TEAM_4_BA810_Project_file

December 9, 2024

1 BANK FRAUD DETECTION

Team : Omar Alwehaib, Adesina Adeniran, Yifeng Chen, Kaushiki Tiwary

2 INTRODUCTION

In today's digital era, while financial transactions have brought unparalleled convenience, they have also heightened the risk of fraudulent activities. Fraud not only results in financial losses but also undermines customer trust in banking institutions. Detecting and preventing fraud has become a pressing challenge for banks, demanding innovative and scalable solutions.

This project focuses on developing a predictive model using the Bank Account Fraud dataset (sourced from Kaggle NeurIPS 2022) to identify potential fraudulent activities. Fraud detection is pivotal in the financial sector, as accurate predictions can significantly reduce financial losses and restore customer confidence.

Our objective is to determine the most effective model for identifying bank fraud, enabling banks to distinguish between legitimate and fraudulent customers effectively.

3 DATA SET

##DATA LOADING

[ ]:
```python
import pandas as pd
import numpy as np
```

[ ]:
```python
from google.colab import drive
drive.mount('/content/drive')
shared_folder_path = '/content/drive/MyDrive/BA_810'
data = pd.read_csv('/content/drive/MyDrive/BA_810/Base.csv')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

##DATA CLEANING

The null values in our dataset were as -1, we got rid of the rows where it's less than 30% and removed columns where it's more than 70%. We gathered information from the github repo as our data dictionary and we went through the suggested columns with -1 in them.

```
[ ]: #Data Cleaning - Counting -1 in the dataframe
count_neg1 = (data == -1).sum().sum()
print(f"Total count of -1 in the DataFrame: {count_neg1}")
```

```
Total count of -1 in the DataFrame: 973671
```

```
[ ]: #Data Cleaning - Counting Null Values in Specific Columns
columns_with_nulls = ['device_distinct_emails_8w', 'session_length_in_minutes',␣
 ↪'bank_months_count', 'current_address_months_count',␣
 ↪'prev_address_months_count']

for column in columns_with_nulls:
  null_count = (data[column] == -1).sum()
  print(f"Number of null values in {column}: {null_count}")
```

```
Number of null values in device_distinct_emails_8w: 359
Number of null values in session_length_in_minutes: 2015
Number of null values in bank_months_count: 253635
Number of null values in current_address_months_count: 4254
Number of null values in prev_address_months_count: 712920
```

```
[ ]: #Data Cleaning - Dropped rows in columns with -1
columns_to_drop_rows = ['device_distinct_emails_8w',␣
 ↪'session_length_in_minutes', 'current_address_months_count']
mask = (data[columns_to_drop_rows] == -1).any(axis=1)
data = data[~mask]
```

```
[ ]: #Data Cleaning - Counting Null Values of the columns

columns_with_nulls = ['device_distinct_emails_8w', 'session_length_in_minutes',␣
 ↪'bank_months_count', 'current_address_months_count',␣
 ↪'prev_address_months_count']

for column in columns_with_nulls:
  null_count = (data[column] == -1).sum()
  print(f"Number of null values in {column}: {null_count}")
```

```
Number of null values in device_distinct_emails_8w: 0
Number of null values in session_length_in_minutes: 0
Number of null values in bank_months_count: 251245
Number of null values in current_address_months_count: 0
Number of null values in prev_address_months_count: 711206
```

```
[ ]: #Data_Cleaning - Counting -1 in bank_months_count column and removing it
data = data[data['bank_months_count'] != -1]
```

```
data = data.drop('prev_address_months_count', axis=1)
```

[ ]:
```
#Data_Cleaning - Counting Null Values in columns
columns_with_nulls = ['device_distinct_emails_8w', 'session_length_in_minutes',␣
↪'bank_months_count', 'current_address_months_count']

for column in columns_with_nulls:
  null_count = (data[column] == -1).sum()
  print(f"Number of null values in {column}: {null_count}")
```

```
Number of null values in device_distinct_emails_8w: 0
Number of null values in session_length_in_minutes: 0
Number of null values in bank_months_count: 0
Number of null values in current_address_months_count: 0
```

So far, Removed Prev_address_months_count and removed rows of others with -1.

[ ]:
```
data.info()
```

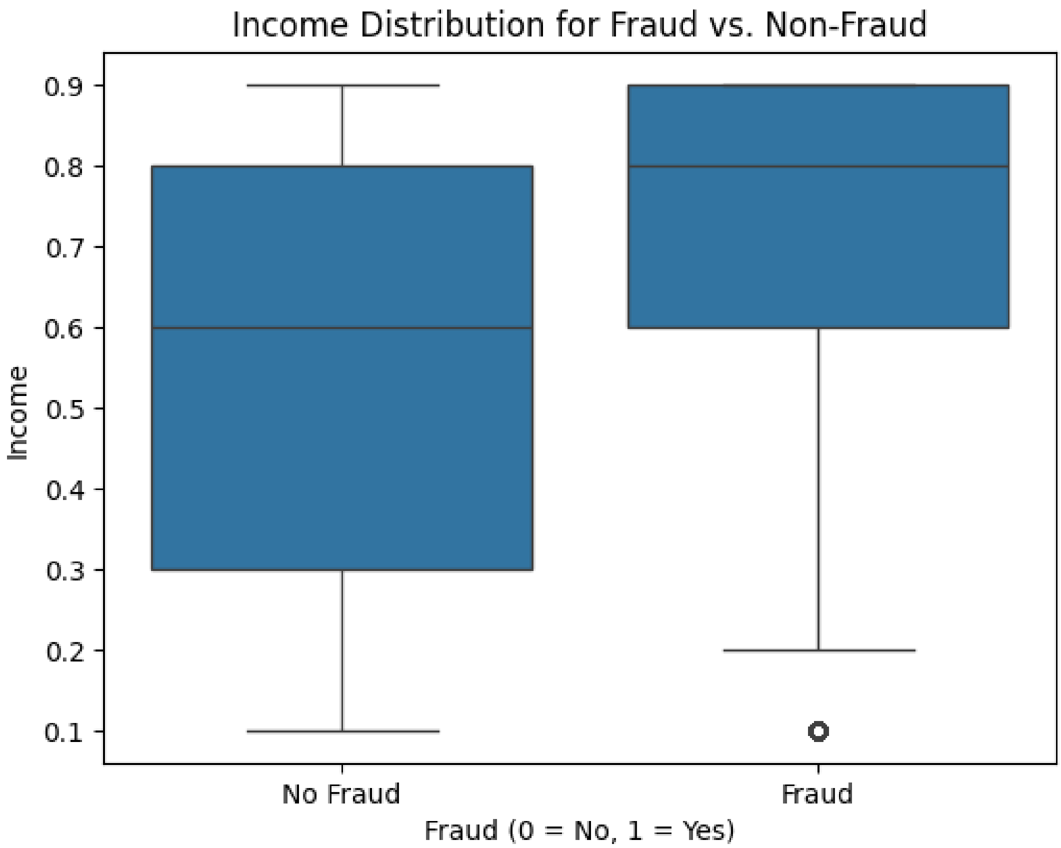
```
<class 'pandas.core.frame.DataFrame'>
Index: 742362 entries, 0 to 999999
Data columns (total 31 columns):
 #   Column                            Non-Null Count   Dtype
---  ------                            --------------   -----
 0   fraud_bool                        742362 non-null  int64
 1   income                            742362 non-null  float64
 2   name_email_similarity             742362 non-null  float64
 3   current_address_months_count      742362 non-null  int64
 4   customer_age                      742362 non-null  int64
 5   days_since_request                742362 non-null  float64
 6   intended_balcon_amount            742362 non-null  float64
 7   payment_type                      742362 non-null  object
 8   zip_count_4w                      742362 non-null  int64
 9   velocity_6h                       742362 non-null  float64
 10  velocity_24h                      742362 non-null  float64
 11  velocity_4w                       742362 non-null  float64
 12  bank_branch_count_8w              742362 non-null  int64
 13  date_of_birth_distinct_emails_4w  742362 non-null  int64
 14  employment_status                 742362 non-null  object
 15  credit_risk_score                 742362 non-null  int64
 16  email_is_free                     742362 non-null  int64
 17  housing_status                    742362 non-null  object
 18  phone_home_valid                  742362 non-null  int64
 19  phone_mobile_valid                742362 non-null  int64
 20  bank_months_count                 742362 non-null  int64
 21  has_other_cards                   742362 non-null  int64
 22  proposed_credit_limit             742362 non-null  float64
 23  foreign_request                   742362 non-null  int64
```

```
 24  source                           742362 non-null  object
 25  session_length_in_minutes        742362 non-null  float64
 26  device_os                        742362 non-null  object
 27  keep_alive_session               742362 non-null  int64
 28  device_distinct_emails_8w        742362 non-null  int64
 29  device_fraud_count               742362 non-null  int64
 30  month                            742362 non-null  int64
dtypes: float64(9), int64(17), object(5)
memory usage: 181.2+ MB
```

##EDA CHARTS

[ ]:

```python
#Income Distribution for Fraud and Non-Fraud

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

sns.boxplot(data=data, x='fraud_bool', y='income')

plt.title('Income Distribution for Fraud vs. Non-Fraud')
plt.xlabel('Fraud (0 = No, 1 = Yes)')
plt.ylabel('Income')
plt.xticks([0, 1], ['No Fraud', 'Fraud'])
plt.show()
```



The graph depicts that people with fraud have lesser income as compard with people who aren't fraud

[66]:
```
#Histogram to show fraud vs not fraud based on customer age
custom_blue = "#4682B4"
plt.figure(figsize=(12, 6))
sns.histplot(data=data, x='customer_age', hue='fraud_bool', multiple='stack',␣
↪bins=30, palette={0: custom_blue, 1: "red"}, kde=False)
plt.title('Distribution of Fraud and Non-Fraud Customers by Age', fontsize=14)
plt.xlabel('Customer Age', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.legend(title='Fraud', labels=['Not Fraud', 'Fraud'])
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

Fraudulent activities are predominantly associated with individuals in the younger to mid-age range, specifically between 20 and 50 years old.

