Identifying Age-Related Conditions

1.1. Import Libraries

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter
%matplotlib inline
import seaborn as sns
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
import random
from scipy.stats import uniform, randint

# Suppress any warnings
import warnings
warnings.filterwarnings('ignore')

# Sklearn
from sklearn.model_selection import train_test_split, RandomizedSearchCV, StratifiedKFold, GridSearchCV
from sklearn import metrics

# kNN Imputation
from sklearn.impute import KNNImputer

# Feature Selection
from sklearn.feature_selection import SelectKBest, f_classif

# Data Encoder and Scaler
import category_encoders as encoders
from sklearn.preprocessing import LabelEncoder, RobustScaler

# SMOTE (Synthetic Minority Over-sampling Technique)
from imblearn.over_sampling import SMOTE
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is req
uired for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

1.2. Load the Data

In [3]:
```
# Read the train, test and greeks data

df_train = pd.read_csv("/input/icr-identify-age-related-conditions/train.csv")
df_test = pd.read_csv("/input/icr-identify-age-related-conditions/test.csv")
df_greeks = pd.read_csv("/input/icr-identify-age-related-conditions/greeks.csv")

print('No. of records for train : {}'.format(df_train.shape))
print('No. of records for test : {}'.format(df_test.shape))
print('No. of records for greeks : {}'.format(df_greeks.shape))
```

```
No. of records for train : (617, 58)
No. of records for test : (5, 57)
No. of records for greeks : (617, 6)
```

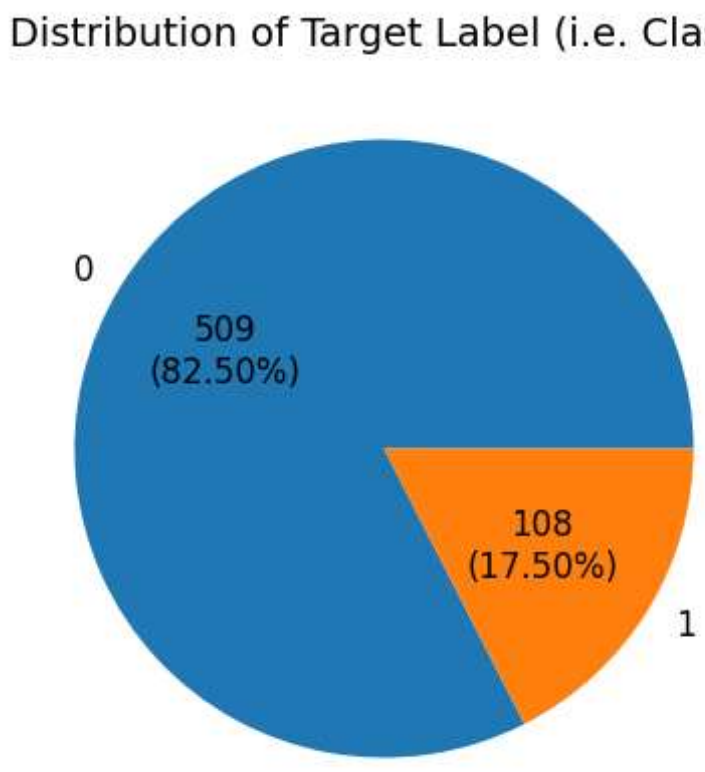
2. EDA

2.1. Distribution of Target Label

To check if the distribution of target feature (i.e. Class) is balanced. If it is not the case, we will do oversampling of SMOTE (Synthetic Minority Over-sampling Technique) algorithm in the modelling process.

In [4]:
```
# check whether the data set is balanced

plt.figure(figsize=(5,5))

def auto_fmt (pct_value):
    return '{:.0f}\n({:.2f}%)'.format(df_train['Class'].value_counts().sum()*pct_value/100,pct_value)

df_transported_count = df_train['Class'].value_counts().rename_axis('Class').reset_index(name='Counts')

fig = plt.gcf()
plt.pie(x=df_transported_count['Counts'], labels=df_transported_count['Class'], autopct=auto_fmt, textprops={'fontsize'
plt.title('Distribution of Target Label (i.e. Class)',  fontsize = 14)
```

Out[4]:
```
Text(0.5, 1.0, 'Distribution of Target Label (i.e. Class)')
```



Observation: the distribution of target feature between 1 and 0 cases is uneven. It may be necessary to do oversampling, e.g. SMOTE (Synthetic Minority Over-sampling Technique).

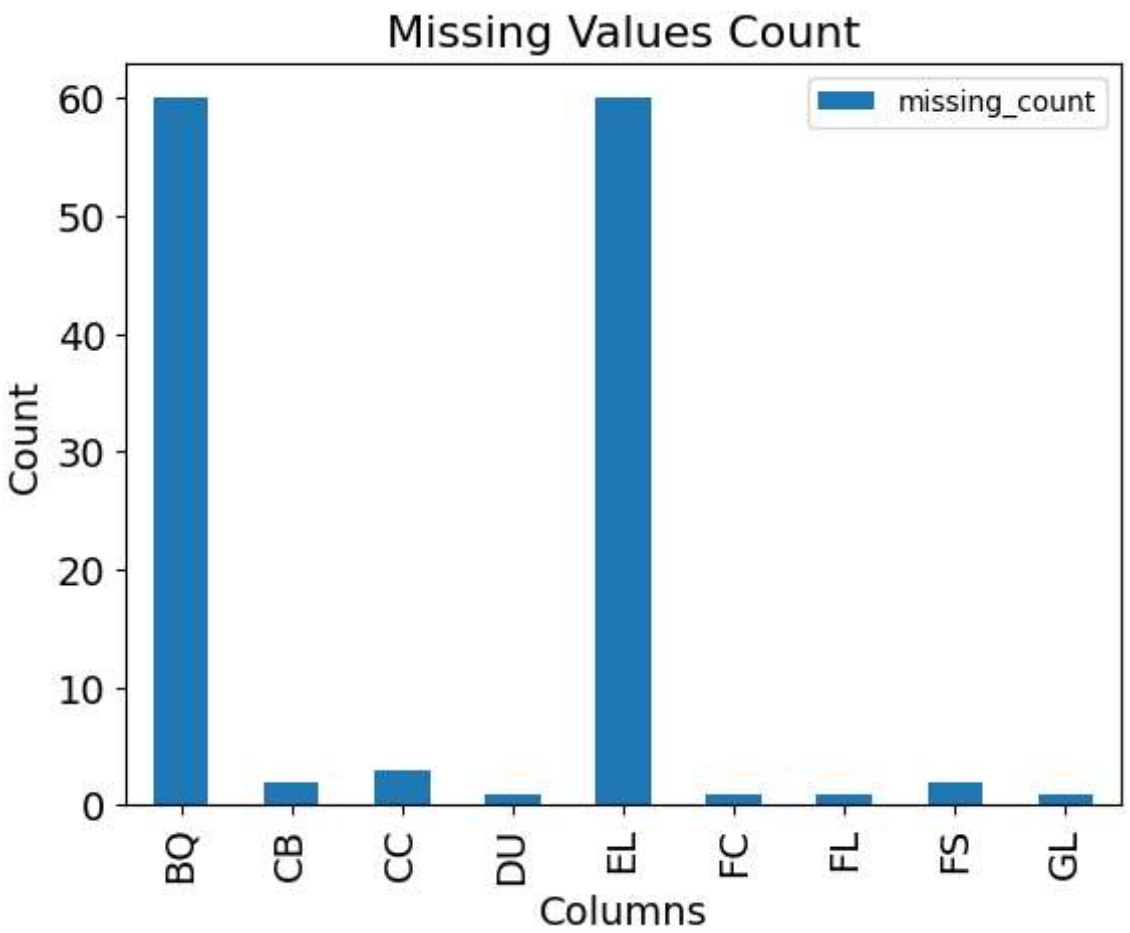
2.2. Missing Value Analysis

Missing values in machine learning can have a significant impact on the performance and accuracy of models, and most machine learning algorithms cannot handle missing value. To address missing values, imputation techniques are often used to estimate or fill in the missing data.

In [5]:
```
# Only include numerical features
df_train_numerical = df_train.drop(['Id', 'EJ', 'Class'], axis=1)
```

In [6]:
```
plt.figure(figsize=(10, 6))

# No. of missing values by features
df_train_missing = df_train_numerical.isna().sum()

# Resetting the index
df_train_missing = df_train_missing.reset_index()

# Renaming the columns
df_train_missing.columns = ['feature', 'missing_count']

# Filtering features with missing values
df_train_missing = df_train_missing.loc[df_train_missing['missing_count'] > 0]

# Create a bar chart
df_train_missing.plot.bar(x='feature', y='missing_count')

# Set the chart title and axis labels
plt.title('Missing Values Count', fontsize=16)
plt.xlabel('Columns', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.tick_params(axis='x', which='major', labelsize=14)
plt.tick_params(axis='y', which='major', labelsize=14)

# Display the chart
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



Observations: The features BQ and EL exhibit a high number of missing values, while some other features also have a few missing values. In order to tackle this issue, we will perform imputation to handle the missing values.

2.3. Descritpive Analysis

Descriptive analysis refers to the process of summarizing and interpreting data to gain insights and understand its main characteristics. It involves the use of various statistical measures and visualization techniques to describe and present data in a meaningful way.

In [7]:
```
# Exclude the target label Class and categorical feature EJ in the Describe analysis
# Since there are too many colums for Describe analysis, we need to transpose the results.

