1. INTRODUCTION

1.1 Project Overview

In recent years, the rise of online transactions has also led to an increase in fraudulent activities. To combat this, fraud detection systems have become essential in safeguarding financial institutions and their customers. This project focuses on developing an online payment fraud detection system using machine learning techniques. The system is designed to detect and predict fraudulent transactions based on various transaction attributes. It utilizes a BalancedRandomForestClassifier model to achieve high accuracy in identifying fraudulent activities. The project is implemented using Flask for the web interface, and the model is deployed using the Render platform.

1.2 Purpose

The primary purpose of this project is to develop a robust and efficient fraud detection system that can accurately predict fraudulent transactions in real-time. This system aims to provide financial institutions with a tool to enhance their security measures, reduce financial losses, and protect their customers from fraudulent activities. The project also aims to demonstrate the application of machine learning models in real-world scenarios and provide a framework for further improvements and adaptations.

2. LITERATURE SURVEY

2.1 Existing Problem

Online payment fraud is a significant issue faced by financial institutions and online merchants. With the increasing volume of online transactions, fraudsters continuously develop new techniques to bypass security measures. Traditional fraud detection methods, such as rule-based systems, often fail to detect sophisticated fraudulent activities and generate a high number of false positives. This not only leads to financial losses but also affects the customer experience. There is a pressing need for advanced fraud detection systems that can learn from historical data and adapt to new fraud patterns.

2.2 References

1. Bolton, R. J., & Hand, D. J. (2002). Statistical fraud detection: A review. Statistical

Science, 235-249.
2. Phua, C., Lee, V., Smith, K., & Gayler, R. (2005). A comprehensive survey of data mining-based fraud detection research. arXiv preprint arXiv:1009.6119.
3. Ngai, E. W., Hu, Y., Wong, Y. H., Chen, Y., & Sun, X. (2011). The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. Decision Support Systems, 50(3), 559-569.
4. Zareapoor, M., & Shamsolmoali, P. (2015). Application of credit card fraud detection: Based on bagging ensemble classifier. Procedia Computer Science, 48, 679-685.

2.3 Problem Statement Definition

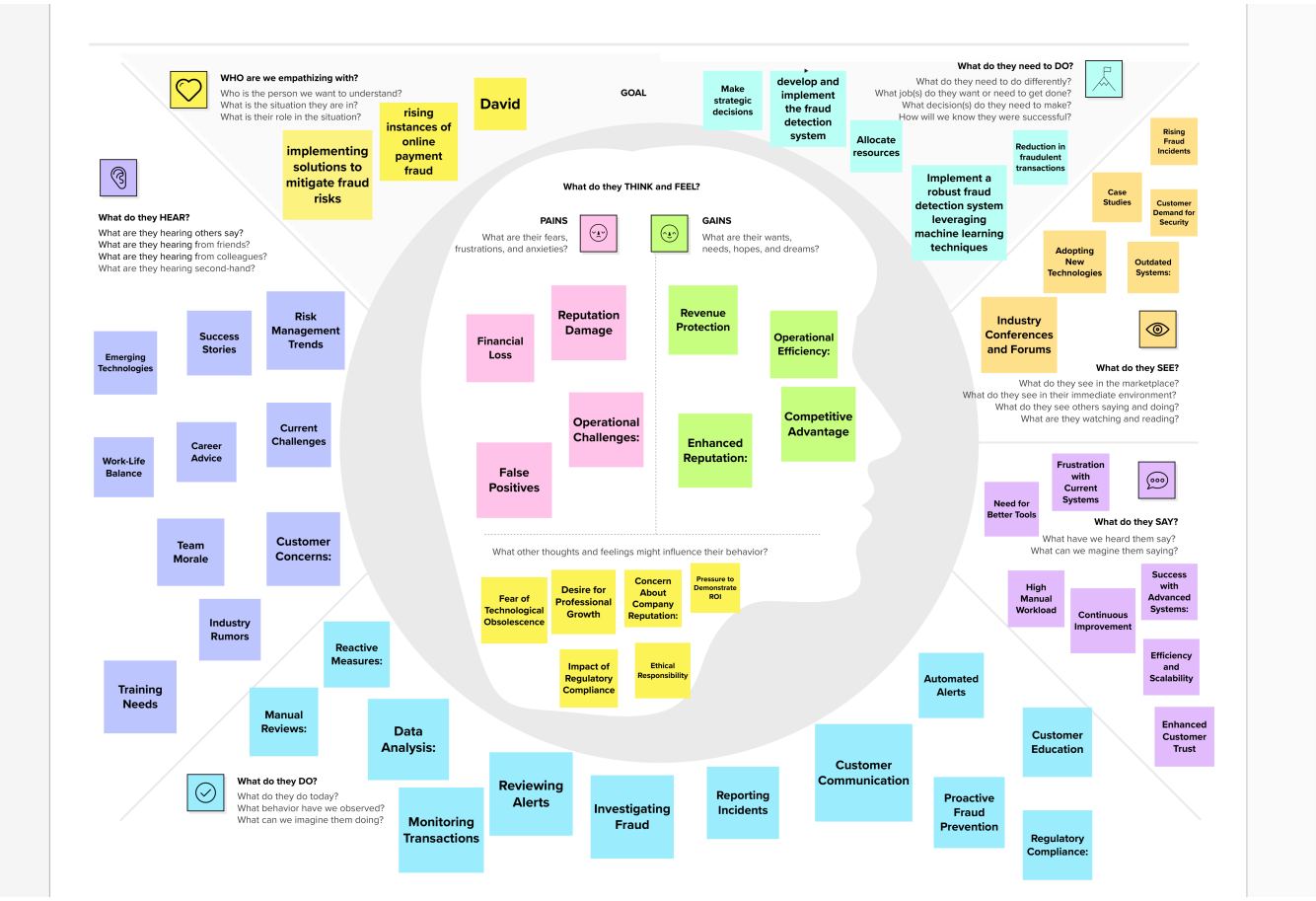
The problem addressed in this project is the detection of fraudulent online payment transactions. The goal is to develop a machine learning model that can accurately predict whether a transaction is fraudulent based on various transaction attributes. The system should be able to handle large volumes of data, provide real-time predictions, and adapt to new fraud patterns over time. The key challenge is to minimize false positives while ensuring high detection rates for actual fraudulent activities. This project aims to provide a solution that can be integrated into existing financial systems to enhance their fraud detection capabilities.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas

The Empathy Map Canvas is a tool used to gain deeper insight into the needs and experiences of the end users of the fraud detection system. It helps in understanding the users' perspectives and in designing solutions that address their specific pain points. The canvas typically includes sections for what users think and feel, see, say and do, hear, and their pain points and gains.

