```
# Financial Fraud Detection - Complete Balanced Implementation
## 1. Install Required Packages
!pip install tensorflow scikit-learn imbalanced-learn transformers > /dev/null
```

```
## 2. Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# 3. Load Data from Google Drive
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Update this path to your dataset location in Drive
data_path = '/content/drive/MyDrive/Colab Notebooks/Fraud Detection/creditcard.csv'
df = pd.read_csv(data_path)

# Verify load
print("Data loaded successfully. Shape:", df.shape)
display(df.head())
```

```
Data loaded successfully. Shape: (284807, 31)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	(
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-(
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-(
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	(

5 rows × 31 columns

```
# Feature engineering
scaler = RobustScaler()
df['Amount'] = scaler.fit_transform(df['Amount'].values.reshape(-1,1))
df['Time'] = scaler.fit_transform(df['Time'].values.reshape(-1,1))
X = df.drop('Class', axis=1)
y = df['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

```
## 5. Data Preprocessing
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

# Normalize Time and Amount
scaler = RobustScaler()
df['Amount'] = scaler.fit_transform(df['Amount'].values.reshape(-1,1))
df['Time'] = scaler.fit_transform(df['Time'].values.reshape(-1,1))

# Train-test split
X = df.drop('Class', axis=1)
y = df['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

# Handle imbalance with SMOTE
sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)

print("\nClass distribution after SMOTE:")
print(pd.Series(y_train_res).value_counts())
```

```
Class distribution after SMOTE:
Class
0    227451
1    227451
```

```
Name: count, dtype: int64
```

```
## 2. LSTM Model (Improved Version)
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

def create_sequences(data, labels, seq_length=30):
    seq, targets = [], []
    for i in range(seq_length, len(data)):
        seq.append(data.iloc[i-seq_length:i].values)
        targets.append(labels.iloc[i])
    return np.array(seq), np.array(targets)

X_train_seq, y_train_seq = create_sequences(pd.DataFrame(X_train), y_train)
X_test_seq, y_test_seq = create_sequences(pd.DataFrame(X_test), y_test)

lstm_model = Sequential([
    LSTM(64, input_shape=(X_train_seq.shape[1], X_train_seq.shape[2])),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])
lstm_model.compile(optimizer='adam', loss='binary_crossentropy',
                   metrics=['Precision', 'Recall', 'AUC'])

# Train with class weights
history = lstm_model.fit(X_train_seq, y_train_seq, epochs=15, batch_size=64,
                         validation_split=0.1, class_weight={0:1, 1:30})
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_di
  super().__init__(**kwargs)
Epoch 1/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 111s 32ms/step - AUC: 0.4790 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.2191 - va
Epoch 2/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 90s 28ms/step - AUC: 0.5247 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.2086 - val
Epoch 3/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 85s 26ms/step - AUC: 0.5597 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.2081 - val
Epoch 4/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 138s 25ms/step - AUC: 0.5742 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.2062 - va
Epoch 5/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 102s 31ms/step - AUC: 0.5756 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.2107 - va
Epoch 6/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 131s 28ms/step - AUC: 0.6699 - Precision: 0.3161 - Recall: 0.0063 - loss: 0.1958 - val_AUC: 0
Epoch 7/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 138s 27ms/step - AUC: 0.7162 - Precision: 0.2155 - Recall: 0.0159 - loss: 0.1801 - val_AUC: 0
Epoch 8/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 141s 26ms/step - AUC: 0.7923 - Precision: 0.2654 - Recall: 0.0245 - loss: 0.1569 - val_AUC: 0
Epoch 9/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 87s 27ms/step - AUC: 0.8599 - Precision: 0.1438 - Recall: 0.0905 - loss: 0.1489 - val_AUC: 0.
Epoch 10/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 86s 27ms/step - AUC: 0.9079 - Precision: 0.1688 - Recall: 0.1483 - loss: 0.1273 - val_AUC: 0.