df_train_numerical.describe(include='all').transpose()
```

Out[7]:

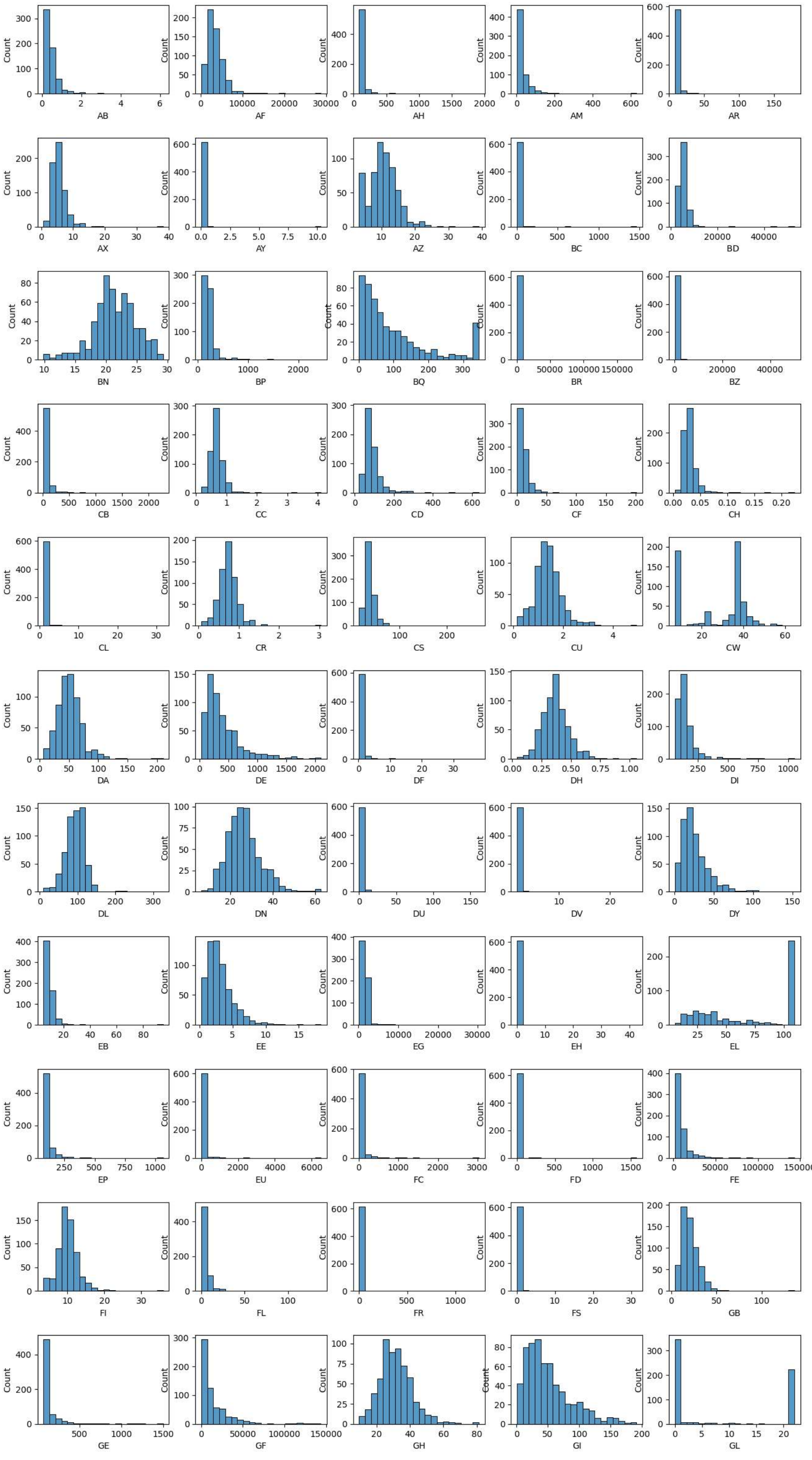
	count	mean	std	min	25%	50%	75%	max
AB	617.0	0.477149	0.468388	0.081187	0.252107	0.354659	0.559763	6.161666
AF	617.0	3502.013221	2300.322717	192.593280	2197.345480	3120.318960	4361.637390	28688.187660
AH	617.0	118.624513	127.838950	85.200147	85.200147	85.200147	113.739540	1910.123198
AM	617.0	38.968552	69.728226	3.177522	12.270314	20.533110	39.139886	630.518230
AR	617.0	10.128242	10.518877	8.138688	8.138688	8.138688	8.138688	178.943634
AX	617.0	5.545576	2.551696	0.699861	4.128294	5.031912	6.431634	38.270880
AY	617.0	0.060320	0.416817	0.025578	0.025578	0.025578	0.036845	10.315851
AZ	617.0	10.566447	4.350645	3.396778	8.129580	10.461320	12.969516	38.971568
BC	617.0	8.053012	65.166943	1.229900	1.229900	1.229900	5.081244	1463.693448
BD	617.0	5350.388655	3021.326641	1693.624320	4155.702870	4997.960730	6035.885700	53060.599240
BN	617.0	21.419492	3.478278	9.886800	19.420500	21.186000	23.657700	29.307300
BP	617.0	231.322223	183.992505	72.948951	156.847239	193.908816	247.803462	2447.810550
BQ	557.0	98.328737	96.479371	1.331155	27.834425	61.642115	134.009015	344.644105
BR	617.0	1218.133238	7575.293707	51.216883	424.990642	627.417402	975.649259	179250.252900
BZ	617.0	550.632525	2076.371275	257.432377	257.432377	257.432377	257.432377	50092.459300
CB	615.0	77.104151	159.049302	12.499760	23.317567	42.554330	77.310097	2271.436167
CC	614.0	0.688801	0.263994	0.176874	0.563688	0.658715	0.772206	4.103032
CD	617.0	90.251735	51.585130	23.387600	64.724192	79.819104	99.813520	633.534408
CF	617.0	11.241064	13.571133	0.510888	5.066306	9.123000	13.565901	200.967526
CH	617.0	0.030615	0.014808	0.003184	0.023482	0.027860	0.034427	0.224074
CL	617.0	1.403761	1.922210	1.050225	1.050225	1.050225	1.228445	31.688153
CR	617.0	0.742262	0.281195	0.069225	0.589575	0.730800	0.859350	3.039675
CS	617.0	36.917590	17.266347	13.784111	29.782467	34.835130	40.529401	267.942823
CU	617.0	1.383792	0.538717	0.137925	1.070298	1.351665	1.660617	4.951507
CW	617.0	27.165653	14.645993	7.030640	7.030640	36.019104	37.935832	64.521624
DA	617.0	51.128326	21.210888	6.906400	37.942520	49.180940	61.408760	210.330920
DE	617.0	401.901299	317.745623	35.998895	188.815690	307.509595	507.896200	2103.405190
DF	617.0	0.633884	1.912384	0.238680	0.238680	0.238680	0.238680	37.895013
DH	617.0	0.367002	0.112989	0.040995	0.295164	0.358023	0.426348	1.060404
DI	617.0	146.972099	86.084419	60.232470	102.703553	130.050630	165.836955	1049.168078
DL	617.0	94.795377	28.243187	10.345600	78.232240	96.264960	110.640680	326.236200
DN	617.0	26.370568	8.038825	6.339496	20.888264	25.248800	30.544224	62.808096
DU	616.0	1.802900	9.034721	0.005518	0.005518	0.251741	1.058690	161.355315
DV	617.0	1.924830	1.484555	1.743070	1.743070	1.743070	1.743070	25.192930
DY	617.0	26.388989	18.116679	0.804068	14.715792	21.642456	34.058344	152.355164
EB	617.0	9.072700	6.200281	4.926396	5.965392	8.149404	10.503048	94.958580
EE	617.0	3.064778	2.058344	0.286201	1.648679	2.616119	3.910070	18.324926
EG	617.0	1731.248215	1790.227476	185.594100	1111.160625	1493.817413	1905.701475	30243.758780
EH	617.0	0.305107	1.847499	0.003042	0.003042	0.085176	0.237276	42.569748
EL	557.0	69.582596	38.555707	5.394675	30.927468	71.949306	109.125159	109.125159
EP	617.0	105.060712	68.445620	78.526968	78.526968	78.526968	112.766654	1063.594578
EU	617.0	69.117005	390.187057	3.828384	4.324656	22.641144	49.085352	6501.264480
FC	616.0	71.341526	165.551545	7.534128	25.815384	36.394008	56.714448	3030.655824
FD	617.0	6.930086	64.754262	0.296850	0.296850	1.870155	4.880214	1578.654237
FE	617.0	10306.810737	11331.294051	1563.136688	5164.666260	7345.143424	10647.951650	143224.682300
FI	617.0	10.111079	2.934025	3.583450	8.523098	9.945452	11.516657	35.851039
FL	616.0	5.433199	11.496257	0.173229	0.173229	3.028141	6.238814	137.932739
FR	617.0	3.533905	50.181948	0.497060	0.497060	1.131000	1.512060	1244.227020
FS	615.0	0.421501	1.305365	0.067730	0.067730	0.250601	0.535067	31.365763
GB	617.0	20.724856	9.991907	4.102182	14.036718	18.771436	25.608406	135.781294
GE	617.0	131.714987	144.181524	72.611063	72.611063	72.611063	127.591671	1497.351958
GF	617.0	14679.595398	19352.959387	13.038894	2798.992584	7838.273610	19035.709240	143790.071200
GH	617.0	31.489716	9.864239	9.432735	25.034888	30.608946	36.863947	81.210825
GI	617.0	50.584437	36.266251	0.897628	23.011684	41.007968	67.931664	191.194764

	count	mean	std	min	25%	50%	75%	max
GL	616.0	8.530961	10.327010	0.001129	0.124392	0.337827	21.978000	21.978000

2.4. Histogram Analysis for Skewness

It examines the shape of the distribution displayed in a histogram to determine the skewness of the data.

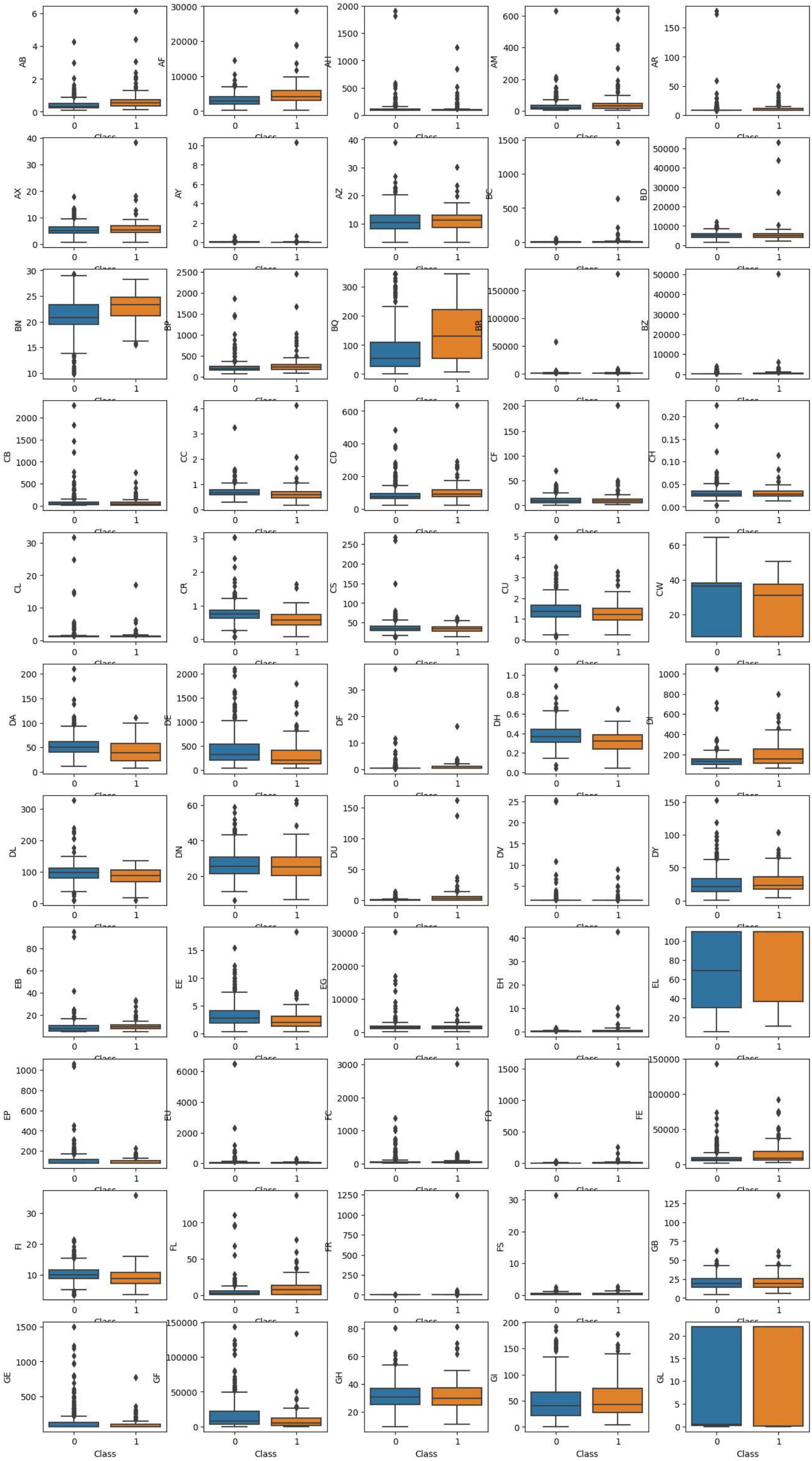
In [8]:
```
# Histgram for numercial features
fig, ax = plt.subplots(11, 5, figsize=(16,30))

