Objectives

In this project, the aim is to utilize machine learning models to predict occurrences of fraudulent credit card transactions.
1. Detecting Fraudulent Transactions
The main goal is to classify transactions accurately as either authentic or fraudulent. Utilizing data encompassing transaction time, amount, and transformed features V1 to V28, various classification models are employed. To achieve this, we'll test and compare different model building techniques, evaluating their performance through various metrics.
2. Exploring Transaction Patterns
(i). Consecutive Frauds: Do fraudsters tend to strike multiple times in a short period? By analyzing transaction sequences, we can uncover potential patterns of follow-up scams and strengthen our defenses.
(ii). Amount Matters: Are fraudulent transactions typically larger than legitimate ones? We'll compare the typical amounts of authentic and fraudulent transactions, seeking any significant differences that might raise red flags.
(iii). Fraudulent Peak Periods: Do fraud rates climb during specific times of day or days of the week? Can we identify peak transaction periods with a higher risk of fraudulent activity?
(iv). Fraud in High-Volume Times: When overall transaction volume spikes, does the number of fraudulent transactions proportionally increase, is it simply due to an overall increase in transactions, or are there other hidden factors at play?
(v). High-Fraud Time Points: Why do certain moments stand out for unusually high fraud rates? Is it purely due to a high volume of transactions, or could other factors be influencing this surge?

About Dataset

Source: kaggle
The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.
It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.
Feature Information:
Time Feature: Represents the time elapsed in seconds from the first transaction, providing a temporal dimension to each entry in the dataset.
Amount Feature: Denotes the transaction amount, suitable for example-dependent cost-sensitive learning applications.
Class Feature: The target variable; assumes a value of 1 for fraud and 0 for non-fraudulent transactions.

Problem Statemet

Security Awareness: Many banks is retaining high-profit customers, but banking fraud poses a substantial threat, leading to financial losses and eroding trust. A bank informs users of an expiring card and requests verification. Users should avoid sharing sensitive details as fraud cases in digital transactions are on the rise.
Growing Digital Transactions: Despite a 51% growth in digital transactions, concerns persist due to a substantial increase in fraudulent activities, with 52,304 cases reported in FY 2019. While making online purchases, regularly monitor account statements for any suspicious transactions and report them to the bank immediately.
Manual System: Slow, clunky manual reviews choke banks' efficiency, leading to costly chargebacks, frustrated customers, and missed legitimate transactions. This reactive approach leaves everyone vulnerable.

Credit card fraud and its impact:

Financial Protection: With an increasing need to detect fraudulent transactions, machine learning models play a crucial role. Understanding model selection, tuning, and handling class imbalances are essential skills for data scientists. Banks cann utilize machine learning algorithms to analyze transaction patterns, flagging unusual activities and preventing potential fraud. Protect profits, safeguard customers, and thrive in the age of digital payments.
Preserving Trust: Detection models play a crucial role in preserving trust between customers and banks. By promptly identifying and addressing fraudulent activities, banks demonstrate their commitment to security, ensuring customers feel secure in their financial transactions.
Operational Efficiency: Implementing a robust fraud detection model enhances operational efficiency by automating the identification process. For example, a machine learning model can analyze vast datasets more rapidly than manual reviews, minimizing the time and resources required to detect fraudulent transactions.
Regulatory Compliance: Financial institutions must comply with regulations and standards to ensure a secure and trustworthy banking environment. A credit card fraud detection model helps banks adhere to these requirements, avoiding legal issues and reputational damage. For instance, if a bank fails to detect and prevent fraudulent activities, it may face regulatory penalties and lose customer trust.

Common method of credit card fraud:

Magnetic Mimicry: Skimming steals card data for shady swaps.
Counterfeit Cabal: Fake cards join the plastic party, uninvited.
Lost & Found Lies: Stolen cards fuel fraudulent feasts.
Telephonic Trickery: Sweet-talking scammers swipe cash by extracting information.
Others: Tampered cards & Identity theft play a deceitful game.

Steps of this project

1. Perform Exploratory Data Analysis (EDA) for an initial overview of the dataset, focusing on feature distribution, class imbalances, and overall data characteristics.
2. Conduct Data Preprocessing, addressing missing values, and applying normalization or standardization to prepare data for model training.
3. Train various fraud detection models, including XGB Classifier, Light GBM Classifier, Gradient Boosting Classifier, Ada Boost Classifier, Random Forest Classifier, Logistic Regression, Support Vector Machine (SVM), Decision Tree Classifier, Hist Gradient Boosting Classifier.
4. Evaluate model performance using metrics such as AUC-ROC, precision, recall, F1-Score, and visualizations like ROC curves and confusion matrices.

5. Select the most suitable model for credit card fraud detection based on comprehensive evaluation metrics.

Important Metric Used

ROC-AUC Score: ROC-AUC evaluates the overall performance of binary classification models, offering insights into their ability to distinguish between classes. It aids in selecting an optimal probability threshold for classification, striking a balance between sensitivity and specificity based on specific application needs. ROC-AUC serves as a versatile metric for comparing models, particularly beneficial in scenarios with imbalanced datasets by offering robust evaluation unaffected by class distribution differences or when ranking predictions is crucial.
F1 Score:
F1 score considers both false positives and false negatives, making it less influenced by dominant classes and more sensitive to performance in minority classes.
Accuracy can be misleading in imbalanced datasets, as high accuracy may result from simply predicting the majority class, disregarding minority class performance.
ROC-AUC's Generalization Challenge: ROC-AUC may not address class-specific performance concerns, providing an overall measure that may not capture the nuances of minority class detection in imbalanced datasets.
Precision:
High precision is crucial in fraud detection to minimize false positives, as each incorrect classification could result in unnecessary financial investigations and inconvenience for customers.
Precision helps maintain trust by reducing the likelihood of flagging legitimate transactions as fraudulent. High precision instills confidence in customers, ensuring a positive user experience.
Focusing on precision allows for more efficient allocation of resources in investigating flagged transactions. This optimization is vital in managing costs and enhancing the effectiveness of fraud prevention measures.
Recall:
Recall assesses how well a model captures all positive instances by calculating the ratio of true positives to the total of true positives and false negatives, indicating the model's ability to identify actual positive cases.
A high recall signifies effective identification of positive cases with minimal misses, while low recall suggests a higher likelihood of overlooking positive instances, potentially yielding inaccurate outcomes.
Catching a wider net of suspicious transactions, even if some are later confirmed legitimate, allows for prompt investigation and potential early intervention to stop larger fraudulent schemes.
Accuracy:
Accuracy provides a comprehensive measure of the model's correctness, considering both true positives and true negatives. It indicates the overall effectiveness in classifying transactions.
In imbalanced datasets, where fraud cases are rare, accuracy can be misleading. A model predicting all transactions as non-fraudulent may still achieve a high accuracy.
Accuracy is straightforward and easy to interpret, making it accessible for non-experts. Its simplicity allows stakeholders to quickly grasp the model's performance on the dataset.

Import libraries and load the data

In [1]:
```
import pandas as pd
import numpy as np
import plotly.express as px
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Modelling Libraries
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier,GradientBoostingC
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier

# Evaluation & CV Libraries
from sklearn.metrics import classification_report,precision_score,accuracy_score,confusi
from sklearn.metrics import plot_precision_recall_curve
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import ADASYN
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report, precision_recall_cu
from imblearn.over_sampling import SMOTE
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import cross_validate, cross_val_score,GridSearchCV
from sklearn.model_selection import KFold, cross_val_predict,StratifiedKFold

import warnings
warnings.filterwarnings('ignore')
```

```
C:\Users\user\anaconda3\lib\site-packages\sklearn\experimental\enable_hist_gradient_boos
ting.py:16: UserWarning: Since version 1.0, it is not needed to import enable_hist_gradi
ent_boosting anymore. HistGradientBoostingClassifier and HistGradientBoostingRegressor a
re now stable and can be normally imported from sklearn.ensemble.
  warnings.warn(
```

In [2]:
```
file_path = r'C:\Users\user\Downloads\Compressed\Credit_card\creditcard.csv'
df = pd.read_csv(file_path)
```

Handling missing values and explore the dataset

In [3]:
```
print('Data dimension     :',df.shape)

print('Data size          :',df.size)

print('Number of Row          :',len(df.index))

print('Number of Columns      :',len(df.columns))

print(df.isnull().sum())

print(df.info())

print(df.describe())
```

```
Data dimension     : (284807, 31)
Data size          : 8829017
Number of Row          : 284807
Number of Columns      : 31
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
<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
None
                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  3.918649e-15  5.682686e-16 -8.761736e-15  2.811118e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 V5            V6            V7            V8            V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean  -1.552103e-15  2.040130e-15 -1.698953e-15 -1.893285e-16 -3.147640e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       ...           V21           V22           V23           V24  \
count  ...  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   ...  1.473120e-16  8.042109e-16  5.282512e-16  4.456271e-15
std    ...  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    ... -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    ... -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    ... -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    ...  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    ...  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28         Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean   1.426896e-15  1.701640e-15 -3.662252e-16 -1.217809e-16      88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

               Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
```

This dataset is completely clean, with zero missing values to fill in. We can jump straight into the good stuff without worrying about replacing any value.

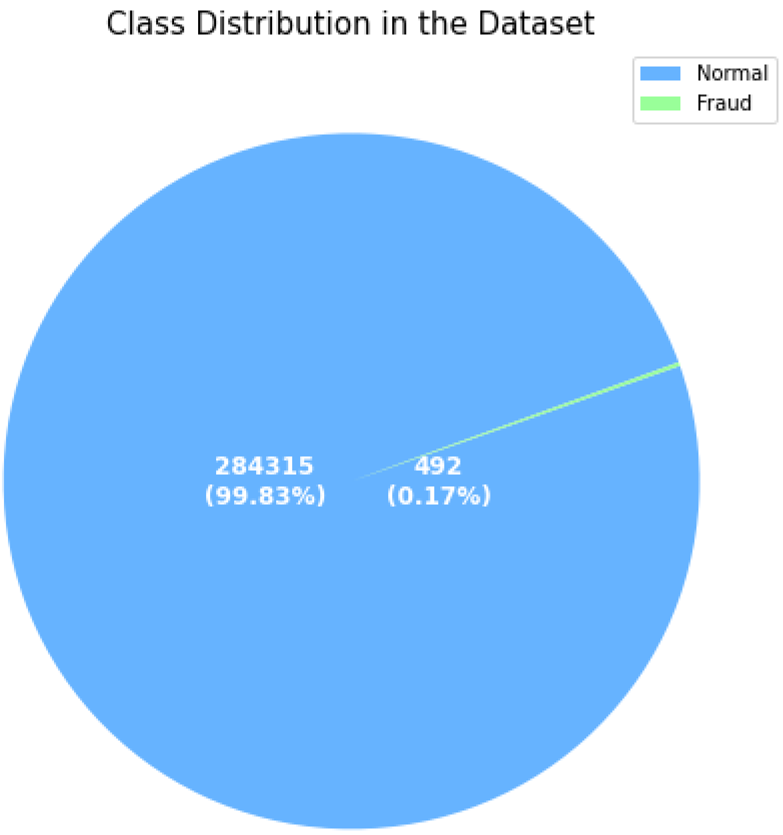
Distribution of classes

In [4]:
```python
classes=df['Class'].value_counts()
normal_share=round(classes[0]/df['Class'].count()*100,2)
fraud_share=round(classes[1]/df['Class'].count()*100, 2)
```

```
print('Non-Fraudulent : {} %'.format(normal_share))
print('    Fraudulent : {} %'.format(fraud_share))
```

```
Non-Fraudulent : 99.83 %
    Fraudulent : 0.17 %
```

In [5]:
```
class_counts = df['Class'].value_counts()

plt.figure(figsize=(8, 8))
plt.pie(class_counts,  startangle=20, colors=['#66b3ff', '#99ff99'])
plt.title('Class Distribution in the Dataset', fontdict ={'fontsize': 15})

for i, count in enumerate(class_counts):
    plt.text(0.5 * (i - 0.5), 0, f'{count}\n({(count / len(df) * 100):.2f}%)', ha='cente
                  color='white',size= 'large', fontweight='bold')
plt.legend(['Normal', 'Fraud'], loc='upper right')
plt.show()
```



Our data is severely imbalanced: out of 284,807 transactions, only 492 are marked as fraudulent (0.17%). That means a whopping 99.83% are normal transactions, throwing off standard analysis techniques

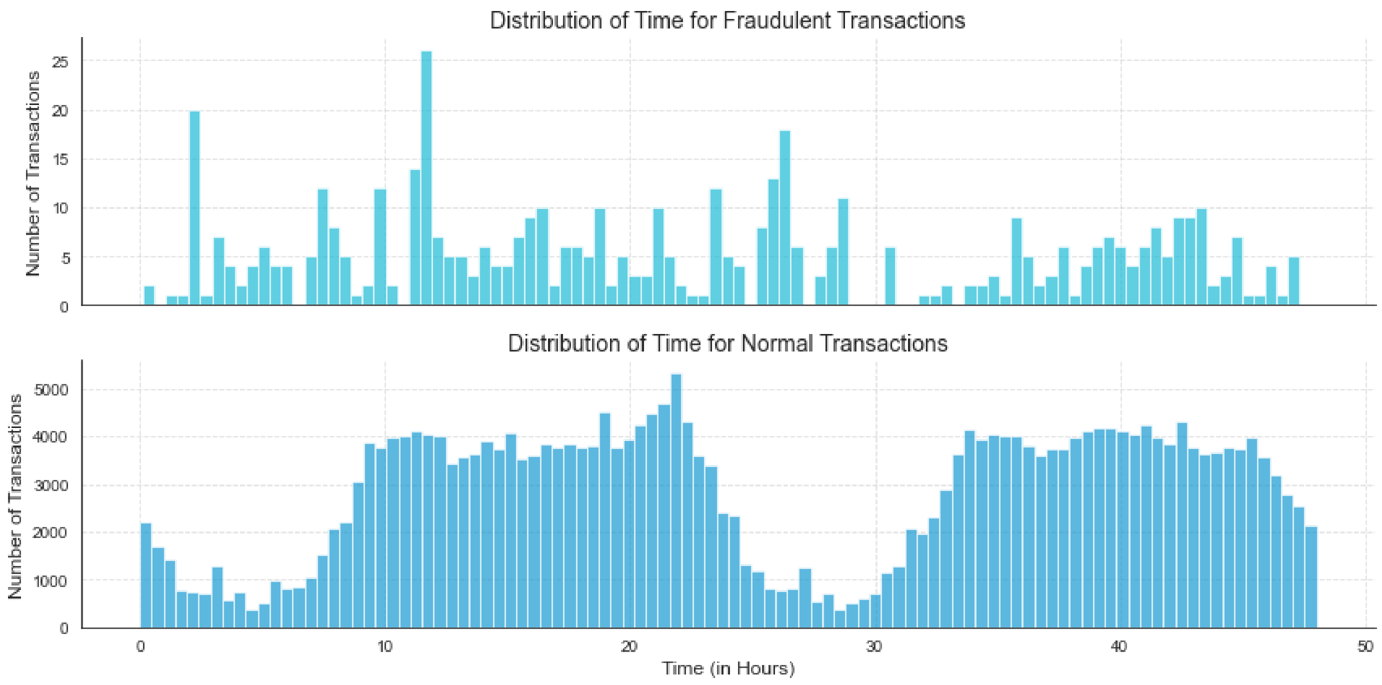
Distribution of Time for Normal & Fraudulent Transactions

This complete transactions occurs over a two-day period.

In [6]:
```
sns.set_style('white')
df.loc[:,'Time'] =df.Time / 3600
```

In [7]:
```
# Create subplots
fig, axs = plt.subplots(2, 1, sharex=True, figsize=(12, 6))

# Plotting for Fraud transactions
sns.histplot(df.Time[df.Class == 1], bins=100, color='#1ebbd7', alpha=0.7, ax=axs[0])
```

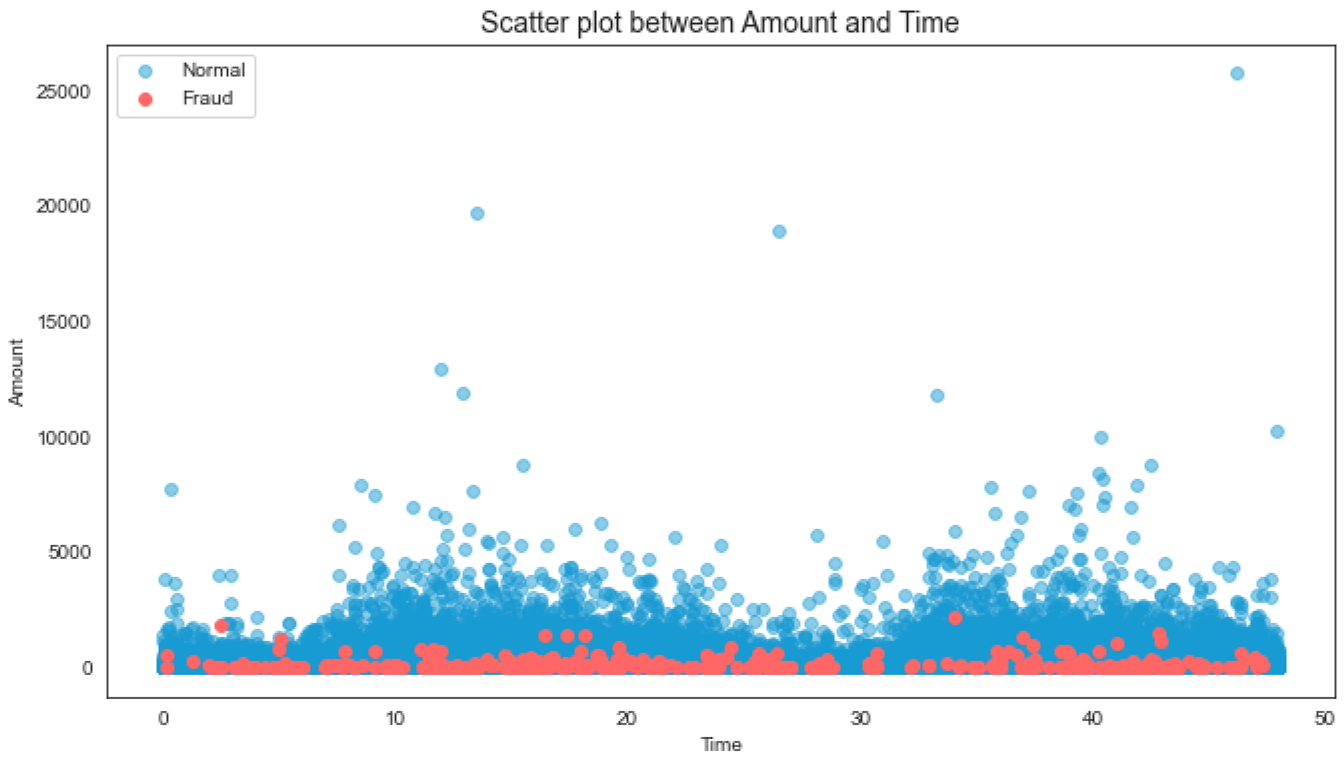
```
axs[0].set_title('Distribution of Time for Fraudulent Transactions', fontsize=14)
axs[0].set_ylabel('Number of Transactions', fontsize=12)
axs[0].grid(True, linestyle='--', alpha=0.6)

# Plotting for Normal transactions
sns.histplot(df.Time[df.Class == 0], bins=100, color='#189ad3', alpha=0.7, ax=axs[1])
axs[1].set_title('Distribution of Time for Normal Transactions', fontsize=14)
axs[1].set_xlabel('Time (in Hours)', fontsize=12)
axs[1].set_ylabel('Number of Transactions', fontsize=12)
axs[1].grid(True, linestyle='--', alpha=0.6)

# Remove top and right spines
for ax in axs:
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)

plt.tight_layout()
```



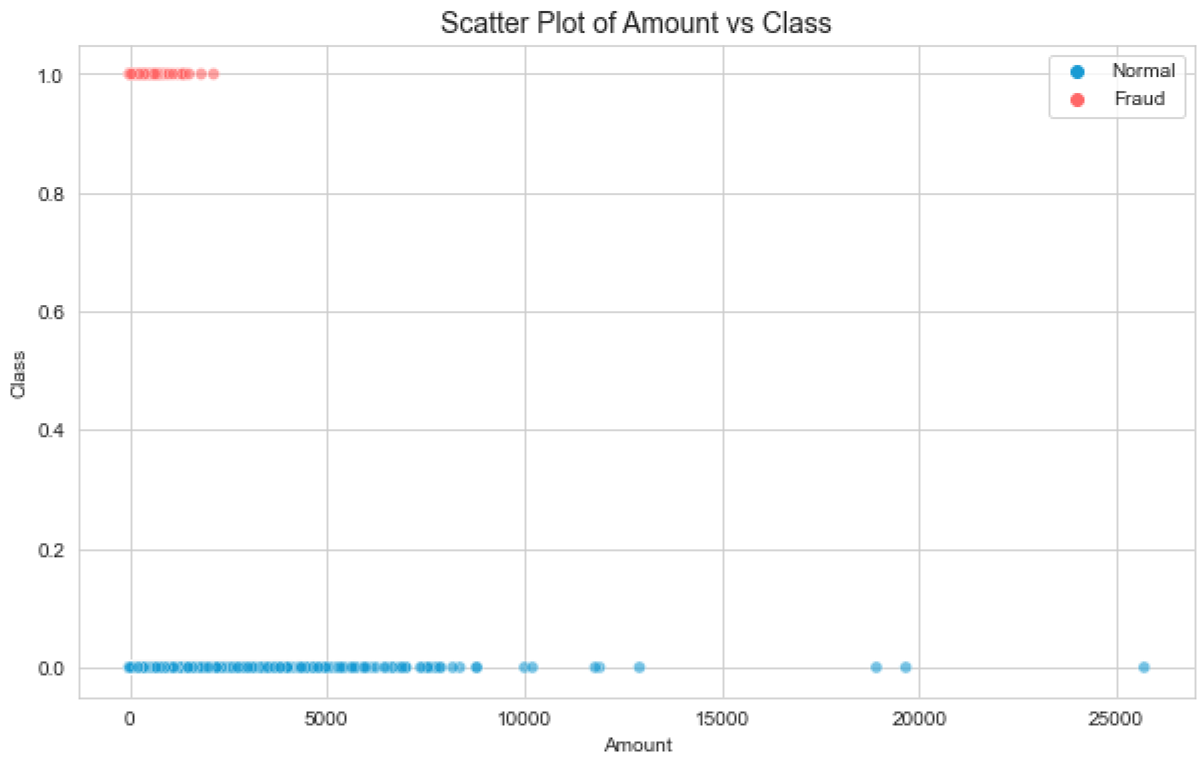
Most of the fraudulent transactions appear to have occurred between the time intervals of 9 hours to 28 hours after a certain event, with another cluster observed between 36 hours and 46 hours thereafter. These time frames suggest potential patterns or anomalies in the timing of fraudulent activities, which could be further investigated for insights into the nature of the fraudulent behavior or to enhance fraud detection algorithms.

Time and Amount

In [8]:
```
df_normal = df[df.Class == 0]
df_fraud = df[df.Class == 1]

fig = plt.figure(figsize = (11,6))

plt.scatter(df_normal.Time.values, df_normal.Amount, alpha=0.5, label='Normal', c='#189a
plt.scatter(df_fraud.Time.values, df_fraud.Amount, alpha=1, label='Fraud', c='#ff6666')

plt.ylabel('Amount')
plt.xlabel('Time')
plt.title('Scatter plot between Amount and Time', fontdict ={'fontsize': 14})
plt.legend()
plt.show()
```



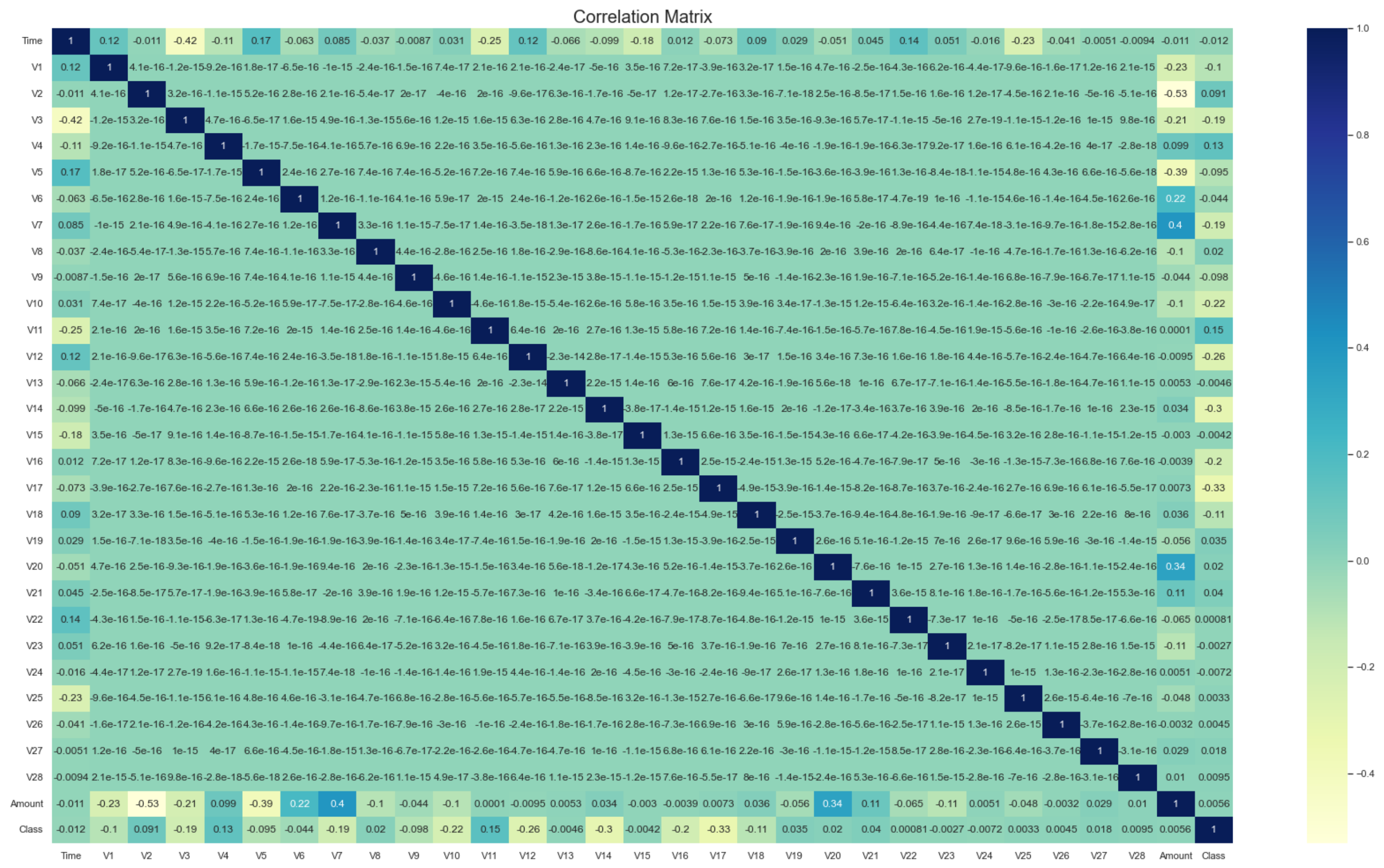
```
In [9]: sns.set_style('whitegrid')

        plt.figure(figsize=(10, 6))
        scatter = sns.scatterplot(x='Amount', y='Class', hue='Class', palette={0: '#189ad3', 1:

        plt.xlabel('Amount')
        plt.ylabel('Class')
        plt.title('Scatter Plot of Amount vs Class', fontdict ={'fontsize': 14})

        # Set legend labels
        legend_labels = {0: 'Normal', 1: 'Fraud'}
        handles, _ = scatter.get_legend_handles_labels()
        plt.legend(handles=handles, labels=[legend_labels[label] for label in df['Class'].unique
        plt.show()
```



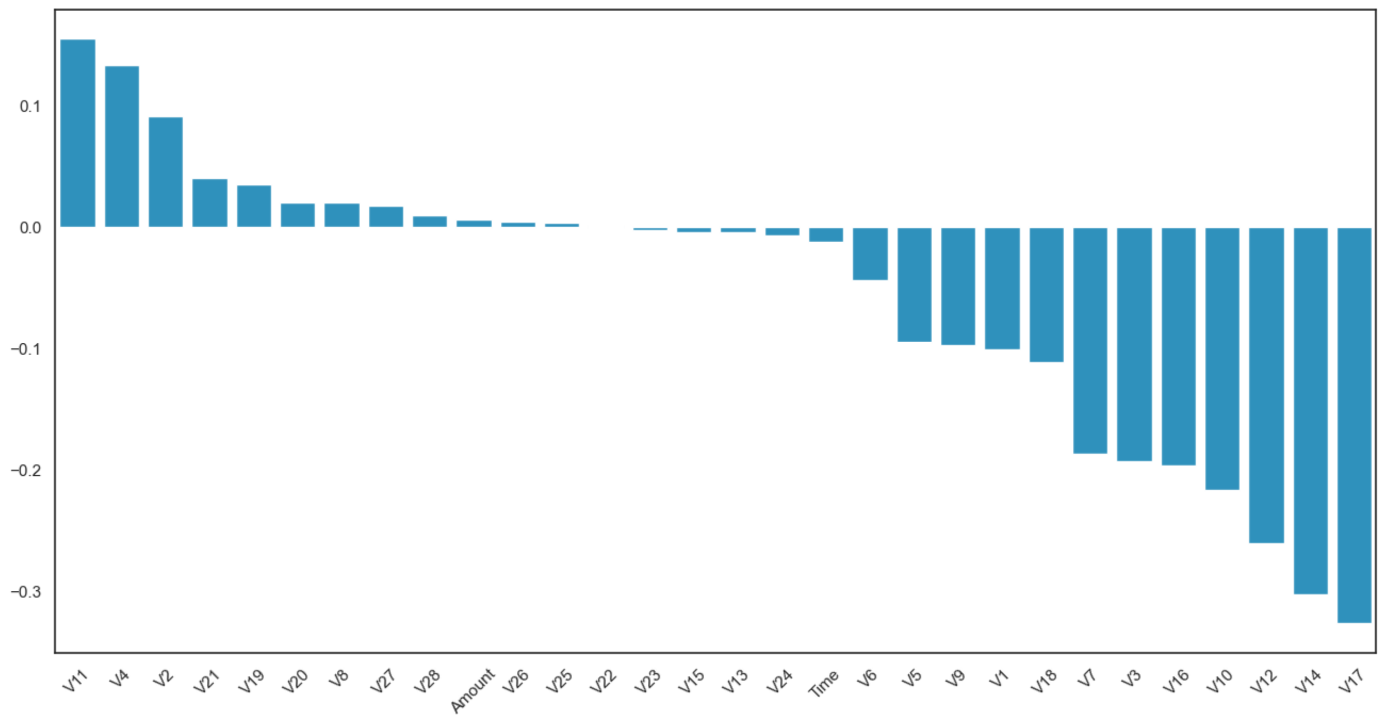
From the above two plots, we can observe a significant trend where the majority of fraudulent transactions appear to involve amounts less than 3500. This suggests that fraudsters may target smaller transactions to avoid detection or suspicion, as these amounts are less likely to trigger alerts or scrutiny compared to larger transactions.
Understanding this pattern can aid in developing more targeted fraud detection strategies, such as implementing additional scrutiny or verification steps for transactions above a certain threshold or focusing on monitoring smaller transactions more closely for suspicious activity.

