import libraries

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

load file

```
df=pd.read_csv('/content/creditcard.csv.zip')
```

```
df.shape
```

```
(284807, 31)
```

```
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.01
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.22
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.24
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.10
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.00

5 rows × 31 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
```

```
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

```
df.isnull().sum().sum()
```

```
np.int64(0)
```

```
df.describe()
```

	Time	V1	V2	V3	V4	V5	V6
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01

8 rows × 31 columns

Statistical summary of 'Amount' and 'Time' by 'Class'

```
print("\nStatistical summary of 'Amount' by Class:")
print(df.groupby('Class')['Amount'].describe())

print("\nStatistical summary of 'Time' by Class:")
print(df.groupby('Class')['Time'].describe())
```

```

Statistical summary of 'Amount' by Class:
          count        mean         std  min   25%    50%     75%       max
Class
0      284315.0   88.291022  250.105092  0.0  5.65  22.00   77.05  25691.16
1         492.0  122.211321  256.683288  0.0  1.00   9.25  105.89   2125.87

Statistical summary of 'Time' by Class:
          count          mean           std    min      25%      50%  \
Class
0      284315.0  94838.202258  47484.015786    0.0  54230.0  84711.0
1         492.0  80746.806911  47835.365138  406.0  41241.5  75568.5

            75%       max
Class
0      139333.0  172792.0
1      128483.0  170348.0
```

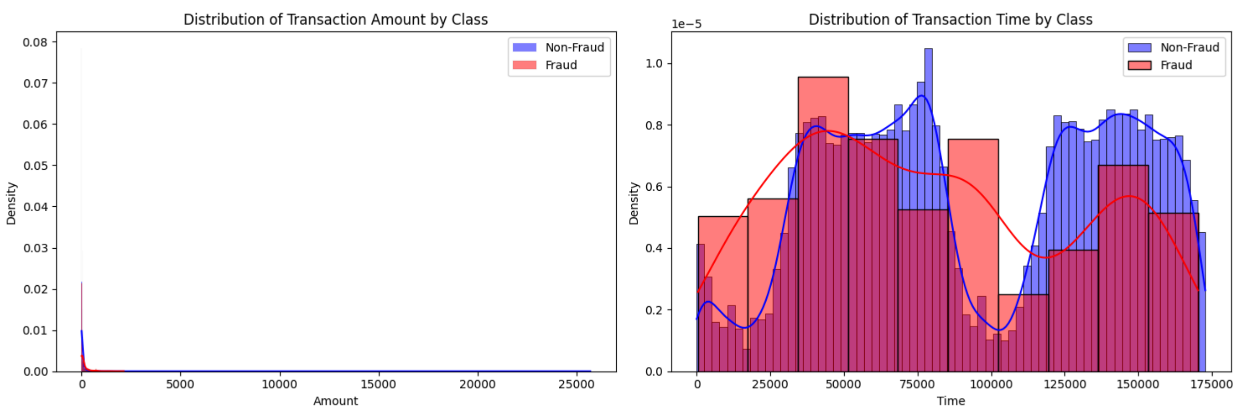
Check the distribution of the target variable ('Class')

```
print(df['Class'].value_counts())
print(f"\nPercentage of fraudulent transactions: {round((df['Class'].value_counts()[1] / df.shape[0]) * 10(

sns.countplot(x='Class', data=df)
plt.title('Distribution of Transaction Classes')
plt.show()
```

```
Class
0    284315
1       492
Name: count, dtype: int64

Percentage of fraudulent transactions: 0.1727%
```

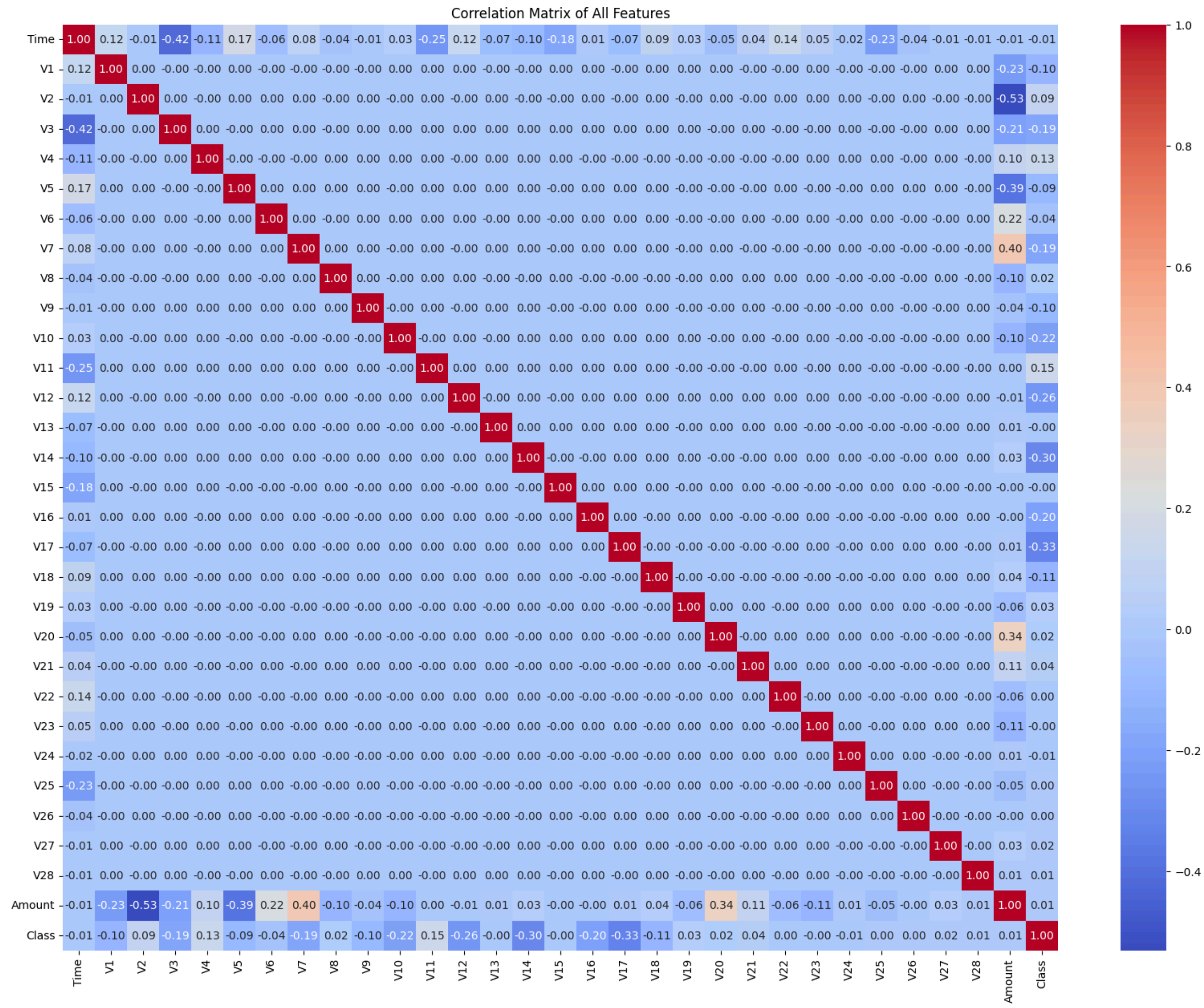
Visualize 'Amount' and 'Time' for both classes

```
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

# Distribution of 'Amount' by 'Class'
sns.histplot(df[df['Class'] == 0]['Amount'], ax=axes[0], color='blue', label='Non-Fraud', kde=True, stat='c
sns.histplot(df[df['Class'] == 1]['Amount'], ax=axes[0], color='red', label='Fraud', kde=True, stat='densit
axes[0].set_title('Distribution of Transaction Amount by Class')
axes[0].legend()

# Distribution of 'Time' by 'Class'
sns.histplot(df[df['Class'] == 0]['Time'], ax=axes[1], color='blue', label='Non-Fraud', kde=True, stat='der
sns.histplot(df[df['Class'] == 1]['Time'], ax=axes[1], color='red', label='Fraud', kde=True, stat='density'
axes[1].set_title('Distribution of Transaction Time by Class')
axes[1].legend()

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



Correlation Analysis

```
# Calculate the correlation matrix
correlation_matrix = df.corr()

# Plot the heatmap
plt.figure(figsize=(20, 15))
sns.heatmap(correlation_matrix, cmap='coolwarm', annot=True, fmt=".2f")
plt.title('Correlation Matrix of All Features')
plt.show()
```



Let's also look specifically at the correlation of each feature with the 'Class' (target) variable. This will highlight which features might be most important for predicting fraud.

