www.reallygreatsite.com

CREDIT CARD FRAUD DETECTION

Presented by :

Amira almaashani
Noor fadhil
Afnan Bait Zayed

INTRODUCTION

This project aims to build a machine learning model that detects fraudulent credit card transactions with high accuracy while handling imbalanced data. The model will analyze transaction features and classify each transaction as Fraud (1) or Not fraud(0).

PROJECT OBJECTIVES

DETECTION

Detect fraudulent transactions with high recall (catch as many frauds as possible).

SCALABILITY

Handle large, highly imbalanced datasets.

COMPARISON

Compare models (KNN, Logistic Regression, Random Forest, etc.).

EVALUATION

Provide evaluation metrics like Accuracy, Precision, Recall, F1-score, Confusion Matrix.

DATASET

DATASET:

Dataset: Credit Card Fraud Detection (Kaggle)

DESCRIPTION :

- 284,807 transactions
  - 492 fraud cases (highly imbalanced)
- 30 features:
  - V1–V28 (PCA-transformed features)
  - Time
  - Amount
- Target label: Class (0 = normal, 1 = fraud)

```
# Load the CSV file into a DataFrame
data = pd.read_csv("creditcard.csv")
data.head()# Display the first 5 rows
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	
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	0.066928	0.128539	-0.1
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	-0.339846	0.167170	0.1
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	-0.689281	-0.327642	-0.1
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	-1.175575	0.647376	-0.2
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	0.141267	-0.206010	0.5

5 rows × 31 columns

DATA PREPROCESSING REQUIREMENTS

- Remove missing or corrupted values
- Remove duplicated

```
data.duplicated().sum()
```

```
np.int64(9144)
```

```
data.drop_duplicates()
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V24
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.066928
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.339846
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.689281
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.175575
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.141267
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.509348
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.016226
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.640134
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.123205
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	-0.915427	...	0.261057	0.643078	0.376777	0.008797

275663 rows × 30 columns

```
#Missing Values
data.isnull().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

DATA PREPROCESSING REQUIREMENTS

- Apply scaling to numerical features:
  - StandardScaler() for:
    - Time
    - Amount

```
#Scale the 'Amount' column
from sklearn.preprocessing import StandardScaler
data['scaled_Amount'] = StandardScaler().fit_transform(data['Amount'].values.reshape(-1,1))
data = data.drop(['Amount'],axis=1)# Drop the original 'Amount' column
```

```
## Convert Time from seconds to hours
data["Time_Hr"] = data["Time"]/3600
print(data["Time_Hr"].tail(5))# Show the last 5 rows of the new column
```

```
284802    47.996111
284803    47.996389
284804    47.996667
284805    47.996667
284806    47.997778
Name: Time_Hr, dtype: float64
```

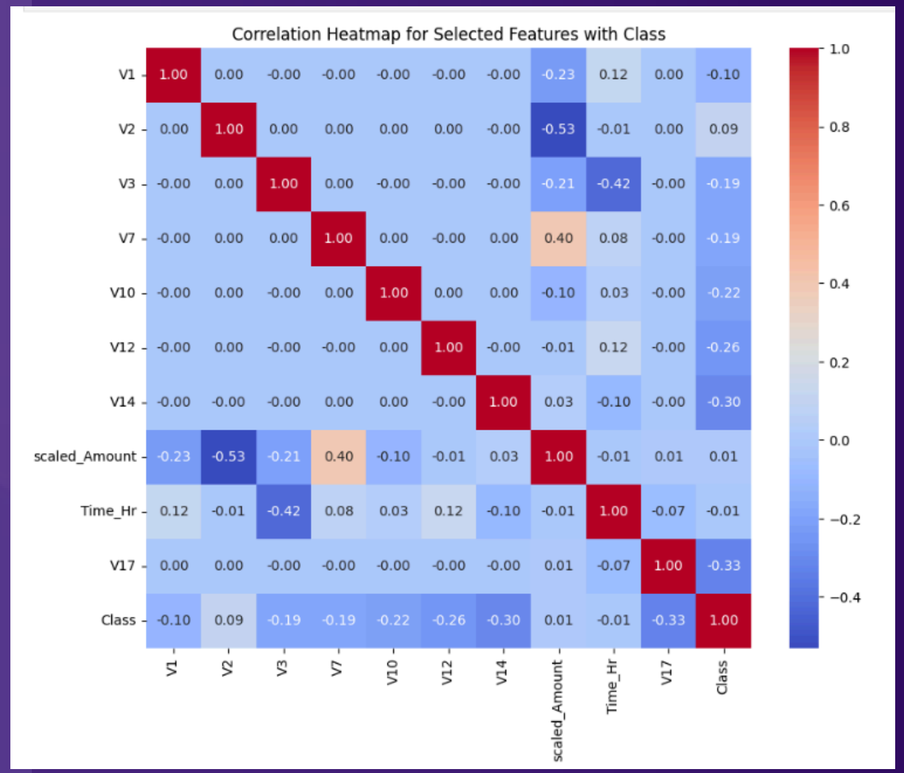
TOP FEATURES

