In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score,f1_score, roc_auc_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_validate,StratifiedKFold,KFold
from sklearn import neighbors
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
rawdata = pd.read_csv('CVD_cleaned.csv')
#Number of rows and columns present in the dataset
rawdata.shape
```

Out[2]:
```
(308854, 19)
```

Data Exploration

In [3]:
```
rawdata.head()
```

Out[3]:

	General_Health	Checkup	Exercise	Heart_Disease	Skin_Cancer	Other_Cancer	Depression	Diab
0	Poor	Within the past 2 years	No	No	No	No	No	
1	Very Good	Within the past year	No	Yes	No	No	No	
2	Very Good	Within the past year	Yes	No	No	No	No	
3	Poor	Within the past year	Yes	Yes	No	No	No	
4	Good	Within the past year	No	No	No	No	No	

In [4]:
```
rawdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 308854 entries, 0 to 308853
Data columns (total 19 columns):
 #   Column                        Non-Null Count   Dtype  
---  ------                        --------------   -----  
 0   General_Health                308854 non-null  object 
 1   Checkup                       308854 non-null  object 
 2   Exercise                      308854 non-null  object 
 3   Heart_Disease                 308854 non-null  object 
 4   Skin_Cancer                   308854 non-null  object 
 5   Other_Cancer                  308854 non-null  object 
 6   Depression                    308854 non-null  object 
 7   Diabetes                      308854 non-null  object 
 8   Arthritis                     308854 non-null  object 
 9   Sex                           308854 non-null  object 
 10  Age_Category                  308854 non-null  object 
 11  Height_(cm)                   308854 non-null  float64
 12  Weight_(kg)                   308854 non-null  float64
 13  BMI                           308854 non-null  float64
 14  Smoking_History               308854 non-null  object 
 15  Alcohol_Consumption           308854 non-null  float64
 16  Fruit_Consumption             308854 non-null  float64
 17  Green_Vegetables_Consumption  308854 non-null  float64
 18  FriedPotato_Consumption       308854 non-null  float64
dtypes: float64(7), object(12)
memory usage: 44.8+ MB
```

Missing Values

In [5]:
```
# Check for missing values in the entire DataFrame
missing_values = rawdata.isnull().sum()

# Check if there are any missing values in the DataFrame
if missing_values.sum() > 0:
    print("Missing values are present.")
    print("Columns with missing values:")
    print(missing_values[missing_values > 0])
else:
    print("No missing values are present.")
```