for i in range(0, (len(ax.flatten()))):
#     print('{}, {}'.format(int(i/5),i % 5))
    sns.histplot(data=df_train_numerical, x =df_train_numerical.iloc[:,i], bins=20, ax=ax[int(i/5),i % 5])
#     ax[int(i/5), i % 5].set_title(df_train_numerical.columns[i])

# Adjust the vertical spacing between subplots
plt.subplots_adjust(hspace=0.5)

plt.show()
```


Count
AB
AF
AH
AM
AR
AX
AY
AZ
BC
BD
BN
BP
BQ
BR
BZ
CB
CC
CD
CF
CH
CL
CR
CS
CU
CW
DA
DE
DF
DH
DI
DL
DN
DU
DV
DY
EB
EE
EG
EH
EL
EP
EU
FC
FD
FE
FI
FL
FR
FS
GB
GE
GF
GH
GI
GL

2.5. Boxplot Analysis for Outliers

It identifies and analyzes outliers in a dataset. A boxplot is a graphical representation that displays the distribution of data and provides insights into the presence of outliers. Here's how you can perform boxplot analysis for outliers

In [9]:
```
# Histgram for numercial features
fig, ax = plt.subplots(11, 5, figsize=(16,30))

for i in range(0, (len(ax.flatten()))):
    sns.boxplot(x="Class",y=df_train_numerical.columns[i],data=df_train, ax=ax[int(i/5),i % 5])

# Adjust the vertical spacing between subplots
plt.subplots_adjust(wspace=0.3)

plt.show()
```


AB
AF
AH
AM
AR
AX
AY
AZ
BC
BD
BN
BP
BQ
BR
BZ
CB
CC
CD
CF
CH
CL
CR
CS
CU
CW
DA
DE
DF
DH
DI
DL
DN
DU
DV
DY
EB
EE
EG
EH
EL
EP
EU
FC
FD
FE
FI
FL
FR
FS
GB
GE
GF
GH
GI
GL
Class

Observations: based on the descriptive and histogram analysis, it is evident that the numerical features display different variances and distributions. To ensure consistency and comparability, it is crucial to standardize the features by rescaling them before proceeding with the modeling process.

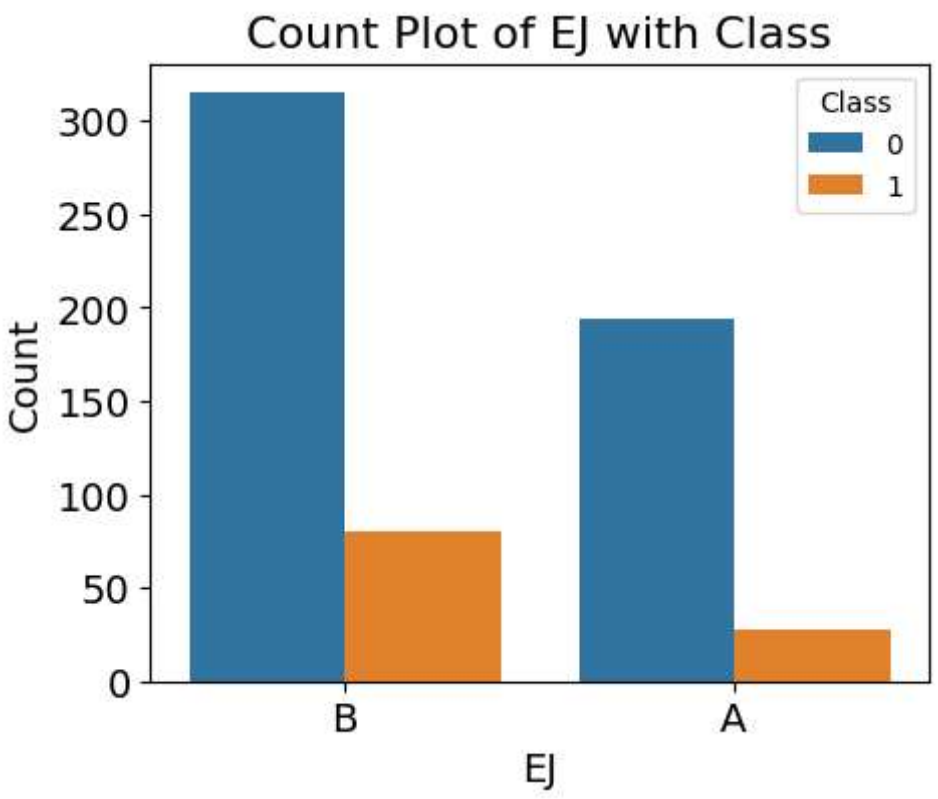
Furthermore, the box plot analysis reveals the presence of outliers in several numerical features. To address the skewness and outliers effectively, we propose utilizing the Robust Scaler for standardizing the numerical features.

The Robust Scaler is well-suited for this task as it can automatically handle outliers during the scaling process, providing robustness to extreme values. By employing the Robust Scaler, we can mitigate the impact of outliers and achieve a more reliable and accurate modeling outcome.

2.6. Count Plot for Distribution of Categorical Features

This plot is a type of visualization that displays the distribution of categorical features in a dataset. It represents the frequency or count of each category or class within a categorical variable.

In [10]:
```
# Set the size of the chart
plt.figure(figsize=(5, 4))
plt.legend(fontsize=13)

# Create the count plot
sns.countplot(data=df_train, x='EJ', hue='Class')

# Set the labels and title
plt.xlabel('EJ', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.title('Count Plot of EJ with Class', fontsize=16)

# Adjust the tick label size
plt.tick_params(axis='x', which='major', labelsize=14)
plt.tick_params(axis='y', which='major', labelsize=14)

# Add a legend
plt.legend(title='Class')

plt.show()
```



3. Data Pre-processing for Model Data

3.1. Missing Value Imputation - kNN Imputer

Missing value imputation using kNN (k-nearest neighbors) imputer is a technique that fills in missing values in a dataset by estimating them based on the values of their k nearest neighbors. It leverages the similarities between samples to impute missing values, making it a useful method for handling incomplete data.

In [11]:
```
# Initialize the KNNImputer with the desired number of neighbors
imputer = KNNImputer(n_neighbors=5)

# Perform KNN imputation
df_train_imputed = pd.DataFrame(imputer.fit_transform(df_train[df_train_numerical.columns]), columns=df_train_numerical
df_test_imputed =pd.DataFrame(imputer.transform(df_test[df_train_numerical.columns]), columns=df_train_numerical.column
```

In [12]:
```
# Check if there are still missing values in the train and test data sets
df_train_null = df_train_imputed[df_train_imputed.isnull().any(axis=1)]
df_test_null = df_test_imputed[df_test_imputed.isnull().any(axis=1)]

# Display the rows with null values
print('No. of records with missing value in Train data set after Imputation : {}'.format(df_train_null.shape[0]))
print('No. of records with missing value in Test data set after Imputation : {}'.format(df_test_null.shape[0]))

# Check the shape of the train and test data set
```

```
print('=' * 50)
print('Shape of the Train data set : {}'.format(df_train_imputed.shape))
print('Shape of the Test data set : {}'.format(df_test_imputed.shape))
```