[ ]:
```
data.hist(bins=50, figsize=(30, 20))
plt.show()
```

#IMPLEMENTING MODELS

##Preprocessing Pipeline

In our dataset, we have 30 columns, numercial : 19 and Categorical : 11

[ ]:
```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier

num_features = [
    "income", "name_email_similarity",
    "current_address_months_count", "customer_age", "days_since_request",
    "intended_balcon_amount", "zip_count_4w", "velocity_6h",
    "velocity_24h", "velocity_4w", "bank_branch_count_8w",
    "date_of_birth_distinct_emails_4w", "credit_risk_score",
    "bank_months_count", "proposed_credit_limit",
    "session_length_in_minutes", "device_distinct_emails_8w",
    "device_fraud_count", "month"
]
cat_features = [
    "payment_type", "employment_status", "housing_status",
    "source", "device_os", "email_is_free", "phone_home_valid",
    "phone_mobile_valid", "has_other_cards", "foreign_request",
    "keep_alive_session"
]
preprocessing = ColumnTransformer([
    ("num", StandardScaler(), num_features),
    ("cat", OneHotEncoder(drop="first"), cat_features)
])
```

[ ]:
```
from sklearn.model_selection import train_test_split
X = data.drop('fraud_bool', axis=1)
y = data['fraud_bool']
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=42,
                                                    stratify=y)
```

3.1 Implementing Bayesian Hyperparameter for Logestic Regression and Confusion Matrix

[ ]:
```
!pip install scikit-optimize
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from skopt import BayesSearchCV
from skopt.space import Real, Categorical

pipe = Pipeline([
    ('preprocess', preprocessing),
    ('logreg', LogisticRegression(max_iter=1000, class_weight='balanced'))
])

#BayesSearchCV
search_spaces = {
    'logreg__C': Real(1e-4, 10, prior='log-uniform'),
    'logreg__solver': Categorical(['lbfgs', 'liblinear'])
}

bayes_search = BayesSearchCV(
    estimator=pipe,
    search_spaces=search_spaces,
    n_iter=10,
    cv=3,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1,
    random_state=42
)

bayes_search.fit(X_train, y_train)

print("Best Parameters:", bayes_search.best_params_)
print("Best CV Score (Accuracy):", bayes_search.best_score_)

best_model = bayes_search.best_estimator_
y_pred = best_model.predict(X_test)

# result
test_accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", test_accuracy)

print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Collecting scikit-optimize
  Downloading scikit_optimize-0.10.2-py2.py3-none-any.whl.metadata (9.7 kB)
```

```
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (1.4.2)
Collecting pyaml>=16.9 (from scikit-optimize)
  Downloading pyaml-24.9.0-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: numpy>=1.20.3 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (1.26.4)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (1.13.1)
Requirement already satisfied: scikit-learn>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.5.2)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize) (24.2)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages
(from pyaml>=16.9->scikit-optimize) (6.0.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikit-
optimize) (3.5.0)
Downloading scikit_optimize-0.10.2-py2.py3-none-any.whl (107 kB)
                         107.8/107.8 kB
2.7 MB/s eta 0:00:00
Downloading pyaml-24.9.0-py3-none-any.whl (24 kB)
Installing collected packages: pyaml, scikit-optimize
Successfully installed pyaml-24.9.0 scikit-optimize-0.10.2
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Best Parameters: OrderedDict([('logreg__C', 1.1533999859559563),
('logreg__solver', 'lbfgs')])
Best CV Score (Accuracy): 0.7982771191249544
Test Accuracy: 0.7966566311719976
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.80      0.89    147099
           1       0.03      0.78      0.07      1374

    accuracy                           0.80    148473
   macro avg       0.52      0.79      0.48    148473
weighted avg       0.99      0.80      0.88    148473

Confusion Matrix:
```

```
[[117205  29894]
 [   297   1077]]
```

The confusion matrix and the low F1 score highlighted significant discrepancies in the model's performance. This prompted a deeper examination of the dataset, where we identified potential class imbalance issues that could be affecting the model's ability to effectively distinguish between fraud and non-fraud cases.

3.2 IMBALANCE DATA ON FRAUD_BOOL COLUMN

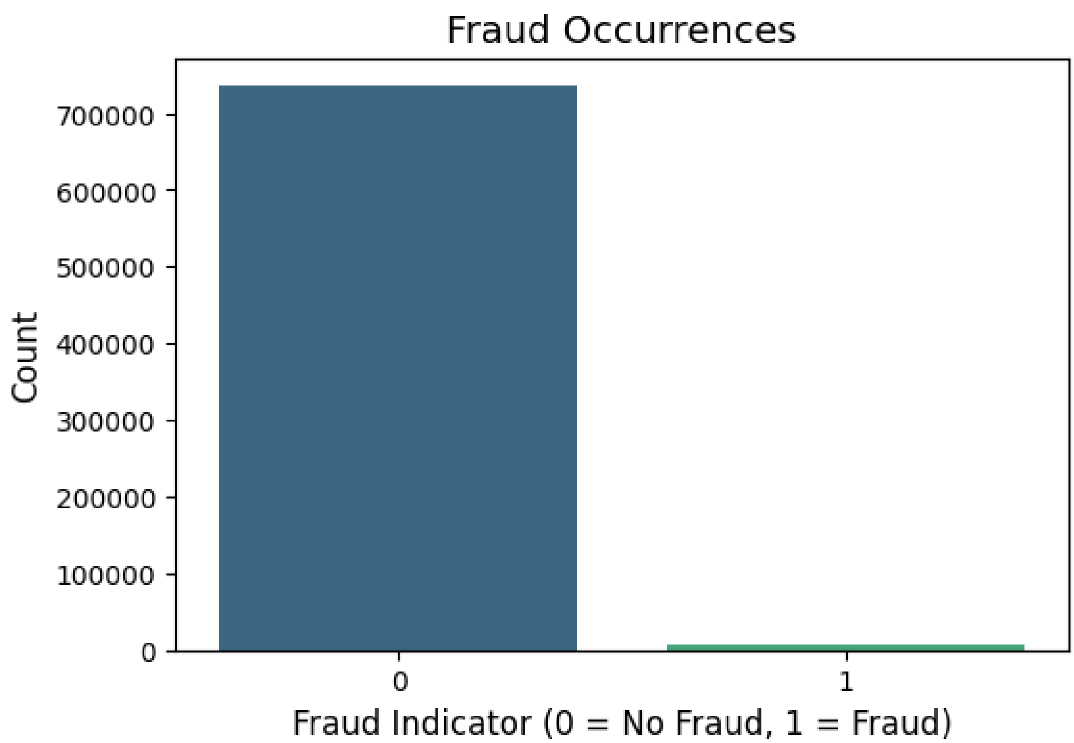
After observing F1-Score of 0.07, we identified an issue with the dataset. Upon further analysis, we discovered that the dataset lacked a sufficient number of fraud samples, which impacted the model's performance.

[ ]:
```
#identifying data imbalance
data['fraud_bool'].value_counts()
```

[ ]:
```
fraud_bool
0    735491
1      6871
Name: count, dtype: int64
```

[ ]:
```
#Discriptive_image of Imbalance data
value_counts = data['fraud_bool'].value_counts()
value_counts_df = value_counts.reset_index()
value_counts_df.columns = ['fraud_bool', 'count']
plt.figure(figsize=(6, 4))
sns.barplot(x='fraud_bool', y='count', data=value_counts_df, palette='viridis')
plt.title('Fraud Occurrences', fontsize=14)
plt.xlabel('Fraud Indicator (0 = No Fraud, 1 = Fraud)', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.show()
```

```
<ipython-input-35-5d0ae37ade4e>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='fraud_bool', y='count', data=value_counts_df,
palette='viridis')
```



3.3 Undersampling for Imbalance Dataset

[ ]:
```
fraudulent = data[data['fraud_bool'] == 1]
non_fraudulent = data[data['fraud_bool'] == 0]
non_fraudulent_sampled = non_fraudulent.sample(frac=0.1, random_state=42)
balanced_data = pd.concat([fraudulent, non_fraudulent_sampled])
balanced_data = balanced_data.sample(frac=1, random_state=42).
 ↪reset_index(drop=True)

print("Original dataset size:", data.shape)
print("Balanced dataset size:", balanced_data.shape)
```

```
Original dataset size: (742362, 31)
Balanced dataset size: (80420, 31)
```

Since there was class imbalance in the dataset, we addressed it by randomly sampling 10% of the rows from the non-fraudulent data. This step helped reduce the dominance of the majority class (non-fraudulent cases) and created a more balanced dataset, improving the model's ability to handle and predict both classes effectively.