```
No missing values are present.
```

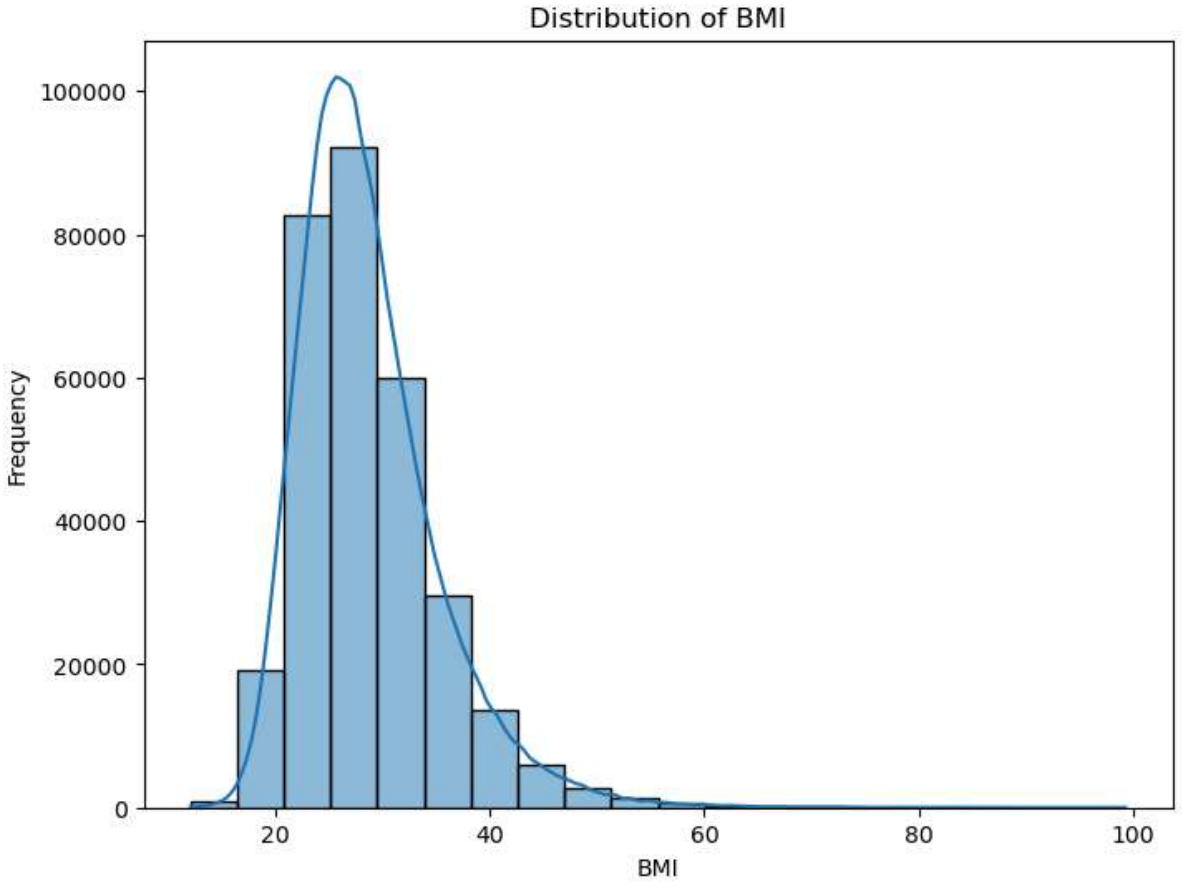
In [6]:
```
rawdata.describe()
```

Out[6]:

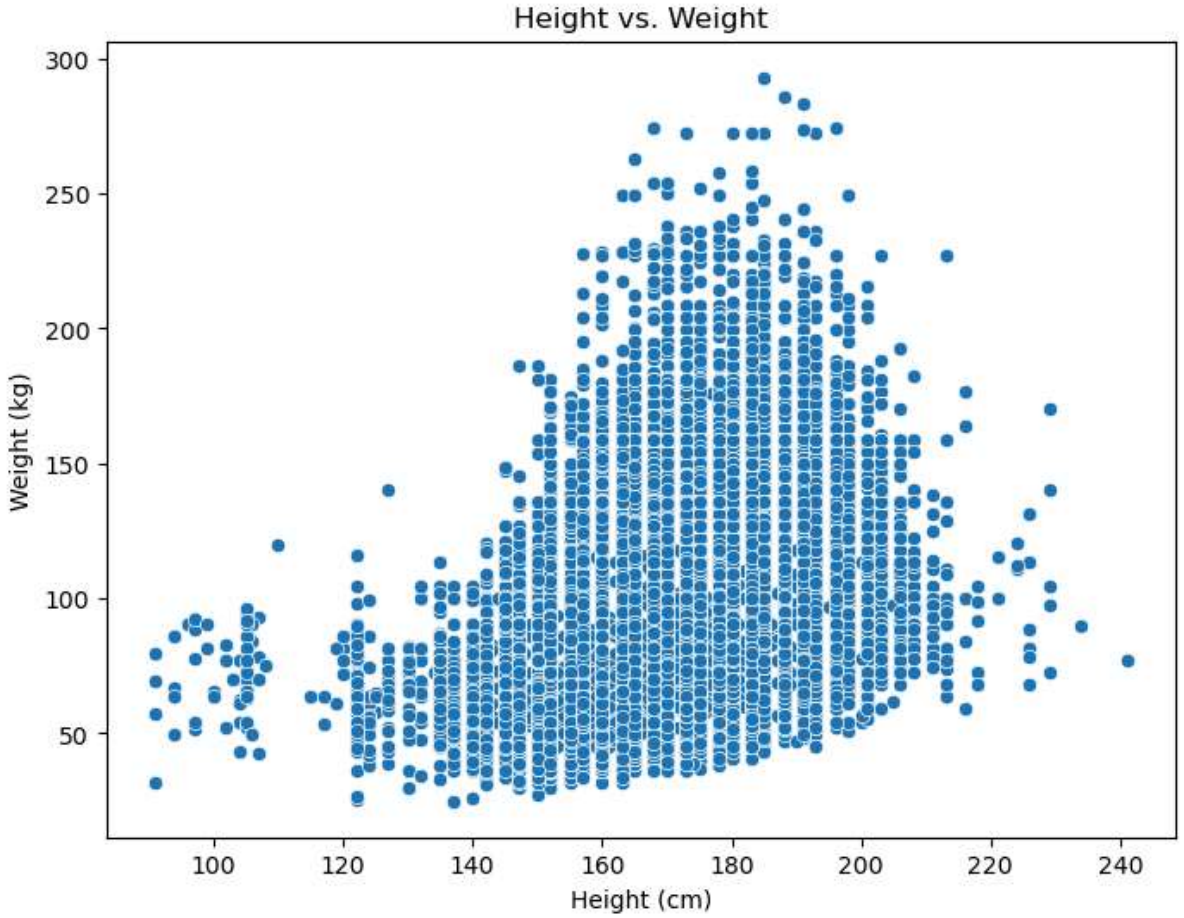
	Height_(cm)	Weight_(kg)	BMI	Alcohol_Consumption	Fruit_Consumption	Gr
count	308854.000000	308854.000000	308854.000000	308854.000000	308854.000000	
mean	170.615249	83.588655	28.626211	5.096366	29.835200	
std	10.658026	21.343210	6.522323	8.199763	24.875735	
min	91.000000	24.950000	12.020000	0.000000	0.000000	
25%	163.000000	68.040000	24.210000	0.000000	12.000000	
50%	170.000000	81.650000	27.440000	1.000000	30.000000	
75%	178.000000	95.250000	31.850000	6.000000	30.000000	
max	241.000000	293.020000	99.330000	30.000000	120.000000	

Visualization of Data

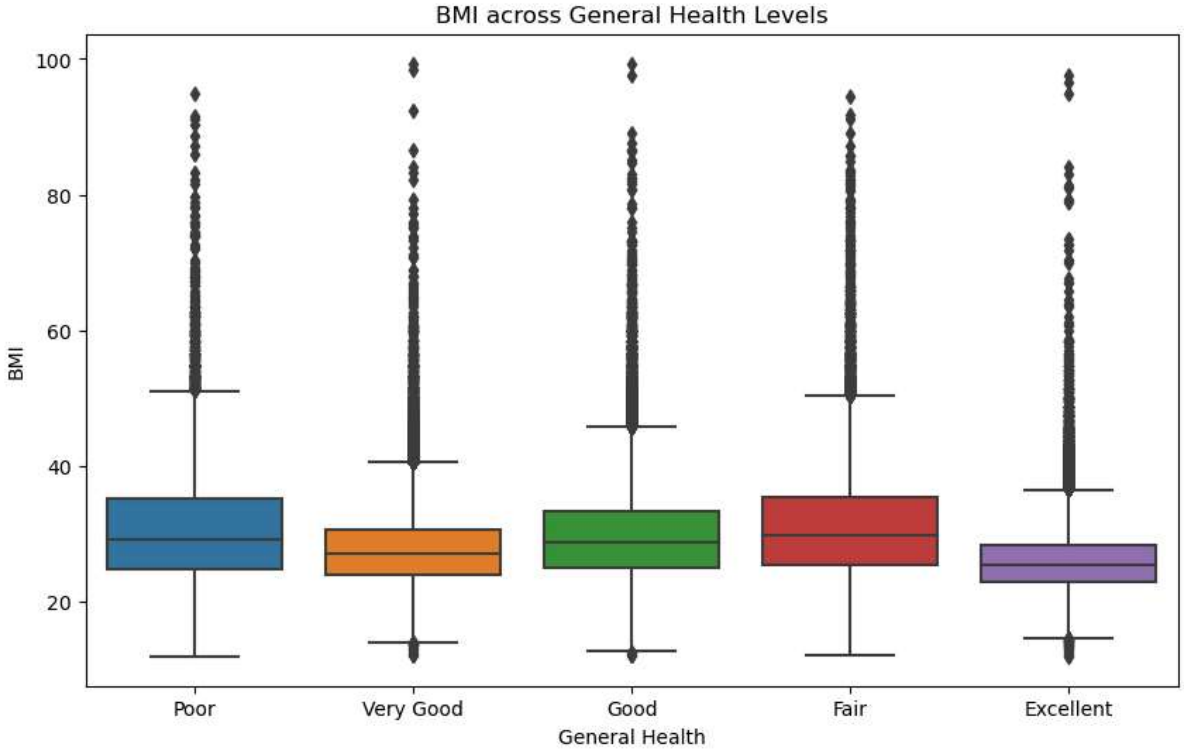
In [7]:
```
##Numerical Data
plt.figure(figsize=(8, 6))
sns.histplot(rawdata['BMI'], bins=20, kde=True)
