



UNVEILING DECEIT: FRAUD DETECTION THROUGH THE ART OF MACHINE LEARNING



AJAYI OLUWASEYI – DATA ANALYST

PROJECT OBJECTIVE

The goal is to detect fraudulent credit card transactions by building a machine learning model that accurately identifies fraud while minimizing false positives that could affect real customers.

DATASET OVERVIEW

① localhost:8888/notebooks/Fraud%20Detection%20analysis.ipynb?

jupyter Fraud Detection analysis Last Checkpoint: 12 hours ago

File Edit View Run Kernel Settings Help

[2]: `fraud_df = pd.read_csv("C:/Users/USER/Downloads/creditcard.csv/creditcard.csv")`

[26]: `print(fraud_df.head())`

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class		
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239591	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0		
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078800	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	-0.125895	-0.008983	0.014724	2.69	0		
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791460	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.221929	0.062723	0.061458	123.50	0		
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237600	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.270533	0.817739	-0.009431	0.798278	-0.137458	0.141267	-0.206010
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592940	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	

[5 rows x 31 columns]

[10]: `# Plot class distribution`

- Source: creditcard.csv (~285,000 rows)
- Features: PCA-transformed V1–V28, Time, Amount, Class
- Class distribution:
- Non-Fraud: 99.83%
- Fraud: 0.17% (492 cases)

KEY CHALLENGES

Severe Class Imbalance

- In the dataset, fraud cases represent only 0.17% of all transactions.
- This extreme imbalance means that if a model simply predicts "non-fraud" for every transaction, it would still achieve over 99% accuracy, but fail at its true purpose: detecting fraud.
- Therefore, accuracy alone is misleading — more advanced evaluation metrics like precision, recall, and F1-score must be prioritized.