Correlation analysis

In [10]:
```
# Calculate correlation matrix
correlation_matrix = df.corr()

# Set the aesthetic style of the plots
sns.set(style='white')

plt.figure(figsize=(30,17))
sns.heatmap(df.corr(),annot=True,cmap='YlGnBu')
plt.title('Correlation Matrix', fontdict ={'fontsize': 20})
plt.show()
```



In [11]:
```
plt.figure(figsize = (16,8),dpi = 100)
sns.barplot(x = df.corr()['Class'].drop(index='Class').sort_values(ascending=False).inde
            y = df.corr()['Class'].drop(index='Class').sort_values(ascending=False).valu
plt.xticks(rotation = 45);
```



There are no features exhibiting a strong linear correlation; however, a discernible relationship emerges among features V17, V14, V12, and V10, indicating negative correlations. Specifically, fraudulent transactions tend to increase as the values of these parameters decrease.
Conversely, features V2, V4, V11, and V19 demonstrate positive correlations, suggesting that the likelihood of a fraudulent transaction rises with higher values of these variables. These insights underscore the importance of considering these features in fraud detection models.

For highly correlated columns, we set multiple threshold value.

In [12]:
```
df_filter = df[(df['Class'] == 1) | (df['V17']< 0.4) & (df['V14']< 0.5) & (df['V12'] < 0
print(df_filter.Class.value_counts(normalize=True))
print('Shape of df      ', df.shape)
print('Shape of df_filter ', df_filter.shape)
```

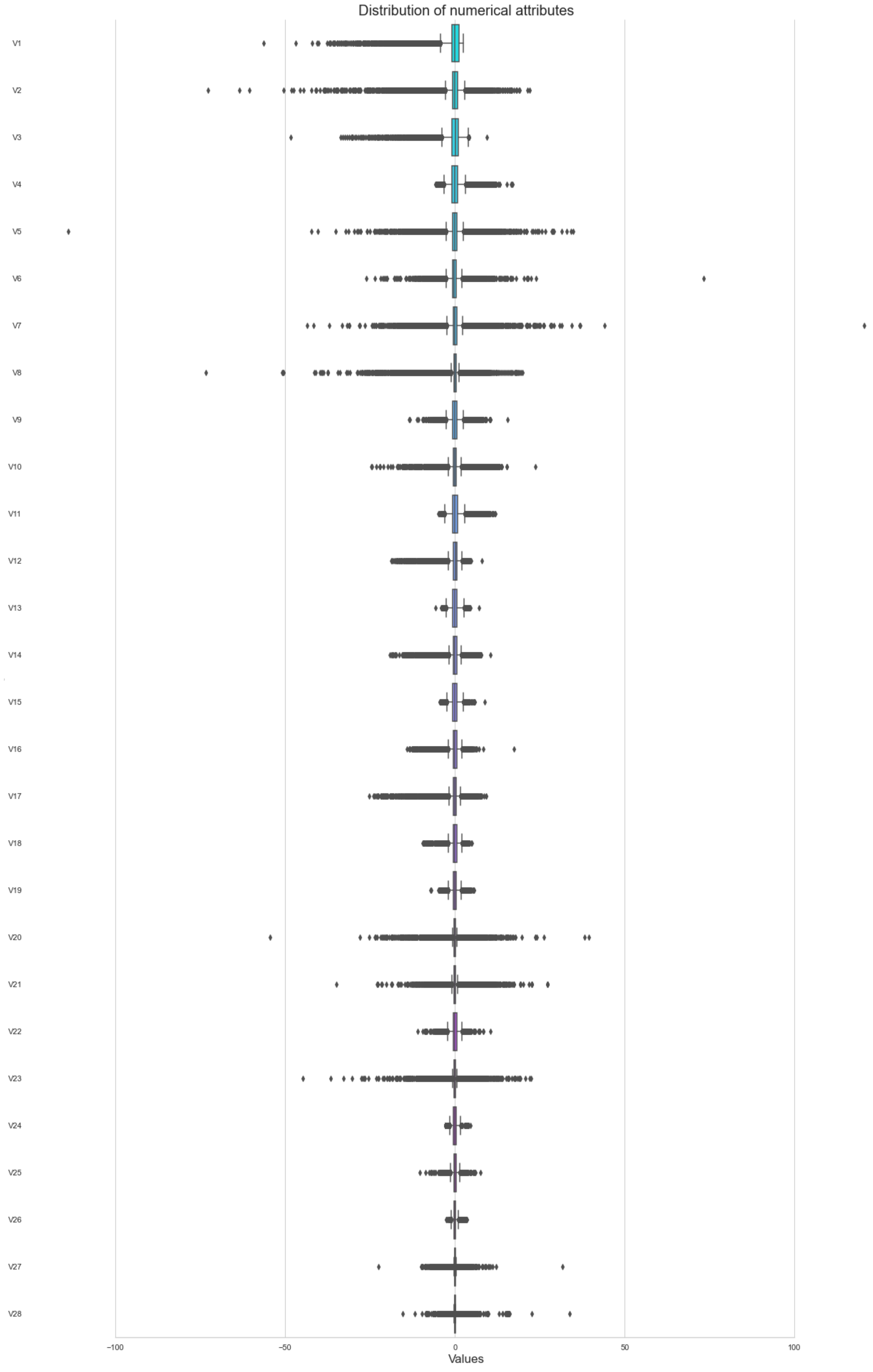
```
0    0.985133
1    0.014867
Name: Class, dtype: float64
Shape of df     (284807, 31)
Shape of df_filter  (33093, 31)
```

Outliers

Distribution of numerical attributes

In [13]:
```
numeric_features = df.select_dtypes(include=[np.number]).columns.tolist()

li_not_plot = ['Class', 'Time', 'Amount']
li_transform_num_feats = [c for c in list(numeric_features) if c not in li_not_plot]

sns.set_style('whitegrid')
f, ax = plt.subplots(figsize=(22, 34))

ax = sns.boxplot(data=df[li_transform_num_feats], orient='h', palette='cool')
ax.set_title('Distribution of numerical attributes', fontsize=20)
ax.set_ylabel('Features', fontsize=1)
ax.set_xlabel('Values', fontsize=17)
```

```
sns.despine(trim=True, left=True)
plt.show()
```


Distribution of numerical attributes
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16
V17
V18
V19
V20
V21
V22
V23
V24
V25
V26
V27
V28
−100
−50
0
50
100
Values

Many variables exhibit outliers, as evidenced by the presence of skewness in the data distribution. To address this issue, we filter the DataFrame based on outlier detection techniques, such as the interquartile range (IQR) method or z-score method, to remove or mitigate the impact of outliers on the analysis and modeling process.

The Interquartile Range (IQR) is computed as the difference between the 25th and 75th percentiles. Our goal is to establish a threshold that will be destroyed if any occurrence exceeds it, somewhere between the 75th and 25th percentiles.

In [14]:
```
def detect_outliers(data: pd.DataFrame, col_name: str, p=1.5) -> int:
    first_quartile = np.percentile(np.array(data[col_name].tolist()), 25)
    third_quartile = np.percentile(np.array(data[col_name].tolist()), 75)
    IQR = third_quartile - first_quartile

    upper_limit = third_quartile + (p * IQR)
    lower_limit = first_quartile - (p * IQR)
    outlier_count = 0

    for value in data[col_name].tolist():
        if (value < lower_limit) or (value > upper_limit):
            outlier_count += 1

    print(f'Lower Limit: {lower_limit}, Upper Limit: {upper_limit}, Outlier Count: {outl
    return lower_limit, upper_limit, outlier_count
```

In [15]:
```
features = ['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount']
iqr=2
print(f'Number of Outliers for {iqr}*IQR after Logarithmed\n')

total=0
for col in features:
    if detect_outliers(df_filter, col)[2] > 0:
        outliers=detect_outliers(df, col, iqr)[2]
        total+=outliers
        print("{} outliers in '{}'".format(outliers,col))
print('\n{} OUTLIERS TOTALLY'.format(total))
```

```
Number of Outliers for 2*IQR after Logarithmed

Lower Limit: -20.035555555555554, Upper Limit: 73.59333333333333, Outlier Count: 0
Lower Limit: -3.448486617547214, Upper Limit: 3.2462408356046417, Outlier Count: 607
Lower Limit: -5.392403540926696, Upper Limit: 5.787671850414239, Outlier Count: 4432
4432 outliers in 'V1'
Lower Limit: -1.9249552717537064, Upper Limit: 2.4270675079540496, Outlier Count: 1523
Lower Limit: -3.4030974828749367, Upper Limit: 3.6082714406501153, Outlier Count: 8960
8960 outliers in 'V2'
Lower Limit: -2.659418295308775, Upper Limit: 3.707226670467681, Outlier Count: 674
Lower Limit: -4.7254855993965315, Upper Limit: 4.862316303706946, Outlier Count: 1816
1816 outliers in 'V3'
Lower Limit: -3.1644317442658156, Upper Limit: 2.57295538195108, Outlier Count: 827
Lower Limit: -4.032602927930994, Upper Limit: 3.9273041010683087, Outlier Count: 4736
4736 outliers in 'V4'
Lower Limit: -2.472922586636611, Upper Limit: 2.7943305108203935, Outlier Count: 1830
Lower Limit: -3.298644092133358, Upper Limit: 3.218973460980894, Outlier Count: 6037
6037 outliers in 'V5'
Lower Limit: -2.4393538908042833, Upper Limit: 2.0044395011918086, Outlier Count: 4350
Lower Limit: -3.1020166180936775, Upper Limit: 2.7322859059892934, Outlier Count: 19016
19016 outliers in 'V6'
Lower Limit: -1.763739005918088, Upper Limit: 2.268749911998928, Outlier Count: 1386
Lower Limit: -2.8030997828647646, Upper Limit: 2.819459976705841, Outlier Count: 5809
```

```
5809 outliers in 'V7'
Lower Limit: -0.8760444766992159, Upper Limit: 1.0388943239345039, Outlier Count: 2253
Lower Limit: -1.2805809559652974, Upper Limit: 1.39929707384928, Outlier Count: 17311
17311 outliers in 'V8'
Lower Limit: -1.8010850308332944, Upper Limit: 1.9167239300288614, Outlier Count: 1263
Lower Limit: -3.123570771364312, Upper Limit: 3.0776122313799887, Outlier Count: 2969
2969 outliers in 'V9'
Lower Limit: -1.5002799988668098, Upper Limit: 0.7727055735177364, Outlier Count: 1177
Lower Limit: -2.5141240697589846, Upper Limit: 2.432621788405168, Outlier Count: 5766
5766 outliers in 'V10'
Lower Limit: -2.819278905389737, Upper Limit: 1.5000691627992948, Outlier Count: 639
Lower Limit: -3.7666694011821447, Upper Limit: 3.743768612990773, Outlier Count: 334
334 outliers in 'V11'
Lower Limit: -1.1567704599249102, Upper Limit: 0.6327173970837997, Outlier Count: 1457
Lower Limit: -2.4531905222135126, Upper Limit: 2.665857069719235, Outlier Count: 8259
8259 outliers in 'V12'
Lower Limit: -2.5373987225106935, Upper Limit: 1.555355733344862, Outlier Count: 412
Lower Limit: -3.2706278162236533, Upper Limit: 3.284593476549059, Outlier Count: 401
401 outliers in 'V13'
Lower Limit: -1.4052767912602544, Upper Limit: 1.2303349067083094, Outlier Count: 1234
Lower Limit: -2.2630217358012783, Upper Limit: 2.330597572564434, Outlier Count: 7069
7069 outliers in 'V14'
Lower Limit: -2.1007578182249844, Upper Limit: 2.825278691428635, Outlier Count: 349
Lower Limit: -3.046294450106684, Upper Limit: 3.112230977266386, Outlier Count: 565
565 outliers in 'V15'
Lower Limit: -1.250824131144544, Upper Limit: 1.8275973351930577, Outlier Count: 1829
Lower Limit: -2.4507029250893817, Upper Limit: 2.505962470851828, Outlier Count: 2865
2865 outliers in 'V16'
Lower Limit: -1.626066623121428, Upper Limit: 1.0589091504793187, Outlier Count: 546
Lower Limit: -2.250594906421913, Upper Limit: 2.166521575365249, Outlier Count: 3957
3957 outliers in 'V17'
Lower Limit: -1.7978054013015319, Upper Limit: 1.831127905166396, Outlier Count: 967
Lower Limit: -2.4981628897695547, Upper Limit: 2.50011983799173, Outlier Count: 1752
1752 outliers in 'V18'
Lower Limit: -1.555310032142419, Upper Limit: 1.4476056189685729, Outlier Count: 1685
Lower Limit: -2.2867954677587004, Upper Limit: 2.2894459047769318, Outlier Count: 4466
4466 outliers in 'V19'
Lower Limit: -0.617804048399319, Upper Limit: 0.600848054681393, Outlier Count: 2825
Lower Limit: -0.9012457760113302, Upper Limit: 0.8225652523313776, Outlier Count: 20808
20808 outliers in 'V20'
Lower Limit: -0.9302316407061801, Upper Limit: 0.8553173583244841, Outlier Count: 1121
Lower Limit: -1.057939247092702, Upper Limit: 1.0159215036927605, Outlier Count: 11156
11156 outliers in 'V21'
Lower Limit: -2.4164832347441565, Upper Limit: 2.1794414466809116, Outlier Count: 46
Lower Limit: -2.6841583886499576, Upper Limit: 2.6703616513232826, Outlier Count: 614
614 outliers in 'V22'
Lower Limit: -0.6422512191583025, Upper Limit: 0.5561060885055015, Outlier Count: 2310
Lower Limit: -0.7808231627567535, Upper Limit: 0.7666188815979189, Outlier Count: 13267
13267 outliers in 'V23'
Lower Limit: -1.6661762141507568, Upper Limit: 1.7107226286406187, Outlier Count: 227
Lower Limit: -1.9428116095648675, Upper Limit: 2.0277520733235552, Outlier Count: 446
446 outliers in 'V24'
Lower Limit: -1.325016395388628, Upper Limit: 1.338415391370764, Outlier Count: 568
Lower Limit: -1.652866287930582, Upper Limit: 1.6864367967326979, Outlier Count: 2219
2219 outliers in 'V25'
Lower Limit: -1.114299038175788, Upper Limit: 1.129324422598612, Outlier Count: 527
Lower Limit: -1.4628561250716694, Upper Limit: 1.3768243729057055, Outlier Count: 1247
1247 outliers in 'V26'
Lower Limit: -0.41844186979181347, Upper Limit: 0.4987614281268849, Outlier Count: 1932
Lower Limit: -0.39460882728502145, Upper Limit: 0.41481441766635907, Outlier Count: 2771
0
27710 outliers in 'V27'
Lower Limit: -0.2659321394270262, Upper Limit: 0.3234837183177757, Outlier Count: 2392
Lower Limit: -0.315439288566583, Upper Limit: 0.3407594503074223, Outlier Count: 21280
21280 outliers in 'V28'
Lower Limit: -96.305, Upper Limit: 171.175, Outlier Count: 3956
Lower Limit: -137.53, Upper Limit: 220.295, Outlier Count: 26132
26132 outliers in 'Amount'

231399 OUTLIERS TOTALLY
```

After applying outlier detection techniques to the DataFrame, we identified a total of **231,399 data points** classified as outliers across various variables.
These outliers are potentially influential points that deviate significantly from the majority of the data and may have a disproportionate impact on the analysis and modeling outcomes.

Splitting the data into train & test data

In [16]:
```
X = df.drop(['Class'],axis=1)
y = df['Class']
```

In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42
print('Shape of train_X data: ', X_train.shape)
print('Shape of test_X data: ', X_test.shape)
print('Fraudulent Count for Full data : ',np.sum(y))
print('Fraudulent Count for Train data : ',np.sum(y_train))
print('Fraudulent Count for Test data : ',np.sum(y_test))
```

```
Shape of train_X data:  (199364, 30)
Shape of test_X data:  (85443, 30)
Fraudulent Count for Full data :  492
Fraudulent Count for Train data :  356
Fraudulent Count for Test data :  136
```

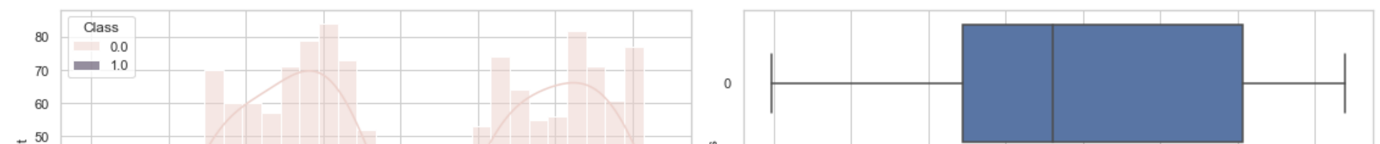
Make a copy of train and test data

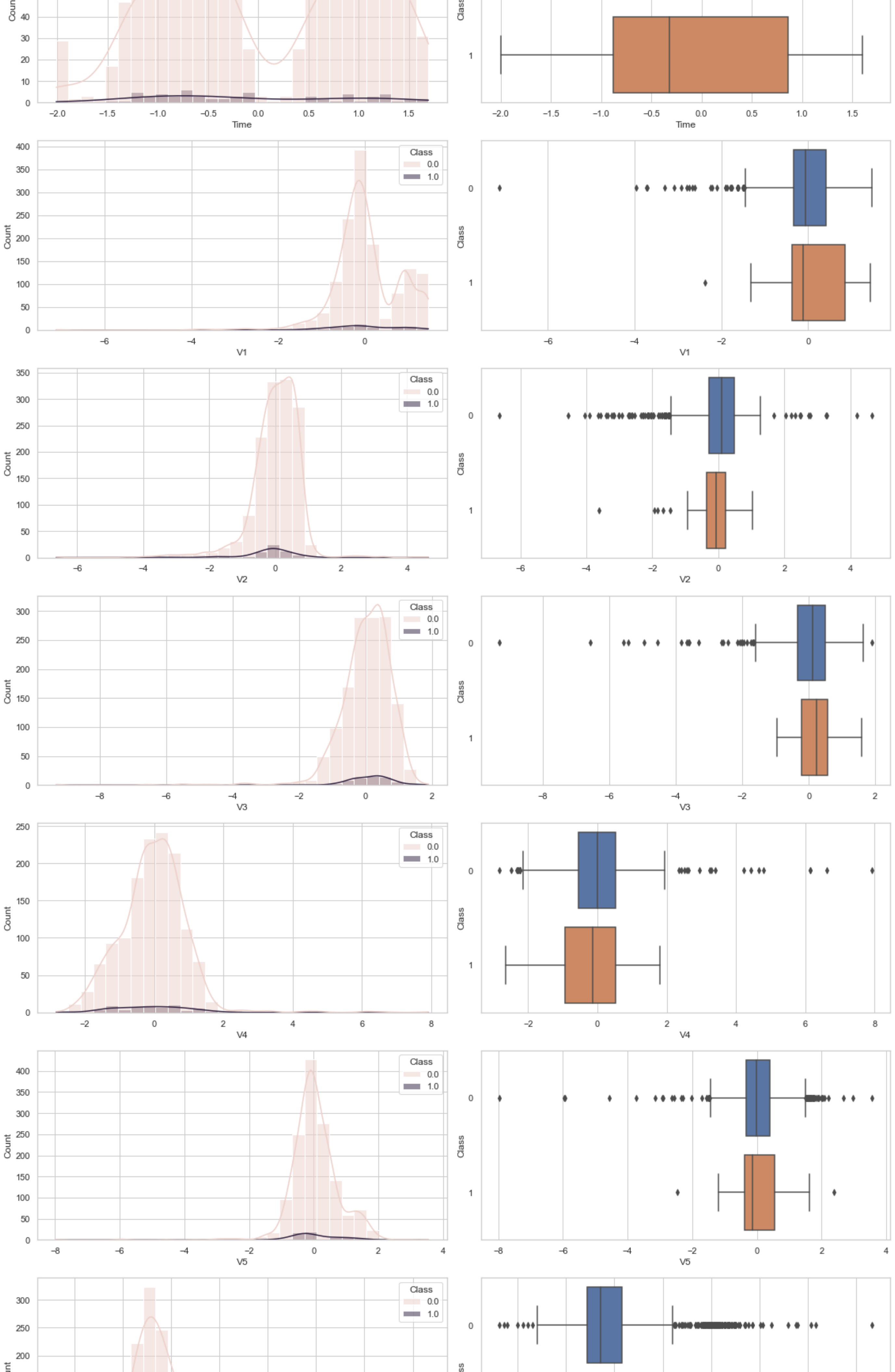
In [18]:
```
X_train_dummy = X_train.copy()
y_train_dummy = y_train.copy()
X_test_dummy = X_test.copy()
y_test_dummy = y_test.copy()
```

In [19]:
```
X = df_filter.drop(['Class'],axis=1)
y = df_filter['Class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test  = scaler.transform(X_test)
```

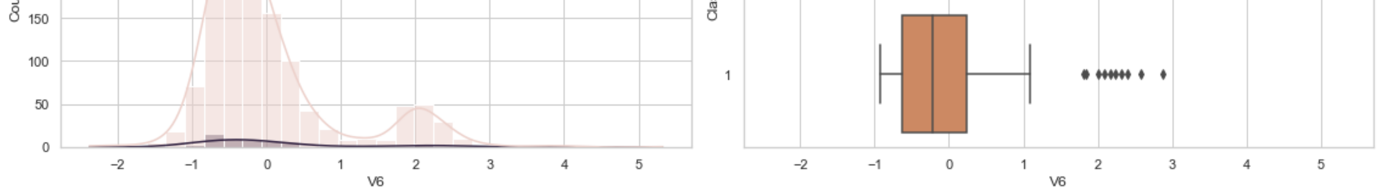
Histograms and boxplots for displaying the distribution and outliers of each feature, differentiated by the target variable with fraud or non-fraud.

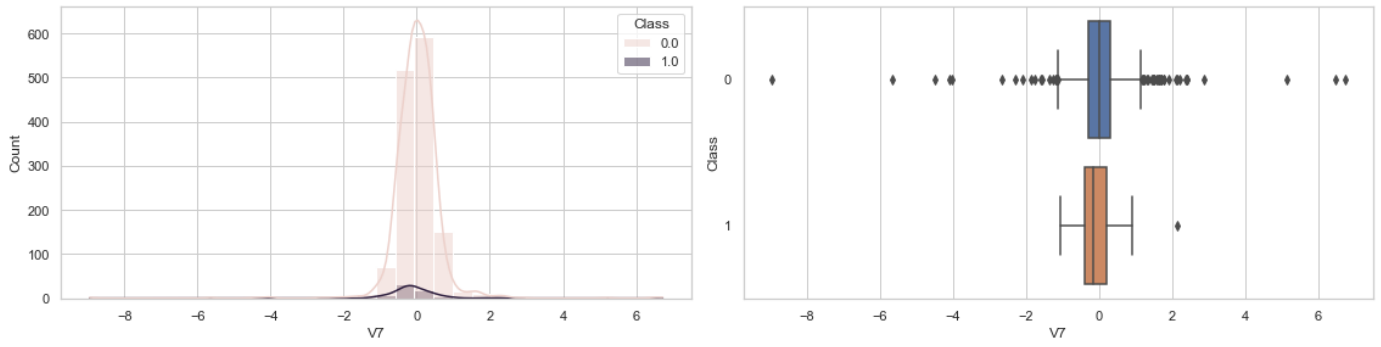
In [20]:
```
X_train_df = pd.DataFrame(X_train, columns=X.columns)
fig, axes = plt.subplots(X_train_df.shape[1], 2, figsize=(16, 4 * X_train_df.shape[1]))

for i in range(X_train_df.shape[1]):
    x = X_train_df.iloc[:, i]
    sns.histplot(x=x, bins=30, kde=True, hue=y_train, ax=axes[i, 0])
    sns.boxplot(x=x, y=y_train, orient='h', ax=axes[i, 1])

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

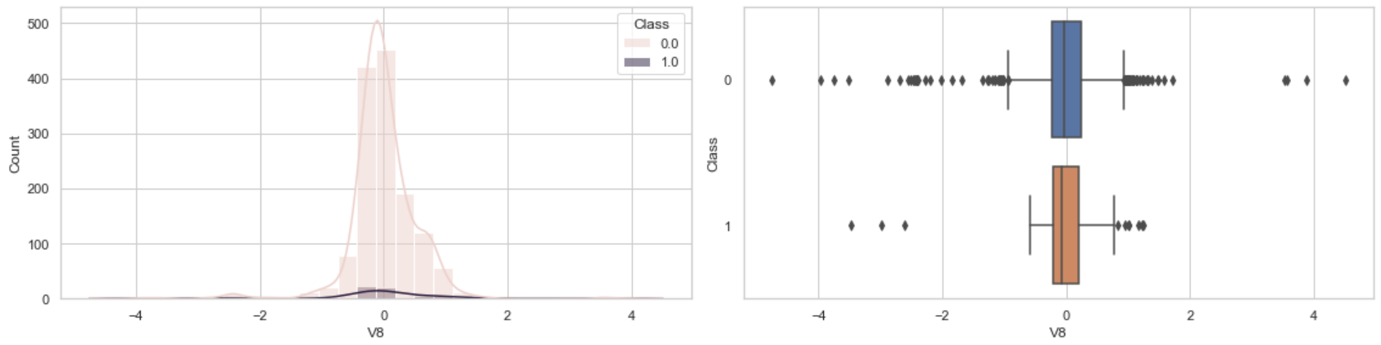


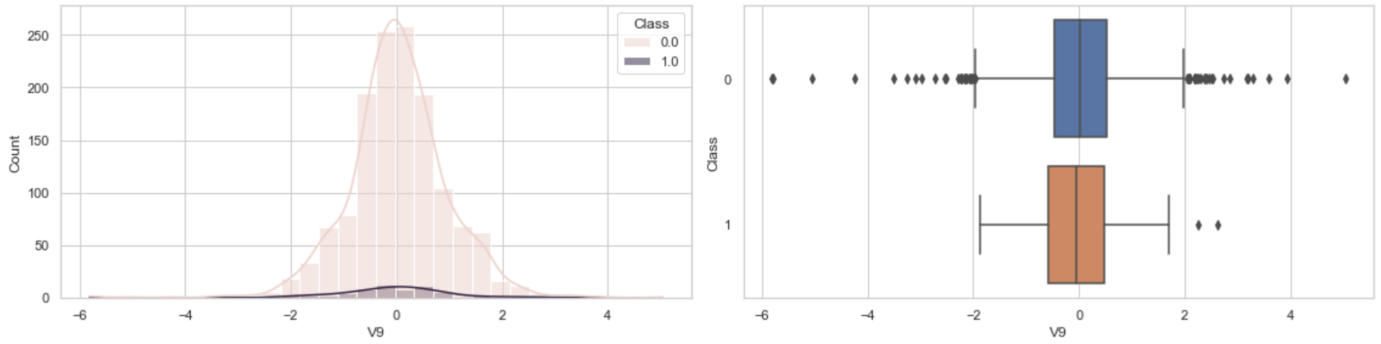


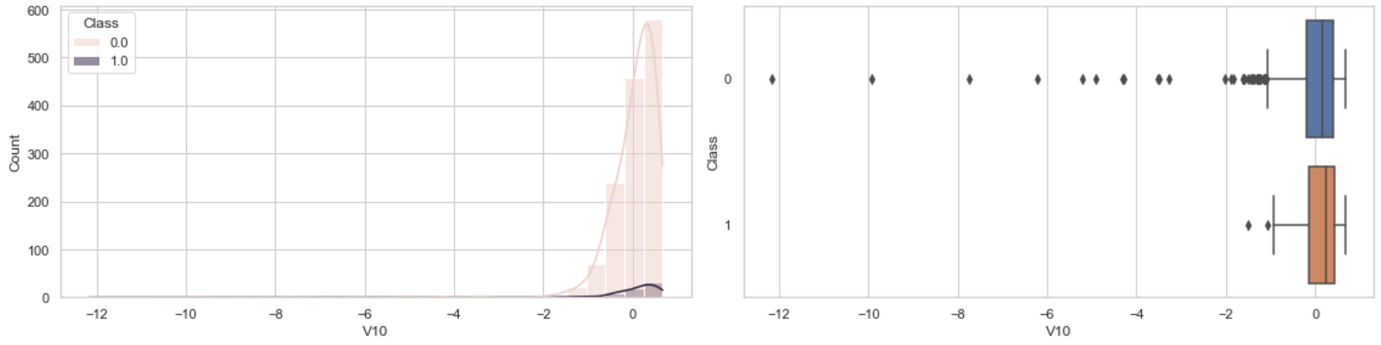
Count
Class
0.0
1.0
0
1
Time
V1
V2
V3
V4
V5

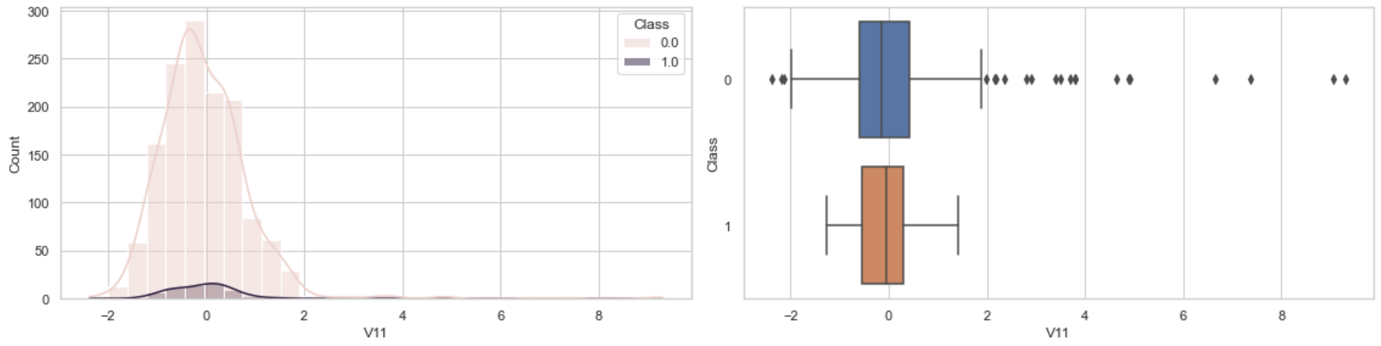

Count
V6
Class
1
V6

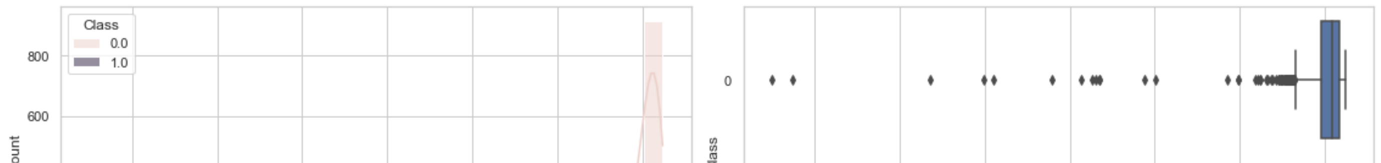

Count
Class
0.0
1.0
V7
Class
0
1
V7

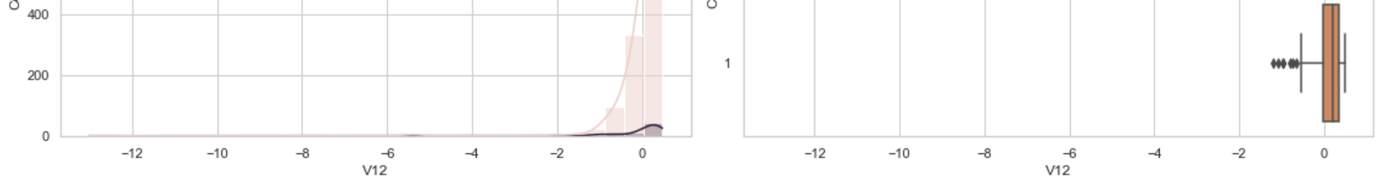

Count
Class
0.0
1.0
V8
Class
0
1
V8


Count
Class
0.0
1.0
V9
Class
0
1
V9

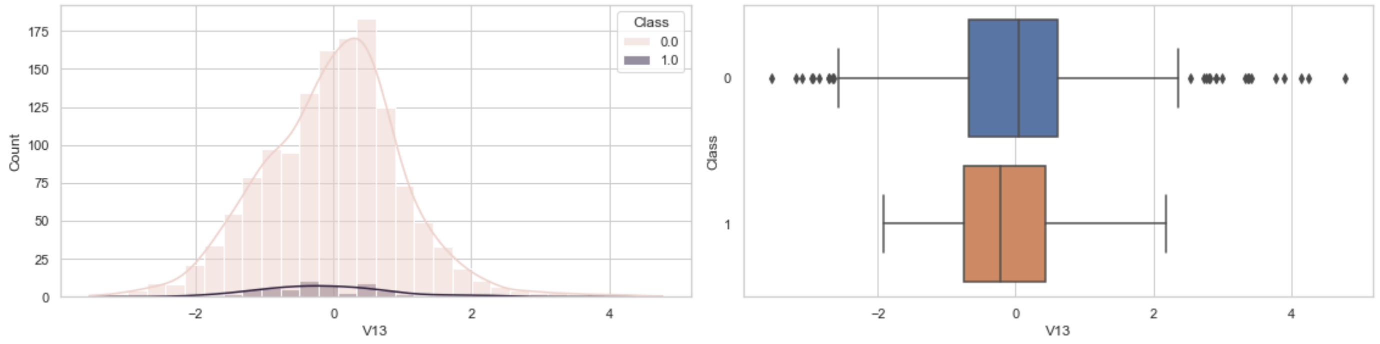

Count
Class
0.0
1.0
V10
Class
0
1
V10


Count
Class
0.0
1.0
V11
Class
0
1
V11

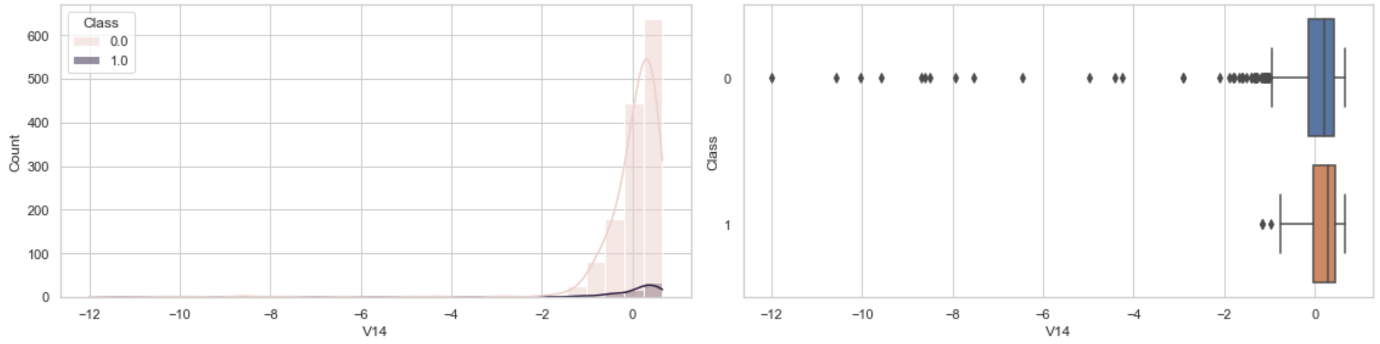

Count
Class
0.0
1.0
Class
0



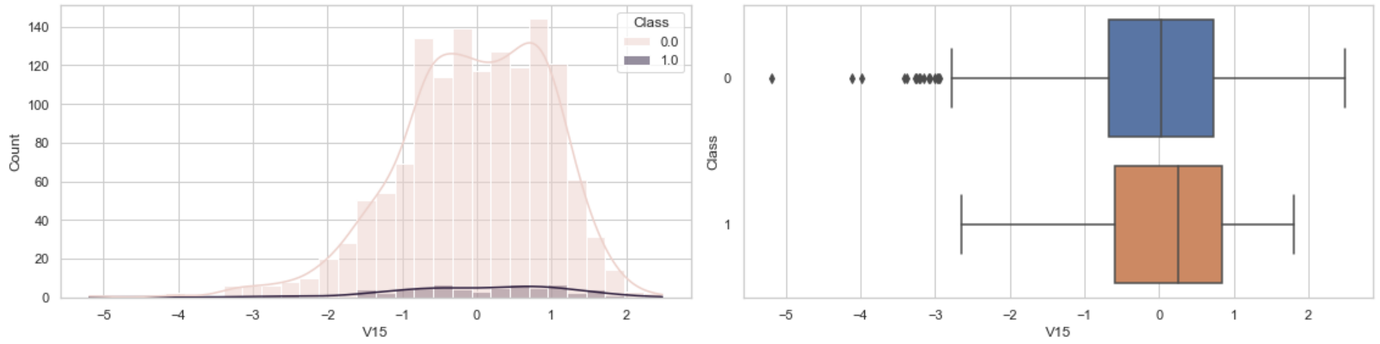
Count
V12
Class
1



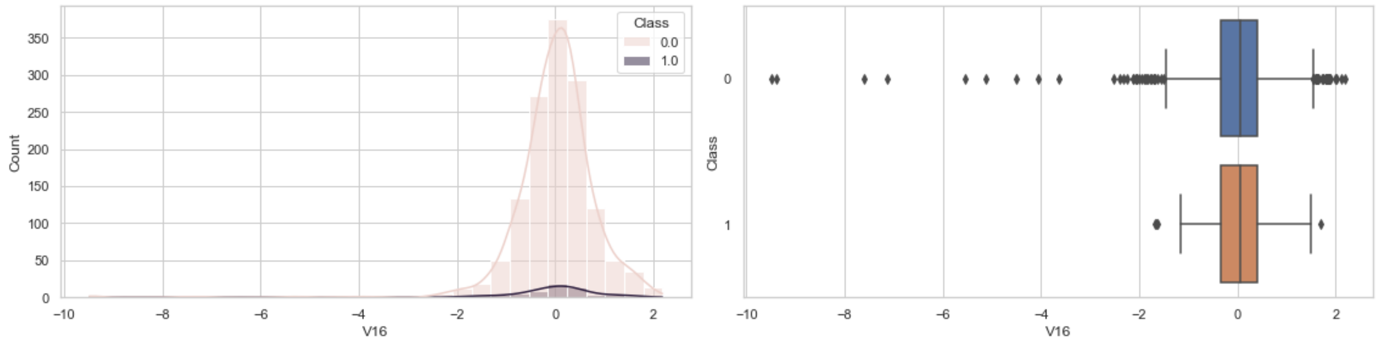
Class
0.0
1.0
Count
V13
Class
0
1



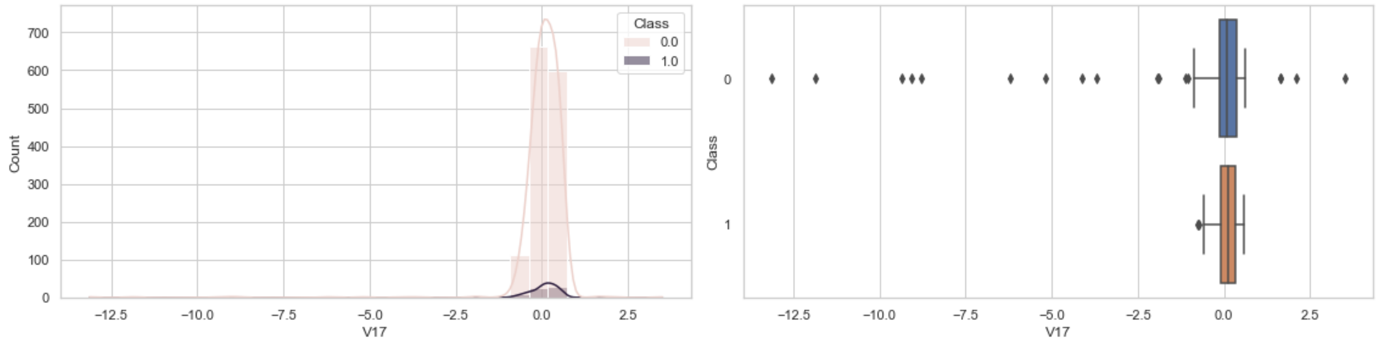
Class
0.0
1.0
Count
V14
Class
0
1



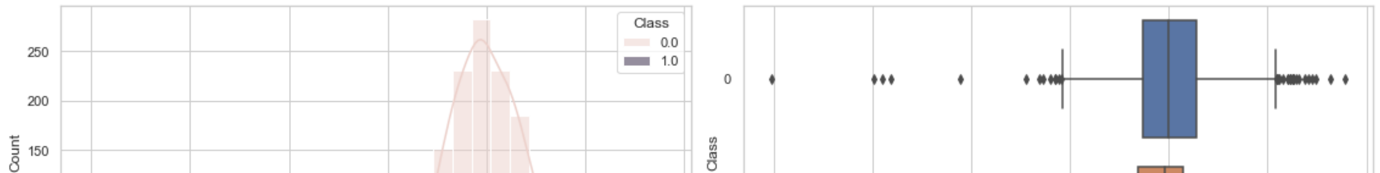
Class
0.0
1.0
Count
V15
Class
0
1



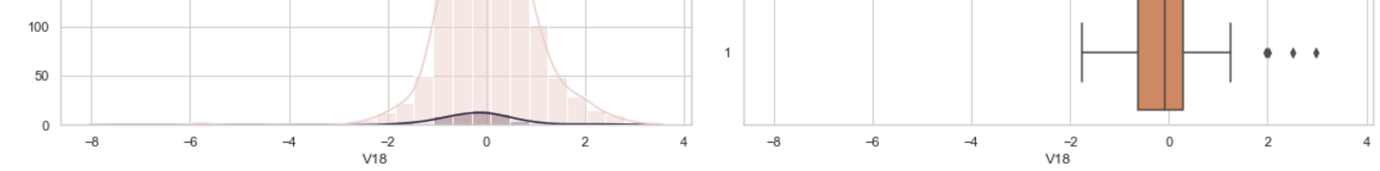
Class
0.0
1.0
Count
V16
Class
0
1

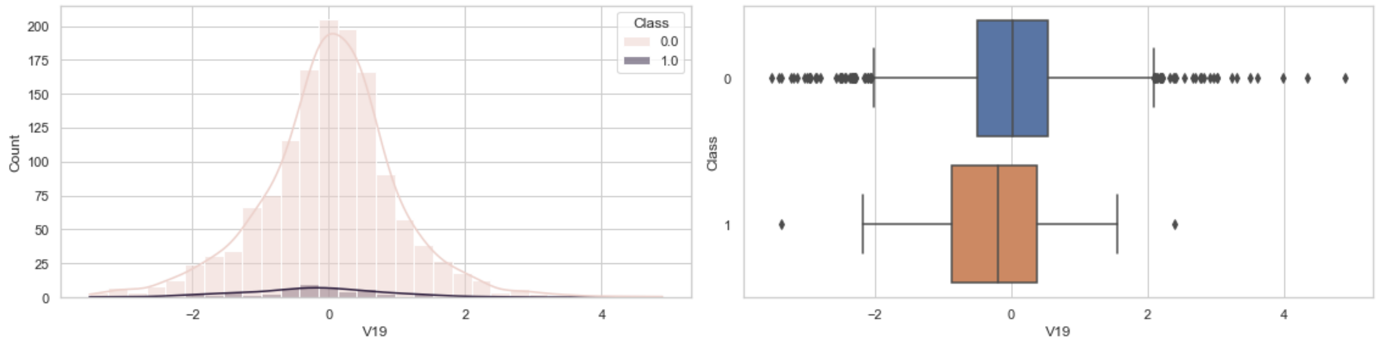


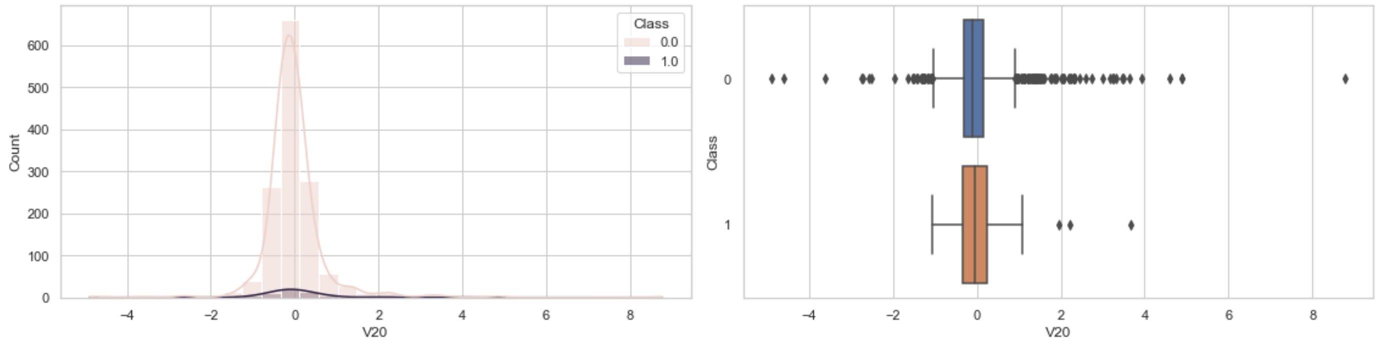
Class
0.0
1.0
Count
V17
Class
0
1

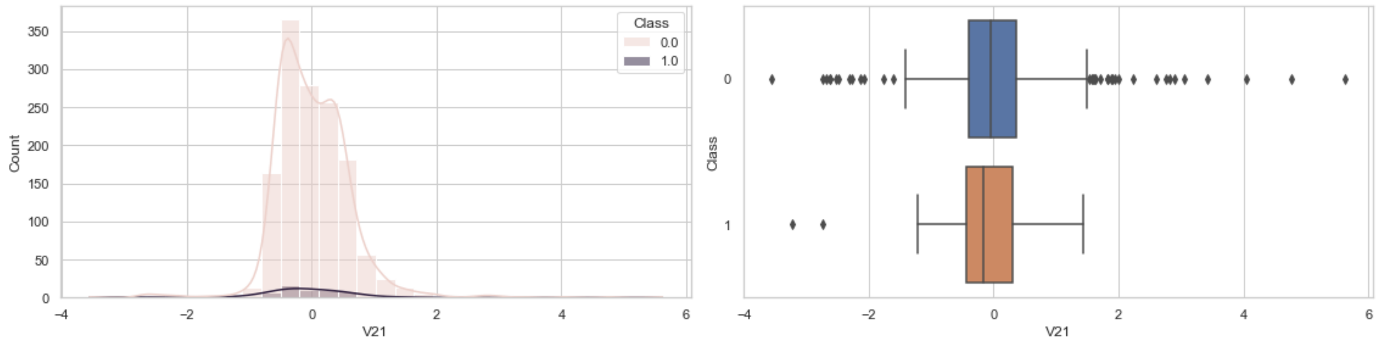


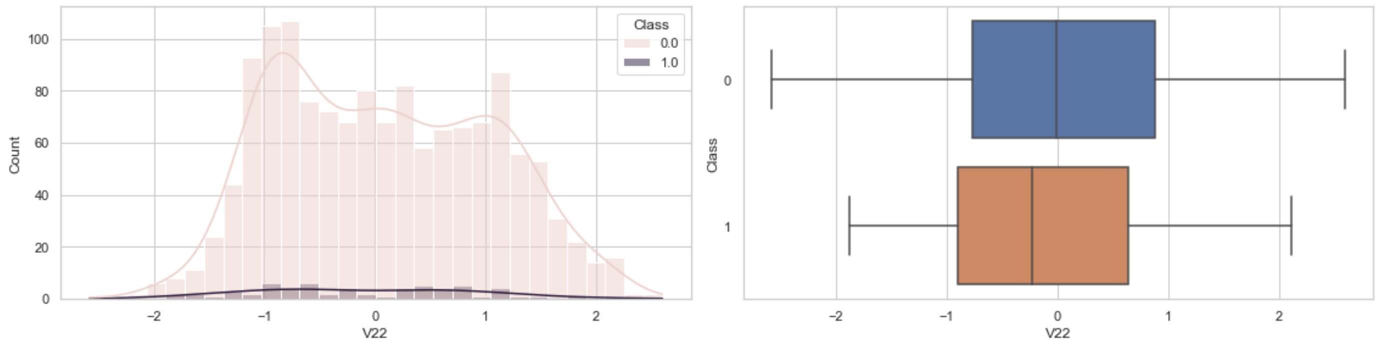
Class
0.0
1.0
Count
Class
0

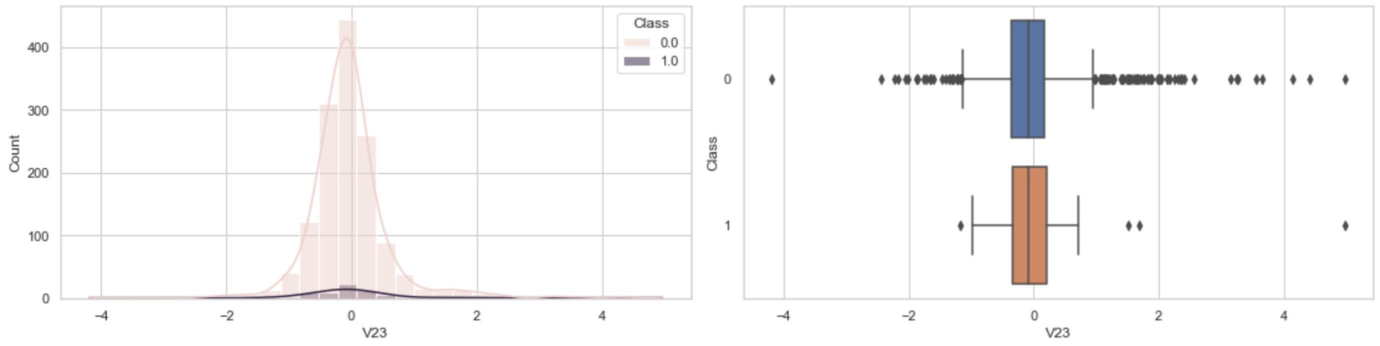

V18
V18
1

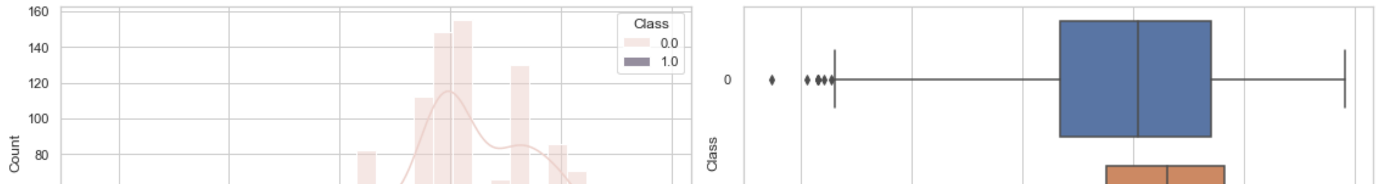

Count
V19
Class
0.0
1.0
Class
0
1
V19

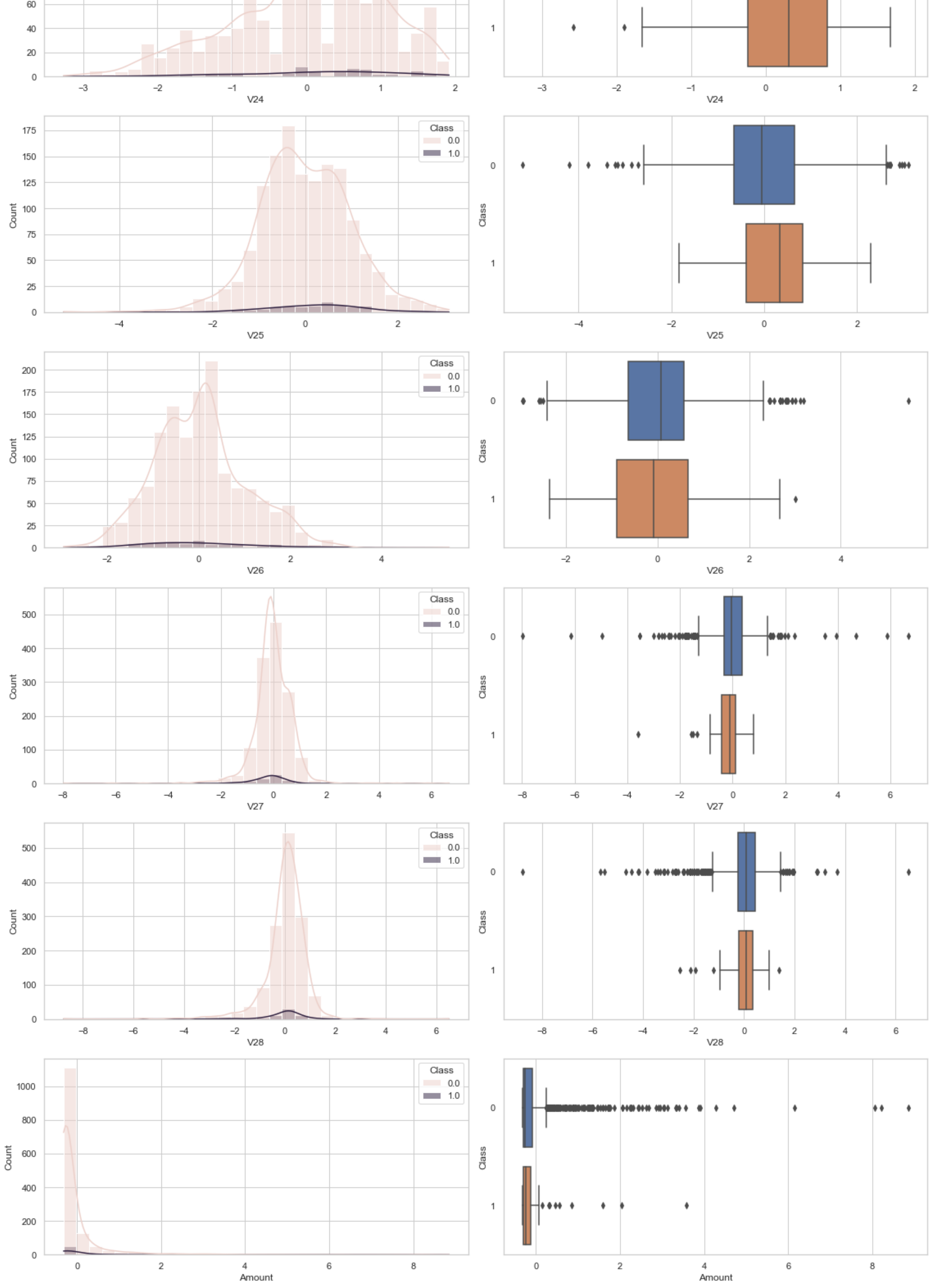

Count
V20
Class
0.0
1.0
Class
0
1
V20


Count
V21
Class
0.0
1.0
Class
0
1
V21


Count
V22
Class
0.0
1.0
Class
0
1
V22


Count
V23
Class
0.0
1.0
Class
0
1
V23


Count
Class
0.0
1.0
Class
0



Most of the datasets contain outliers; however, in our analysis, we initially proceed with the original data to understand the baseline performance and distribution characteristics before implementing outlier detection and removal techniques or other preprocessing steps

Comparison of models with scale in Orjinal Data

In [21]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test  = scaler.transform(X_test)

Models = [
    ('XGB Classifier', XGBClassifier(random_state=42)),
    ('Light GBM Classifier', LGBMClassifier(random_state=42)),
    ('Gradient Boosting Classifier', GradientBoostingClassifier(random_state=42)),
    ('Ada Boost Classifier', AdaBoostClassifier(random_state=42)),
    ('Random Forest Classifier', RandomForestClassifier(random_state=42)),
    ('Logistic Regression', LogisticRegression(random_state=42)),
    ('Support Vector Machine (SVM)', SVC(gamma='auto', random_state=42)),
    ('Decision Tree Classifier', DecisionTreeClassifier(random_state=42)),
    ('Hist Gradient Boosting Classifier', HistGradientBoostingClassifier(random_state=42
]

name_list = []
accuracy_scores = []
accuracy_scores_train = []
recall_scores = []
recall_scores_train = []
f1_scores = []
f1_scores_train = []
precision_scores = []
precision_scores_train = []

for name, model in Models:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_pred_train = model.predict(X_train)
    accuracy_scores.append(accuracy_score(y_test, y_pred))
    accuracy_scores_train.append(accuracy_score(y_train, y_pred_train))
    recall_scores.append(recall_score(y_test, y_pred))
    recall_scores_train.append(recall_score(y_train, y_pred_train))
    f1_scores.append(f1_score(y_test, y_pred))
    f1_scores_train.append(f1_score(y_train, y_pred_train))
    precision_scores.append(precision_score(y_test, y_pred))
    precision_scores_train.append(precision_score(y_train, y_pred_train))
    name_list.append(name)

result = {
    'Model': name_list,
    'Accuracy Score Test': accuracy_scores,
    'Accuracy Score Train': accuracy_scores_train,
    'Recall Score Test': recall_scores,
    'Recall Score Train': recall_scores_train,
    'F1 Score Test': f1_scores,
    'F1 Score Train': f1_scores_train,
    'Precision Score Test': precision_scores,
    'Precision Score Train': precision_scores_train
}

dataframe = pd.DataFrame(result).sort_values(by='F1 Score Test', ascending=False)
dataframe.reset_index(drop=True)
```

```
[LightGBM] [Info] Number of positive: 346, number of negative: 22819
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.
013475 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 23165, number of used feature
s: 30
```

```
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.014936 -> initscore=-4.188910
[LightGBM] [Info] Start training from score -4.188910
```

Out[21]:

	Model	Accuracy Score Test	Accuracy Score Train	Recall Score Test	Recall Score Train	F1 Score Test	F1 Score Train	Precision Score Test	Precision Score Train
0	Random Forest Classifier	0.999093	1.000000	0.972603	1.000000	0.969283	1.000000	0.965986	1.000000
1	Ada Boost Classifier	0.998489	0.999914	0.938356	0.997110	0.948097	0.997110	0.958042	0.997110
2	Gradient Boosting Classifier	0.998288	0.999957	0.952055	0.997110	0.942373	0.998553	0.932886	1.000000
3	XGB Classifier	0.997985	1.000000	0.904110	1.000000	0.929577	1.000000	0.956522	1.000000
4	Light GBM Classifier	0.997885	1.000000	0.897260	1.000000	0.925795	1.000000	0.956204	1.000000
5	Decision Tree Classifier	0.997784	1.000000	0.931507	1.000000	0.925170	1.000000	0.918919	1.000000
6	Hist Gradient Boosting Classifier	0.997784	0.999827	0.904110	0.991329	0.923077	0.994203	0.942857	0.997093
7	Support Vector Machine (SVM)	0.997079	0.998489	0.828767	0.910405	0.892989	0.947368	0.968000	0.987461
8	Logistic Regression	0.995770	0.995597	0.787671	0.794798	0.845588	0.843558	0.912698	0.898693

Top 2 model based on Recall score, F1 score and Precision score: XGB Classifier, Random Forest Classifier.

The comparative analysis reveals that XGB Classifier and Random Forest Classifier outperform other models in terms of accuracy, recall, F1 score, and precision, showcasing robust performance on both test and train data. Conversely, Ada Boost Classifier, SVM, Logistic Regression, and Decision Tree Classifier exhibit slightly lower recall and F1 scores, indicating potential issues with correctly identifying positive cases. Hist Gradient Boosting Classifier, Light GBM Classifier, and Gradient Boosting Classifier demonstrate suboptimal performance, especially in recall and precision, suggesting the need for further optimization or exploration of alternative models. To enhance overall model performance, focusing on hyperparameter tuning for boosting algorithms like XGB and Random Forest, and considering ensemble techniques could be beneficial. Additionally, addressing class imbalance and exploring feature engineering may improve the performance of models with lower recall and precision.

Roc Curve, Classification Report and Confusion Matrix

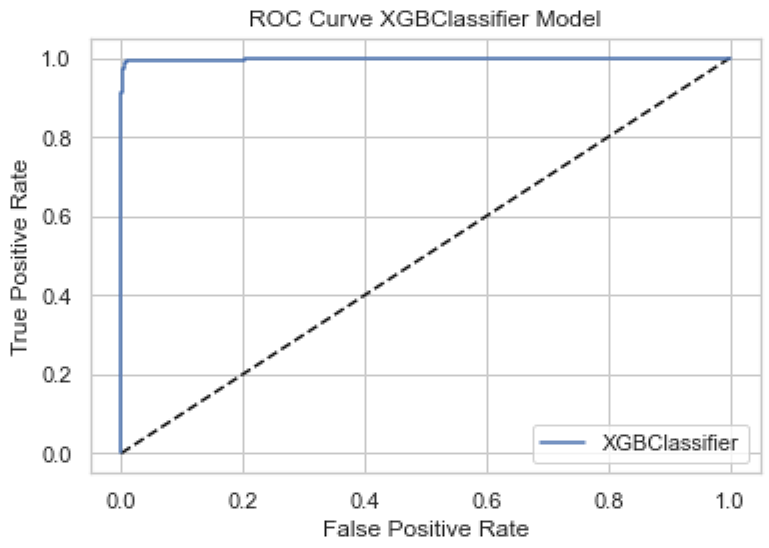
Confusion matrix: Confusion matrix gives us a clear and succinct picture of our classification model's performance in comparison to the real class labels. It functions similarly to a performance snapshot. Within the matrix, four elements are visible to us:

True Positive (TP): Examples of transactions that the model properly detected as fraudulent.
False Positive (FP): Known also as a type I error, these are situations in which the model mistakenly anticipated non-fraudulent transactions to be fraudulent.
True Negative (TN): Transactions that the model accurately classified as not fraudulent.
False Negative (FN): Also referred to as a type I error, these are situations in which the model mispredicted fraudulent transactions as non-fraudulent.

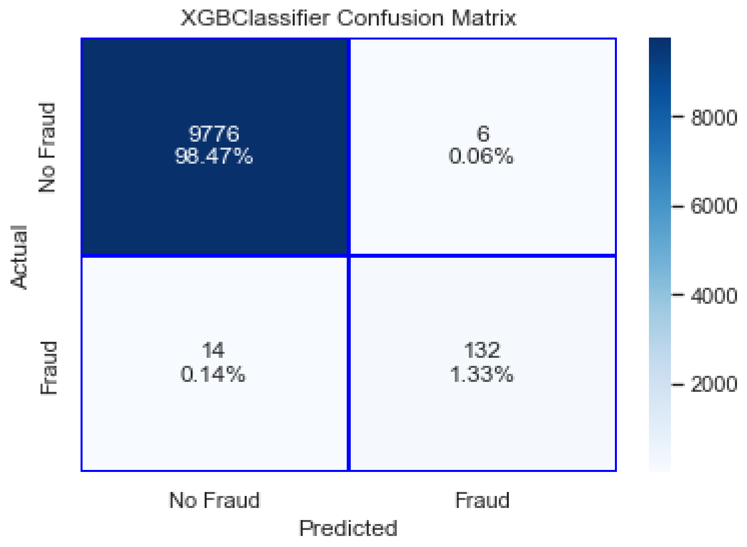
We may discover areas for improvement in our model's ability to distinguish between classes by looking at the confusion matrix, which provides insightful information.

In [22]:
```
XGB = XGBClassifier()
RF = RandomForestClassifier()
SVM = SVC(probability=True)
DTC = DecisionTreeClassifier()
LightGBM = LGBMClassifier()
GBC = GradientBoostingClassifier()
ADA = AdaBoostClassifier()
LR = LogisticRegression()
HGB = HistGradientBoostingClassifier()

Models = [XGB, RF, SVM, DTC, LightGBM, GBC, ADA, LR, HGB]

for model in Models:
    model.fit(X_train, y_train)
    y_pred_prob = model.predict_proba(X_test)[:, 1]
    threshold = 0.5
    y_pred = (y_pred_prob > threshold).astype(int)
    fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
    print('\n' )
    print(type(model).__name__+' Roc Curve, Classification Report and Confusion Matrix:\
    print(type(model).__name__, 'Model AUC Score is: ', auc(fpr, tpr))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.plot(fpr, tpr, label=type(model).__name__)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve ' + type(model).__name__ + ' Model')
    plt.legend()
    plt.show()

    # Convert probability scores to binary predictions using a threshold
    threshold = 0.5
    y_pred_binary = (y_pred_prob > threshold).astype(int)

    labelsNF = ['No Fraud', 'Fraud']
    report = classification_report(y_test, y_pred_binary, target_names=labelsNF)
    print(type(model).__name__,' Classification Report:\n', report)
    cm = confusion_matrix(y_test, y_pred_binary)

    counts = [value for value in cm.flatten()]
    percentages = ['{0:.2%}'.format(value) for value in cm.flatten() / np.sum(cm)]
    labels = [f'{v1}\n{v2}' for v1, v2 in zip(counts, percentages)]

    labels = np.array(labels).reshape(cm.shape)

    sns.heatmap(cm, annot=labels, fmt='', cmap='Blues', linewidths=1.5, linecolor='blue'
                xticklabels=labelsNF, yticklabels=labelsNF)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(type(model).__name__ + ' Confusion Matrix')
    plt.show()
```

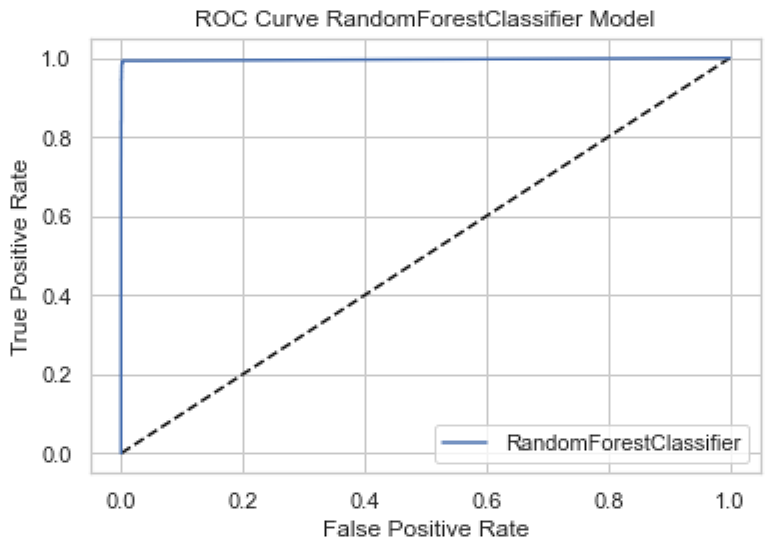
```
XGBClassifier Roc Curve, Classification Report and Confusion Matrix:

XGBClassifier Model AUC Score is:  0.9983909501096506
```



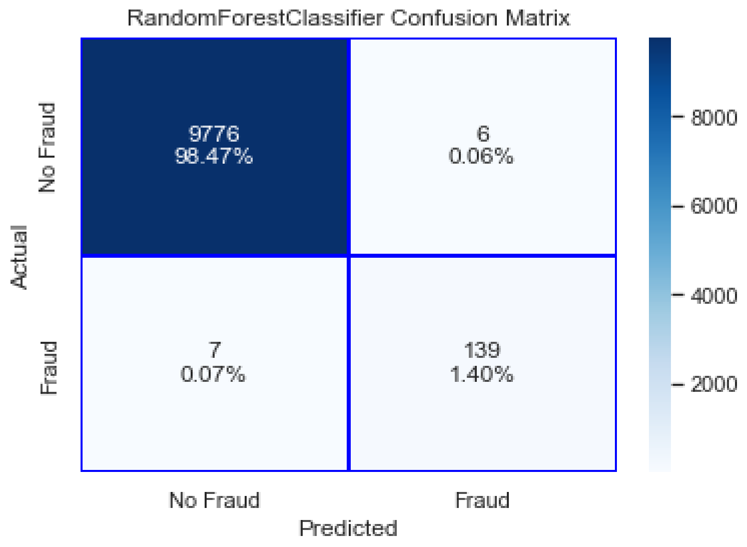
```
XGBClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9782
       Fraud       0.96      0.90      0.93       146

    accuracy                           1.00      9928
   macro avg       0.98      0.95      0.96      9928
weighted avg       1.00      1.00      1.00      9928
```



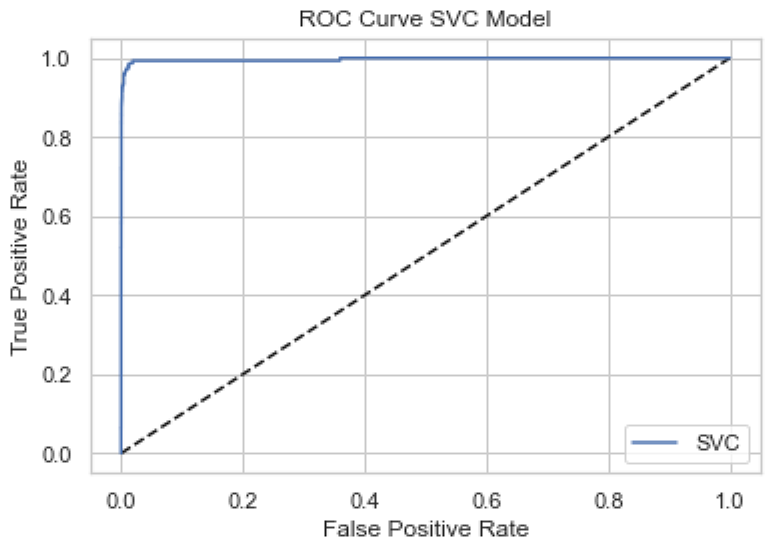
```
RandomForestClassifier Roc Curve, Classification Report and Confusion Matrix:

RandomForestClassifier Model AUC Score is:  0.9963414770769907
```



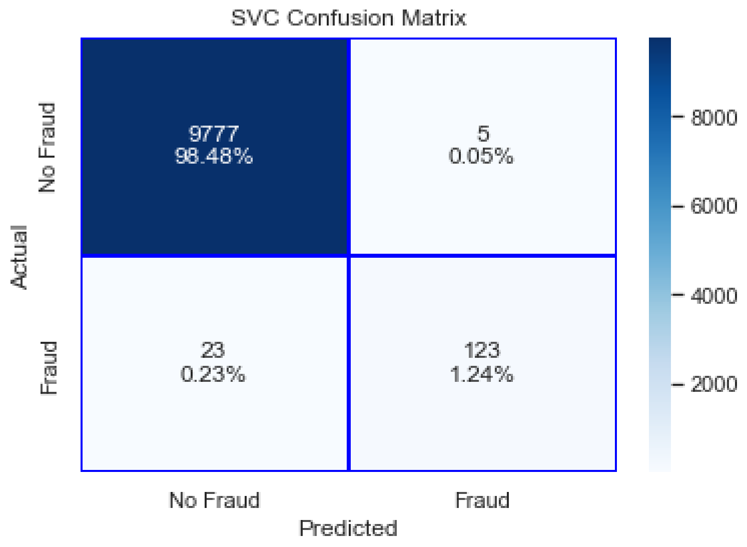
```
RandomForestClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9782
       Fraud       0.96      0.95      0.96       146

    accuracy                           1.00      9928
   macro avg       0.98      0.98      0.98      9928
weighted avg       1.00      1.00      1.00      9928
```



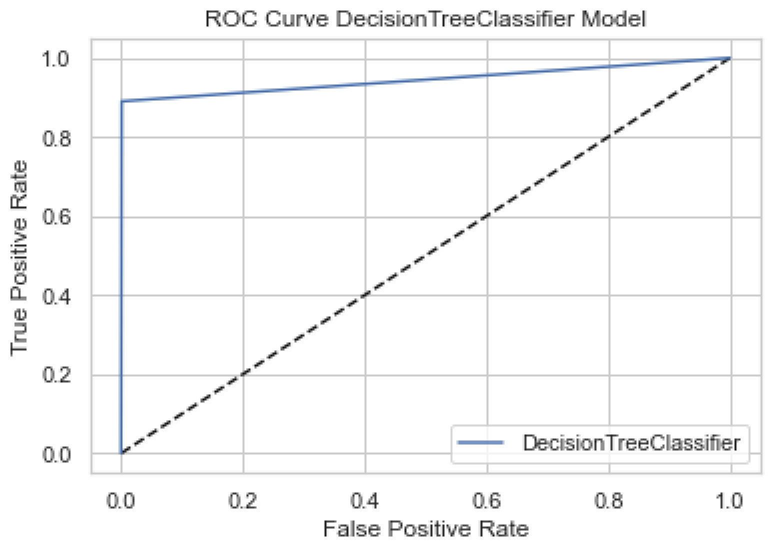
```
SVC Roc Curve, Classification Report and Confusion Matrix:

SVC Model AUC Score is:  0.996775948555216
```



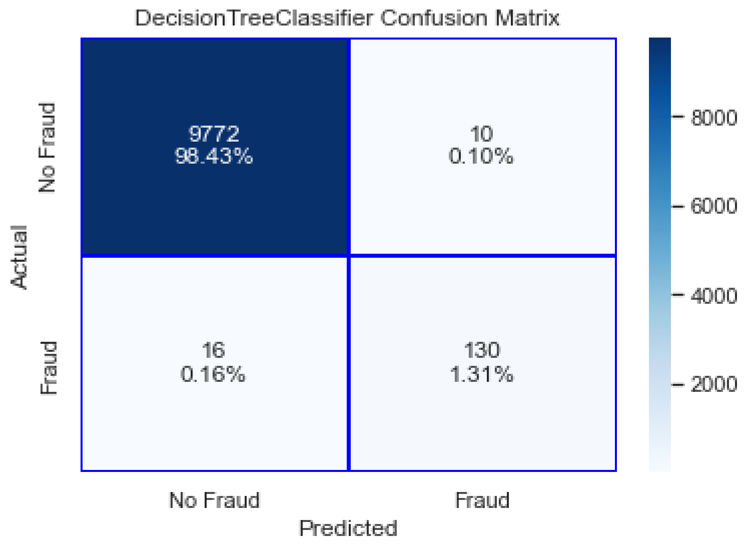
```
SVC  Classification Report:
                precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9782
       Fraud       0.96      0.84      0.90       146

    accuracy                           1.00      9928
   macro avg       0.98      0.92      0.95      9928
weighted avg       1.00      1.00      1.00      9928
```



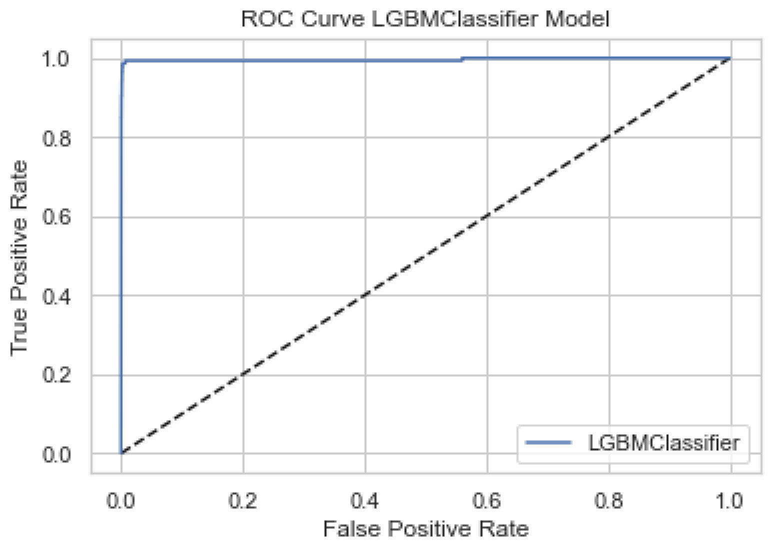
```
DecisionTreeClassifier Roc Curve, Classification Report and Confusion Matrix:

DecisionTreeClassifier Model AUC Score is:  0.9446943365364956
```



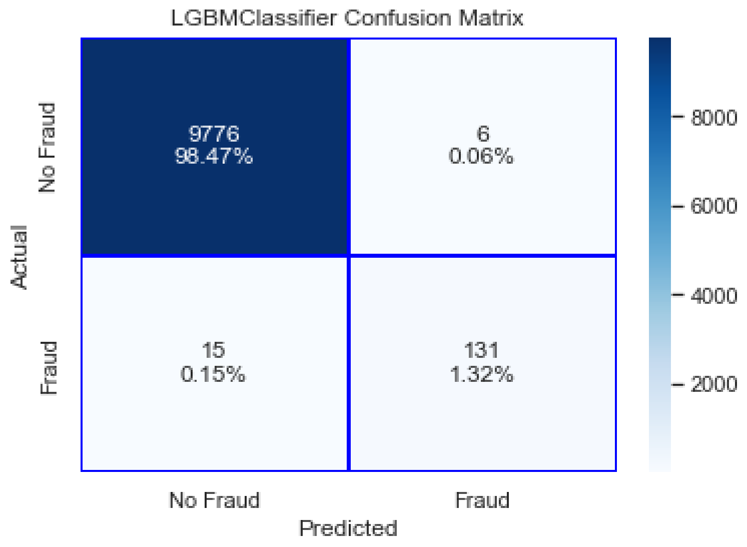
```
DecisionTreeClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9782
       Fraud       0.93      0.89      0.91       146

    accuracy                           1.00      9928
   macro avg       0.96      0.94      0.95      9928
weighted avg       1.00      1.00      1.00      9928
```



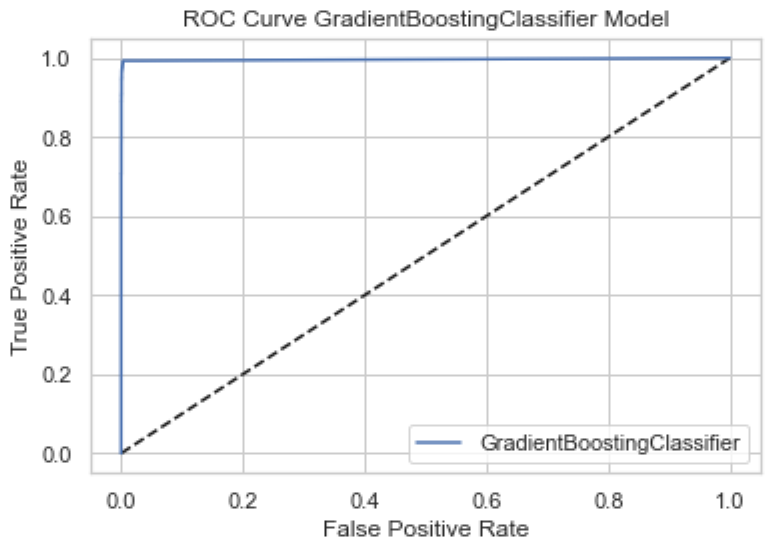
```
[LightGBM] [Info] Number of positive: 346, number of negative: 22819
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.
011973 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 23165, number of used feature
s: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.014936 -> initscore=-4.188910
[LightGBM] [Info] Start training from score -4.188910


LGBMClassifier Roc Curve, Classification Report and Confusion Matrix:

LGBMClassifier Model AUC Score is:  0.995954969009335
```



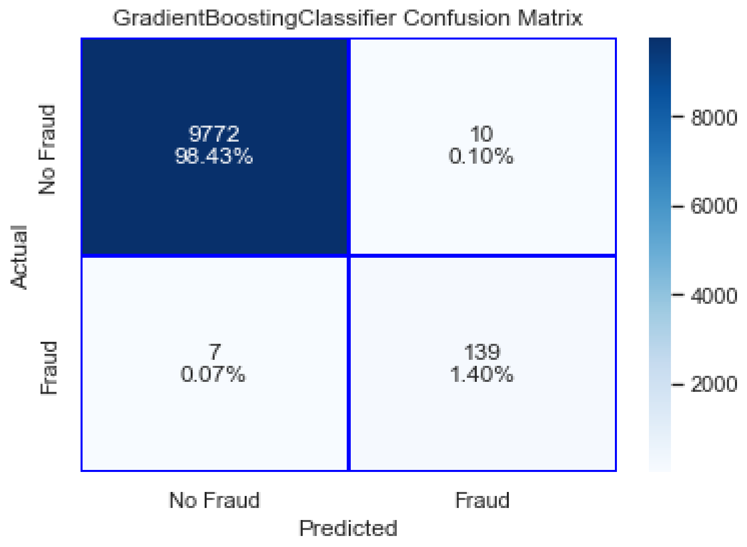
```
LGBMClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9782
       Fraud       0.96      0.90      0.93       146

    accuracy                           1.00      9928
   macro avg       0.98      0.95      0.96      9928
weighted avg       1.00      1.00      1.00      9928
```



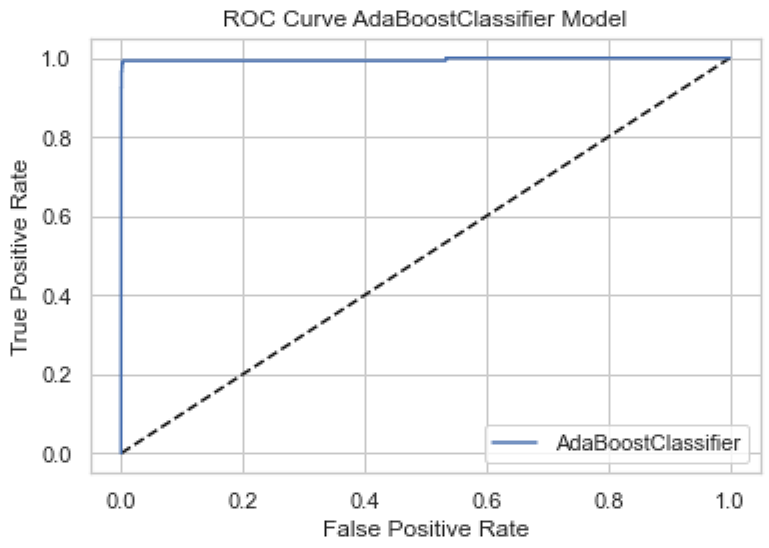
```
GradientBoostingClassifier Roc Curve, Classification Report and Confusion Matrix:

GradientBoostingClassifier Model AUC Score is:  0.9962077396840157
```



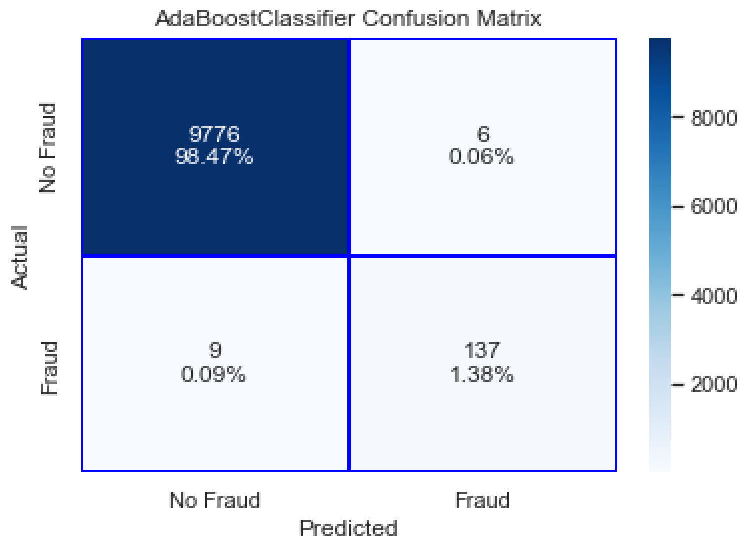
```
GradientBoostingClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9782
       Fraud       0.93      0.95      0.94       146

    accuracy                           1.00      9928
   macro avg       0.97      0.98      0.97      9928
weighted avg       1.00      1.00      1.00      9928
```



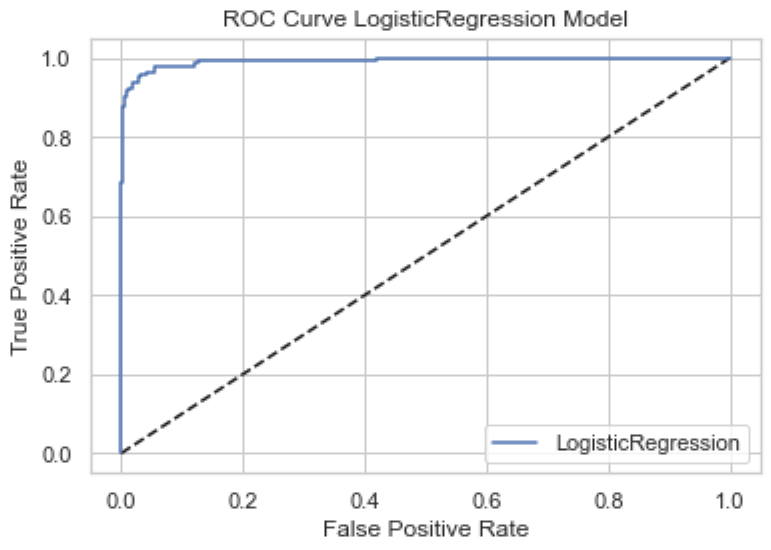
```
AdaBoostClassifier Roc Curve, Classification Report and Confusion Matrix:

AdaBoostClassifier Model AUC Score is:  0.9962049389009167
```



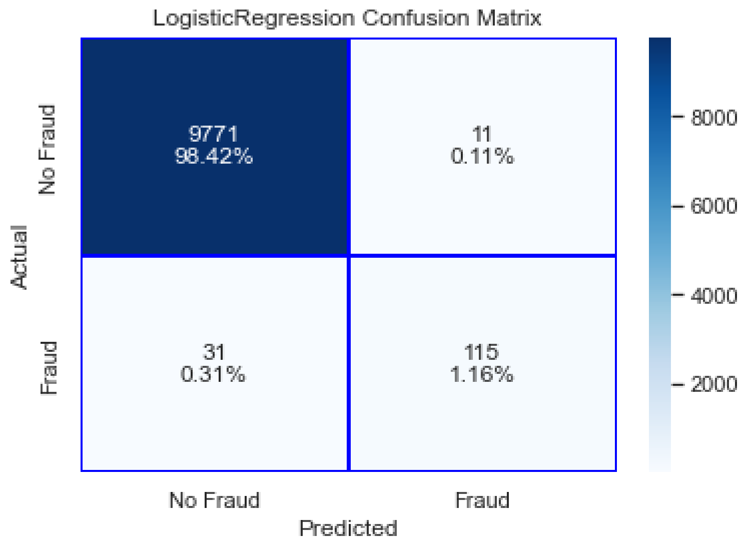
```
AdaBoostClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9782
       Fraud       0.96      0.94      0.95       146

    accuracy                           1.00      9928
   macro avg       0.98      0.97      0.97      9928
weighted avg       1.00      1.00      1.00      9928
```



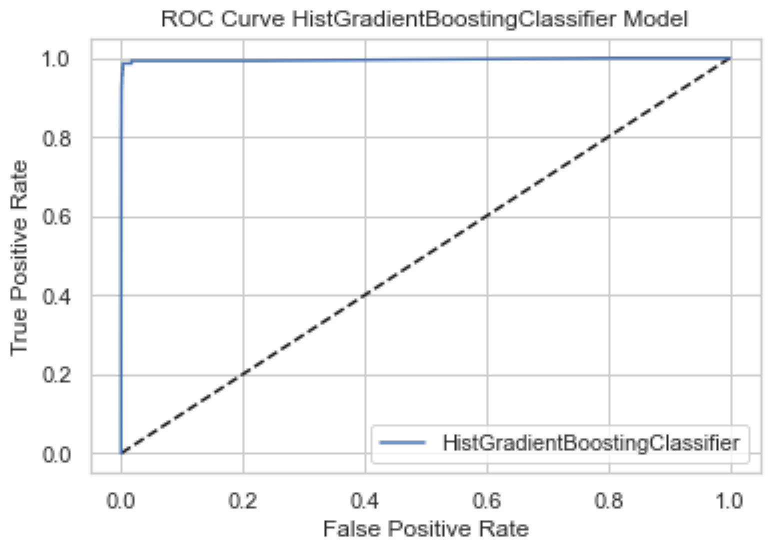
```
LogisticRegression Roc Curve, Classification Report and Confusion Matrix:

LogisticRegression Model AUC Score is:  0.9929406261990854
```



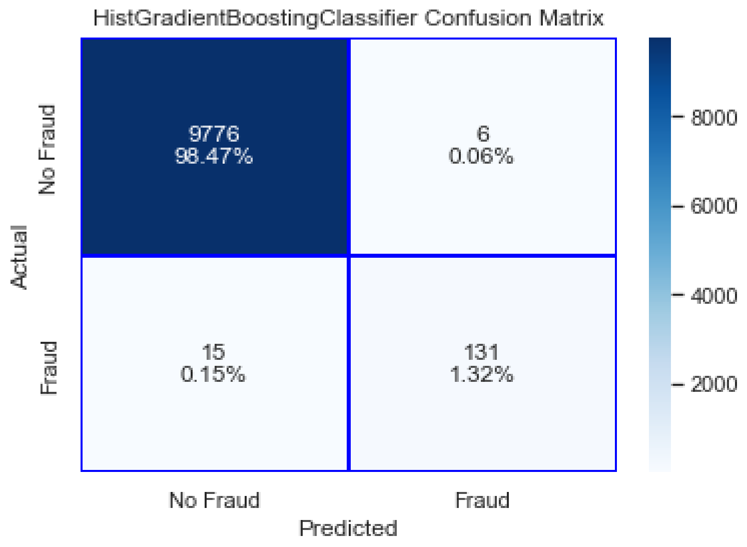
```
LogisticRegression  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9782
       Fraud       0.91      0.79      0.85       146

    accuracy                           1.00      9928
   macro avg       0.95      0.89      0.92      9928
weighted avg       1.00      1.00      1.00      9928
```



```
HistGradientBoostingClassifier Roc Curve, Classification Report and Confusion Matrix:

HistGradientBoostingClassifier Model AUC Score is:  0.9962728578910665
```



```
HistGradientBoostingClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9782
       Fraud       0.96      0.90      0.93       146

    accuracy                           1.00      9928
   macro avg       0.98      0.95      0.96      9928
weighted avg       1.00      1.00      1.00      9928
```



Top 2 model based on ROC-AUC score: XGB Classifier, AdaBoost Classifier

After setting the threshold at 0.5, there are discernible shifts in model performance. XGB Classifier, Random Forest Classifier, and Ada Boost Classifier maintain high precision and recall, indicating effective fraud detection.
However, SVC and Logistic Regression show slightly decreased recall, potentially impacting fraud identification accuracy. Decision Tree Classifier and LGBM Classifier improve recall, suggesting better fraud detection. Conversely, Gradient Boosting Classifier and Hist Gradient Boosting Classifier struggle with recall, indicating challenges in fraud identification.
Optimizing threshold values could enhance models with lower recall, ensuring accurate fraud detection while minimizing false positives. Adjusting thresholds tailored to specific model strengths could further optimize fraud detection performance.

Comparison of the models with scaling in df_filter

```
In [23]: X = df_filter.drop(['Class'],axis=1)
         y = df_filter['Class']

         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42
         scaler = StandardScaler()
         X_train = scaler.fit_transform(X_train)
         X_test  = scaler.transform(X_test)


         Models = [
             ('XGB Classifier', XGBClassifier(random_state=42)),
             ('Light GBM Classifier', LGBMClassifier(random_state=42)),
             ('Gradient Boosting Classifier', GradientBoostingClassifier(random_state=42)),
             ('Ada Boost Classifier', AdaBoostClassifier(random_state=42)),
             ('Random Forest Classifier', RandomForestClassifier(random_state=42)),
             ('Logistic Regression', LogisticRegression(random_state=42)),
             ('Support Vector Machine (SVM)', SVC(gamma='auto', random_state=42)),
             ('Decision Tree Classifier', DecisionTreeClassifier(random_state=42)),
             ('Hist Gradient Boosting Classifier', HistGradientBoostingClassifier(random_state=42
         ]

         name_list = []
         accuracy_scores = []
         accuracy_scores_train = []
         recall_scores = []
         recall_scores_train = []
         f1_scores = []
         f1_scores_train = []
         precision_scores = []
         precision_scores_train = []

         for name, model in Models:
             model.fit(X_train, y_train)
             y_pred = model.predict(X_test)
             y_pred_train = model.predict(X_train)
             accuracy_scores.append(accuracy_score(y_test, y_pred))
             accuracy_scores_train.append(accuracy_score(y_train, y_pred_train))
             recall_scores.append(recall_score(y_test, y_pred))
             recall_scores_train.append(recall_score(y_train, y_pred_train))
             f1_scores.append(f1_score(y_test, y_pred))
             f1_scores_train.append(f1_score(y_train, y_pred_train))
             precision_scores.append(precision_score(y_test, y_pred))
             precision_scores_train.append(precision_score(y_train, y_pred_train))
             name_list.append(name)

         result = {
             'Model': name_list,
             'Accuracy Score Test': accuracy_scores,
             'Accuracy Score Train': accuracy_scores_train,
             'Recall Score Test': recall_scores,
             'Recall Score Train': recall_scores_train,
             'F1 Score Test': f1_scores,
             'F1 Score Train': f1_scores_train,
             'Precision Score Test': precision_scores,
             'Precision Score Train': precision_scores_train
         }

         dataframe = pd.DataFrame(result).sort_values(by='F1 Score Test', ascending=False)
         dataframe.reset_index(drop=True)
```

```
[LightGBM] [Info] Number of positive: 346, number of negative: 22819
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.
011529 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
```

```
[LightGBM] [Info] Number of data points in the train set: 23165, number of used feature
s: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.014936 -> initscore=-4.188910
[LightGBM] [Info] Start training from score -4.188910
```

Out[23]:

	Model	Accuracy Score Test	Accuracy Score Train	Recall Score Test	Recall Score Train	F1 Score Test	F1 Score Train	Precision Score Test	Precision Score Train
0	Random Forest Classifier	0.999093	1.000000	0.972603	1.000000	0.969283	1.000000	0.965986	1.000000
1	Ada Boost Classifier	0.998489	0.999914	0.938356	0.997110	0.948097	0.997110	0.958042	0.997110
2	Gradient Boosting Classifier	0.998288	0.999957	0.952055	0.997110	0.942373	0.998553	0.932886	1.000000
3	XGB Classifier	0.997985	1.000000	0.904110	1.000000	0.929577	1.000000	0.956522	1.000000
4	Light GBM Classifier	0.997885	1.000000	0.897260	1.000000	0.925795	1.000000	0.956204	1.000000
5	Decision Tree Classifier	0.997784	1.000000	0.931507	1.000000	0.925170	1.000000	0.918919	1.000000
6	Hist Gradient Boosting Classifier	0.997784	0.999827	0.904110	0.991329	0.923077	0.994203	0.942857	0.997093
7	Support Vector Machine (SVM)	0.997079	0.998489	0.828767	0.910405	0.892989	0.947368	0.968000	0.987461
8	Logistic Regression	0.995770	0.995597	0.787671	0.794798	0.845588	0.843558	0.912698	0.898693

Top 2 model based on Recall score, F1 score and Precision score: XGB Classifier, Random Forest Classifier

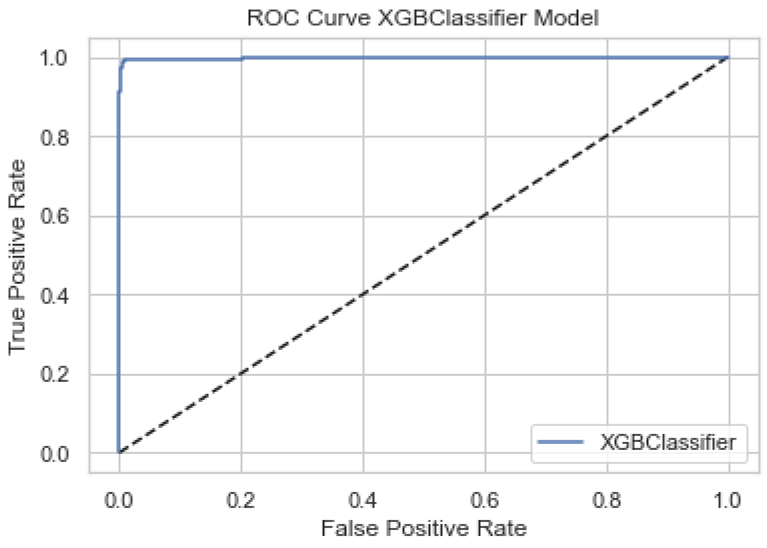
This results depict enhanced performance across various classifiers compared to the previous outcomes. Notably, Random Forest, AdaBoost, and Gradient Boosting classifiers exhibit improved accuracy, recall, F1, and precision scores, suggesting better fraud detection capabilities.
Conversely, while SVM and Logistic Regression maintain high accuracy, their recall, F1, and precision scores demonstrate slight declines, implying potential challenges in correctly identifying fraudulent transactions.
The decision to employ Random Forest, AdaBoost, or Gradient Boosting classifiers could be advantageous for accurate fraud detection, considering their balanced performance metrics.

Roc Curve, Classification Report and Confusion Matrix

In [24]:

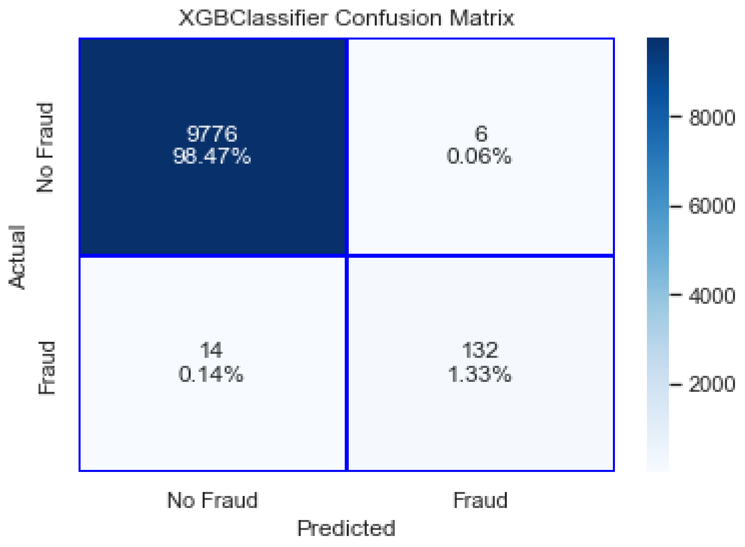
```
XGB = XGBClassifier()
RF = RandomForestClassifier()
SVM = SVC(probability=True)
DTC = DecisionTreeClassifier()
LightGBM = LGBMClassifier()
GBC = GradientBoostingClassifier()
ADA = AdaBoostClassifier()
LR = LogisticRegression()
HGB = HistGradientBoostingClassifier()

Models = [XGB, RF, SVM, DTC, LightGBM, GBC, ADA, LR, HGB]

for model in Models:
    model.fit(X_train, y_train)
    y_pred_prob = model.predict_proba(X_test)[:, 1]
    threshold = 0.5
    y_pred = (y_pred_prob > threshold).astype(int)
    fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
    print('\n' )
    print(type(model).__name__+' Roc Curve, Classification Report and Confusion Matrix:\
    print(type(model).__name__, 'Model AUC Score is: ', auc(fpr, tpr))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.plot(fpr, tpr, label=type(model).__name__)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve ' + type(model).__name__ + ' Model')
    plt.legend()
    plt.show()

    # Convert probability scores to binary predictions using a threshold
    threshold = 0.5
    y_pred_binary = (y_pred_prob > threshold).astype(int)

    labelsNF = ['No Fraud', 'Fraud']
    report = classification_report(y_test, y_pred_binary, target_names=labelsNF)
    print(type(model).__name__,' Classification Report:\n', report)

    cm = confusion_matrix(y_test, y_pred_binary)

    counts = [value for value in cm.flatten()]
    percentages = ['{0:.2%}'.format(value) for value in cm.flatten() / np.sum(cm)]
    labels = [f'{v1}\n{v2}' for v1, v2 in zip(counts, percentages)]

    labels = np.array(labels).reshape(cm.shape)

    sns.heatmap(cm, annot=labels, fmt='', cmap='Blues', linewidths=1.5, linecolor='blue'
                xticklabels=labelsNF, yticklabels=labelsNF)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(type(model).__name__ + ' Confusion Matrix')
    plt.show()
```

```
XGBClassifier Roc Curve, Classification Report and Confusion Matrix:

XGBClassifier Model AUC Score is:  0.9983909501096506
```

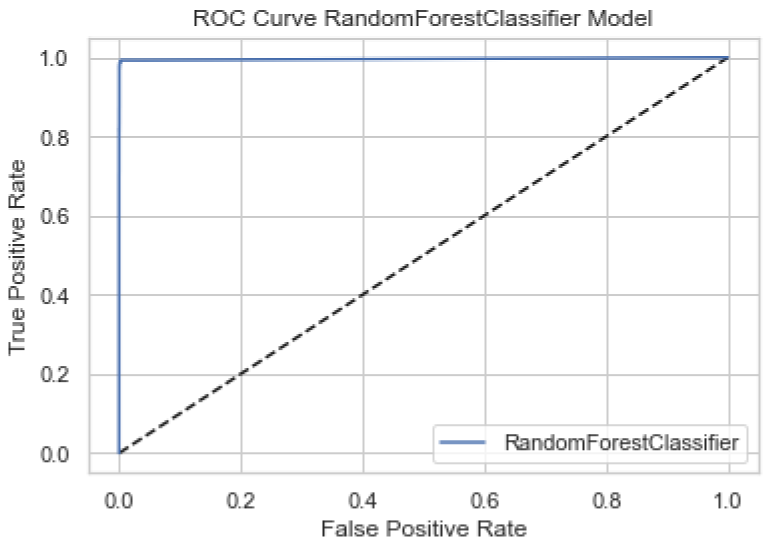


```
XGBClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9782
       Fraud       0.96      0.90      0.93       146

    accuracy                           1.00      9928
   macro avg       0.98      0.95      0.96      9928
```

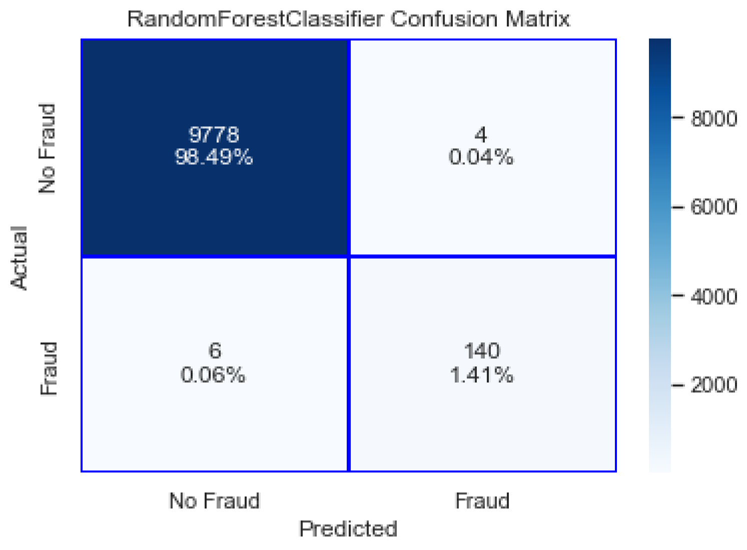
```
weighted avg       1.00      1.00      1.00      9928
```



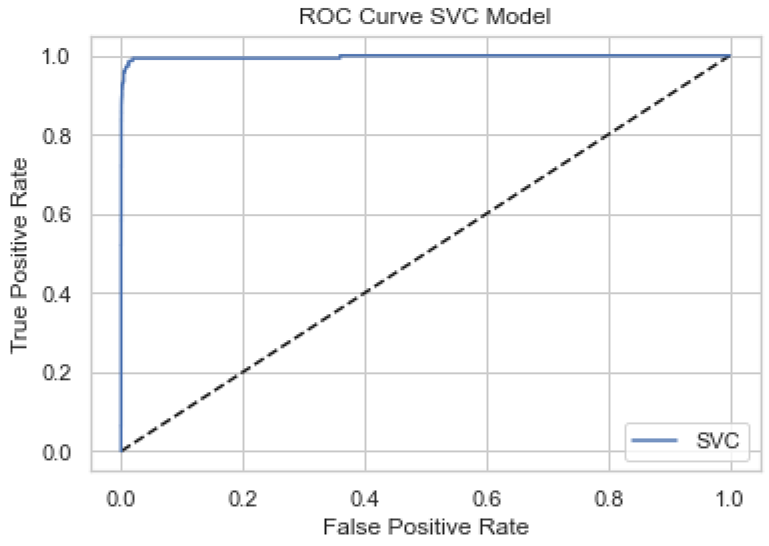
```
RandomForestClassifier Roc Curve, Classification Report and Confusion Matrix:

RandomForestClassifier Model AUC Score is:  0.9963432275664276
```



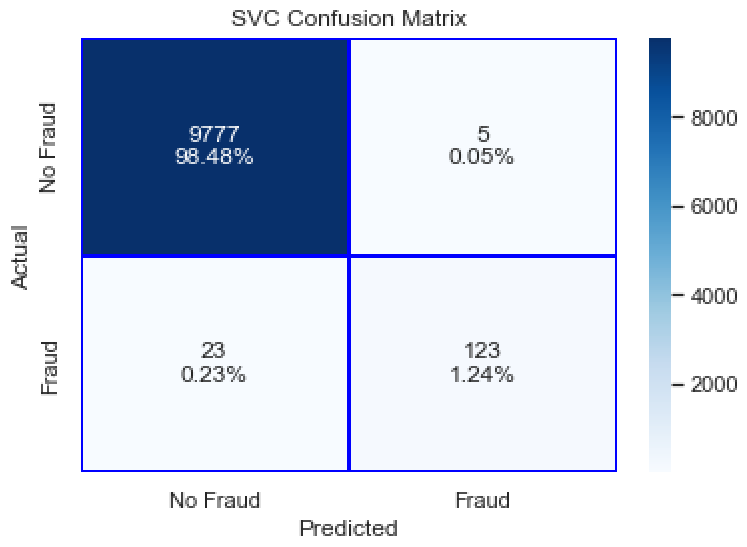
```
RandomForestClassifier  Classification Report:
              precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9782
       Fraud       0.97      0.96      0.97       146

    accuracy                           1.00      9928
   macro avg       0.99      0.98      0.98      9928
weighted avg       1.00      1.00      1.00      9928
```



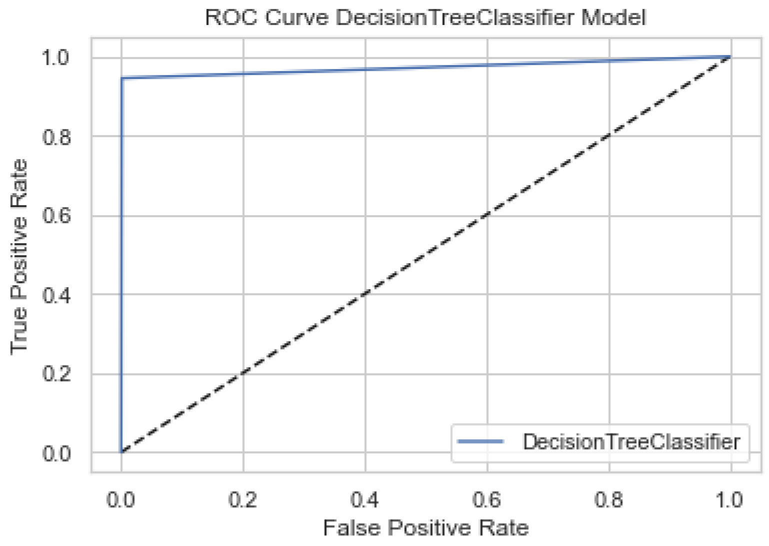
```
SVC Roc Curve, Classification Report and Confusion Matrix:

SVC Model AUC Score is:  0.9967755984573285
```



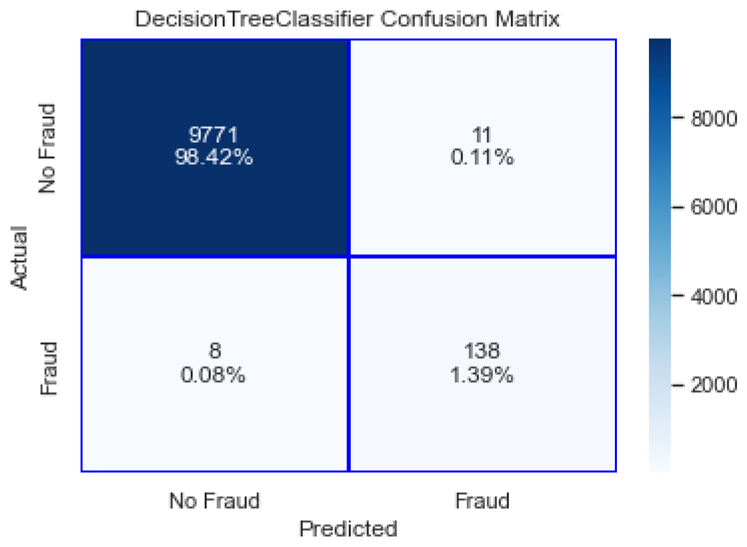
```
SVC  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9782
       Fraud       0.96      0.84      0.90       146

    accuracy                           1.00      9928
   macro avg       0.98      0.92      0.95      9928
weighted avg       1.00      1.00      1.00      9928
```



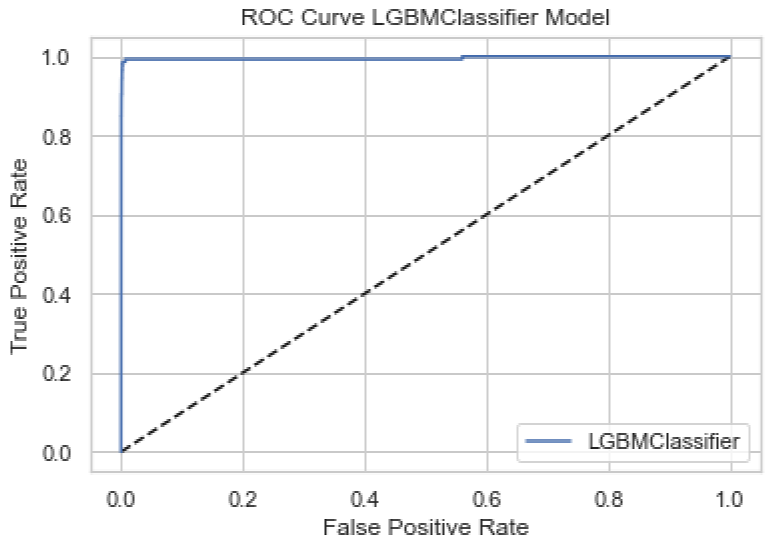
```
DecisionTreeClassifier Roc Curve, Classification Report and Confusion Matrix:

DecisionTreeClassifier Model AUC Score is:  0.9720404825189123
```



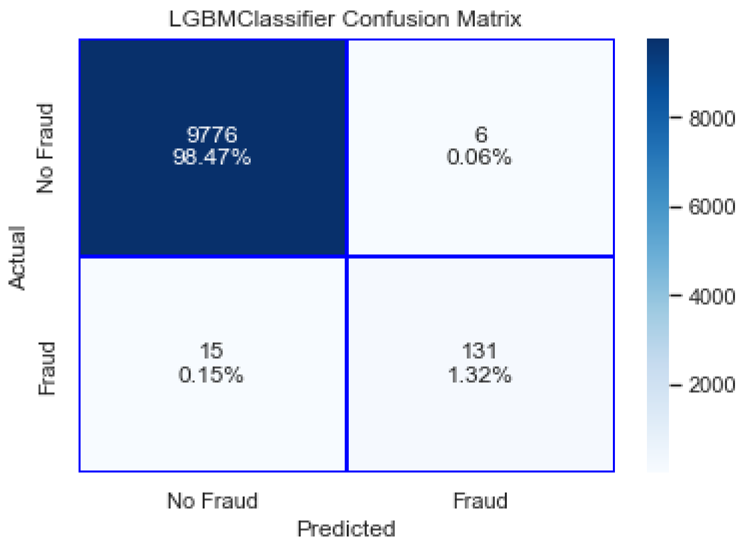
```
DecisionTreeClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9782
       Fraud       0.93      0.95      0.94       146

    accuracy                           1.00      9928
   macro avg       0.96      0.97      0.97      9928
weighted avg       1.00      1.00      1.00      9928
```



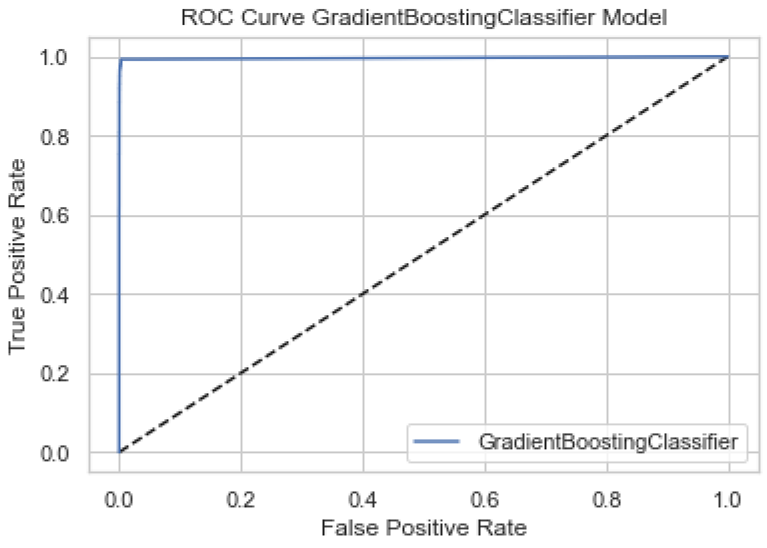
```
[LightGBM] [Info] Number of positive: 346, number of negative: 22819
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.
013409 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 23165, number of used feature
s: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.014936 -> initscore=-4.188910
[LightGBM] [Info] Start training from score -4.188910


LGBMClassifier Roc Curve, Classification Report and Confusion Matrix:

LGBMClassifier Model AUC Score is:  0.995954969009335
```



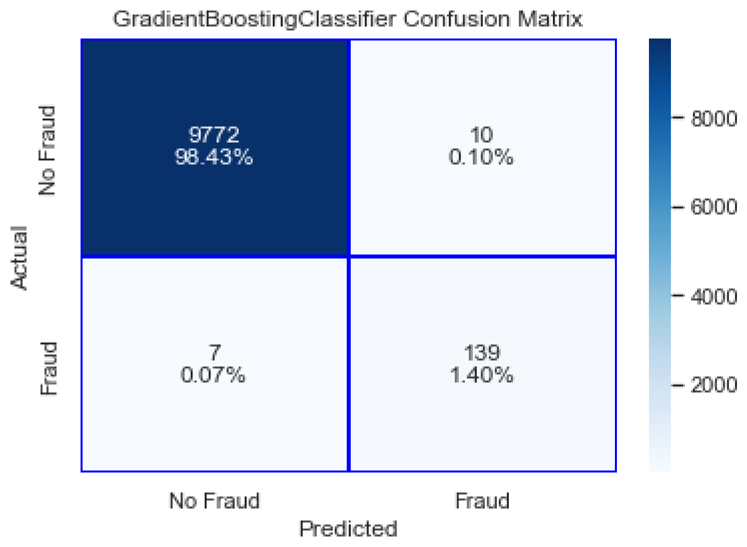
```
LGBMClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9782
       Fraud       0.96      0.90      0.93       146

    accuracy                           1.00      9928
   macro avg       0.98      0.95      0.96      9928
weighted avg       1.00      1.00      1.00      9928
```



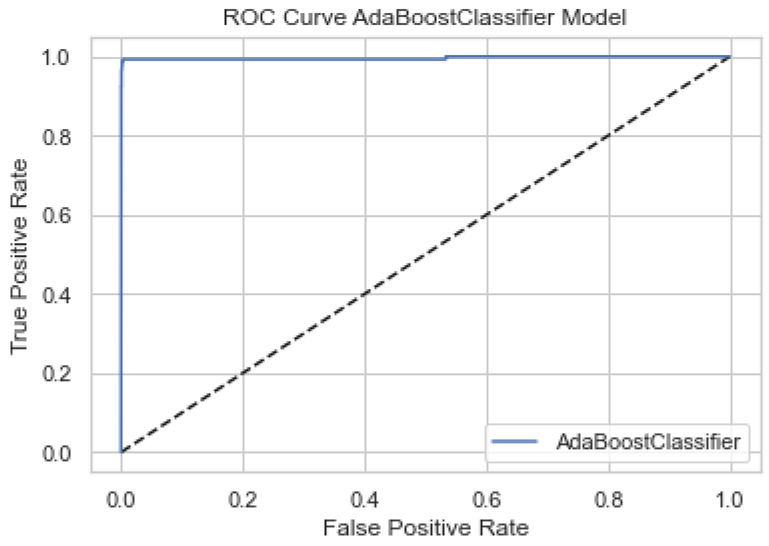
```
GradientBoostingClassifier Roc Curve, Classification Report and Confusion Matrix:

GradientBoostingClassifier Model AUC Score is:  0.9962084398797904
```



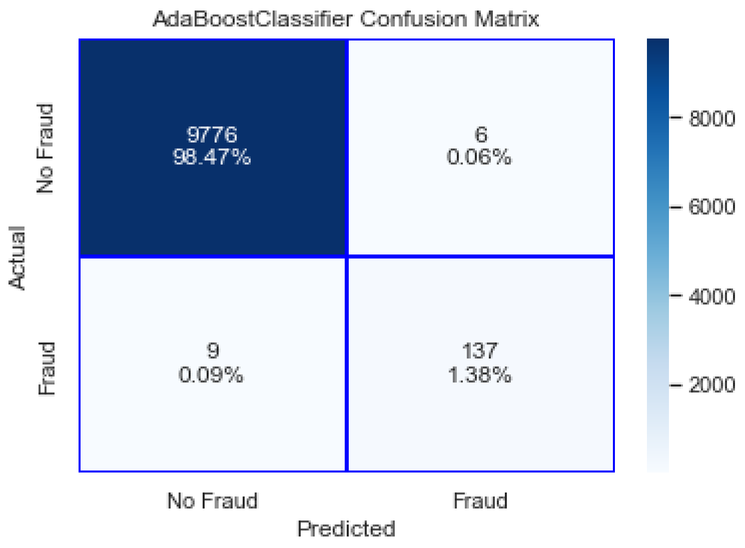
```
GradientBoostingClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9782
       Fraud       0.93      0.95      0.94       146

    accuracy                           1.00      9928
   macro avg       0.97      0.98      0.97      9928
weighted avg       1.00      1.00      1.00      9928
```



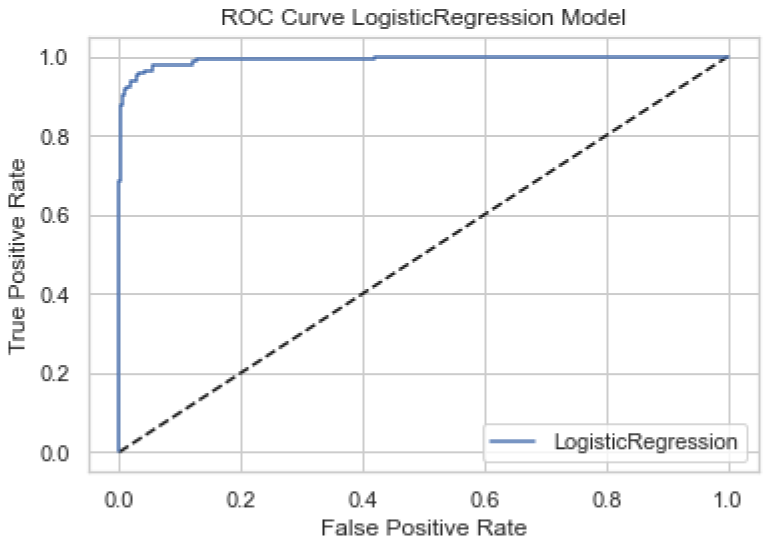
```
AdaBoostClassifier Roc Curve, Classification Report and Confusion Matrix:

AdaBoostClassifier Model AUC Score is:  0.9962049389009167
```



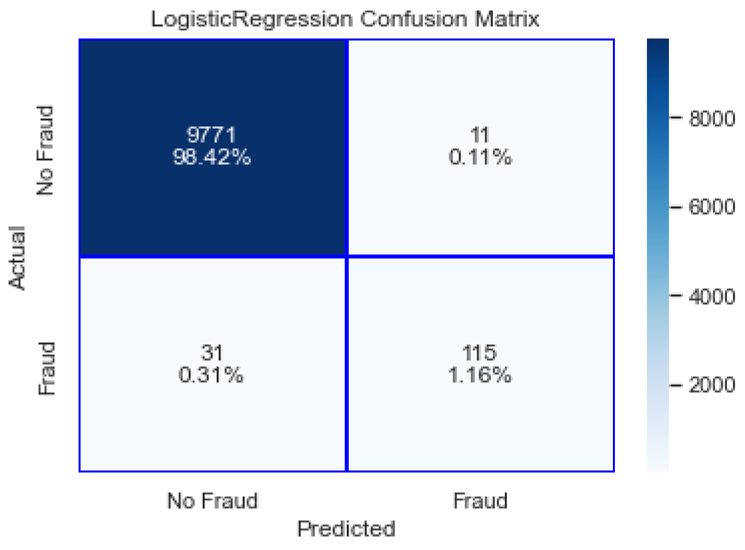
```
AdaBoostClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9782
       Fraud       0.96      0.94      0.95       146

    accuracy                           1.00      9928
   macro avg       0.98      0.97      0.97      9928
weighted avg       1.00      1.00      1.00      9928
```



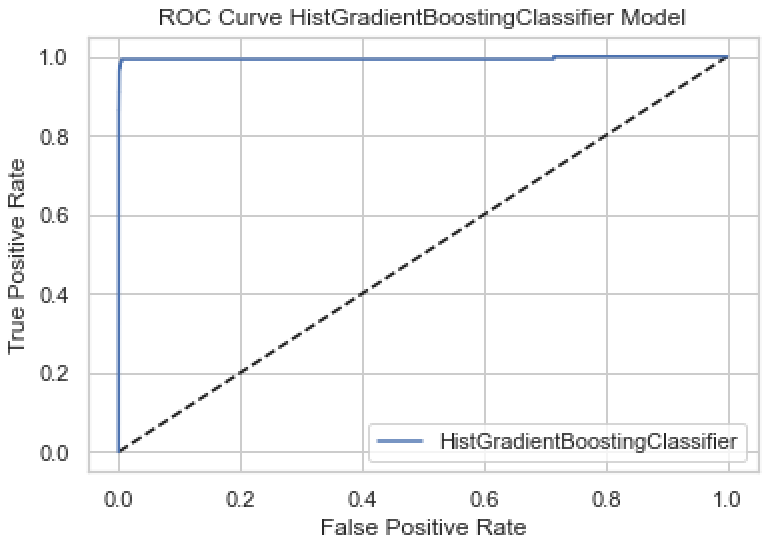
```
LogisticRegression Roc Curve, Classification Report and Confusion Matrix:

LogisticRegression Model AUC Score is:  0.9929406261990854
```



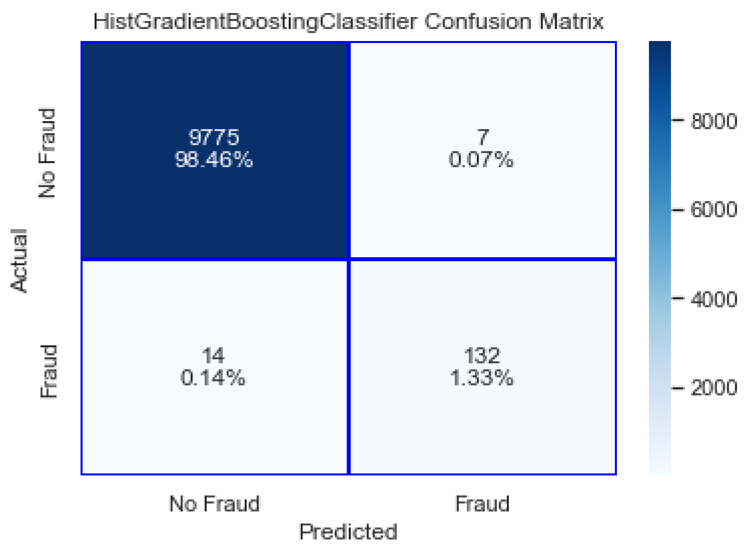
```
LogisticRegression  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9782
       Fraud       0.91      0.79      0.85       146

    accuracy                           1.00      9928
   macro avg       0.95      0.89      0.92      9928
weighted avg       1.00      1.00      1.00      9928
```



```
HistGradientBoostingClassifier Roc Curve, Classification Report and Confusion Matrix:

HistGradientBoostingClassifier Model AUC Score is:  0.994926381416244
```



```
HistGradientBoostingClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9782
       Fraud       0.95      0.90      0.93       146

    accuracy                           1.00      9928
   macro avg       0.97      0.95      0.96      9928
weighted avg       1.00      1.00      1.00      9928
```



Top 2 model based on ROC-AUC score: XGB Classifier, SVC Model

After threshold application has led to notable improvements in the precision, recall, and F1-score metrics across various classifiers compared to the previous results. Specifically, classifiers like XGB Classifier, Random Forest Classifier, and Gradient Boosting Classifier exhibit enhanced precision and recall, indicating better performance in identifying fraudulent transactions while minimizing false positives.
However, some classifiers like Logistic Regression and Hist Gradient Boosting Classifier still demonstrate lower recall rates, suggesting a potential need for further optimization or the exploration of alternative methods to improve their fraud detection capabilities. Overall, the recent adjustments have resulted in more balanced model performances, emphasizing the importance of tuning thresholds to optimize classifier outcomes for specific objectives such as fraud detection.

Comparison of the models with scaling in df_filter with Smote and Scale

SMOTE: The dataset exhibits severe imbalance, with a majority of transactions being non-fraudulent. Conventional algorithms may inaccurately classify new observations, emphasizing the need for specialized techniques. To counter this issue, undersampling and oversampling methods are employed. Oversampling, such as the Synthetic Minority Oversampling Technique (SMOTE), involves augmenting the minority class instances, preserving information from the original set and mitigating loss. While oversampling prevents information loss, it is susceptible to overfitting. SMOTE, a specific form of oversampling, generates synthetic points for the minority class, contributing to a more balanced dataset and addressing the challenges posed by class imbalance.

In [25]:
```
X = df_filter.drop(['Class'], axis=1)
y = df_filter['Class']

over = SMOTE(sampling_strategy=.2)
under = RandomUnderSampler(sampling_strategy=1)
steps = [('o', over), ('u', under)]
pipeline = Pipeline(steps=steps)

X, y = pipeline.fit_resample(X, y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

Models = [
    ('XGB Classifier', XGBClassifier(random_state=42)),
    ('Light GBM Classifier', LGBMClassifier(random_state=42)),
    ('Gradient Boosting Classifier', GradientBoostingClassifier(random_state=42)),
    ('Ada Boost Classifier', AdaBoostClassifier(random_state=42)),
    ('Random Forest Classifier', RandomForestClassifier(random_state=42)),
    ('Logistic Regression', LogisticRegression(random_state=42)),
    ('Support Vector Machine (SVM)', SVC(gamma='auto', random_state=42)),
    ('Decision Tree Classifier', DecisionTreeClassifier(random_state=42)),
    ('Hist Gradient Boosting Classifier', HistGradientBoostingClassifier(random_state=42
]

name_list = []
accuracy_scores = []
accuracy_scores_train = []
recall_scores = []
recall_scores_train = []
f1_scores = []
f1_scores_train = []
precision_scores = []
precision_scores_train = []
roc_auc_scores_list = []
roc_auc_scores_train_list = []

for name, model in Models:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_pred_train = model.predict(X_train)
    accuracy_scores.append(accuracy_score(y_test, y_pred))
    accuracy_scores_train.append(accuracy_score(y_train, y_pred_train))
    recall_scores.append(recall_score(y_test, y_pred))
    recall_scores_train.append(recall_score(y_train, y_pred_train))
    f1_scores.append(f1_score(y_test, y_pred))
    f1_scores_train.append(f1_score(y_train, y_pred_train))
    precision_scores.append(precision_score(y_test, y_pred))
    precision_scores_train.append(precision_score(y_train, y_pred_train))
    name_list.append(name)

result = {
    'Model': name_list,
    'Accuracy Score Test': accuracy_scores,
    'Accuracy Score Train': accuracy_scores_train,
    'Recall Score Test': recall_scores,
    'Recall Score Train': recall_scores_train,
    'F1 Score Test': f1_scores,
    'F1 Score Train': f1_scores_train,
    'Precision Score Test': precision_scores,
    'Precision Score Train': precision_scores_train
}

dataframe = pd.DataFrame(result).sort_values(by='F1 Score Test', ascending=False)
dataframe.reset_index(drop=True)
```

```
[LightGBM] [Info] Number of positive: 4564, number of negative: 4564
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.
004905 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 9128, number of used features:
30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

Out[25]:

	Model	Accuracy Score Test	Accuracy Score Train	Recall Score Test	Recall Score Train	F1 Score Test	F1 Score Train	Precision Score Test	Precision Score Train
0	XGB Classifier	0.997188	1.000000	0.995399	1.000000	0.997183	1.000000	0.998974	1.000000

1	Light GBM Classifier	0.996933	1.000000	0.994888	1.000000	0.996926	1.000000	0.998973	1.000000
2	Random Forest Classifier	0.996421	1.000000	0.993865	1.000000	0.996412	1.000000	0.998972	1.000000
3	Hist Gradient Boosting Classifier	0.996166	1.000000	0.993354	1.000000	0.996155	1.000000	0.998972	1.000000
4	Ada Boost Classifier	0.995654	0.998138	0.993354	0.998466	0.995644	0.998138	0.997946	0.997810
5	Gradient Boosting Classifier	0.995399	0.998795	0.992331	0.999124	0.995385	0.998795	0.998457	0.998467
6	Decision Tree Classifier	0.991820	1.000000	0.989775	1.000000	0.991803	1.000000	0.993840	1.000000
7	Support Vector Machine (SVM)	0.985174	0.988935	0.975971	0.980719	0.985036	0.988843	0.994271	0.997104
8	Logistic Regression	0.979806	0.983019	0.968814	0.973269	0.979581	0.982852	0.990591	0.992626

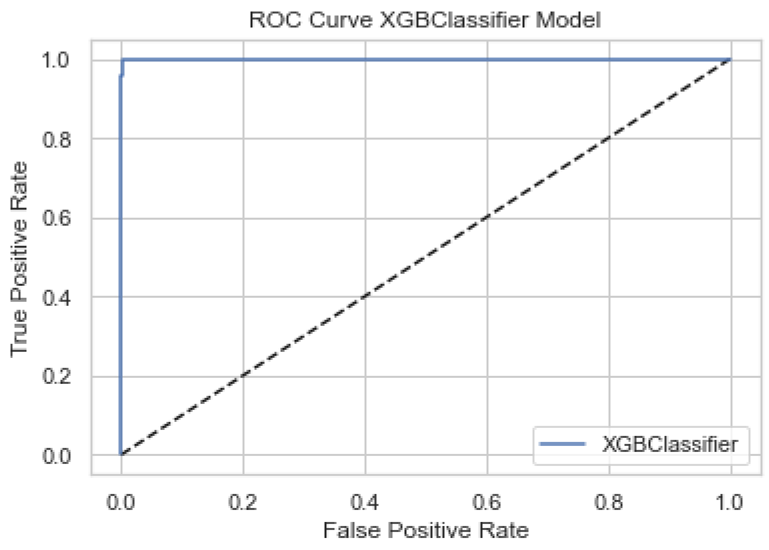
Top 2 model based on Recall score, F1 score and Precision score: XGB Classifier, Random Forest Classifier

SMOTE adjustments in model performance show consistent or slightly improved accuracy, recall, precision, and F1 scores across most classifiers compared to the previous results. Specifically, classifiers like Hist Gradient Boosting, XGB, Light GBM, and Random Forest demonstrate robust performance with high scores across all metrics, indicating their effectiveness in accurately detecting fraudulent transactions. However, classifiers like Gradient Boosting, Ada Boost, and SVM exhibit slightly lower scores but still maintain reasonable performance levels. Notably, Logistic Regression shows a decline in performance compared to the previous results, suggesting potential areas for further optimization or exploration of alternative methods.

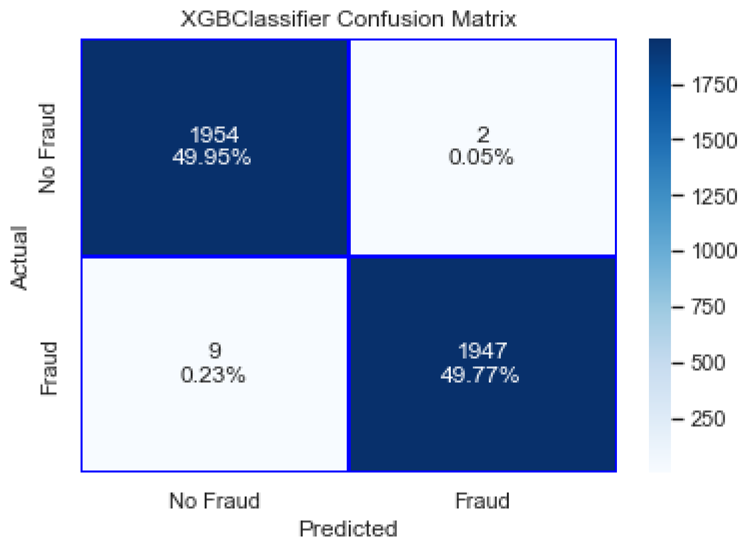
Roc Curve, Classification Report and Confusion Matrix

In [26]:
```
XGB = XGBClassifier()
RF = RandomForestClassifier()
SVM = SVC(probability=True)
DTC = DecisionTreeClassifier()
LightGBM = LGBMClassifier()
GBC = GradientBoostingClassifier()
ADA = AdaBoostClassifier()
LR = LogisticRegression()
HGB = HistGradientBoostingClassifier()

Models = [XGB, RF, SVM, DTC, LightGBM, GBC, ADA, LR, HGB]

for model in Models:
    model.fit(X_train, y_train)
    y_pred_prob = model.predict_proba(X_test)[:, 1]
    threshold = 0.5
    y_pred = (y_pred_prob > threshold).astype(int)
    fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
    print('\n' )
    print(type(model).__name__+' Roc Curve, Classification Report and Confusion Matrix:\
    print(type(model).__name__, 'Model AUC Score is: ', auc(fpr, tpr))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.plot(fpr, tpr, label=type(model).__name__)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve ' + type(model).__name__ + ' Model')
    plt.legend()
    plt.show()

    # Convert probability scores to binary predictions using a threshold
    threshold = 0.5
    y_pred_binary = (y_pred_prob > threshold).astype(int)

    labelsNF = ['No Fraud', 'Fraud']
    report = classification_report(y_test, y_pred_binary, target_names=labelsNF)
    print(type(model).__name__,' Classification Report:\n', report)

    cm = confusion_matrix(y_test, y_pred_binary)

    counts = [value for value in cm.flatten()]
    percentages = ['{0:.2%}'.format(value) for value in cm.flatten() / np.sum(cm)]
    labels = [f'{v1}\n{v2}' for v1, v2 in zip(counts, percentages)]

    labels = np.array(labels).reshape(cm.shape)

    sns.heatmap(cm, annot=labels, fmt='', cmap='Blues', linewidths=1.5, linecolor='blue'
                xticklabels=labelsNF, yticklabels=labelsNF)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(type(model).__name__ + ' Confusion Matrix')
    plt.show()
```

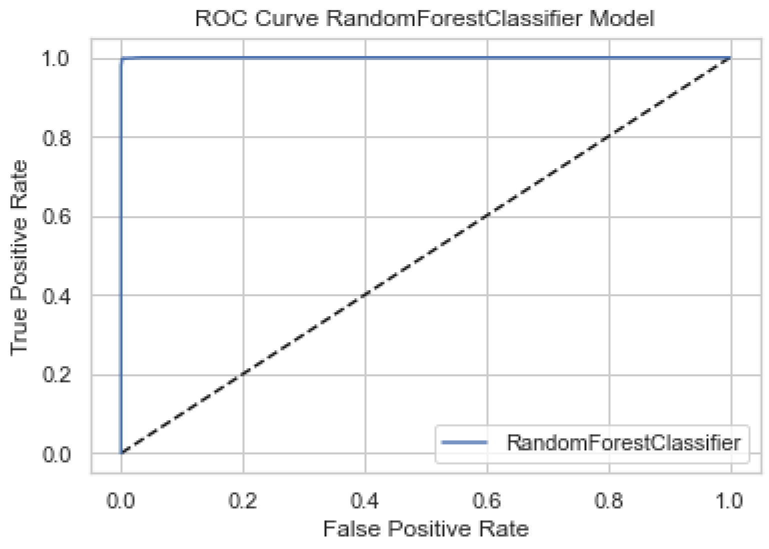
```
XGBClassifier Roc Curve, Classification Report and Confusion Matrix:

XGBClassifier Model AUC Score is:  0.9999106101095262
```



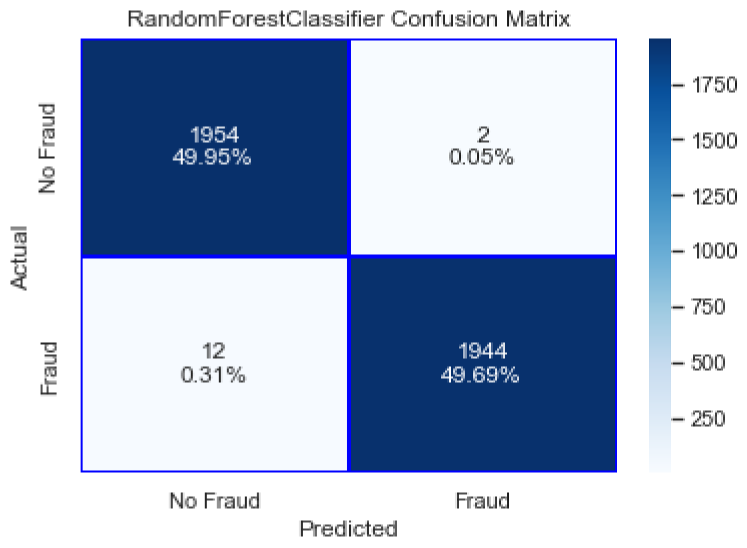
```
XGBClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      1956
       Fraud       1.00      1.00      1.00      1956

    accuracy                           1.00      3912
   macro avg       1.00      1.00      1.00      3912
weighted avg       1.00      1.00      1.00      3912
```



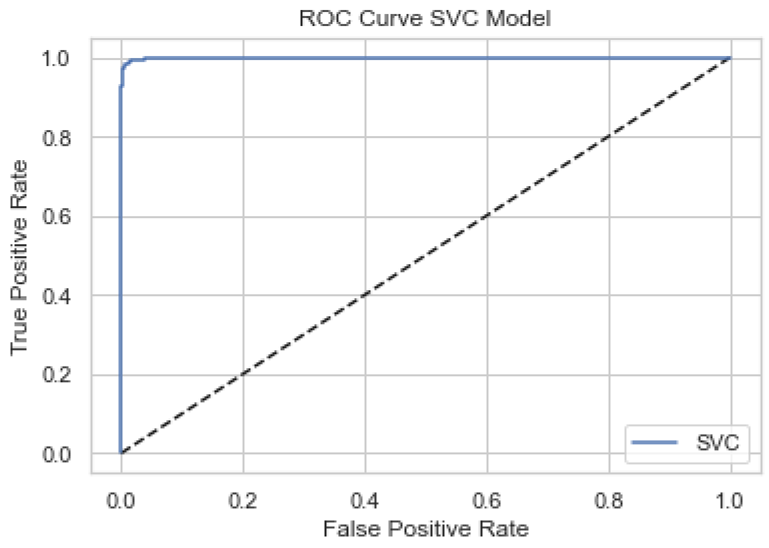
```
RandomForestClassifier Roc Curve, Classification Report and Confusion Matrix:

RandomForestClassifier Model AUC Score is:  0.9999368781913759
```



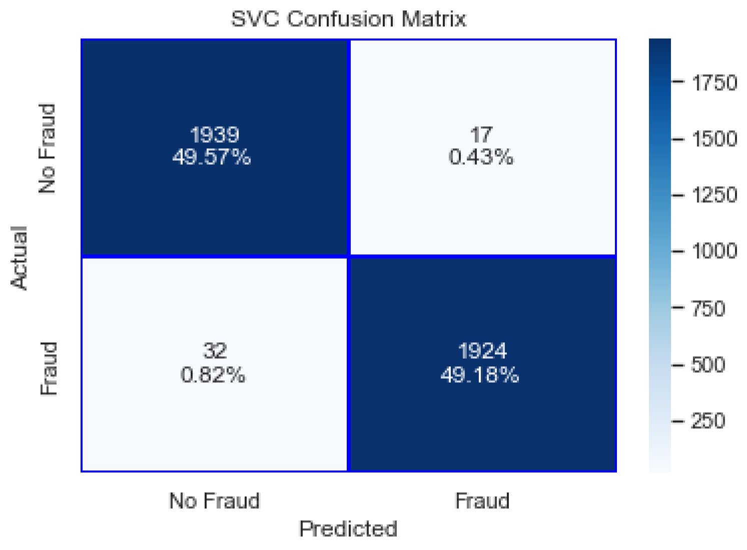
```
RandomForestClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       0.99      1.00      1.00      1956
       Fraud       1.00      0.99      1.00      1956

    accuracy                           1.00      3912
   macro avg       1.00      1.00      1.00      3912
weighted avg       1.00      1.00      1.00      3912
```



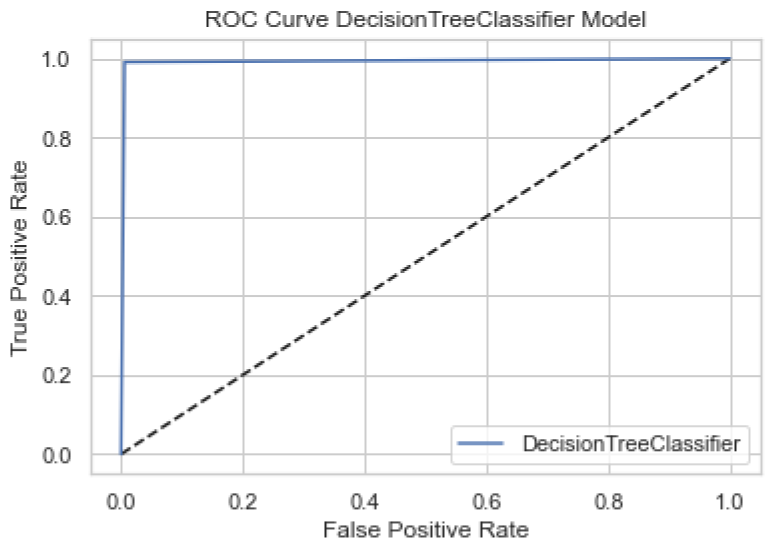
```
SVC Roc Curve, Classification Report and Confusion Matrix:

SVC Model AUC Score is:  0.9991949682378378
```



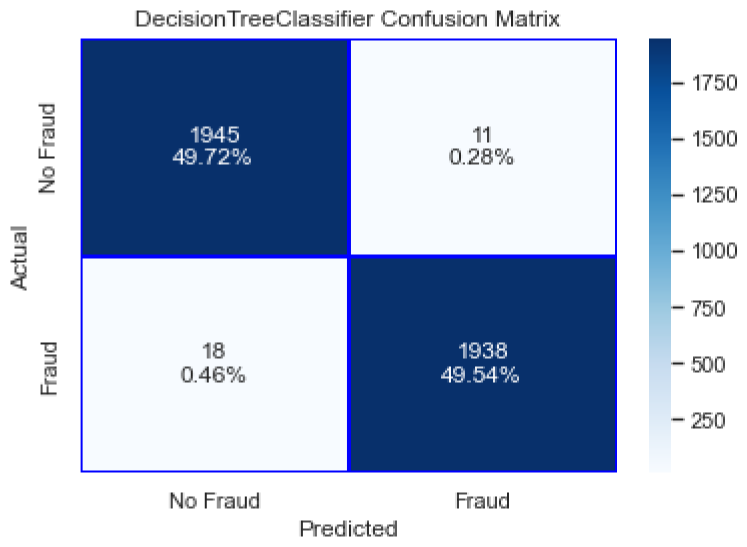
```
SVC  Classification Report:
               precision    recall  f1-score   support

    No Fraud       0.98      0.99      0.99      1956
       Fraud       0.99      0.98      0.99      1956

    accuracy                           0.99      3912
   macro avg       0.99      0.99      0.99      3912
weighted avg       0.99      0.99      0.99      3912
```



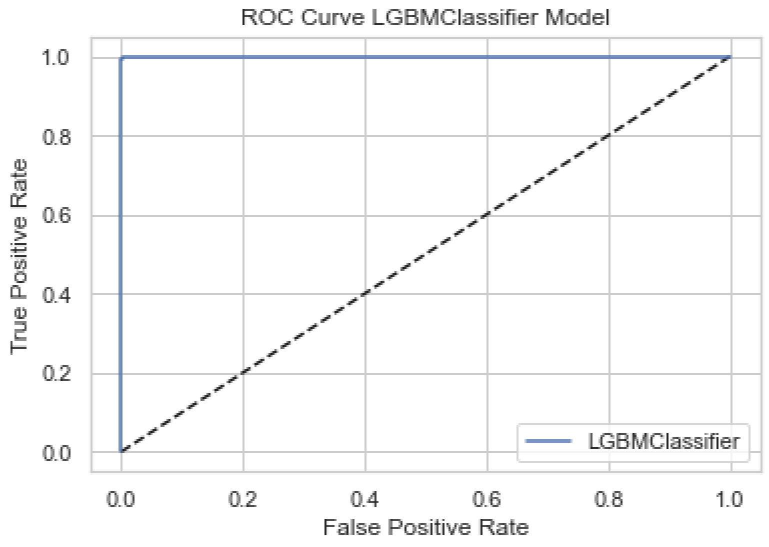
```
DecisionTreeClassifier Roc Curve, Classification Report and Confusion Matrix:

DecisionTreeClassifier Model AUC Score is:  0.9925869120654397
```



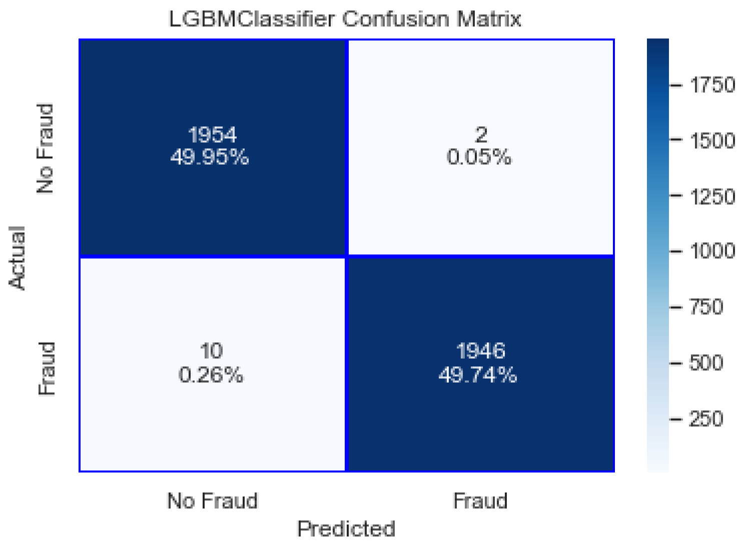
```
DecisionTreeClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       0.99      0.99      0.99      1956
       Fraud       0.99      0.99      0.99      1956

    accuracy                           0.99      3912
   macro avg       0.99      0.99      0.99      3912
weighted avg       0.99      0.99      0.99      3912
```



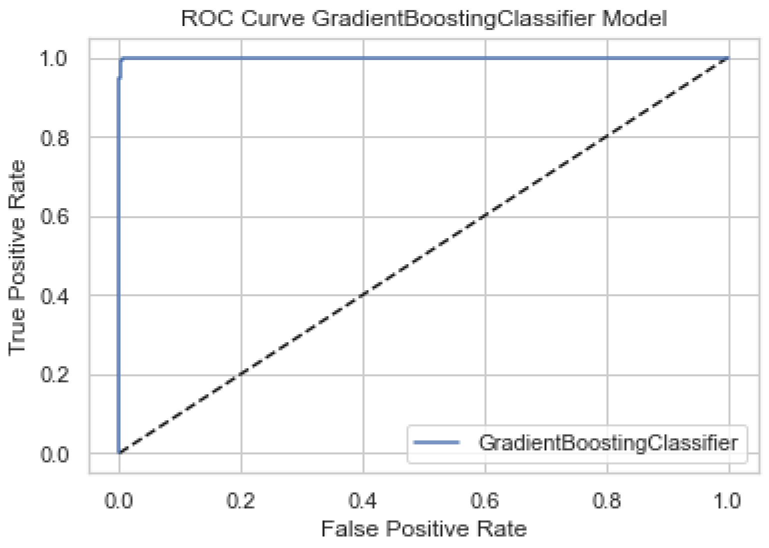
```
[LightGBM] [Info] Number of positive: 4564, number of negative: 4564
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.
007435 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 9128, number of used features:
30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf


LGBMClassifier Roc Curve, Classification Report and Confusion Matrix:

LGBMClassifier Model AUC Score is:  0.9999221105632714
```



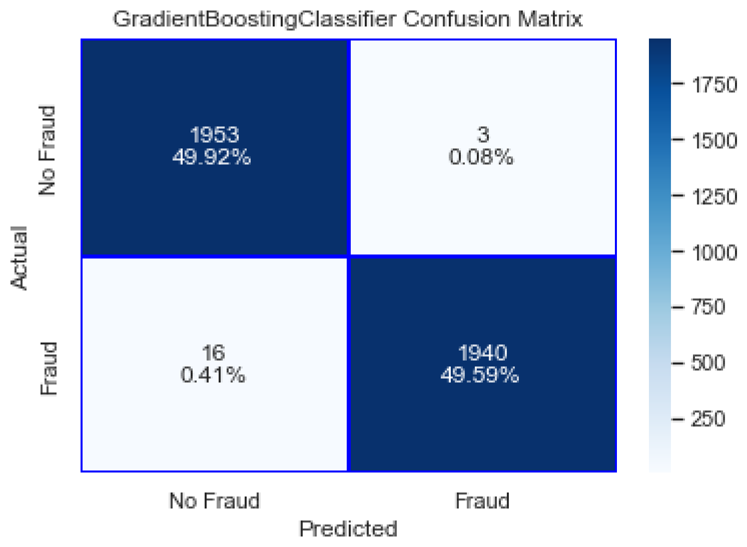
```
LGBMClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       0.99      1.00      1.00      1956
       Fraud       1.00      0.99      1.00      1956

    accuracy                           1.00      3912
   macro avg       1.00      1.00      1.00      3912
weighted avg       1.00      1.00      1.00      3912
```



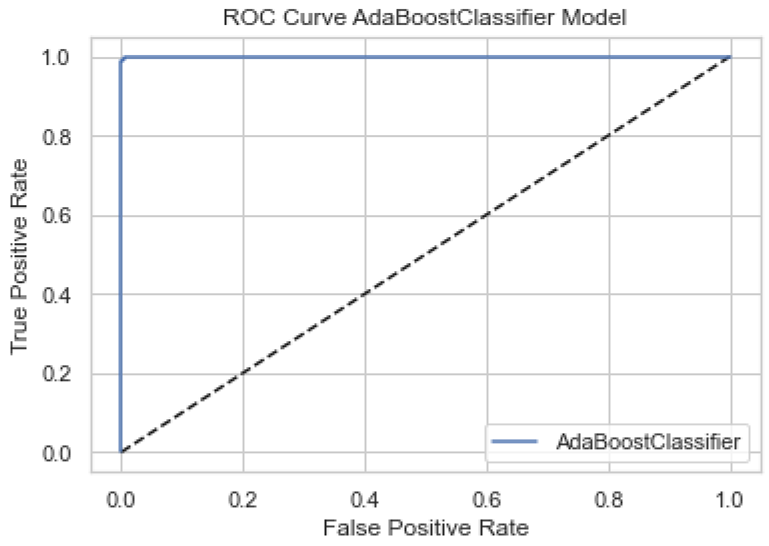
```
GradientBoostingClassifier Roc Curve, Classification Report and Confusion Matrix:

GradientBoostingClassifier Model AUC Score is:  0.9996628276061074
```



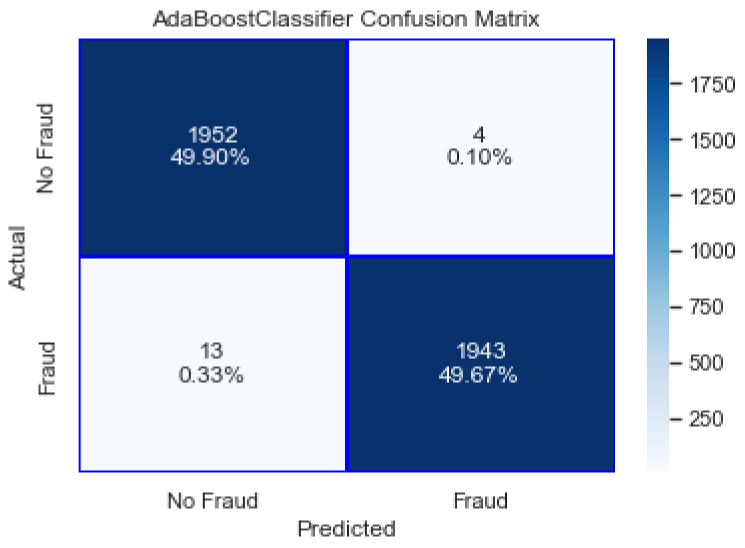
```
GradientBoostingClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       0.99      1.00      1.00      1956
       Fraud       1.00      0.99      1.00      1956

    accuracy                           1.00      3912
   macro avg       1.00      1.00      1.00      3912
weighted avg       1.00      1.00      1.00      3912
```



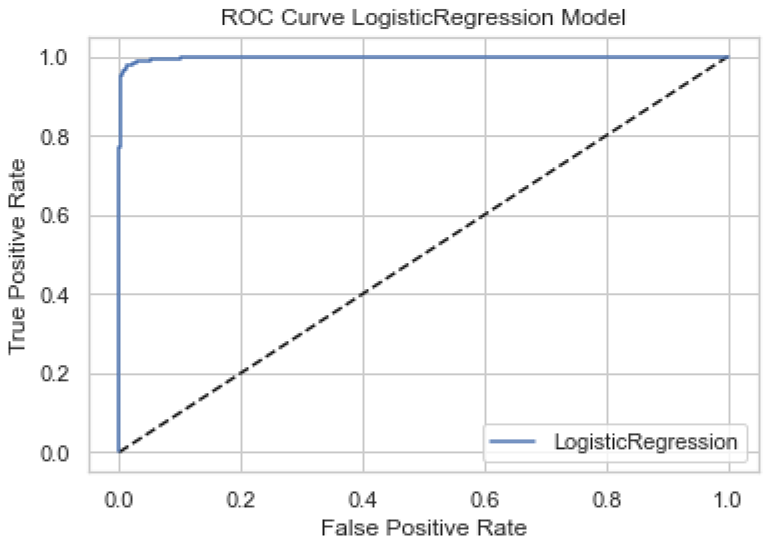
```
AdaBoostClassifier Roc Curve, Classification Report and Confusion Matrix:

AdaBoostClassifier Model AUC Score is:  0.9998666992861354
```



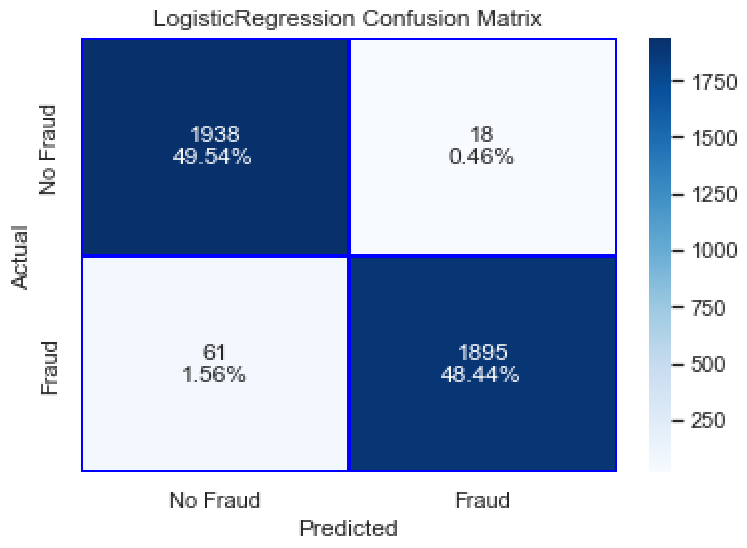
```
AdaBoostClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       0.99      1.00      1.00      1956
       Fraud       1.00      0.99      1.00      1956

    accuracy                           1.00      3912
   macro avg       1.00      1.00      1.00      3912
weighted avg       1.00      1.00      1.00      3912
```



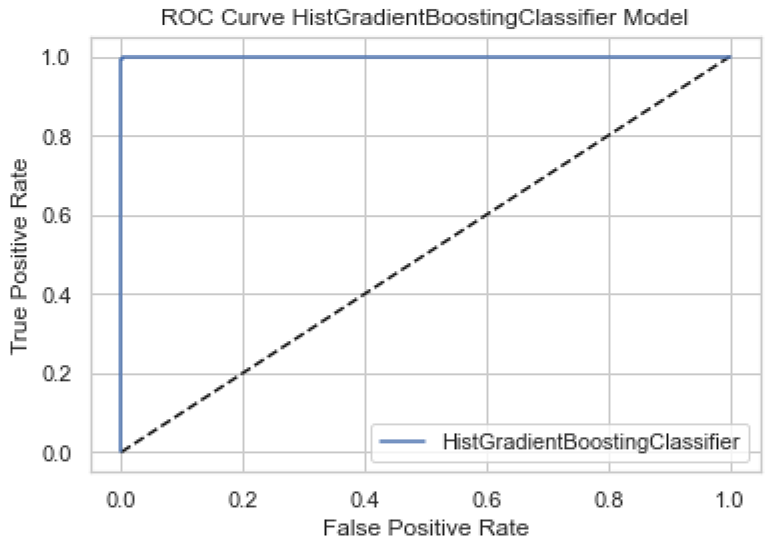
```
LogisticRegression Roc Curve, Classification Report and Confusion Matrix:

LogisticRegression Model AUC Score is:  0.9975545853354577
```



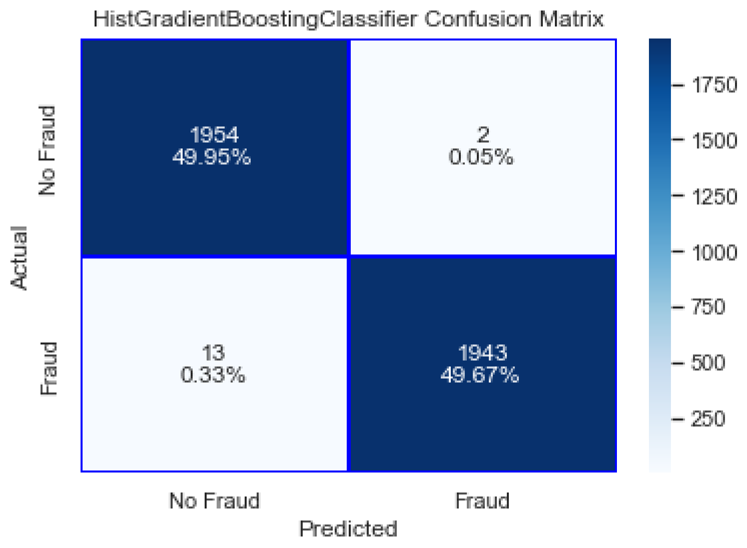
```
LogisticRegression  Classification Report:
               precision    recall  f1-score   support

    No Fraud       0.97      0.99      0.98      1956
       Fraud       0.99      0.97      0.98      1956

    accuracy                           0.98      3912
   macro avg       0.98      0.98      0.98      3912
weighted avg       0.98      0.98      0.98      3912
```



```
HistGradientBoostingClassifier Roc Curve, Classification Report and Confusion Matrix:

HistGradientBoostingClassifier Model AUC Score is:  0.9999025075171147
```



```
HistGradientBoostingClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       0.99      1.00      1.00      1956
       Fraud       1.00      0.99      1.00      1956

    accuracy                           1.00      3912
   macro avg       1.00      1.00      1.00      3912
weighted avg       1.00      1.00      1.00      3912
```



Top 2 model based on ROC-AUC score: XGB Classifier, Hist Gradient Boosting Classifier.

After applying the 0.5 threshold, exhibit consistent or improved performance across most classifiers compared to the previous iteration. Classifiers like XGB, RandomForest, LGBM, and Hist Gradient Boosting demonstrate exceptionally high AUC scores, precision, recall, and F1-scores, signifying robust detection capabilities for both fraud and non-fraud instances.
However, there are some disparities in the performance of certain models. For instance, SVM and Logistic Regression exhibit a slight decrease in accuracy and recall compared to the previous results. This suggests potential areas for further tuning or consideration of alternative algorithms.

Comparison of the models with scaling in df_filter using ADASYN

ADASYN: ADASYN (Adaptive Synthetic Sampling) is an oversampling technique designed to address class imbalance in datasets. Unlike conventional oversampling methods, ADASYN focuses on generating synthetic instances for the minority class based on the distribution of existing examples. This adaptability ensures that more synthetic samples are created for regions with fewer instances, effectively emphasizing the underrepresented class. By introducing diversity to the synthetic samples, ADASYN helps in training a more robust and generalizable AI model, enabling better classification performance, particularly in scenarios where class imbalances are pronounced.

In [27]:
```
X = df_filter.drop(['Class'], axis=1)
y = df_filter['Class']

adasyn = ADASYN(random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train_adasyn, y_train_adasyn = adasyn.fit_resample(X_train, y_train)

Models = [
    ('XGB Classifier', XGBClassifier(random_state=42)),
    ('Light GBM Classifier', LGBMClassifier(random_state=42)),
    ('Gradient Boosting Classifier', GradientBoostingClassifier(random_state=42)),
    ('Ada Boost Classifier', AdaBoostClassifier(random_state=42)),
    ('Random Forest Classifier', RandomForestClassifier(random_state=42)),
    ('Logistic Regression', LogisticRegression(random_state=42)),
    ('Support Vector Machine (SVM)', SVC(gamma='auto', random_state=42)),
    ('Decision Tree Classifier', DecisionTreeClassifier(random_state=42)),
    ('Hist Gradient Boosting Classifier', HistGradientBoostingClassifier(random_state=42
]

name_list = []
accuracy_scores = []
accuracy_scores_train = []
recall_scores = []
recall_scores_train = []
f1_scores = []
f1_scores_train = []
precision_scores = []
precision_scores_train = []

for name, model in Models:
    model.fit(X_train_adasyn, y_train_adasyn)
    y_pred = model.predict(X_test)
    y_pred_train = model.predict(X_train_adasyn)
    accuracy_scores.append(accuracy_score(y_test, y_pred))
    accuracy_scores_train.append(accuracy_score(y_train_adasyn, y_pred_train))
    recall_scores.append(recall_score(y_test, y_pred))
    recall_scores_train.append(recall_score(y_train_adasyn, y_pred_train))
    f1_scores.append(f1_score(y_test, y_pred))
    f1_scores_train.append(f1_score(y_train_adasyn, y_pred_train))
    precision_scores.append(precision_score(y_test, y_pred))
    precision_scores_train.append(precision_score(y_train_adasyn, y_pred_train))

    name_list.append(name)

result = {
    'Model': name_list,
    'Accuracy Score Test': accuracy_scores,
    'Accuracy Score Train': accuracy_scores_train,
    'Recall Score Test': recall_scores,
    'Recall Score Train': recall_scores_train,
    'F1 Score Test': f1_scores,
    'F1 Score Train': f1_scores_train,
    'Precision Score Test': precision_scores,
    'Precision Score Train': precision_scores_train
}

dataframe = pd.DataFrame(result).sort_values(by='F1 Score Test', ascending=False)
dataframe.reset_index(drop=True)
```

```
[LightGBM] [Info] Number of positive: 22830, number of negative: 22821
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.
012805 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 45651, number of used feature
s: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500099 -> initscore=0.000394
[LightGBM] [Info] Start training from score 0.000394
```

Out[27]:

	Model	Accuracy Score Test	Accuracy Score Train	Recall Score Test	Recall Score Train	F1 Score Test	F1 Score Train	Precision Score Test	Precision Score Train
0	Random Forest Classifier	0.998288	1.000000	0.972973	1.000000	0.944262	1.000000	0.917197	1.000000
1	Hist Gradient Boosting Classifier	0.998086	1.000000	0.972973	1.000000	0.938111	1.000000	0.905660	1.000000
2	Light GBM Classifier	0.998086	1.000000	0.966216	1.000000	0.937705	1.000000	0.910828	1.000000
3	XGB Classifier	0.997885	1.000000	0.966216	1.000000	0.931596	1.000000	0.899371	1.000000
4	Support Vector	0.996475	0.999518	0.864865	1.000000	0.879725	0.999518	0.895105	0.999037

	Machine (SVM)								
5	Gradient Boosting Classifier	0.995568	0.997941	0.966216	0.998861	0.866667	0.997943	0.785714	0.997027
6	Ada Boost Classifier	0.994662	0.996429	0.979730	0.996934	0.845481	0.996432	0.743590	0.995931
7	Decision Tree Classifier	0.994662	1.000000	0.932432	1.000000	0.838906	1.000000	0.762431	1.000000
8	Logistic Regression	0.943191	0.950275	0.945946	0.953876	0.331754	0.950463	0.201149	0.947073

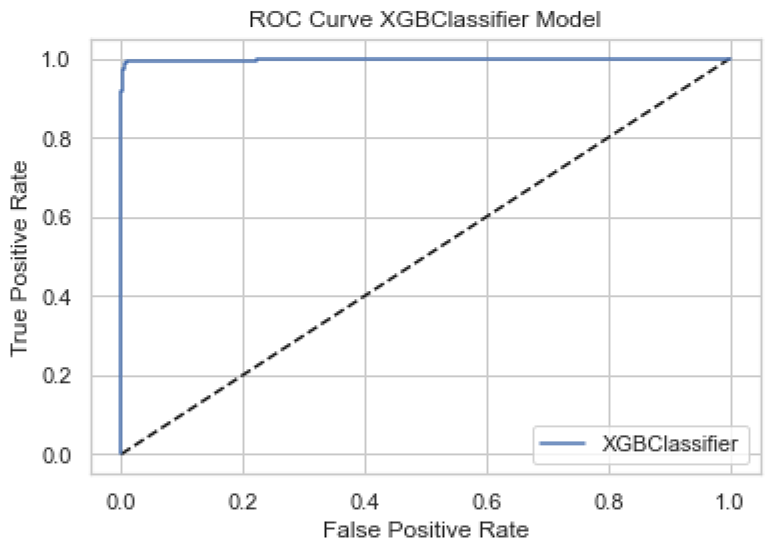
Top 2 model based on Recall score, F1 score and Precision score: XGB Classifier, Random Forest Classifier.

ADASYN showcase marginal to moderate shifts in model performance metrics compared to the previous iteration. Models like Hist Gradient Boosting, Light GBM, XGB, and RandomForest retain their high accuracy and recall scores, indicating consistent fraud detection capabilities.
Notably, SVM demonstrates a slight decrease in accuracy and recall, suggesting potential areas for optimization or alternative model selection. Similarly, Logistic Regression exhibits a noticeable decline in all metrics, highlighting the need for revisiting feature engineering or considering more sophisticated algorithms.

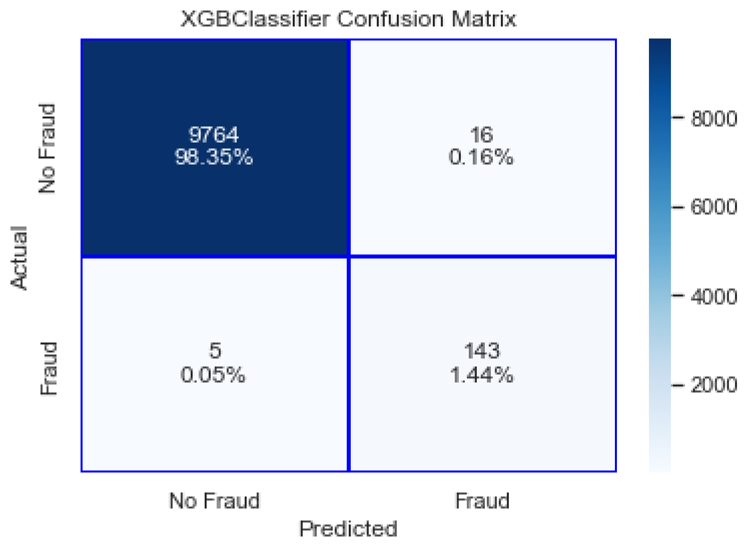
Roc Curve, Classification Report and Confusion Matrix

In [28]:
```python
XGB = XGBClassifier()
RF = RandomForestClassifier()
SVM = SVC(probability=True)
DTC = DecisionTreeClassifier()
LightGBM = LGBMClassifier()
GBC = GradientBoostingClassifier()
ADA = AdaBoostClassifier()
LR = LogisticRegression()
HGB = HistGradientBoostingClassifier()

Models = [XGB, RF, SVM, DTC, LightGBM, GBC, ADA, LR, HGB]

for model in Models:
    model.fit(X_train_adasyn, y_train_adasyn)
    y_pred_prob = model.predict_proba(X_test)[:, 1]
    threshold = 0.5
    y_pred = (y_pred_prob > threshold).astype(int)
    fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
    print('\n' )
    print(type(model).__name__+' Roc Curve and Confusion Matrix:\n')
    print(type(model).__name__, 'Model AUC Score is: ', auc(fpr, tpr))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.plot(fpr, tpr, label=type(model).__name__)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve ' + type(model).__name__ + ' Model')
    plt.legend()
    plt.show()

    # Convert probability scores to binary predictions using a threshold
    threshold = 0.5
    y_pred_binary = (y_pred_prob > threshold).astype(int)

    labelsNF = ['No Fraud', 'Fraud']
    report = classification_report(y_test, y_pred_binary, target_names=labelsNF)
    print(type(model).__name__,' Classification Report:\n', report)

    cm = confusion_matrix(y_test, y_pred_binary)

    counts = [value for value in cm.flatten()]
    percentages = ['{0:.2%}'.format(value) for value in cm.flatten() / np.sum(cm)]
    labels = [f'{v1}\n{v2}' for v1, v2 in zip(counts, percentages)]

    labels = np.array(labels).reshape(cm.shape)

    sns.heatmap(cm, annot=labels, fmt='', cmap='Blues', linewidths=1.5, linecolor='blue'
                xticklabels=labelsNF, yticklabels=labelsNF)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(type(model).__name__ + ' Confusion Matrix')
    plt.show()
```

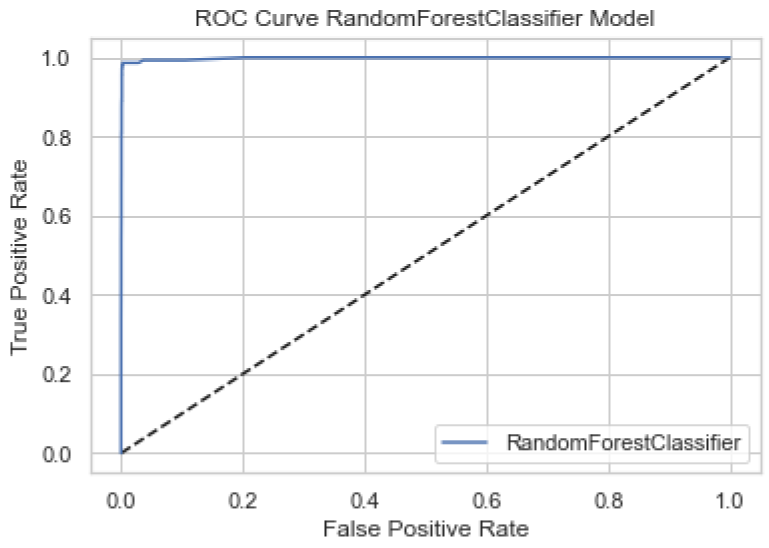
```
XGBClassifier Roc Curve and Confusion Matrix:

XGBClassifier Model AUC Score is:  0.9981961255734262
```



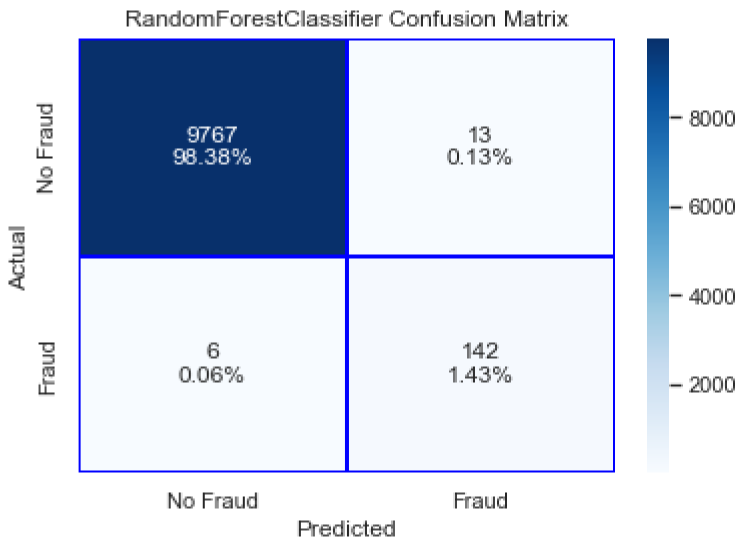
```
XGBClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9780
       Fraud       0.90      0.97      0.93       148

    accuracy                           1.00      9928
   macro avg       0.95      0.98      0.97      9928
weighted avg       1.00      1.00      1.00      9928
```



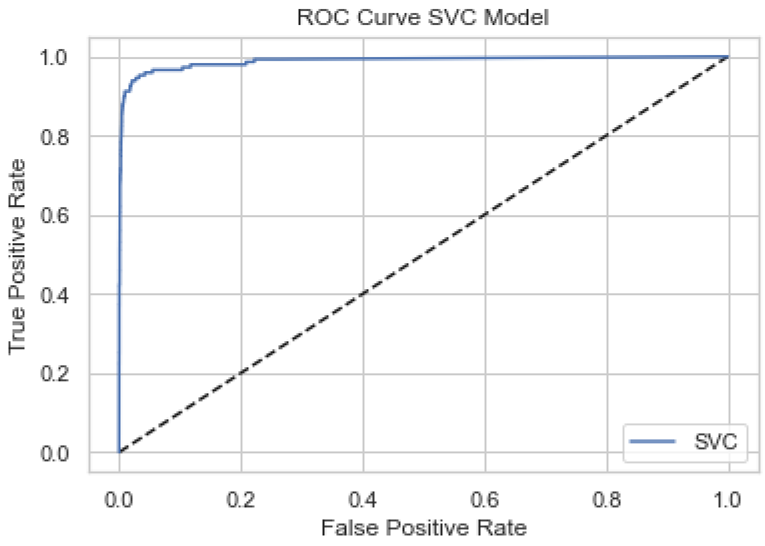
```
RandomForestClassifier Roc Curve and Confusion Matrix:

RandomForestClassifier Model AUC Score is:  0.9983871524899133
```



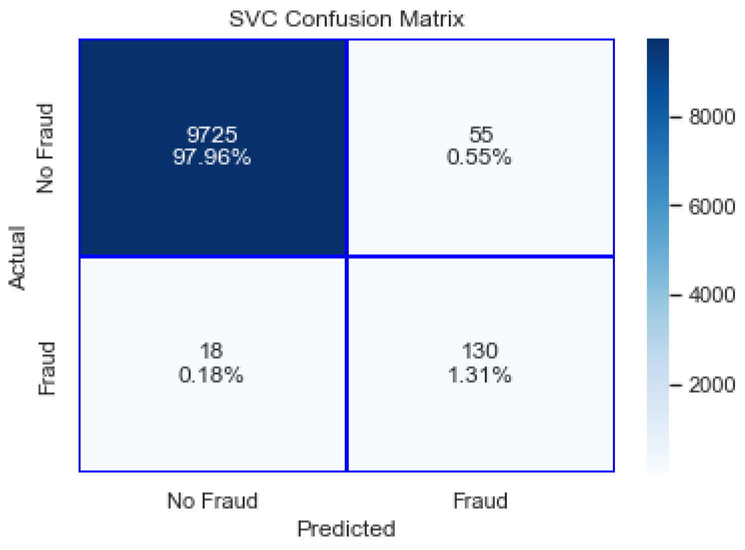
```
RandomForestClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9780
       Fraud       0.92      0.96      0.94       148

    accuracy                           1.00      9928
   macro avg       0.96      0.98      0.97      9928
weighted avg       1.00      1.00      1.00      9928
```



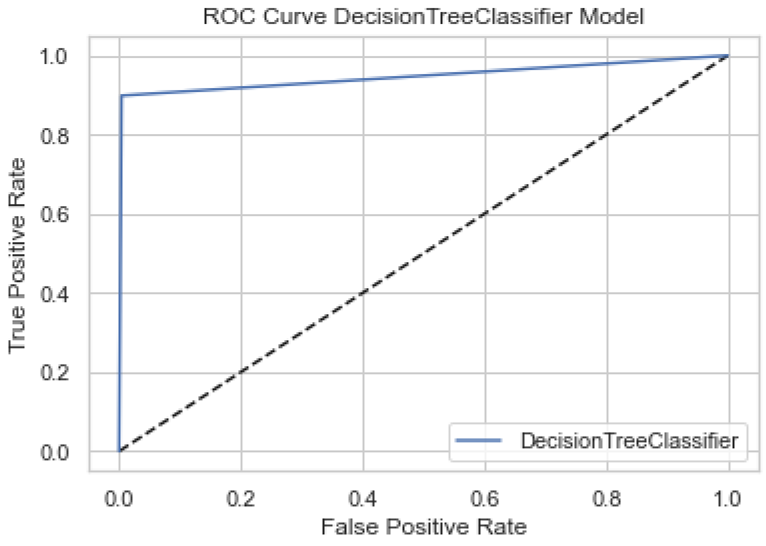
```
SVC Roc Curve and Confusion Matrix:

SVC Model AUC Score is:  0.9883988282761289
```



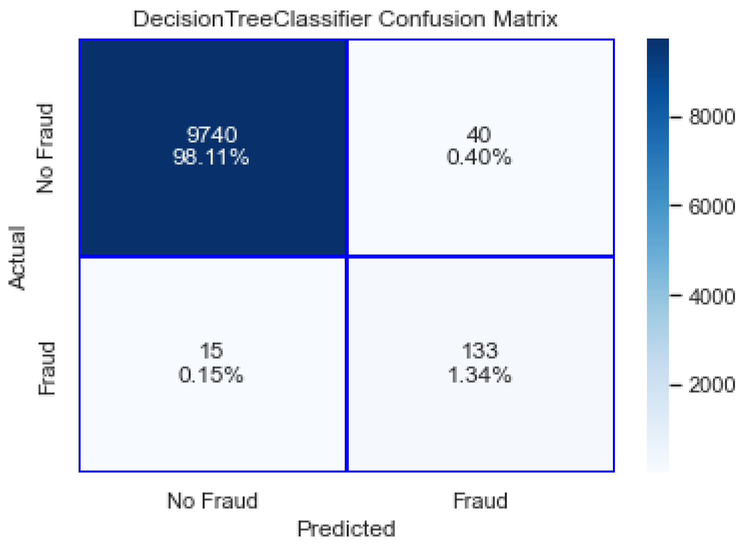
```
SVC  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      0.99      1.00      9780
       Fraud       0.70      0.88      0.78       148

    accuracy                           0.99      9928
   macro avg       0.85      0.94      0.89      9928
weighted avg       0.99      0.99      0.99      9928
```



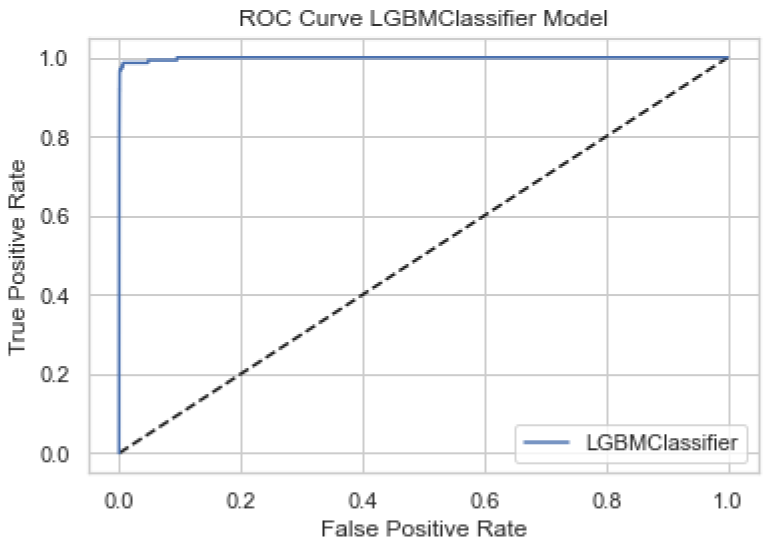
```
DecisionTreeClassifier Roc Curve and Confusion Matrix:

DecisionTreeClassifier Model AUC Score is:  0.9472793345492733
```



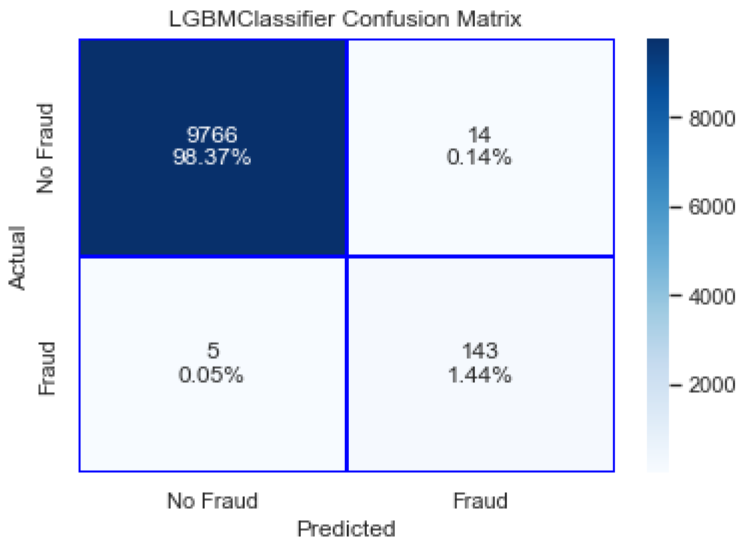
```
DecisionTreeClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9780
       Fraud       0.77      0.90      0.83       148

    accuracy                           0.99      9928
   macro avg       0.88      0.95      0.91      9928
weighted avg       1.00      0.99      0.99      9928
```



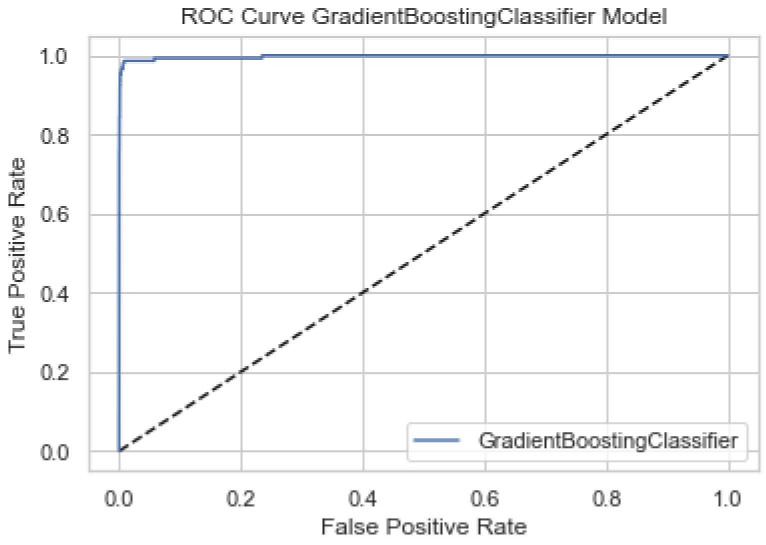
```
[LightGBM] [Info] Number of positive: 22830, number of negative: 22821
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.
005600 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 45651, number of used feature
s: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500099 -> initscore=0.000394
[LightGBM] [Info] Start training from score 0.000394


LGBMClassifier Roc Curve and Confusion Matrix:

LGBMClassifier Model AUC Score is:  0.9987274083899851
```



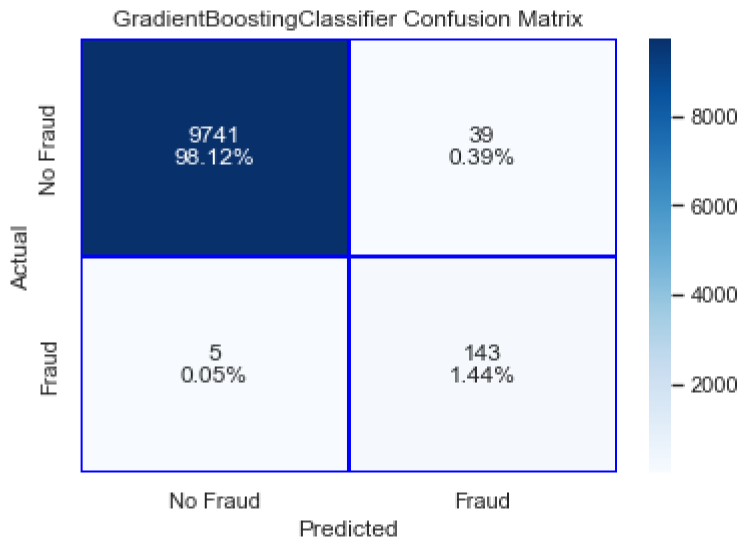
```
LGBMClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9780
       Fraud       0.91      0.97      0.94       148

    accuracy                           1.00      9928
   macro avg       0.96      0.98      0.97      9928
weighted avg       1.00      1.00      1.00      9928
```



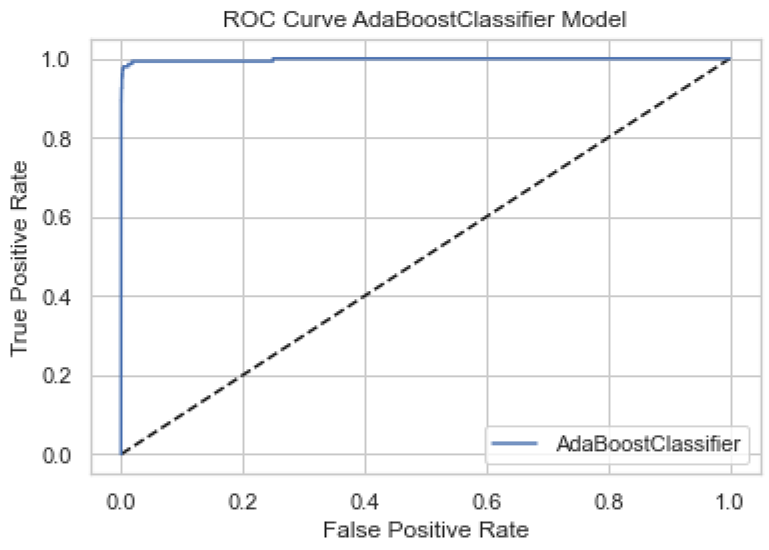
```
GradientBoostingClassifier Roc Curve and Confusion Matrix:

GradientBoostingClassifier Model AUC Score is:  0.9975505029569446
```