plt.title('Distribution of BMI')
plt.xlabel('BMI')
plt.ylabel('Frequency')
plt.show()
```



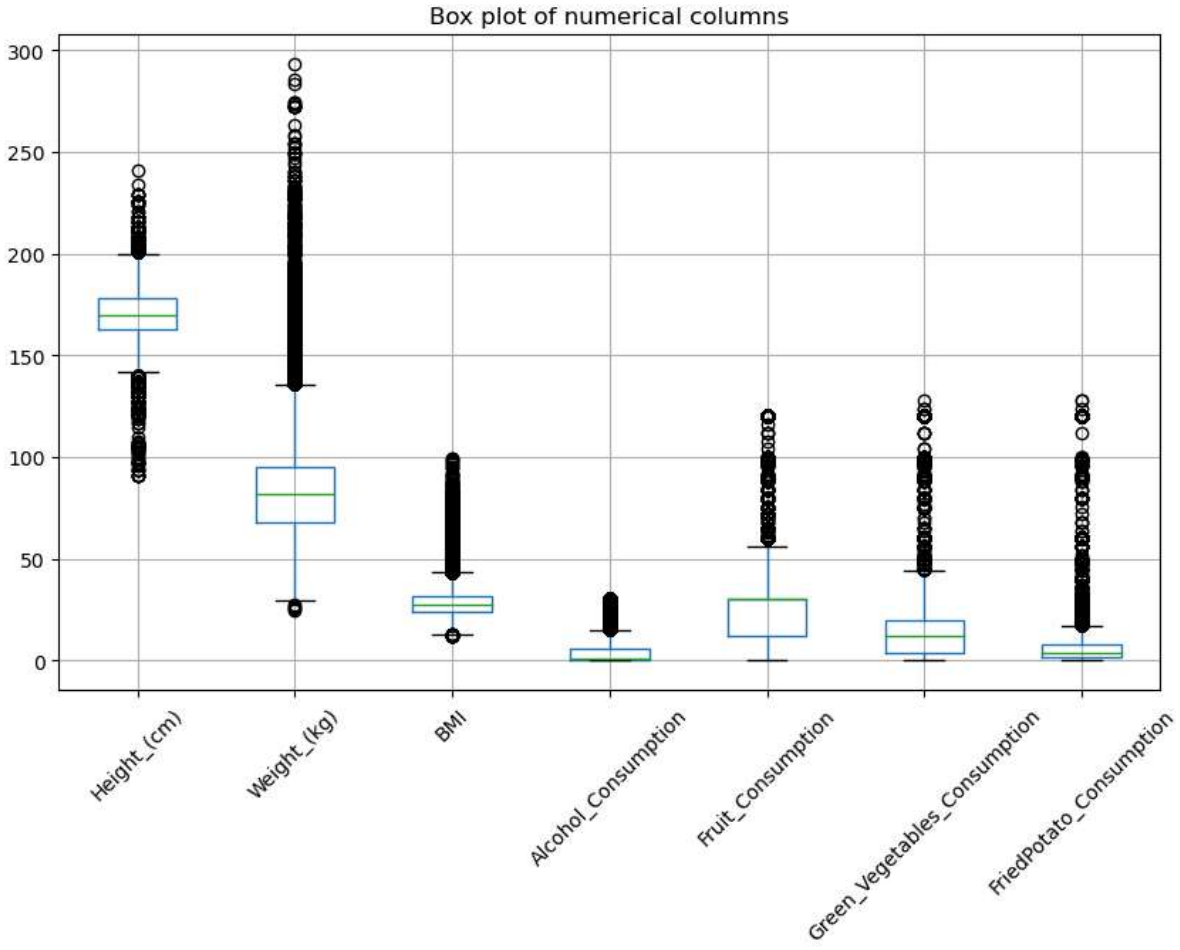
In [8]:
```
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Height_(cm)', y='Weight_(kg)', data=rawdata)
plt.title('Height vs. Weight')
plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')
plt.show()
```



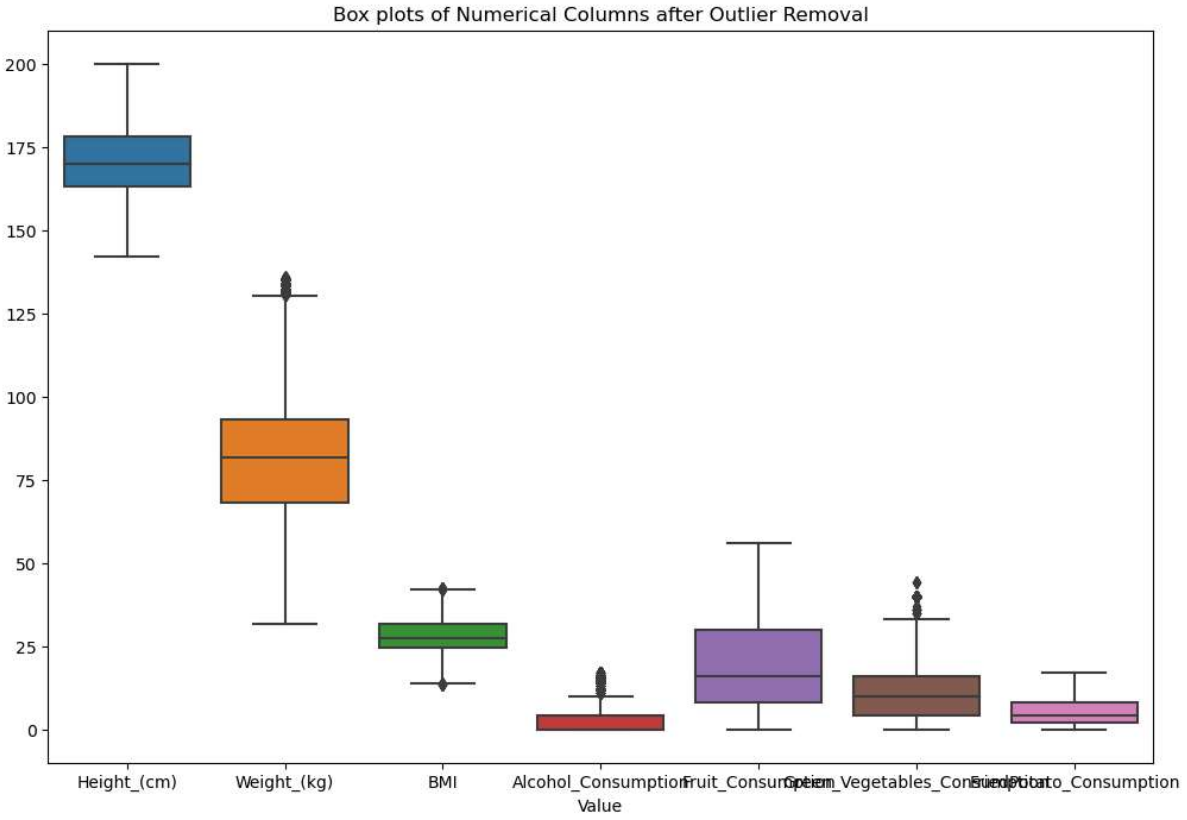
In [9]:
```
##Categorical data
plt.figure(figsize=(10, 6))
sns.boxplot(x='General_Health', y='BMI', data=rawdata)
plt.title('BMI across General Health Levels')
plt.xlabel('General Health')
plt.ylabel('BMI')
plt.show()
```



In [10]:
```
# Checkig outliers
plt.figure(figsize=(10, 6))
rawdata.boxplot()
plt.title("Box plot of numerical columns")
plt.xticks(rotation=45)
plt.show()
```

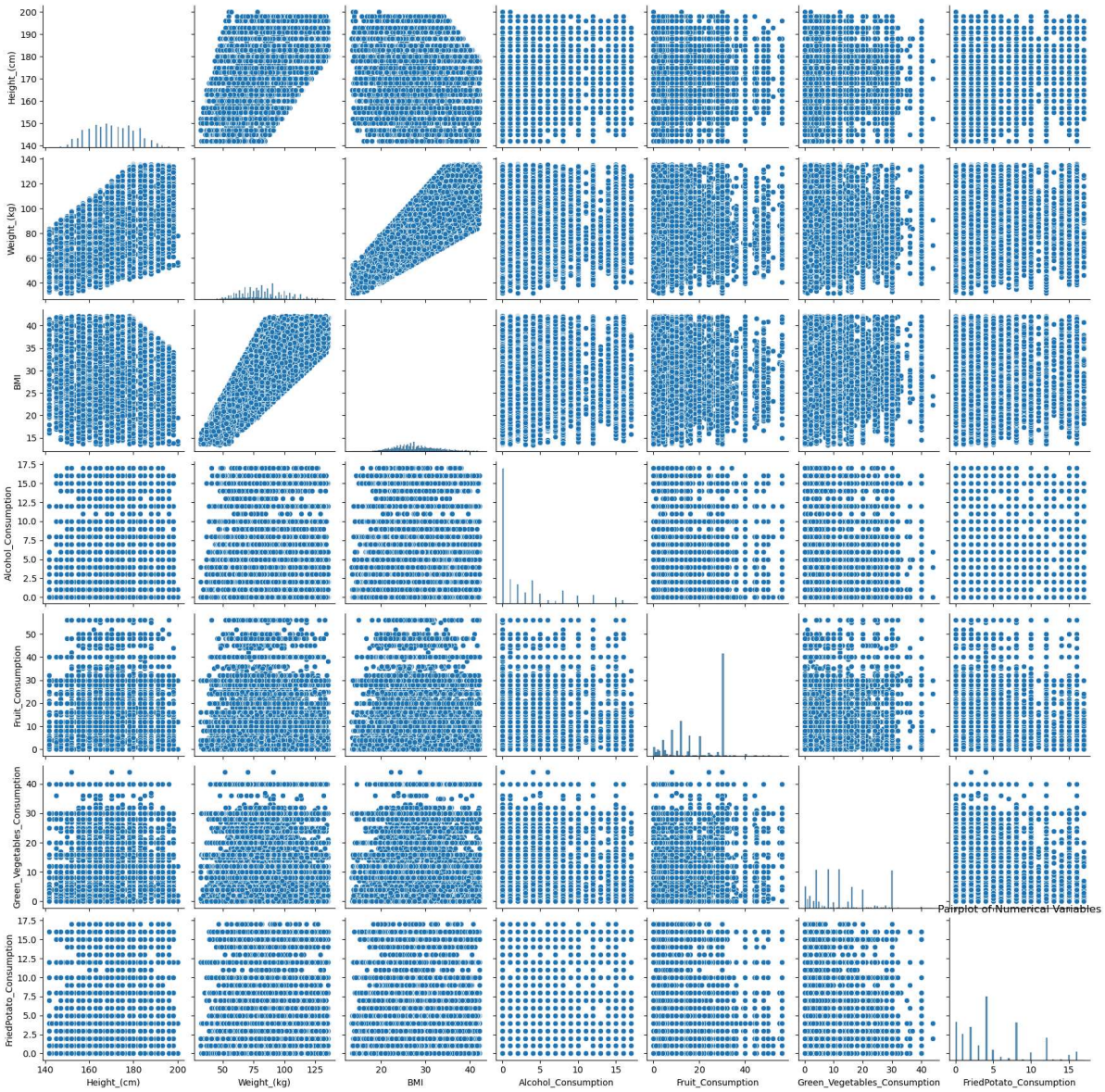


In [11]:
```
#Removing all the outliers
df = rawdata.copy()