```
No. of records with missing value in Train data set after Imputation : 0
No. of records with missing value in Test data set after Imputation : 0
==================================================
Shape of the Train data set : (617, 55)
Shape of the Test data set : (5, 55)
```

In [13]:
```
# Replace the imputed columns in the train data sets
df_train_2 = df_train.drop(df_train_numerical.columns, axis=1)
df_train_2 = pd.concat ([df_train_2, df_train_imputed], axis=1)

# Replace the imputed columns in the test data sets
df_test_2 = df_test.drop(df_train_numerical.columns, axis=1)
df_test_2 = pd.concat ([df_test_2, df_test_imputed], axis=1)

# Check the shape of the train and test data set
print('Shape of the Train data set : {}'.format(df_train_2.shape))
print('Shape of the Test data set : {}'.format(df_test_2.shape))
```

```
Shape of the Train data set : (617, 58)
Shape of the Test data set : (5, 57)
```

3.2. Data Standarization for Numercial Features

The RobustScaler is a data preprocessing technique available in the scikit-learn library in Python. It is used to scale numerical data in a robust manner, meaning it is less sensitive to the presence of outliers compared to other scaling methods like standardization or min-max scaling.

In [14]:
```
# Create a RobustScaler object
scaler = RobustScaler()

# Extract the index
index = df_train_2.index

# Fit the scaler to the data and transform it
scaler_train = scaler.fit_transform(df_train_2[df_train_numerical.columns])
scaler_df_train = pd.DataFrame(scaler_train, columns=df_train_numerical.columns)

# Reassign the index to the scaled DataFrame
scaler_df_train.index = index
print('Shape of Scaled Train Data Set: {}'.format(scaler_df_train.shape))


# Extract the index
index = df_test_2.index

scaler_test = scaler.transform(df_test_2[df_train_numerical.columns])
scaler_df_test = pd.DataFrame(scaler_test, columns=df_train_numerical.columns)

# Reassign the index to the scaled DataFrame
scaler_df_test.index = index
print('Shape of Scaled Test Data Set: {}'.format(scaler_df_test.shape))
```

```
Shape of Scaled Train Data Set: (617, 55)
Shape of Scaled Test Data Set: (5, 55)
```

In [15]:
```
# Replace the scaled columns in the train data sets
df_train_2 = df_train_2.drop(df_train_numerical.columns, axis=1)
df_train_2 = pd.concat ([df_train_2, scaler_df_train], axis=1)

# Replace the imputed columns in the test data sets
df_test_2 = df_test_2.drop(df_train_numerical.columns, axis=1)
df_test_2 = pd.concat ([df_test_2, scaler_df_test], axis=1)

# Check the shape of the train and test data set
print('Shape of the Train data set : {}'.format(df_train_2.shape))
print('Shape of the Test data set : {}'.format(df_test_2.shape))
```

```
Shape of the Train data set : (617, 58)
Shape of the Test data set : (5, 57)
```

3.3. Data Encoding for Categorical Features

CATBoostEncode, or CatBoostEncoder, is a categorical encoding technique specifically designed for the CatBoost algorithm. CatBoost is a gradient boosting algorithm that is known for its strong performance in handling categorical features. The CatBoostEncoder is a specialized encoding method that is compatible with the CatBoost algorithm and aims to effectively encode categorical variables for improved model performance.

In [16]:
```
# Load the CatBoost Encoder
CATBoostENCODE = encoders.CatBoostEncoder()

categorical_cols = ['EJ']

# Use CatBoost to encode the categorical values
encoder_train = CATBoostENCODE.fit_transform(df_train[categorical_cols], df_train['Class'])
encoded_df_train = pd.DataFrame(encoder_train)
print('Shape of the Encoded Train Data Set: {}'.format(encoded_df_train.shape))

encoder_test = CATBoostENCODE.transform(df_test[categorical_cols])
```

```
encoded_df_test = pd.DataFrame(encoder_test)
print('Shape of the Encoded Test Data Set: {}'.format(encoded_df_test.shape))
```

```
Shape of the Encoded Train Data Set: (617, 1)
Shape of the Encoded Test Data Set: (5, 1)
```

In [17]:
```
# Replace the encoded columns in the train data sets
df_train_2 = df_train_2.drop(categorical_cols, axis=1)
df_train_2 = pd.concat ([df_train_2, encoded_df_train], axis=1)

# Replace the imputed columns in the test data sets
df_test_2 = df_test_2.drop(categorical_cols, axis=1)
df_test_2 = pd.concat ([df_test_2, encoded_df_test], axis=1)

# Check the shape of the train and test data set
print('Shape of the Train data set : {}'.format(df_train_2.shape))
print('Shape of the Test data set : {}'.format(df_test_2.shape))
```

```
Shape of the Train data set : (617, 58)
Shape of the Test data set : (5, 57)
```

3.4. SelectKBest Method from SKLearn for Feature Selection

SelectKBest is a feature selection method in Scikit-learn (sklearn) library, which selects the top k features (columns) from a dataset based on some statistical test score.The basic idea behind SelectKBest is to evaluate the statistical significance of each feature using a scoring function and select the top k features with the highest scores.

In [18]:
```
# Before we do the oversampling, we will split the data into training and testing data for model training
train = df_train_2.drop(['Id', 'Class'], axis=1)
test = df_train_2['Class']

# Renaming the columns
test.columns = ['Class']
```

In [19]:
```
threshold = 0.1
t_score = 5

# Initiate the SelectKBest function
# For regression tasks: f_regression, mutual_info_regression
# For classification tasks: chi2, f_classif, mutual_info_classif
fs = SelectKBest(score_func=f_classif, k=len(train.columns))

# apply feature selection
X_selected = fs.fit_transform(train, test.values)
print('Befoe the SelectKBest = {}'.format(train.shape))

new_features = [] # The list of features less than the p-values
drop_features = [] # The list of features higher than the p-values

for i in range(len(train.columns)):
    print('Feature {}: {:.3f} with p-value {:.3f}'.format(train.columns[i], fs.scores_[i], fs.pvalues_[i]))
    if fs.pvalues_[i] <= threshold and fs.scores_[i] >= t_score:
        new_features.append(train.columns[i])
    else:
        drop_features.append(train.columns[i])

X_selected_final =  pd.DataFrame(X_selected)
X_selected_final.columns = train.columns
#    print(X_selected_final.shape)
X_selected_final = X_selected_final[new_features]
#    print(X_selected_final.shape)

print('=' * 30)
print('After the SelectKBest = {}'.format(X_selected_final.shape))
print('Drop-out Features = {}'.format(len(drop_features)))
```

```
Befoe the SelectKBest = (617, 56)
Feature AB: 52.566 with p-value 0.000
Feature AF: 62.007 with p-value 0.000
Feature AH: 1.228 with p-value 0.268
Feature AM: 37.314 with p-value 0.000
Feature AR: 2.560 with p-value 0.110
Feature AX: 5.917 with p-value 0.015
Feature AY: 4.206 with p-value 0.041
Feature AZ: 0.112 with p-value 0.738
Feature BC: 15.316 with p-value 0.000
Feature BD : 7.839 with p-value 0.005
Feature BN: 25.169 with p-value 0.000
Feature BP: 15.600 with p-value 0.000
Feature BQ: 50.639 with p-value 0.000
Feature BR: 4.861 with p-value 0.028
Feature BZ: 7.872 with p-value 0.005
Feature CB: 0.131 with p-value 0.717
Feature CC: 1.517 with p-value 0.219
Feature CD : 18.593 with p-value 0.000
Feature CF: 7.498 with p-value 0.006
Feature CH: 0.041 with p-value 0.840
Feature CL: 0.175 with p-value 0.676
Feature CR: 33.582 with p-value 0.000
Feature CS: 1.387 with p-value 0.239
Feature CU: 4.285 with p-value 0.039
Feature CW : 2.423 with p-value 0.120
Feature DA: 26.873 with p-value 0.000
Feature DE: 9.758 with p-value 0.002
Feature DF: 2.551 with p-value 0.111
Feature DH: 27.487 with p-value 0.000
Feature DI: 44.868 with p-value 0.000
Feature DL: 13.719 with p-value 0.000
Feature DN: 0.044 with p-value 0.834
Feature DU: 44.959 with p-value 0.000
Feature DV: 0.147 with p-value 0.701
Feature DY: 2.430 with p-value 0.120
Feature EB: 4.800 with p-value 0.029
Feature EE: 11.472 with p-value 0.001
Feature EG: 0.373 with p-value 0.542
Feature EH: 20.559 with p-value 0.000
Feature EL: 3.016 with p-value 0.083
Feature EP: 2.889 with p-value 0.090
Feature EU: 0.973 with p-value 0.324
Feature FC: 0.596 with p-value 0.440
Feature FD : 10.755 with p-value 0.001
Feature FE: 30.203 with p-value 0.000
Feature FI: 5.521 with p-value 0.019
Feature FL: 38.983 with p-value 0.000
Feature FR: 6.738 with p-value 0.010
Feature FS: 0.003 with p-value 0.957
Feature GB: 4.302 with p-value 0.038
Feature GE: 3.095 with p-value 0.079
Feature GF: 10.278 with p-value 0.001
Feature GH: 0.693 with p-value 0.406
Feature GI: 3.658 with p-value 0.056
Feature GL: 8.809 with p-value 0.003
Feature EJ: 0.799 with p-value 0.372
==============================
After the SelectKBest = (617, 28)
Drop-out Features = 28
```

In [20]:
```python
# Drop out low informative features for model training
train = train.drop(drop_features, axis=1)
df_train_2  = df_train_2.drop(drop_features, axis=1)
df_test_2  = df_test_2.drop(drop_features, axis=1)
```

3.5. Oversampling with SMOTE (Synthetic Minority Over-sampling Technique)

SMOTE (Synthetic Minority Over-sampling Technique) is a technique to address the problem of imbalanced datasets. Imbalanced datasets occur when the classes in the target variable are not represented equally, resulting in a skewed distribution. This is a common issue in many real-world classification problems, such as fraud detection, rare disease prediction, or anomaly detection.

In [21]:
```python
X_train, X_test, y_train, y_test = train_test_split(train, test, test_size=0.3)

print('Shape of train : {}'.format(X_train.shape))
print('Shape of test : {}'.format(X_test.shape))
print('='*50)
print('Shape of df_train (incl. ID and Class): {}'.format(df_train_2.shape))
print('Shape of df_test (incl. ID): {}'.format(df_test_2.shape))
```