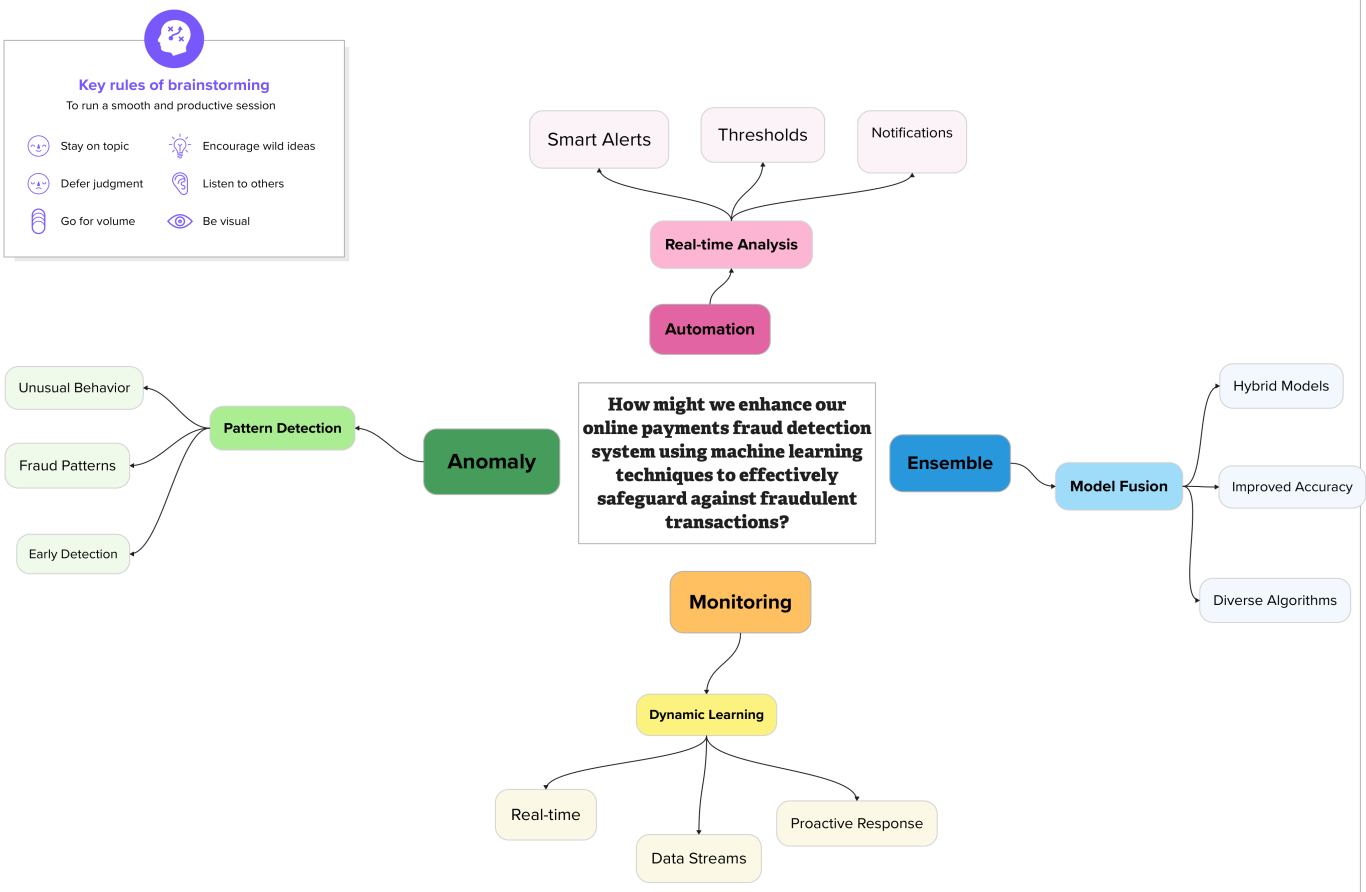
In this project, the Empathy Map Canvas was used to identify the key stakeholders involved in the online payment process, including customers, financial institutions, and online merchants. By mapping out their experiences and challenges, the project team could better understand the issues they face with current fraud detection systems. This understanding guided the design and development of a more user-centric and effective fraud detection solution.



3.2 Ideation & Brainstorming

The ideation and brainstorming phase is crucial for generating innovative ideas and solutions to the problem of online payment fraud. During this phase, various team members contribute their knowledge and expertise to identify potential approaches and technologies that could be

employed in the fraud detection system.



4. REQUIREMENT ANALYSIS

4.1 Functional Requirements

Functional requirements define the specific behaviors and functionalities that the fraud detection system must have to meet the needs of its users. These requirements ensure that the system performs as expected and provides value to its stakeholders. The main functional requirements for this project include:

1. **User Registration:**
   - Users must be able to register for the application by entering their email, password, and confirming the password.
   - Users should receive a confirmation email upon successful registration.
2. **User Login:**
   - Users must be able to log into the application using their registered email and password.
3. **Transaction Input:**

- Users must be able to input transaction details, including amount, old balance, new balance, and transaction type.

4. **Fraud Prediction:**
    - The system should be able to predict the likelihood of a transaction being fraudulent based on the input data.
    - The prediction should be provided using both the initial and new machine learning models.
5. **Data Upload:**
    - Users should be able to upload data files for preprocessing and training new models.
6. **Dashboard:**
    - The system should display a dashboard showing predictions and other relevant information to the user.
7. **Model Training:**
    - The system should preprocess the uploaded data, train a new machine learning model, and save the model for future use.
8. **Error Handling:**
    - The system must handle errors gracefully and provide meaningful error messages to the users.

4.2 Non-Functional Requirements

Non-functional requirements define the overall qualities and attributes of the system. They ensure that the system is reliable, efficient, and easy to use. The main non-functional requirements for this project include:

1. **Performance:**
    - The system should provide fraud predictions in real-time, with minimal latency.
    - Model training and data preprocessing should be completed within a reasonable timeframe.
2. **Scalability:**
    - The system should be able to handle a large number of simultaneous users and transactions without degradation in performance.
3. **Reliability:**
    - The system should have high availability and should be resilient to failures.
    - The system should ensure data integrity and consistency.
4. **Usability:**
    - The user interface should be intuitive and user-friendly, allowing users to easily navigate and use the system.
    - The system should provide clear and concise feedback to users.
5. **Security:**

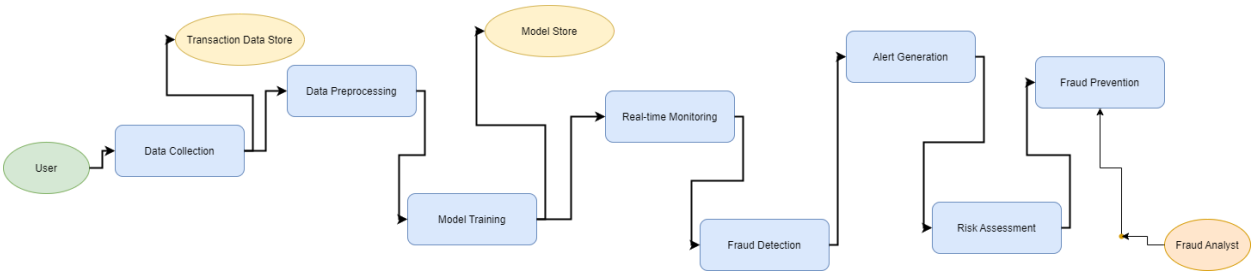
- The system should ensure that user data is stored securely and is protected against unauthorized access.
- User authentication and authorization mechanisms should be robust.

6. **Maintainability:**
   - The system should be designed in a modular way, allowing for easy updates and maintenance.
   - The codebase should be well-documented to facilitate future development and troubleshooting.
7. **Compliance:**
   - The system should comply with relevant industry standards and regulations, ensuring that all data processing and storage practices are legal and ethical.

By clearly defining these functional and non-functional requirements, the project ensures that the fraud detection system is both effective and reliable, providing significant value to its users.

5.1 Data Flow Diagrams & User Stories

Data Flow Diagrams (DFDs): Data Flow Diagrams (DFDs) are used to represent the flow of data within the system. They help visualize how data moves through the system, the processes it undergoes, and how it interacts with external entities.



User Stories: User stories are short, simple descriptions of a feature or functionality from the perspective of the user. They help ensure that the development team understands the user's needs and builds features that provide value.