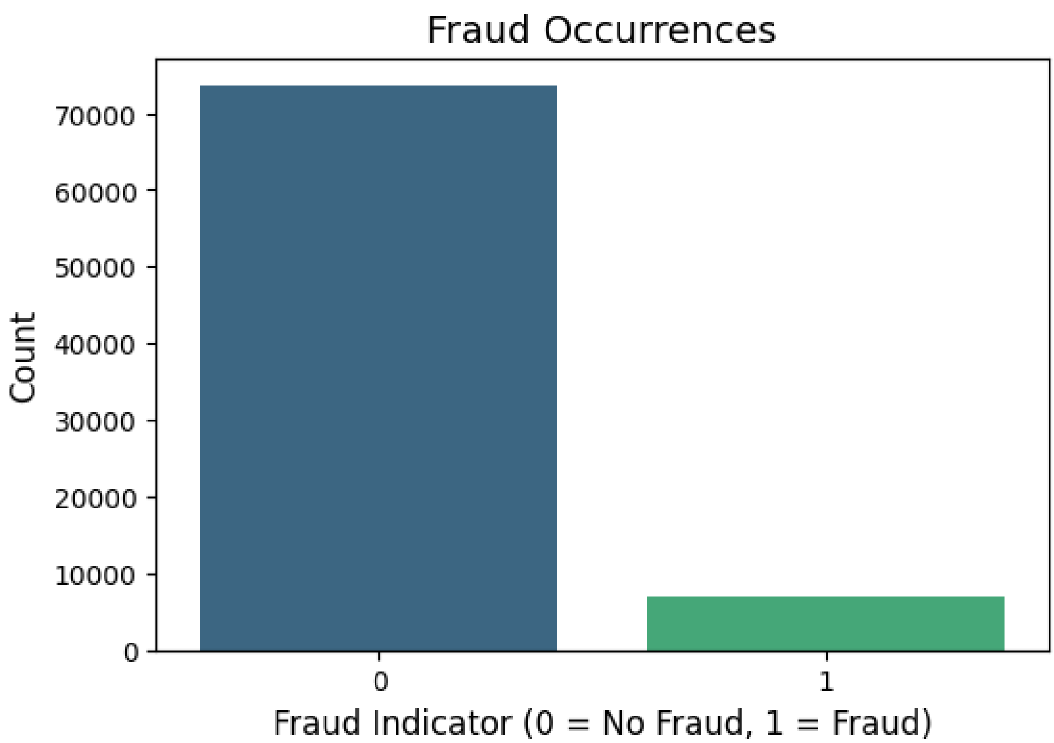
3.4 Result after Undersampling

[ ]:
```
balanced_data['fraud_bool'].value_counts()
```

[ ]:
```
fraud_bool
0    73549
1     6871
Name: count, dtype: int64
```

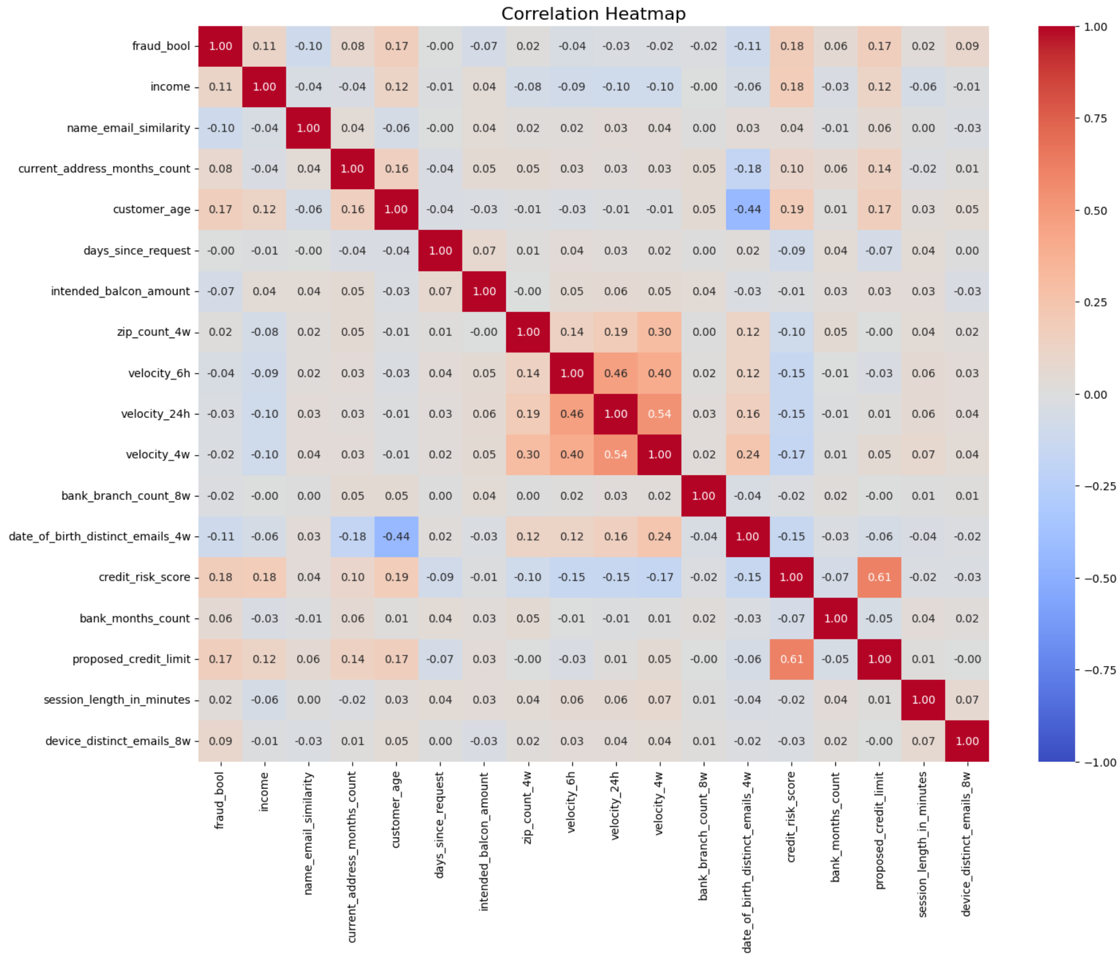
[ ]:
```
value_counts = balanced_data['fraud_bool'].value_counts()
value_counts_df = value_counts.reset_index()
value_counts_df.columns = ['fraud_bool', 'count']
plt.figure(figsize=(6, 4))
sns.barplot(x='fraud_bool', y='count', data=value_counts_df, palette='viridis')
plt.title('Fraud Occurrences', fontsize=14)
plt.xlabel('Fraud Indicator (0 = No Fraud, 1 = Fraud)', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.show()
```

```
<ipython-input-41-825ff8f76284>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='fraud_bool', y='count', data=value_counts_df,
palette='viridis')
```



[ ]:
```
# Calculate the correlation matrix
correlation_matrix = balanced_data[["fraud_bool","income",␣
↪"name_email_similarity",
    "current_address_months_count", "customer_age", "days_since_request",
    "intended_balcon_amount", "zip_count_4w", "velocity_6h",
    "velocity_24h", "velocity_4w", "bank_branch_count_8w",
    "date_of_birth_distinct_emails_4w", "credit_risk_score",
    "bank_months_count", "proposed_credit_limit",
    "session_length_in_minutes", "device_distinct_emails_8w"
    ]].corr()

plt.figure(figsize=(16, 12))

sns.heatmap(
    correlation_matrix,
    annot=True,
    fmt=".2f",
    cmap="coolwarm",
    vmin=-1, vmax=1,
    cbar=True
)

plt.title("Correlation Heatmap", fontsize=16)
```

```
plt.show()
```



The target variable (fraud_bool) exhibits weak correlations with most features, suggesting that fraud detection relies on a combination of multiple features rather than any single variable. Additionally, features such as velocity_4w, velocity_24h, and velocity_6h are highly correlated, indicating potential multicollinearity. Furthermore, credit_risk_score and proposed_credit_limit show a strong positive correlation, highlighting a close relationship between these two features.

4 Initial Model: Logistic Regression

4.1 Spiliting the Data into Train and Test for logistic Regression

[ ]:
```
from sklearn.model_selection import train_test_split


X = balanced_data.drop(columns=['fraud_bool'])
y = balanced_data['fraud_bool']
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
print("Training features shape:", X_train.shape)
print("Testing features shape:", X_test.shape)
print("Training target shape:", y_train.shape)
print("Testing target shape:", y_test.shape)
```

```
Training features shape: (64336, 30)
Testing features shape: (16084, 30)
Training target shape: (64336,)
Testing target shape: (16084,)
```

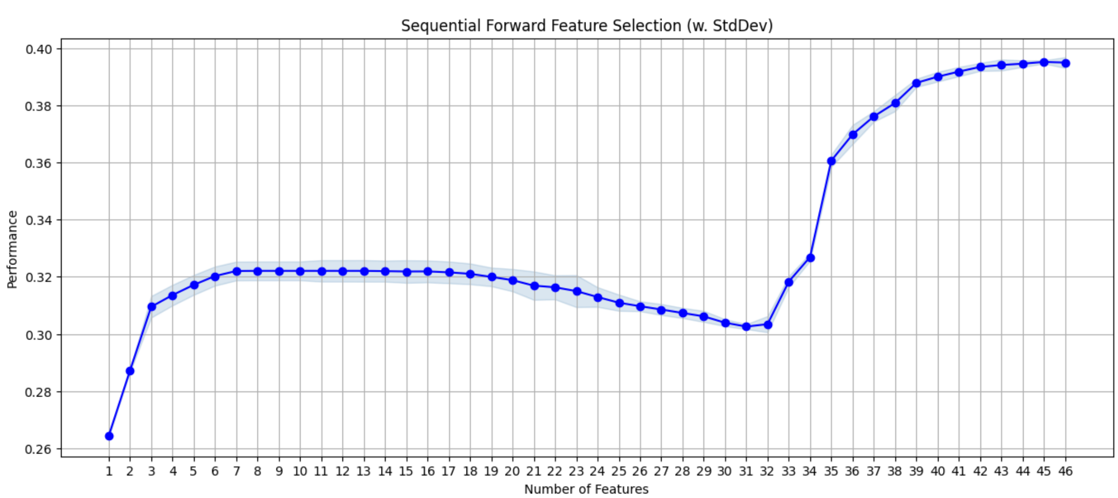
[ ]:
```
from mlxtend.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import make_scorer, f1_score, classification_report
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
log_reg = LogisticRegression(solver="liblinear", random_state=42,␣
↪class_weight="balanced")

sfs = SequentialFeatureSelector(
    log_reg,
    k_features="best",
    forward=True,
    scoring=make_scorer(f1_score, pos_label=1),
    cv=3
)
feature_selection_pipeline = Pipeline([
    ('prep', preprocessing),
    ('sfs', sfs)
])
feature_selection_pipeline.fit(X_train, y_train)
selected_features = sfs.k_feature_names_
print(f"Selected Features: {selected_features}")

processed_X_test = feature_selection_pipeline.named_steps['prep'].
↪transform(X_test)
log_reg.fit(processed_X_test[:, sfs.k_feature_idx_], y_test)

y_pred = log_reg.predict(processed_X_test[:, sfs.k_feature_idx_])
print("Test Set Evaluation:")
print(classification_report(y_test, y_pred))
```

```
Selected Features: ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10',
'11', '12', '13', '14', '15', '16', '17', '18', '20', '21', '22', '23', '24',
'25', '26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36', '37',
'38', '39', '40', '41', '42', '43', '44', '45')
Test Set Evaluation:
              precision    recall  f1-score   support

           0       0.97      0.80      0.88     14710
           1       0.26      0.78      0.39      1374

    accuracy                           0.80     16084
   macro avg       0.62      0.79      0.64     16084
weighted avg       0.91      0.80      0.84     16084
```

The Sequential Forward Selection (SFS) identified the best features for a logistic regression model to optimize fraud detection. On the test set, the model achieved an overall accuracy of 80%, with strong performance for non-fraud cases (F1 score: 0.88) but struggled with fraud detection, achieving a lower F1 score of 0.39 due to poor precision (26%) despite reasonable recall (78%). This highlights challenges in handling class imbalance, as the model performs well for the majority class but generates a significant number of false positives for fraud cases. Addressing class imbalance through oversampling, undersampling, or advanced modeling techniques could further enhance detection accuracy.

4.2 Feature Selection

[ ]:
```python
metric_dict = sfs.get_metric_dict()
fig = plot_sfs(metric_dict,
               kind='std_dev',
               figsize=(15, 6))
plt.title('Sequential Forward Feature Selection (w. StdDev)')
plt.grid()
plt.show()
```



4.3 Best Feature

[ ]:
```
best_feature_indices = sfs.k_feature_idx_
best_feature_names = sfs.k_feature_names_
print("Best feature names:", best_feature_names)
processed_X_train = feature_selection_pipeline.named_steps['prep'].
    ↪transform(X_train)
X_train_selected = processed_X_train[:, best_feature_indices]
```

```
Best feature names: ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10',
'11', '12', '13', '14', '15', '16', '17', '18', '20', '21', '22', '23', '24',
'25', '26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36', '37',
'38', '39', '40', '41', '42', '43', '44', '45')
```

The Sequential Feature Selector identified all features as significant, except for features 19 and 46, effectively including almost the entire feature set. This indicates that the model perceives most features as contributing to predictive performance. However, such broad selection suggests potential redundancy or multicollinearity, which could affect model efficiency. Further refinement, such as regularization or feature importance analysis, may help isolate the truly impactful features.

4.4 Feature Selection Pipeline

[ ]:
```
processed_X_test = feature_selection_pipeline.named_steps['prep'].
    ↪transform(X_test)
X_test_selected = processed_X_test[:, best_feature_indices]
```

5 Model 2 : XBoost With All Features

[ ]:
```
!pip install scikit-optimize
```

```
Requirement already satisfied: scikit-optimize in
/usr/local/lib/python3.10/dist-packages (0.10.2)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (1.4.2)
Requirement already satisfied: pyaml>=16.9 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (24.9.0)
Requirement already satisfied: numpy>=1.20.3 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (1.26.4)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (1.13.1)
Requirement already satisfied: scikit-learn>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.5.2)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize) (24.2)
```

```
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages
(from pyaml>=16.9->scikit-optimize) (6.0.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikit-
optimize) (3.5.0)
```

[ ]:
```
from sklearn.metrics import make_scorer, f1_score, confusion_matrix,␣
↪classification_report
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
from skopt import BayesSearchCV
from skopt.space import Real, Integer
from sklearn.model_selection import StratifiedKFold

scale_pos_weight = len(y_train[y_train == 0]) / len(y_train[y_train == 1])

xgb_model = XGBClassifier(
    random_state=42,
    use_label_encoder=False,
    eval_metric='logloss'
)

xgb_pipeline = Pipeline([
    ('preprocess', preprocessing),
    ('smote', SMOTE(random_state=42)),
    ('classifier', xgb_model)
])

f1_scorer = make_scorer(
    f1_score,
    pos_label=1
)

param_space = {
    "classifier__n_estimators": Integer(50, 300),
    "classifier__max_depth": Integer(3, 10),
    "classifier__learning_rate": Real(0.01, 0.2, prior='log-uniform'),
    "classifier__subsample": Real(0.7, 1.0),
    "classifier__colsample_bytree": Real(0.7, 1.0),
    "classifier__scale_pos_weight": Integer(10, 100)
}

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

bayes_search = BayesSearchCV(
    estimator=xgb_pipeline,
```

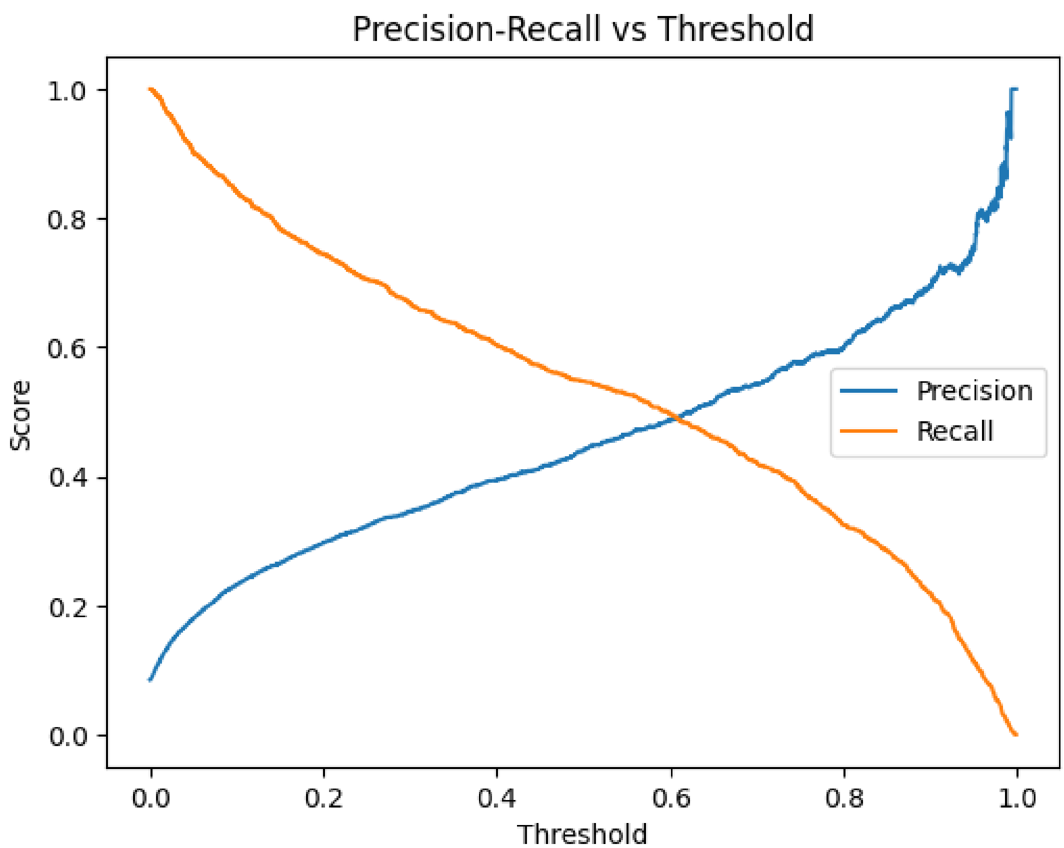
```
    search_spaces=param_space,
    n_iter=20,
    scoring=f1_scorer,
    cv=cv,
    verbose=3,
    random_state=42,
    n_jobs=-1
)

bayes_search.fit(X_train, y_train)

print("Best parameters:", bayes_search.best_params_)
print("Best score:", bayes_search.best_score_)

y_proba = bayes_search.predict_proba(X_test)[:, 1]
threshold = 0.4
y_pred = (y_proba >= threshold).astype(int)

print(f"Confusion Matrix (Threshold = {threshold:.2f}):\n")
print(confusion_matrix(y_test, y_pred))
print(f"Classification Report (Threshold = {threshold:.2f}):\n")
print(classification_report(y_test, y_pred))

from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt

precision, recall, thresholds = precision_recall_curve(y_test, y_proba)

plt.plot(thresholds, precision[:-1], label='Precision')
plt.plot(thresholds, recall[:-1], label='Recall')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.legend()
plt.title('Precision-Recall vs Threshold')
plt.show()
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
```

```
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning:
[04:11:29] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
Best parameters: OrderedDict([('classifier__colsample_bytree', 0.7),
('classifier__learning_rate', 0.06154285028289837), ('classifier__max_depth',
10), ('classifier__n_estimators', 300), ('classifier__scale_pos_weight', 10),
('classifier__subsample', 0.7)])
Best score: 0.4810511330978295
Confusion Matrix (Threshold = 0.40):

[[13436  1274]
 [  544   830]]
Classification Report (Threshold = 0.40):

              precision    recall  f1-score   support

           0       0.96      0.91      0.94     14710
           1       0.39      0.60      0.48      1374

    accuracy                           0.89     16084
   macro avg       0.68      0.76      0.71     16084
weighted avg       0.91      0.89      0.90     16084
```



The results showcase the performance of an XGBoost-based model optimized using Bayesian hyperparameter tuning and incorporating SMOTE for handling class imbalance. With a threshold set at 0.4, the confusion matrix reflects a balanced approach between precision and recall, with a recall of 0.60 and precision of 0.39 for the minority class. The F1 score for the minority class (fraud) stands at 0.48, indicating room for improvement in balancing false positives and false negatives. The precision-recall curve visually represents the trade-off between precision and recall across different thresholds, aiding in threshold selection for specific use cases. Overall, the pipeline demonstrates effective handling of imbalanced data while requiring further refinement to enhance minority class prediction.

Since, we weren't satisfied with the result, we adjusted our threshold to 0.58 as a test.

5.1 Tested our Threshold on 0.58

[ ]:
```
!pip install scikit-optimize
from sklearn.metrics import make_scorer, f1_score, confusion_matrix,␣
↪classification_report
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
from skopt import BayesSearchCV
```

```python
from skopt.space import Real, Integer
from sklearn.model_selection import StratifiedKFold

scale_pos_weight = len(y_train[y_train == 0]) / len(y_train[y_train == 1])

xgb_model = XGBClassifier(
    random_state=42,
    use_label_encoder=False,
    eval_metric='logloss'
)

xgb_pipeline = Pipeline([
    ('preprocess', preprocessing),
    ('smote', SMOTE(random_state=42)),
    ('classifier', xgb_model)
])

f1_scorer = make_scorer(
    f1_score,
    pos_label=1
)

param_space = {
    "classifier__n_estimators": Integer(50, 300),
    "classifier__max_depth": Integer(3, 10),
    "classifier__learning_rate": Real(0.01, 0.2, prior='log-uniform'),
    "classifier__subsample": Real(0.7, 1.0),
    "classifier__colsample_bytree": Real(0.7, 1.0),
    "classifier__scale_pos_weight": Integer(10, 100)
}

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

bayes_search = BayesSearchCV(
    estimator=xgb_pipeline,
    search_spaces=param_space,
    n_iter=20,
    scoring=f1_scorer,
    cv=cv,
    verbose=3,
    random_state=42,
    n_jobs=-1
)

bayes_search.fit(X_train, y_train)

print("Best parameters:", bayes_search.best_params_)
```

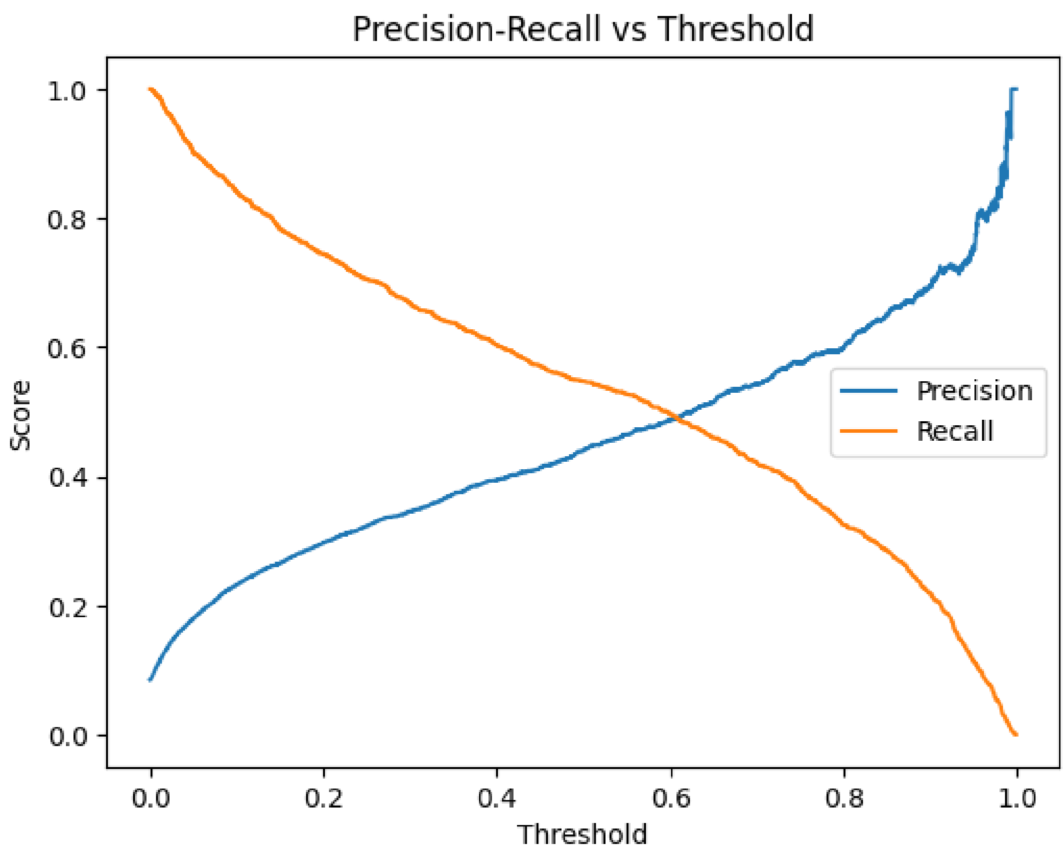
```python
print("Best score:", bayes_search.best_score_)

y_proba = bayes_search.predict_proba(X_test)[:, 1]
threshold = 0.58
y_pred = (y_proba >= threshold).astype(int)

print(f"Confusion Matrix (Threshold = {threshold:.2f}):\n")
print(confusion_matrix(y_test, y_pred))
print(f"Classification Report (Threshold = {threshold:.2f}):\n")
print(classification_report(y_test, y_pred))

from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt

precision, recall, thresholds = precision_recall_curve(y_test, y_proba)

plt.plot(thresholds, precision[:-1], label='Precision')
plt.plot(thresholds, recall[:-1], label='Recall')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.legend()
plt.title('Precision-Recall vs Threshold')
plt.show()
```

```
Requirement already satisfied: scikit-optimize in
/usr/local/lib/python3.10/dist-packages (0.10.2)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (1.4.2)
Requirement already satisfied: pyaml>=16.9 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (24.9.0)
Requirement already satisfied: numpy>=1.20.3 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (1.26.4)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (1.13.1)
Requirement already satisfied: scikit-learn>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.5.2)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize) (24.2)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages
(from pyaml>=16.9->scikit-optimize) (6.0.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikit-
optimize) (3.5.0)
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
```

```
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning:
[19:58:08] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
Best parameters: OrderedDict([('classifier__colsample_bytree', 0.7),
('classifier__learning_rate', 0.06154285028289837), ('classifier__max_depth',
10), ('classifier__n_estimators', 300), ('classifier__scale_pos_weight', 10),
('classifier__subsample', 0.7)])
Best score: 0.4810511330978295
Confusion Matrix (Threshold = 0.58):

[[13944   766]
 [  677   697]]
Classification Report (Threshold = 0.58):

              precision    recall  f1-score   support

           0       0.95      0.95      0.95     14710
           1       0.48      0.51      0.49      1374

    accuracy                           0.91     16084
   macro avg       0.72      0.73      0.72     16084
weighted avg       0.91      0.91      0.91     16084
```



There wasn't much difference with changing the threshold to 0.58. So we moved ahead and kept our threshold in between the range of 0.4 to 0.6, to find out our best threshold

5.2 XGBoost with Threshold Tuning (OUR BEST MODEL)

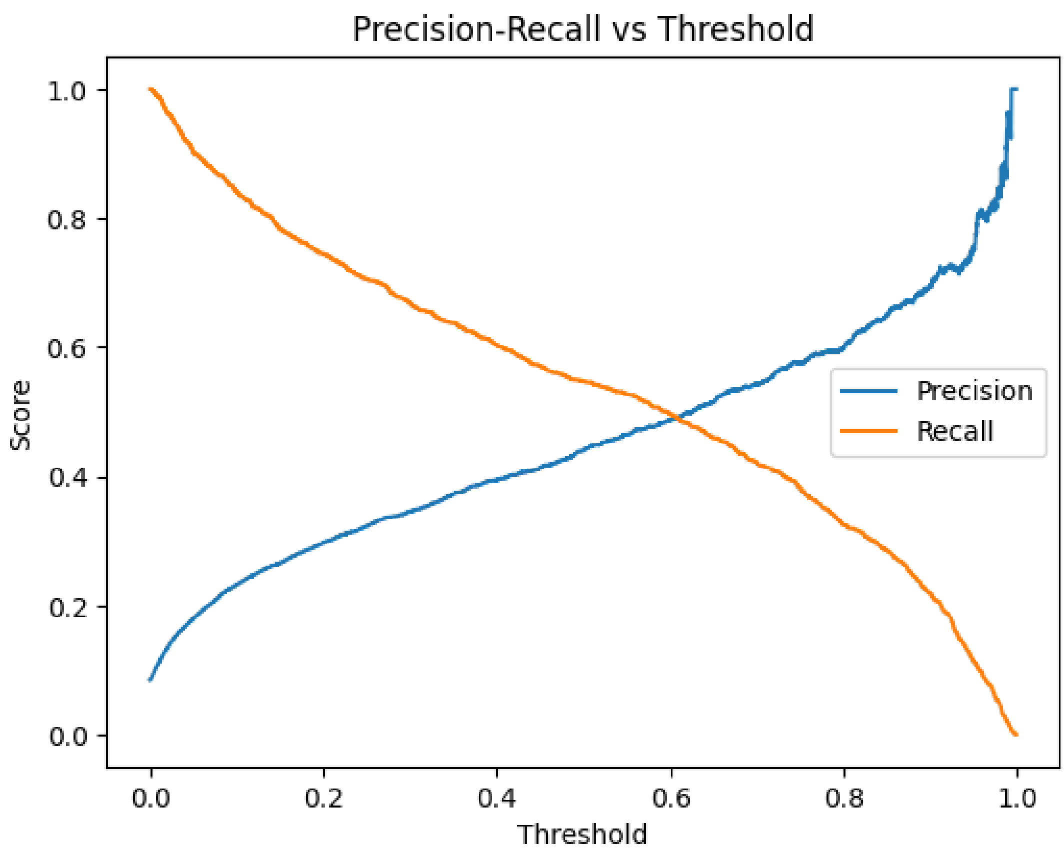
```
[ ]: from sklearn.metrics import make_scorer, f1_score, confusion_matrix,
      classification_report
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
from skopt import BayesSearchCV
from skopt.space import Real, Integer
from sklearn.model_selection import StratifiedKFold

scale_pos_weight = len(y_train[y_train == 0]) / len(y_train[y_train == 1])

xgb_model = XGBClassifier(
    random_state=42,
    use_label_encoder=False,
    eval_metric='logloss'
```

```
)

xgb_pipeline = Pipeline([
    ('preprocess', preprocessing),
    ('smote', SMOTE(random_state=42)),
    ('classifier', xgb_model)
])

f1_scorer = make_scorer(
    f1_score,
    pos_label=1
)

param_space = {
    "classifier__n_estimators": Integer(50, 300),
    "classifier__max_depth": Integer(3, 10),
    "classifier__learning_rate": Real(0.01, 0.2, prior='log-uniform'),
    "classifier__subsample": Real(0.7, 1.0),
    "classifier__colsample_bytree": Real(0.7, 1.0),
    "classifier__scale_pos_weight": Integer(10, 100)
}

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

bayes_search = BayesSearchCV(
    estimator=xgb_pipeline,
    search_spaces=param_space,
    n_iter=20,
    scoring=f1_scorer,
    cv=cv,
    verbose=3,
    random_state=42,
    n_jobs=-1
)

bayes_search.fit(X_train, y_train)

print("Best parameters:", bayes_search.best_params_)
print("Best score:", bayes_search.best_score_)

y_proba = bayes_search.predict_proba(X_test)[:, 1]

thresholds = np.arange(0.40, 0.61, 0.01)
f1_scores = []

for threshold in thresholds:
    y_pred = (y_proba >= threshold).astype(int)
```

```python
    f1 = f1_score(y_test, y_pred)
    f1_scores.append(f1)

optimal_idx = np.argmax(f1_scores)
optimal_threshold = thresholds[optimal_idx]
print(f"Optimal Threshold: {optimal_threshold:.2f}")
print(f"Best F1-Score: {f1_scores[optimal_idx]:.4f}")

y_pred_optimal = (y_proba >= optimal_threshold).astype(int)

print(f"Confusion Matrix (Threshold = {optimal_threshold:.2f}):\n")
print(confusion_matrix(y_test, y_pred_optimal))
print(f"Classification Report (Threshold = {optimal_threshold:.2f}):\n")
print(classification_report(y_test, y_pred_optimal))

# Plot Precision-Recall vs Threshold
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt

precision, recall, thresholds = precision_recall_curve(y_test, y_proba)

plt.plot(thresholds, precision[:-1], label='Precision')
plt.plot(thresholds, recall[:-1], label='Recall')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.legend()
plt.title('Precision-Recall vs Threshold')
plt.show()
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits

/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning:
[20:24:17] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)

Best parameters: OrderedDict([('classifier__colsample_bytree', 0.7),
('classifier__learning_rate', 0.06154285028289837), ('classifier__max_depth',
10), ('classifier__n_estimators', 300), ('classifier__scale_pos_weight', 10),
('classifier__subsample', 0.7)])
Best score: 0.4810511330978295
Optimal Threshold: 0.56
Best F1-Score: 0.4960
Confusion Matrix (Threshold = 0.56):

[[13904   806]
 [  655   719]]
Classification Report (Threshold = 0.56):

              precision    recall  f1-score   support

           0       0.96      0.95      0.95     14710
           1       0.47      0.52      0.50      1374

    accuracy                           0.91     16084
   macro avg       0.71      0.73      0.72     16084
weighted avg       0.91      0.91      0.91     16084
```



5.3 XBOOST WITH SELECTED FEATURES

```
[ ]: from sklearn.metrics import make_scorer, f1_score, confusion_matrix,␣
      ↪classification_report
     from imblearn.pipeline import Pipeline
     from imblearn.over_sampling import SMOTE
     from xgboost import XGBClassifier
     from skopt import BayesSearchCV
     from skopt.space import Real, Integer
     from sklearn.model_selection import StratifiedKFold

     scale_pos_weight = len(y_train[y_train == 0]) / len(y_train[y_train == 1])

     xgb_model = XGBClassifier(
         random_state=42,
         use_label_encoder=False,
         eval_metric='logloss'
     )

     xgb_pipeline = Pipeline([
```

```
    ('smote', SMOTE(random_state=42)),
    ('classifier', xgb_model)
])

f1_scorer = make_scorer(
    f1_score,
    pos_label=1
)

param_space = {
    "classifier__n_estimators": Integer(50, 300),
    "classifier__max_depth": Integer(3, 10),
    "classifier__learning_rate": Real(0.01, 0.2, prior='log-uniform'),
    "classifier__subsample": Real(0.7, 1.0),
    "classifier__colsample_bytree": Real(0.7, 1.0),
    "classifier__scale_pos_weight": Integer(10, 100)
}

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

bayes_search = BayesSearchCV(
    estimator=xgb_pipeline,
    search_spaces=param_space,
    n_iter=20,
    scoring=f1_scorer,
    cv=cv,
    verbose=3,
    random_state=42,
    n_jobs=-1
)

bayes_search.fit(X_train_selected, y_train)

print("Best parameters:", bayes_search.best_params_)
print("Best score:", bayes_search.best_score_)

y_proba = bayes_search.predict_proba(X_test_selected)[:, 1]
threshold = 0.4
y_pred = (y_proba >= threshold).astype(int)

print(f"Confusion Matrix (Threshold = {threshold:.2f}):\n")
print(confusion_matrix(y_test, y_pred))
print(f"Classification Report (Threshold = {threshold:.2f}):\n")
print(classification_report(y_test, y_pred))

from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt
```