```
Shape of train : (431, 28)
Shape of test : (186, 28)
==================================================
Shape of df_train (incl. ID and Class): (617, 30)
Shape of df_test (incl. ID): (5, 29)
```

In [22]:
```python
# Load the SMOTE library
smote = SMOTE(sampling_strategy={0: 360, 1: 360})
# df_train_numerical = df_train.drop(['Id', 'EJ', 'Class'], axis=1)

X_smote, y_smote = smote.fit_resample(X_train, y_train)
print("length of original data is ",len(df_train_2))
print("Proportion of True data in original data is {:.2%}".format(len(y_train[y_train==1])/len(y_train)))
```

```
print("Proportion of False data in original data is {:.2%}".format(len(y_train[y_train==0])/len(y_train)))
print("length of oversampled data is ",len(X_smote))
print("Proportion of True data in oversampled data is {:.2%}".format(len(y_smote[y_smote ==1])/len(y_smote)))
print("Proportion of False data in oversampled data is {:.2%}".format(len(y_smote[y_smote ==0])/len(y_smote)))
```

```
length of original data is  617
Proportion of True data in original data is 16.94%
Proportion of False data in original data is 83.06%
length of oversampled data is  720
Proportion of True data in oversampled data is 50.00%
Proportion of False data in oversampled data is 50.00%
```

4. Modelling

In [23]:

```
# Parameter Setup
p_folds = 3
p_iter = 1000
p_estimators = 1000
p_learning_rate = 0.01
```

4.1. XGBoost

In [24]:

```
# A parameter grid for XGBoost
params = {
        'max_depth': randint(5, 10),
        'gamma': uniform(0.0, 0.5),
        'subsample': uniform(0.6, 1.0),
        'colsample_bytree': uniform(0.6, 1.0),
        'reg_alpha': uniform(0.0, 1.0),
        'reg_lambda': uniform(0.0, 1.0),
        'min_child_weight': randint(3, 7),
        'scale_pos_weight': randint(1, 10)
        }
```

In [25]:

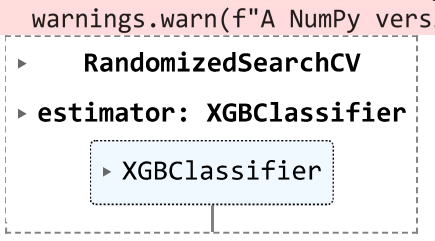
```
xgb = XGBClassifier(learning_rate=p_learning_rate, n_estimators = p_estimators, objective='binary:logistic')
```

In [26]:

```
folds = p_folds

skf = StratifiedKFold(n_splits=folds, shuffle = True, random_state = 1001)

xg_model = RandomizedSearchCV(xgb, param_distributions=params, n_iter=p_iter, scoring='roc_auc', n_jobs=-1, cv=skf.spli

xg_model.fit(X_smote, y_smote)
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is req
uired for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is req
uired for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is req
uired for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is req
uired for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Out[26]:



In [27]:

```
print(xg_model.best_estimator_)
print(xg_model.best_params_)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.7585347694635894, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0.1969518054709632, gpu_id=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.01, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=9,
              max_leaves=None, min_child_weight=3, missing=nan,
              monotone_constraints=None, n_estimators=1000, n_jobs=None,
              num_parallel_tree=None, predictor=None, random_state=None, ...)
{'colsample_bytree': 0.7585347694635894, 'gamma': 0.1969518054709632, 'max_depth': 9, 'min_child_weight': 3, 'reg_alph
a': 0.09263631482059975, 'reg_lambda': 0.03907966546393071, 'scale_pos_weight': 9, 'subsample': 0.8887504086408556}
```

In [28]:

```
# Retrieve the best estimator and build the optimal model for analysis of Global Importance
best_xgb =XGBClassifier(**xg_model.best_estimator_.get_params())
best_xgb.fit(X_smote,y_smote)
accuracy = best_xgb.score(X_test, y_test)
print('Accuracy of XGBoost : {}'.format(accuracy))
```

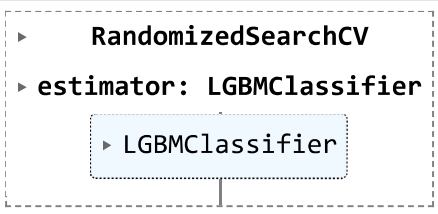
```
Accuracy of XGBoost : 0.8870967741935484
```

4.2. LGBM

In [29]:
```
params = {
#     'learning_rate': [0.01, 0.05, 0.1],
#     'n_estimators': [50, 100, 500, 1000],
    'num_leaves': randint(5, 50),
    'max_depth': randint(5, 10)
}

lgbm = LGBMClassifier(random_state=42, n_estimators=p_estimators, learning_rate = p_learning_rate)
```

In [30]:
```
folds = p_folds

skf = StratifiedKFold(n_splits=folds, shuffle = True, random_state = 1001)

lgbm_model = RandomizedSearchCV(lgbm, param_distributions=params, n_iter=p_iter, scoring='roc_auc', n_jobs=-1, cv=skf.s

lgbm_model.fit(X_smote, y_smote)
```

Out[30]:



In [31]:
```
print(lgbm_model.best_estimator_)
print(lgbm_model.best_params_)
```

```
LGBMClassifier(learning_rate=0.01, max_depth=8, n_estimators=1000,
               num_leaves=14, random_state=42)
{'max_depth': 8, 'num_leaves': 14}
```

In [32]:
```
# Retrieve the best estimator and build the optimal model for analysis of Global Importance

best_lgbm =LGBMClassifier(**lgbm_model.best_estimator_.get_params())
best_lgbm.fit(X_smote,y_smote)
accuracy = best_lgbm.score(X_test, y_test)
print('Accuracy of LGBM : {}'.format(accuracy))
```

```
Accuracy of LGBM : 0.8978494623655914
```

4.3. Logistic Regression

In [33]:
```
# Specify the hyperparameter grid

params = {
    'C': [0.1, 1, 10],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'saga'],
    'max_iter': [100, 200, 300, 400, 500],
    'tol': [0.1, 0.001, 0.01]
}

# Define the logistic regression model
logreg = LogisticRegression()
```

In [34]:
```
folds = p_folds

skf = StratifiedKFold(n_splits=folds, shuffle = True, random_state = 1001)

logreg_model = RandomizedSearchCV(logreg, param_distributions=params, n_iter=p_iter, scoring='roc_auc', n_jobs=-1, cv=s

logreg_model.fit(X_smote, y_smote)
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
```

```
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
```

```
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, in
crease the number of iterations.
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
```

Out[34]:

```
▸ RandomizedSearchCV
▸ estimator: LogisticRegression
    ▸ LogisticRegression
```

In [35]:

```
print(logreg_model.best_estimator_)
print(logreg_model.best_params_)
```

```
LogisticRegression(C=1, max_iter=200, penalty='l1', solver='liblinear',
                   tol=0.01)
{'tol': 0.01, 'solver': 'liblinear', 'penalty': 'l1', 'max_iter': 200, 'C': 1}
```

In [36]:
```
# Retrieve the best estimator and build the optimal model for analysis of Global Importance

best_logreg =LogisticRegression(**logreg_model.best_estimator_.get_params())
best_logreg.fit(X_smote,y_smote)
accuracy = best_logreg.score(X_test, y_test)

print('Accuracy of Logistic Regression : {}'.format(accuracy))
```

```
Accuracy of Logistic Regression : 0.8655913978494624
```

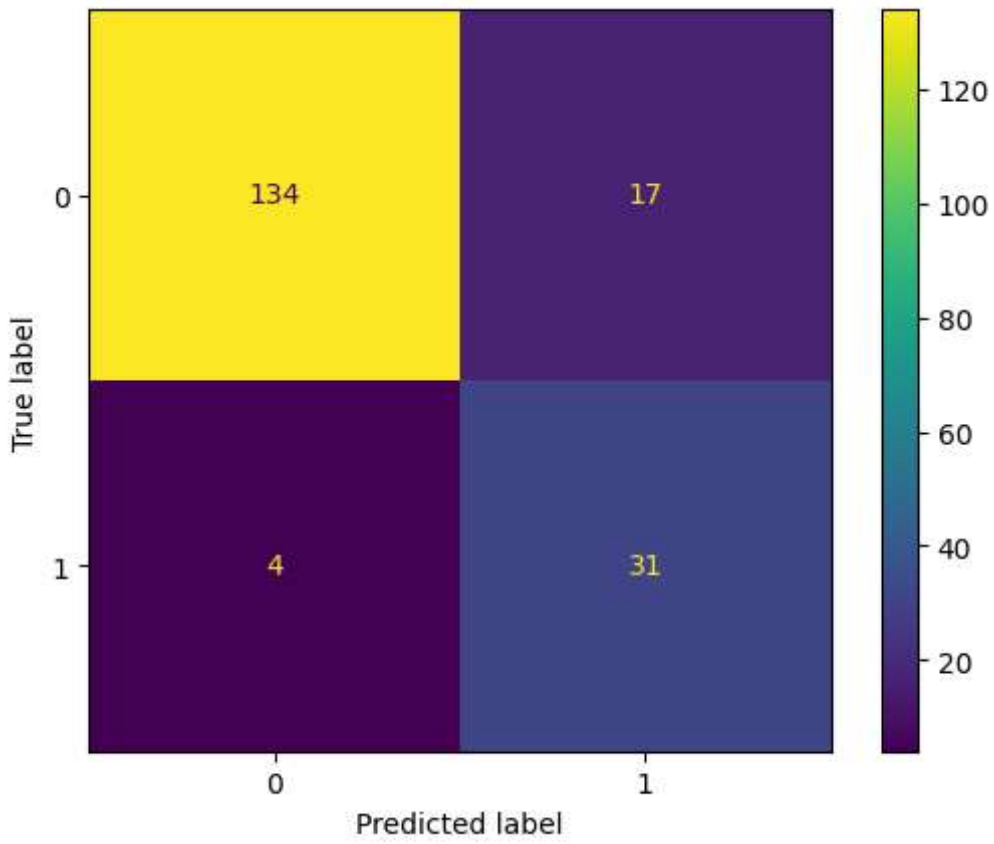
4.4. Model Performance

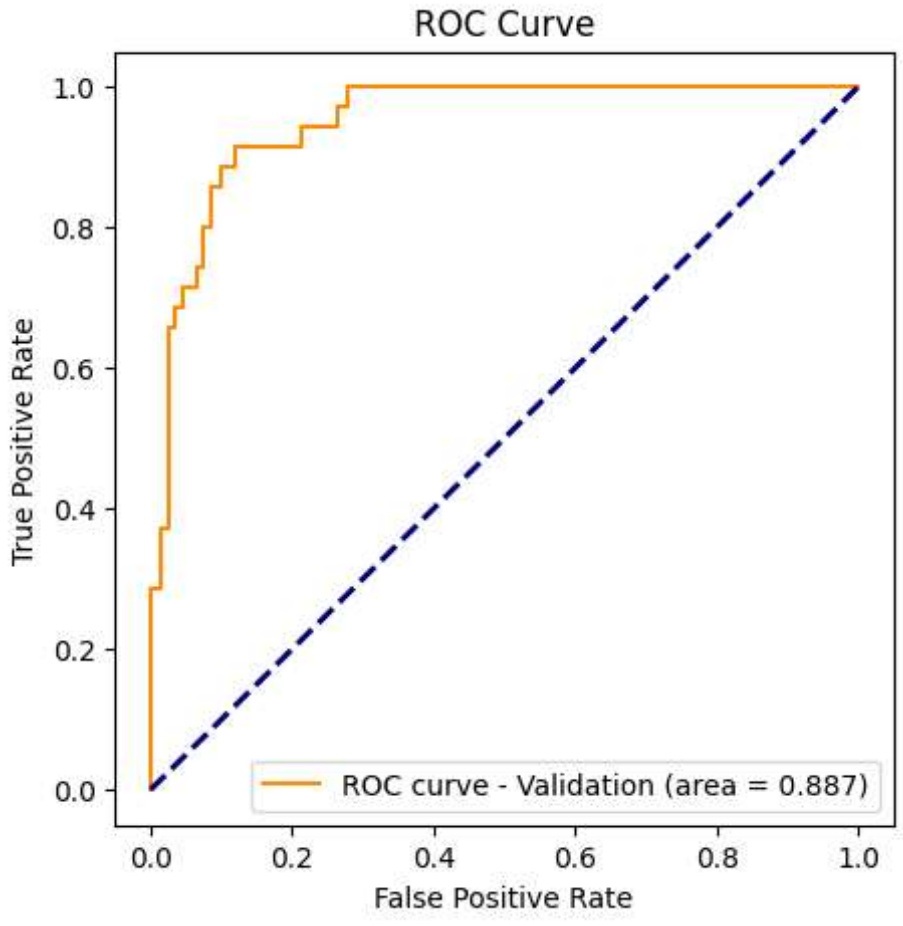
In [37]:
```
def model_performance (p_test, p_test_prob, Y_test, model_name):
    predicted_test = pd.DataFrame(p_test)
    print('===========================================')
    print('Scoring Metrics for {} (Validation)'.format(model_name))
    print('===========================================')
    print('Balanced Accuracy Score = {:2.3f}'.format(metrics.balanced_accuracy_score(Y_test, predicted_test)))
    print('Accuracy Score = {:2.3f}'.format(metrics.accuracy_score(Y_test, predicted_test)))
    print('Precision Score = {:2.3f}'.format(metrics.precision_score(Y_test, predicted_test)))
    print('F1 Score = {:2.3f}'.format(metrics.f1_score(Y_test, predicted_test, labels=['0','1'])))
    print('Recall Score = {:2.3f}'.format(metrics.recall_score(Y_test, predicted_test, labels=['0','1'])))
    print('ROC AUC Score = {:2.3f}'.format(metrics.roc_auc_score(Y_test, predicted_test, labels=['0','1'])))
    print('Confusion Matrix')
    print('==================')
    print(metrics.confusion_matrix(Y_test, predicted_test))
    print('==================')
    print(metrics.classification_report(Y_test, predicted_test, target_names=['0','1']))
    metrics.ConfusionMatrixDisplay(metrics.confusion_matrix(Y_test, predicted_test)).plot()

    # ROC Curve
    fpr_test, tpr_test, _ = metrics.roc_curve(Y_test, p_test_prob)

    roc_auc_test = metrics.roc_auc_score(Y_test, predicted_test, labels=['0','1'])

    # Precision x Recall Curve
    precision_test, recall_test, thresholds_test = metrics.precision_recall_curve(Y_test, p_test_prob)

    print('======= ROC Curve =======')
    fig, ax = plt.subplots(1, 1, figsize=(5, 5))
    plt.plot(fpr_test, tpr_test, color='darkorange', label='ROC curve - Validation (area = %0.3f)' % roc_auc_test)
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    # plt.xlim([0.0, 1.0])
    # plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend(loc="lower right")

    plt.show()
```

In [38]:
```
# Prediction for XGBoost and LGBM

xg_pred_test = xg_model.predict(X_test)
xg_proba_test = xg_model.predict_proba(X_test)[:,1]

lgbm_pred_test = lgbm_model.predict(X_test)
lgbm_proba_test = lgbm_model.predict_proba(X_test)[:,1]

logreg_pred_test = logreg_model.predict(X_test)
logreg_proba_test = logreg_model.predict_proba(X_test)[:,1]
```

In [39]:
```
model_performance(xg_pred_test, xg_proba_test, y_test, 'XGBoost')

# Use the roc_auc measure as a weighting to the meta classification decision
xg_roc_auc = metrics.roc_auc_score(y_test, xg_pred_test, labels=['0','1'])
```

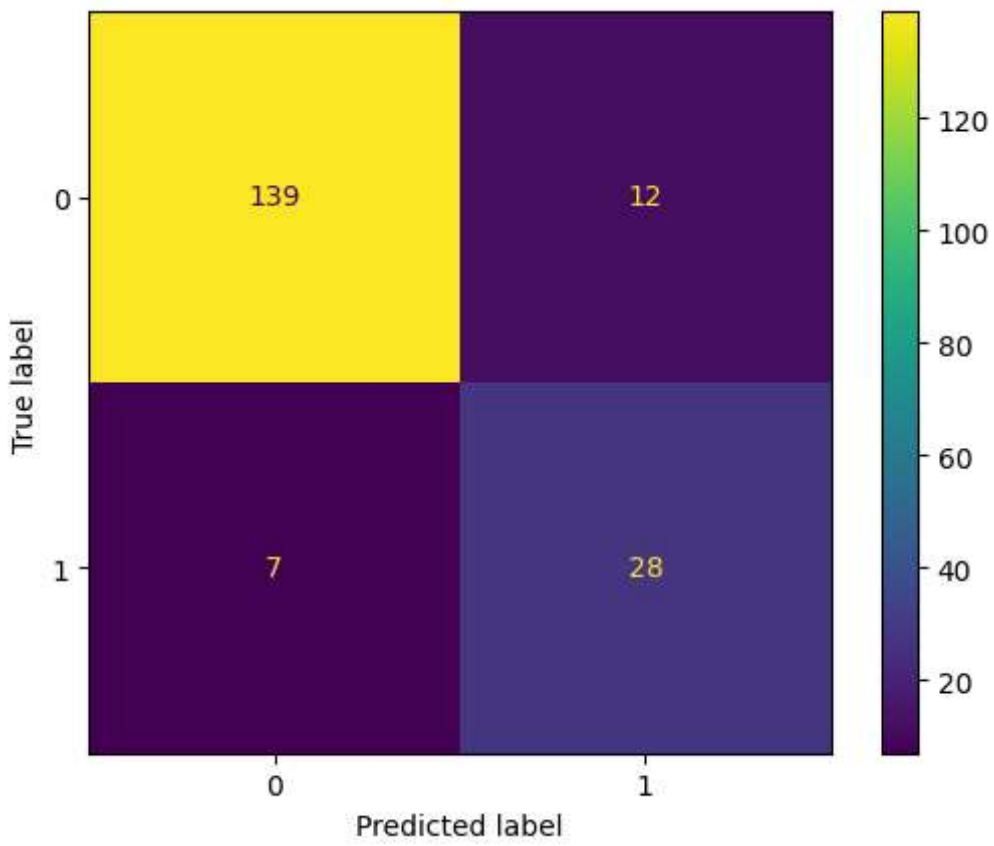
```
================================================
Scoring Metrics for XGBoost (Validation)
================================================
Balanced Accuracy Score = 0.887
Accuracy Score = 0.887
Precision Score = 0.646
F1 Score = 0.747
Recall Score = 0.886
ROC AUC Score = 0.887
Confusion Matrix
==================
[[134  17]
 [  4  31]]
==================
              precision    recall  f1-score   support

           0       0.97      0.89      0.93       151
           1       0.65      0.89      0.75        35

    accuracy                           0.89       186
   macro avg       0.81      0.89      0.84       186
weighted avg       0.91      0.89      0.89       186

======= ROC Curve =======
```

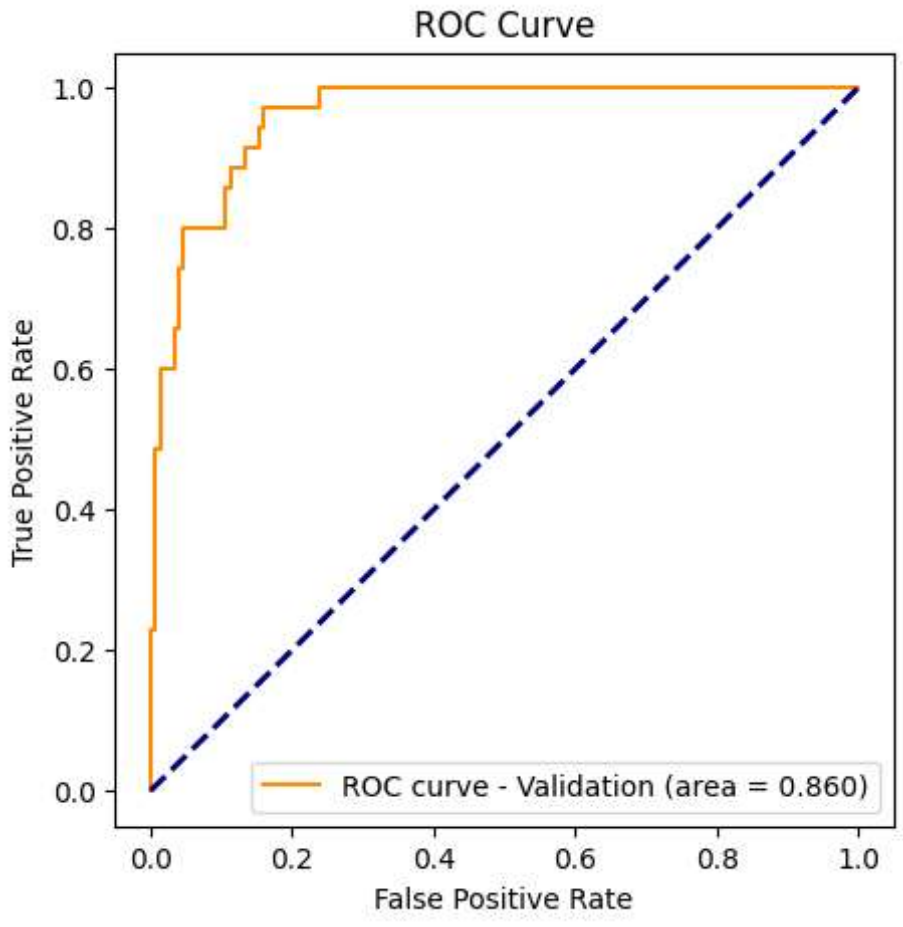




In [40]:
```
model_performance(lgbm_pred_test, lgbm_proba_test, y_test, 'LGBM')

# Use the roc_auc measure as a weighting to the meta classification decision
lgbm_roc_auc = metrics.roc_auc_score(y_test, lgbm_pred_test, labels=['0','1'])
```

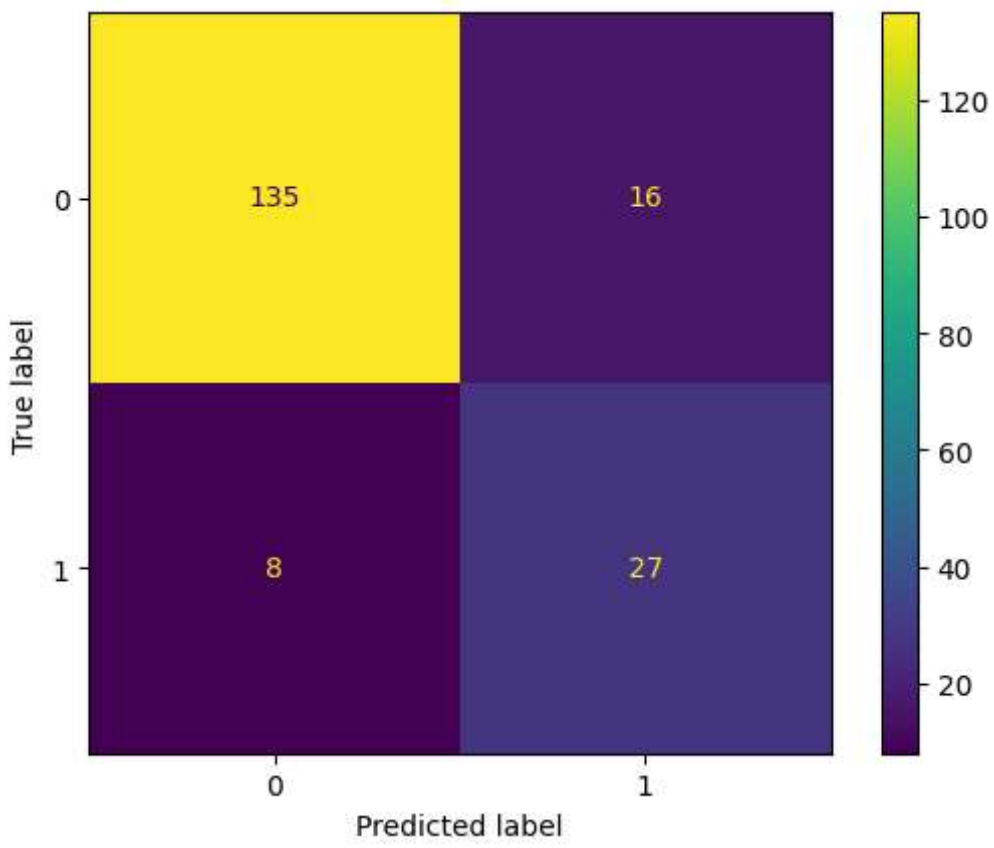
```
==============================================
Scoring Metrics for LGBM (Validation)
==============================================
Balanced Accuracy Score = 0.860
Accuracy Score = 0.898
Precision Score = 0.700
F1 Score = 0.747
Recall Score = 0.800
ROC AUC Score = 0.860
Confusion Matrix
==================
[[139  12]
 [  7  28]]
==================
              precision    recall  f1-score   support

           0       0.95      0.92      0.94       151
           1       0.70      0.80      0.75        35

    accuracy                           0.90       186
   macro avg       0.83      0.86      0.84       186
weighted avg       0.90      0.90      0.90       186

======= ROC Curve =======
```

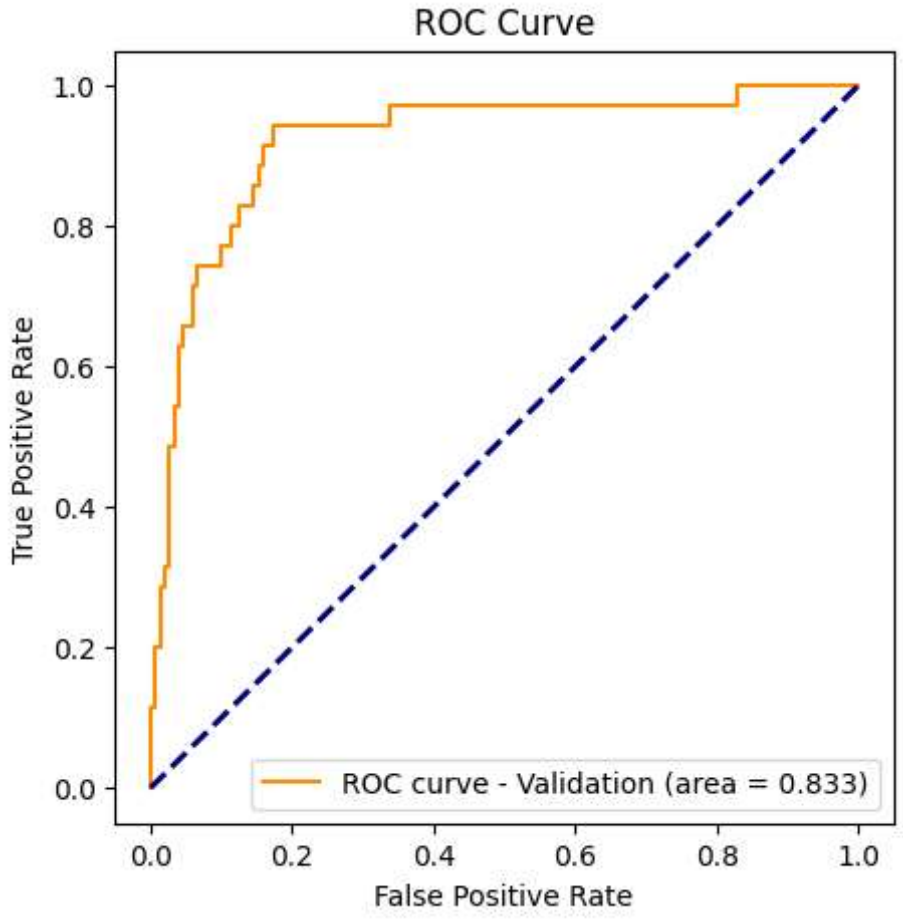




In [41]:
```
model_performance(logreg_pred_test, logreg_proba_test, y_test, 'Logistic Regression')

# Use the roc_auc measure as a weighting to the meta classification decision
logreg_roc_auc = metrics.roc_auc_score(y_test, logreg_pred_test, labels=['0','1'])
```

```
================================================
Scoring Metrics for Logistic Regression (Validation)
================================================
Balanced Accuracy Score = 0.833
Accuracy Score = 0.871
Precision Score = 0.628
F1 Score = 0.692
Recall Score = 0.771
ROC AUC Score = 0.833
Confusion Matrix
==================
[[135  16]
 [  8  27]]
==================
              precision    recall  f1-score   support

           0       0.94      0.89      0.92       151
           1       0.63      0.77      0.69        35

    accuracy                           0.87       186
   macro avg       0.79      0.83      0.81       186
weighted avg       0.88      0.87      0.88       186

======= ROC Curve =======
```



