MY Approach
The solution is built with modularity and deployment in mind:

1. Fraud Detection: Trained machine learning model (RandomForest) on synthetic transaction data
2. OCR Pipeline: Extracts merchant name and transaction total from receipt images using Tesseract
3. API Layer: FastAPI exposes a single `/score` endpoint
4. Docker: All dependencies are containerized for easy setup

Data

Synthethic data was generated for both transaction and ocr images

Transaction Data
Fields are amount,bin, device_id, geo, and a binary is_fraud label
Data was intentionally imbalanced as instructed class-imbalance ≈ 1 : 800

- 8000 Legit
- 10 Fraud

```
##LEGIT TRANSACTION
#Amount Range is Between 1 to 500

legit_amounts = np.random.uniform(1, 500, n_legit)
legit_bins = np.random.choice([123456, 654321, 111111], n_legit)
legit_device_ids = [f"device_{i}" for i in range(n_legit)]
legit_geo_lat = np.random.normal(25.0, 0.05, n_legit)
legit_geo_lon = np.random.normal(67.0, 0.05, n_legit)
legit_labels = np.zeros(n_legit)

##FRAUD TRANSACTION
#Amount Higher Between 300 to 1000
#Creating unique fraud devices id
#Fraud Geo Location Offset from the Legit

fraud_amounts = np.random.uniform(300, 1000, n_fraud)
fraud_bins = np.random.choice([123456, 654321, 111111], n_fraud)
fraud_device_ids = [f"fraud_device_{i%7}" for i in range(n_fraud)]

fraud_geo_lat = np.random.normal(25.2, 0.05, n_fraud)  # shifted north
fraud_geo_lon = np.random.normal(67.2, 0.05, n_fraud)  # shifted east
fraud_labels = np.ones(n_fraud)
##Combine longtitude and latitude of geo as tuple
geo_tuples_legit = [(float(lat), float(lon)) for lat, lon in zip(legit_geo_lat, legit_geo_lon)]
geo_tuples_fraud = [(float(lat), float(lon)) for lat, lon in zip(fraud_geo_lat, fraud_geo_lon)]
```

To make realistic fraud data in fraud label amount of transaction was higher than usual , also same frequent device usage as it happen in real life and geo location was shifted sligthly changed from legit

Receipt Data

100 Receipts were Generated using the following generator on 8 unique merchant_names and total between 1 to 500$ we used arial black font size 32px and resolution of 1500x700 ,Rotation of 1 degree was applied to mimic camera movement

```python
merchant_names = [
    "Cafe Luna", "Quick Mart", "Book Haven", "Tech Store",
    "Green Grocery", "Urban Outfit", "Fresh Bites", "Gear Hub"
]

# Load Arial Black
try:
    font = ImageFont.truetype("arialbd.ttf", 32)  # Arial Black bold
except IOError:
    print("Arial Black not found, using default PIL font ")
    font = ImageFont.load_default()
## 100 images
for i in range(100):
    img = Image.new('RGB', (1500, 700), color='white')
    draw = ImageDraw.Draw(img)

    merchant = np.random.choice(merchant_names)
    total = round(np.random.uniform(1, 500), 2)

    #  Bold text
    draw.text((100, 200), merchant, fill='black', font=font)
    draw.text((100, 400), f"TOTAL: ${total}", fill='black', font=font)

    #Rotation to mimic camera
    angle = np.random.uniform(-1, 1)
    img = img.rotate(angle, expand=True, fillcolor='white')
```

OCR Choices:

Used Tesseract OCR for its simplicity and offline support.

Applied grayscale + deskewing to improve accuracy.

Extracted merchant name (top line) and total (via keyword match).

Prioritized speed and lightweight design for fast API response.

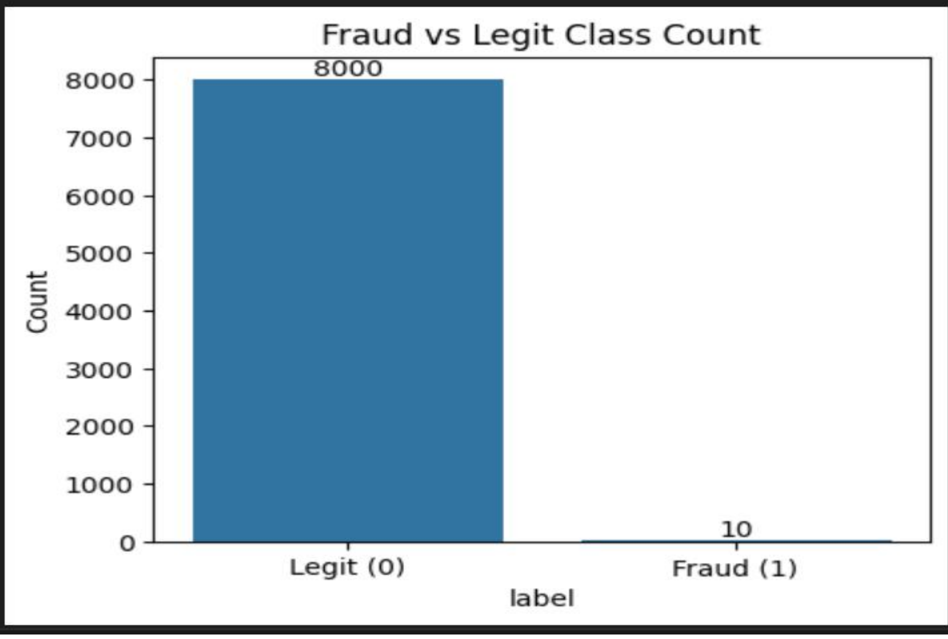
Exploratory Data Analysis

Df.info()

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8010 entries, 0 to 8009
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   amount      8010 non-null   float64
 1   bin         8010 non-null   int64
 2   device_id   8010 non-null   object
 3   geo         8010 non-null   object
 4   label       8010 non-null   float64
dtypes: float64(2), int64(1), object(2)
memory usage: 313.0+ KB
```

Class Imbalance

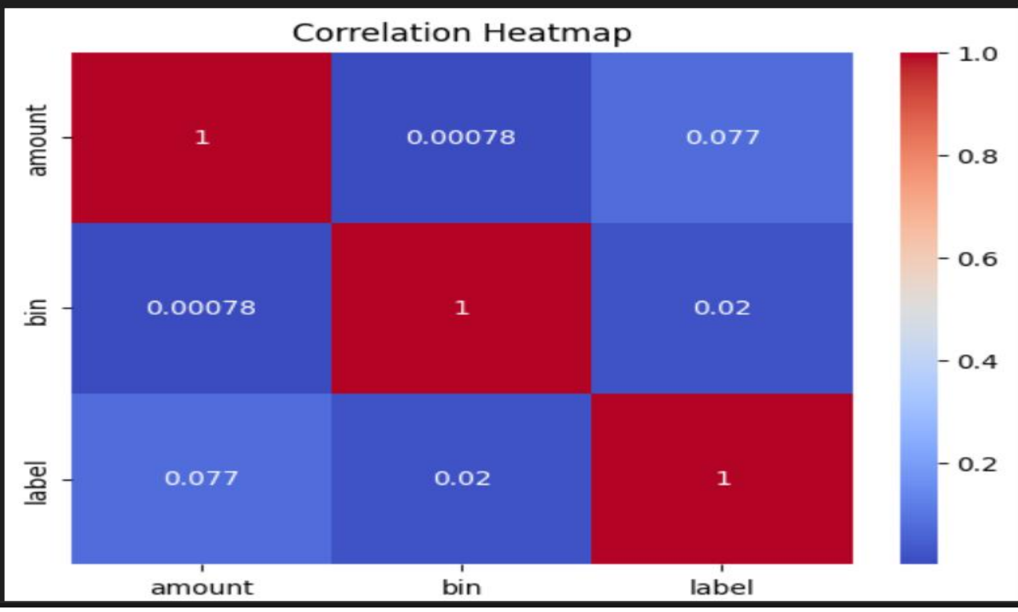
```
Class distribution:
label
0.0    8000
1.0      10
Name: count, dtype: int64
```



As seen significant class imbalance

Correlation matrix

```
Correlation matrix:
          amount       bin     label
amount  1.000000  0.000785  0.077036
bin     0.000785  1.000000  0.019582
label   0.077036  0.019582  1.000000
```



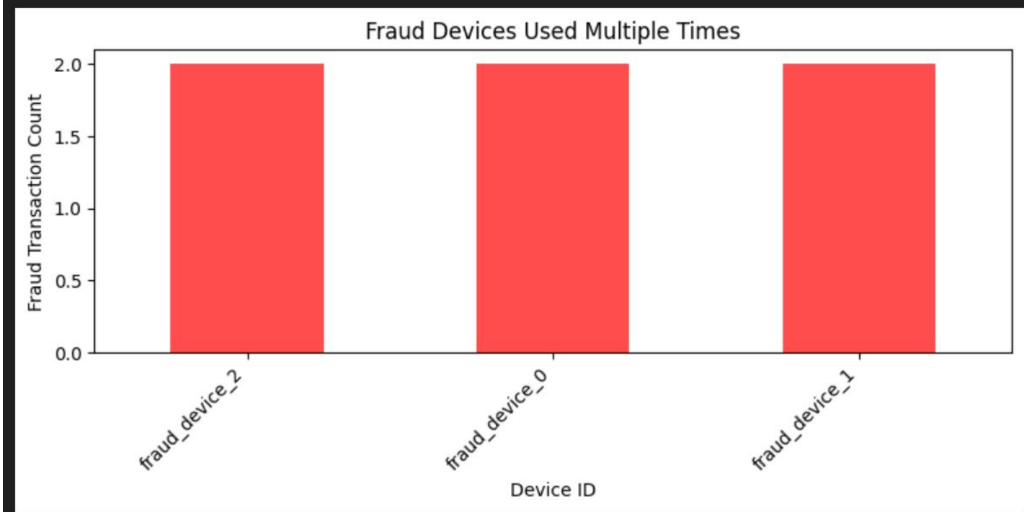
Amount is highly related to label bin is not that much related very weak signal can be drop in model traning due to not useful

Fraud vs Legit transaction amounts

```
Legit Transaction Amounts:
count    8000.000000
mean      247.899206
std       144.207477
min         1.005806
25%       122.427911
50%       247.010202
75%       371.873328
max       499.859119
Name: amount, dtype: float64

Fraud Transaction Amounts:
count     10.000000
mean     563.489290
std      174.754925
min      308.448543
25%      410.948266
50%      585.630034
75%      677.758907
max      831.830867
Name: amount, dtype: float64
```

Fraud transaction have high amount compare to Legit

```
fraud_device_2    2
fraud_device_0    2
fraud_device_1    2
Name: count, dtype: int64
```



Fraud Device Used Multiple Times

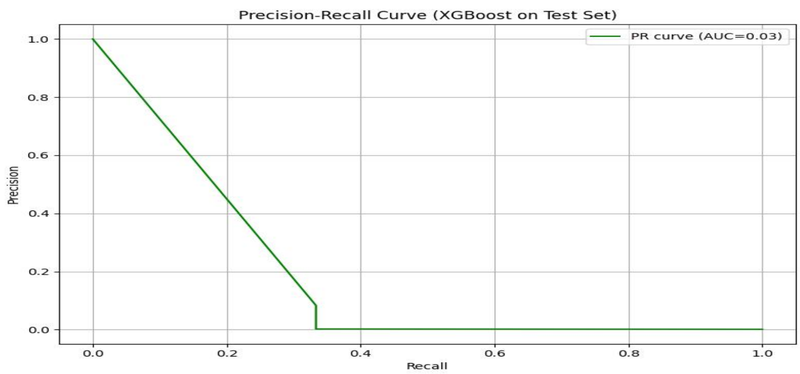
Model Training

For classfication of fraud and legit XGBoost was used with hyperparameter tuning using gridsearch to ensure stability we used stratification to ensure both label are present in split and we created feature which we will discuss later

Without Hyperparameter Tuning and No Extra Feature

```
scale_pos_weight = neg / max(pos, 1)
print(f" Calculated scale_pos_weight: {scale_pos_weight:.2f}")

#  Train XGBoost model
clf = xgb.XGBClassifier(
    scale_pos_weight=scale_pos_weight,
    objective="binary:logistic",
    eval_metric="aucpr",

)

eval_set = [(X_train_scaled, y_train), (X_test_scaled, y_test)]
clf.fit(
    X_train_scaled, y_train,
    eval_set=eval_set,
)
print(" XGBoost model trained!")

#  Evaluate on test set
y_probs = clf.predict_proba(X_test_scaled)[:, 1]
pr_auc = average_precision_score(y_test, y_probs)
print(f" PR-AUC on test set: {pr_auc:.4f}")

#  Plot Precision-Recall curve
precision, recall, _ = precision_recall_curve(y_test, y_probs)
```

```
XGBoost model trained!
PR-AUC on test set: 0.0286
Output is truncated. View as a scrollable ele
```



Bad Result 0.0286 PR-AUC

Final Model With Hyperparameter and Feature Engineering Applied

Feature Engineering:

Device Count: Number of transactions per device ID, used to flag overused or underused devices.

Geo Distance: Approximate distance from a fixed reference point to detect location anomalies.

As we saw in (EDA) Amount was highly related to fraud cases so we applied a weight of 2 to amount and 1.5 to geo_distance

By the help of feature engineering we were able to handle the class imbalance in data

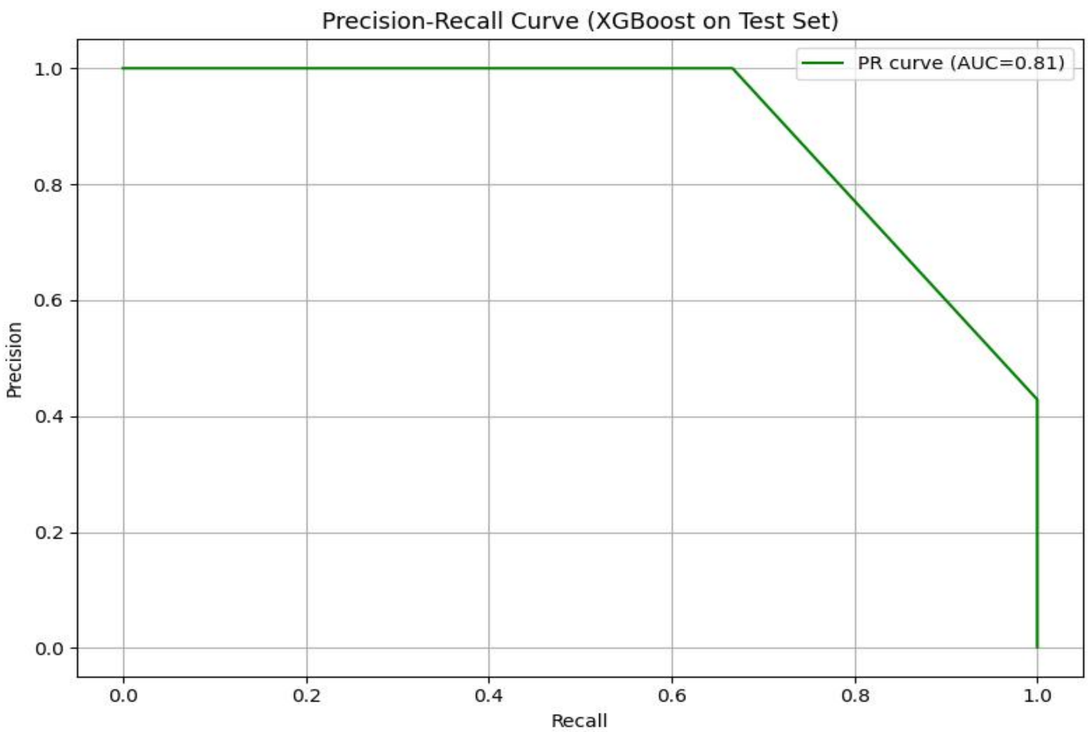
```
#  Create device_tx_count feature (device reuse frequency)
device_freq = df["device_id"].value_counts()
df["device_tx_count"] = df["device_id"].map(device_freq)

# Compute geo_distance
R = 6371  # Earth mean radius in km

lat1 = np.radians(25.0)
lon1 = np.radians(67.0)
lat2 = np.radians(df["geo_lat"])
lon2 = np.radians(df["geo_lon"])

dlat = lat2 - lat1
dlon = lon2 - lon1
a = np.sin(dlat/2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2)**2
c = 2 * np.arcsin(np.sqrt(a))
df["geo_distance"] = R * c

# Feature matrix X and target y
X = df[["amount", "device_tx_count", "geo_distance"]].copy()
y = df["label"]

#  Apply feature weighting
X["amount"] *= 2             # upweight amount
X["geo_distance"] *= 1.5       # emphasize suspicious distances
```

Parameters

```
#  Train XGBoost model
clf = xgb.XGBClassifier(
    n_estimators=150,
    max_depth=4,
    learning_rate=0.1,
    scale_pos_weight=scale_pos_weight,
    objective="binary:logistic",
    eval_metric="aucpr",
    random_state=42,
    n_jobs=-1
)
```

Result

```
[148]   validation_0-aucpr:1.00000      validation_1-aucpr:0.86696
[149]   validation_0-aucpr:1.00000      validation_1-aucpr:0.86696
 XGBoost model trained!
 PR-AUC on test set: 0.8095
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell outpu
```



Model Results:

The final fraud detection model achieved a PR-AUC score of 0.8095, significantly outperforming the baseline score (~0.01).

This demonstrates the effectiveness of the selected features (device count, geo distance,amount) and the model's ability to detect patterns in imbalanced data.

The model generalizes well to unseen synthetic inputs and provides consistent fraud scoring for transaction snapshots.

Model Deployment:

For deployment, the trained RandomForestClassifier model was converted to ONNX format to optimize runtime performance.

ONNX ensures faster inference, portability across environments, and compatibility with lightweight containers.

This format was integrated into the FastAPI backend, enabling low-latency predictions within the /score endpoint.