# Define a function to detect and remove outliers using IQR for a specific column
def remove_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df_cleaned = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    return df_cleaned

# Apply the function to remove outliers from all numerical columns
for column in df.select_dtypes(include=['float64', 'int64']).columns:
    df = remove_outliers_iqr(df, column)

# Visualize box plots for all numerical columns after outlier removal
plt.figure(figsize=(12, 8))
sns.boxplot(data=df)
plt.title('Box plots of Numerical Columns after Outlier Removal')
plt.xlabel('Value')
plt.show()
```



In [12]:
```
# Scatter plot to visually inspect outliers in numerical variables
sns.pairplot(df)
plt.title('Pairplot of Numerical Variables')
plt.show()
```

```
/Users/yashwanth/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:118: U
serWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



Feature Engineering

In [13]:
```
# encoding
df['General_Health'] = df['General_Health'].replace({
    'Very Good':5,
    'Good':4,
    'Excellent':3,
    'Fair':2,
    'Poor':1
})



df['Sex'] = df['Sex'].replace({
    'Male':0,
    'Female':1
})

df['Checkup'] = df['Checkup'].replace({
    'Never':0,
    'Within the past year':1,
    'Within the past 2 years':2,
    'Within the past 5 years':3,
    '5 or more years ago':4
})
```

```
age_category_df = pd.DataFrame(df, columns=['Age_Category'])

df_dummies = pd.get_dummies(age_category_df, columns=['Age_Category'])

df=df.join(df_dummies)

df = df.drop('Age_Category_80+',axis=1)



df['Diabetes'] = df['Diabetes'].replace({
    'Yes, but female told only during pregnancy':1,
    'No, pre-diabetes or borderline diabetes':0,
    'Yes':1,
    'No':0
})

df['Exercise'] = df['Exercise'].replace({
    'Yes':1,
    'No': 0
})



df['Skin_Cancer'] = df['Skin_Cancer'].replace({
    'Yes':1,
    'No': 0
})

df['Other_Cancer'] = df['Other_Cancer'].replace({
    'Yes':1,
    'No': 0
})

df['Depression'] = df['Depression'].replace({
    'Yes':1,
    'No': 0
})
df['Arthritis'] = df['Arthritis'].replace({
    'Yes':1,
    'No': 0
})
df['Smoking_History'] = df['Smoking_History'].replace({
    'Yes':1,
    'No': 0
})
```

In [14]:
```
df= df.drop('Age_Category',axis=1)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 185803 entries, 0 to 308853
Data columns (total 30 columns):
 #   Column                        Non-Null Count   Dtype
---  ------                        --------------   -----
 0   General_Health                185803 non-null  int64
 1   Checkup                       185803 non-null  int64
 2   Exercise                      185803 non-null  int64
 3   Heart_Disease                 185803 non-null  object
 4   Skin_Cancer                   185803 non-null  int64
 5   Other_Cancer                  185803 non-null  int64
 6   Depression                    185803 non-null  int64
 7   Diabetes                      185803 non-null  int64
 8   Arthritis                     185803 non-null  int64
 9   Sex                           185803 non-null  int64
 10  Height_(cm)                   185803 non-null  float64
 11  Weight_(kg)                   185803 non-null  float64
 12  BMI                           185803 non-null  float64
 13  Smoking_History               185803 non-null  int64
 14  Alcohol_Consumption           185803 non-null  float64
 15  Fruit_Consumption             185803 non-null  float64
 16  Green_Vegetables_Consumption  185803 non-null  float64
 17  FriedPotato_Consumption       185803 non-null  float64
 18  Age_Category_18-24            185803 non-null  bool
 19  Age_Category_25-29            185803 non-null  bool
 20  Age_Category_30-34            185803 non-null  bool
 21  Age_Category_35-39            185803 non-null  bool
 22  Age_Category_40-44            185803 non-null  bool
 23  Age_Category_45-49            185803 non-null  bool
 24  Age_Category_50-54            185803 non-null  bool
 25  Age_Category_55-59            185803 non-null  bool
 26  Age_Category_60-64            185803 non-null  bool
 27  Age_Category_65-69            185803 non-null  bool
 28  Age_Category_70-74            185803 non-null  bool
 29  Age_Category_75-79            185803 non-null  bool
dtypes: bool(12), float64(7), int64(10), object(1)
memory usage: 33.1+ MB
```

In [15]:
```
df.tail()
```

Out[15]:

	General_Health	Checkup	Exercise	Heart_Disease	Skin_Cancer	Other_Cancer	Depression
308848	4	3	1	No	0	0	0
308849	5	1	1	No	0	0	0
308851	5	4	1	No	0	0	1
308852	5	1	1	No	0	0	0
308853	3	1	1	No	0	0	0

5 rows × 30 columns

In [16]:
```
numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns
scaler = MinMaxScaler()
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
df.head(5)
```

Out[16]:

	General_Health	Checkup	Exercise	Heart_Disease	Skin_Cancer	Other_Cancer	Depression	Diab
0	0.00	0.50	0.0	No	0.0	0.0	0.0	
1	1.00	0.25	0.0	Yes	0.0	0.0	0.0	
2	1.00	0.25	1.0	No	0.0	0.0	0.0	
3	0.00	0.25	1.0	Yes	0.0	0.0	0.0	
4	0.75	0.25	0.0	No	0.0	0.0	0.0	

5 rows × 30 columns

Split Data

In [17]:
```
from sklearn import preprocessing
before_encoding = df['Heart_Disease']
label_e = preprocessing.LabelEncoder()
label_e.fit(before_encoding)
y = label_e.transform(before_encoding)
df['Heart_Disease'] = y
X = df.drop('Heart_Disease',axis=1)

#Splitting data for Training and Testing
X_train_temp, X_test, y_train_temp, y_test = train_test_split(X, y, test_size=0.25,
#Splitting data for Training and Validation
X_train, X_val, y_train, y_val = train_test_split(X_train_temp, y_train_temp, test_
```

Model Selection and Training

Classifier

In [ ]:

In [ ]:

In [45]:
```
replace_cols = ['Age_Category_18-24', 'Age_Category_25-29',
       'Age_Category_30-34', 'Age_Category_35-39', 'Age_Category_40-44',
       'Age_Category_45-49', 'Age_Category_50-54', 'Age_Category_55-59',
       'Age_Category_60-64', 'Age_Category_65-69', 'Age_Category_70-74',
       'Age_Category_75-79']

for i in replace_cols:
     X[i].replace([True, False], [1, 0], inplace=True)

x = np.array(X)
```

In [ ]:

In [ ]:

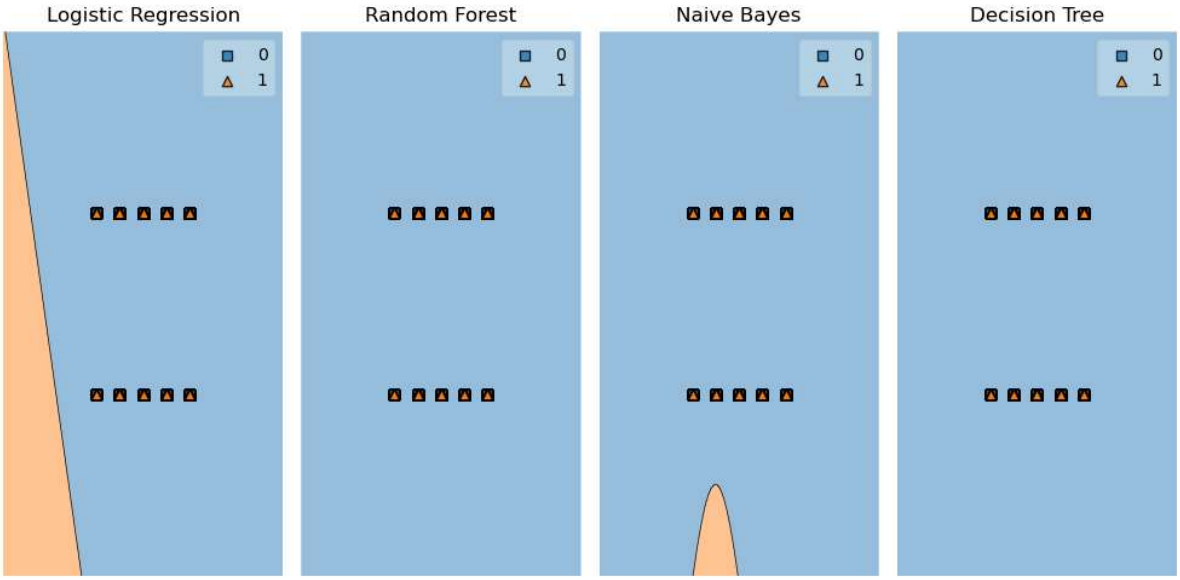
In [ ]:

In [ ]:
```
# USINF THE MODEL_SELCETION FUCNTION FROM SKLEARN TO DETERMINE THE BEST MODEL AND T
```

In [30]:
```
from sklearn import model_selection

# Create three different classifiers
clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()
clf4 = DecisionTreeClassifier(random_state=1)

# Print a header for the cross-validation results
print('5-fold cross validation:\n')

# Create a list of labels for the classifiers
labels = ['Logistic Regression', 'Random Forest', 'Naive Bayes', 'Decision Tree']

# Loop through the classifiers and perform 5-fold cross-validation for each
for clf, label in zip([clf1, clf2, clf3, clf4], labels):

    # Use cross_val_score to compute accuracy scores using 5-fold cross-validation
    scores = model_selection.cross_val_score(clf, X_train, y_train,
                                             cv=5,
                                             scoring='accuracy')

    # Print the mean accuracy and standard deviation of accuracy for the current cl
    print("Accuracy: %0.2f (+/- %0.2f) [%s]"
          % (scores.mean(), scores.std(), label))
```

```
5-fold cross validation:

Accuracy: 0.92 (+/- 0.00) [Logistic Regression]
Accuracy: 0.91 (+/- 0.00) [Random Forest]
Accuracy: 0.57 (+/- 0.00) [Naive Bayes]
Accuracy: 0.85 (+/- 0.00) [Decision Tree]
```

In [76]:
```
clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()
clf4 = DecisionTreeClassifier(random_state=1)


clf1.fit(X_train[:,1:3], y_train)
clf2.fit(X_train[:,1:3], y_train)
clf3.fit(X_train[:,1:3], y_train)
clf4.fit(X_train[:,1:3], y_train)

classifiers = [clf1, clf2, clf3, clf4]
titles = ['Logistic Regression', 'Random Forest', 'Naive Bayes', 'Decision Tree']

fig, axes = plt.subplots(1, 4, figsize=(10, 5))
plt.subplots_adjust(wspace=0.4, hspace=0.4)



for clf, title, ax in zip(classifiers, titles, axes.flatten()):
    plot_decision_regions(X_train[:,1:3], y_train, clf=clf, ax=ax)
    ax.set_title(title)
    ax.set_axis_off()

plt.tight_layout()
plt.savefig('3 classifiers.png')
plt.show()
```



In [ ]:
```
# Based on the above model selection method, we chose only logistic regression & ra
# we will use an ensemble model of both the classifiers and use a voting method for
```

In [78]:
```
from mlxtend.plotting import plot_decision_regions

# Create a Hard Voting Classifier
voting_clf_hard = VotingClassifier(
    estimators=[
        (labels[0], clf1),  # Include the first classifier (Logistic Regression)
        (labels[1], clf2),  # Include the second classifier (Random Forest)
        (labels[3], clf4),
    ],
    voting='hard'  # Specify hard voting, where the majority class prediction is ch
)



# Create a Soft Voting Classifier
voting_clf_soft = VotingClassifier(
    estimators=[
        (labels[0], clf1),  # Include the first classifier (Logistic Regression)
        (labels[1], clf2),  # Include the second classifier (Random Forest)
        (labels[3], clf4),
    ],
    voting='soft'  # Specify soft voting, where class probabilities are combined
)



# Create a new list of labels that includes Voting Classifiers
labels_new = ['Logistic Regression', 'Random Forest',
              'Voting_Classifier_Hard', 'Voting_Classifier_Soft']

# Loop through the classifiers and perform 5-fold cross-validation for each
for clf, label in zip([clf1, clf2, clf4, voting_clf_hard, voting_clf_soft], labels_

    # Use cross_val_score to compute accuracy scores using 5-fold cross-validation
    scores = model_selection.cross_val_score(clf, X_train, y_train, cv=5, scoring='

    # Print the mean accuracy and standard deviation of accuracy for the current cl
    print("Accuracy: %0.2f (+/- %0.2f) [%s]"
          % (scores.mean(), scores.std(), label))
```

```
Accuracy: 0.92 (+/- 0.00) [Logistic Regression]
Accuracy: 0.91 (+/- 0.00) [Random Forest]
Accuracy: 0.85 (+/- 0.00) [Voting_Classifier_Hard]
Accuracy: 0.91 (+/- 0.00) [Voting_Classifier_Soft]
```

In [81]:

```
## DO HYPERPARAMETER TURNING FOR LOGISTIC REGRESSION BELOW (SAME AS HOW YOU'VE DONE

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_validate, StratifiedKFold
import numpy as np

# Define hyperparameter values to tune
penalty_values = ['l1', 'l2']
C_values = [0.001, 0.01, 0.1, 1, 10, 100]

# Define N-fold cross-validation
n_splits = 10
kk = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=22)

# Perform cross-validation for each hyperparameter setting
for penalty in penalty_values:
    for C in C_values:
        clf = LogisticRegression(penalty=penalty, C=C, solver='liblinear')
        scores = cross_validate(clf, X_train, y_train,
                                scoring=['accuracy', 'precision', 'recall', 'roc_au
                                cv=kk, n_jobs=None, return_estimator=True)

        print('Penalty:', penalty, 'C:', C)
        print('Accuracy:', np.mean(scores['test_accuracy']),
              'Precision:', np.mean(scores['test_precision']),
              'Recall:', np.mean(scores['test_recall']),
              'ROC-AUC:', np.mean(scores['test_roc_auc']))
```

```
/Users/yashwanth/anaconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/yashwanth/anaconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/yashwanth/anaconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/yashwanth/anaconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/yashwanth/anaconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/yashwanth/anaconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/yashwanth/anaconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/yashwanth/anaconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/yashwanth/anaconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/yashwanth/anaconda3/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
Penalty: l1 C: 0.001
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.746536370328890
4
Penalty: l1 C: 0.01
Accuracy: 0.9156524618976822 Precision: 0.5286461736197093 Recall: 0.0378787008998
40696 ROC-AUC: 0.8235450798761599
Penalty: l1 C: 0.1
Accuracy: 0.9155591664720202 Precision: 0.5141492279639289 Recall: 0.0578771939895
80786 ROC-AUC: 0.8298239961180107
Penalty: l1 C: 1
Accuracy: 0.9153080054929461 Precision: 0.500419564304617 Recall: 0.06101257193702
5493 ROC-AUC: 0.8299550875795442
Penalty: l1 C: 10
Accuracy: 0.9153797662140569 Precision: 0.5035970714275612 Recall: 0.0617752838014
3228 ROC-AUC: 0.8299549194622508
Penalty: l1 C: 100
Accuracy: 0.9153654143787973 Precision: 0.5030230003099226 Recall: 0.0616058640336
39976 ROC-AUC: 0.8299436504291725
Penalty: l2 C: 0.001
Accuracy: 0.9155017529517272 Precision: 0.5682094557190465 Recall: 0.0072878485626
94643 ROC-AUC: 0.8040694757044546
Penalty: l2 C: 0.01
Accuracy: 0.9155806955122541 Precision: 0.5173819694747368 Recall: 0.0421156302472
768 ROC-AUC: 0.8226511331967613
Penalty: l2 C: 0.1
Accuracy: 0.9156022245524881 Precision: 0.5169972191763289 Recall: 0.0556738759167
0374 ROC-AUC: 0.8297430077164032
Penalty: l2 C: 1
Accuracy: 0.9153295335033041 Precision: 0.5013825709955108 Recall: 0.0608430804116
0176 ROC-AUC: 0.8299673934268146
Penalty: l2 C: 10
Accuracy: 0.9153438863684393 Precision: 0.501884712032429 Recall: 0.06177528380143
227 ROC-AUC: 0.8299504134249247
Penalty: l2 C: 100
Accuracy: 0.9153582387186369 Precision: 0.5025097120324291 Recall: 0.0619447753268
5601 ROC-AUC: 0.8299480941404911
```

In [84]:

```
## DO HYPERPARAMETER TURNING FOR RANDOM FOREST BELOW

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_validate, StratifiedKFold
import numpy as np

# Define hyperparameter values to tune
n_estimators_values = [100, 200, 300]  # Example values for number of trees
max_depth_values = [None, 5, 10, 15, 20]
min_samples_leaf_values = [1, 5, 10, 15, 20]

# Define N-fold cross-validation
n_splits = 10
kk = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=22)

# Perform cross-validation for each hyperparameter setting
for n_estimators in n_estimators_values:
    for max_depth in max_depth_values:
        for min_samples_leaf in min_samples_leaf_values:
            clf = RandomForestClassifier(n_estimators=n_estimators,
                                         max_depth=max_depth,
                                         min_samples_leaf=min_samples_leaf,
                                         random_state=42)
            scores = cross_validate(clf, X_train, y_train,
                                    scoring=['accuracy', 'precision', 'recall', 'ro
                                    cv=kk, n_jobs=None, return_estimator=True)
```

```
            print('n_estimators:', n_estimators, 'Max Depth:', max_depth, 'Min Samp
            print('Accuracy:', np.mean(scores['test_accuracy']),
                  'Precision:', np.mean(scores['test_precision']),
                  'Recall:', np.mean(scores['test_recall']),
                  'ROC-AUC:', np.mean(scores['test_roc_auc']))
```

```
n_estimators: 100 Max Depth: None Min Samples Leaf: 1
Accuracy: 0.9146693377527366 Precision: 0.4573869036828272 Recall: 0.0362686031659
467 ROC-AUC: 0.8030296004121675
n_estimators: 100 Max Depth: None Min Samples Leaf: 5
Accuracy: 0.9153941170194402 Precision: 0.5204425039128384 Recall: 0.0146602276152
06878 ROC-AUC: 0.8199992044667382
n_estimators: 100 Max Depth: None Min Samples Leaf: 10
Accuracy: 0.9154586984758248 Precision: 0.5693256692455777 Recall: 0.0083047259576
05591 ROC-AUC: 0.8234563791543309
n_estimators: 100 Max Depth: None Min Samples Leaf: 15
Accuracy: 0.9153582351140711 Precision: 0.5950840825840826 Recall: 0.0051692762525
29456 ROC-AUC: 0.8241567954861366
n_estimators: 100 Max Depth: None Min Samples Leaf: 20
Accuracy: 0.9152936490232451 Precision: 0.5378205128205129 Recall: 0.0028812841745
71966 ROC-AUC: 0.8249901502205572
n_estimators: 100 Max Depth: 5 Min Samples Leaf: 1
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.802887986238949
1
n_estimators: 100 Max Depth: 5 Min Samples Leaf: 5
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.802806464335207
1
n_estimators: 100 Max Depth: 5 Min Samples Leaf: 10
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.802929392753987
1
n_estimators: 100 Max Depth: 5 Min Samples Leaf: 15
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.802867178897571
1
n_estimators: 100 Max Depth: 5 Min Samples Leaf: 20
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.802909552682515
1
n_estimators: 100 Max Depth: 10 Min Samples Leaf: 1
Accuracy: 0.9153151734290377 Precision: 0.5539502164502165 Recall: 0.0030507039423
642705 ROC-AUC: 0.8174440231496231
n_estimators: 100 Max Depth: 10 Min Samples Leaf: 5
Accuracy: 0.9153295278389866 Precision: 0.6088095238095238 Recall: 0.0022031745576
142022 ROC-AUC: 0.8177405710746459
n_estimators: 100 Max Depth: 10 Min Samples Leaf: 10
Accuracy: 0.915322350634012 Precision: 0.5152777777777777 Recall: 0.00161009773389
39996 ROC-AUC: 0.818051624769069
n_estimators: 100 Max Depth: 10 Min Samples Leaf: 15
Accuracy: 0.915279294613296 Precision: 0.3055555555555555 Recall: 0.00101694915254
2373 ROC-AUC: 0.8178678869592485
n_estimators: 100 Max Depth: 10 Min Samples Leaf: 20
Accuracy: 0.9152362396224557 Precision: 0.3083333333333333 Recall: 0.0004237288135
59322 ROC-AUC: 0.8181328653775812
n_estimators: 100 Max Depth: 15 Min Samples Leaf: 1
Accuracy: 0.9154228227497109 Precision: 0.5262762476433851 Recall: 0.0172870592287
4898 ROC-AUC: 0.8197281431568639
n_estimators: 100 Max Depth: 15 Min Samples Leaf: 5
Accuracy: 0.9154587000206389 Precision: 0.5675304325304324 Recall: 0.0100842434592
91894 ROC-AUC: 0.821988154173237
n_estimators: 100 Max Depth: 15 Min Samples Leaf: 10
Accuracy: 0.9153223537236398 Precision: 0.5330820783104064 Recall: 0.0062708276525
20845 ROC-AUC: 0.8225501922505334
n_estimators: 100 Max Depth: 15 Min Samples Leaf: 15
Accuracy: 0.9153869362099003 Precision: 0.5936125541125541 Recall: 0.0046607299186
26845 ROC-AUC: 0.8231317252155943
n_estimators: 100 Max Depth: 15 Min Samples Leaf: 20
Accuracy: 0.9153654087144801 Precision: 0.5141666666666667 Recall: 0.0026271186440
677964 ROC-AUC: 0.8230788669491046
n_estimators: 100 Max Depth: 20 Min Samples Leaf: 1
Accuracy: 0.9150711927445656 Precision: 0.4820307641309915 Recall: 0.0289808263608
83484 ROC-AUC: 0.8135301607199621
n_estimators: 100 Max Depth: 20 Min Samples Leaf: 5
```

```
Accuracy: 0.9152505878531496 Precision: 0.4942028176930136 Recall: 0.0124567660270
66978 ROC-AUC: 0.8215910894394437
n_estimators: 100 Max Depth: 20 Min Samples Leaf: 10
Accuracy: 0.9154156424551088 Precision: 0.5687479990298326 Recall: 0.0077963231389
6583 ROC-AUC: 0.8231769379512069
n_estimators: 100 Max Depth: 20 Min Samples Leaf: 15
Accuracy: 0.9154012911347873 Precision: 0.5629889116731223 Recall: 0.0049999999999
99999 ROC-AUC: 0.823699909846163
n_estimators: 100 Max Depth: 20 Min Samples Leaf: 20
Accuracy: 0.9153223526937639 Precision: 0.5430555555555555 Recall: 0.0027966101694
91525 ROC-AUC: 0.8241098484595811
n_estimators: 200 Max Depth: None Min Samples Leaf: 1
Accuracy: 0.9147913270650967 Precision: 0.4616911357335741 Recall: 0.0329636619354
46836 ROC-AUC: 0.8081094884277377
n_estimators: 200 Max Depth: None Min Samples Leaf: 5
Accuracy: 0.9153797656991189 Precision: 0.5243695357104968 Recall: 0.0127960361084
40134 ROC-AUC: 0.8216681495875339
n_estimators: 200 Max Depth: None Min Samples Leaf: 10
Accuracy: 0.9154299942903681 Precision: 0.5719039302630635 Recall: 0.0074573400881
18372 ROC-AUC: 0.8242114466156056
n_estimators: 200 Max Depth: None Min Samples Leaf: 15
Accuracy: 0.9153510573941587 Precision: 0.5624898785425101 Recall: 0.0042372163779
61796 ROC-AUC: 0.8251793791501143
n_estimators: 200 Max Depth: None Min Samples Leaf: 20
Accuracy: 0.915386937239776 Precision: 0.5793650793650793 Recall: 0.00313559322033
8983 ROC-AUC: 0.825451210272252
n_estimators: 200 Max Depth: 5 Min Samples Leaf: 1
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.803441721146784
6
n_estimators: 200 Max Depth: 5 Min Samples Leaf: 5
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.803419071326873
8
n_estimators: 200 Max Depth: 5 Min Samples Leaf: 10
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.803527297308713
3
n_estimators: 200 Max Depth: 5 Min Samples Leaf: 15
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.803528351862523
8
n_estimators: 200 Max Depth: 5 Min Samples Leaf: 20
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.803538120120967
3
n_estimators: 200 Max Depth: 10 Min Samples Leaf: 1
Accuracy: 0.9153367029842094 Precision: 0.5847835497835497 Recall: 0.0027965384118
60101 ROC-AUC: 0.8177696675874628
n_estimators: 200 Max Depth: 10 Min Samples Leaf: 5
Accuracy: 0.9153367024692715 Precision: 0.5783333333333334 Recall: 0.0022031027999
827777 ROC-AUC: 0.8180097853946299
n_estimators: 200 Max Depth: 10 Min Samples Leaf: 10
Accuracy: 0.915322351663888 Precision: 0.5347222222222221 Recall: 0.00169484349660
58642 ROC-AUC: 0.818288493642549
n_estimators: 200 Max Depth: 10 Min Samples Leaf: 15
Accuracy: 0.9153079993136906 Precision: 0.5458333333333333 Recall: 0.0011864406779
661016 ROC-AUC: 0.81818079503619
n_estimators: 200 Max Depth: 10 Min Samples Leaf: 20
Accuracy: 0.9152864718182705 Precision: 0.4 Recall: 0.0003389830508474576 ROC-AUC:
0.8182261401531953
n_estimators: 200 Max Depth: 15 Min Samples Leaf: 1
Accuracy: 0.91551610890649 Precision: 0.5456236434536292 Recall: 0.016100690308414
302 ROC-AUC: 0.8212896784309285
n_estimators: 200 Max Depth: 15 Min Samples Leaf: 5
Accuracy: 0.9154299937754301 Precision: 0.5527702155829555 Recall: 0.0094062055999
65559 ROC-AUC: 0.8228170726822857
n_estimators: 200 Max Depth: 15 Min Samples Leaf: 10
Accuracy: 0.9153797610646774 Precision: 0.5768731471368388 Recall: 0.0064402474203
```

```
1315 ROC-AUC: 0.8233534718848693
n_estimators: 200 Max Depth: 15 Min Samples Leaf: 15
Accuracy: 0.9153510579090967 Precision: 0.5635714285714286 Recall: 0.0038982333271
143387 ROC-AUC: 0.8237023874126865
n_estimators: 200 Max Depth: 15 Min Samples Leaf: 20
Accuracy: 0.9153438801891841 Precision: 0.5361111111111111 Recall: 0.0025423728813
55932 ROC-AUC: 0.8235716341184176
n_estimators: 200 Max Depth: 20 Min Samples Leaf: 1
Accuracy: 0.9151860058818271 Precision: 0.48781222604320396 Recall: 0.026353635959
184257 ROC-AUC: 0.8161095527603403
n_estimators: 200 Max Depth: 20 Min Samples Leaf: 5
Accuracy: 0.9152936474784313 Precision: 0.4986046236644063 Recall: 0.0116096671881
05455 ROC-AUC: 0.8225217263195675
n_estimators: 200 Max Depth: 20 Min Samples Leaf: 10
Accuracy: 0.9154084657650723 Precision: 0.5680604029968822 Recall: 0.0069489372694
78608 ROC-AUC: 0.8241807872504928
n_estimators: 200 Max Depth: 20 Min Samples Leaf: 15
Accuracy: 0.9154228196600833 Precision: 0.6041780441780442 Recall: 0.0047456909542
32982 ROC-AUC: 0.8245538294414233
n_estimators: 200 Max Depth: 20 Min Samples Leaf: 20
Accuracy: 0.9153151760037274 Precision: 0.5402411477411477 Recall: 0.0027118644067
796608 ROC-AUC: 0.8248328087588016
n_estimators: 300 Max Depth: None Min Samples Leaf: 1
Accuracy: 0.915013785918466 Precision: 0.48019513127426006 Recall: 0.0338957218100
145 ROC-AUC: 0.8095685069798474
n_estimators: 300 Max Depth: None Min Samples Leaf: 5
Accuracy: 0.9154515264202298 Precision: 0.5454106980218152 Recall: 0.0133043671694
48472 ROC-AUC: 0.8222362385486303
n_estimators: 300 Max Depth: None Min Samples Leaf: 10
Accuracy: 0.9154012901049114 Precision: 0.56805173992674 Recall: 0.007287848562694
643 ROC-AUC: 0.8244049545983512
n_estimators: 300 Max Depth: None Min Samples Leaf: 15
Accuracy: 0.9153725854045167 Precision: 0.5855772005772006 Recall: 0.0044067079033
855245 ROC-AUC: 0.8253477944078138
n_estimators: 300 Max Depth: None Min Samples Leaf: 20
Accuracy: 0.9153295273240488 Precision: 0.5452380952380953 Recall: 0.0027966101694
91525 ROC-AUC: 0.825582319878029
n_estimators: 300 Max Depth: 5 Min Samples Leaf: 1
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.803657182397480
3
n_estimators: 300 Max Depth: 5 Min Samples Leaf: 5
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.803620866345304
4
n_estimators: 300 Max Depth: 5 Min Samples Leaf: 10
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.803689829894632
4
n_estimators: 300 Max Depth: 5 Min Samples Leaf: 15
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.803711082818287
7
n_estimators: 300 Max Depth: 5 Min Samples Leaf: 20
Accuracy: 0.9153151734290376 Precision: 0.0 Recall: 0.0 ROC-AUC: 0.803691257316212
3
n_estimators: 300 Max Depth: 10 Min Samples Leaf: 1
Accuracy: 0.9153223501190741 Precision: 0.5506493506493506 Recall: 0.0022880638355
88915 ROC-AUC: 0.8180194060153912
n_estimators: 300 Max Depth: 10 Min Samples Leaf: 5
Accuracy: 0.9153295273240488 Precision: 0.5847619047619047 Recall: 0.0022031745576
142022 ROC-AUC: 0.8180632851223955
n_estimators: 300 Max Depth: 10 Min Samples Leaf: 10
Accuracy: 0.9153510548194689 Precision: 0.6413888888888889 Recall: 0.0016948434966
058642 ROC-AUC: 0.8183017541674948
n_estimators: 300 Max Depth: 10 Min Samples Leaf: 15
Accuracy: 0.9153295268091108 Precision: 0.3483333333333333 Recall: 0.0012711864406
779662 ROC-AUC: 0.8182544683884254
```

```
n_estimators: 300 Max Depth: 10 Min Samples Leaf: 20
Accuracy: 0.915279294613296 Precision: 0.2833333333333333 Recall: 0.00033898305084
74576 ROC-AUC: 0.8183114857122955
n_estimators: 300 Max Depth: 15 Min Samples Leaf: 1
Accuracy: 0.9155663416172428 Precision: 0.5586198899140076 Recall: 0.0159313422982
5342 ROC-AUC: 0.8214106723003614
n_estimators: 300 Max Depth: 15 Min Samples Leaf: 5
Accuracy: 0.9154443461256276 Precision: 0.5559069707570458 Recall: 0.0092367858321
73253 ROC-AUC: 0.8231172062257155
n_estimators: 300 Max Depth: 15 Min Samples Leaf: 10
Accuracy: 0.9153582335692574 Precision: 0.5615384615384615 Recall: 0.0058472423542
24371 ROC-AUC: 0.8235928557084062
n_estimators: 300 Max Depth: 15 Min Samples Leaf: 15
Accuracy: 0.9153438817339978 Precision: 0.5729978354978356 Recall: 0.0038134875644
024744 ROC-AUC: 0.823869791102531
n_estimators: 300 Max Depth: 15 Min Samples Leaf: 20
Accuracy: 0.9153295273240488 Precision: 0.5310515873015873 Recall: 0.0026271186440
677964 ROC-AUC: 0.8237141788239206
n_estimators: 300 Max Depth: 20 Min Samples Leaf: 1
Accuracy: 0.9152362411672696 Precision: 0.4940162391427445 Recall: 0.0254217913575
10874 ROC-AUC: 0.816519549943717
n_estimators: 300 Max Depth: 20 Min Samples Leaf: 5
Accuracy: 0.9153654081995419 Precision: 0.5083602971102971 Recall: 0.0114399603897
87455 ROC-AUC: 0.8228800862084457
n_estimators: 300 Max Depth: 20 Min Samples Leaf: 10
Accuracy: 0.9153582330543195 Precision: 0.5481601731601732 Recall: 0.0061862254050
718294 ROC-AUC: 0.8243855678008949
n_estimators: 300 Max Depth: 20 Min Samples Leaf: 15
Accuracy: 0.9153725864343926 Precision: 0.5617604617604618 Recall: 0.0045761994288
09253 ROC-AUC: 0.8249610557042153
n_estimators: 300 Max Depth: 20 Min Samples Leaf: 20
Accuracy: 0.9153080003435667 Precision: 0.5342857142857144 Recall: 0.0026271186440
677964 ROC-AUC: 0.8251242685267552
```