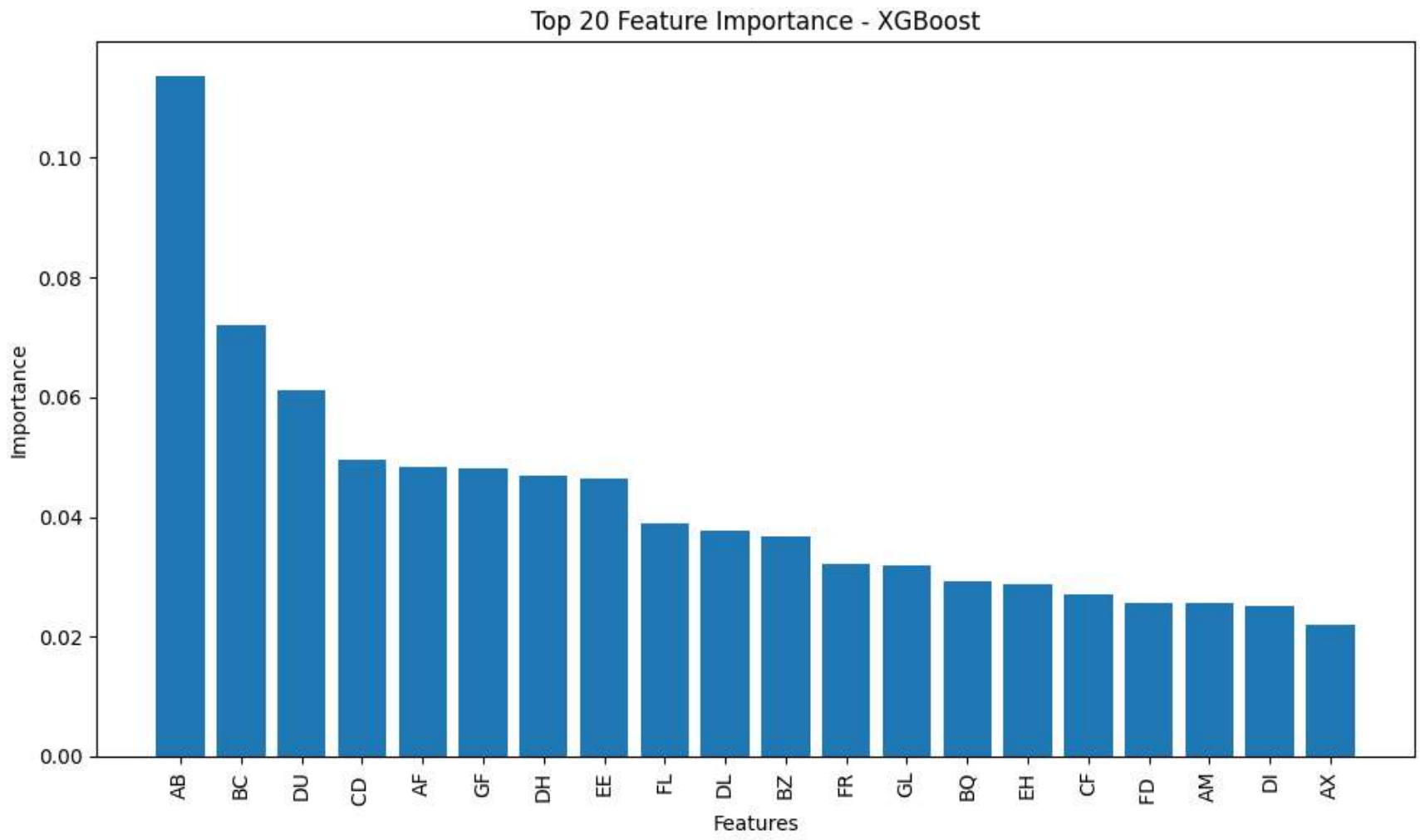


4.5 Model Explanation

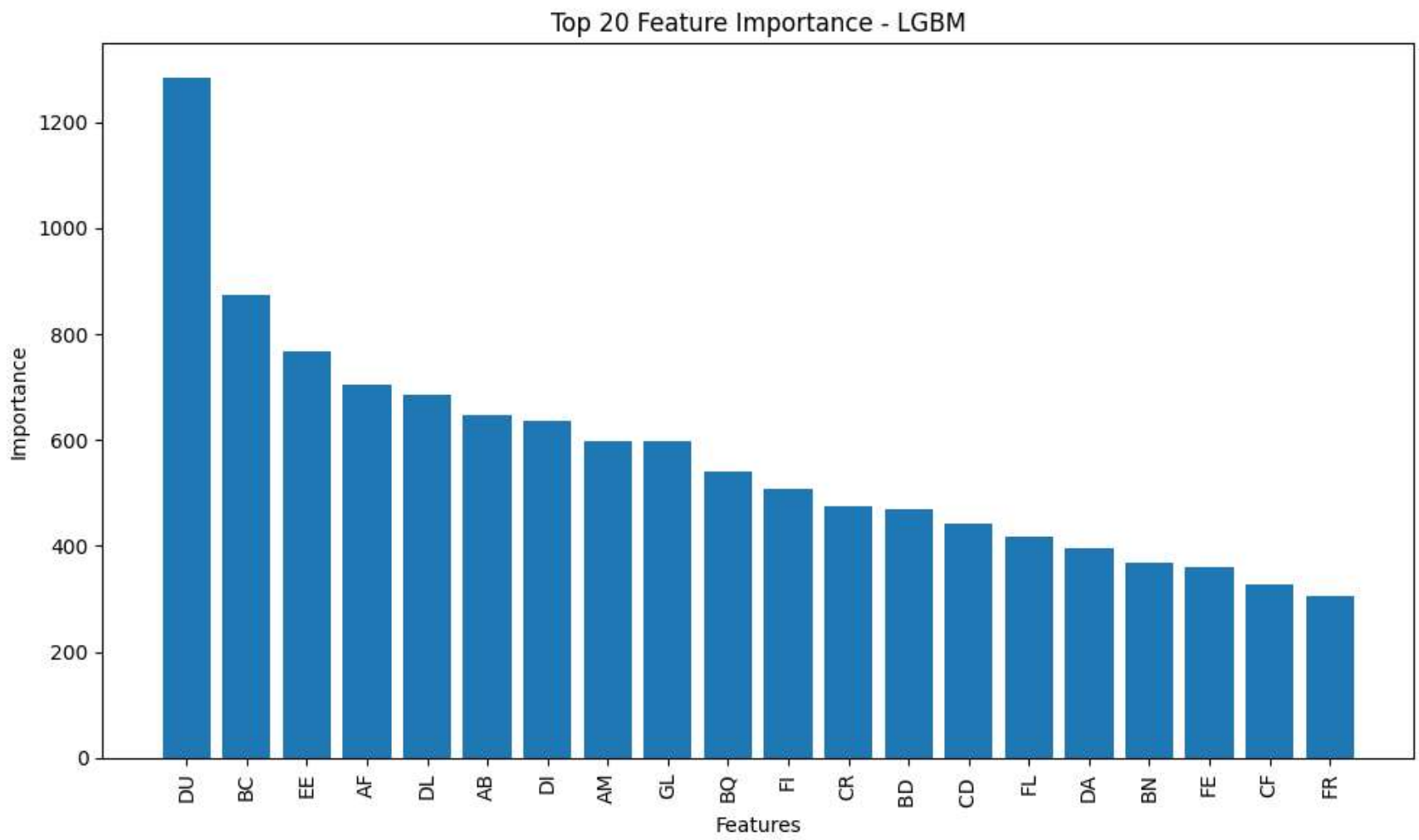
In [42]:
```python
def model_explanation (model_name, name):

    # Explain the Global Importance
    importances = model_name.feature_importances_

    # Get the indices of the top 20 features.
    top_indices = importances.argsort()[::-1][:20]

    # Get the names of the top 20 features.
    top_feature_names = X_smote.columns[top_indices]
```

```
    # Get the importances of the top 20 features.
    top_importances = importances[top_indices]

    # Plot the feature importances for the top 20 features.
    plt.figure(figsize = (10, 6))
    plt.bar(range(len(top_feature_names)), top_importances)
    plt.xticks(range(len(top_feature_names)), top_feature_names, rotation = 90)
    plt.xlabel('Features')
    plt.ylabel('Importance')
    plt.title('Top 20 Feature Importance - ' + name)
    plt.tight_layout()
    plt.show()
```

In [43]:
```
# Global Importance for XGBoost
model_explanation(best_xgb, 'XGBoost')
```



In [44]:
```
# Global Importance for LGBM
model_explanation(best_lgbm, 'LGBM')
```



4.6. Prediction and Submission

In [45]:
```
# Use the roc_auc as weighting in the meta classification

# prediction probability of XGBoost
xg_pred = xg_model.predict_proba(df_test_2.drop(['Id'], axis=1))
print('========== XGBoost ==========')
print(xg_pred)

# prediction probability of LGBM
```

```
lgbm_pred = lgbm_model.predict_proba(df_test_2.drop(['Id'], axis=1))
print('========== LGBM ==========')
print(lgbm_pred)

# prediction probability of Logistic Regression
logreg_pred = logreg_model.predict_proba(df_test_2.drop(['Id'], axis=1))
print('========== Logistic Regression ==========')
print(logreg_pred)
```

```
========== XGBoost ==========
[[0.9456958  0.05430419]
 [0.9456958  0.05430419]
 [0.9456958  0.05430419]
 [0.9456958  0.05430419]
 [0.9456958  0.05430419]]
========== LGBM ==========
[[0.99809874 0.00190126]
 [0.99809874 0.00190126]
 [0.99809874 0.00190126]
 [0.99809874 0.00190126]
 [0.99809874 0.00190126]]
========== Logistic Regression ==========
[[0.11911336 0.88088664]
 [0.11911336 0.88088664]
 [0.11911336 0.88088664]
 [0.11911336 0.88088664]
 [0.11911336 0.88088664]]
```

In [46]:
```
# weight of XGBoost
xg_weight = xg_roc_auc/(lgbm_roc_auc + xg_roc_auc + logreg_roc_auc)
lgbm_weight = lgbm_roc_auc/(lgbm_roc_auc + xg_roc_auc + logreg_roc_auc)
logreg_weight = logreg_roc_auc/(lgbm_roc_auc + xg_roc_auc + logreg_roc_auc)

print('Weight of XGBoost : {}, Weight of LGBM : {}, Weight of Reg : {}'.format(xg_weight, lgbm_weight, logreg_weight))
```

```
Weight of XGBoost : 0.3436881097337343, Weight of LGBM : 0.33349226142448474, Weight of Reg : 0.322819628841781
```

In [47]:
```
submission = df_test_2[['Id']].copy()
submission['Class_0'] = (xg_pred[:,0] * xg_weight) + (lgbm_pred[:,0] * lgbm_weight) + (logreg_pred[:,0] * logreg_weight
submission['Class_1'] = (xg_pred[:,1] * xg_weight) + (lgbm_pred[:,1] * lgbm_weight) + (logreg_pred[:,1] * logreg_weight
submission.to_csv('submission.csv',index=False)
submission.head()
```

Out[47]:

	Id	Class_0	Class_1
0	00eed32682bb	0.696335	0.303665
1	010ebe33f668	0.696335	0.303665
2	02fa521e1838	0.696335	0.303665
3	040e15f562a2	0.696335	0.303665
4	046e85c7cc7f	0.696335	0.303665