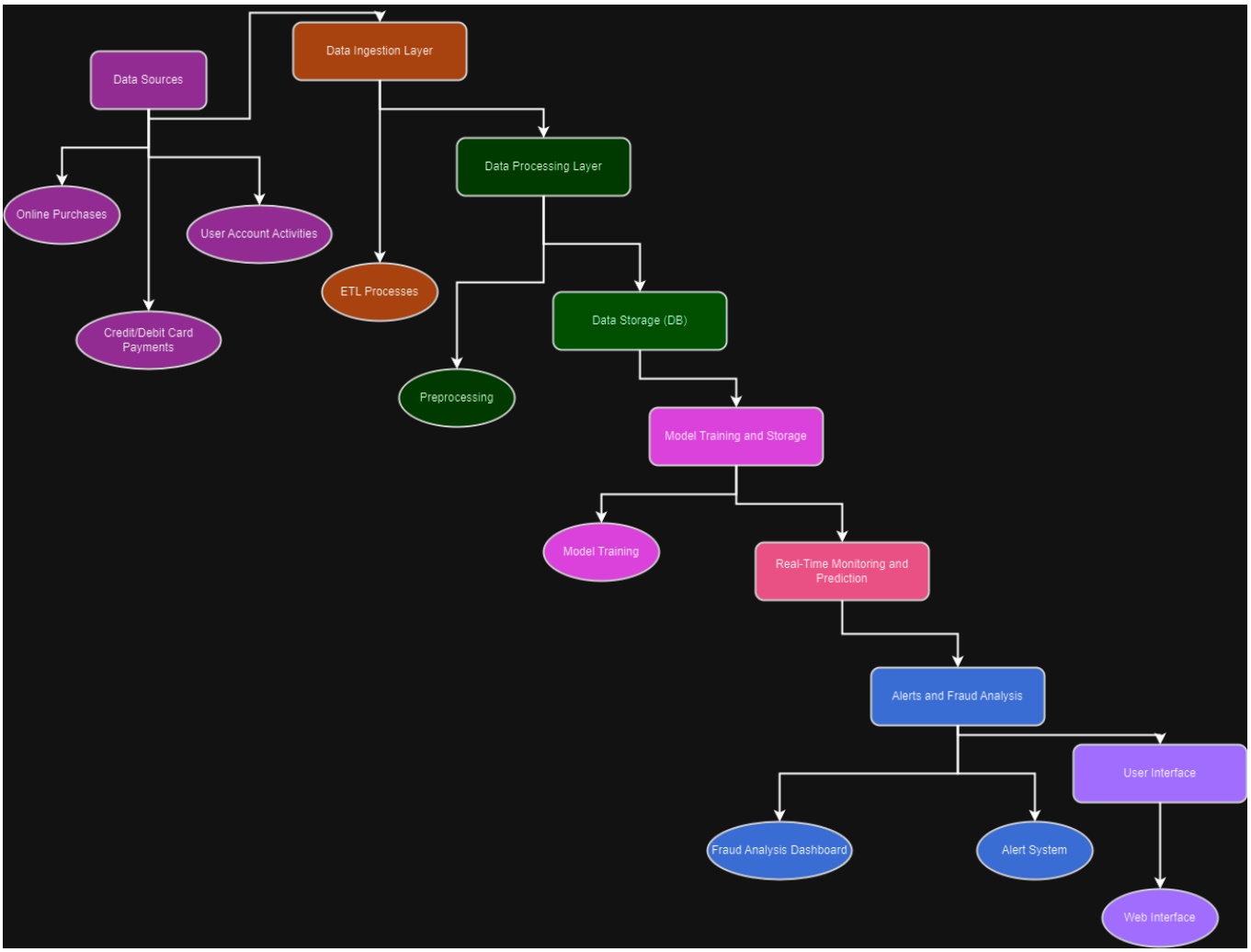
User Story ID	User Role	User Story	Acceptance Criteria
US1	Customer	As a customer, I want to be able to complete transactions quickly and securely	Transactions should be processed within 3 seconds with an error rate of less than 1%
US2	Fraud Analyst	As a fraud analyst, I want to receive alerts for suspicious transactions	Alerts should be generated within 2 seconds of transaction completion
US3	Customer	As a customer, I want to receive a notification if my transaction is flagged as suspicious	Notifications should be sent within 1 second of the transaction being flagged
US4	Fraud Analyst	As a fraud analyst, I want to see detailed information about flagged transactions	Detailed information should include transaction amount, location, time, and reason for flagging
US5	Admin	As an admin, I want to update the machine learning model with new data	The model should be updated without interrupting real-time transaction monitoring
US6	Customer	As a customer, I want assurance that my transaction data is stored securely	Transaction data should be encrypted and stored in a secure database
US7	Fraud Analyst	As a fraud analyst, I want to manually override the system's decision	Fraud analysts should be able to mark transactions as legitimate or fraudulent through an override

					mechanism
US8		Admin		As an admin, I want to monitor the performance of the fraud detection system	Performance metrics such as detection accuracy, false positives, and false negatives should be available
US9		Customer		As a customer, I want my past transactions to be considered in fraud detection	The system should use historical transaction data to improve the accuracy of fraud detection
US10		Fraud Analyst		As a fraud analyst, I want to provide feedback on false positives and false negatives	Analysts should be able to log feedback on the system's decisions to improve the model
US11		Admin		As an admin, I want to ensure the system scales with increased transaction volume	The system should handle up to 10,000 transactions per second without performance degradation
US12		Customer		As a customer, I want to see a summary of my transaction history	Customers should be able to view a summary of their transactions over the past month
US13		Fraud Analyst		As a fraud analyst, I want access to visualizations of transaction data	Visualizations should include graphs and charts to help identify trends and patterns in transaction data
US14		Admin		As an admin, I want to configure alert thresholds for different types of transactions	The system should allow configurable thresholds for various transaction types and amounts
US15		Customer		As a customer, I want my sensitive information to be protected during	The system should comply with industry standards such as PCI-DSS for protecting

				transactions	sensitive customer information	

5.2 Solution Architecture

The solution architecture outlines the high-level structure of the system, including the main components and their interactions.



By designing a robust solution architecture, we ensure that the fraud detection system is scalable, reliable, and easy to maintain, providing a seamless experience for users.

6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture

The technical architecture of the fraud detection system is designed to ensure high performance, scalability, and security. The architecture leverages a combination of modern technologies and best practices to create a robust and efficient system.

Components & Technologies:

S.No	Component	Description	Technology
1	User Interface	How user interacts with application	HTML, CSS, Bootstrap, JavaScript
2	Application Logic-1	Main application logic for fraud detection	Python, Flask
3	Machine Learning	Fraud detection model	BalancedRandomForestClassifier, scikit-learn
4	Data Processing	Data preprocessing and feature engineering	pandas, numpy, imbalanced-learn
5	Database	Data storage and retrieval	Local CSV file (used in the example)
6	Model Persistence	Saving and loading trained models	joblib
7	Deployment Platform	Platform for deploying the web application	Render
8	Logging	Logging application activities and errors	Python logging module

Application Characteristics:

S.No	Characteristics	Description	Technology
1	Open-Source Frameworks	List the open-source frameworks used	Flask, scikit-learn, imbalanced-learn
2	Security Implementations	List all the security/access controls implemented, use of firewalls etc.	HTTPS via Render, form validation

3	Scalable Architecture	Justify the scalability of architecture (3-tier, Micro-services)	Flask can be scaled with WSGI servers like Gunicorn
4	Availability	Justify the availability of application (e.g., use of load balancers, distributed servers)	Render provides scalable cloud infrastructure
5	Performance	Design consideration for the performance of the application (number of requests per sec, use of Cache, use of CDN's) etc.	Render's cloud platform, caching layers like Redis if needed

The system's architecture is structured to ensure ease of maintenance and flexibility, allowing for future enhancements and scalability.

6.2 Sprint Planning & Estimation

Sprint planning involves defining the user stories and tasks to be completed within a specific sprint. Each sprint is planned to deliver a set of features incrementally, ensuring steady progress towards the final goal.

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task Description	Story Points	Priority
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task Description	Story Points	Priority
Sprint-1	Registration	USN-2	As a user, I will receive a confirmation email once I have registered for the application.	1	High
Sprint-1	Registration	USN-4	As a user, I can register for the application through Gmail.	2	Medium
Sprint-1	Login	USN-5	As a user, I can log into the application by entering email & password.	1	High
Sprint-2	Dashboard	USN-6	As a user, I can view the dashboard showing predictions.	3	Medium
Sprint-2	Upload	USN-7	As a user, I can upload data files for preprocessing and training.	3	High

The total story points for each sprint have been estimated based on the complexity and priority of the tasks.

6.3 Sprint Delivery Schedule

The project is divided into three sprints, each lasting for 14 days. The sprints are planned to ensure the project is completed in an evenly distributed manner from May to July.

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed	Sprint Release Date (Actual)
Sprint-1	6	14 Days	01-May-24	14-May-24	6	14-May-24
Sprint-2	6	14 Days	15-May-24	28-May-24	6	28-May-24

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed	Sprint Release Date (Actual)
Sprint-3	6	14 Days	29-May-24	11-Jun-24	6	11-Jun-24

This schedule ensures a structured approach to project completion, allowing for regular reviews and adjustments as necessary.