```
# to sort the relation  with class
corr = data.corr()['Class'].abs().sort_values(ascending=False)

# top 10 Features
top10 = corr.index[:10]
print(top10)
```

```
Index(['Class', 'V17', 'V14', 'V12', 'V10', 'V16', 'V3', 'V7', 'V11', 'V4'], dtype='object')
```

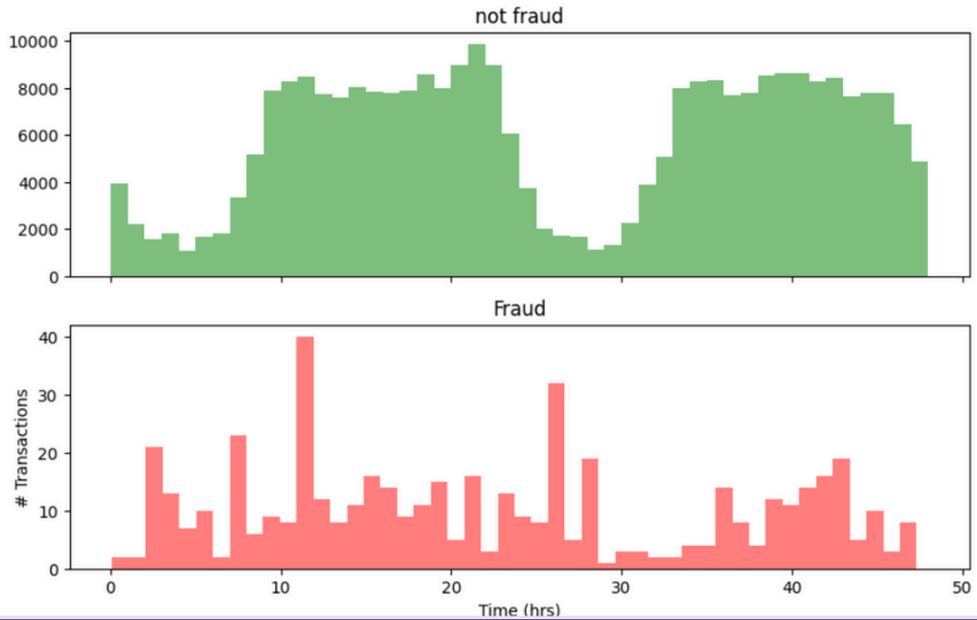
CORRELATION