In [85]:

```
#   HYPERPARAMETER TURNING FOR DECISION TREE (FINDING THE BEST HYPERPARAMETER FOR 1

# model selection and training simultaneously through the process of hyperparameter
max_depth_values = [None, 5, 10, 15, 20]
min_samples_leaf_values = [1, 5, 10, 15, 20]

# Define N-fold cross-validation
n_splits = 10
kk = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=22)

# Perform cross-validation for each hyperparameter setting
for max_depth in max_depth_values:
    for min_samples_leaf in min_samples_leaf_values:
        clf = DecisionTreeClassifier(max_depth=max_depth, min_samples_leaf=min_samp
        scores = cross_validate(clf, X_train, y_train,
                                scoring=['accuracy', 'precision', 'recall', 'roc_au
                                cv=kk, n_jobs=None, return_estimator=True)

        print('Max Depth:', max_depth, 'Min Samples Leaf:', min_samples_leaf)
        print('Accuracy:', np.mean(scores['test_accuracy']),
              'Precision:', np.mean(scores['test_precision']),
              'Recall:', np.mean(scores['test_recall']),
              'ROC-AUC:', np.mean(scores['test_roc_auc']))
```

```
Max Depth: None Min Samples Leaf: 1
Accuracy: 0.8533856870734511 Precision: 0.19172393174784677 Recall: 0.227269406851
41864 ROC-AUC: 0.5693244410450394
Max Depth: None Min Samples Leaf: 5
Accuracy: 0.8847594549814477 Precision: 0.24630016100731403 Recall: 0.174647598272
07624 ROC-AUC: 0.633330180615004
Max Depth: None Min Samples Leaf: 10
Accuracy: 0.8993053682897922 Precision: 0.2801758470635311 Recall: 0.1204987872960
2892 ROC-AUC: 0.6811488024637051
Max Depth: None Min Samples Leaf: 15
Accuracy: 0.9051897268233644 Precision: 0.3180882940377058 Recall: 0.1043963030468
2904 ROC-AUC: 0.7118116547595573
Max Depth: None Min Samples Leaf: 20
Accuracy: 0.9084692084621194 Precision: 0.3466850639982084 Recall: 0.0915181762080
3973 ROC-AUC: 0.7298390685196249
Max Depth: 5 Min Samples Leaf: 1
Accuracy: 0.9150568460586855 Precision: 0.48219155804114544 Recall: 0.016099255155
785818 ROC-AUC: 0.7714225769212407
Max Depth: 5 Min Samples Leaf: 5
Accuracy: 0.9150568460586855 Precision: 0.48219155804114544 Recall: 0.016099255155
785818 ROC-AUC: 0.7714225769212407
Max Depth: 5 Min Samples Leaf: 10
Accuracy: 0.9150568460586855 Precision: 0.48219155804114544 Recall: 0.016099255155
785818 ROC-AUC: 0.7714289552773523
Max Depth: 5 Min Samples Leaf: 15
Accuracy: 0.9150568460586855 Precision: 0.48219155804114544 Recall: 0.016099255155
785818 ROC-AUC: 0.7714289552773523
Max Depth: 5 Min Samples Leaf: 20
Accuracy: 0.9150568460586855 Precision: 0.48219155804114544 Recall: 0.016099255155
785818 ROC-AUC: 0.7714289552773523
Max Depth: 10 Min Samples Leaf: 1
Accuracy: 0.9112104630651496 Precision: 0.3423517969664099 Recall: 0.0531309289742
9641 ROC-AUC: 0.7720446046272191
Max Depth: 10 Min Samples Leaf: 5
Accuracy: 0.9113683353127549 Precision: 0.34916561992499884 Recall: 0.054063419394
652625 ROC-AUC: 0.7673397535828956
Max Depth: 10 Min Samples Leaf: 10
Accuracy: 0.9119352459364187 Precision: 0.3615800992875855 Recall: 0.0516067968828
48485 ROC-AUC: 0.7723002740258813
Max Depth: 10 Min Samples Leaf: 15
Accuracy: 0.9120357103280485 Precision: 0.37183749112499875 Recall: 0.056098250548
94587 ROC-AUC: 0.7767834016739682
Max Depth: 10 Min Samples Leaf: 20
Accuracy: 0.9124232196638731 Precision: 0.38586343423734293 Recall: 0.057793022287
920326 ROC-AUC: 0.7783099144774936
Max Depth: 15 Min Samples Leaf: 1
Accuracy: 0.8949710156883052 Precision: 0.24890475511791244 Recall: 0.119226524490
8796 ROC-AUC: 0.64408789654284
Max Depth: 15 Min Samples Leaf: 5
Accuracy: 0.9000588574062698 Precision: 0.2836945767976516 Recall: 0.1174484421418
2179 ROC-AUC: 0.6544137406463822
Max Depth: 15 Min Samples Leaf: 10
Accuracy: 0.9047232950095921 Precision: 0.30290750948379613 Recall: 0.095755249070
73868 ROC-AUC: 0.6939460018967981
Max Depth: 15 Min Samples Leaf: 15
Accuracy: 0.9079668700259569 Precision: 0.3363282002237812 Recall: 0.0887209919774
968 ROC-AUC: 0.7182425628226351
Max Depth: 15 Min Samples Leaf: 20
Accuracy: 0.9099618271357464 Precision: 0.35920770833667776 Recall: 0.080417198869
09972 ROC-AUC: 0.7340327156008882
Max Depth: 20 Min Samples Leaf: 1
Accuracy: 0.8748564775279025 Precision: 0.21699495983467337 Recall: 0.183121313451
68559 ROC-AUC: 0.5713835800094915
Max Depth: 20 Min Samples Leaf: 5
```

```
Accuracy: 0.8907227663743056 Precision: 0.26103569516675224 Recall: 0.158123824968
78544 ROC-AUC: 0.6218550914486654
Max Depth: 20 Min Samples Leaf: 10
Accuracy: 0.901264437314957 Precision: 0.28734101400535517 Recall: 0.1120250003587
8815 ROC-AUC: 0.6745161425535471
Max Depth: 20 Min Samples Leaf: 15
Accuracy: 0.9058570992639682 Precision: 0.32022534590639556 Recall: 0.099058324602
82152 ROC-AUC: 0.7080342683066219
Max Depth: 20 Min Samples Leaf: 20
Accuracy: 0.9089499972605303 Precision: 0.3519729446247373 Recall: 0.0894000344436
631 ROC-AUC: 0.7269094148365879
```