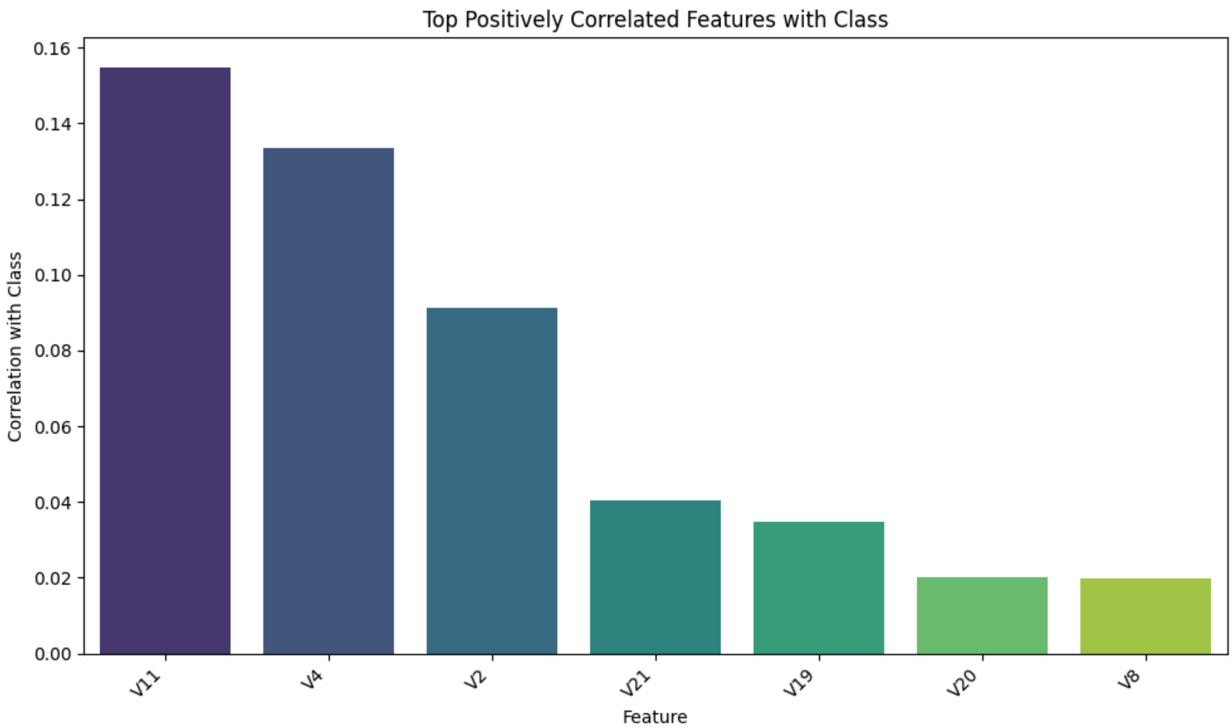
```
# Get correlations with the 'Class' variable
class_correlations = correlation_matrix['Class'].sort_values(ascending=False)

print("\nCorrelation of Features with 'Class':")
print(class_correlations)
```

```
Correlation of Features with 'Class':
Class     1.000000
V11       0.154876
V4        0.133447
V2        0.091289
V21       0.040413
V19       0.034783
V20       0.020090
V8        0.019875
V27       0.017580
V28       0.009536
Amount    0.005632
V26       0.004455
```

```
V25      0.003308
V22      0.000805
V23     -0.002685
V15     -0.004223
V13     -0.004570
V24     -0.007221
Time    -0.012323
V6      -0.043643
V5      -0.094974
V9      -0.097733
V1      -0.101347
V18     -0.111485
V7      -0.187257
V3      -0.192961
V16     -0.196539
V10     -0.216883
V12     -0.260593
V14     -0.302544
V17     -0.326481
Name: Class, dtype: float64
```

Visualize Top Positively Correlated Features with 'Class'

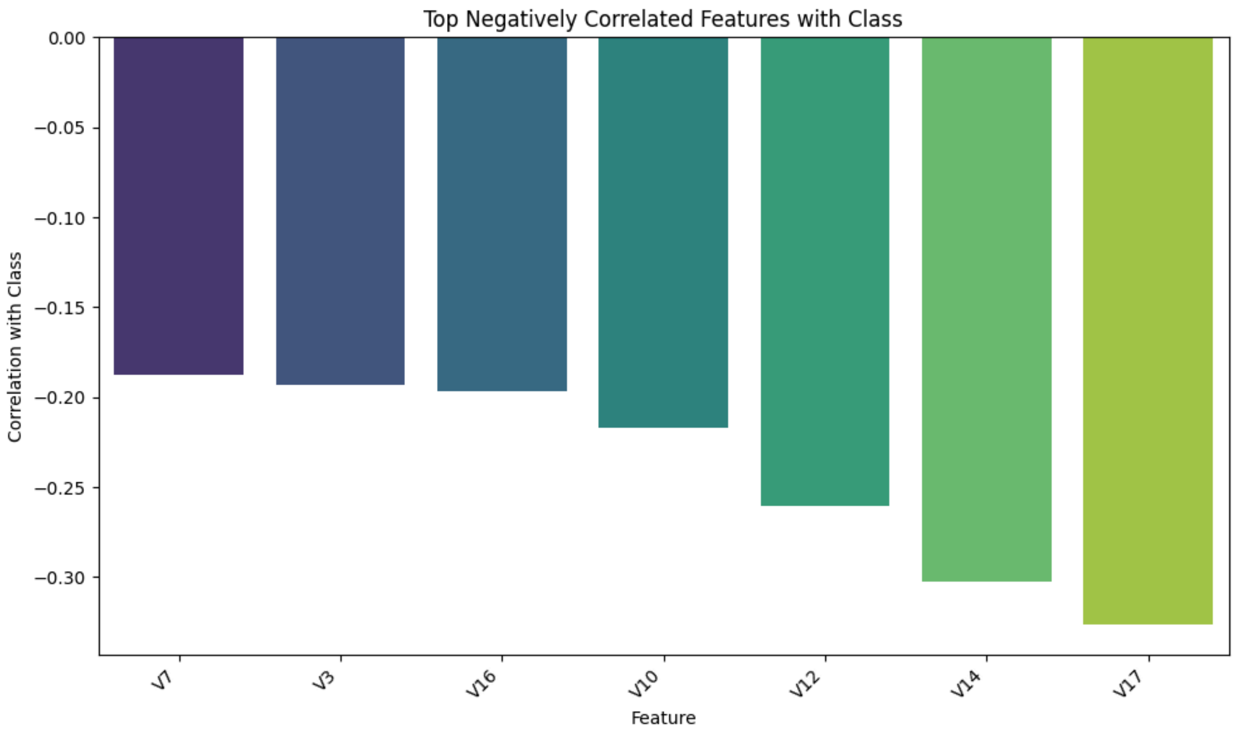
```python
# Filter out 'Class' itself and get the top positively correlated features
top_positive_correlations = class_correlations[1:].head(7)

plt.figure(figsize=(10, 6))
sns.barplot(x=top_positive_correlations.index, y=top_positive_correlations.values, hue=top_positive_correla
plt.title('Top Positively Correlated Features with Class')
plt.xlabel('Feature')
plt.ylabel('Correlation with Class')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



Visualize Top Negatively Correlated Features with 'Class'