Fraud is Rare but High-Impact

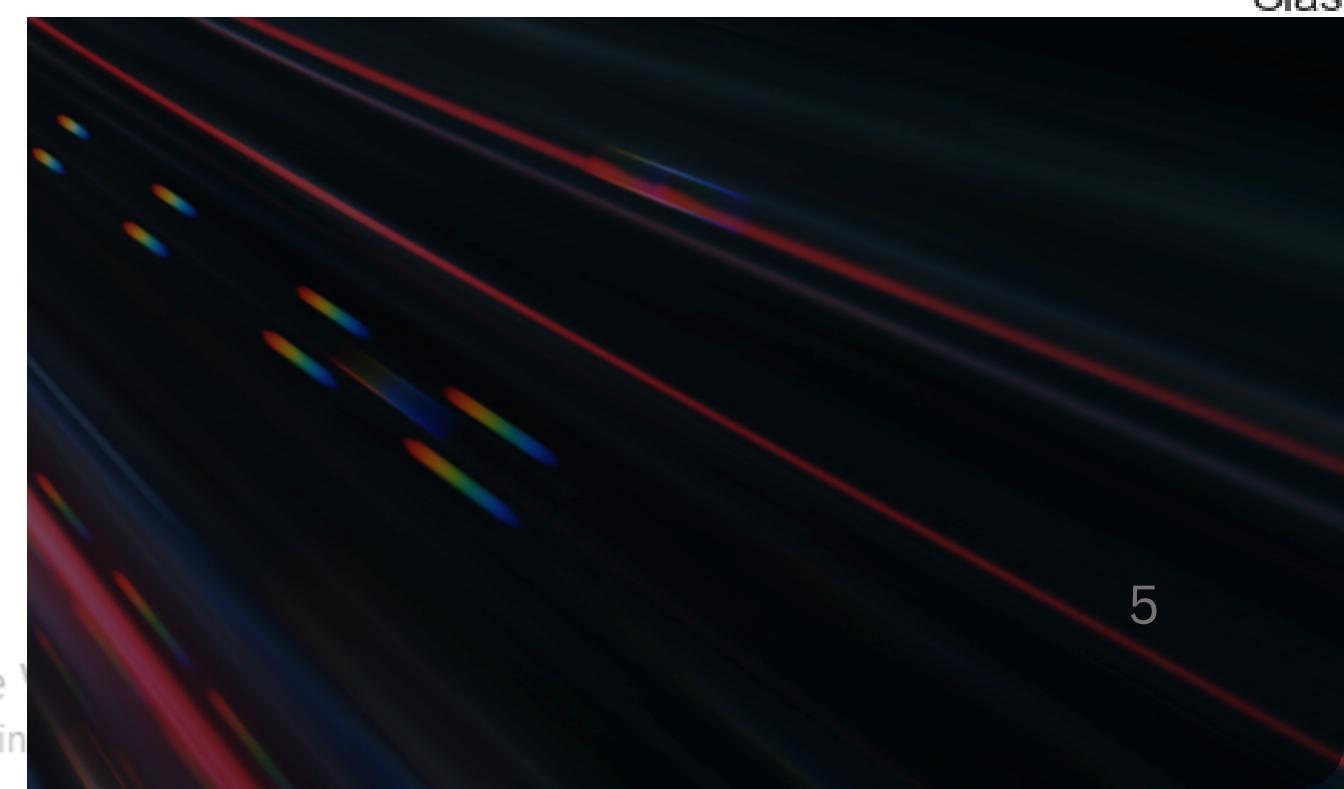
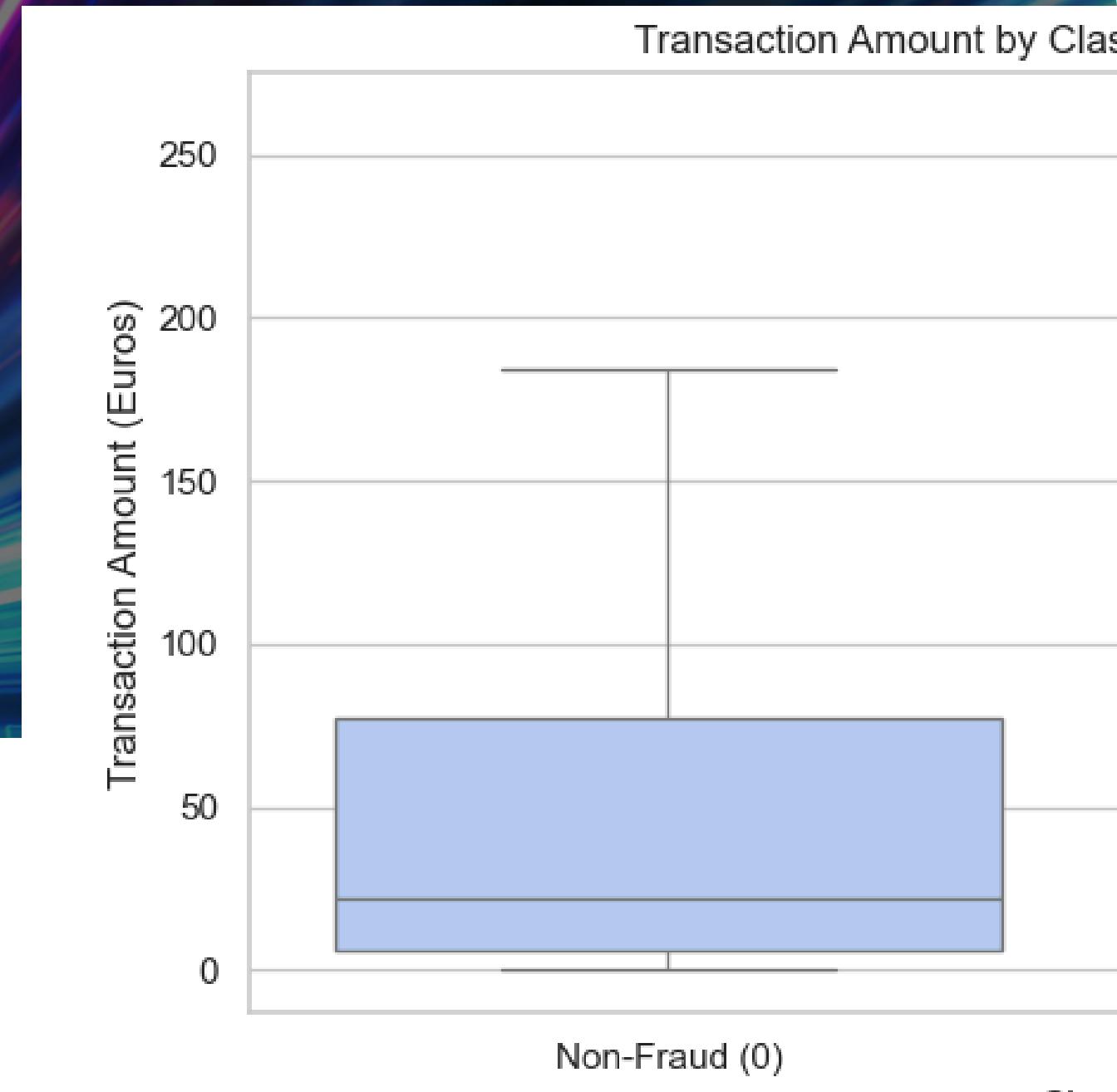
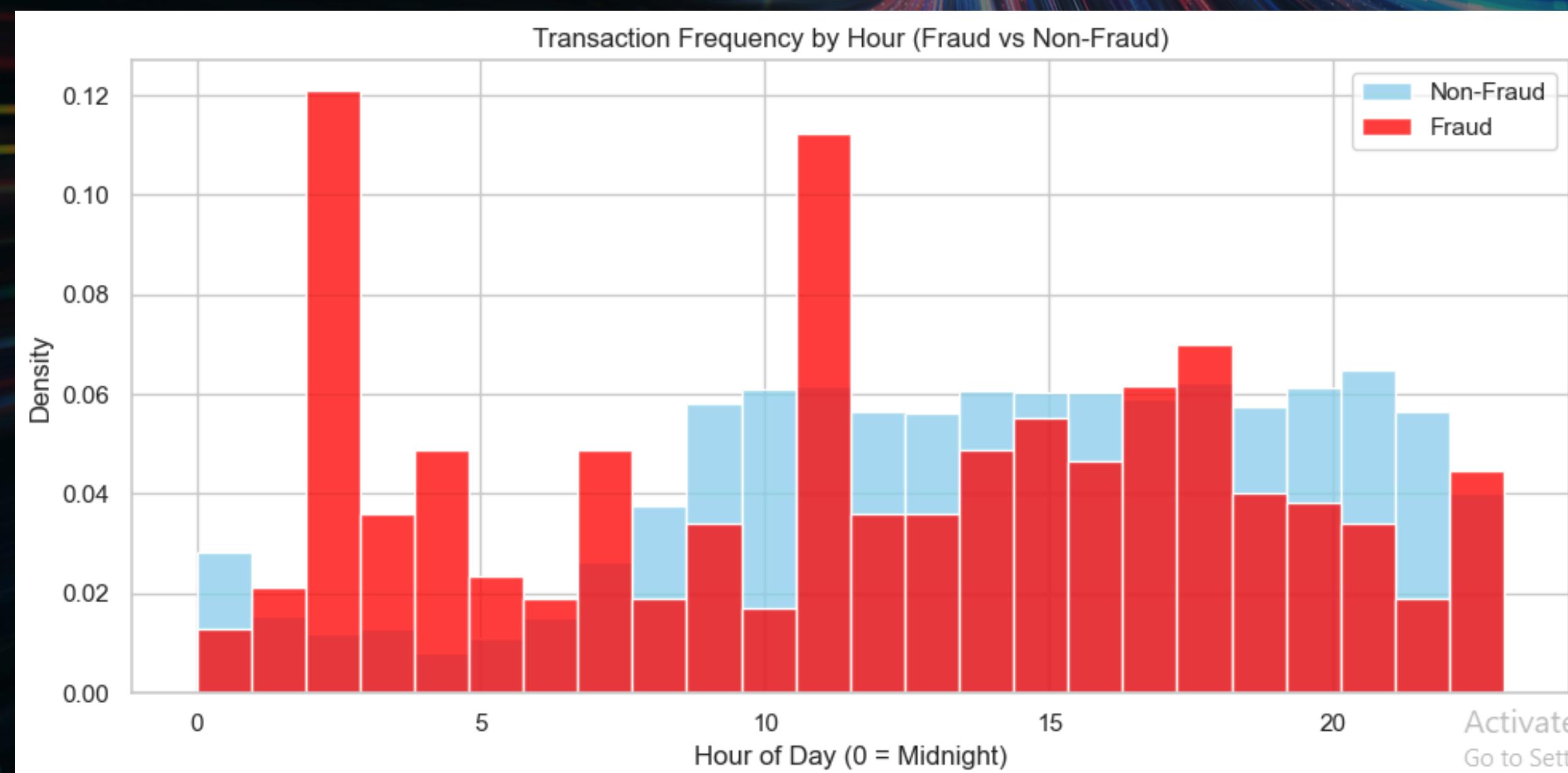
- Although fraud is rare, the financial and reputational impact of missing fraudulent transactions is enormous.
- Even a single undetected fraud can cause serious losses to customers and businesses.
- The cost of missing a fraud is much higher than the cost of wrongly flagging a legitimate transaction.

Balancing Detection vs. Customer Experience

- It's not enough to just detect fraud — we must also avoid too many false positives.
- Wrongly flagging legitimate transactions can:
 - Frustrate loyal customers
 - Increase operational costs (manual review teams)
 - Damage brand trust
 - Therefore, our model must carefully balance sensitivity (catching fraud) with specificity (not bothering real customers).

- Fraud spikes around 2AM and 11AM
- Amount of fraud transactions is inconsistent (not always high-value)
- Features V10, V12, V14 showed strong correlation with fraud
- Class distribution visual confirmed imbalance

EDA INSIGHTS



PREPROCESSING STEPS

Feature Scaling (to standardize)

Some models (like Logistic Regression and Neural Networks) are sensitive to the scale of numbers. Features like:

Amount → can range from 0 to 25,000+ Time → can go up to 170,000+

Other features (V1 to V28) are already standardized.

So we only need to scale Amount and maybe Time to bring everything to a similar scale.

```
[22]: from sklearn.preprocessing import StandardScaler
```

```
[24]: # this create a copy to keep original data safe  
scaled_df = fraud_df.copy()
```

```
# Columns to scale  
cols_to_scale = ['Amount', 'Time']  
  
# activating SCaler  
scaler = StandardScaler()
```

```
# Apply scaler  
scaled_df[cols_to_scale] = scaler.fit_transform(scaled_df[cols_to_scale])  
  
# Preview scaled data
```

jupyter Fraud Detection analysis Last Checkpoint: 2 days ago

File Edit View Run Kernel Settings Help

Code

```
[22]: from sklearn.preprocessing import StandardScaler
```

```
[24]: # this create a copy to keep original data safe  
scaled_df = fraud_df.copy()
```

```
|  
# Columns to scale  
cols_to_scale = ['Amount', 'Time']  
  
# activating SCaler  
scaler = StandardScaler()
```

```
# Apply scaler  
scaled_df[cols_to_scale] = scaler.fit_transform(scaled_df[cols_to_scale])
```

```
# Preview scaled data  
print(scaled_df[['Amount', 'Time']].head())
```

	Amount	Time
0	0.244964	-1.996583
1	-0.342475	-1.996583
2	1.160686	-1.996562
3	0.140534	-1.996562
4	-0.073403	-1.996541

jupyter Fraud Detection analysis Last Checkpoint: 2 days ago

File Edit View Run Kernel Settings Help

Code

prepare for modeling

Define features (X) and labels (y) Split into training & testing sets

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

```
[28]: # Define features  
X = scaled_df.drop(['Class'], axis=1)  
y = scaled_df['Class']
```

```
# Split the data with stratification  
X_train, X_test, y_train, y_test = train_test_split(  
    X, y, test_size=0.2, stratify=y, random_state=42  
)
```

```
[30]: print("Training set:", X_train.shape)  
print("Testing set:", X_test.shape)  
print("Fraud in train:", sum(y_train))  
print("Fraud in test:", sum(y_test))
```