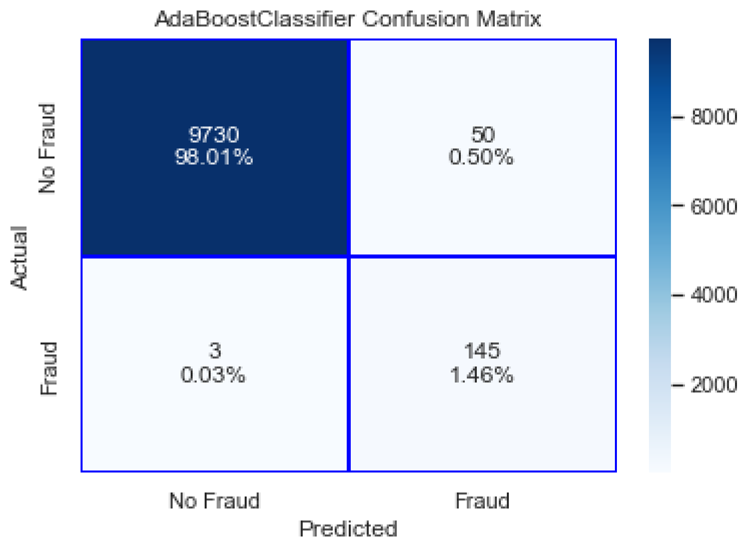
```
GradientBoostingClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9780
       Fraud       0.79      0.97      0.87       148

    accuracy                           1.00      9928
   macro avg       0.89      0.98      0.93      9928
weighted avg       1.00      1.00      1.00      9928
```



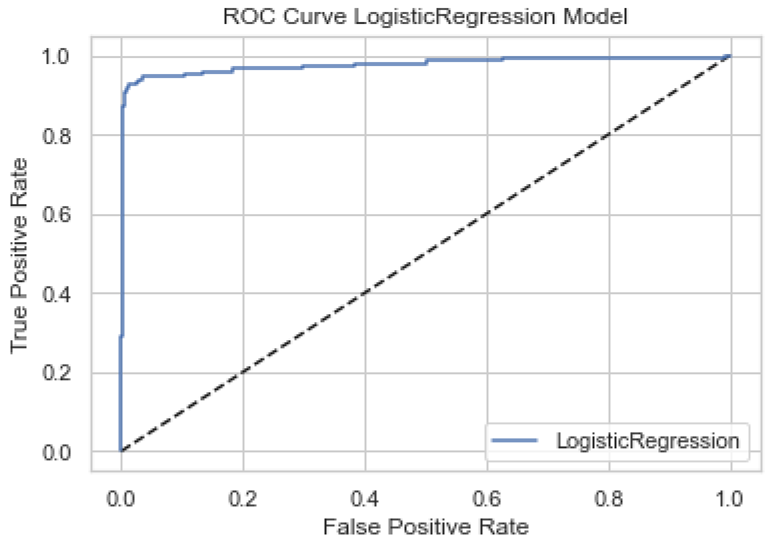
```
AdaBoostClassifier Roc Curve and Confusion Matrix:

AdaBoostClassifier Model AUC Score is:  0.9979456834134748
```



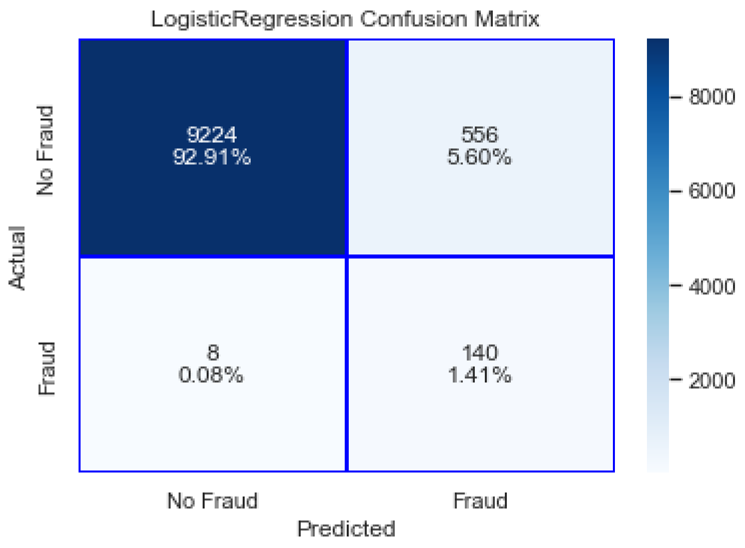
```
AdaBoostClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      0.99      1.00      9780
       Fraud       0.74      0.98      0.85       148

    accuracy                           0.99      9928
   macro avg       0.87      0.99      0.92      9928
weighted avg       1.00      0.99      1.00      9928
```



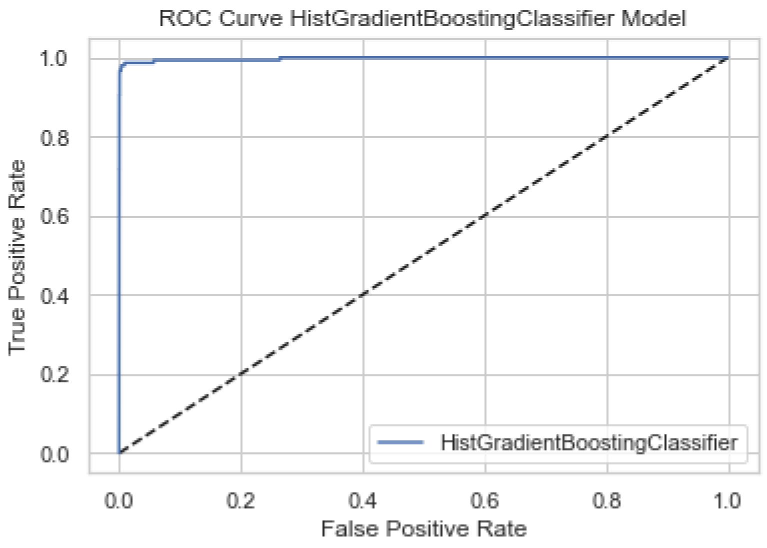
```
LogisticRegression Roc Curve and Confusion Matrix:

LogisticRegression Model AUC Score is:  0.9762187033659426
```



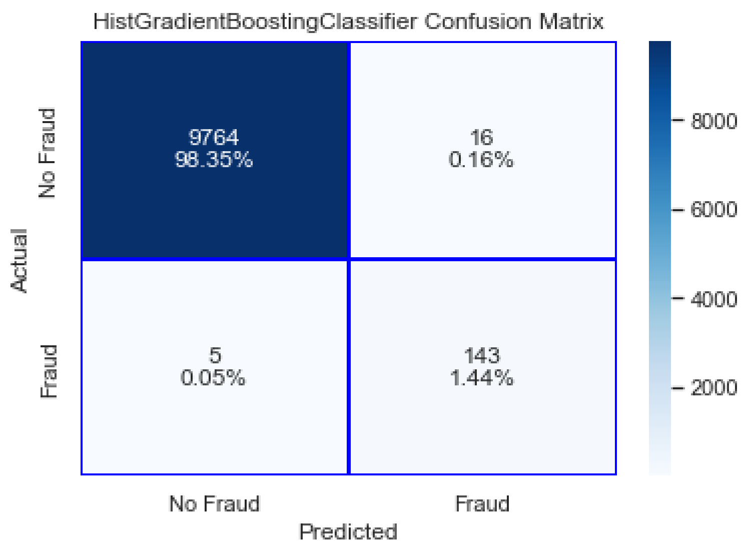
```
LogisticRegression  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      0.94      0.97      9780
       Fraud       0.20      0.95      0.33       148

    accuracy                           0.94      9928
   macro avg       0.60      0.94      0.65      9928
weighted avg       0.99      0.94      0.96      9928
```



```
HistGradientBoostingClassifier Roc Curve and Confusion Matrix:

HistGradientBoostingClassifier Model AUC Score is:  0.9975225225225225
```



```
HistGradientBoostingClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9780
       Fraud       0.90      0.97      0.93       148

    accuracy                           1.00      9928
   macro avg       0.95      0.98      0.97      9928
weighted avg       1.00      1.00      1.00      9928
```



Top 2 model based on ROC-AUC score: Light GBM Classifier, Random Forest Classifier

With a threshold of 0.5 demonstrates consistent high accuracy across models like XGB, RandomForest, and LGBM, ensuring reliable fraud detection. However, there are notable variations in recall and precision, especially in models like SVC, DecisionTree, and Logistic Regression, which exhibit reduced recall for fraud instances. This suggests a potential compromise in identifying fraudulent transactions.
The SVC and Logistic Regression models notably suffer from decreased precision, indicating a higher false positive rate. It's imperative to recalibrate these models or explore alternative algorithms to enhance their fraud detection capabilities. Overall, the new results underscore the importance of fine-tuning model thresholds to strike a balance between precision and recall, crucial for effective fraud detection systems.

Feature Scaling using RobustScaler Scaler

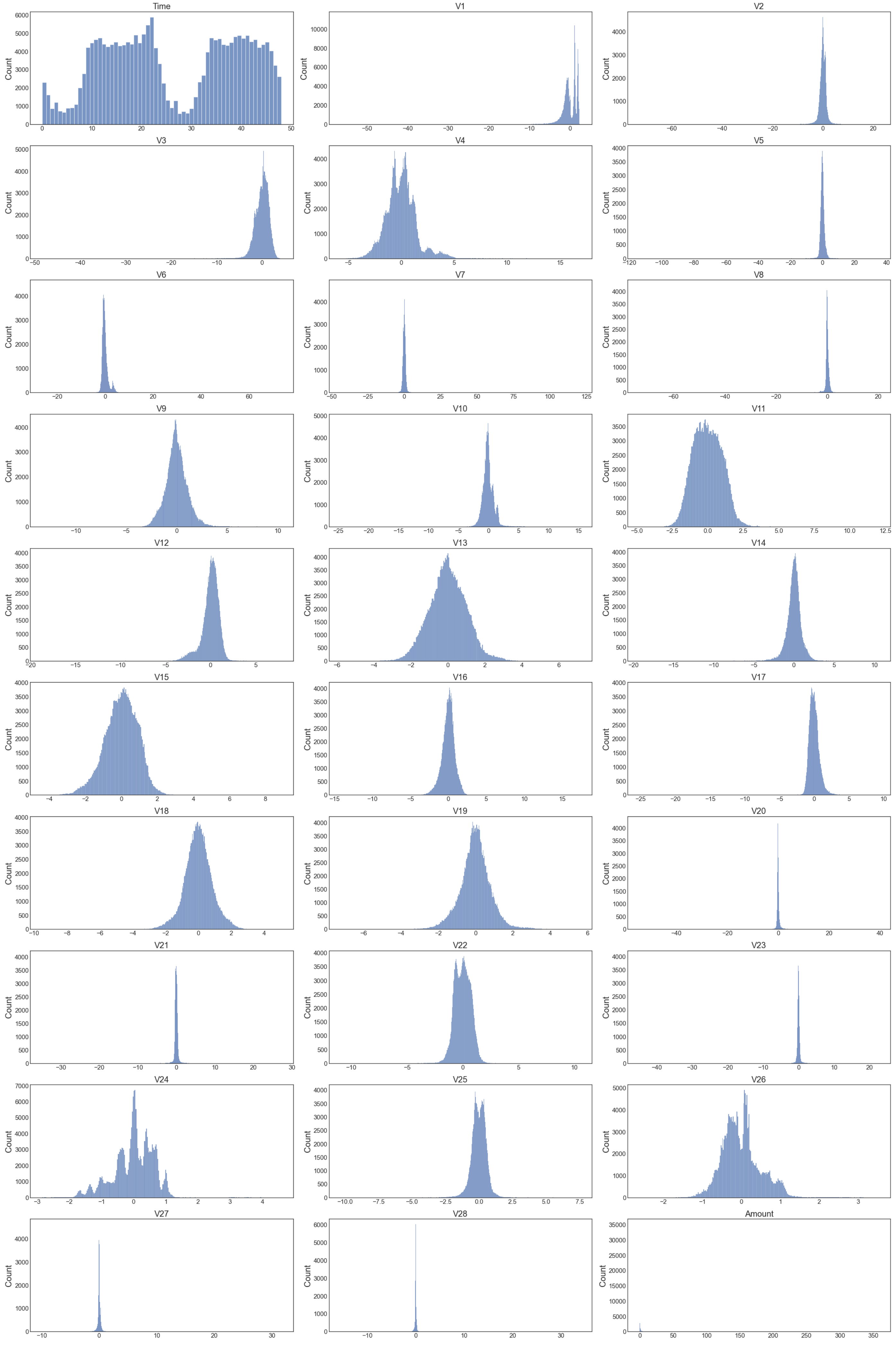
Apply scaling techniques on the features “Amount” to transform the range of values.

In [29]:
```
scaler = RobustScaler()

X_train_dummy[['Amount']] = scaler.fit_transform(X_train_dummy[['Amount']])
X_test_dummy[['Amount']] = scaler.transform(X_test_dummy[['Amount']])
```

Checking Skewness

The absence of symmetry in a distribution is known as its skewness. The Mean, Meadian, and Mode are all equal in a symmetrical distribution. There is no skewness in the normal distribution. The distribution of our data is revealed via skewness.
To improve the gaussianity of the distribution, the Power Transformer package is included in the preprocessing library offered by sklearn.

In [30]:
```
var = X_train_dummy.columns

with plt.style.context('seaborn-white'):
    fig, axes = plt.subplots(10, 3, figsize=(30, 45))
    axes = axes.flatten()

    for i, ax in enumerate(axes):
        if i < len(var):
            sns.histplot(X_train_dummy[var[i]], ax=ax)
            ax.set_title(var[i], fontsize=20)
            ax.set_ylabel('Count', fontsize=20)
            ax.tick_params(axis='both', labelsize=15)
            ax.set_xlabel('')

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


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

Upon conducting skewness analysis using the skew() function on the features, it was observed that many of them exhibit a substantial skew.
To mitigate the effects of skewness and improve the distributional properties of the data, we intend to apply a power transform. Specifically, we will apply the power transform if the skewness of a feature exceeds the threshold of -1 to 1.
This transformation aims to make the data more symmetric and alleviate the impact of extreme values, thereby enhancing the performance and interpretability of subsequent analyses and machine learning models.

Check the skewness of the features

```
In [31]: var = X_train_dummy.columns
         skew_list = []
         for i in var:
             skew_list.append(X_train_dummy[i].skew())
```

```
In [32]: check_skew = pd.concat([pd.DataFrame(var, columns=['Features']), pd.DataFrame(skew_list,
         check_skew.set_index('Features', inplace=True)
         check_skew
```

Out[32]:

	Skewness
Features	
Time	-0.033636
V1	-3.357011
V2	-5.035310
V3	-2.237770
V4	0.686692
V5	-3.189437
V6	2.063992
V7	4.253673
V8	-8.623674
V9	0.534814
V10	1.191228
V11	0.365317
V12	-2.266855
V13	0.070504
V14	-2.015958
V15	-0.309181
V16	-1.090862
V17	-3.822872
V18	-0.255628
V19	0.111665
V20	-1.929253
V21	3.175323

V22	-0.216657
V23	-6.782943
V24	-0.550056
V25	-0.424576
V26	0.577836
V27	0.229730
V28	12.062181
Amount	19.134850

In [33]:
```
skew_df = check_skew.loc[(check_skew['Skewness'] > 1) | (check_skew['Skewness'] <-1 )].i
skew_df.tolist()
```

Out[33]:
```
['V1',
 'V2',
 'V3',
 'V5',
 'V6',
 'V7',
 'V8',
 'V10',
 'V12',
 'V14',
 'V16',
 'V17',
 'V20',
 'V21',
 'V23',
 'V28',
 'Amount']
```

We will utilize the PowerTransformer package from the preprocessing module provided by the scikit-learn library to transform the distribution of the data into a more Gaussian or normal distribution.
This transformation aims to reduce skewness and make the distribution more symmetric, which can improve the performance of certain statistical and machine learning algorithms that assume Gaussian-distributed data.

Comparison of the models with scaling with Power Transformer

In [36]:
```
pt= preprocessing.PowerTransformer(method='yeo-johnson', copy=True)
pt.fit(X_train_dummy)

X_train_pt = pt.transform(X_train_dummy)
X_test_pt = pt.transform(X_test_dummy)

y_train_pt = y_train_dummy
y_test_pt = y_test_dummy
```

In [37]:
```
pt= preprocessing.PowerTransformer(method='yeo-johnson', copy=True)
pt.fit(X_train)

X_train = pt.transform(X_train)
X_test = pt.transform(X_test)

y_train = y_train
y_test = y_test
```

In [40]:
```
Models = [
    ('XGB Classifier', XGBClassifier(random_state=42)),
    ('Random Forest Classifier', RandomForestClassifier(random_state=42)),
    ('Light GBM Classifier', LGBMClassifier(random_state=42))
]

name_list = []
accuracy_scores = []
accuracy_scores_train = []
recall_scores = []
recall_scores_train = []
f1_scores = []
f1_scores_train = []
precision_scores = []
precision_scores_train = []

for name, model in Models:
    model.fit(X_train_pt, y_train_pt)
    y_pred = model.predict(X_test_pt)
    y_pred_train = model.predict(X_train_pt)
    accuracy_scores.append(accuracy_score(y_test_pt, y_pred))
    accuracy_scores_train.append(accuracy_score(y_train_pt, y_pred_train))
    recall_scores.append(recall_score(y_test_pt, y_pred))
    recall_scores_train.append(recall_score(y_train_pt, y_pred_train))
    f1_scores.append(f1_score(y_test_pt, y_pred))
    f1_scores_train.append(f1_score(y_train_pt, y_pred_train))
    precision_scores.append(precision_score(y_test_pt, y_pred))
    precision_scores_train.append(precision_score(y_train_pt, y_pred_train))
    name_list.append(name)

result = {
    'Model': name_list,
    'Accuracy Score Test': accuracy_scores,
    'Accuracy Score Train': accuracy_scores_train,
    'Recall Score Test': recall_scores,
    'Recall Score Train': recall_scores_train,
    'F1 Score Test': f1_scores,
    'F1 Score Train': f1_scores_train,
    'Precision Score Test': precision_scores,
    'Precision Score Train': precision_scores_train
}

dataframe = pd.DataFrame(result).sort_values(by='F1 Score Test', ascending=False)
dataframe.reset_index(drop=True)
```

```
[LightGBM] [Info] Number of positive: 356, number of negative: 199008
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.
075778 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 199364, number of used feature
s: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.001786 -> initscore=-6.326170
[LightGBM] [Info] Start training from score -6.326170
```

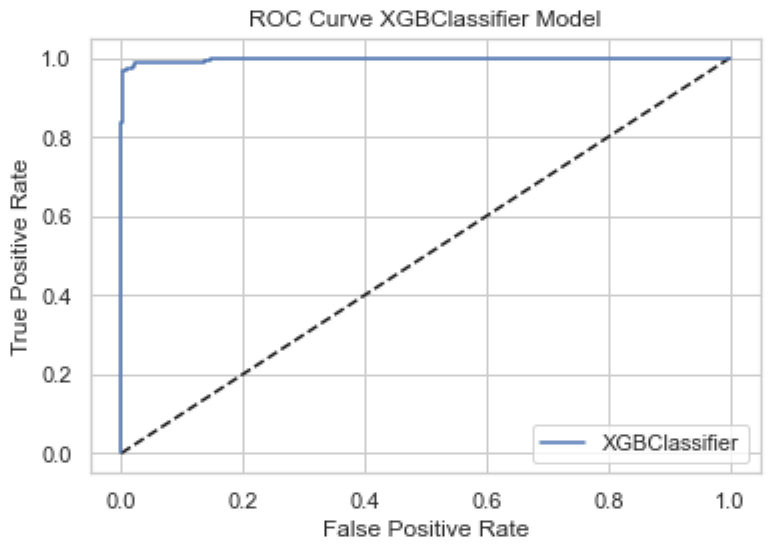
Out[40]:

	Model	Accuracy Score Test	Accuracy Score Train	Recall Score Test	Recall Score Train	F1 Score Test	F1 Score Train	Precision Score Test	Precision Score Train
0	XGB Classifier	0.999614	1.000000	0.816176	1.000000	0.870588	1.000000	0.932773	1.000000
1	Random Forest Classifier	0.999614	1.000000	0.801471	1.000000	0.868526	1.000000	0.947826	1.000000
2	Light GBM Classifier	0.993902	0.995952	0.647059	0.716292	0.252511	0.387244	0.156863	0.265349

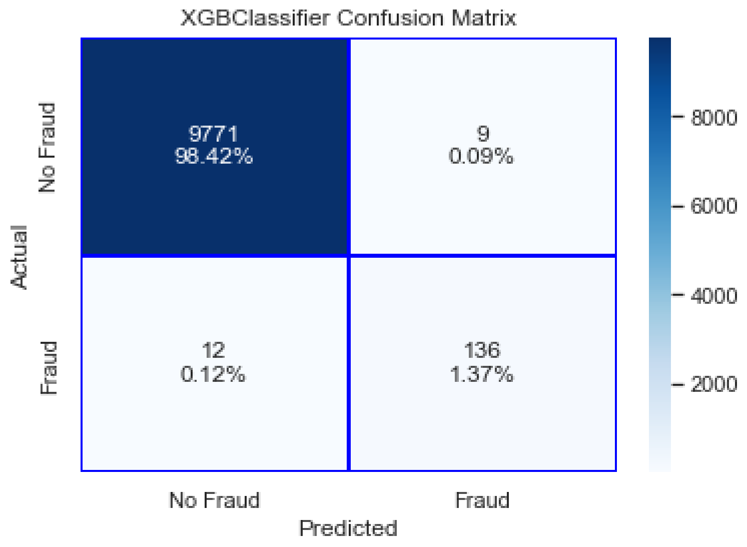
Comparing the recent and previous results, both XGB and Light GBM classifiers exhibit a slight decrease in recall and precision scores. While the accuracy remains high, the decline in recall suggests a potential increase in false negatives, impacting fraud detection.
However, despite this decrease, the F1 scores remain robust, indicating a balance between precision and recall. To address the decline in recall, further optimization of the models, such as adjusting hyperparameters or exploring ensemble techniques, may be necessary. Additionally, evaluating the impact of threshold adjustments on model performance could help mitigate false negatives while maintaining precision.

In [42]:
```python
XGB = XGBClassifier()
RF = RandomForestClassifier()
LightGBM = LGBMClassifier()

Models = [XGB,RF,LightGBM]

for model in Models:
    model.fit(X_train, y_train)
    y_pred_prob = model.predict_proba(X_test)[:, 1]
    threshold = 0.5
    y_pred = (y_pred_prob > threshold).astype(int)
    fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
    print('\n' )
    print(type(model).__name__+' Roc Curve, Classification Report and Confusion Matrix:\
    print(type(model).__name__, 'Model AUC Score is: ', auc(fpr, tpr))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.plot(fpr, tpr, label=type(model).__name__)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve ' + type(model).__name__ + ' Model')
    plt.legend()
    plt.show()

    # Convert probability scores to binary predictions using a threshold
    threshold = 0.5
    y_pred_binary = (y_pred_prob > threshold).astype(int)

    labelsNF = ['No Fraud', 'Fraud']
    report = classification_report(y_test, y_pred_binary, target_names=labelsNF)
    print(type(model).__name__,' Classification Report:\n', report)

    cm = confusion_matrix(y_test, y_pred_binary)

    counts = [value for value in cm.flatten()]
    percentages = ['{0:.2%}'.format(value) for value in cm.flatten() / np.sum(cm)]
    labels = [f'{v1}\n{v2}' for v1, v2 in zip(counts, percentages)]

    labels = np.array(labels).reshape(cm.shape)

    sns.heatmap(cm, annot=labels, fmt='', cmap='Blues', linewidths=1.5, linecolor='blue'
                xticklabels=labelsNF, yticklabels=labelsNF)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(type(model).__name__ + ' Confusion Matrix')
    plt.show()
```

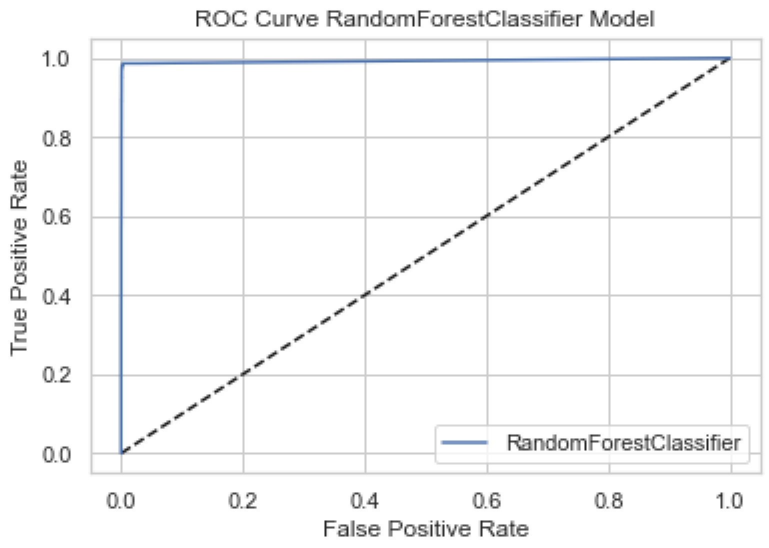
```
XGBClassifier Roc Curve, Classification Report and Confusion Matrix:

XGBClassifier Model AUC Score is:  0.9974755430276903
```



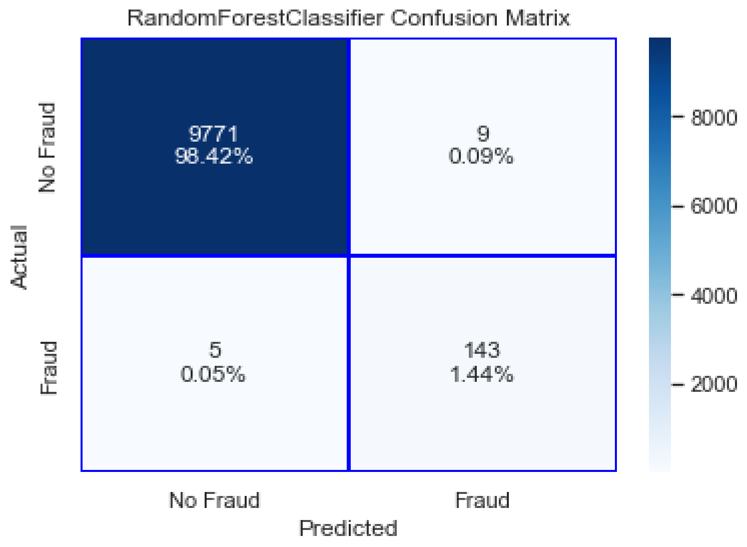
```
XGBClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9780
       Fraud       0.94      0.92      0.93       148

    accuracy                           1.00      9928
   macro avg       0.97      0.96      0.96      9928
weighted avg       1.00      1.00      1.00      9928
```



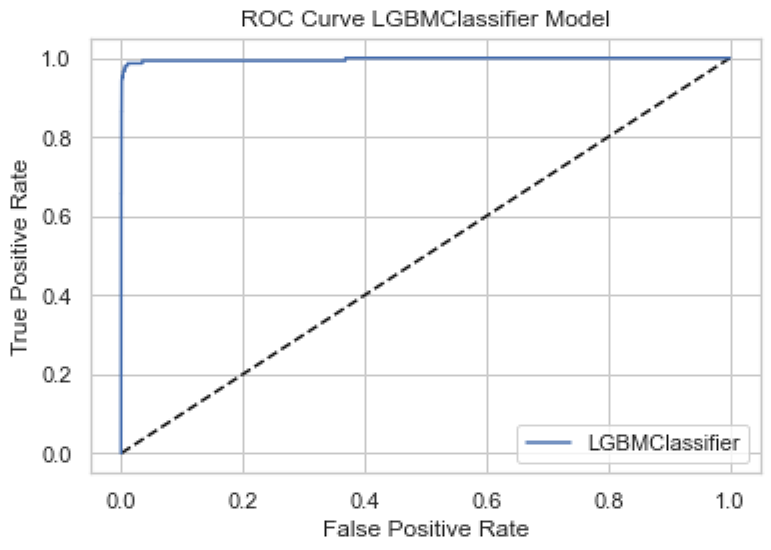
```
RandomForestClassifier Roc Curve, Classification Report and Confusion Matrix:

RandomForestClassifier Model AUC Score is:  0.9927903747305588
```



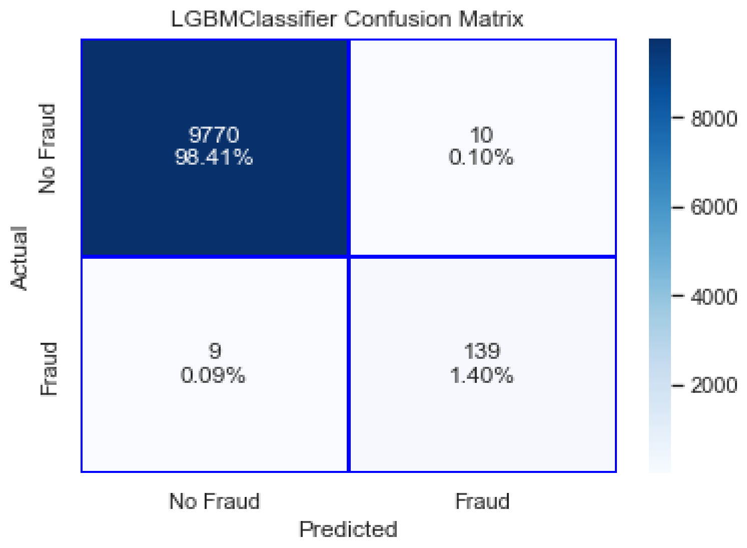
```
RandomForestClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9780
       Fraud       0.94      0.97      0.95       148

    accuracy                           1.00      9928
   macro avg       0.97      0.98      0.98      9928
weighted avg       1.00      1.00      1.00      9928
```



```
[LightGBM] [Info] Number of positive: 344, number of negative: 22821
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.
007091 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7650
[LightGBM] [Info] Number of data points in the train set: 23165, number of used feature
s: 30
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.014850 -> initscore=-4.194795
[LightGBM] [Info] Start training from score -4.194795


LGBMClassifier Roc Curve, Classification Report and Confusion Matrix:

LGBMClassifier Model AUC Score is:  0.9968281932238987
```



```
LGBMClassifier  Classification Report:
               precision    recall  f1-score   support

    No Fraud       1.00      1.00      1.00      9780
       Fraud       0.93      0.94      0.94       148

    accuracy                           1.00      9928
   macro avg       0.97      0.97      0.97      9928
weighted avg       1.00      1.00      1.00      9928
```



XGBClassifier and LGBMClassifier maintain high precision and recall, yielding balanced F1 scores, indicating robust fraud detection. In contrast, the previous LGBMClassifier exhibits lower precision, recall, and F1 score, suggesting a compromised fraud detection capability.

Summary & Conclusion:

- **ML Models:** XGB Classifier, Random Forest Classifier, SVC, Decision Tree Classifier, Light GBM Classifier, Gradient Boosting Classifier, Ada Boost Classifier, Logistic Regression, Hist Gradient Boosting Classifier.
- **Oversampling Techniques:** SMOTE, ADASYN.
- **Scalers & Transforms:** RobustScaler, PowerTransformer.
- Using a threshold of 0.5 and SMOTE, XGB and Random Forest classifier models stand out as the top performers (ROC-AUC scores of approximately 99.99%, recall and F1-score 100%), showcasing outstanding predictive accuracy and robustness.