```python
# Filter out 'Class' itself and get the top negatively correlated features (sort ascending and take the hea
top_negative_correlations = class_correlations[class_correlations.index != 'Class'].tail(7)

plt.figure(figsize=(10, 6))
sns.barplot(x=top_negative_correlations.index, y=top_negative_correlations.values, hue=top_negative_correla
```

```
plt.title('Top Negatively Correlated Features with Class')
plt.xlabel('Feature')
plt.ylabel('Correlation with Class')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```




```
# Separate features (X) and target (y)
X = df.drop('Class', axis=1)
y = df['Class']

print(f"Original target variable distribution:\n{y.value_counts()}")
```

```
Original target variable distribution:
Class
0    284315
1       492
Name: count, dtype: int64
```

Splitting Data into Training and Testing Sets

```
from sklearn.model_selection import train_test_split

# Split the resampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print(f"Shape of X_train: {X_train.shape}")
print(f"Shape of X_test: {X_test.shape}")
print(f"Shape of y_train: {y_train.shape}")
print(f"Shape of y_test: {y_test.shape}")

print(f"\nDistribution of y_train:\n{y_train.value_counts(normalize=True)}")
print(f"\nDistribution of y_test:\n{y_test.value_counts(normalize=True)}")
```

```
Shape of X_train: (227845, 30)
Shape of X_test: (56962, 30)
Shape of y_train: (227845,)
Shape of y_test: (56962,)

Distribution of y_train:
Class
0    0.998271
1    0.001729
```

```
Name: proportion, dtype: float64

Distribution of y_test:
Class
0    0.99828
1    0.00172
Name: proportion, dtype: float64
```

Standard Scaling

```
from sklearn.preprocessing import StandardScaler

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the scaler on the training data and transform both training and test data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("X_train_scaled shape:", X_train_scaled.shape)
print("X_test_scaled shape:", X_test_scaled.shape)

# Display the first few rows of the scaled training data (as a DataFrame for readability)
display(pd.DataFrame(X_train_scaled, columns=X_train.columns).head())
```

```
X_train_scaled shape: (227845, 30)
X_test_scaled shape: (56962, 30)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
0	1.411588	0.993379	-0.456037	-0.894052	-0.467284	1.089217	3.024383	-1.194852	0.957057	1.281376	...
1	0.623141	1.038507	-0.029349	-2.018302	0.175133	2.133506	2.478840	-0.001832	0.566704	0.041121	...
2	-1.130680	-0.506766	0.366065	0.470114	-0.700918	-0.598748	1.470411	-1.786684	-4.227592	0.000064	...
3	0.794699	1.166419	-0.909447	-0.493095	-1.178149	-1.010692	-0.262292	-1.153123	0.008765	-1.019866	...
4	-0.748102	-0.229485	-0.613041	0.076742	-2.440089	0.518711	-0.109914	0.407186	-0.095161	-0.041449	...

5 rows × 30 columns

Task

Implement and evaluate Logistic Regression, Random Forest, and XGBoost models on the provided scaled training and test data (X_train_scaled, y_train, X_test_scaled, y_test) for credit card fraud detection. For each model, calculate and display the classification report, confusion matrix, and AUC-ROC score, and plot the confusion matrix. Finally, summarize the performance of all three models and discuss which model is most suitable for this imbalanced dataset.

Implement Logistic Regression

```
from sklearn.linear_model import LogisticRegression

# Initialize the Logistic Regression model
model = LogisticRegression(solver='liblinear', random_state=42)

# Train the model
model.fit(X_train_scaled, y_train)

print("Logistic Regression model trained successfully.")
```

```
Logistic Regression model trained successfully.
```

```
y_pred_lr = model.predict(X_test_scaled)
y_pred_proba_lr = model.predict_proba(X_test_scaled)[:, 1]

print("Predictions made successfully.")
```

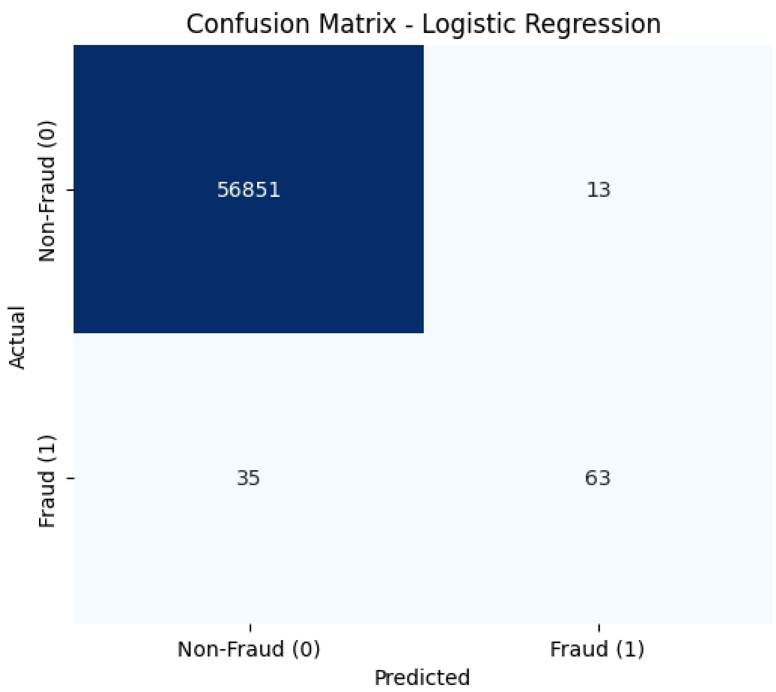
```
Predictions made successfully.
```

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

# Evaluate Logistic Regression model
print("Classification Report for Logistic Regression:")
print(classification_report(y_test, y_pred_lr))
```

```
Classification Report for Logistic Regression:
              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
```

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

# Confusion Matrix for Logistic Regression
conf_matrix_lr = confusion_matrix(y_test, y_pred_lr)
print("\nConfusion Matrix for Logistic Regression:")
print(conf_matrix_lr)

plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix_lr, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Non-Fraud (0)', 'Fraud (1)'], yticklabels=['Non-Fraud (0)', 'Fraud (1)'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Logistic Regression')
plt.show()
```

```
Confusion Matrix for Logistic Regression:
[[56851    13]
 [   35    63]]
```




```
from sklearn.metrics import roc_auc_score

# AUC-ROC Score for Logistic Regression
roc_auc_lr = roc_auc_score(y_test, y_pred_proba_lr)
print(f"\nAUC-ROC Score for Logistic Regression: {roc_auc_lr:.4f}")
```

```
AUC-ROC Score for Logistic Regression: 0.9575
```

Implement Random Forest

```
from sklearn.ensemble import RandomForestClassifier

# Initialize the Random Forest model
# Using class_weight='balanced' to handle imbalanced dataset
model_rf = RandomForestClassifier(random_state=42, class_weight='balanced')

# Train the model
model_rf.fit(X_train_scaled, y_train)

print("Random Forest Classifier model trained successfully.")
```

```
Random Forest Classifier model trained successfully.
```

```
y_pred_rf = model_rf.predict(X_test_scaled)
y_pred_proba_rf = model_rf.predict_proba(X_test_scaled)[:, 1]

print("Predictions made successfully with Random Forest Classifier.")
```

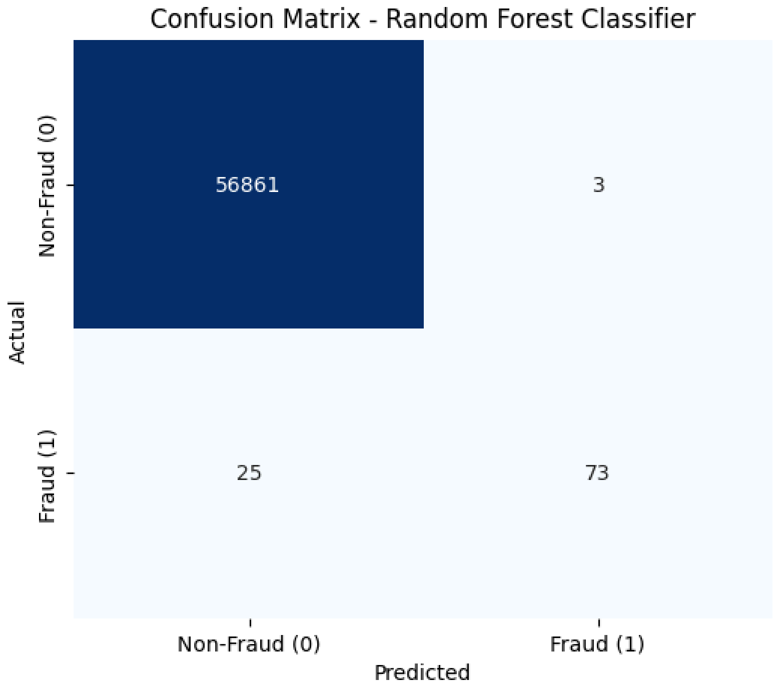
```
Predictions made successfully with Random Forest Classifier.
```

```
from sklearn.metrics import classification_report

# Evaluate Random Forest model
print("Classification Report for Random Forest Classifier:")
print(classification_report(y_test, y_pred_rf))
```

```
Classification Report for Random Forest Classifier:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.96      0.74      0.84        98

    accuracy                           1.00     56962
   macro avg       0.98      0.87      0.92     56962
weighted avg       1.00      1.00      1.00     56962
```

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Confusion Matrix for Random Forest
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
print("\nConfusion Matrix for Random Forest Classifier:")
print(conf_matrix_rf)

plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix_rf, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Non-Fraud (0)', 'Fraud (1)'], yticklabels=['Non-Fraud (0)', 'Fraud (1)'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Random Forest Classifier')
plt.show()
```

```
Confusion Matrix for Random Forest Classifier:
[[56861     3]
 [   25    73]]
```




```
from sklearn.metrics import roc_auc_score

# AUC-ROC Score for Random Forest
roc_auc_rf = roc_auc_score(y_test, y_pred_proba_rf)
print(f"\nAUC-ROC Score for Random Forest Classifier: {roc_auc_rf:.4f}")
```

```
AUC-ROC Score for Random Forest Classifier: 0.9529
```

Implement XGBoost

```
from xgboost import XGBClassifier

# Calculate the scale_pos_weight for handling imbalanced classes
scale_pos_weight_value = sum(y_train == 0) / sum(y_train == 1)

# Initialize the XGBoost Classifier model
# Using scale_pos_weight to handle imbalanced dataset
model_xgb = XGBClassifier(objective='binary:logistic', eval_metric='logloss', random_state=42, scale_pos_we

# Train the model
model_xgb.fit(X_train_scaled, y_train)

print("XGBoost Classifier model trained successfully.")
```

```
XGBoost Classifier model trained successfully.
```

```
y_pred_xgb = model_xgb.predict(X_test_scaled)
y_pred_proba_xgb = model_xgb.predict_proba(X_test_scaled)[:, 1]

print("Predictions made successfully with XGBoost Classifier.")
```

```
Predictions made successfully with XGBoost Classifier.
```

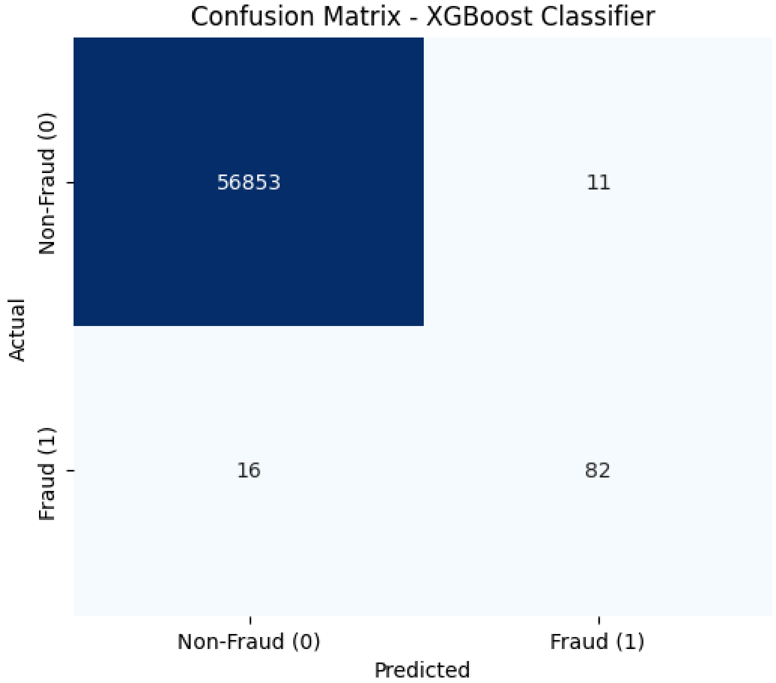
```
from sklearn.metrics import classification_report

# Evaluate XGBoost model
print("Classification Report for XGBoost Classifier:")
print(classification_report(y_test, y_pred_xgb))
```

```
Classification Report for XGBoost Classifier:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.88      0.84      0.86        98

    accuracy                           1.00     56962
   macro avg       0.94      0.92      0.93     56962
weighted avg       1.00      1.00      1.00     56962
```

Evaluate XGBoost

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Confusion Matrix for XGBoost
conf_matrix_xgb = confusion_matrix(y_test, y_pred_xgb)
print("\nConfusion Matrix for XGBoost Classifier:")
print(conf_matrix_xgb)

plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix_xgb, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Non-Fraud (0)', 'Fraud (1)'], yticklabels=['Non-Fraud (0)', 'Fraud (1)'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - XGBoost Classifier')
plt.show()
```

```
Confusion Matrix for XGBoost Classifier:
[[56853    11]
 [   16    82]]
```




```
from sklearn.metrics import roc_auc_score

# AUC-ROC Score for XGBoost
roc_auc_xgb = roc_auc_score(y_test, y_pred_proba_xgb)
print(f"\nAUC-ROC Score for XGBoost Classifier: {roc_auc_xgb:.4f}")
```

```
AUC-ROC Score for XGBoost Classifier: 0.9682
```

Model Performance Summary and Discussion

We implemented and evaluated three classification models: Logistic Regression, Random Forest, and XGBoost, on a credit card fraud detection dataset. The dataset is highly imbalanced, with a very small percentage of fraudulent transactions (0.1727%). Given this imbalance, metrics such as Recall, Precision, F1-score, and AUC-ROC are crucial for evaluating performance, as accuracy can be misleading.

Evaluation Results:

1. Logistic Regression

- **Classification Report:**
  - Precision (Fraud): 0.83
  - Recall (Fraud): 0.64
  - F1-score (Fraud): 0.72
- **Confusion Matrix:**
  - True Negatives (0,0): 56851
  - False Positives (0,1): 13
  - False Negatives (1,0): 35
  - True Positives (1,1): 63
- **AUC-ROC Score:** 0.9575

2. Random Forest Classifier

- **Classification Report:**
  - Precision (Fraud): 0.96
  - Recall (Fraud): 0.74
  - F1-score (Fraud): 0.84
- **Confusion Matrix:**
  - True Negatives (0,0): 56861
  - False Positives (0,1): 3
  - False Negatives (1,0): 25
  - True Positives (1,1): 73
- **AUC-ROC Score:** 0.9529

3. XGBoost Classifier

- **Classification Report:**
  - Precision (Fraud): 0.88
  - Recall (Fraud): 0.84
  - F1-score (Fraud): 0.86
- **Confusion Matrix:**
  - True Negatives (0,0): 56853
  - False Positives (0,1): 11
  - False Negatives (1,0): 16
  - True Positives (1,1): 82
- **AUC-ROC Score:** 0.9682

Discussion:

- **Fraud Detection Goal:** In credit card fraud detection, the primary goal is often to minimize **False Negatives** (fraudulent transactions classified as non-fraudulent), even if it means a slight increase in False Positives (legitimate transactions flagged as fraud). High **Recall** for the minority class (fraud) is therefore very important, as is a good **AUC-ROC score**.

- **Logistic Regression:** Performed reasonably well with an AUC-ROC of 0.9575. Its recall for fraud was 0.64, meaning it missed 35 fraudulent transactions. While its precision was high, its recall was the lowest among the three models.
- **Random Forest:** Showed strong precision (0.96) for fraud, indicating that when it predicted fraud, it was usually correct. It also had a decent recall of 0.74, correctly identifying 73 out of 98 fraud cases, with only 3 false positives. The AUC-ROC was 0.9529.
- **XGBoost:** Achieved the highest **recall (0.84)**, meaning it identified the most fraudulent transactions (82 out of 98) among the three models. It also had the highest **AUC-ROC score (0.9682)**, indicating its superior ability to distinguish between fraud and non-fraud across different thresholds. Its precision (0.88) and F1-score (0.86) were also very competitive. XGBoost successfully leveraged `scale_pos_weight` to address the class imbalance, leading to a better balance between precision and recall for the minority class.

Conclusion:

Considering the importance of identifying as many fraudulent transactions as possible (high recall) and its overall discriminative power (high AUC-ROC), the **XGBoost Classifier** appears to be the most suitable model for this imbalanced credit card fraud detection dataset. It demonstrated the best balance of correctly identifying fraud cases while maintaining good overall performance.

Define Hyperparameter Grid

Subtask:

Define a dictionary for the hyperparameter grid for XGBoost, including parameters like `n_estimators`, `max_depth`, `learning_rate`, and `subsample`.

```
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5],
    'learning_rate': [0.01, 0.1],
    'subsample': [0.7, 1.0]
}

print("XGBoost Hyperparameter Grid:")
print(param_grid)
```

```
XGBoost Hyperparameter Grid:
{'n_estimators': [100, 200], 'max_depth': [3, 5], 'learning_rate': [0.01, 0.1], 'subsample': [0.7, 1.0]}
```

Perform GridSearchCV

Subtask:

Initialize `GridSearchCV` with the XGBoost model, the defined hyperparameter grid, appropriate scoring metrics (e.g., 'roc_auc', 'f1'), and cross-validation settings. Fit `GridSearchCV` on the scaled training data (X_train_scaled, y_train) to find the best parameters.

