Fraud Detection
Final Project

INTRODUCTION

In today's digital era, fraud detection has become a critical component of financial security systems.

This project focuses on detecting fraudulent transactions in credit card data with machine learning techniques.

The dataset, from Kaggle *“*Credit Card Transactions Fraud Detection Dataset*”, contains simulated credit card transactions from 1,000 customers and 800 merchants, from January 1, 2019, to December 31, 2020.

Dataset

The dataset includes over 1.3 million legitimate transactions and only 7,506 fraudulent ones.

That represents a highly imbalanced classification problem, so metrics will probably not show high performances.

Fraudulent transactions are statistically rare, but they are difficult to detect.

They pose significant financial risks, making this dataset ideal for exploring anomaly detection methods.

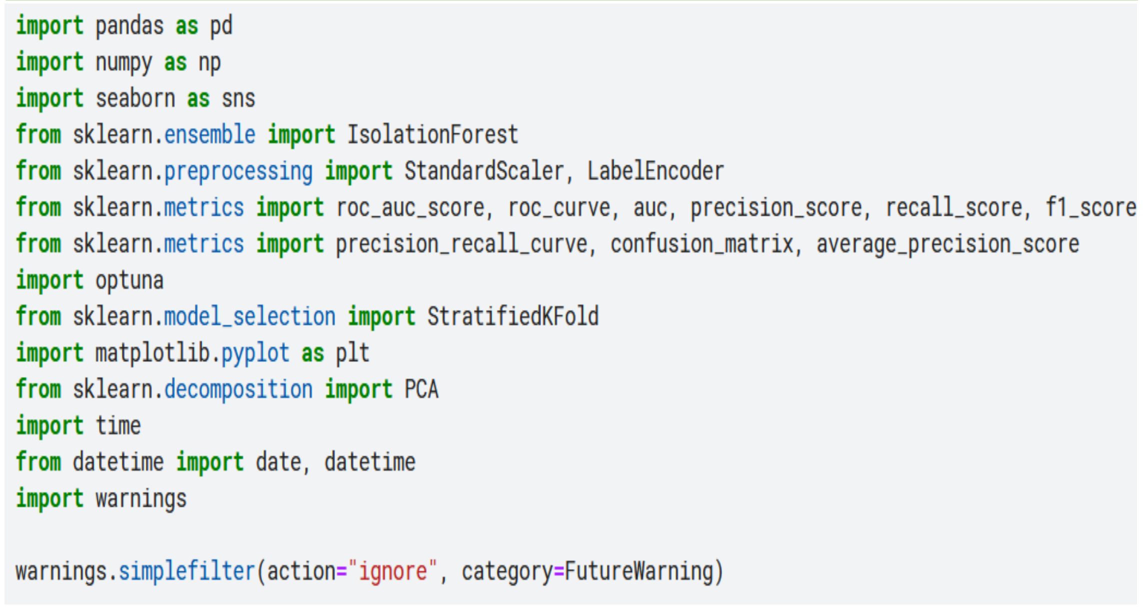
Method

I am going to utilize an unsupervised learning technique, Isolation Forest, to build a model capable of identifying these outliers.

Today, I am just getting started with fraud detection but wish to develop further this type of models in future projects.

**”https://www.kaggle.com/datasets/kartik2112/fraud-detection”*

Libraries



```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import roc_auc_score, roc_curve, auc, precision_score, recall_score, f1_score
from sklearn.metrics import precision_recall_curve, confusion_matrix, average_precision_score
import optuna
from sklearn.model_selection import StratifiedKFold
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import time
from datetime import date, datetime
import warnings

warnings.simplefilter(action="ignore", category=FutureWarning)
```

I use Optuna for hyperparameter tuning and StratifiedKfold to adapt to class huge imbalance.