Epoch 11/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 146s 28ms/step - AUC: 0.9471 - Precision: 0.1432 - Recall: 0.2792 - loss: 0.1088 - val_AUC: 0
Epoch 12/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 137s 27ms/step - AUC: 0.9733 - Precision: 0.1885 - Recall: 0.4542 - loss: 0.0810 - val_AUC: 0
Epoch 13/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 139s 26ms/step - AUC: 0.9853 - Precision: 0.1866 - Recall: 0.5891 - loss: 0.0651 - val_AUC: 0
Epoch 14/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 144s 26ms/step - AUC: 0.9885 - Precision: 0.2264 - Recall: 0.7266 - loss: 0.0504 - val_AUC: 0
Epoch 15/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 146s 28ms/step - AUC: 0.9924 - Precision: 0.2565 - Recall: 0.8238 - loss: 0.0401 - val_AUC: 0
```

```
## 3. Autoencoder Model
from tensorflow.keras import Model, Input

input_dim = X_train.shape[1]
encoding_dim = 14

input_layer = Input(shape=(input_dim,))
encoder = Dense(encoding_dim, activation='relu')(input_layer)
decoder = Dense(input_dim, activation='sigmoid')(encoder)

autoencoder = Model(inputs=input_layer, outputs=decoder)
autoencoder.compile(optimizer='adam', loss='mse')

# Train on normal transactions only
normal_idx = y_train == 0
```

```
autoencoder.fit(X_train[normal_idx], X_train[normal_idx],
               epochs=20, batch_size=64, validation_split=0.1)
```

```
Epoch 1/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - loss: 1.2769 - val_loss: 1.1652
Epoch 2/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - loss: 1.1764 - val_loss: 1.1460
Epoch 3/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - loss: 1.1446 - val_loss: 1.1405
Epoch 4/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - loss: 1.0911 - val_loss: 1.1377
Epoch 5/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step - loss: 1.0813 - val_loss: 1.1357
Epoch 6/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - loss: 1.1152 - val_loss: 1.1348
Epoch 7/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - loss: 1.0850 - val_loss: 1.1338
Epoch 8/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step - loss: 1.0925 - val_loss: 1.1335
Epoch 9/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 18s 3ms/step - loss: 1.0768 - val_loss: 1.1331
Epoch 10/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - loss: 1.0467 - val_loss: 1.1325
Epoch 11/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step - loss: 1.1283 - val_loss: 1.1322
Epoch 12/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step - loss: 1.0775 - val_loss: 1.1320
Epoch 13/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - loss: 1.1499 - val_loss: 1.1319
Epoch 14/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 12s 3ms/step - loss: 1.0789 - val_loss: 1.1317
Epoch 15/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step - loss: 1.1134 - val_loss: 1.1317
Epoch 16/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 18s 3ms/step - loss: 1.0629 - val_loss: 1.1318
Epoch 17/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step - loss: 1.0853 - val_loss: 1.1315
Epoch 18/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - loss: 1.0458 - val_loss: 1.1314
Epoch 19/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - loss: 1.0556 - val_loss: 1.1315
Epoch 20/20
3199/3199 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step - loss: 1.0832 - val_loss: 1.1313
<keras.src.callbacks.history.History at 0x7d29aa7a4510>
```

```
# 1. TRANSFORMER IMPLEMENTATION
from tensorflow.keras.layers import LayerNormalization, MultiHeadAttention, Embedding, GlobalAveragePooling1D
from tensorflow.keras import Input, Model

class TransformerBlock(tf.keras.layers.Layer):
    def __init__(self, embed_dim, num_heads):
        super().__init__()
        self.att = MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim)
        self.layernorm1 = LayerNormalization()
        self.layernorm2 = LayerNormalization()
        self.dense = tf.keras.Sequential([
            Dense(embed_dim, activation='gelu'),
            Dense(embed_dim)
        ])

    def call(self, inputs):
        attn_output = self.att(inputs, inputs)
        out1 = self.layernorm1(inputs + attn_output)
        ff_output = self.dense(out1)
        return self.layernorm2(out1 + ff_output)

def build_transformer_model(input_shape):
    inputs = Input(shape=input_shape)

    # Feature Embedding
    x = Dense(64)(inputs)

    # Positional Encoding
    positions = tf.range(start=0, limit=input_shape[0], delta=1)
    position_embedding = Embedding(input_dim=input_shape[0], output_dim=64)(positions)
    x += position_embedding

    # Transformer Blocks
    x = TransformerBlock(embed_dim=64, num_heads=4)(x)
    x = GlobalAveragePooling1D()(x)
```

```
    outputs = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=inputs, outputs=outputs)
    model.compile(optimizer='adam',
                 loss='binary_crossentropy',
                 metrics=['Precision', 'Recall', 'AUC'])
    return model

# 2. DATA PREPARATION FOR TRANSFORMER
# Using same sequences as LSTM for fair comparison
X_train_trans = X_train_seq
y_train_trans = y_train_seq
X_test_trans = X_test_seq
y_test_trans = y_test_seq

# 3. TRAINING THE TRANSFORMER
transformer = build_transformer_model(X_train_trans.shape[1:])
transformer.fit(X_train_trans, y_train_trans,
                epochs=15,
                batch_size=64,
                validation_split=0.1,
                class_weight={0:1, 1:30})
```

```
Epoch 1/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 302s 91ms/step - AUC: 0.4615 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.2202 - va
Epoch 2/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 315s 89ms/step - AUC: 0.5030 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.2031 - va
Epoch 3/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 318s 88ms/step - AUC: 0.4955 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.2096 - va
Epoch 4/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 317s 86ms/step - AUC: 0.5604 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.2226 - va
Epoch 5/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 323s 87ms/step - AUC: 0.5077 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.2015 - va
Epoch 6/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 322s 87ms/step - AUC: 0.5277 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.2009 - va
Epoch 7/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 326s 88ms/step - AUC: 0.5523 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.2023 - va
Epoch 8/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 320s 87ms/step - AUC: 0.5449 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.2030 - va
Epoch 9/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 280s 87ms/step - AUC: 0.5455 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.2057 - va
Epoch 10/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 321s 87ms/step - AUC: 0.5271 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.1974 - va
Epoch 11/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 279s 87ms/step - AUC: 0.5563 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.2046 - va
Epoch 12/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 334s 91ms/step - AUC: 0.5510 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.2112 - va
Epoch 13/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 277s 87ms/step - AUC: 0.5649 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.1943 - va
Epoch 14/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 301s 94ms/step - AUC: 0.5659 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.2047 - va
Epoch 15/15
3204/3204 ━━━━━━━━━━━━━━━━━━━━ 305s 89ms/step - AUC: 0.5742 - Precision: 0.0000e+00 - Recall: 0.0000e+00 - loss: 0.2151 - va
<keras.src.callbacks.history.History at 0x7d29acf1f610>
```

```
## 5. Comparative Evaluation
def evaluate_model(model, X_test, y_test, model_type='lstm'):
    if model_type == 'autoencoder':
        reconstructions = model.predict(X_test)
        mse = np.mean(np.power(X_test - reconstructions, 2), axis=1)
        y_pred = (mse > np.percentile(mse, 95)).astype(int)  # Anomaly threshold
    else:
        y_pred = (model.predict(X_test) > 0.5).astype(int)

    from sklearn.metrics import classification_report
    print(classification_report(y_test, y_pred))

print("LSTM Performance:")
evaluate_model(lstm_model, X_test_seq, y_test_seq)

print("\nAutoencoder Performance:")
evaluate_model(autoencoder, X_test, y_test, 'autoencoder')

print("\n=== Transformer Evaluation ===")
evaluate_model(transformer, X_test_trans, y_test_trans)
```

```
LSTM Performance:
1780/1780 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     56834
           1       0.00      0.00      0.00        98

    accuracy                           0.99     56932
   macro avg       0.50      0.50      0.50     56932
weighted avg       1.00      0.99      0.99     56932


Autoencoder Performance:
1781/1781 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
              precision    recall  f1-score   support

           0       1.00      0.95      0.97     56864
           1       0.03      0.87      0.06        98

    accuracy                           0.95     56962
   macro avg       0.51      0.91      0.52     56962
weighted avg       1.00      0.95      0.97     56962


=== Transformer Evaluation ===
1780/1780 ━━━━━━━━━━━━━━━━━━━━ 25s 14ms/step
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56834
           1       0.00      0.00      0.00        98

    accuracy                           1.00     56932
   macro avg       0.50      0.50      0.50     56932
weighted avg       1.00      1.00      1.00     56932

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```