```
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier

# Initialize an XGBClassifier model
# Using scale_pos_weight to handle imbalanced dataset
model_xgb_tuned = XGBClassifier(objective='binary:logistic', eval_metric='logloss', random_state=42, scale_p

# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=model_xgb_tuned,
    param_grid=param_grid,
    scoring=['roc_auc', 'f1'],
    cv=3,
    verbose=2,
```

```
    n_jobs=-1,
    refit='roc_auc'  # Refit the model with the best parameters based on ROC AUC
)

print("Starting GridSearchCV...")
# Fit GridSearchCV to the scaled training data
grid_search.fit(X_train_scaled, y_train)

print("GridSearchCV completed.")

# Print the best hyperparameters and their corresponding ROC AUC score
print("\nBest hyperparameters found:", grid_search.best_params_)
print("Best ROC AUC score:", grid_search.best_score_)
```

```
Starting GridSearchCV...
Fitting 3 folds for each of 16 candidates, totalling 48 fits
GridSearchCV completed.

Best hyperparameters found: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100, 'subsample': 0.7}
Best ROC AUC score: 0.9808841718093223
```

Train Tuned XGBoost Model

Subtask:

Train a new XGBoost model using the best hyperparameters found by `GridSearchCV`.

```
from xgboost import XGBClassifier

# Initialize a new XGBClassifier model using the best hyperparameters
# and other fixed parameters for handling imbalanced dataset and reproducibility
model_xgb_best = XGBClassifier(
    objective='binary:logistic',
    eval_metric='logloss',
    random_state=42,
    scale_pos_weight=scale_pos_weight_value,
    **grid_search.best_params_
)

# Train the model
model_xgb_best.fit(X_train_scaled, y_train)

print("Tuned XGBoost Classifier model trained successfully.")
```

```
Tuned XGBoost Classifier model trained successfully.
```

```
y_pred_xgb_tuned = model_xgb_best.predict(X_test_scaled)
y_pred_proba_xgb_tuned = model_xgb_best.predict_proba(X_test_scaled)[:, 1]

print("Predictions made successfully with Tuned XGBoost Classifier.")
```

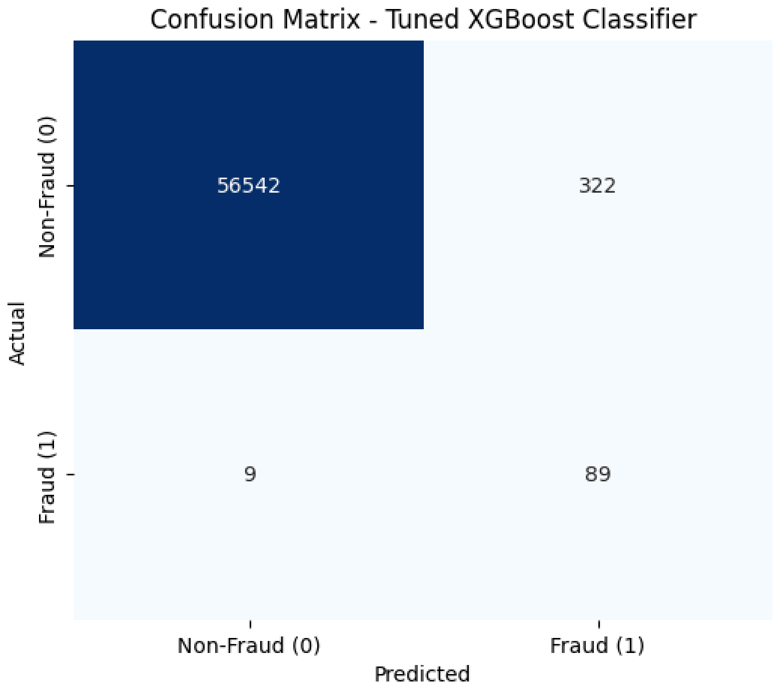
```
Predictions made successfully with Tuned XGBoost Classifier.
```

```
from sklearn.metrics import classification_report

# Evaluate Tuned XGBoost model
print("Classification Report for Tuned XGBoost Classifier:")
print(classification_report(y_test, y_pred_xgb_tuned))
```

```
Classification Report for Tuned XGBoost Classifier:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     56864
           1       0.22      0.91      0.35        98

    accuracy                           0.99     56962
   macro avg       0.61      0.95      0.67     56962
weighted avg       1.00      0.99      1.00     56962
```

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Confusion Matrix for Tuned XGBoost
conf_matrix_xgb_tuned = confusion_matrix(y_test, y_pred_xgb_tuned)
print("\nConfusion Matrix for Tuned XGBoost Classifier:")
print(conf_matrix_xgb_tuned)

plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix_xgb_tuned, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Non-Fraud (0)', 'Fraud (1)'], yticklabels=['Non-Fraud (0)', 'Fraud (1)'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Tuned XGBoost Classifier')
plt.show()
```

```
Confusion Matrix for Tuned XGBoost Classifier:
[[56542   322]
 [    9    89]]
```




```
from sklearn.metrics import roc_auc_score
```