Credit Card Fraud Detection
Maryah Austria

This project focuses on building and evaluating machine learning models for detecting fraudulent transactions in credit card data. The dataset used for this task contains features from anonymized credit card transactions, with a binary target variable ('Class') indicating whether the transaction is fraudulent (1) or not (0). The main objective of the project is to develop robust models that can accurately classify transactions as either fraudulent or non-fraudulent, using techniques such as SMOTE for imbalance handling, Isolation Forest for anomaly detection, and models like Logistic Regression and XGBoost for classification.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_s
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# For imbalance handling
from imblearn.over_sampling import SMOTE

# For anomaly detection
from sklearn.ensemble import IsolationForest
import xgboost as xgb
```

Data Preprocessing and Feature Description
In order to protect customer privacy and ensure compliance with data protection regulations (such as GDPR), the dataset has undergone anonymization. The original features such as customer ID and transaction details have been replaced with the following variables:

V1, V2, V3, ..., V28: These are anonymized features derived from the original data. They represent various transformation of the raw features but are not directly interpretable.

Time: This feature represents the number of seconds elapsed between the current transaction and the first transaction in the dataset. It provides a temporal dimension to the transactions.

Amount: This feature represents the monetary amount of the transaction.

The 'Class' feature is the target variable, where 1 denotes a fraudulent transaction and 0 represents a non-fraudulent transaction.

Data Loading and Inspection
The dataset was loaded using pandas and basic exploratory data analysis (EDA) was conducted using .info() and .describe() methods. Visualizations were generated to check for

In [3]:
```python
# Load the dataset
df = pd.read_csv("/content/creditcard.csv")


# Display basic information
df.info()
df.describe()
```

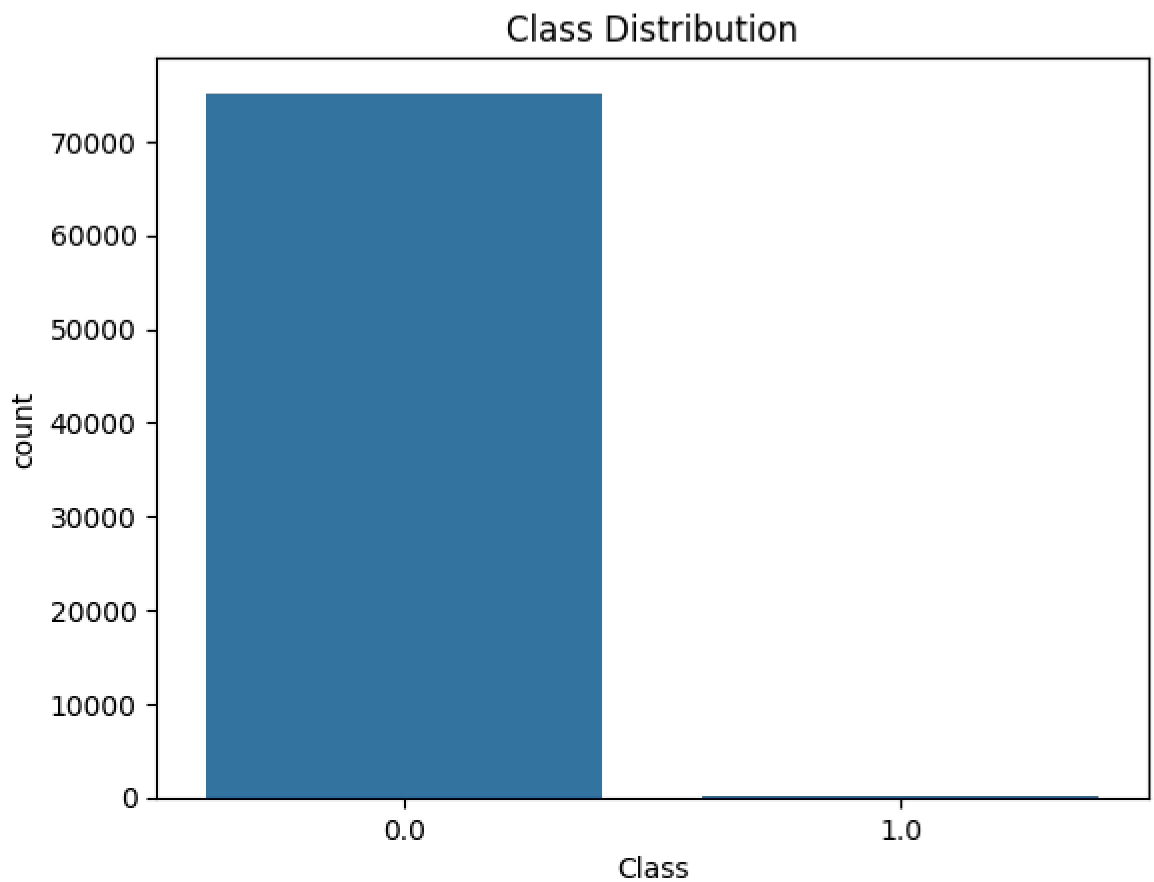
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 75357 entries, 0 to 75356
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Time    75357 non-null  int64  
 1   V1      75357 non-null  float64
 2   V2      75357 non-null  float64
 3   V3      75357 non-null  float64
 4   V4      75357 non-null  float64
 5   V5      75357 non-null  float64
 6   V6      75357 non-null  float64
 7   V7      75357 non-null  float64
 8   V8      75357 non-null  float64
 9   V9      75357 non-null  float64
 10  V10     75357 non-null  float64
 11  V11     75357 non-null  float64
 12  V12     75357 non-null  float64
 13  V13     75357 non-null  float64
 14  V14     75357 non-null  float64
 15  V15     75357 non-null  float64
 16  V16     75357 non-null  float64
 17  V17     75357 non-null  float64
 18  V18     75357 non-null  float64
 19  V19     75357 non-null  float64
 20  V20     75357 non-null  float64
 21  V21     75357 non-null  float64
 22  V22     75357 non-null  float64
 23  V23     75357 non-null  float64
 24  V24     75357 non-null  float64
 25  V25     75357 non-null  float64
 26  V26     75357 non-null  float64
 27  V27     75356 non-null  float64
 28  V28     75356 non-null  float64
 29  Amount  75356 non-null  float64
 30  Class   75356 non-null  float64
dtypes: float64(30), int64(1)
memory usage: 17.8 MB
```