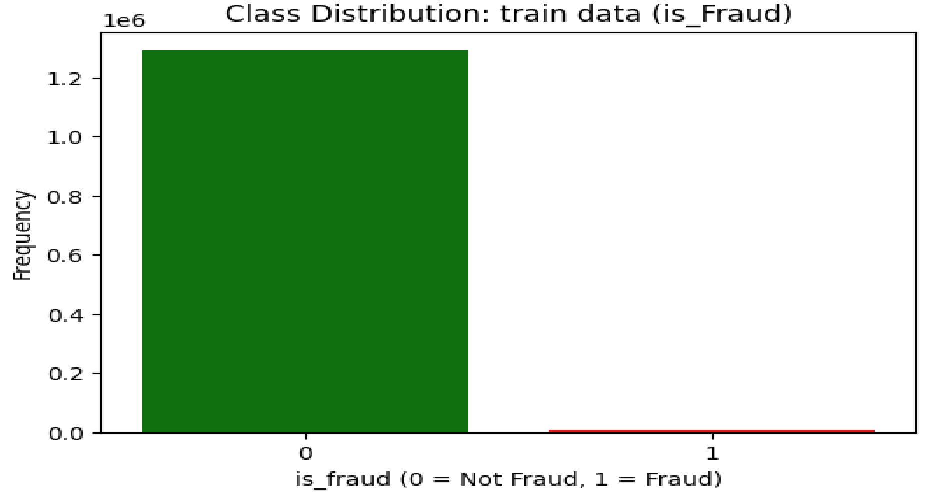
I focus on metric AUC, more suitable than accuracy to capture performance on imbalanced classes.

Exploratory Data Analysis Visualizations Part 1

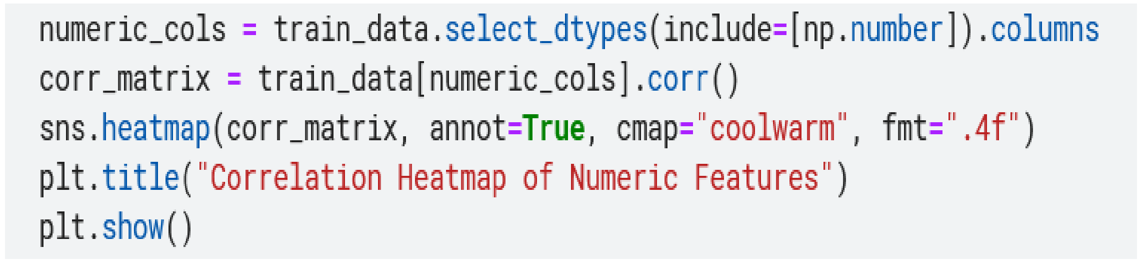
```
train0_data = pd.read_csv("/kaggle/input/fraud-detection/fraudTrain.csv")
test0_data = pd.read_csv("/kaggle/input/fraud-detection/fraudTest.csv")

print("Class distribution in train data (is_fraud):")
print(train0_data["is_fraud"].value_counts())

plt.figure(figsize=(6, 4))
sns.barplot(x=train0_data["is_fraud"].value_counts().index,
            y=train0_data["is_fraud"].value_counts().values,
            palette= ["green", "red"])
plt.title("Class Distribution: train data (is_Fraud)")
plt.xlabel("is_fraud (0 = Not Fraud, 1 = Fraud)")
plt.ylabel("Frequency")
plt.show()
```

Classes Fraud / Not Fraud

We can see below that the classes are significantly imbalanced, as expected.



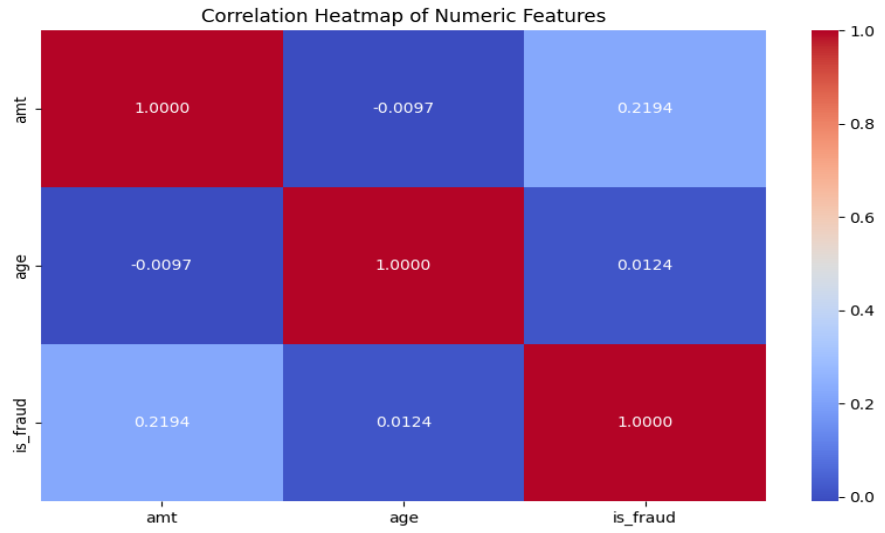
Exploratory Data Analysis Visualizations Part 2



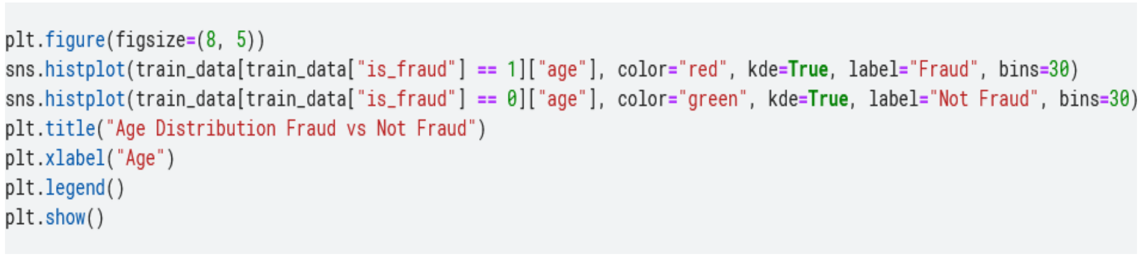
```
numeric_cols = train_data.select_dtypes(include=[np.number]).columns
corr_matrix = train_data[numeric_cols].corr()
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".4f")
plt.title("Correlation Heatmap of Numeric Features")
plt.show()
```

Correlation heatmap for selected numeric features

There is little correlation between selected numeric features.



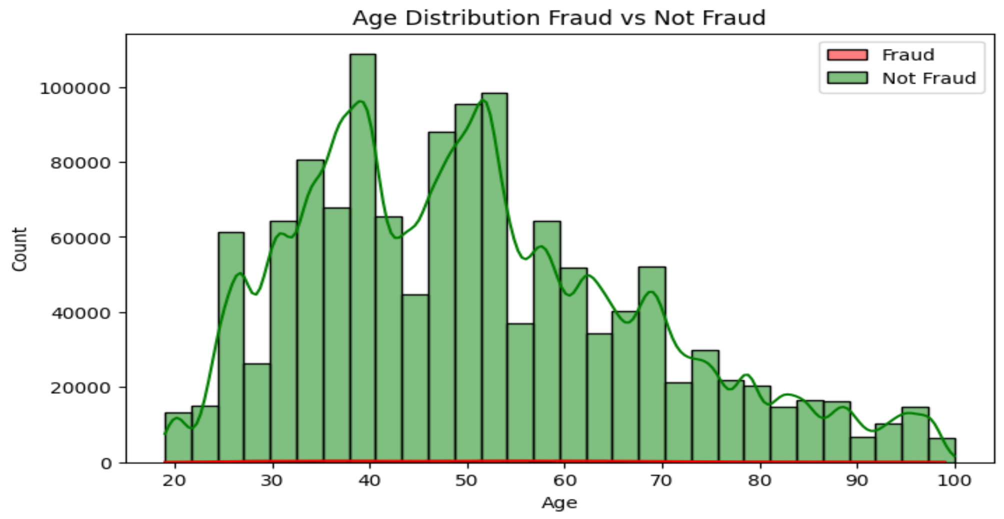
Exploratory Data Analysis Visualizations Part 3



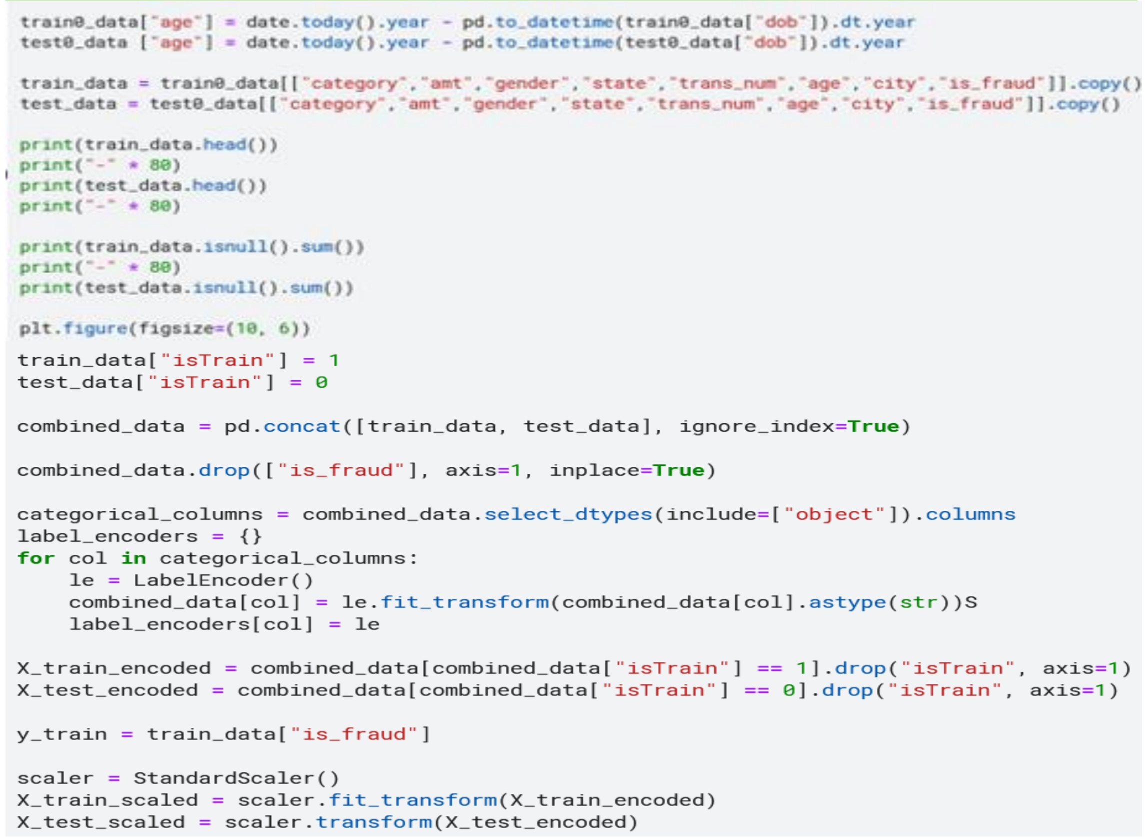
```
plt.figure(figsize=(8, 5))
sns.histplot(train_data[train_data["is_fraud"] == 1]["age"], color="red", kde=True, label="Fraud", bins=30)
sns.histplot(train_data[train_data["is_fraud"] == 0]["age"], color="green", kde=True, label="Not Fraud", bins=30)
plt.title("Age Distribution Fraud vs Not Fraud")
plt.xlabel("Age")
plt.legend()
plt.show()
```

Age vs Fraud/Not Fraud

The classes are so unbalanced that trends are basically not visible.



Exploratory Data Analysis
Data Preparation



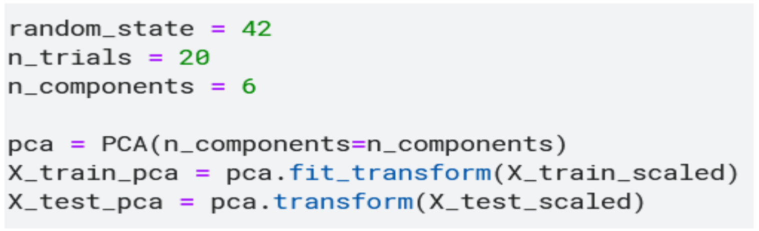
```
train0_data["age"] = date.today().year - pd.to_datetime(train0_data["dob"]).dt.year
test0_data ["age"] = date.today().year - pd.to_datetime(test0_data["dob"]).dt.year

train_data = train0_data[["category","amt","gender","state","trans_num","age","city","is_fraud"]].copy()
test_data = test0_data[["category","amt","gender","state","trans_num","age","city","is_fraud"]].copy()

print(train_data.head())
print("-" * 80)
print(test_data.head())
print("-" * 80)

print(train_data.isnull().sum())
print("-" * 80)
print(test_data.isnull().sum())

plt.figure(figsize=(10, 6))
train_data["isTrain"] = 1
test_data["isTrain"] = 0

combined_data = pd.concat([train_data, test_data], ignore_index=True)

combined_data.drop(["is_fraud"], axis=1, inplace=True)

categorical_columns = combined_data.select_dtypes(include=["object"]).columns
label_encoders = {}
for col in categorical_columns:
    le = LabelEncoder()
    combined_data[col] = le.fit_transform(combined_data[col].astype(str))S
    label_encoders[col] = le

X_train_encoded = combined_data[combined_data["isTrain"] == 1].drop("isTrain", axis=1)
X_test_encoded = combined_data[combined_data["isTrain"] == 0].drop("isTrain", axis=1)

y_train = train_data["is_fraud"]

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_encoded)
X_test_scaled = scaler.transform(X_test_encoded)
```

Data cleaning and preparation

Among the 23 columns, I selected 8 to build the model. I left out features like addresses, numbers and ids.

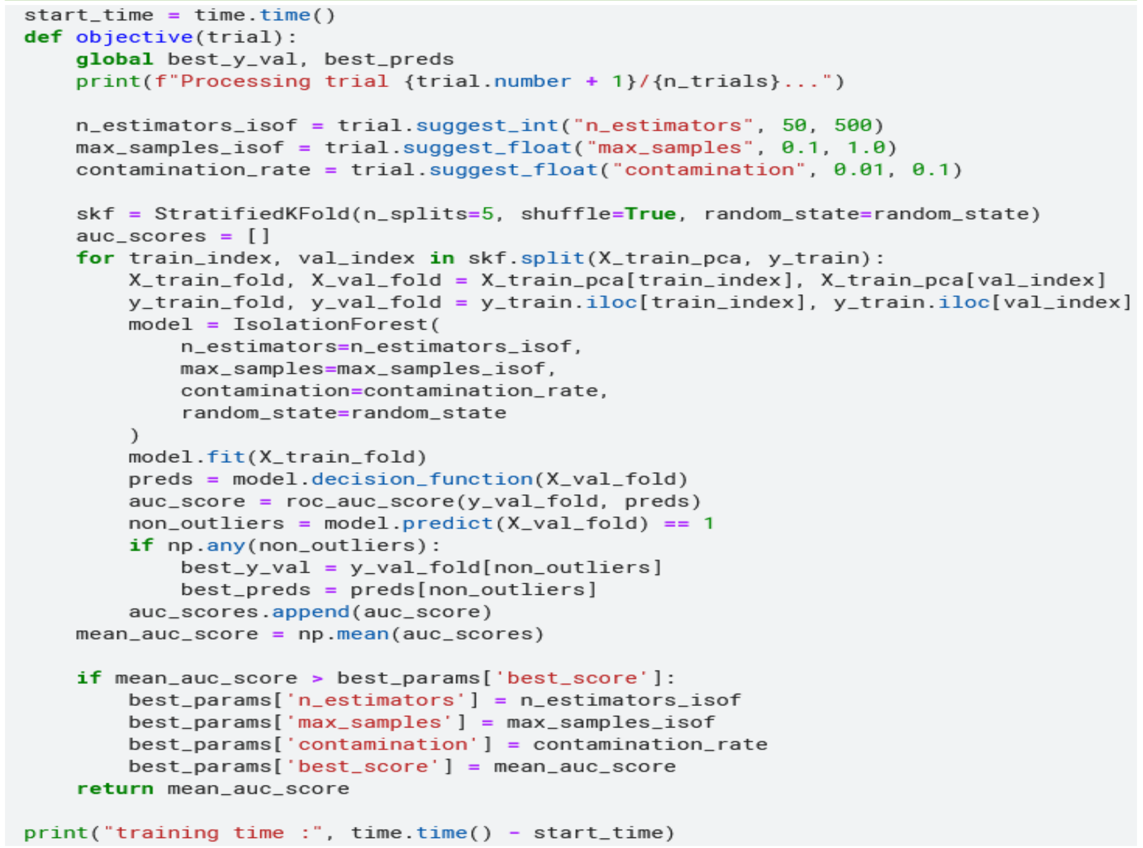
There are no missing values.

I used label encoding for categorical variables like category/gender and scaled the data.
PCA was applied to reduce dimensionality for Isolation Forest.



```
random_state = 42
n_trials = 20
n_components = 6

pca = PCA(n_components=n_components)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
```

Model Architecture
Isolation Forest



```
start_time = time.time()
def objective(trial):
    global best_y_val, best_preds
    print(f"Processing trial {trial.number + 1}/{n_trials}...")

    n_estimators_isof = trial.suggest_int("n_estimators", 50, 500)
    max_samples_isof = trial.suggest_float("max_samples", 0.1, 1.0)
    contamination_rate = trial.suggest_float("contamination", 0.01, 0.1)

    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=random_state)
    auc_scores = []
    for train_index, val_index in skf.split(X_train_pca, y_train):
        X_train_fold, X_val_fold = X_train_pca[train_index], X_train_pca[val_index]
        y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]
        model = IsolationForest(
            n_estimators=n_estimators_isof,
            max_samples=max_samples_isof,
            contamination=contamination_rate,
            random_state=random_state
        )
        model.fit(X_train_fold)
        preds = model.decision_function(X_val_fold)
        auc_score = roc_auc_score(y_val_fold, preds)
        non_outliers = model.predict(X_val_fold) == 1
        if np.any(non_outliers):
            best_y_val = y_val_fold[non_outliers]
            best_preds = preds[non_outliers]
        auc_scores.append(auc_score)
    mean_auc_score = np.mean(auc_scores)

    if mean_auc_score > best_params['best_score']:
        best_params['n_estimators'] = n_estimators_isof
        best_params['max_samples'] = max_samples_isof
        best_params['contamination'] = contamination_rate
        best_params['best_score'] = mean_auc_score
    return mean_auc_score

print("training time :", time.time() - start_time)
```

Why Isolation Forest for fraud detection?

Isolation Forest is well-suited for detecting anomalies like fraud cases, especially in highly imbalanced datasets.

It isolates observations by randomly selecting a feature and splitting it, making anomalies quicker to isolate than normal data points.

It performs well for fraud detection due to its ability to model outliers and its scalability to large datasets.

Fraud detection datasets typically have a very low ratio of fraud to non-fraud cases, making classification challenging. Isolation Forest models the "normal" behavior and flags anything different as potentially fraudulent.

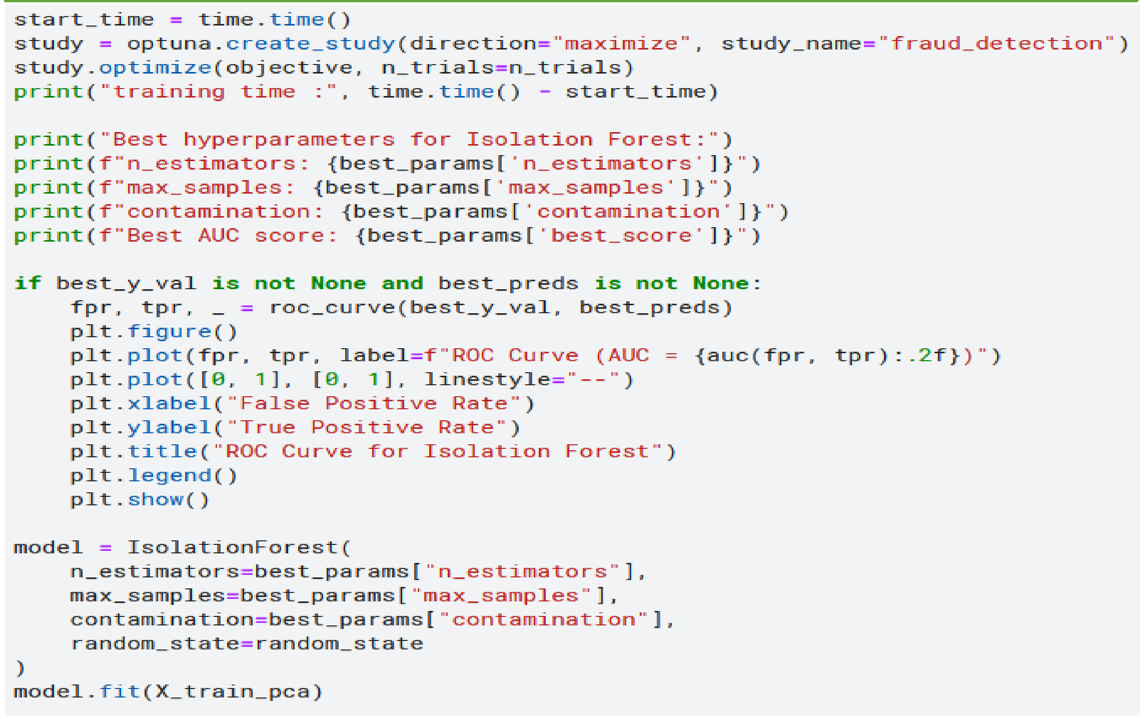
In this project, I tuned the critical hyperparameters below :

n_estimators: the number of trees in the ensemble.

max_samples: the fraction of the dataset used to train each tree.

contamination: the proportion of outliers assumed to be in the data.

Model Training and Optimization Optuna tuning



```python
start_time = time.time()
study = optuna.create_study(direction="maximize", study_name="fraud_detection")
study.optimize(objective, n_trials=n_trials)
print("training time :", time.time() - start_time)

print("Best hyperparameters for Isolation Forest:")
print(f"n_estimators: {best_params['n_estimators']}")
print(f"max_samples: {best_params['max_samples']}")
print(f"contamination: {best_params['contamination']}")
print(f"Best AUC score: {best_params['best_score']}")

if best_y_val is not None and best_preds is not None:
    fpr, tpr, _ = roc_curve(best_y_val, best_preds)
    plt.figure()
    plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {auc(fpr, tpr):.2f})")
    plt.plot([0, 1], [0, 1], linestyle="--")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve for Isolation Forest")
    plt.legend()
    plt.show()

model = IsolationForest(
    n_estimators=best_params["n_estimators"],
    max_samples=best_params["max_samples"],
    contamination=best_params["contamination"],
    random_state=random_state
)
model.fit(X_train_pca)
```

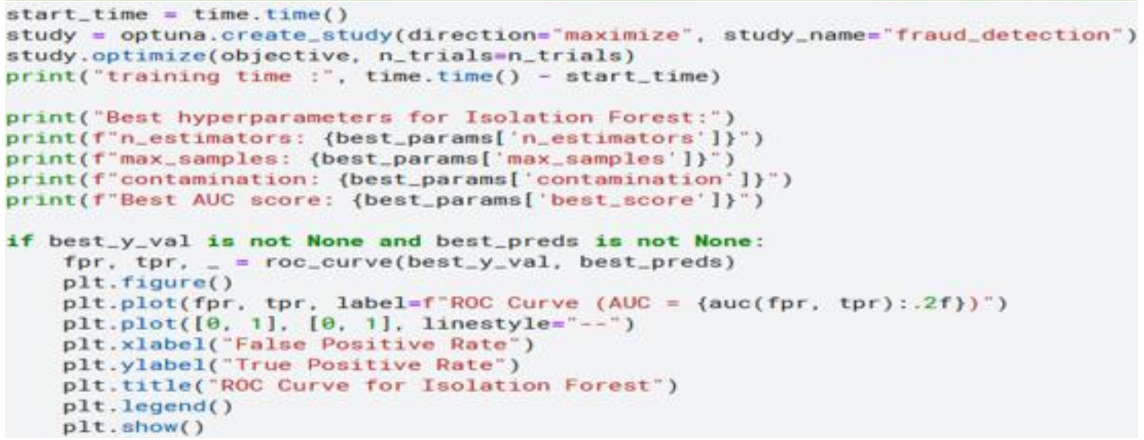
Training process

With Optuna, multiple trials were conducted to identify the best combination of parameters for the model.

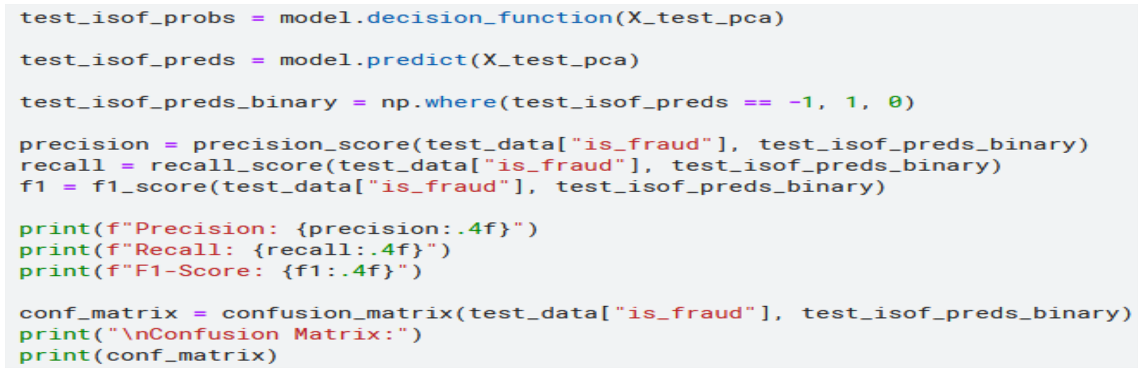
AUC was used as the optimization metric during training.

It is more suitable to evaluate performance because of its ability to distinguish fraud cases from legitimate ones.

Model Performance Results

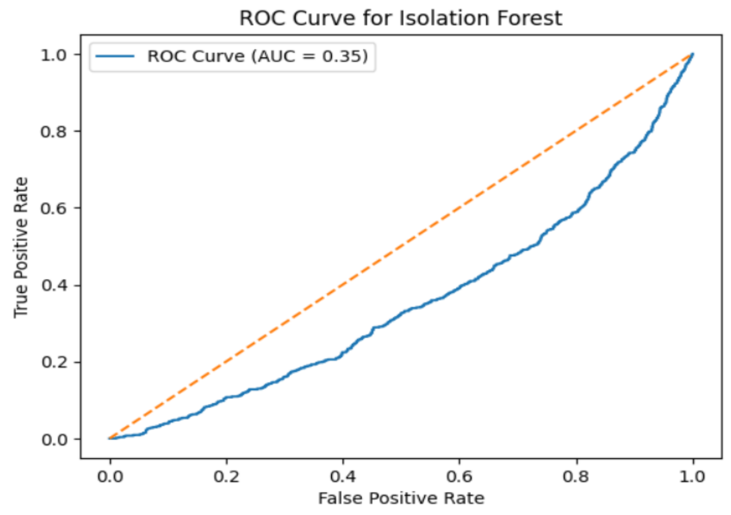


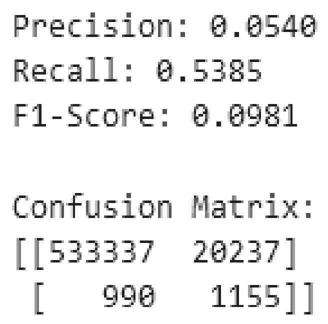
```
start_time = time.time()
study = optuna.create_study(direction="maximize", study_name="fraud_detection")
study.optimize(objective, n_trials=n_trials)
print("training time :", time.time() - start_time)

print("Best hyperparameters for Isolation Forest:")
print(f"n_estimators: {best_params['n_estimators']}")
print(f"max_samples: {best_params['max_samples']}")
print(f"contamination: {best_params['contamination']}")
print(f"Best AUC score: {best_params['best_score']}")

if best_y_val is not None and best_preds is not None:
    fpr, tpr, _ = roc_curve(best_y_val, best_preds)
    plt.figure()
    plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {auc(fpr, tpr):.2f})")
    plt.plot([0, 1], [0, 1], linestyle="--")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve for Isolation Forest")
    plt.legend()
    plt.show()
```



```
test_isof_probs = model.decision_function(X_test_pca)

test_isof_preds = model.predict(X_test_pca)

test_isof_preds_binary = np.where(test_isof_preds == -1, 1, 0)

precision = precision_score(test_data["is_fraud"], test_isof_preds_binary)
recall = recall_score(test_data["is_fraud"], test_isof_preds_binary)
f1 = f1_score(test_data["is_fraud"], test_isof_preds_binary)

print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")

conf_matrix = confusion_matrix(test_data["is_fraud"], test_isof_preds_binary)
print("\nConfusion Matrix:")
print(conf_matrix)
```

AUC





```
Precision: 0.0540
Recall: 0.5385
F1-Score: 0.0981

Confusion Matrix:
[[533337  20237]
 [   990   1155]]
```

After 20 trials and about 8 hours with accelerator, the AUC level is very low.

The other metrics also showed a low performance.

CONCLUSION

In conclusion, the model after 20 trials, yielded a low performance with an AUC of 0.35 with best hyperparameters n_estimators=**183**, max_sample=**0.216** and contamination=**0.0391**.

While recall was reasonable at 53.85%, precision was quite low at 5.40%, indicating that many false positives were generated.

The overall F1-score of 0.0981 shows room for improvement, particularly in balancing precision and recall.

The next step could involve experimenting with ensemble methods, such as combining Isolation Forest with other anomaly detection techniques like One-Class SVM.

But the challenge will be computation time because of the large datasets we have to process when we build fraud detection models.