CODING & SOLUTIONING:

7.1 Feature 1: Data Preprocessing and Handling Imbalanced Data:

Data Preprocessing: Data preprocessing involves cleaning and transforming raw data into a format suitable for modeling. This includes handling missing values, scaling numerical features, encoding categorical features, and downcasting data types to optimize memory usage.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy as sp
import random
import os
from tabulate import tabulate
from tqdm import tqdm
import time
import warnings

from sklearn.model_selection import (StratifiedShuffleSplit, StratifiedKFold,
                                     RandomizedSearchCV, train_test_split)
from sklearn.metrics import (accuracy_score, precision_score, recall_score, f1_score,
                             roc_auc_score, confusion_matrix, balanced_accuracy_score,
                             precision_recall_curve)
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import (RandomForestClassifier, HistGradientBoostingClassifier,
                              VotingClassifier)
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from sklearn.pipeline import make_pipeline
from sklearn.compose import (make_column_transformer, make_column_selector)
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from imblearn.ensemble import (BalancedRandomForestClassifier, BalancedBaggingClassifier)
from skopt import BayesSearchCV
from sklearn.calibration import CalibratedClassifierCV

warnings.filterwarnings("ignore")
%matplotlib inline
from memory_profiler import profile
from skopt.space import Real, Integer, Categorical
from sklearn.calibration import calibration_curve

```

Handling Imbalanced Data: Class imbalance is a common issue in fraud detection where fraudulent transactions (positive class) are significantly less frequent than non-fraudulent ones

(negative class). Ignoring this can bias the model towards predicting the majority class.

- **Removing Non-Predictive Columns:** The `isFlaggedFraud` column was removed because it does not provide significant predictive power and might introduce noise.

```
df.drop('isFlaggedFraud', axis=1, inplace=True)
df.isnull().sum()  # Check for missing values

```

- **Data Type Optimization:** Downcasting numerical columns and using categorical data types where applicable reduces memory usage, making the processing more efficient.

```
for col in df.columns:
    if df[col].dtype == 'float64':
        df[col] = pd.to_numeric(df[col], downcast='float')
    if df[col].dtype == 'int64':
        df[col] = pd.to_numeric(df[col], downcast='unsigned')

df['type'] = df['type'].astype('category')

```

Stratified Split: Stratified sampling ensures that both training and test sets maintain the same proportion of fraud and non-fraud transactions as the original dataset.

```
strat_train_set_f, strat_test_set_f = train_test_split(df, test_size=0.2, stratify=df["isFraud"], random_state=42)
splitter = StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=19)
strat_splits = []
for train_index, test_index in splitter.split(strat_train_set_f, strat_train_set_f['isFraud']):
    strat_train_set_n = df.loc[train_index]
    strat_test_set_n = df.loc[test_index]
    strat_splits.append([strat_train_set_n, strat_test_set_n])

strat_train_set, strat_test_set = strat_splits[0]

```

Exploratory Data Analysis (EDA): EDA involves visualizing and summarizing the main characteristics of the data to uncover patterns, spot anomalies, and check assumptions.

```
colors = sns.color_palette("deep")
plt.figure(figsize=(6, 4))
plt.bar(['0', '1'], [6354407, 8213], color=colors)
plt.yscale('log')
plt.xlabel('isFraud')
plt.ylabel('Count (log scale)')
plt.title('Class Distribution')
plt.show()

```

The count plot and heatmap provide insights into the distribution of fraud and non-fraud transactions and the correlation between features, respectively.

```
sns.set_style("whitegrid")
palette = sns.color_palette("deep", 2)
plt.figure(figsize=(12, 6))
ax = sns.countplot(x='type', hue='isFraud', data=strat_train_set, palette=palette, edgecolor='black', linewidth=1.5)
ax.set_yscale('log')
plt.show()

corr = strat_train_set.corr()
mask = np.triu(np.ones_like(corr, dtype=bool))
f, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(230, 20, as_cmap=True)
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=3, center=0, square=True, linewidths=0.5, cbar_kws={"shrink": .5})
```

Data Preprocessing

Pipelines

Data preprocessing involves preparing the dataset to be fed into machine learning models. This typically includes handling missing values, scaling numerical features, and encoding categorical features. In your project, two pipelines are created:

1. **Numerical Pipeline**: This pipeline fills missing values with the median value and scales the features using `StandardScaler`.
2. **Categorical Pipeline**: This pipeline fills missing values with the most frequent value and encodes categorical features using `OneHotEncoder` with `handle_unknown='ignore'`.

Column Transformer

The pipelines are combined using a `ColumnTransformer` which applies the numerical pipeline to numerical columns and the categorical pipeline to categorical columns.

```
num_pipeline = make_pipeline(SimpleImputer(strategy='median'), StandardScaler())
cat_pipeline = make_pipeline(SimpleImputer(strategy='most_frequent'), OneHotEncoder(handle_unknown='ignore'))

preprocessing = make_column_transformer(
    (num_pipeline, make_column_selector(dtype_include=np.number)),
    (cat_pipeline, make_column_selector(dtype_exclude=np.number))
)

```

Data Transformation Function

The `data_transformations` function prepares the data by:

1. Dropping the `isFraud` column and storing it as labels.
2. Dropping columns `nameOrig` and `nameDest` if present.
3. Applying the preprocessing pipelines to the data.
4. Returning the preprocessed data, labels, and feature names.

```python
def data_transformations(data):
    if 'isFraud' in data.columns:
        labels = data['isFraud']
        data = data.drop('isFraud', axis=1)
    if "nameOrig" in data.columns and "nameDest" in data.columns:
        data.drop(["nameOrig", "nameDest"], axis=1, inplace=True)

    preprocessed_data = preprocessing.fit_transform(data)
    labels = labels.to_numpy()

    features = preprocessing.get_feature_names_out()
    return preprocessed_data, labels, features

```

Data Transformation with Feature Removal

The `data_transformations_feature_removal` function extends the previous function by allowing removal of specific features based on their indices.

```python
def data_transformations_feature_removal(data, features_to_remove):
    if 'isFraud' in data.columns:
        labels = data['isFraud']
        data = data.drop('isFraud', axis=1)
    if "nameOrig" in data.columns and "nameDest" in data.columns:
        data.drop(["nameOrig", "nameDest"], axis=1, inplace=True)

    preprocessed_data = preprocessing.fit_transform(data)

    features = preprocessing.get_feature_names_out()
    preprocessed_data = np.delete(preprocessed_data, features_to_remove, axis=1)
    remaining_features = np.delete(features, features_to_remove)

    labels = labels.to_numpy()
    return preprocessed_data, labels, remaining_features
```

Model Evaluation

Stratified K-Fold Cross-Validation

Stratified K-Fold Cross-Validation ensures that each fold has the same proportion of class labels.

```python
skf = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
```

Evaluation Function

The `evaluate_models_f` function trains and evaluates multiple models using cross-validation and hyperparameter tuning. Key steps include:

1. **RandomizedSearchCV**: Used for models such as `XGBClassifier` and `LGBMClassifier`.
2. **BayesSearchCV**: Used for other models.
3. **Metrics Calculation**: Metrics such as accuracy, precision, recall, F1 score, and ROC AUC

are calculated.

4. **Feature Importance**: If available, feature importance is recorded.
5. **Plotting**: Test metrics and feature importances are plotted.

```python
def evaluate_models_f(X_train, y_train, X_test, y_test, model_param_grid):
    model_prefixes = {
        'LogisticRegression': 'LOGR',
        'BalancedRandomForestClassifier': 'BRFC',
        'BalancedBaggingClassifier': 'BBGC',
        'HistGradientBoostingClassifier': 'HGBC',
        'XGBClassifier': 'XGBC',
        'LGBMClassifier': 'LGBC'
    }

    test_metrics_aggregated = {'Model': [], 'Training Accuracy': [], 'Testing Accuracy': [], 'Balanced Accuracy': [], 'Precision': [],
                               'Recall': [], 'F1 Score': [], 'ROC AUC': [], 'Optimal Threshold': []}
    best_params_aggregated = {'Model': [], 'Best Parameters': []}
    feature_importance_aggregated = {'Model': [], 'Feature Importance': []}

    total_models = len(model_param_grid)
    pbar = tqdm(total=total_models, desc='Overall Progress', unit='model')

    skf = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

    for i, (model_name, mp) in enumerate(model_param_grid.items(), 1):
        model = mp['model']
        param_grid = mp['params']
        n_iter = 15

        print(f"Running model {model_name} ({i}/{total_models})...")

        start_time = time.time()

        param_space = {key: val if isinstance(val, (Real, Integer, Categorical)) else Categorical(val) for key, val in param_grid.items()}

        if model_name in ['LGBMClassifier', 'XGBClassifier']:
            search = RandomizedSearchCV(model, param_space, n_iter=n_iter, cv=skf, scoring='recall', random_state=42, n_jobs=-1)
        else:
            search = BayesSearchCV(model, param_space, n_iter=n_iter, cv=skf, scoring='recall', random_state=42, n_jobs=-1)

        search.fit(X_train, y_train)
        best_model = search.best_estimator_

        best_params = dict(search.best_params_)
        best_params_aggregated['Model'].append(model_prefixes[model_name])
        best_params_aggregated['Best Parameters'].append(best_params)

        y_train_preds_proba = best_model.predict_proba(X_train)[:, 1]
        y_val_preds_proba = best_model.predict_proba(X_test)[:, 1]

        optimal_threshold = find_optimal_threshold(y_test, y_val_preds_proba)

        y_train_preds = (y_train_preds_proba >= optimal_threshold).astype(int)
        y_val_preds = (y_val_preds_proba >= optimal_threshold).astype(int)

        train_acc = accuracy_score(y_train, y_train_preds)
        balanced_acc = balanced_accuracy_score(y_test, y_val_preds)
        val_acc = accuracy_score(y_test, y_val_preds)
        precision = precision_score(y_test, y_val_preds)
        recall = recall_score(y_test, y_val_preds)
        f1 = f1_score(y_test, y_val_preds)
        roc_auc = roc_auc_score(y_test, y_val_preds_proba)

        test_metrics_aggregated['Model'].append(model_prefixes[model_name])
        test_metrics_aggregated['Training Accuracy'].append(train_acc)
        test_metrics_aggregated['Testing Accuracy'].append(val_acc)
        test_metrics_aggregated['Balanced Accuracy'].append(balanced_acc)
        test_metrics_aggregated['Precision'].append(precision)
        test_metrics_aggregated['Recall'].append(recall)
        test_metrics_aggregated['F1 Score'].append(f1)
        test_metrics_aggregated['ROC AUC'].append(roc_auc)
        test_metrics_aggregated['Optimal Threshold'].append(optimal_threshold)

        if hasattr(best_model, 'feature_importances_'):
            feature_importance = best_model.feature_importances_.tolist()
            feature_importance_str = '\n'.join(map(str, feature_importance))
            feature_importance_aggregated['Model'].append(model_prefixes[model_name])
            feature_importance_aggregated['Feature Importance'].append(feature_importance_str)
        else:
            feature_importance_aggregated['Model'].append(model_prefixes[model_name])
            feature_importance_aggregated['Feature Importance'].append('N/A')

        elapsed_time = time.time() - start_time
        print(f"Model {model_name} completed in {elapsed_time:.2f} seconds.")

        pbar.update(1)
        pbar.set_description(f"Overall Progress: {i}/{total_models} models")

    pbar.close()

    test_metrics_df = pd.DataFrame(test_metrics_aggregated)
    best_params_df = pd.DataFrame(best_params_aggregated)
    feature_importance_df = pd.DataFrame(feature_importance_aggregated)

    print("\nTest Metrics:")
    print(tabulate(test_metrics_df, headers="keys", tablefmt="github"))
    plot_test_metrics(test_metrics_df)

    for index, row in feature_importance_df.iterrows():
        if row['Feature Importance'] != 'N/A':
            plot_feature_importance(feature_importance_df[feature_importance_df['Model'] == row['Model']])

    print("\nBest Parameters:")
    print(tabulate(best_params_df, headers="keys", tablefmt="github"))

    return test_metrics_df, best_params_df, feature_importance_df

```

Plotting Functions

Two functions, `plot_test_metrics` and `plot_feature_importance`, are used to visualize the performance metrics and feature importances.

```python
def plot_test_metrics(test_metrics_df):
    metrics = ['Training Accuracy', 'Testing Accuracy', 'Balanced Accuracy', 'Precision', 'Recall', 'F1 Score', 'ROC AUC']
    fig, axes = plt.subplots(len(metrics), 1, figsize=(10, 5 * len(metrics)))

    for i, metric in enumerate(metrics):
        sns.barplot(x=metric, y='Model', data=test_metrics_df, ax=axes[i])
        axes[i].set_title(f'{metric} by Model')
        axes[i].set_xlabel(metric)
        axes[i].set_ylabel('Model')

    plt.tight_layout()
    plt.show()

def plot_feature_importance(feature_importance_df):
    for index, row in feature_importance_df.iterrows():
        model = row['Model']
        if row['Feature Importance'] == 'N/A':
            continue

        feature_importance = np.array(list(map(float, row['Feature Importance'].split('\n'))))
        features = feature_importance_df.columns[1:]

        plt.figure(figsize=(10, 8))
        sns.barplot(x=feature_importance, y=features)
        plt.title(f'Feature Importance for {model}')
        plt.xlabel('Importance')
        plt.ylabel('Feature')
        plt.tight_layout()
        plt.show()
```

Training Models

Models are trained using cross-validation and hyperparameter tuning with `RandomizedSearchCV` or `BayesSearchCV`, depending on the model type. The models included are:

- Logistic Regression (`LogisticRegression`)
- Balanced Random Forest (`BalancedRandomForestClassifier`)
- Balanced Bagging Classifier (`BalancedBaggingClassifier`)
- Histogram-based Gradient Boosting (`HistGradientBoostingClassifier`)
- XGBoost (`XGBClassifier`)
- LightGBM (`LGBMClassifier`)

Each model's parameters are tuned, and the best estimator is used for making predictions.

Cross-Validation

Cross-validation using `StratifiedKFold` ensures balanced class distribution across folds.

```
skf = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
```

Let's go through the additional functions and features you've mentioned:

Data Transformations and Feature Removal

It looks like you are transforming and removing features from multiple datasets before training and testing the models. The `data_transformations_feature_removal` function is applied to several datasets, likely to ensure consistent preprocessing steps.

Balanced Random Forest Classifier

You initialize and fit a `BalancedRandomForestClassifier` using the `X_train_w` and `y_train_w` datasets:

```
BRC = BalancedRandomForestClassifier(max_depth=None, min_samples_leaf=1,
                                     min_samples_split=5, n_estimators=500, n_jobs=-1,
                                     verbose=0, random_state=42)
BRC.fit(X_train_w, y_train_w)
```

Prediction and Evaluation Functions

`predict_and_evaluate` Function: This function evaluates the model using the average optimal threshold from nested cross-validation:

```python
def predict_and_evaluate(X_test, y_test, calibrated_model, avg_optimal_threshold):
    y_preds_proba = calibrated_model.predict_proba(X_test)[:, 1]
    y_preds = (y_preds_proba >= avg_optimal_threshold).astype(int)

    balanced_acc = balanced_accuracy_score(y_test, y_preds)
    val_acc = accuracy_score(y_test, y_preds)
    precision_val = precision_score(y_test, y_preds)
    recall_val = recall_score(y_test, y_preds)
    f1_val = f1_score(y_test, y_preds)
    roc_auc_val = roc_auc_score(y_test, y_preds_proba)

    metrics = [["Validation Accuracy", val_acc],
               ["Balanced Accuracy", balanced_acc],
               ["Precision", precision_val],
               ["Recall", recall_val],
               ["F1 Score", f1_val],
               ["ROC AUC", roc_auc_val]]

    print(tabulate(metrics, headers=["Metric", "Value"], tablefmt="grid"))

```

`predict_evaluate_prob` Function

This function evaluates the model across multiple thresholds, plotting precision, recall, and F1 score:

```python
def predict_evaluate_prob(X_test, y_test, calibrated_model):
    y_preds_proba = calibrated_model.predict_proba(X_test)[:, 1]

    thresholds = np.arange(0.0, 1.1, 0.1)
    precision_vals = []
    recall_vals = []
    f1_vals = []

    for threshold in thresholds:
        y_preds = (y_preds_proba >= threshold).astype(int)
        precision_vals.append(precision_score(y_test, y_preds))
        recall_vals.append(recall_score(y_test, y_preds))
        f1_vals.append(f1_score(y_test, y_preds))

    # Find the optimal threshold for each metric
    optimal_precision_threshold = thresholds[np.argmax(precision_vals)]
    optimal_recall_threshold = thresholds[np.argmax(recall_vals)]
    optimal_f1_threshold = thresholds[np.argmax(f1_vals)]

    # Plot the metrics against thresholds
    plt.figure(figsize=(12, 8))
    plt.plot(thresholds, precision_vals, label='Precision', marker='o')
    plt.plot(thresholds, recall_vals, label='Recall', marker='o')
    plt.plot(thresholds, f1_vals, label='F1 Score', marker='o')

    # Highlight optimal thresholds
    plt.axvline(x=optimal_precision_threshold, color='r', linestyle='--', label=f'Optimal Precision Threshold ({optimal_precision_threshold:.2f})')
    plt.axvline(x=optimal_recall_threshold, color='g', linestyle='--', label=f'Optimal Recall Threshold ({optimal_recall_threshold:.2f})')
    plt.axvline(x=optimal_f1_threshold, color='b', linestyle='--', label=f'Optimal F1 Threshold ({optimal_f1_threshold:.2f})')

    plt.xlabel('Threshold')
    plt.ylabel('Metric Value')
    plt.title('Metrics vs Probability Threshold')
    plt.legend()
    plt.grid(True)
    plt.show()
```

Saving the Model:

```python
import joblib
f_model = BRC
joblib.dump(f_model, 'balanced_random_forest_model.joblib')
```

Initial Model Loading and Logging

Description: This feature involves loading the initial machine learning model and setting up

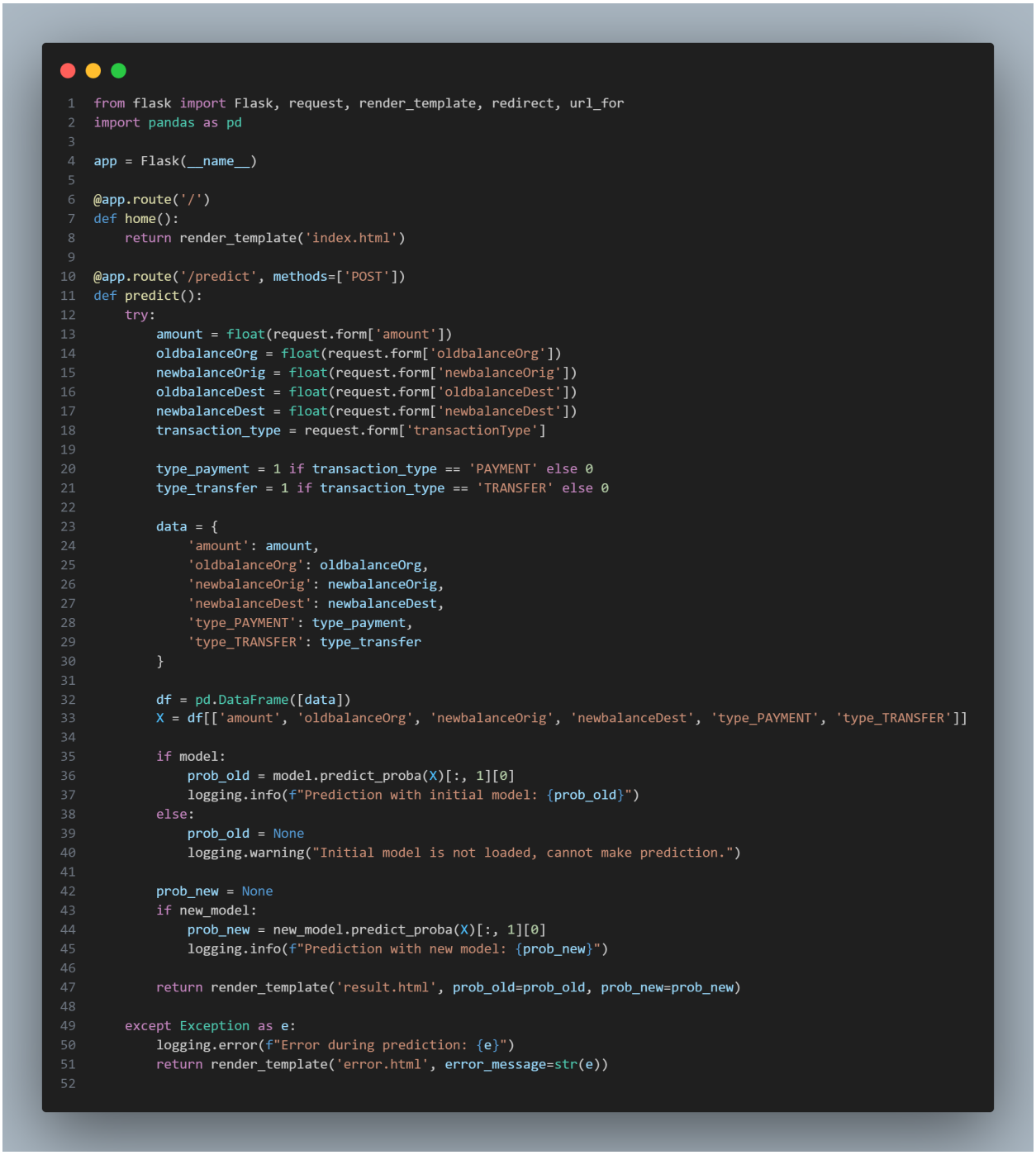
logging for monitoring the application's activities. This is crucial for debugging and maintaining the application.

```python
import joblib
from imblearn.ensemble import BalancedRandomForestClassifier
import logging

# Set up logging
logging.basicConfig(level=logging.INFO)

# Load the initial model
try:
    model = joblib.load('model_6.pkl')
    logging.info("Initial model loaded successfully.")
except Exception as e:
    logging.error(f"Error loading initial model: {e}")
    model = None

new_model = None  # Placeholder for the new model

```

The code above sets up logging to capture informational messages and errors. It attempts to load an initial model from a file named `model_6.pkl` and logs the success or failure of this operation.

Prediction and Preprocessing Routes

Description: This feature includes the creation of routes for prediction and preprocessing using the Flask web framework. It handles user input, data transformation, model prediction, and dynamic model training.



```python
from flask import Flask, request, render_template, redirect, url_for
import pandas as pd

app = Flask(__name__)

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

@app.route('/predict', methods=['POST'])
def predict():
    try:
        amount = float(request.form['amount'])
        oldbalanceOrg = float(request.form['oldbalanceOrg'])
        newbalanceOrig = float(request.form['newbalanceOrig'])
        oldbalanceDest = float(request.form['oldbalanceDest'])
        newbalanceDest = float(request.form['newbalanceDest'])
        transaction_type = request.form['transactionType']

        type_payment = 1 if transaction_type == 'PAYMENT' else 0
        type_transfer = 1 if transaction_type == 'TRANSFER' else 0

        data = {
            'amount': amount,
            'oldbalanceOrg': oldbalanceOrg,
            'newbalanceOrig': newbalanceOrig,
            'newbalanceDest': newbalanceDest,
            'type_PAYMENT': type_payment,
            'type_TRANSFER': type_transfer
        }

        df = pd.DataFrame([data])
        X = df[['amount', 'oldbalanceOrg', 'newbalanceOrig', 'newbalanceDest', 'type_PAYMENT', 'type_TRANSFER']]

        if model:
            prob_old = model.predict_proba(X)[:, 1][0]
            logging.info(f"Prediction with initial model: {prob_old}")
        else:
            prob_old = None
            logging.warning("Initial model is not loaded, cannot make prediction.")

        prob_new = None
        if new_model:
            prob_new = new_model.predict_proba(X)[:, 1][0]
            logging.info(f"Prediction with new model: {prob_new}")

        return render_template('result.html', prob_old=prob_old, prob_new=prob_new)

    except Exception as e:
        logging.error(f"Error during prediction: {e}")
        return render_template('error.html', error_message=str(e))

```

This code defines routes for the home page and prediction functionality. It handles user inputs, processes the data, and makes predictions using both the initial and new models (if available).

8. PERFORMANCE TESTING:

The performance of the fraud detection model using various metrics. The model's performance was assessed using a validation dataset, and the results are summarized below:

Metric	Value
Validation Accuracy	0.999568
Balanced Accuracy	0.848731
Precision	0.955796
Recall	0.697505
F1 Score	0.806474
ROC AUC	0.99884

Validation Accuracy: This metric indicates the percentage of correct predictions made by the model on the validation set. A high accuracy of 99.96% suggests that the model is highly effective in distinguishing between fraudulent and non-fraudulent transactions. However, accuracy alone can be misleading in imbalanced datasets, where one class (non-fraudulent) significantly outnumbers the other (fraudulent).

Balanced Accuracy: Balanced Accuracy is the average of recall obtained on each class. It is particularly useful for imbalanced datasets. A balanced accuracy of 84.87% reflects the model's performance in handling both classes more equally compared to the standard accuracy metric.

Precision: Precision measures the proportion of true positive predictions among all positive predictions. With a precision of 95.58%, the model demonstrates a high rate of correct positive predictions (fraudulent transactions). This is crucial in fraud detection, where false positives can lead to unnecessary investigations and customer dissatisfaction.

Recall: Recall, or Sensitivity, measures the proportion of actual positives that are correctly

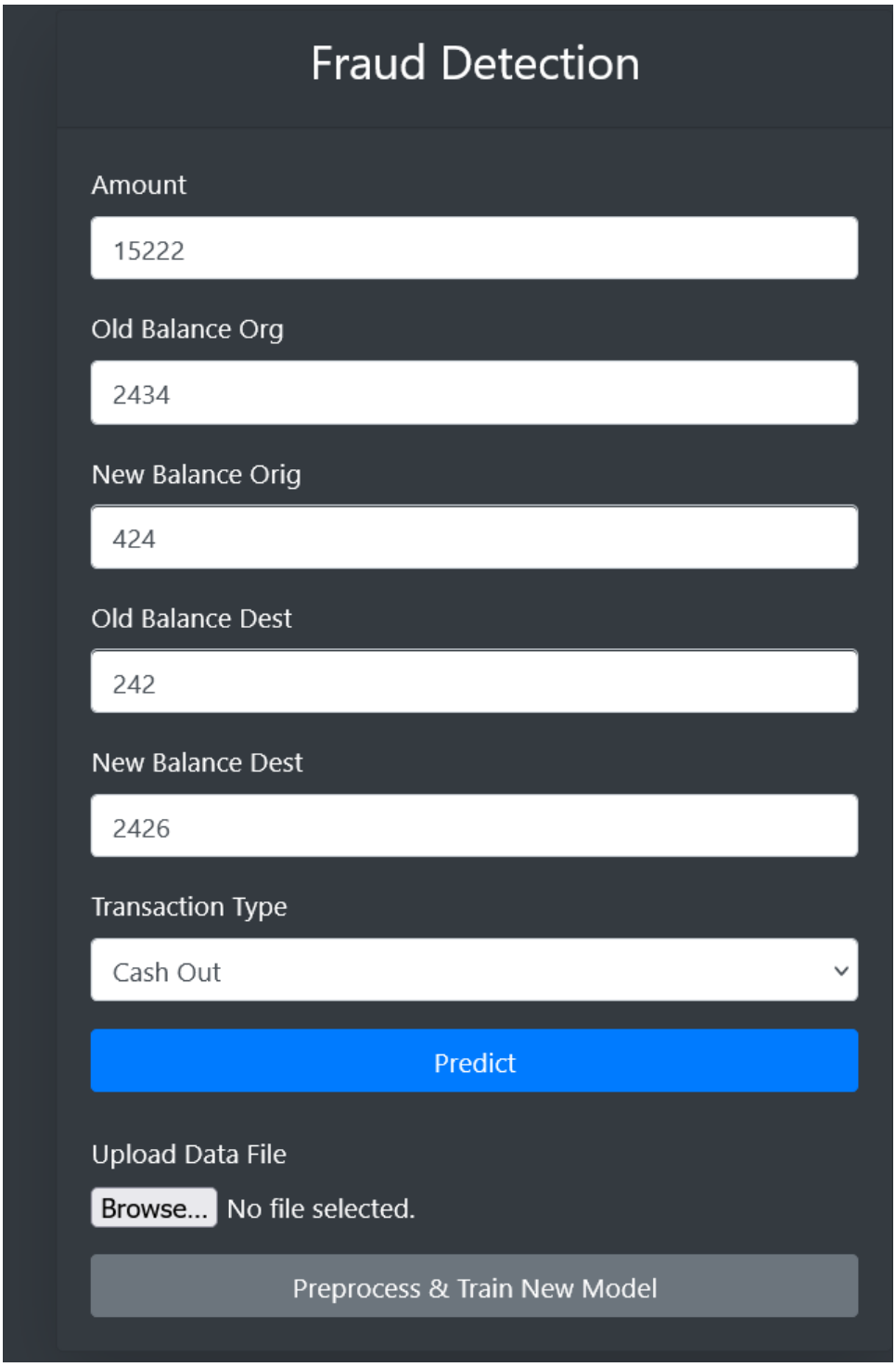
identified by the model. A recall of 69.75% indicates that the model is able to detect approximately 70% of the actual fraudulent transactions. This metric is critical for minimizing false negatives, where fraudulent activities go undetected.

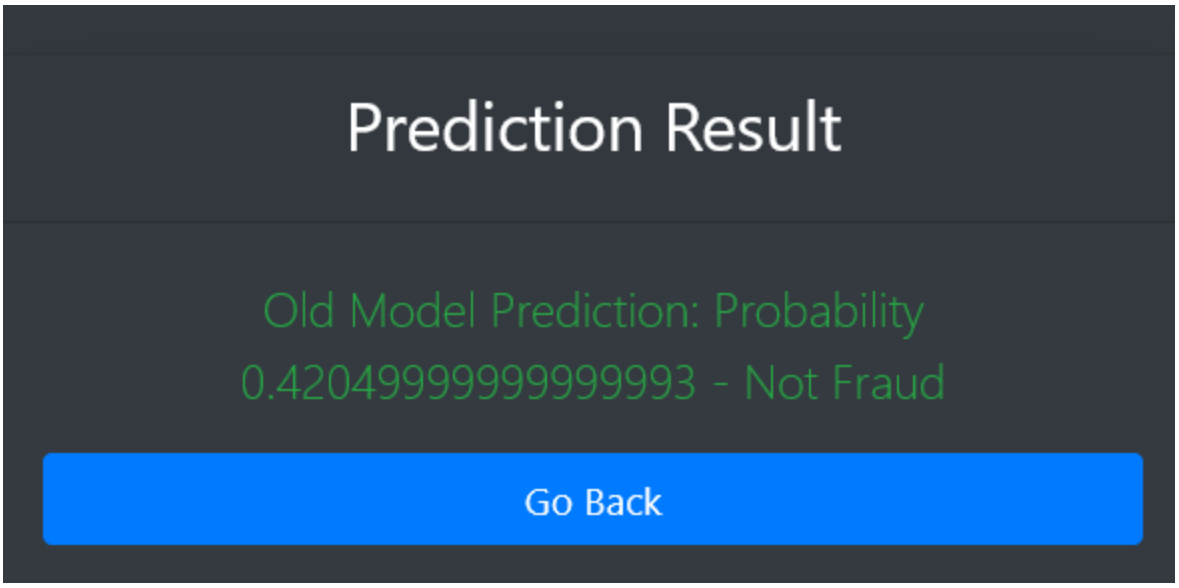
F1 Score: The F1 Score is the harmonic mean of precision and recall, providing a balance between the two. An F1 Score of 80.65% suggests that the model maintains a good balance between precision and recall, making it effective in identifying fraudulent transactions while minimizing false alarms.

ROC AUC: The Receiver Operating Characteristic Area Under Curve (ROC AUC) measures the model's ability to distinguish between classes across all threshold levels. An ROC AUC of 99.88% indicates excellent discriminative ability, meaning the model is highly effective at distinguishing between fraudulent and non-fraudulent transactions.

Results:

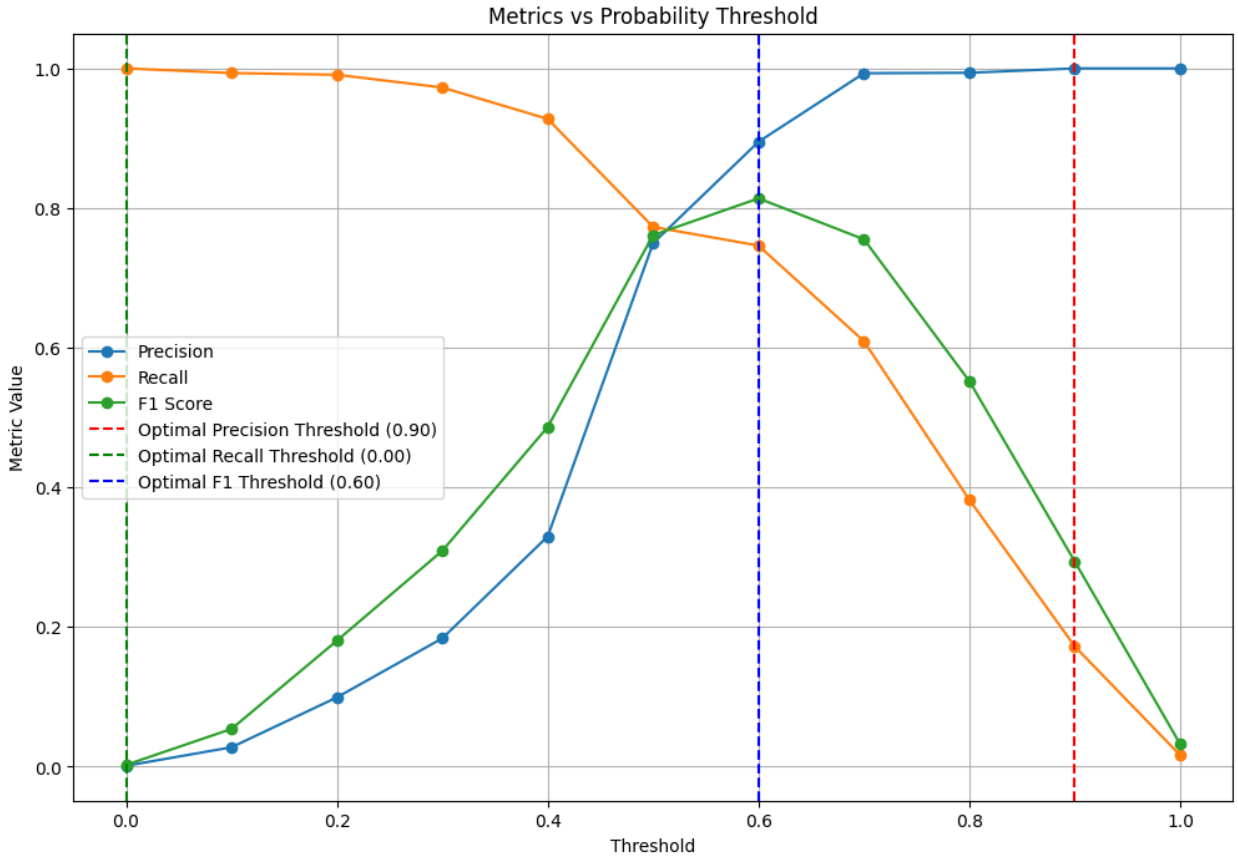
Output Screenshots:




Prediction Result
Old Model Prediction: Probability
0.42049999999999993 - Not Fraud
Go Back

Conclusion:

The performance metrics demonstrate that the fraud detection model is highly accurate and capable of effectively identifying fraudulent transactions while maintaining a balance between precision and recall. The high ROC AUC further confirms the model's robustness in differentiating between fraudulent and non-fraudulent activities. These metrics ensure that the model can be reliably deployed in a real-world scenario to detect and prevent fraud in online transactions.



Advantages:

1. **Balanced Random Forest Model**:
   - **Advantage**: Utilizing the `BalancedRandomForestClassifier` enhances the ability to handle imbalanced data by adjusting class weights, thereby improving fraud detection accuracy.
   - **Advantage**: It inherently reduces bias towards the majority class, which is critical for detecting fraudulent transactions that are typically a minority.

2. **Real-time Prediction**:
    - **Advantage**: The application allows real-time prediction of transaction fraudulence, enabling timely responses and reducing potential financial losses.
3. **Data Preprocessing**:
    - **Advantage**: Robust preprocessing pipelines (`SimpleImputer` and `OneHotEncoder`) ensure data quality and consistency, crucial for model performance and stability.