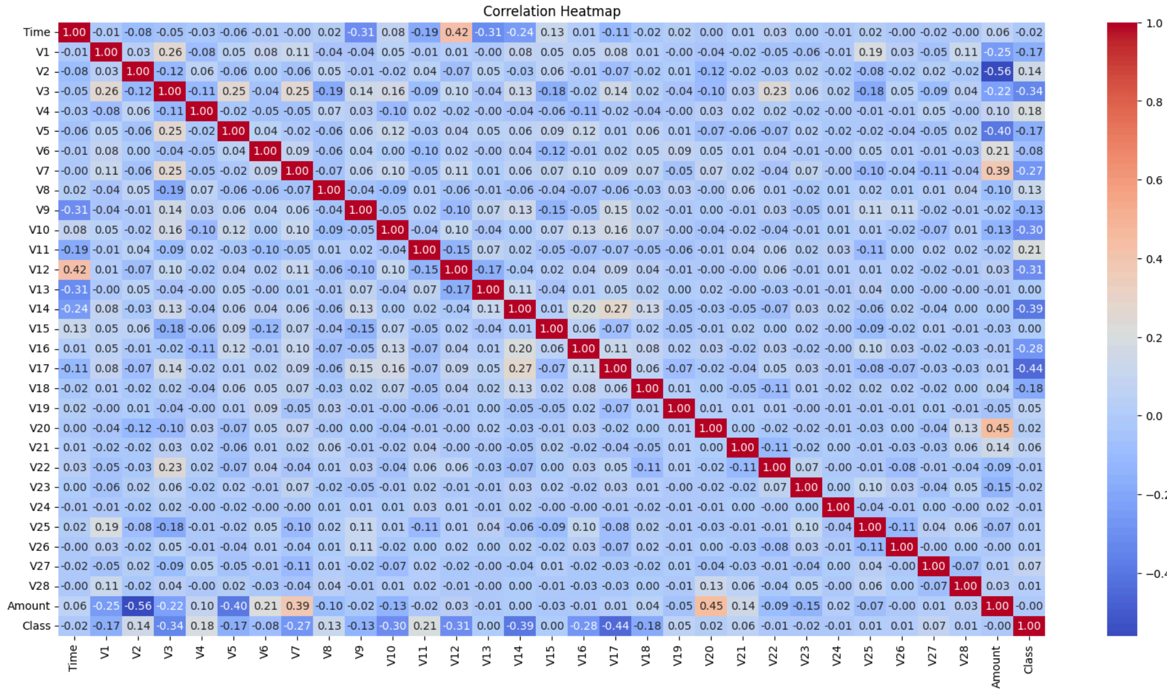
Out[3]:

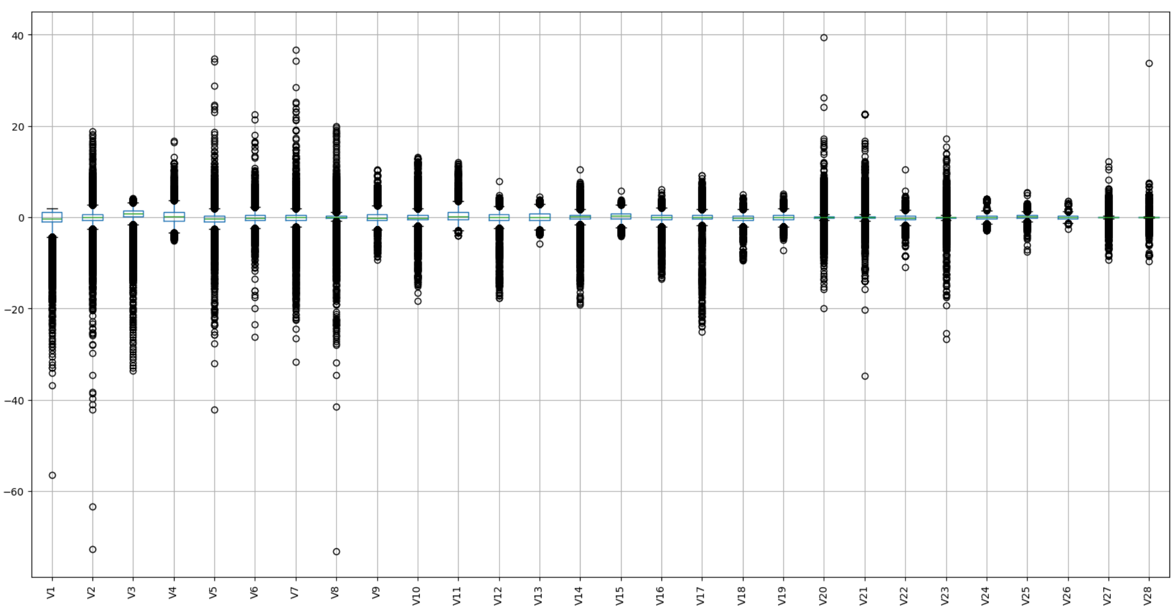
	Time	V1	V2	V3	V4	V5	
count	75357.000000	75357.000000	75357.000000	75357.000000	75357.000000	75357.000000	75
mean	36110.239845	-0.252779	-0.028574	0.678975	0.166227	-0.274493	
std	14826.849221	1.877372	1.660941	1.402475	1.370891	1.387110	
min	0.000000	-56.407510	-72.715728	-33.680984	-5.172595	-42.147898	
25%	29808.000000	-1.014563	-0.595725	0.190522	-0.725740	-0.891744	
50%	39061.000000	-0.246462	0.070645	0.766814	0.185626	-0.306681	
75%	47465.000000	1.153590	0.724071	1.398870	1.049550	0.263561	
max	56021.000000	1.960497	18.902453	4.226108	16.715537	34.801666	

8 rows × 31 columns

In [4]:
```
# Distribution of the target variable
sns.countplot(x='Class', data=df)
plt.title('Class Distribution')
plt.show()

# Correlation heatmap
plt.figure(figsize=(20, 10))
sns.heatmap(df.corr(), annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()

# Boxplot for features
plt.figure(figsize=(20, 10))
df.drop(['Time', 'Amount', 'Class'], axis=1).boxplot()
plt.xticks(rotation=90)
plt.show()
```







Handling Class Imbalance

The dataset is imbalanced, with far fewer fraudulent transactions than non-fraudulent ones. To address this, SMOTE (Synthetic Minority Over-sampling Technique) was used to oversample the minority class (fraudulent transactions) and balance the dataset.

