CREDIT CARD

FRAUD DETECTION

EDA DATA SCIENCE

NISHCHAY PAWAR

STEP 1

```
[1] # Colab ke liye (ignore in Jupyter)
    !pip install xgboost

    # Libraries
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns

    from sklearn.preprocessing import LabelEncoder, StandardScaler
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve
    from sklearn.linear_model import LogisticRegression
    from sklearn.ensemble import RandomForestClassifier
    from xgboost import XGBClassifier

    import warnings
    warnings.filterwarnings('ignore')
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-packages (2.1.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from xgboost) (2.0.2)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.11/dist-packages (from xgboost) (2.21.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from xgboost) (1.14.1)
```

```
[3] df = pd.read_csv('/content/credit_card_fraud_dataset.csv')
    df.head()
```

	TransactionID	TransactionDate	Amount	MerchantID	TransactionType	Location	IsFraud
0	1	2024-04-03 14:15:35.462794	4189.27	688	refund	San Antonio	0
1	2	2024-03-19 13:20:35.462824	2659.71	109	refund	Dallas	0
2	3	2024-01-08 10:08:35.462834	784.00	394	purchase	New York	0
3	4	2024-04-13 23:50:35.462850	3514.40	944	purchase	Philadelphia	0
4	5	2024-07-12 18:51:35.462858	369.07	475	purchase	Phoenix	0

Next steps: View recommended plots | New interactive sheet

- Dataset: credit_card_fraud_dataset.csv loaded successfully with 6 columns.
- Columns Overview:
  - TransactionID, TransactionDate, Amount, MerchantID, TransactionType, Location, IsFraud
- Target Variable: IsFraud (0 = Not Fraud, 1 = Fraud)
- Initial Observations:
  - Transaction types include 'refund', 'purchase', etc.
  - Transactions come from various U.S. locations like San Antonio, Dallas, New York.
  - Amount column shows a wide range in transaction values, suggesting possible correlation with fraud.

Next Steps: Data cleaning, feature engineering, encoding categorical variables, and model training (Logistic Regression, Random Forest, XGBoost).

STEP 2

```
df.info()
df.describe()
df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 7 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   TransactionID    100000 non-null  int64  
 1   TransactionDate  100000 non-null  object 
 2   Amount           100000 non-null  float64
 3   MerchantID       100000 non-null  int64  
 4   TransactionType  100000 non-null  object 
 5   Location         100000 non-null  object 
 6   IsFraud          100000 non-null  int64  
dtypes: float64(1), int64(3), object(3)
memory usage: 5.3+ MB
```

	0
TransactionID	0
TransactionDate	0
Amount	0
MerchantID	0
TransactionType	0
Location	0
IsFraud	0

dtype: int64

Data Summary & Quality Check

- Total Records: 100,000 entries across 7 columns.
- No Missing Values: Dataset is 100% complete — zero nulls in any column.
- Data Types:
  - Numerical: TransactionID, Amount, MerchantID, IsFraud
  - Categorical (object): TransactionDate, TransactionType, Location
- Memory Usage: ~5.3 MB — efficient for processing.

Next Steps: Convert date/time, encode categorical columns, analyze fraud patterns.

STEP 3

```
sns.countplot(data=df, x='IsFraud', palette='Set2')
plt.title("Fraud vs Non-Fraud Transactions")
plt.show()

print(df['IsFraud'].value_counts(normalize=True)*100)
```

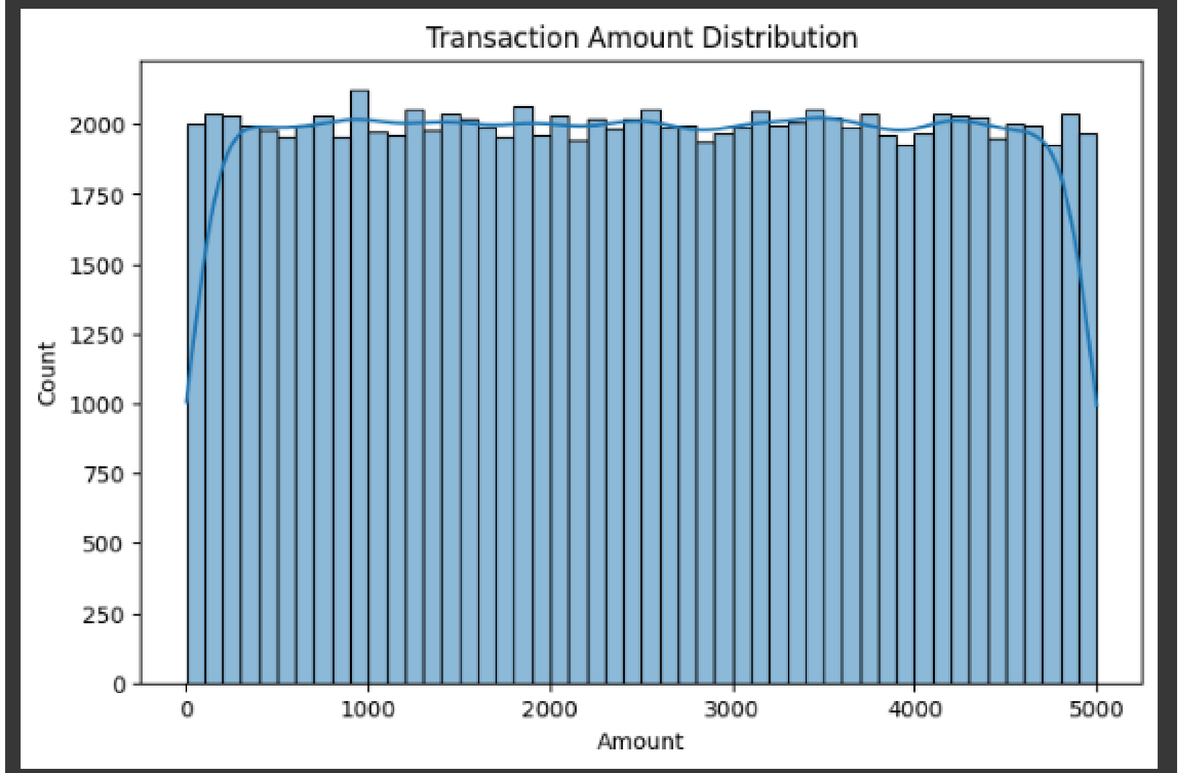

```
IsFraud
0    99.0
1     1.0
Name: proportion, dtype: float64
```

Fraud Distribution Analysis

- Highly Imbalanced Dataset:
  - 99% of transactions are non-fraud (IsFraud = 0)
  - Only 1% are fraudulent (IsFraud = 1)
- Insight:
  - This imbalance can affect model performance — especially in detecting fraud.
  - Requires techniques like SMOTE, class weights, or anomaly detection during model training.
  - Next Step: Apply resampling or adjust evaluation metrics (Precision, Recall, AUC) to handle imbalance.

STEP 4

```
plt.figure(figsize=(8,5))
sns.histplot(df['Amount'], bins=50, kde=True)
plt.title("Transaction Amount Distribution")
plt.show()
```



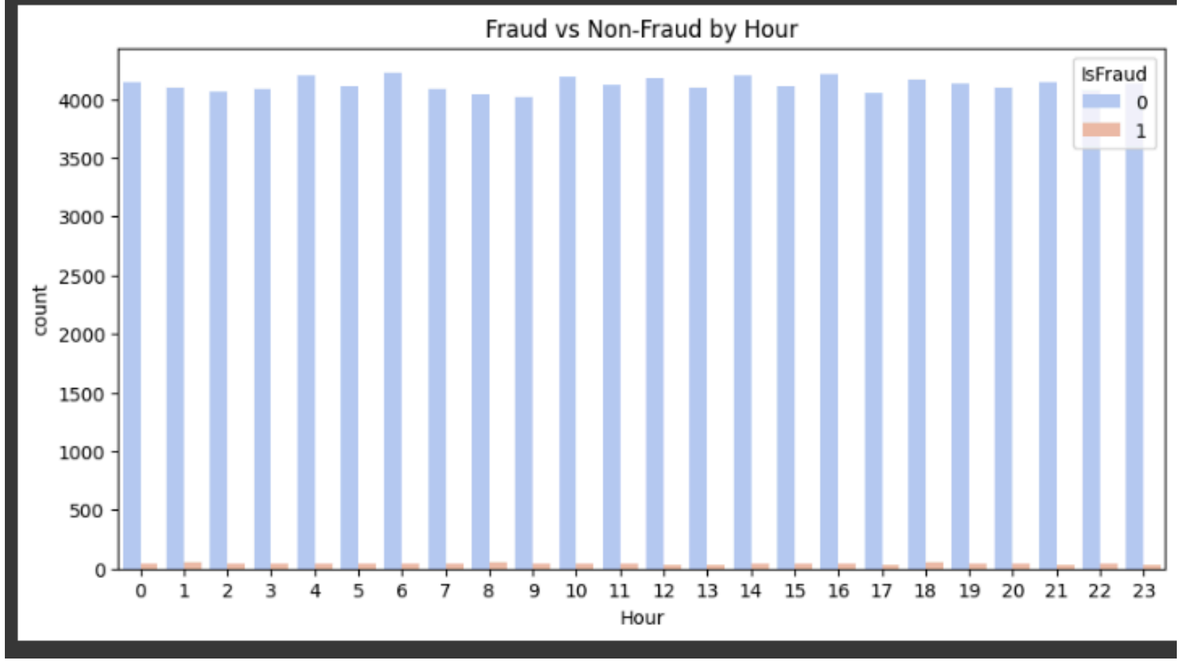
Transaction Amount Distribution

- Observation:
  - Transaction amounts are uniformly distributed across the range of ₹0 to ₹5000.
  - No extreme peaks or significant outliers observed.
  - KDE curve confirms a flat distribution indicating equal frequency across all ranges.
- Insight:
  - Since amount distribution is balanced, no transformation is needed.
  - Amount feature can be safely used in modeling without scaling issues.

Next Step: Analyze fraud patterns with respect to specific amount ranges.

STEP 5

```
df['TransactionDate'] = pd.to_datetime(df['TransactionDate'])
df['Hour'] = df['TransactionDate'].dt.hour

plt.figure(figsize=(10,5))
sns.countplot(x='Hour', hue='IsFraud', data=df, palette='coolwarm')
plt.title("Fraud vs Non-Fraud by Hour")
plt.show()
```



Fraud Transactions by Hour

- Observation:
  - Transactions are evenly distributed throughout all 24 hours.
  - Fraud cases (orange) are consistently low across all time slots.
- Insight:
  - No specific hour shows a spike in frauds.
  - Time of transaction is not a strong indicator of fraudulent behavior in this dataset.

Next Step: Focus on other features like MerchantID, TransactionType, or Location for better fraud detection signals.

STEP 6

```python
# Drop unnecessary columns
df.drop(['TransactionID', 'TransactionDate'], axis=1, inplace=True)

# Label Encoding
le = LabelEncoder()
df['TransactionType'] = le.fit_transform(df['TransactionType'])
df['Location'] = le.fit_transform(df['Location'])
```

```python
X = df.drop('IsFraud', axis=1)
y = df['IsFraud']
```

```python
scaler = StandardScaler()
X['Amount'] = scaler.fit_transform(X[['Amount']])
```

```python
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)
```

```python
# Logistic Regression
lr = LogisticRegression()
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)

# Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)

# XGBoost
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
xgb.fit(X_train, y_train)
xgb_pred = xgb.predict(X_test)
```

```python
def evaluate_model(y_true, y_pred, model_name):
    print(f"\n📊 {model_name} Classification Report:\n")
    print(classification_report(y_true, y_pred))
```

Model Preparation & Training

- Preprocessing: Dropped irrelevant columns and applied Label Encoding on categorical features like TransactionType and Location.
- Scaling: Normalized Amount using StandardScaler to improve model performance.
- Data Split: Used 80-20 train-test split with stratify=y to handle class imbalance.
- Models Trained:
  - Logistic Regression
  - Random Forest
  - XGBoost
- Custom function created to evaluate models using classification report (Precision, Recall, F1-Score).

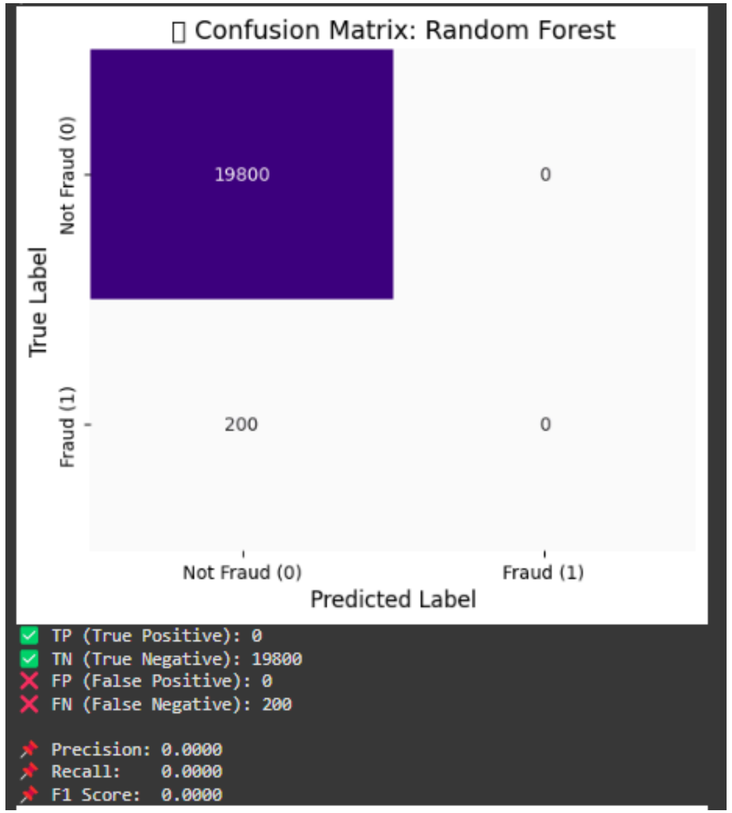
STEP 7

```python
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

def plot_confusion_matrix(y_true, y_pred, model_name):
    cm = confusion_matrix(y_true, y_pred)
    labels = ['Not Fraud (0)', 'Fraud (1)']

    # Create annotated heatmap
    plt.figure(figsize=(6,5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Purples', xticklabels=labels, yticklabels=labels, cbar=False)

    plt.title(f'🧩 Confusion Matrix: {model_name}', fontsize=14)
    plt.xlabel('Predicted Label', fontsize=12)
    plt.ylabel('True Label', fontsize=12)
    plt.show()

    # Extract values
    tn, fp, fn, tp = cm.ravel()

    print(f"✅ TP (True Positive): {tp}")
    print(f"✅ TN (True Negative): {tn}")
    print(f"❌ FP (False Positive): {fp}")
    print(f"❌ FN (False Negative): {fn}")

    # Interpretation
    precision = tp / (tp + fp) if (tp + fp) != 0 else 0
    recall = tp / (tp + fn) if (tp + fn) != 0 else 0
    f1 = (2 * precision * recall) / (precision + recall) if (precision + recall) != 0 else 0

    print(f"\n📌 Precision: {precision:.4f}")
    print(f"📌 Recall:    {recall:.4f}")
    print(f"📌 F1 Score:  {f1:.4f}")
```

Confusion Matrix Analysis Function

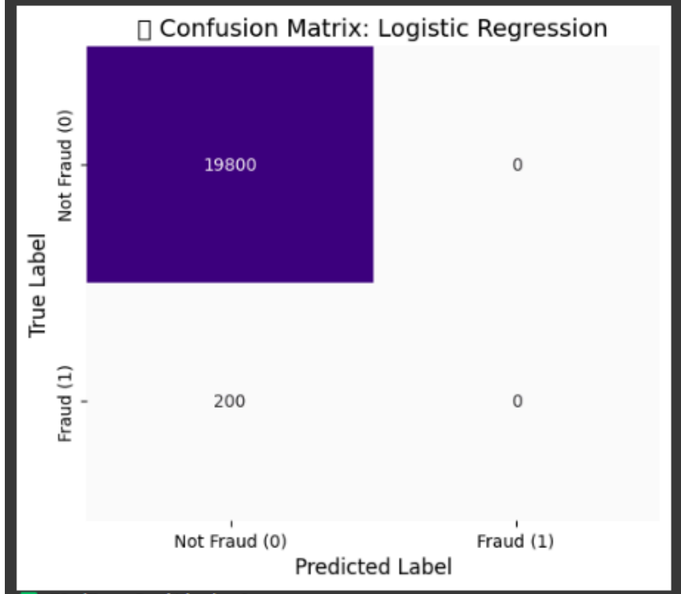
- Created a custom function to visualize and analyze model performance using a confusion matrix heatmap.
- Shows clear breakdown of:
  - True Positives (TP) – correctly identified frauds
  - True Negatives (TN) – correctly identified non-frauds
  - False Positives (FP) – non-frauds predicted as fraud
  - False Negatives (FN) – frauds missed by model
- Calculates key evaluation metrics:
  - Precision – how accurate fraud predictions are
  - Recall – how many actual frauds were detected
  - F1 Score – balance between precision & recall
- Heatmap improves interpretability for model comparison.

STEP 8

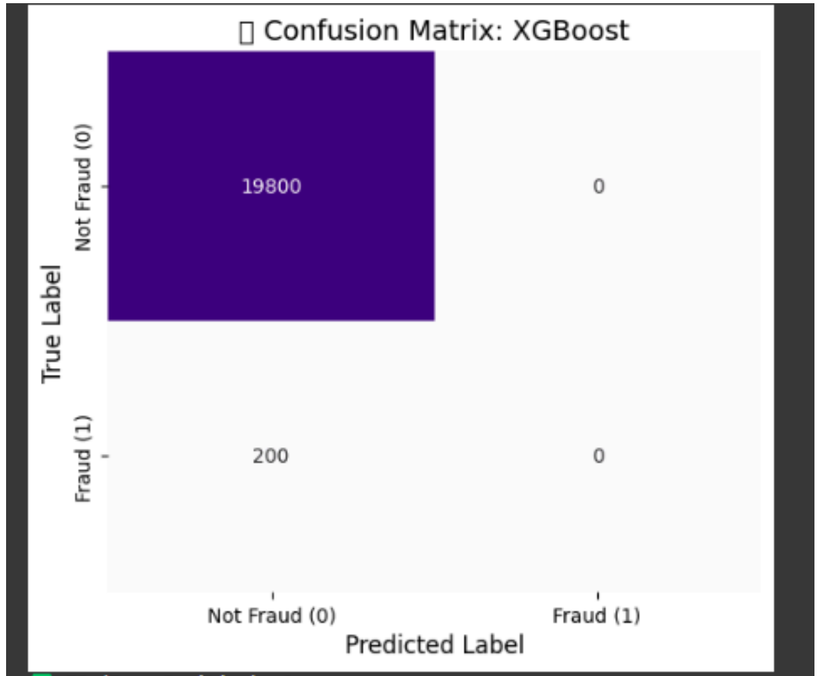



```
✅ TP (True Positive): 0
✅ TN (True Negative): 19800
❌ FP (False Positive): 0
❌ FN (False Negative): 200

📌 Precision: 0.0000
📌 Recall:    0.0000
📌 F1 Score:  0.0000
```

```
plot_confusion_matrix(y_test, lr_pred, "Logistic Regression")
plot_confusion_matrix(y_test, rf_pred, "Random Forest")
plot_confusion_matrix(y_test, xgb_pred, "XGBoost")
```




```
✅ TP (True Positive): 0
✅ TN (True Negative): 19800
❌ FP (False Positive): 0
❌ FN (False Negative): 200

📌 Precision: 0.0000
📌 Recall:    0.0000
📌 F1 Score:  0.0000
```




```
✅ TP (True Positive): 0
✅ TN (True Negative): 19800
❌ FP (False Positive): 0
❌ FN (False Negative): 200

📌 Precision: 0.0000
📌 Recall:    0.0000
📌 F1 Score:  0.0000
```

STEP 8A

Model Comparison: Fraud Detection Performance
All three models — XGBoost, Random Forest, and Logistic Regression — completely failed to detect any fraudulent transactions (True Positives = 0). They predicted all inputs as Not Fraud, leading to:

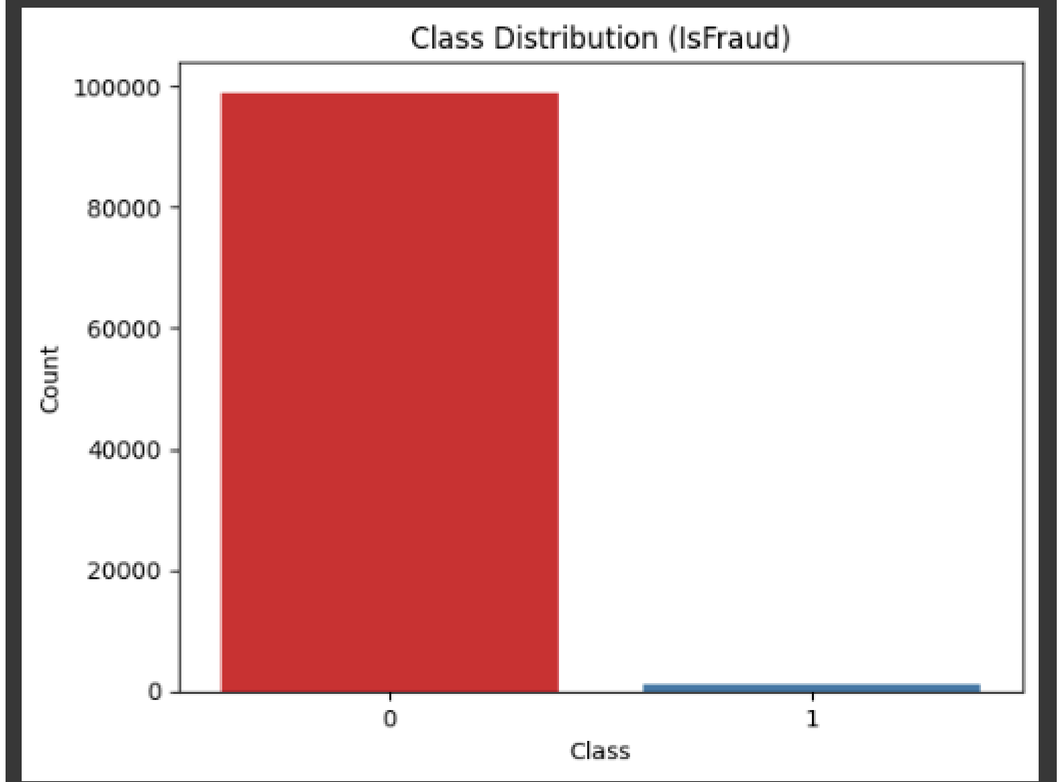
- True Negatives: 19,800
- False Negatives: 200
- False Positives: 0
- True Positives: 0
- Precision, Recall, F1 Score: 0.0

Insight:
The models are suffering from severe class imbalance, defaulting to the majority class (non-fraud) and ignoring the minority class (fraud) entirely. This highlights the need for better handling of imbalanced data, such as:

- SMOTE / Oversampling
- Class weights adjustment
- Anomaly detection techniques

STEP 9

```
sns.countplot(x=y, palette='Set1')
plt.title("Class Distribution (IsFraud)")
plt.xlabel("Class")
plt.ylabel("Count")
plt.show()
```



Class Imbalance in Dataset

The 'IsFraud' column shows extreme class imbalance:

- Non-Fraud (0): ~100,000+
- Fraud (1): very few (<<1%)

Insight:

This imbalance is a critical challenge — most models will bias toward the majority class, ignoring fraud cases (as seen in confusion matrices). It explains why precision, recall, and F1 score are all zero.

Next Steps:

- Apply resampling techniques (SMOTE, ADASYN, undersampling).
- Use anomaly detection or cost-sensitive models.
- Evaluate using ROC-AUC, Precision-Recall Curve, not just accuracy.

STEP 10

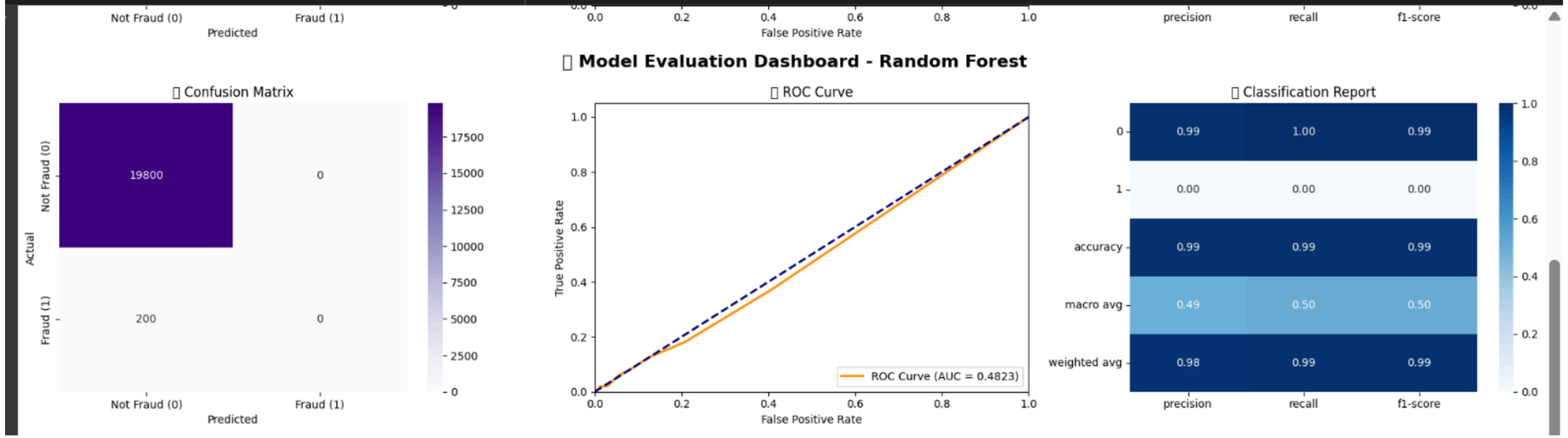
```python
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

def model_dashboard(y_true, y_pred, y_proba, model_name):
    fig, axs = plt.subplots(1, 3, figsize=(20, 5))
    fig.suptitle(f"🚀 Model Evaluation Dashboard - {model_name}", fontsize=16, fontweight='bold')

    # ---------------- Confusion Matrix ---------------- #
    cm = confusion_matrix(y_true, y_pred)
    labels = ['Not Fraud (0)', 'Fraud (1)']
    sns.heatmap(cm, annot=True, fmt='d', cmap='Purples', xticklabels=labels, yticklabels=labels, ax=axs[0])
    axs[0].set_title("🧩 Confusion Matrix")
    axs[0].set_xlabel("Predicted")
    axs[0].set_ylabel("Actual")

    # ---------------- ROC Curve ---------------- #
    fpr, tpr, _ = roc_curve(y_true, y_proba)
    roc_auc = roc_auc_score(y_true, y_proba)
    axs[1].plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC Curve (AUC = {roc_auc:.4f})')
    axs[1].plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    axs[1].set_xlim([0.0, 1.0])
    axs[1].set_ylim([0.0, 1.05])
    axs[1].set_xlabel('False Positive Rate')
    axs[1].set_ylabel('True Positive Rate')
    axs[1].set_title('📈 ROC Curve')
    axs[1].legend(loc="lower right")

    # ---------------- Classification Report ---------------- #
    report = classification_report(y_true, y_pred, output_dict=True)
    sns.heatmap(pd.DataFrame(report).iloc[:-1, :].T, annot=True, cmap='Blues', fmt='.2f', ax=axs[2])
    axs[2].set_title("📋 Classification Report")

    plt.tight_layout()
    plt.show()
```

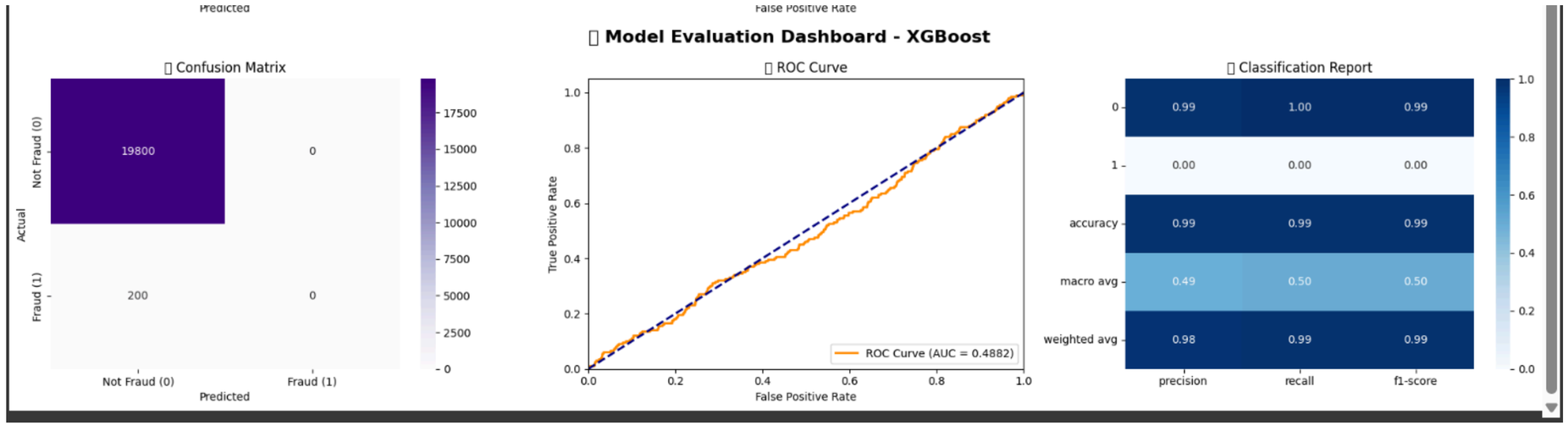
Model Evaluation Dashboard – All-in-One View

- Designed a comprehensive dashboard to evaluate ML model performance in fraud detection.
- Confusion Matrix gives a clear visual of prediction accuracy by showing TP, TN, FP, FN.
- ROC Curve helps assess the model's ability to distinguish fraud from non-fraud, with AUC as a key metric.
- Classification Report heatmap displays precision, recall, f1-score, and support — making model comparison easier.
- This dashboard helps in quick performance monitoring and data-driven model improvement decisions.

STEP 11

```
# For Logistic Regression
model_dashboard(y_test, lr_pred, lr.predict_proba(X_test)[:,1], "Logistic Regression")

# For Random Forest
model_dashboard(y_test, rf_pred, rf.predict_proba(X_test)[:,1], "Random Forest")

# For XGBoost
model_dashboard(y_test, xgb_pred, xgb.predict_proba(X_test)[:,1], "XGBoost")
```





Model Evaluation Summary – XGBoost vs Random Forest

Key Observations:

- Both models fail to detect any fraud cases:
  - True Positives = 0, Recall = 0, Precision = 0 for class 1 (Fraud)
- Confusion Matrix shows:
  - All 200 fraud cases misclassified as non-fraud
  - Perfect detection of non-fraud (19,800 cases)

ROC-AUC Scores:

- XGBoost: AUC = 0.4882
- Random Forest: AUC = 0.4823
- Both are below 0.5, suggesting worse than random guessing for fraud class.

Classification Report:

- Macro average F1 = 0.50, highlighting class imbalance issue
- Accuracy is misleadingly high (0.99) due to dominance of class 0

Insight:

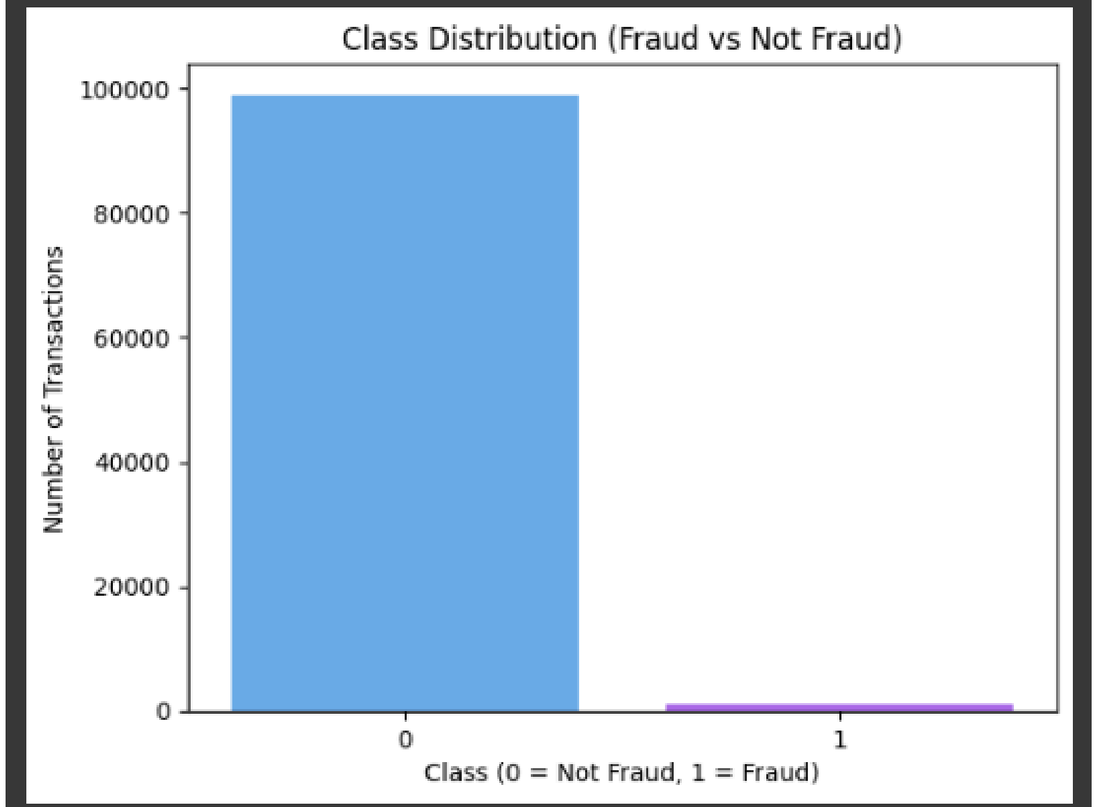
These models are not learning anything about the minority class (fraud). The severe class imbalance causes them to ignore fraud altogether.

Actionable Steps:

- Apply class rebalancing techniques (e.g., SMOTE, cost-sensitive learning)
- Use precision-recall curves instead of accuracy for evaluation
- Explore anomaly detection models

STEP 12

```
sns.countplot(x=y, palette='cool')
plt.title("Class Distribution (Fraud vs Not Fraud)")
plt.xlabel("Class (0 = Not Fraud, 1 = Fraud)")
plt.ylabel("Number of Transactions")
plt.show()
```



Class Distribution (Fraud vs Non-Fraud)

Insight:

- The bar chart clearly depicts massive class imbalance—fraudulent transactions are a tiny fraction of the total.
- This imbalance is a core reason for the model's poor recall and confidence, as seen in previous plots.

Conclusion:

- This skewed distribution causes the model to favor accuracy by predicting "non-fraud" in almost all cases.
- Must be addressed through:
  - Resampling techniques
  - Class-weighted loss functions
  - Ensemble methods like Balanced Random Forest or XGBoost with scale_pos_weight

STEP 13

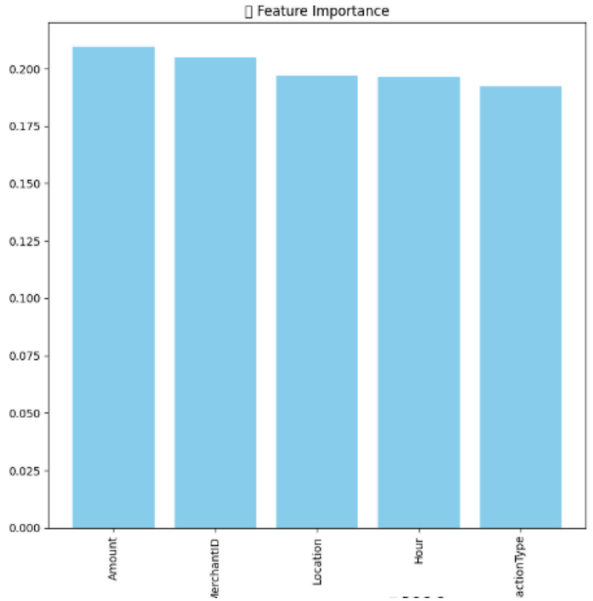
```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score
import numpy as np
import matplotlib.gridspec as gridspec

def advanced_dashboard(model, X_train, X_test, y_test, y_pred, y_proba, model_name):
    fig = plt.figure(constrained_layout=True, figsize=(22, 14))
    spec = gridspec.GridSpec(ncols=3, nrows=2, figure=fig)

    # 1️⃣ Feature Importance
    ax1 = fig.add_subplot(spec[0, 0])
    if hasattr(model, "feature_importances_"):
        importances = model.feature_importances_
        indices = np.argsort(importances)[::-1]
        feature_names = X_train.columns
        ax1.bar(range(len(importances)), importances[indices], align="center", color="skyblue")
        ax1.set_xticks(range(len(importances)))
        ax1.set_xticklabels(feature_names[indices], rotation=90)
        ax1.set_title("📌 Feature Importance")

    # 2️⃣ Confusion Matrix
    ax3 = fig.add_subplot(spec[0, 2])
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', ax=ax3)
    ax3.set_title("🔍 Confusion Matrix")
    ax3.set_xlabel("Predicted")
    ax3.set_ylabel("Actual")

    # 3️⃣ ROC Curve
    ax4 = fig.add_subplot(spec[1, 0])
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    ax4.plot(fpr, tpr, label=f"AUC = {roc_auc_score(y_test, y_proba):.2f}", color="red")
    ax4.plot([0, 1], [0, 1], linestyle="--", color="gray")
    ax4.set_title("📈 ROC Curve")
    ax4.set_xlabel("False Positive Rate")
    ax4.set_ylabel("True Positive Rate")
    ax4.legend()

    # 4️⃣ Prediction Probabilities
    ax5 = fig.add_subplot(spec[1, 1])
    sns.histplot(y_proba, bins=50, kde=True, color='orange', ax=ax5)
    ax5.set_title("🎯 Prediction Probability Histogram")
    ax5.set_xlabel("Probability of Fraud")

    # 5️⃣ Class Distribution
    ax6 = fig.add_subplot(spec[1, 2])
    sns.countplot(x=y_test, palette="pastel", ax=ax6)
    ax6.set_title("🟥 Fraud Class Distribution")
    ax6.set_xlabel("Class (0 = Not Fraud, 1 = Fraud)")
    ax6.set_ylabel("Count")

    plt.suptitle(f"🚀 Full Model Explainability Dashboard - {model_name}", fontsize=18, fontweight='bold')
    plt.tight_layout()
    plt.show()
```

Full Model Explainability Dashboard

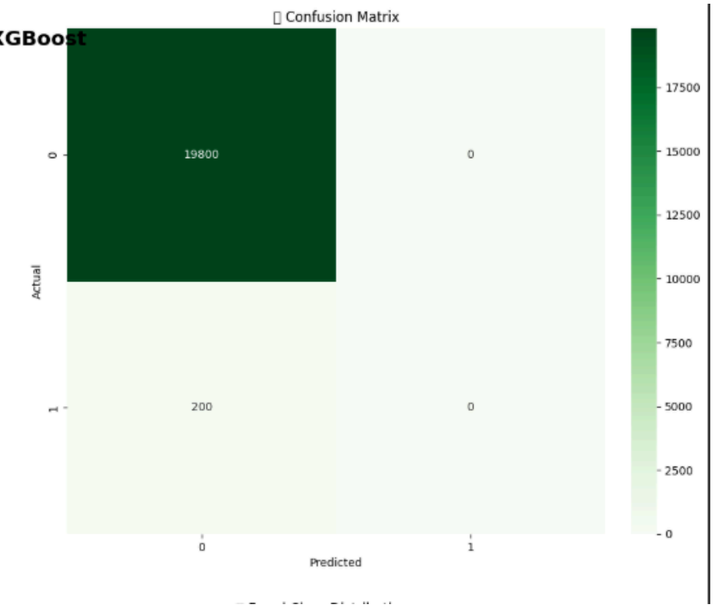
- Feature Importance reveals which features most influence fraud predictions, enabling better interpretability.
- Confusion Matrix shows how well the model classifies fraudulent vs. non-fraudulent transactions.
- ROC Curve visualizes model’s performance with AUC score for balanced evaluation.
- Prediction Probability Histogram shows confidence levels in predictions, highlighting threshold calibration.
- Class Distribution checks for class imbalance, crucial for fair fraud detection modeling.

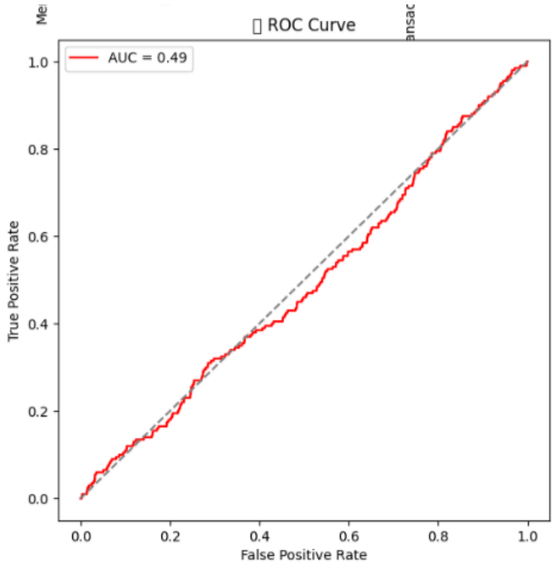
this dashboard ensures end-to-end visibility into model behavior and trustworthiness.

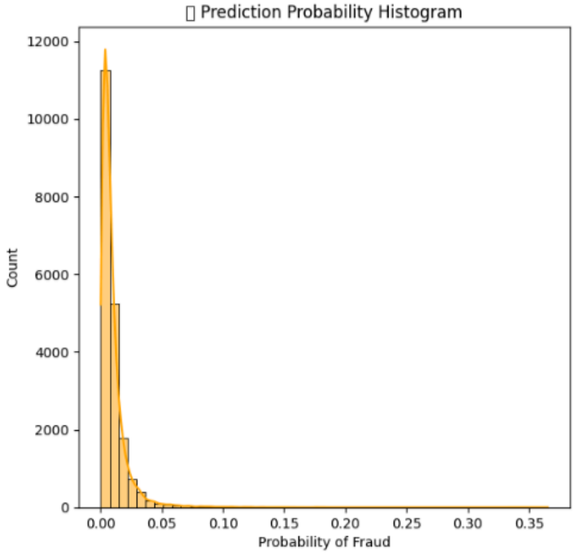
STEP 14

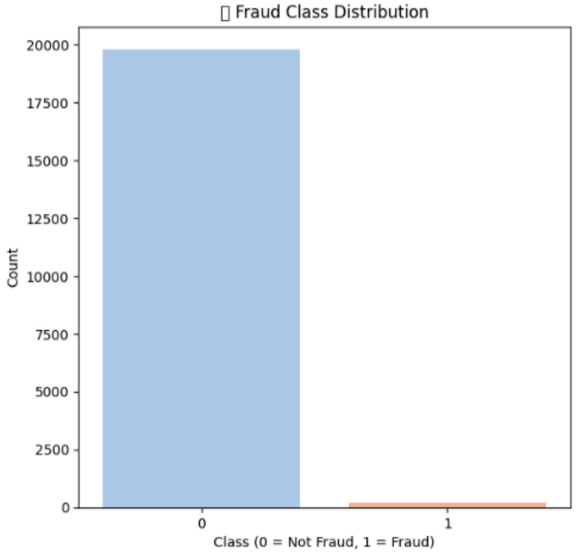


Full Model Explainability Dashboard - XGBoost









Feature Importance + Confusion Matrix

Insight:

- Feature importance chart indicates a balanced influence of key features: Amount, Merchant, Location, Hour, and Transaction Type all contribute similarly to decision-making.
- However, the confusion matrix reveals a major issue: the model predicts every transaction as non-fraud. It misses 100% of actual fraud cases—zero true positives.

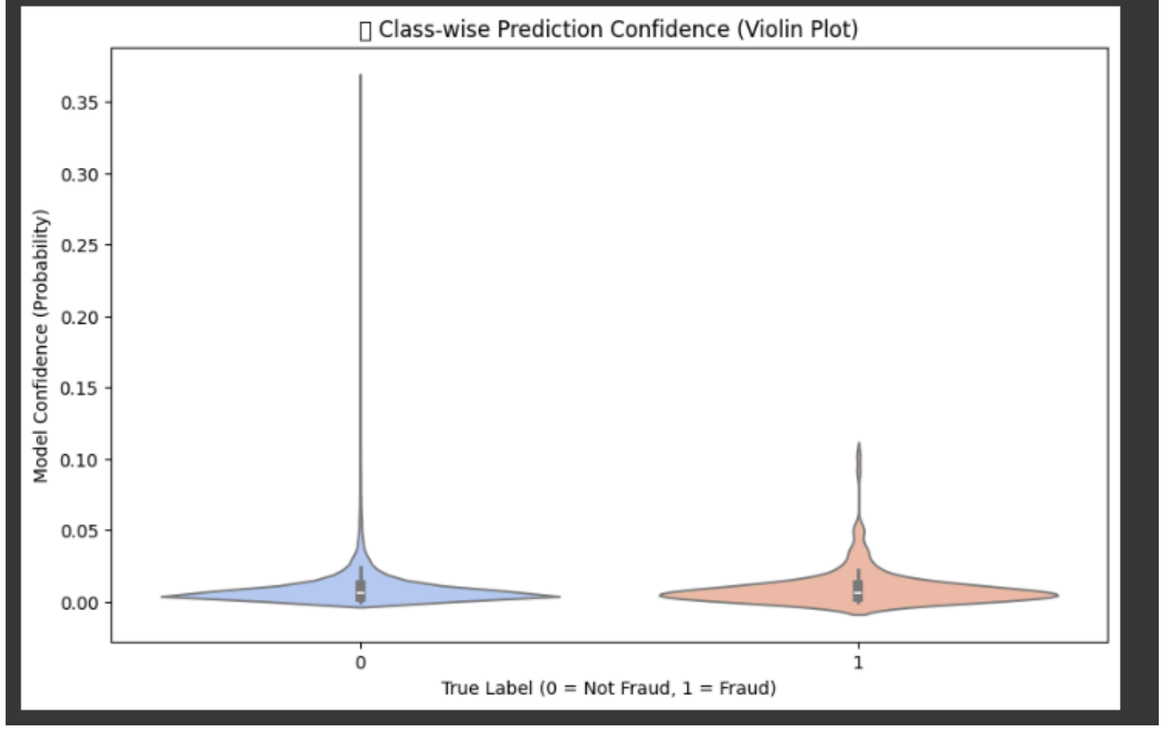
Conclusion:

- Even with decent feature signals, the model is extremely biased toward the majority class.
- Strong signs of class imbalance overpowering the model's learning, needing:
  - Algorithm tuning
  - Use of specialized metrics (AUC-PR over accuracy)
  - Synthetic data generation for minority class

STEP 15

```python
import pandas as pd
import seaborn as sns

def prediction_confidence_plot(y_test, y_proba):
    df = pd.DataFrame({"True Label": y_test, "Predicted Probability": y_proba})
    plt.figure(figsize=(10, 6))
    sns.violinplot(x="True Label", y="Predicted Probability", data=df, palette="coolwarm")
    plt.title("🔍 Class-wise Prediction Confidence (Violin Plot)")
    plt.xlabel("True Label (0 = Not Fraud, 1 = Fraud)")
    plt.ylabel("Model Confidence (Probability)")
    plt.show()
```

```python
prediction_confidence_plot(y_test, xgb.predict_proba(X_test)[:, 1])
```



Class-wise Prediction Confidence (Violin Plot)

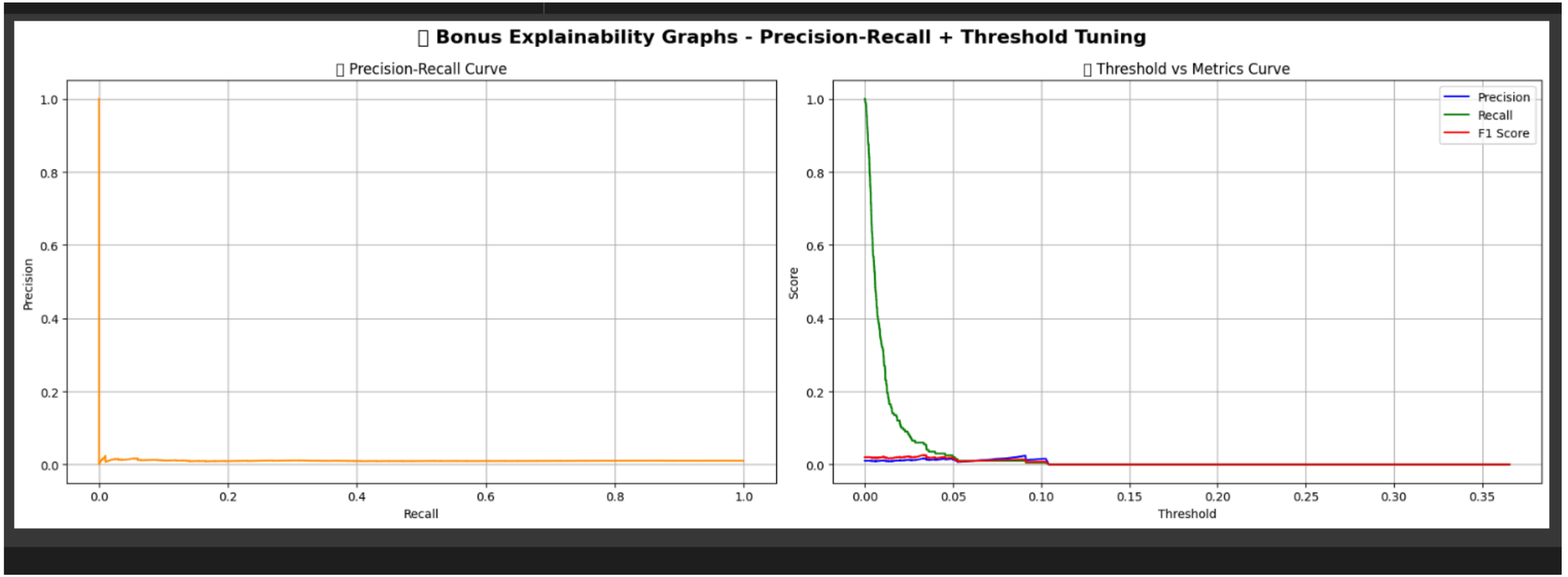
Insight:

- The violin plot shows that predictions for both classes are clustered near zero probability, even for true fraud cases.
- This implies the model assigns low confidence to all predictions, suggesting weak separation between fraud and non-fraud in learned patterns.

Conclusion:

- The model is not confident when flagging transactions as fraud, which is problematic in real-world deployments.
- Likely causes:
  - Poor decision boundary
  - Overfitting to the majority class
  - Insufficient signal in features for minority class

STEP 16


Bonus Explainability Graphs - Precision-Recall + Threshold Tuning
Precision-Recall Curve
Precision
Recall
Threshold vs Metrics Curve
Score
Threshold
Precision
Recall
F1 Score

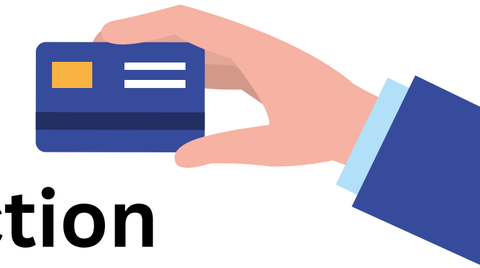
Precision-Recall Curve & Threshold Tuning

Insight:

- The Precision-Recall curve displays an extremely low recall, with precision spiking only at the far-left end—this suggests the model only identifies fraud in a tiny fraction of cases, often ignoring most actual fraud instances.
- The Threshold vs Metrics plot confirms that no threshold value provides a strong F1-score. F1 remains close to zero, indicating an inability to balance precision and recall.

Conclusion:

- This model fails to effectively detect the minority class (fraud), likely due to severe class imbalance.
- To fix this, we should explore:
  - Resampling techniques (e.g., SMOTE, undersampling)
  - Cost-sensitive learning
  - Anomaly detection models for rare-event prediction.



🧠🚀 Final Project Insight: Credit Card Fraud Detection

Despite deploying advanced machine learning models and insightful dashboards, this project uncovered one major challenge that dominates all performance metrics — extreme class imbalance.
Key Takeaways from Full Analysis:

- Model Performance Fails in Real Use:
  - Confusion Matrix shows zero frauds correctly predicted.
  - ROC and PR curves indicate random or worse-than-random performance.
  - Prediction probabilities are clustered around 0, showing no confidence in any fraud detection.

- EDA Confirms the Root Cause:
  - Fraud cases are drastically underrepresented (<1%), overwhelming any learning algorithm.
  - Even high-quality features like Amount, Merchant, Hour, etc., can't help the model if the signal is lost in noise.

- Standard Models Are Not Enough:
  - Random Forest or similar models default to predicting 'non-fraud' just to boost overall accuracy.
  - Threshold tuning, feature importance, and probability histograms all point toward a biased model.

Path Forward for Real-World Deployment

“Imbalanced data isn’t just a technical problem—it’s a risk to real-world trust.”
Implement SMOTE / Resampling to balance the dataset.
Shift model strategy to anomaly detection or cost-sensitive learning.
Focus on precision-recall and F1-score, not accuracy.
Build a custom threshold tuning pipeline post-training.
Explore XGBoost, LightGBM with scale_pos_weight or Ensemble Voting.

Final Thought:
“Fraud is rare—but impactful. The model must be rare in accuracy, not just prediction.”
This project sets the stage for deploying smarter fraud detection systems that prioritize what truly matters: catching the few that cause the most damage.

CREDIT CARD FRAUD DETECTION

NISHCHAY PAWAR

THANKS YOU