MODEL EVALUATION ON TEST DATA AFTER CHOOSING THE BEST HYPERPARAMETERS

In [88]:
```
from sklearn.metrics import confusion_matrix as cm


clf1 = LogisticRegression(penalty='l1', C=0.01, solver='liblinear')
clf2 = RandomForestClassifier(n_estimators=100,
                                        max_depth=10,
                                        min_samples_leaf=10)
clf3 = DecisionTreeClassifier(max_depth=5, min_samples_leaf=5)


# Create a Soft Voting Classifier
voting_clf_soft = VotingClassifier(
    estimators=[
        (labels[0], clf1),  # Include the first classifier (Logistic Regression)
        (labels[1], clf2),  # Include the second classifier (Random Forest)
        (labels[3], clf3),
    ],
    voting='soft'  # Specify soft voting, where class probabilities are combined
)




## TRAINING AND EVALUATING ON TEST DATA


clf_vsoft=voting_clf_soft.fit(X_train, y_train)
y_pred=clf_vsoft.predict(X_test)

confM = cm(y_test, y_pred)
print(confM)

acc=accuracy_score(y_test, y_pred)
f1= f1_score(y_test, y_pred)
auc = roc_auc_score(y_test,y_pred)
print('Accuracy = ',acc, ', F1 = ', f1, ',AUC= ', auc)
```

```
[[42469    48]
 [ 3865    69]]
Accuracy =  0.9157606940647134 , F1 =  0.03406566279930881 ,AUC=  0.50820521996052
2
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: