1) SVM model

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from imblearn.over_sampling import SMOTE
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import RFECV
from imblearn.pipeline import make_pipeline
from google.colab import drive

# Load the data from the CSV file
drive.mount('/content/drive')
data = pd.read_csv('/content/drive/MyDrive/mac_combined_May_17.csv')

# Separate the features (X) and Flags (y)
X = data.drop(columns=['Website', 'Flag'])
y = data['Flag']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Define the pipeline to handle class imbalance using SMOTE, standardize features, and
train the SVM model
pipe = make_pipeline(SMOTE(random_state=42), StandardScaler(), SVC(kernel='rbf',
class_weight='balanced'))

# Hyperparameter tuning using GridSearchCV with wider range of hyperparameters
param_grid = {'svc__C': [0.01, 0.1, 1, 10, 100, 1000, 10000],
          'svc__gamma': [10, 1, 0.1, 0.01, 0.001, 0.0001, 0.00001]}
grid = GridSearchCV(pipe, param_grid, verbose=3, cv=StratifiedKFold(5))
grid.fit(X_train, y_train)

# Print the best parameters
print("Best parameters found by GridSearchCV:")
print(grid.best_params_)

# Make predictions on the test set
y_pred = grid.predict(X_test)

# Print classification report
```

```
print(classification_report(y_test, y_pred))

Best parameters found by GridSearchCV:
{'svc__C': 10000, 'svc__gamma': 1}
              precision    recall  f1-score   support

           0       0.97      0.64      0.77      4070
           1       0.73      0.98      0.84      4121

    accuracy                           0.81      8191
   macro avg       0.85      0.81      0.80      8191
weighted avg       0.85      0.81      0.80      8191
```

2) Logistic Regression Model

```
from google.colab import drive
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import time
from tabulate import tabulate
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import RFE
from sklearn.model_selection import GridSearchCV

# Load the data
print("Loading data...")
start_time = time.time()
drive.mount('/content/drive')
df = pd.read_csv('/content/drive/MyDrive/mac_combined_May_17.csv')
print(f"Data loaded in {time.time() - start_time:.2f} seconds.\n")

# Extract feature columns
features = df.drop(['Website', 'Flag'], axis=1)

# Extract target column 'Flag'
target = df['Flag']

# Split the data into training set and test set
```

```
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.3,
random_state=42)

# Initialize the StandardScaler
scaler = StandardScaler()

# Scale the features
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Balance the dataset with SMOTE
print("Balancing the dataset...")
sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)

# Feature selection with Recursive Feature Elimination
print("Applying Recursive Feature Elimination...")
model = LogisticRegression(max_iter=1000)
rfe = RFE(estimator=model, n_features_to_select=20)  # choose the top 10 features
rfe = rfe.fit(X_train_res, y_train_res)
X_train_res = rfe.transform(X_train_res)
X_test = rfe.transform(X_test)

# Hyperparameter tuning with GridSearchCV
print("Tuning hyperparameters...")
parameters = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}
clf = GridSearchCV(model, parameters, cv=5)
clf.fit(X_train_res, y_train_res)
print(f"Best parameters: {clf.best_params_}")

# Train the model with best parameters
print("Training the model with best parameters...")
model = LogisticRegression(max_iter=1000, C=clf.best_params_['C'])
model.fit(X_train_res, y_train_res)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = metrics.accuracy_score(y_test, y_pred)
classification_report = metrics.classification_report(y_test, y_pred)

print("Model Evaluation:\n")
print(f"Accuracy: {accuracy}")
```

```
print("\nClassification Report:")
print(tabulate(pd.DataFrame(metrics.classification_report(y_test, y_pred,
output_dict=True)).transpose(), headers='keys', tablefmt='psql'))
```

```
Model Evaluation:

Accuracy: 0.7814674642900745

Classification Report:
+--------------+-------------+----------+------------+-------------+
|              |   precision |   recall |   f1-score |     support |
|--------------+-------------+----------+------------+-------------|
| 0            |    0.985106 | 0.568796 |   0.721184 | 4070        |
| 1            |    0.699538 | 0.991507 |   0.820317 | 4121        |
| accuracy     |    0.781467 | 0.781467 |   0.781467 |    0.781467 |
| macro avg    |    0.842322 | 0.780151 |   0.770751 | 8191        |
| weighted avg |    0.841433 | 0.781467 |   0.771059 | 8191        |
```

3) Naïve Bayes Model

```
from google.colab import drive
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif

drive.mount('/content/drive')
data = pd.read_csv('/content/drive/MyDrive/mac_combined_May_17.csv')

# Drop the columns with constant values
constant_columns = [col for col in data.columns if data[col].nunique() <= 1]
data = data.drop(columns=constant_columns)

# Feature Selection - Select the top K most informative features
X = data.drop(columns=['Website', 'Flag'])
y = data['Flag']
selector = SelectKBest(score_func=f_classif, k=10)
X_selected = selector.fit_transform(X, y)

# Data Preprocessing - Scale numerical features
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_selected)

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3,
random_state=42)

# Hyperparameter Tuning - Perform GridSearchCV to find the best hyperparameters
param_grid = {'var_smoothing': [1e-9, 1e-8, 1e-7]}
model = GaussianNB()
grid_search = GridSearchCV(model, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Train the model with the best hyperparameters
best_model = grid_search.best_estimator_
best_model.fit(X_train, y_train)

# Cross-Validation - Evaluate the model using StratifiedKFold cross-validation
cv = StratifiedKFold(n_splits=5)
accuracy_scores = []
classification_reports = []
for train_index, test_index in cv.split(X_scaled, y):
    X_train_cv, X_test_cv = X_scaled[train_index], X_scaled[test_index]
    y_train_cv, y_test_cv = y.iloc[train_index], y.iloc[test_index]
    model_cv = GaussianNB(var_smoothing=best_model.var_smoothing)
    model_cv.fit(X_train_cv, y_train_cv)
    y_pred_cv = model_cv.predict(X_test_cv)
    accuracy_scores.append(accuracy_score(y_test_cv, y_pred_cv))
    classification_reports.append(classification_report(y_test_cv, y_pred_cv))

# Predict on the test set
y_pred = best_model.predict(X_test)

# Print accuracy and classification report
accuracy = accuracy_score(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print results
print("Model Performance:")
print("Metric               Score")
print("---------------------  ---------------------------------------------------")
print(f"Accuracy             {accuracy}")
print("Classification Report\n", class_report)
```

```
# Print cross-validation results
print("Cross-Validation Results:")
print("Fold    Accuracy")
print("----------------")
for i in range(len(accuracy_scores)):
    print(f"{i+1:<8} {accuracy_scores[i]:.4f}")
```

```
Model Performance:
Metric               Score
---------------------  --------------------------------------------------
Accuracy             0.7651080454157002
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.53      0.69      4070
           1       0.68      1.00      0.81      4121

    accuracy                           0.77      8191
   macro avg       0.84      0.76      0.75      8191
weighted avg       0.84      0.77      0.75      8191

Cross-Validation Results:
Fold    Accuracy
----------------
1       0.7982
2       0.7790
3       0.7663
4       0.7509
5       0.7496
```

4) KNN Model

```
from google.colab import drive
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.decomposition import PCA
from tabulate import tabulate
import matplotlib.pyplot as plt
import time

# Load the dataset
```

```
print("Loading data...")
start_time = time.time()
drive.mount('/content/drive')
data = pd.read_csv('/content/drive/MyDrive/mac_combined_May_17.csv')
print(f"Data loaded in {time.time() - start_time:.2f} seconds.\n")

# Drop the columns with constant values
constant_columns = [col for col in data.columns if data[col].nunique() <= 1]
data = data.drop(columns=constant_columns)

# Split into features and target
X = data.drop(columns=['Website', 'Flag'])
y = data['Flag']

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply Polynomial Features
poly_features = PolynomialFeatures(degree=2)
X_train_poly = poly_features.fit_transform(X_train_scaled)
X_test_poly = poly_features.transform(X_test_scaled)

# Apply PCA for dimensionality reduction
pca = PCA(n_components=0.95)  # Retain 95% of the variance
X_train_pca = pca.fit_transform(X_train_poly)
X_test_pca = pca.transform(X_test_poly)

# Define the parameter grid for KNN
param_grid_knn = {'n_neighbors': [3, 5, 7]}

# Perform grid search for KNN
grid_search_knn = GridSearchCV(KNeighborsClassifier(), param_grid_knn)
grid_search_knn.fit(X_train_pca, y_train)

# Get the best KNN model and its parameters
best_knn_model = grid_search_knn.best_estimator_
best_knn_params = grid_search_knn.best_params_

# Train the best KNN model
```

```
print("Training the model...")
start_time = time.time()
best_knn_model.fit(X_train_pca, y_train)
print(f"Model trained in {time.time() - start_time:.2f} seconds.\n")

# Predict on the test set
y_pred = best_knn_model.predict(X_test_pca)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
confusion_mat = confusion_matrix(y_test, y_pred)

# Print accuracy, classification report, and confusion matrix
print("Model Performance:")
print("Accuracy:", accuracy)
print("Classification Report:\n", class_report)
print("Confusion Matrix:\n", confusion_mat)

# Print results in a tabular format
print("Model Performance:")
print(tabulate([
    ["Accuracy", accuracy],
    ["Classification Report", "\n" + class_report]
], headers=["Metric", "Score"]))

# Plot confusion matrix
plt.figure()
plt.imshow(confusion_mat, cmap='Blues')
plt.title("Confusion Matrix")
plt.colorbar()
plt.xticks([0, 1], ['Class 0', 'Class 1'])
plt.yticks([0, 1], ['Class 0', 'Class 1'])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
Model Performance:
Accuracy: 0.8042973995849103
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.64      0.77      4070
           1       0.73      0.96      0.83      4121
```

```
   accuracy                       0.80      8191
  macro avg       0.84      0.80      0.80      8191
weighted avg       0.84      0.80      0.80      8191

Confusion Matrix:
 [[2625 1445]
 [ 158 3963]]
Model Performance:
Metric                Score
---------------------  ---------------------------------------------------
Accuracy              0.8042973995849103
Classification Report  precision    recall  f1-score   support

                           0       0.94      0.64      0.77      4070
                           1       0.73      0.96      0.83      4121

                    accuracy                           0.80      8191
                   macro avg       0.84      0.80      0.80      8191
                weighted avg       0.84      0.80      0.80      8191
```

5) Decision Tree

```
from google.colab import drive
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
from tabulate import tabulate
import time
# Load the dataset
print("Loading data...")
start_time = time.time()
drive.mount('/content/drive')
data = pd.read_csv('/content/drive/MyDrive/mac_combined_May_17.csv')
print(f"Data loaded in {time.time() - start_time:.2f} seconds.\n")
# Drop the columns with constant values
constant_columns = [col for col in data.columns if data[col].nunique() <= 1]
data = data.drop(columns=constant_columns)

# Split into features and target
X = data.drop(columns=['Website', 'Flag'])
y = data['Flag']
```

```
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Train the model
print("Training the model...")
start_time = time.time()
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
print(f"Model trained in {time.time() - start_time:.2f} seconds.\n")
# Predict on the test set
y_pred = model.predict(X_test)

# Print accuracy and classification report
accuracy = accuracy_score(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print results in a tabular format
print("Model Performance:")
print(tabulate([
   ["Accuracy", accuracy],
   ["Classification Report", "\n" + class_report]
], headers=["Metric", "Score"]))
```

```
Model Performance:
Metric              Score
--------------------  ----------------------------------------------------
Accuracy              0.8315224026370407
Classification Report  precision    recall  f1-score   support

                   0       0.97      0.68      0.80      4070
                   1       0.76      0.98      0.85      4121

            accuracy                           0.83      8191
           macro avg       0.86      0.83      0.83      8191
        weighted avg       0.86      0.83      0.83      8191
```

6) Random Forest

```
from google.colab import drive
import pandas as pd
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from tabulate import tabulate
from sklearn.preprocessing import LabelEncoder
```

```
import time
import numpy as np

# Load the dataset
print("Loading data...")
start_time = time.time()
drive.mount('/content/drive')
data = pd.read_csv('/content/drive/MyDrive/mac_combined_May_17.csv')
print(f"Data loaded in {time.time() - start_time:.2f} seconds.\n")

# Convert categorical features to numeric
le = LabelEncoder()
for col in data.columns:
    if data[col].dtype == 'object':
        data[col] = le.fit_transform(data[col].astype(str))

# Split into features and target
target_col = 'Flag'  # Update this to your actual target column name
X = data.drop(columns=[target_col])
y = data[target_col]

# Train a RandomForest model to compute feature importance
print("Computing feature importance...")
model = RandomForestClassifier(n_estimators=100)
model.fit(X, y)

# Select the top 20 most important features
important_features = np.argsort(model.feature_importances_)[-20:]

# Select only the important features from X
X = X[X.columns[important_features]]

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Define the parameter grid for the random search
param_grid = {
    'n_estimators': [100, 200, 500, 1000],
    'max_depth': [10, 15, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}
```

```
# Initialize a RandomizedSearchCV object
random_search = RandomizedSearchCV(
    estimator=RandomForestClassifier(),
    param_distributions=param_grid,
    cv=5,
    n_iter=10,  # Number of random combinations to try
    n_jobs=-1,
    random_state=42
)

print("Performing random search...")
start_time = time.time()
random_search.fit(X_train, y_train)
print(f"Random search completed in {time.time() - start_time:.2f} seconds.\n")

# Print the best parameters
print(f"Best parameters: {random_search.best_params_}")

# Train the model with the best parameters
print("Training the model...")
start_time = time.time()
model = RandomForestClassifier(**random_search.best_params_)
model.fit(X_train, y_train)
print(f"Model trained in {time.time() - start_time:.2f} seconds.\n")

# Evaluate cross-validated results
cv_results = random_search.cv_results_
print("Cross-Validation Results:")
print(tabulate([
    ["Mean Train Score", "-"],
    ["Mean Test Score", np.mean(cv_results['mean_test_score'])]
], headers=["Metric", "Score"]))

# Predict on the test set
y_pred = model.predict(X_test)

# Print accuracy and classification report
accuracy = accuracy_score(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print results in a tabular format
print("Model Performance:")
print(tabulate([
    ["Accuracy", accuracy],
```

```
    ["Classification Report", "\n" + class_report]
], headers=["Metric", "Score"]))
```

```
Best parameters: {'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 1,
'max_depth': None, 'bootstrap': False}
Training the model...
Model trained in 3.37 seconds.

Cross-Validation Results:
Metric           Score
---------------  ------------------
Mean Train Score  -
Mean Test Score   0.8500545364728316
Model Performance:
Metric                 Score
---------------------  ------------------------------------------------------
Accuracy               0.8791356366743988
Classification Report  precision    recall  f1-score   support

                  0       0.88      0.88      0.88      4070
                  1       0.88      0.88      0.88      4121

           accuracy                           0.88      8191
          macro avg       0.88      0.88      0.88      8191
       weighted avg       0.88      0.88      0.88      8191
```

7) XGBoost

```
from google.colab import drive
import pandas as pd
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report
from tabulate import tabulate
from sklearn.preprocessing import LabelEncoder
import time
import numpy as np

# Load the dataset
print("Loading data...")
start_time = time.time()
drive.mount('/content/drive')
data = pd.read_csv('/content/drive/MyDrive/mac_combined_May_17.csv')
print(f"Data loaded in {time.time() - start_time:.2f} seconds.\n")
```

```
# Convert categorical features to numeric
le = LabelEncoder()
for col in data.columns:
    if data[col].dtype == 'object':
        data[col] = le.fit_transform(data[col].astype(str))

# Split into features and target
target_col = 'Flag'  # Update this to your actual target column name
X = data.drop(columns=[target_col])
y = data[target_col]

# Train an XGBoost model to compute feature importance
print("Computing feature importance...")
model = XGBClassifier(use_label_encoder=False, eval_metric='logloss',
tree_method='gpu_hist')
model.fit(X, y)

# Select the top 20 most important features
important_features = np.argsort(model.feature_importances_)[-20:]

# Select only the important features from X
X = X[X.columns[important_features]]

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the model
print("Training the model...")
start_time = time.time()
model = XGBClassifier(use_label_encoder=False, eval_metric='logloss',
tree_method='gpu_hist')
model.fit(X_train, y_train)
print(f"Model trained in {time.time() - start_time:.2f} seconds.\n")

# Predict on the test set
y_pred = model.predict(X_test)

# Print accuracy and classification report
accuracy = accuracy_score(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print results in a tabular format
print("Model Performance:")
```

```
print(tabulate([
    ["Accuracy", accuracy],
    ["Classification Report", "\n" + class_report]
], headers=["Metric", "Score"]))

Model Performance:
Metric                Score
---------------------  ---------------------------------------------------
Accuracy              0.8550848492247589
Classification Report  precision    recall  f1-score   support

                   0       0.92      0.78      0.84      4070
                   1       0.81      0.93      0.87      4121

            accuracy                           0.86      8191
           macro avg       0.86      0.85      0.85      8191
        weighted avg       0.86      0.86      0.85      8191
```

8) Ensemble_Random_forest_XGBoost

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.metrics import accuracy_score, classification_report
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
from google.colab import drive

# Load the dataset
print("Loading data...")
start_time = time.time()
drive.mount('/content/drive')
data = pd.read_csv('/content/drive/MyDrive/mac_combined_May_17.csv')
print(f"Data loaded in {time.time() - start_time:.2f} seconds.\n")

# Define target and drop it from main data
y = data['Flag']
X = data.drop(['Flag'], axis=1)

# Encoding the Website feature
le = LabelEncoder()
```

```
X['Website'] = le.fit_transform(X['Website'])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Define classifiers with 'balanced' class weights
rf_clf = RandomForestClassifier(class_weight='balanced', random_state=42)
xgb_clf = XGBClassifier(use_label_encoder=False, eval_metric='logloss', verbosity=2,
tree_method='gpu_hist')

# Parameters for GridSearchCV
parameters_rf = {
    'n_estimators': [200, 300, 500],
    'max_depth': [None, 10, 15, 20],
    'min_samples_split': [2, 5, 10]
}

parameters_xgb = {
    'n_estimators': [200, 300, 500],
    'max_depth': [10, 15, 20],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0]
}

# Grid search for hyperparameter tuning
grid_rf = GridSearchCV(rf_clf, parameters_rf, cv=5)
grid_xgb = GridSearchCV(xgb_clf, parameters_xgb, cv=5)

# Training the models
for clf, label in zip([grid_rf, grid_xgb], ['Random Forest', 'XGBoost']):
    start_time = time.time()
    clf.fit(X_train, y_train)
    print(f"\nTraining time for {label}: {time.time() - start_time:.2f} seconds.\n")

    # Predict on the test set
    y_pred = clf.predict(X_test)

    # Print accuracy and classification report
    accuracy = accuracy_score(y_test, y_pred)
    class_report = classification_report(y_test, y_pred)

    print(f"{label} Model Performance:")
    print(f"Best Parameters: {clf.best_params_}")
```

```
    print(f"Accuracy: {accuracy}")
    print(f"Classification Report: \n{class_report}")
    print("\n---------------------------------\n")

# Create the ensemble model
ensemble_model = VotingClassifier(estimators=[('rf', grid_rf.best_estimator_), ('xgb',
grid_xgb.best_estimator_)], voting='hard')
ensemble_model.fit(X_train, y_train)

# Make predictions
y_pred_ensemble = ensemble_model.predict(X_test)

# Print accuracy and classification report
accuracy_ensemble = accuracy_score(y_test, y_pred_ensemble)
class_report_ensemble = classification_report(y_test, y_pred_ensemble)

print(f"Ensemble Model Performance:")
print(f"Accuracy: {accuracy_ensemble}")
print(f"Classification Report: \n{class_report_ensemble}")
```

```
Random Forest Model Performance:
Best Parameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 300}
Accuracy: 0.87950189232084
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.86      0.88      4070
           1       0.87      0.90      0.88      4121

    accuracy                           0.88      8191
   macro avg       0.88      0.88      0.88      8191
weighted avg       0.88      0.88      0.88      8191

XGBoost Model Performance:
Best Parameters: {'colsample_bytree': 0.6, 'learning_rate': 0.1, 'max_depth': 15,
'n_estimators': 200, 'subsample': 1.0}
Accuracy: 0.8559394457331218
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.80      0.85      4070
           1       0.82      0.91      0.86      4121

    accuracy                           0.86      8191
```

```
   macro avg       0.86      0.86      0.86      8191
weighted avg       0.86      0.86      0.86      8191
```

```
/usr/local/lib/python3.10/dist-packages/xgboost/sklearn.py:1395: UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
Ensemble Model Performance:
Accuracy: 0.8730313759003785
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      4070
           1       0.89      0.85      0.87      4121

    accuracy                           0.87      8191
   macro avg       0.87      0.87      0.87      8191
weighted avg       0.87      0.87      0.87      8191
```