In [6]:
```
# Using SMOTE for oversampling
sm = SMOTE(random_state=42)
df = df.dropna(subset=['Class'])
X = df.drop(['Class', 'Time'], axis=1)
y = df['Class']
X_res, y_res = sm.fit_resample(X, y)
print(y.value_counts())
print(y_res.value_counts())
```

```
Class
0.0    75173
1.0      183
Name: count, dtype: int64
Class
0.0    75173
1.0    75173
Name: count, dtype: int64
```

Feature Scaling

The features were scaled using StandardScaler to ensure that all features contribute equally to model training, especially for models like Logistic Regression that are sensitive to feature scaling.

In [7]:
```
# Scale features
res_scaler = StandardScaler()
X_res_scaled = res_scaler.fit_transform(X_res)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Convert scaled features to DataFrame
X_res_scaled = pd.DataFrame(X_res_scaled, columns=X.columns)
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)
```

Anomaly Detection

Isolation Forest was applied as an unsupervised anomaly detection technique. It helped identify outliers and anomalies in the data, which may represent fraudulent transactions. The predicted anomaly labels were transformed to match the class format for evaluation.

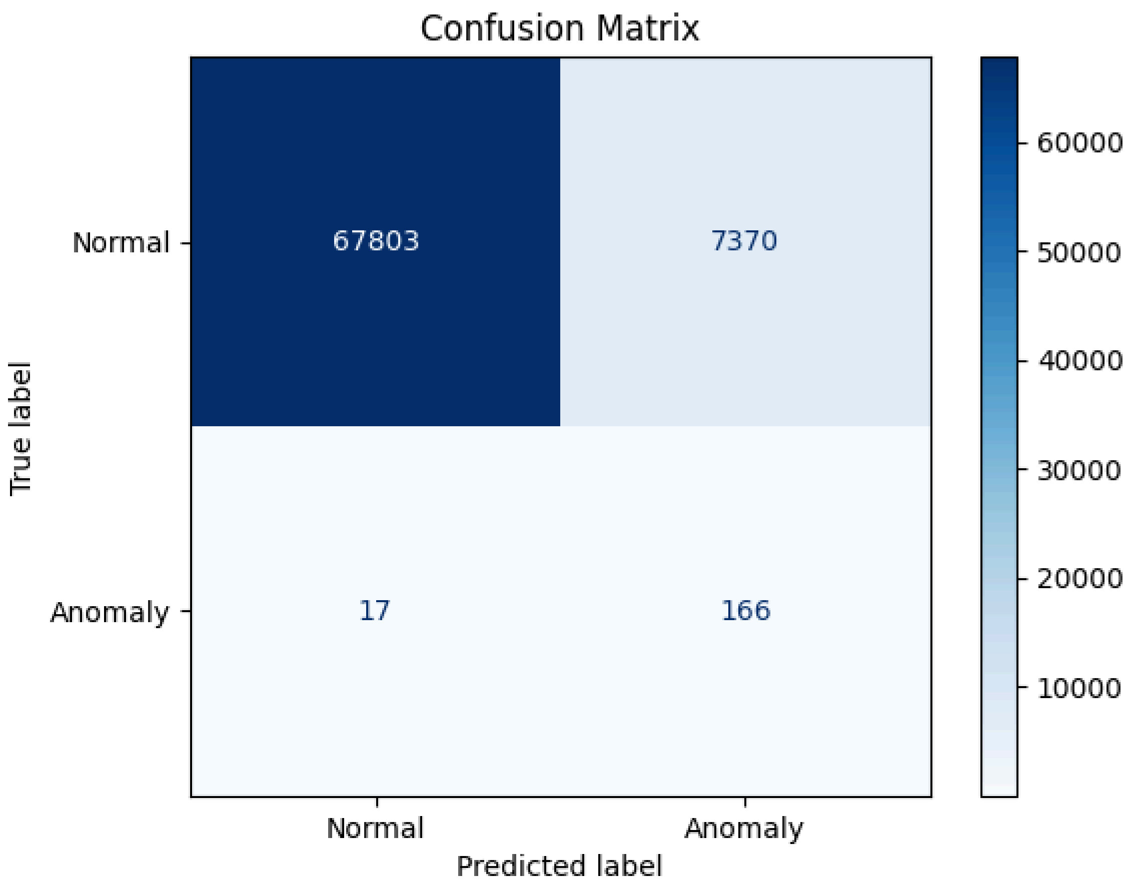
In [8]:
```
# Isolation Forest
iso_forest = IsolationForest(contamination=0.1,random_state=42).fit(X_scaled)
y_pred_iso = iso_forest.predict(X_scaled)
print(pd.Series(y_pred_iso).value_counts())
```

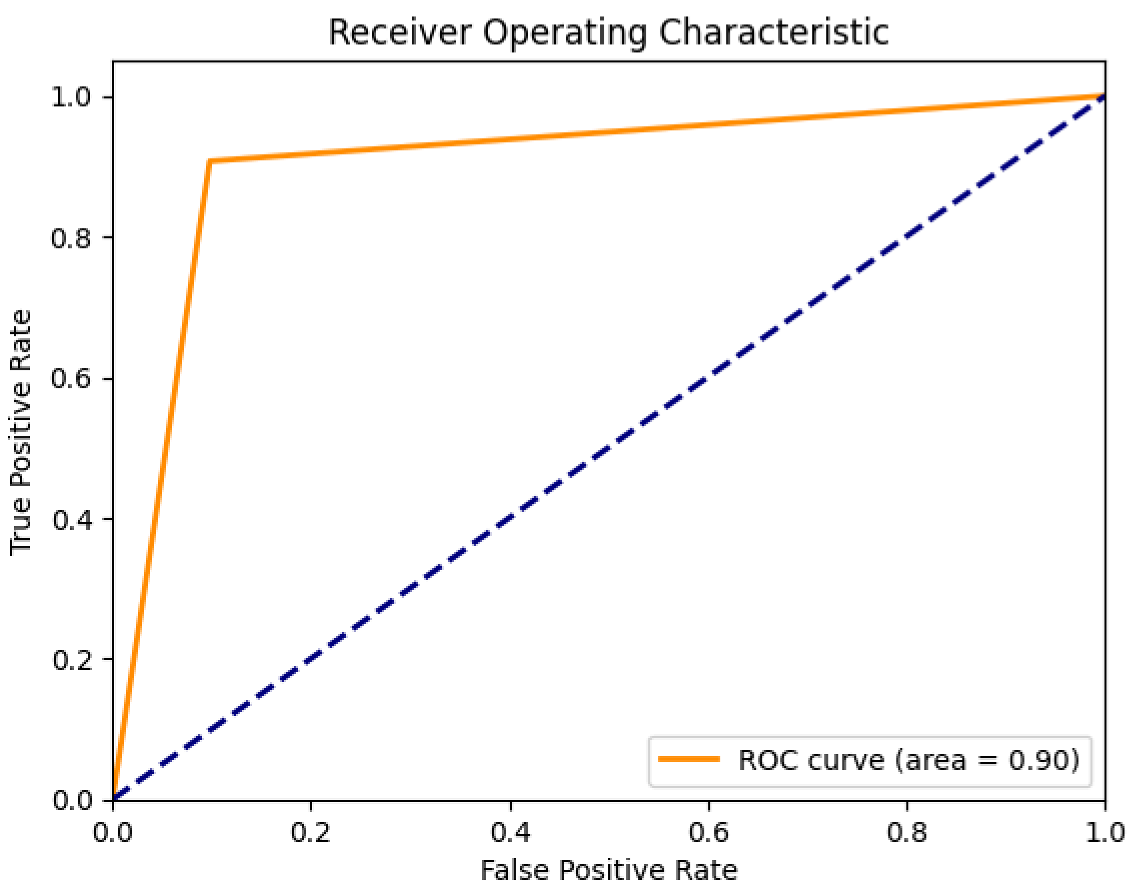
```
 1    67820
-1     7536
Name: count, dtype: int64
```

In [9]:

```
# Replace 1 with 0 for normal and -1 with 1 for anomalies to match true labels
y_pred_iso_binary = np.where(y_pred_iso == 1, 0, 1)

print(classification_report(y, y_pred_iso_binary))

# Compute the confusion matrix
cm = confusion_matrix(y, y_pred_iso_binary)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Normal', '

# Plot the confusion matrix
disp.plot(cmap='Blues')
plt.title('Confusion Matrix')
plt.show()

# ROC-AUC for Isolation Forest
fpr, tpr, _ = roc_curve(y, y_pred_iso_binary)
roc_auc = roc_auc_score(y, y_pred_iso_binary)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_au
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.90      0.95     75173
         1.0       0.02      0.91      0.04       183

    accuracy                           0.90     75356
   macro avg       0.51      0.90      0.50     75356
weighted avg       1.00      0.90      0.95     75356
```





Modeling: Several classification models were trained and evaluated on the dataset

Logistic Regression: A simple linear model that is easy to interpret and serves as a baseline.

XGBoost: A gradient boosting machine learning algorithm known for its high performance on structured data.

Both models were evaluated using metrics such as classification accuracy, confusion matrix, and ROC-AUC score.

In [10]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
import xgboost as xgb

# Split data
X_train, X_test, y_train, y_test = train_test_split(X_res_scaled, y_res, test_

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Logistic Regression
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

# XGBoost Classifier
xgb_clf = xgb.XGBClassifier(eval_metric='logloss')
xgb_clf.fit(X_train, y_train)
y_pred_xgb = xgb_clf.predict(X_test)
```

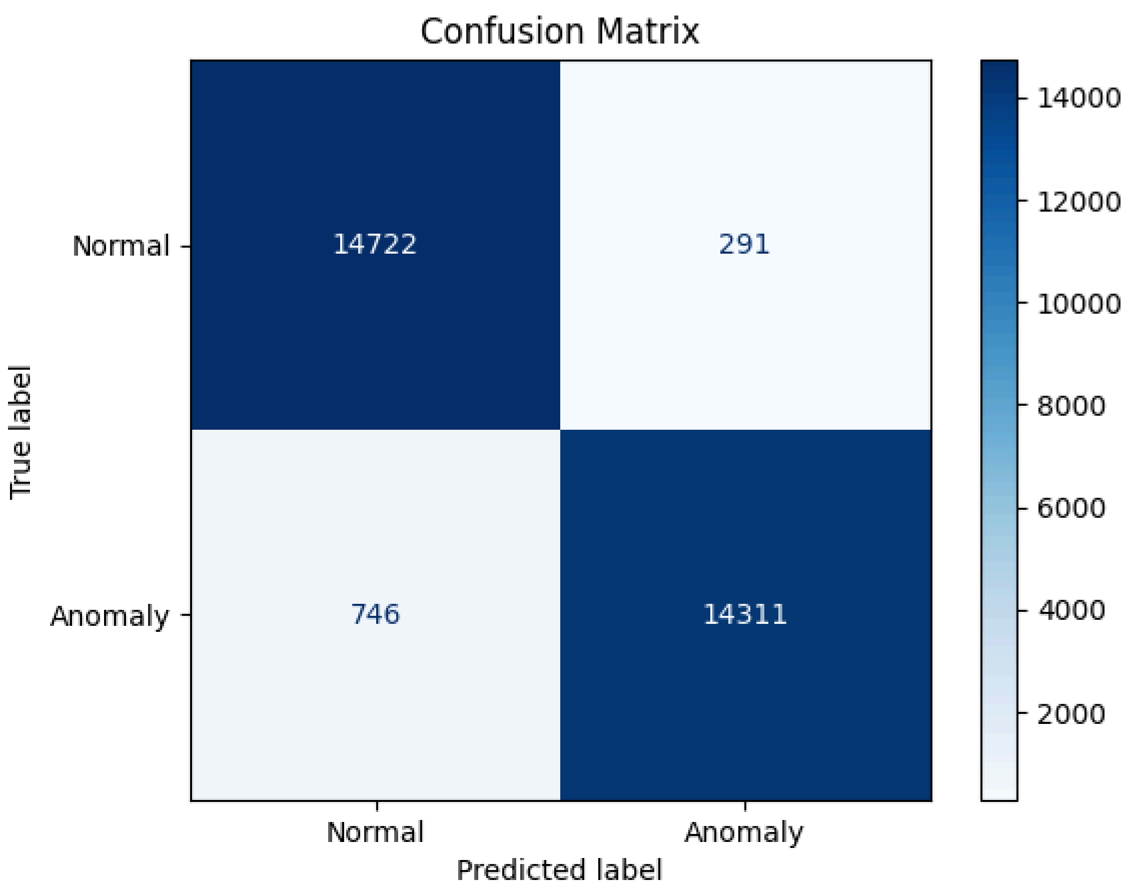
Evaluation

The performance of each model was evaluated using classification reports, confusion matrices, and ROC-AUC curves.

ROC-AUC provides a comprehensive measure of model performance, balancing both false positives and false negatives.

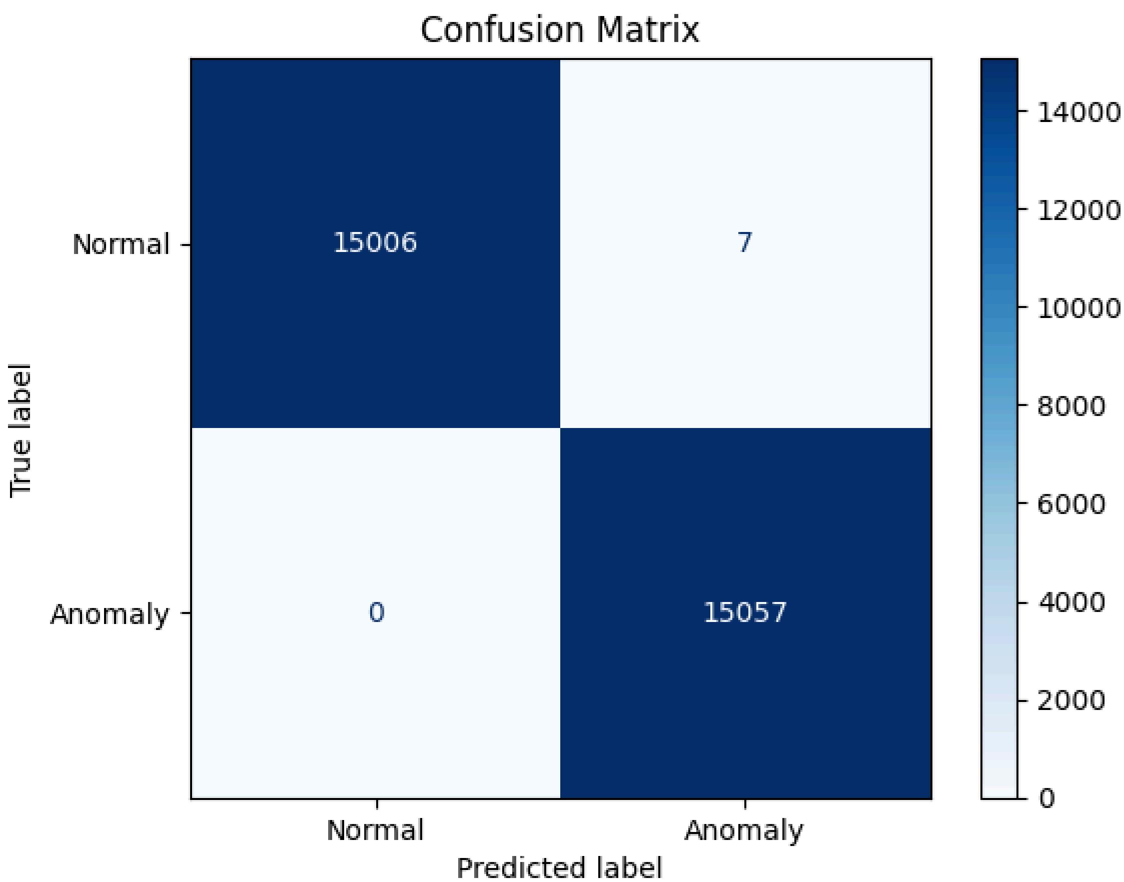
In [11]:
```
# Evaluation function
def evaluate_model(y_test, y_pred):
    print(classification_report(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Normal
    #sns.heatmap(cm, annot=True, fmt='d')
    disp.plot(cmap='Blues')
    plt.title('Confusion Matrix')
    plt.show()

# Logistic Regression Evaluation
print("Logistic Regression")
evaluate_model(y_test, y_pred_lr)

print("XGBoost classifier")
evaluate_model(y_test, y_pred_xgb)
```

```
Logistic Regression
              precision    recall  f1-score   support

         0.0       0.95      0.98      0.97     15013
         1.0       0.98      0.95      0.97     15057

    accuracy                           0.97     30070
   macro avg       0.97      0.97      0.97     30070
weighted avg       0.97      0.97      0.97     30070
```



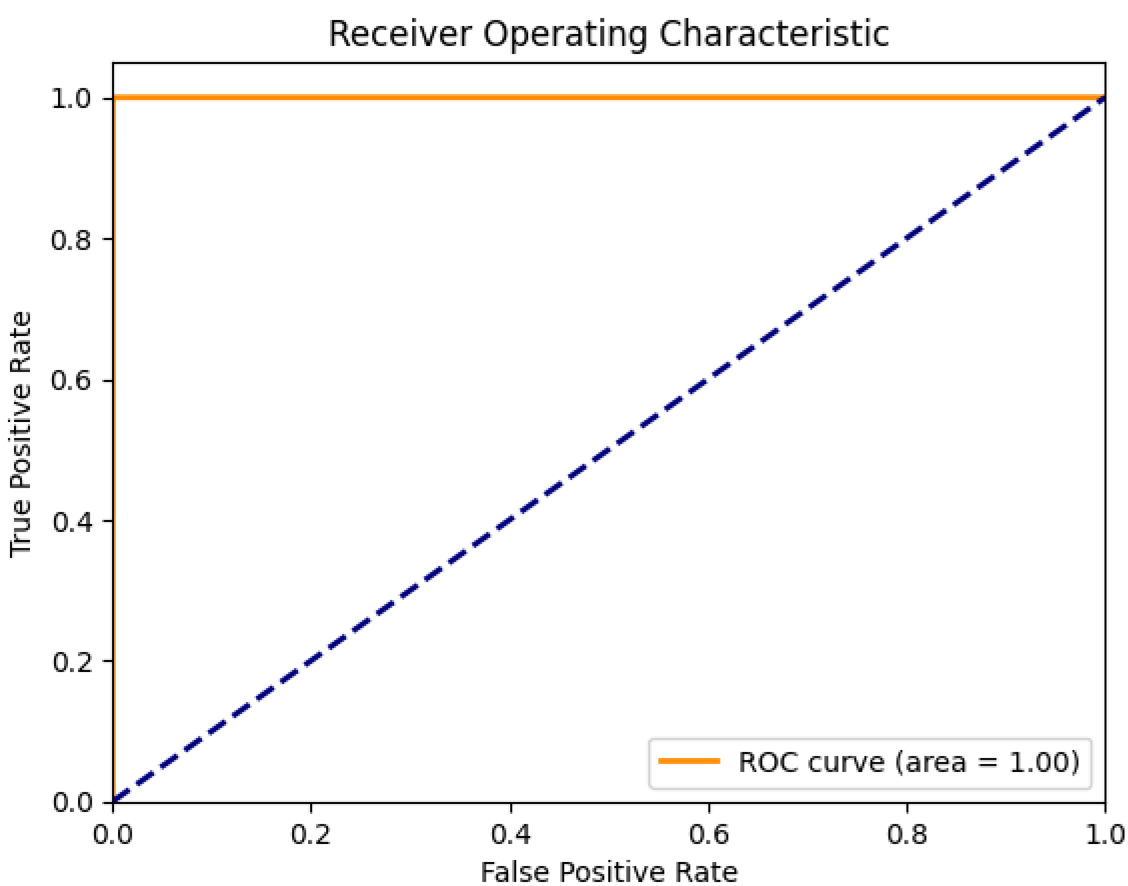
XGBoost classifier

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	15013
1.0	1.00	1.00	1.00	15057
accuracy			1.00	30070
macro avg	1.00	1.00	1.00	30070
weighted avg	1.00	1.00	1.00	30070



In [12]:

```
# ROC-AUC for XGBoost
y_pred_xgb_clf_proba = xgb_clf.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_pred_xgb_clf_proba)
roc_auc = roc_auc_score(y_test, y_pred_xgb_clf_proba)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_au
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



In [ ]: