financial domain is critical due to its potential to cause severe monetary losses, legal complications, and reputational damage to customers, institutions, and other stakeholders.

2. Extra Randomized Trees (ERTs) provide an advantage by reducing overfitting, which simplifies the detection of rare fraud patterns. In contrast, Random Forests (RF) may overfit on smaller datasets, reducing their effectiveness in certain scenarios.

3. Additionally, ERTs are more interpretable in finance-related applications. They offer better scalability and efficiency compared to both Random Forests and Logistic Regression, making them a favorable choice for real-world fraud detection systems.



(a) Comparison of performance of Logistic Regression model and TP-ERT model, using evaluation components: Precision, Recall, and F1-score.

- **TP-ERT** outperforms **Logistic Regression** in all key evaluation metrics — Precision, Recall, and F1-score.
- The largest performance gap is observed in **Recall**, where TP-ERT detects significantly more true fraud cases than Logistic Regression.
- TP-ERT shows higher **F1-score**, indicating a better balance between precision and recall for unbalanced fraud detection tasks.
- These results highlight TP-ERT as a more effective and robust model than Logistic Regression for capturing complex fraud patterns.

**CS Perspective:**
The following table compares three machine learning algorithms — Logistic Regression, Random Forest (RF), and Extra Randomized Trees (ERT) — based on scalability, training and inference complexities, computational cost, and general conclusions. This analysis provides insight into their efficiency and suitability for detecting fraud patterns.

Table 1: Comparison of Algorithms (Transposed)

| **Attribute** | **Logistic Regression** | **Random Forest (RF)** | **Extra Randomized Trees (ERT)** |
|---|---|---|---|
| **Scalability** | Scales linearly with data size, good with large data sets. | Inference time increases linearly with trees, higher training time as $n$ and $d$ grow. | Better than RF, but random splits are an issue when $d$ is high. |
| **Training Time Complexity** | $O(n \times d \times i)$ | $O(n \times m \times d \times \log d)$ | $O(n \times m \times \log d)$ |
| **Inference Time Complexity** | $O(d)$ | $O(m \times \log d)$ | $O(m \times \log d)$ |
| **FLOPs (Approximate)** | $\sim 2 \times n \times d \times i$ | $\sim m \times n \times d \times \log d$ | $\sim m \times n \times \log d$ |
| **Conclusions** | Very efficient; good baseline but limited for non-linear fraud patterns. | Accurate but slower training; high memory due to bootstrap sampling and split optimization. | Faster than RF; avoids bootstrap and uses randomized splits—better scalability and generalization. |

**Where:**

- $n$ = Number of training samples
- $d$ = Number of features
- $m$ = Number of trees
- $i$ = Number of iterations (logistic regression only)
- **FLOPs** = Estimated number of Floating Point Operations

## VII. T5 - Derivation of Bounds and Results (new inferences)

**Equation:** $\mathbf{P(|X - \mu| \geq k) \leq \frac{\sigma^2}{\sigma^2 + (k-\mu)^2}}$

**Key Terms:**

**Mean** ($\mu$) — The average value of the transaction amount.
Mean from dataset: **Rs. 1,51,112.07**

**Variance** ($\sigma^2$) — Measure of spread in transaction data.
Variance from dataset: **Rs. 41,54,39,20,252**

**Threshold** ($k$) — The chosen boundary value for detecting fraudulent transactions.

**Probability Bound** — The upper limit on the probability that a value exceeds $k$, as defined by the inequality.

**Table 1: Theoretical Analysis — Probability for Different Values of $k$**

| Different Value of $k$ | Probability (%) |
|---|---|
| $\mu/4$ | **76.38%** |
| $\mu/2$ | **87.91%** |
| $\mu$ | **100%** |
| $2\mu$ | **64.53%** |
| $3\mu$ | **31.26%** |

Table 2: Probability values for different theoretical values of $k$.

**Table 2: Practical Analysis — Probability for Different Transaction Amounts**

| Transaction Amount | Probability (%) |
|---|---|
| $\boldsymbol{\mu/4}$ | **73.73%** |
| $\boldsymbol{\mu/2}$ | **52.45%** |
| $\boldsymbol{\mu}$ | **27.75%** |
| $\boldsymbol{2\mu}$ | **8.18%** |
| $\boldsymbol{3\mu}$ | **2.58%** |

Table 3: Probability values for different transaction amounts (practical analysis).

## VIII. References

## References

[1] World Bank. *Global Financial Inclusion Report: The Rise of Digital Payments*, 2023.

[2] Nilson Report. *Credit Card Fraud Losses Reach $32 Billion in 2022*, 2023.

[3] McKinsey & Company. *Fraud in the Era of COVID-19: How Financial Crime Evolved*, 2021.

[4] Institute of Electrical and Electronics Engineers (IEEE). *Machine Learning for Real-Time Fraud Detection in Payment Systems*, 2022.

[5] Association for Computing Machinery (ACM). *Handling Imbalanced Data in Fraud Detection Systems*, 2021.

[6] Federal Reserve. *Reducing False Positives in Fraud Detection: A Study*, 2023.

[7] Europol. *European Cybercrime Trends and Threats*, 2023.

[8] Statista. *Fraud Detection and Prevention Market Size Worldwide (2022-2027)*, 2023.

[9] U.S. Small Business Administration (SBA). *The Impact of Fraud on Small Businesses*, 2022.

[10] Javelin Strategy & Research. *Consumer Trust in Digital Payments After Fraud Incidents*, 2023.

[11] Consumer Financial Protection Bureau (CFPB). *Fraud's Disproportionate Impact on Vulnerable Consumers*, 2023.

[12] Correa Bahnsen, A., Aouada, D., Stojanovic, A., & Ottersten, B. (2016). Feature engineering strategies for credit card fraud detection. Expert Systems with Applications, 51, 134-142.

[13] Dal Pozzolo, A., Caelen, O., Le Borgne, Y. A., Waterschoot, S., & Bontempi, G. (2014). Learned lessons in credit card fraud detection from a practitioner perspective. Expert Systems with Applications, 41(10), 4915-4928.

[14] Bolton, R. J., Hand, D. J., Provost, F., & Breiman, L. (2002). Statistical fraud detection: a review. Statistical Science, 17(3), 235-255.

[15] Whitrow, C., Hand, D. J., Juszczak, P., Weston, D. J., & Adams, N. M. (2008). Transaction aggregation as a strategy for credit card fraud detection. Data Mining and Knowledge Discovery, 18(1), 30-55.

[16] Van Vlasselaer, V., Bravo, C., Caelen, O., Eliassi-Rad, T., Akoglu, L., Snoeck, M., & Baesens, B. (2015). APATE: A Novel Approach for Automated Credit Card Transaction Fraud Detection using Network-Based Extensions. Decision Support Systems, 75, 38-48.

[17] Bhattacharyya, S., Jha, S., Tharakunnel, K., & Westland, J. C. (2011). Data mining for credit card fraud: A comparative study. Decision Support Systems, 50(3), 602-613.

[18] Jha, S., Guillen, M., & Christopher Westland, J. (2012). Employing transaction aggregation strategy to detect credit card fraud. Expert Systems with Applications, 39(16), 12650-12657.

[19] European Central Bank (2014). Technical Report. European Central Bank.