```
import seaborn as sns
import matplotlib.pyplot as plt

# Select the features plus the target
df_corr = data[features + ['Class']]

# Compute the correlation matrix
corr_matrix = df_corr.corr()

# Plot the heatmap
plt.figure(figsize=(10,8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap for Selected Features with Class")
plt.show()
```

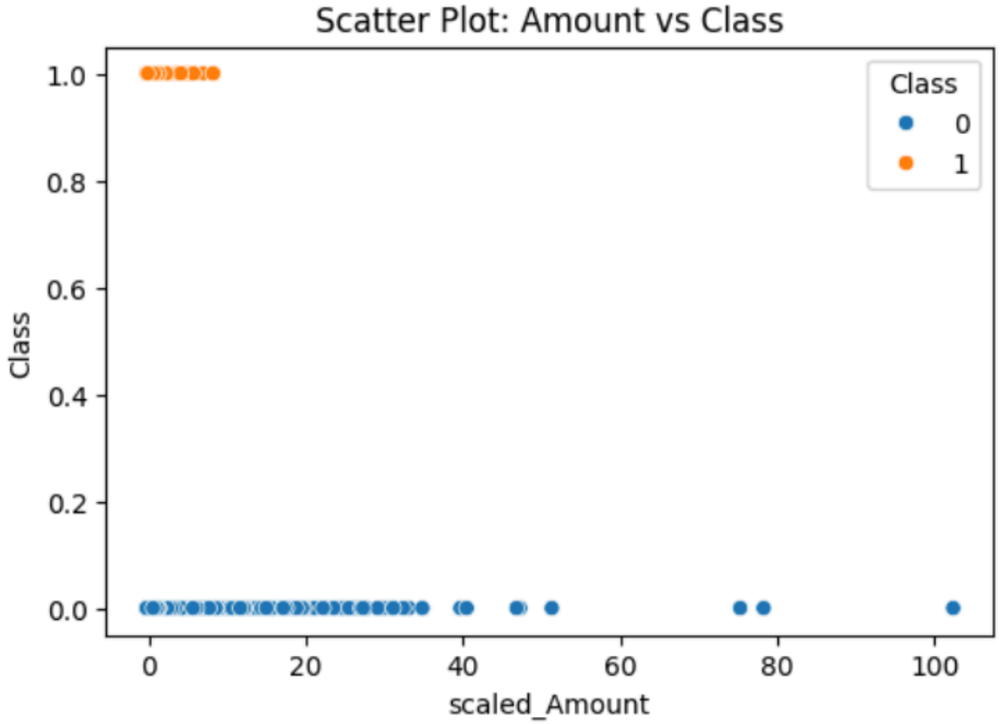


FEATURE REQUIREMENTS

```
fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(10,6))

#Normal transactions
ax1.hist(data.Time_Hr[data.Class==0], bins=48, color='g', alpha=0.5)
ax1.set_title('not fraud')

# Fraud transactions
ax2.hist(data.Time_Hr[data.Class==1], bins=48, color='r', alpha=0.5)
ax2.set_title('Fraud')

plt.xlabel('Time (hrs)')
plt.ylabel('# Transactions')
plt.show()
```




```
plt.figure(figsize=(6,4))
sns.scatterplot(data=data, x='scaled_Amount', y='Class', hue='Class')
plt.title("Scatter Plot: Amount vs Class")
plt.show()
```



BASELINE MODEL

- A baseline model was built using the original imbalanced dataset without applying any sampling techniques.

Purpose of the baseline model:

- Serve as a reference point
- Evaluate how imbalance affects model performance
- Compare with advanced techniques later.

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
model = LogisticRegression(
    max_iter=1000,
    random_state=42
)

model.fit(X_train_scaled, y_train)
```

```
LogisticRegression
▶ Parameters
```

```
y_pred = model.predict(X_test_scaled)
```

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[56851    13]
 [   35    63]]
```

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.83      0.64      0.72        98

    accuracy                           1.00     56962
   macro avg       0.91      0.82      0.86     56962
weighted avg       1.00      1.00      1.00     56962
```

TRAIN - TEST SPLIT

- Split dataset into:
  - 80% training
  - 20% testing

```python
#train-test split
X = data.drop('Class', axis=1)
y = data['Class']

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

```python
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LogisticRegression(
    max_iter=1000,
    random_state=42
)
```

```python
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)
```

```python
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     55038
           1       0.85      0.58      0.69        95

    accuracy                           1.00     55133
   macro avg       0.92      0.79      0.84     55133
weighted avg       1.00      1.00      1.00     55133
```

IMBALANCED DATA

- Handle imbalanced data using:
  - SMOTE
  - Oversampling
  - Class weights

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X_train, y_train)
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

```
rf = RandomForestClassifier(
    n_estimators=200,
    class_weight='balanced',
    random_state=42,
    n_jobs=-1
)

rf.fit(X_train, y_train)
```

RandomForestClassifier

▶ Parameters

```
#Threshold
y_prob = rf.predict_proba(X_test)[:,1]
y_pred = (y_prob > 0.3).astype(int)
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nROC AUC:")
print(roc_auc_score(y_test, y_prob))
```

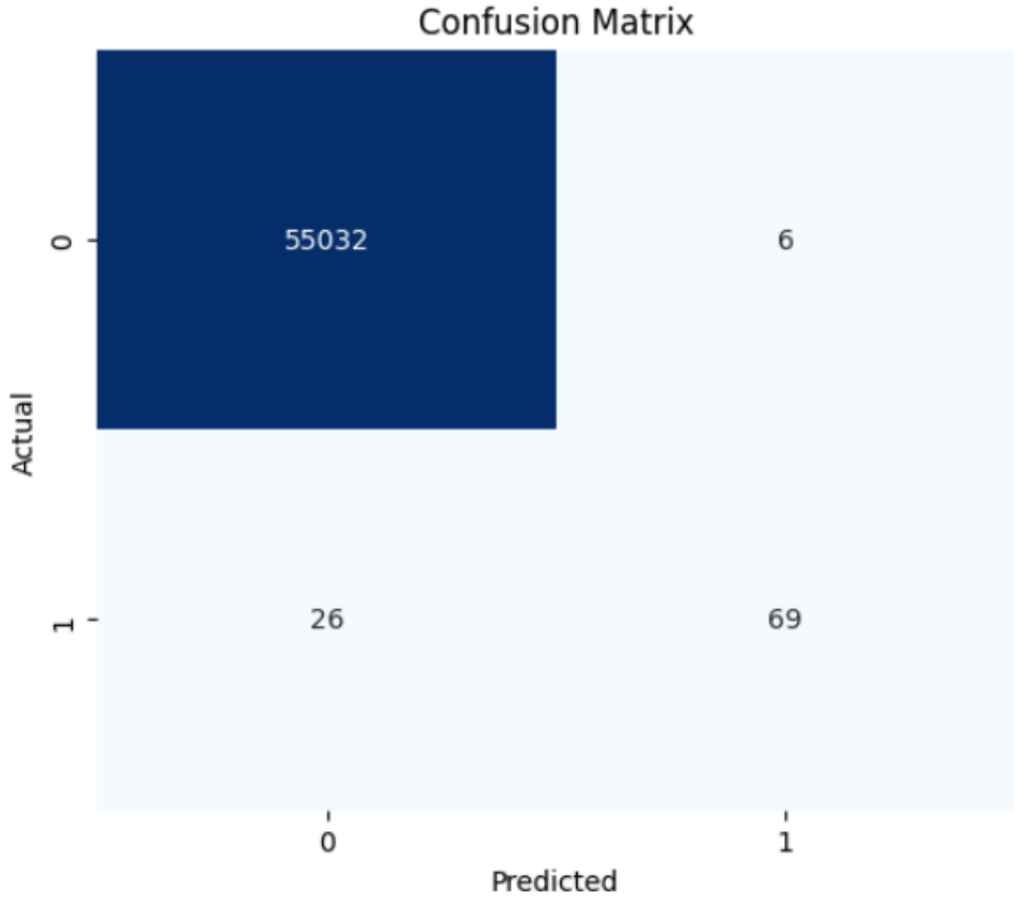
```
Confusion Matrix:
[[55029     9]
 [   19    76]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     55038
           1       0.89      0.80      0.84        95

    accuracy                           1.00     55133
   macro avg       0.95      0.90      0.92     55133
weighted avg       1.00      1.00      1.00     55133


ROC AUC:
0.9394031109606569
```

EVALUATION

CONFUSION MATRIX

- The model correctly classified 55,032 normal transactions.
- It successfully detected 69 fraudulent transactions.
- Only 6 normal transactions were wrongly classified as fraud.
- The model missed 26 fraud cases, which is a critical concern.
- Overall, the model performs well but fraud recall can be improved.

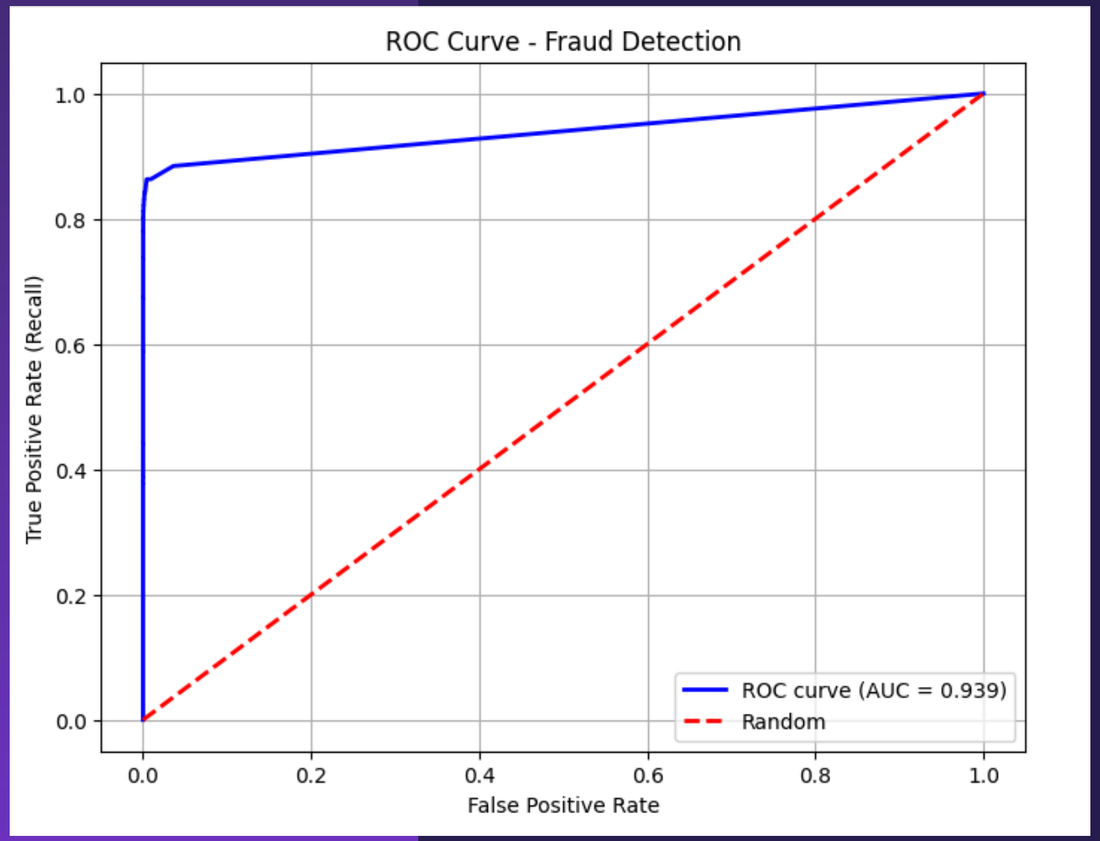
```python
# Compute confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



EVALUATION

ROC CURVE

```python
#ROC Curve
from sklearn.metrics import roc_curve, auc, precision_recall_curve, average_precision_score

fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.3f})')
plt.plot([0,1], [0,1], color='red', lw=2, linestyle='--', label= 'Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate (Recall)')
plt.title('ROC Curve - Fraud Detection')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



KNN CLASSIFIER

```python
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import RobustScaler

# Define KNN
knn = KNeighborsClassifier()

# Grid Parameters
param_grid = {
    'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance'],
    'metric': ['minkowski']
}

# Grid Search use Recall
grid_knn = GridSearchCV(
    estimator=knn,
    param_grid=param_grid,
    scoring='recall',
    cv=3,
    verbose=2,
    n_jobs=-1
)

# Scale Data
scaler = RobustScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# trining Grid Search
grid_knn.fit(X_train_scaled, y_train)

# Best model
best_knn = grid_knn.best_estimator_
print("Best Parameters:", grid_knn.best_params_)
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
Best Parameters: {'metric': 'minkowski', 'n_neighbors': 3, 'weights': 'distance'}
```

```python
y_prob_knn = best_knn.predict_proba(X_test_scaled)[:,1]
```

```python
from sklearn.metrics import precision_recall_curve, average_precision_score, confusion_matrix, classification_repo
precision, recall, thresholds = precision_recall_curve(y_test, y_prob_knn)
f1_scores = 2 * (precision * recall) / (precision + recall + 1e-6)
best_idx = np.argmax(f1_scores)
best_threshold = thresholds[best_idx]
print(f"Best Threshold: {best_threshold:.3f}")
```

```
Best Threshold: 0.402
```

```python
y_pred_knn = (y_prob_knn > best_threshold).astype(int)

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_knn))

print("\nClassification Report:")
print(classification_report(y_test, y_pred_knn))

print("ROC AUC:", roc_auc_score(y_test, y_prob_knn))
```

```
Confusion Matrix:
[[55030     8]
 [   25    70]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     55038
           1       0.90      0.74      0.81        95

    accuracy                           1.00     55133
   macro avg       0.95      0.87      0.90     55133
weighted avg       1.00      1.00      1.00     55133

ROC AUC: 0.8998792222024592
```

LOGISTIC REGRESSION

```python
# Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LogisticRegression(max_iter=1000, random_state=42)
model.fit(X_train_scaled, y_train)  # ✅ تدريب النموذج
```

```
▾ LogisticRegression
▶ Parameters
```

```python
threshold = 0.3
y_prob_rf = best_rf.predict_proba(X_test)[:,1]
y_pred_rf = (y_prob_rf > threshold).astype(int)
```

```python
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Calculate ROC AUC
roc_auc = roc_auc_score(y_test, y_prob)
print("\nROC AUC:", roc_auc)
```

```
Confusion Matrix:
[[55022    16]
 [   21    74]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     55038
           1       0.82      0.78      0.80        95

    accuracy                           1.00     55133
   macro avg       0.91      0.89      0.90     55133
weighted avg       1.00      1.00      1.00     55133


ROC AUC: 0.9800425925819674
```

GRID SEARCH

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 100}
Confusion Matrix:
 [[55022    16]
 [   21    74]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     55038
           1       0.82      0.78      0.80        95

    accuracy                           1.00     55133
   macro avg       0.91      0.89      0.90     55133
weighted avg       1.00      1.00      1.00     55133

ROC AUC: 0.9800425925819674
```

```python
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Define the Random Forest
rf = RandomForestClassifier(
    class_weight='balanced',  # important for fraud detection
    random_state=42,
    n_jobs=-1
)

# Define Hyperparameter Grid (fast)
param_grid = {
    'n_estimators': [100, 200],      # number of trees
    'max_depth': [10, 20],           # depth of trees
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Grid Search (optimize Recall)
grid = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    scoring='recall',   # important for detecting fraud
    cv=3,
    verbose=2,
    n_jobs=-1
)

# Train
grid.fit(X_train, y_train)

#  Best model
best_rf = grid.best_estimator_
print("Best Parameters:", grid.best_params_)

# Evaluate on test set
y_prob = best_rf.predict_proba(X_test)[:,1]
y_pred = (y_prob > 0.5).astype(int)  # default threshold 0.5

from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("ROC AUC:", roc_auc_score(y_test, y_prob))
```

CONCLUSION

The project demonstrates the importance of balancing techniques in fraud detection for imbalanced datasets. The Voting Classifier with cost-sensitive learning proved to be the most effective approach, achieving:

- High precision and accuracy in identifying fraudulent transactions.
- Robust performance across various metrics (F1 Macro: 0.938).

However, the recall metric remains an area for improvement, indicating that some fraud cases remain undetected. This highlights the complexity of detecting sophisticated fraud patterns, which may closely mimic legitimate transactions.
The study underscores the need for a careful trade-off between precision and recall to ensure both high detection rates and minimal false positives.

FUTURE DIRECTIONS

1. **Model Specialization**:
   - Develop models tailored to specific fraud types.
   - Integrate semi-supervised or unsupervised methods, such as autoencoders, to detect anomalies.
2. **Enhancing Interpretability**:
   - Improve feature transparency for clearer insights into fraud detection.
3. **Robustness Improvements**:
   - Expand experiments with different data augmentation and preprocessing techniques.
   - Investigate adversarial training to counter sophisticated fraud techniques.

Thank you