Training set: (227845, 31)
Testing set: (56962, 31)
Fraud in train: 394
Fraud in test: 98

jupyter Fraud Detection analysis Last Checkpoint: 2 days ago

File Edit View Run Kernel Settings Help

Code

My model will learn from 394 real fraud examples and 227,451 non-fraud

It'll be tested on 98 fraud cases, which is enough to get a reliable measure of R

```
[32]: # let use Logistic Regression to Detect Fraud
```

```
from sklearn.linear_model import LogisticRegression  
from sklearn.metrics import classification_report, confusion_matrix,
```

```
[34]: #activating the model  
log_reg = LogisticRegression(max_iter=1000)
```

```
#this train the model  
log_reg.fit(X_train, y_train)
```

```
#telling the model to make prediction
```

```
y_pred = log_reg.predict(X_test)
```

```
[36]: print("Report:")  
print(classification_report(y_test, y_pred, digits=4))
```

Report:
precision recall f1-score support

MODELS TESTED

Model	Precision	Recall	F1-Score
Logistic Regression (raw)	82.9%	64%	72.4%
Logistic Regression + SMOTE	9.5%	91%	17.3%
Random Forest (raw)	96.3%	80.6%	87.8%
Random Forest + SMOTE	85.1%	81.6%	83.3%

This table highlights the performance trade-offs across different models. While Logistic Regression with SMOTE improved recall, it significantly reduced precision, leading to many false positives. Random Forest, even without any oversampling, achieved the best balance — combining high precision (96.3%) and strong recall (80.6%), resulting in the highest F1-score. These results support Random Forest (trained on imbalanced data) as the most reliable model for real-world fraud detection.

FINAL MODEL SELECTION

RANDOM FOREST (TRAINED ON ORIGINAL IMBALANCED DATA)

jupyter Fraud Detection analysis Last Checkpoint: 3 days ago

File Edit View Run Kernel Settings Help Trust JupyterLab Python 3 (ipykernel)

Why Start With Imbalanced Data (original data) for Random Forest?

Test raw model ability | We want to see how well Random Forest performs before any boosting or balancing. RF handles imbalance better than LR | Because it can internally split based on Gini/entropy and still catch minority class patterns. We compare apples to apples | This keeps it fair when comparing with original Logistic Regression

```
[72]: print("Random Forest - Classification Report (Imbalanced):")
print(classification_report(y_test, y_pred_rf, digits=4))

Random Forest - Classification Report (Imbalanced):
precision    recall    f1-score   support
          0       0.9997    0.9999    0.9998    56864
          1       0.9634    0.8061    0.8778     98

      accuracy         0.9996    56962
     macro avg       0.9815    0.9030    0.9388    56962
  weighted avg       0.9996    0.9996    0.9996    56962
```

```
[74]: print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_rf))

Confusion Matrix:
[[56861    3]
 [ 19    79]]
```

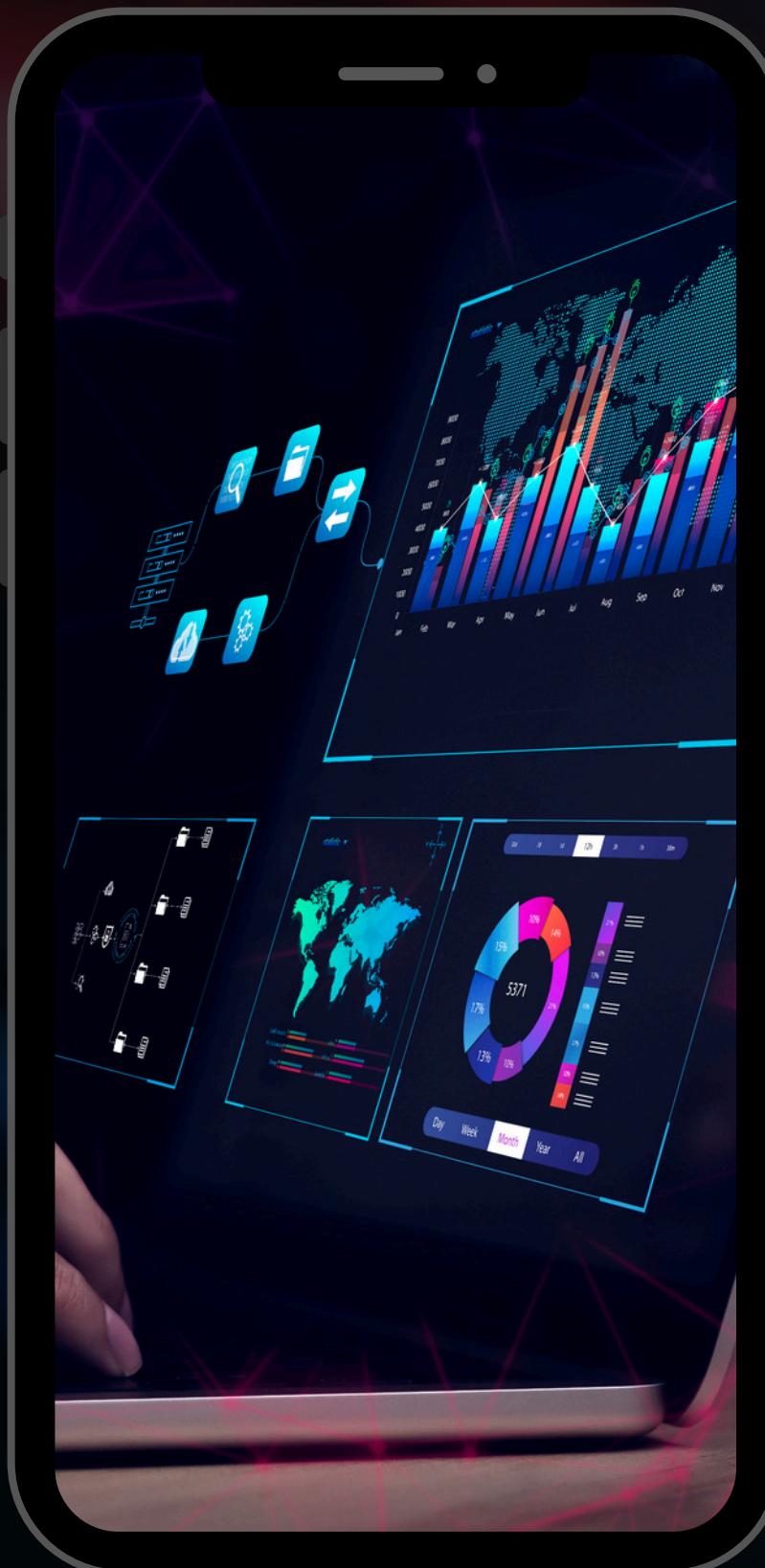
```
[76]: print("ROC AUC Score:", roc_auc_score(y_test, rf_model.predict_proba(X_test)[:,1]))
```

ROC AUC Score: 0.9527451104245863

Activate Windows
Go to Settings to activate W

- BEST BALANCE OF RECALL AND PRECISION
- VERY FEW FALSE ALARMS (ONLY 3)
- STRONG F1-SCORE AND ROC AUC
- NO NEED FOR EXTRA OVERSAMPLING STEPS

BUSINESS IMPACT



- Catches majority of frauds
 - Reduces false alarms — smoother experience for real customers
 - Scalable and explainable model for real-world deployment
 - Supports security teams in early fraud detection
-
- The selected Random Forest model achieves a strong balance between catching fraudulent activity and minimizing false alarms. With over 80% fraud recall and only a few false positives, it ensures that suspicious transactions are flagged early without negatively impacting genuine users. This reduces financial risk, protects customer trust, and supports fraud investigation teams with accurate, scalable, and explainable predictions — making it suitable for both real-time systems and batch processing environments.

Thank You

Ajayi Oluwaseyi

Data Analyst | Psychology Meets Data Science