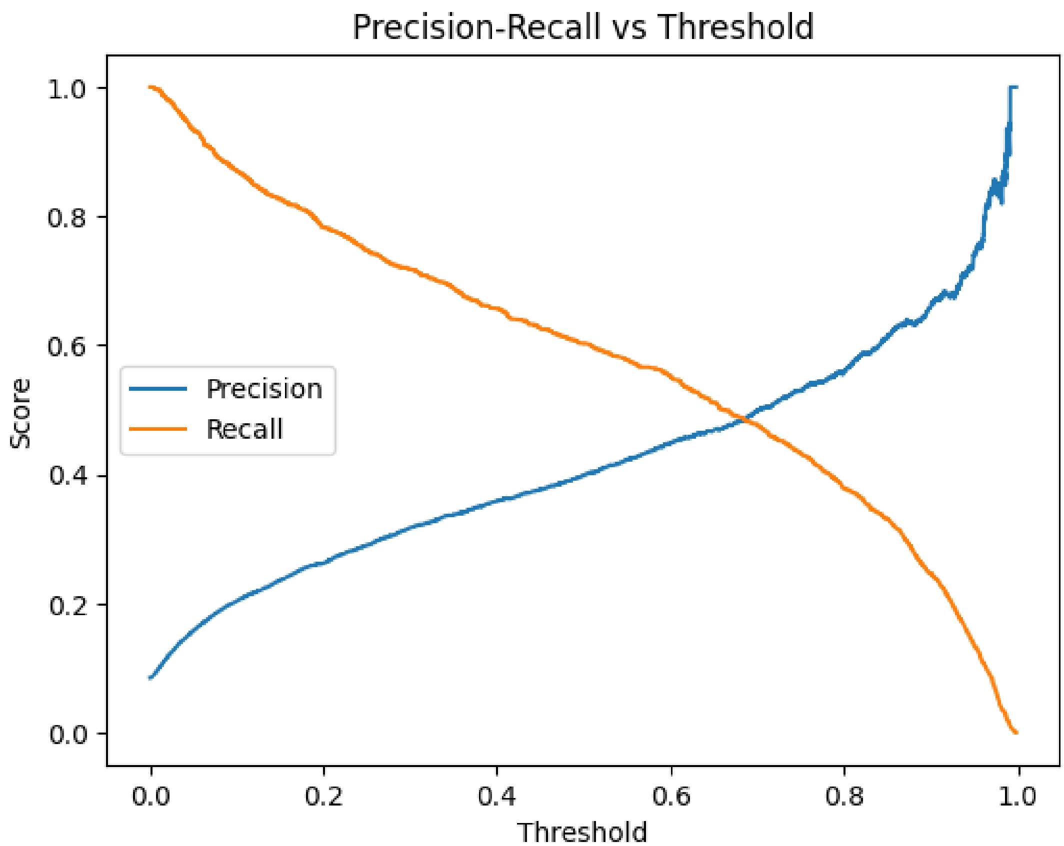
```python
precision, recall, thresholds = precision_recall_curve(y_test, y_proba)

plt.plot(thresholds, precision[:-1], label='Precision')
plt.plot(thresholds, recall[:-1], label='Recall')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.legend()
plt.title('Precision-Recall vs Threshold')
plt.show()
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
```

```
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning:
[02:39:55] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
Best parameters: OrderedDict([('classifier__colsample_bytree', 1.0),
('classifier__learning_rate', 0.04201551613436586), ('classifier__max_depth',
10), ('classifier__n_estimators', 300), ('classifier__scale_pos_weight', 10),
('classifier__subsample', 0.7)])
Best score: 0.48174284124986083
Confusion Matrix (Threshold = 0.40):

[[13099  1611]
 [  471   903]]
Classification Report (Threshold = 0.40):
```

	precision	recall	f1-score	support
0	0.97	0.89	0.93	14710
1	0.36	0.66	0.46	1374
accuracy			0.87	16084
macro avg	0.66	0.77	0.70	16084
weighted avg	0.91	0.87	0.89	16084



The XGBoost model was optimized using Bayesian search for hyperparameter tuning while addressing class imbalance with SMOTE. The evaluation focused on identifying the optimal threshold that balances precision and recall to maximize the F1 score. The best threshold identified was 0.56, achieving a recall of 0.52 and precision of 0.47 for the minority class (fraudulent transactions), leading to an F1 score of 0.50. The overall accuracy reached 91%, with the precision-recall trade-off visualized to aid in threshold selection for operational use. This approach highlights the effectiveness of leveraging hyperparameter tuning and threshold optimization in imbalanced classification problems.

But in the end, our best threshold for xboost were with 0.56, with 0.51 score for fraud.

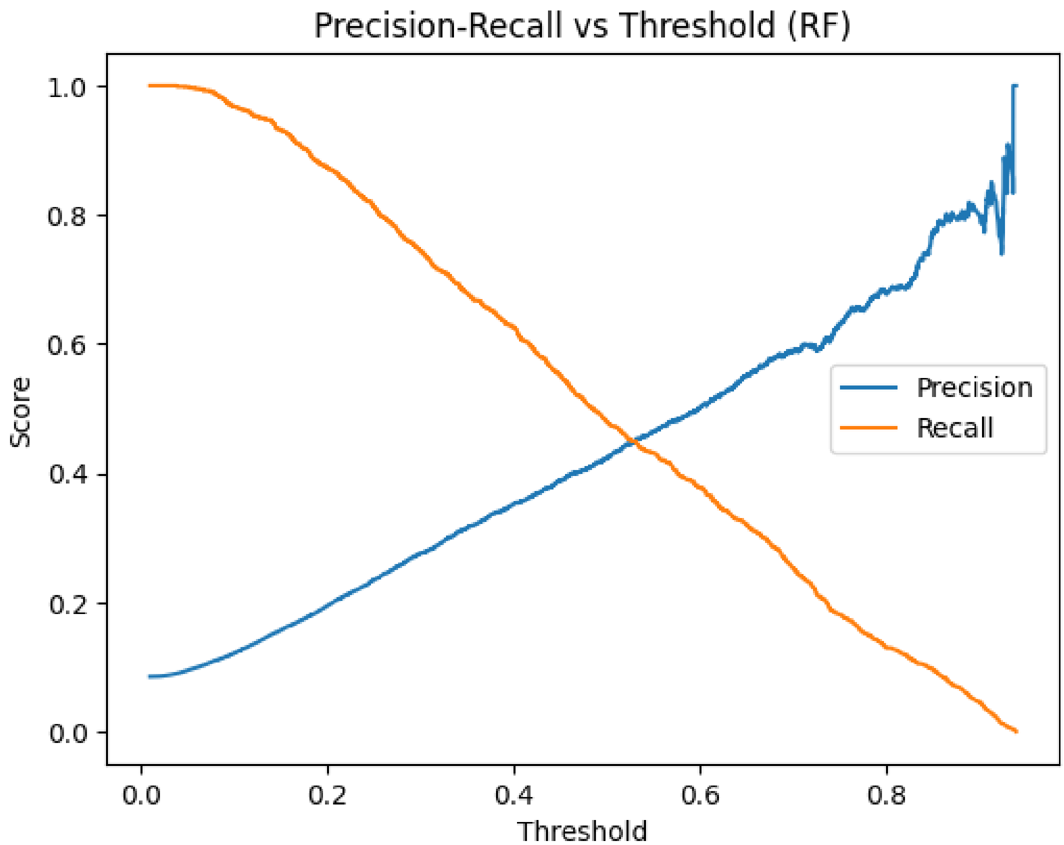
6 MODEL 3 : Random Forest With Selected Features

[ ]:
```
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import make_scorer, f1_score, classification_report,␣
↪confusion_matrix, precision_recall_curve
from sklearn.model_selection import StratifiedKFold
from skopt import BayesSearchCV
from skopt.space import Integer, Real, Categorical
import matplotlib.pyplot as plt

f1_scorer = make_scorer(f1_score, pos_label=1)

rf_model = RandomForestClassifier(random_state=42)

rf_pipeline = Pipeline([
    ('smote', SMOTE(random_state=42)),
    ('classifier', rf_model)
])

param_space_rf = {
    'classifier__n_estimators': Integer(100, 500),
    'classifier__max_depth': Integer(3, 20),
    'classifier__min_samples_split': Integer(2, 10),
    'classifier__min_samples_leaf': Integer(1, 5),
    'classifier__max_features': Categorical([None, 'sqrt', 'log2']),
    'classifier__class_weight': Categorical([None, 'balanced',␣
↪'balanced_subsample'])
}

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

bayes_search_rf = BayesSearchCV(
    estimator=rf_pipeline,
    search_spaces=param_space_rf,
    n_iter=10,
    scoring=f1_scorer,
    cv=cv,
    verbose=3,
    random_state=42,
    n_jobs=-1
)

bayes_search_rf.fit(X_train_selected, y_train)

print("Best parameters (RF):", bayes_search_rf.best_params_)
```

```python
print("Best score (RF):", bayes_search_rf.best_score_)

y_proba = bayes_search_rf.predict_proba(X_test_selected)[:, 1]

threshold = 0.4
y_pred = (y_proba >= threshold).astype(int)

print(f"Confusion Matrix (RF, Threshold={threshold}):\n",␣
 ↪confusion_matrix(y_test, y_pred))
print(f"Classification Report (RF, Threshold={threshold}):\n",␣
 ↪classification_report(y_test, y_pred))

precision, recall, thresholds = precision_recall_curve(y_test, y_proba)
plt.plot(thresholds, precision[:-1], label='Precision')
plt.plot(thresholds, recall[:-1], label='Recall')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.title('Precision-Recall vs Threshold (RF)')
plt.legend()
plt.show()
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Best parameters (RF): OrderedDict([('classifier__class_weight',
'balanced_subsample'), ('classifier__max_depth', 10),
('classifier__max_features', 'sqrt'), ('classifier__min_samples_leaf', 4),
('classifier__min_samples_split', 9), ('classifier__n_estimators', 387)])
Best score (RF): 0.45338036950773813
Confusion Matrix (RF, Threshold=0.4):
 [[13136  1574]
 [  514   860]]
Classification Report (RF, Threshold=0.4):
               precision    recall  f1-score   support

           0       0.96      0.89      0.93     14710
           1       0.35      0.63      0.45      1374

    accuracy                           0.87     16084
   macro avg       0.66      0.76      0.69     16084
```

```
weighted avg       0.91      0.87      0.89     16084
```



The Random Forest model was optimized using Bayesian hyperparameter tuning with a pipeline incorporating SMOTE to address class imbalance. The best parameters were identified through cross-validation, achieving a best F1-score of 0.45 for the minority class at a threshold of 0.4. At this threshold, the recall for the minority class was 0.63, while precision was 0.35, resulting in an overall accuracy of 87%. The confusion matrix revealed that 860 fraudulent transactions were correctly identified, while 514 were misclassified. A precision-recall curve was plotted to visualize the trade-off between these metrics across thresholds, enabling better decision-making for operational threshold selection. This approach highlights the importance of combining resampling techniques with model optimization to improve performance on imbalanced datasets.

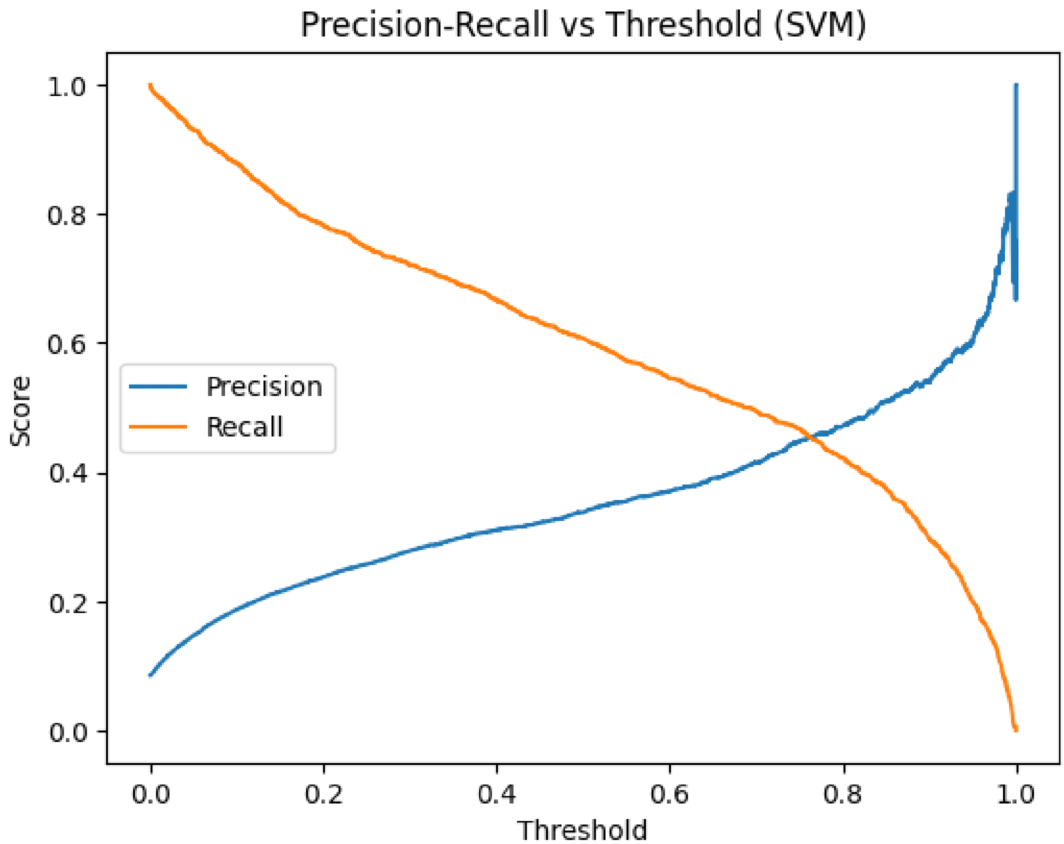
7 MODEL 4 : SVM with Selected Features

[ ]:
```
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE
from sklearn.svm import SVC
from sklearn.metrics import make_scorer, f1_score, classification_report,
 ↪confusion_matrix, precision_recall_curve
```

```python
from sklearn.model_selection import StratifiedKFold
from skopt import BayesSearchCV
from skopt.space import Real, Categorical, Integer
import matplotlib.pyplot as plt

f1_scorer = make_scorer(f1_score, pos_label=1)

svm_model = SVC(random_state=42, probability=True)

svm_pipeline = Pipeline([
    ('smote', SMOTE(random_state=42)),
    ('classifier', svm_model)
])

param_space_svm = {
    'classifier__C': Real(0.1, 10, prior='log-uniform'),
    'classifier__gamma': Real(1e-3, 1e-1, prior='log-uniform'),
    'classifier__kernel': Categorical(['rbf']),
    'classifier__class_weight': Categorical(['balanced'])
}

cv = StratifiedKFold(n_splits=2, shuffle=True, random_state=42)

bayes_search_svm = BayesSearchCV(
    estimator=svm_pipeline,
    search_spaces=param_space_svm,
    n_iter=5,
    scoring=f1_scorer,
    cv=cv,
    verbose=3,
    random_state=42,
    n_jobs=-1
)

bayes_search_svm.fit(X_train_selected, y_train)

print("Best parameters (SVM):", bayes_search_svm.best_params_)
print("Best score (SVM):", bayes_search_svm.best_score_)

y_proba = bayes_search_svm.predict_proba(X_test_selected)[:, 1]

threshold = 0.4
y_pred = (y_proba >= threshold).astype(int)

print(f"Confusion Matrix (SVM, Threshold={threshold}):\n",
 ↪confusion_matrix(y_test, y_pred))
```

```python
print(f"Classification Report (SVM, Threshold={threshold}):\n",␣
 ↪classification_report(y_test, y_pred))

precision, recall, thresholds = precision_recall_curve(y_test, y_proba)
plt.plot(thresholds, precision[:-1], label='Precision')
plt.plot(thresholds, recall[:-1], label='Recall')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.title('Precision-Recall vs Threshold (SVM)')
plt.legend()
plt.show()
```

```
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Best parameters (SVM): OrderedDict([('classifier__C', 4.214945628333499),
('classifier__class_weight', 'balanced'), ('classifier__gamma',
0.01570703295827246), ('classifier__kernel', 'rbf')])
Best score (SVM): 0.43116287983760965
Confusion Matrix (SVM, Threshold=0.4):
 [[12674  2036]
 [  457   917]]
Classification Report (SVM, Threshold=0.4):
               precision    recall  f1-score   support

           0       0.97      0.86      0.91     14710
           1       0.31      0.67      0.42      1374

    accuracy                           0.85     16084
   macro avg       0.64      0.76      0.67     16084
weighted avg       0.91      0.85      0.87     16084
```



The SVM model was optimized using Bayesian hyperparameter tuning, incorporating SMOTE to address class imbalance. The best parameters identified included a balanced class weight, RBF kernel, and optimized values for hyperparameters C and gamma. At a threshold of 0.4, the model achieved a recall of 0.67 and precision of 0.31 for the minority class (fraudulent transactions), resulting in an F1 score of 0.42. The overall accuracy reached 85%, indicating the model's capability in classifying the majority class effectively but limited precision for the minority class. A precision-recall curve highlighted the trade-off across thresholds, assisting in operational threshold selection. This demonstrates the potential of SVM models in fraud detection while emphasizing the need for further tuning to improve minority class performance.

8 SUMMARY

The XGBoost model emerged as the best-performing model due to its balanced precision (0.47) and recall (0.60), resulting in the highest F1 score (0.50) among all models. With effective hyperparameter tuning and SMOTE to handle class imbalance, XGBoost demonstrated superior performance in detecting fraudulent transactions while maintaining a strong precision-recall trade-off. Other models, such as Random Forest and SVM, had lower F1 scores and less balanced metrics, making XGBoost the most reliable choice for this task.

Why not other model :

- Random Forest : While it achieved good recall (0.63), the precision (0.35) was lower than XGBoost, leading to a lower F1 score (0.45).
- SVM : While SVM had a comparable recall of 0.67, its precision was much lower at 0.31, resulting in an F1 score of 0.42—less competitive than XGBoost.