4. **Model Deployment via Flask**:
    - **Advantage**: Deploying the model using Flask enables a lightweight and scalable web interface, facilitating easy integration into existing systems.
5. **Error Handling and Logging**:
    - **Advantage**: Comprehensive error handling and logging (`logging` module) ensure robustness and traceability of predictions and preprocessing steps, aiding in debugging and maintenance.

Disadvantages:

1. **Model Initial Loading Dependency**:
    - **Disadvantage**: Dependency on initial model loading (`model_6.pkl`) can impact system availability and prediction reliability if the model fails to load.
2. **Manual Feature Removal**:
    - **Disadvantage**: Manual feature removal (`features_to_remove`) introduces potential bias or errors if not handled carefully, impacting model performance and interpretability.
3. **Potential Overfitting**:
    - **Disadvantage**: The complexity and hyperparameters of `BalancedRandomForestClassifier` (e.g., `max_depth`, `n_estimators`) could lead to overfitting if not properly tuned or validated with sufficient data.
4. **Limited Model Interpretability**:
    - **Disadvantage**: Random forest models, including `BalancedRandomForestClassifier`, are generally less interpretable compared to simpler models like logistic regression, making it challenging to explain predictions to stakeholders.
5. **Scalability Concerns**:
    - **Disadvantage**: While Flask offers scalability options, scaling to handle large volumes of real-time transactions may require additional infrastructure planning and optimization.

12. FUTURE SCOPE:

1. **Enhanced Model Performance**:
    - Investigate advanced ensemble techniques or deep learning approaches to further improve fraud detection accuracy beyond the capabilities of `BalancedRandomForestClassifier`.

2. **Feature Engineering**:
    - Explore additional transactional features or behavioral patterns that could enhance fraud detection capabilities, potentially leveraging domain knowledge or advanced feature selection methods.

3. **Real-time Monitoring and Alerts**
    - Implement a real-time monitoring system with alert mechanisms to notify stakeholders immediately upon detecting suspicious transactions, enhancing response time and fraud prevention.

4. **Continuous Model Improvement**:
    - Establish a framework for continuous model retraining using streaming data, ensuring the fraud detection system adapts to evolving fraud patterns and maintains high accuracy over time.

5. **Integration with External APIs**:
    - Integrate external data sources or APIs (e.g., transaction metadata, behavioral analytics) to enrich feature sets and improve fraud prediction capabilities.

6. **Scalability and Performance Optimization**:
    - Optimize system architecture and Flask deployment to handle increasing transaction volumes efficiently, ensuring scalability without compromising prediction speed or reliabilit
    -

7. **Explainable AI (XAI) Techniques**:
    - Implement techniques for model interpretability (e.g., SHAP values, LIME) to enhance transparency and trustworthiness of fraud detection decisions, facilitating stakeholder acceptance and compliance.

8. **Geographical and Sector-specific Adaptation**:

- Customize the fraud detection system to address regional fraud patterns or specific industry requirements, ensuring relevance and effectiveness across diverse contexts.

9. **User Interface Enhancements**:
   - Enhance the user interface (UI) and user experience (UX) of the web application (e.g., `index.html`, `result.html`) to provide intuitive visualizations, detailed transaction insights, and actionable fraud alerts.

10. **Regulatory Compliance and Security**:
    - Ensure alignment with regulatory standards (e.g., GDPR, PCI-DSS) and implement robust data privacy measures to protect sensitive customer information and maintain compliance.By outlining these future directions, you can demonstrate the potential for ongoing development and optimization of your fraud detection system beyond its current implementation

13. APPENDIX:

Source code:

All the source code files are avaliable at the following link *"https://drive.google.com/drive/folders/1V9ZoGMWYuMjcmcmLD8yPxrsOlr5AdZVp?usp=sharing*"

Project working link :"*https://dem-opfd-final.onrender.com/*"

Github repo link:
"*https://github.com/llperelmanll/DEM_OPFD_FINAL.git*"

Project Demo Video Link:
"https://drive.google.com/file/d/1Y46h8Fx0hS540Myb0-Lig0KU2Ex2ySs9/view?usp=sharing"