## CONTENTS


- 1 OBJECTIVE
- 2 INTRODUCTION
  - 2.1 GAMMA & HADRON RAYS
  - 2.2 MOTIVATION BEHIND CLASSIFICATION
- 3 IMPORTING LIBRARIES
- 4 IMPORTING DATA
  - 4.1 DESCRIPTION OF VARIABLES
- 5 DATA DESCRIPTION
  - 5.1 DROPPING THE UNIQUE COLUMNS
  - 5.2 INFORMATION
  - 5.3 STATISTICAL SUMMARY
- 6 EXPLORATORY DATA ANALYSIS
  - 6.1 VISUAL REPORT
  - 6.2 DATA PREPROCESSING
  - 6.3 DATA VISUALIZATION
    - 6.3.1 COMPARISON OF FLENGTH OF HADRON AND GAMMA RAYS
    - 6.3.2 COMPARISON OF FWIDTH OF HADRON AND GAMMA RAYS
    - 6.3.3 COMPARISON OF FConc (PIXEL RATIO) OF HADRON AND GAMMA RAYS
    - 6.3.4 CORRELATION HEAT MAP
  - 6.4 OUTLIERS
    - 6.4.1 OUTLIERS DETECTION
    - 6.4.2 OUTLIERS TREATMENT
      - 6.4.2.1 CHECKING FOR NULL VALUES
      - 6.4.2.2 HANDLING NULL VALUES
  - 6.5 DATA TRANSFORMATION
    - 6.5.1 STANDARDIZATION
- 7 RANDOM SAMPLING
  - 7.1 CHECKING FOR CLASS IMBALANCE
  - 7.2 OVER SAMPLING
- 8 PREDICTIVE ANALYSIS
  - 8.1 LOGISTIC REGRESSION
  - 8.2 DECISION TREE
    - 8.2.1 HYPER PARAMETER TUNING
    - 8.2.2 PREDICTION
  - 8.3 RANDOM FOREST
    - 8.3.1 HYPER PARAMETER TUNING
    - 8.3.2 PREDICTION
- 9 ENSEMBLE TECHNIQUES : BOOSTING
  - 9.1 GRADIENT BOOSTING
    - 9.1.1 HYPER PARAMETER TUNING
    - 9.1.2 PREDICTION
  - 9.2 EXTREME GRADIENT BOOSTING
    - 9.2.1 HYPER PARAMETER TUNING
    - 9.2.2 PREDICTION
- 10 COMPARING THE MODELS


# Source


```
Author > R. K. Bock

Name > Major Atmospheric Gamma Imaging Cherenkov Telescope project (MAGIC)

Site > http://wwwmagic.mppmu.mpg.de

Contact > rkb@mail.cern.ch
```


## OBJECTIVE

```
* The objective of the MAGIC gamma telescope is to develop an classification prediction system for distinguishing
  between gamma rays and hadrons in the field of astrophysics.

* The classification of Gamma and Hadron rays is crucial for unraveling the mysteries of the universe, and the MAGIC
  Telescope dataset provides valuable information for this purpose.

* This study utilizes machine learning techniques such as Logistic Regression, Decision Tree, Random Forest, and
  Ensemble Techniques to predict and differentiate between these high-energy particles.

* By applying these advanced algorithms, our aim is to create a reliable and accurate prediction model that can
  effectively classify Gamma and Hadron rays.
```

## INTRODUCTION

### GAMMA & HADRON RAYS

```
* Gamma Rays: Gamma rays are the highest energy form of electromagnetic radiation.

  They originate from various astrophysical sources such as supernovae, pulsars, and black holes.

  Gamma rays carry valuable information about these extreme cosmic events, allowing scientists to study the most
  energetic processes in the universe.

* Hadron Rays: Hadron rays are particles composed of quarks, such as protons and neutrons.

  They are abundant in cosmic ray showers that result from interactions between high-energy particles and the Earth's
  atmosphere.

  Hadrons can produce signals that mimic gamma rays, making their distinction crucial for accurate astrophysical
  observations.
```

### MOTIVATION BEHIND CLASSIFICATION

```
*  Distinguishing between gamma and hadron rays helps us gain insights into the most energetic and exotic phenomena
   occurring in the universe.

* By separating these two types of particles, scientists can study the processes associated with gamma rays, such as
  the birth and death of stars, cosmic explosions, and the behavior of matter in extreme conditions.

* Discriminating between gamma and hadron rays enables scientists to filter out background noise and focus on the
  genuine gamma-ray signals.

* This distinction ensures that observations and measurements are not contaminated by hadronic signals that can mimic
  gamma-ray signatures.
```

## IMPORTING LIBRARIES

In [1]:
```python
import warnings
import numpy as np
import pandas as pd
import sweetviz as sv
import catboost as cb
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from sklearn.impute import SimpleImputer
from imblearn.over_sampling import SMOTE
pd.set_option("display.max_columns", None)
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import SelectKBest
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from imblearn.under_sampling import RandomUnderSampler
from sklearn.preprocessing import LabelEncoder , StandardScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.ensemble import RandomForestClassifier , AdaBoostClassifier , GradientBoostingClassifier
from sklearn.model_selection import train_test_split, KFold, cross_validate, cross_val_score , GridSearchCV , K
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score, roc_curve, r2_score ,
```

## IMPORTING DATA

In [2]:
```python
DF = pd.read_csv ( 'MagicTelescope.csv' )
DF
```

Out[2]:

| | id | ID | fLength: | fWidth: | fSize: | fConc: | fConc1: | fAsym: | fM3Long: | fM3Trans: | fAlpha: | fDist: | class: |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 28.7967 | 16.0021 | 2.6449 | 0.3918 | 0.1982 | 27.7004 | 22.0110 | -8.2027 | 40.0920 | 81.8828 | g |
| 1 | 2 | 2 | 31.6036 | 11.7235 | 2.5185 | 0.5303 | 0.3773 | 26.2722 | 23.8238 | -9.9574 | 6.3609 | 205.2610 | g |
| 2 | 3 | 3 | 162.0520 | 136.0310 | 4.0612 | 0.0374 | 0.0187 | 116.7410 | -64.8580 | -45.2160 | 76.9600 | 256.7880 | g |
| 3 | 4 | 4 | 23.8172 | 9.5728 | 2.3385 | 0.6147 | 0.3922 | 27.2107 | -6.4633 | -7.1513 | 10.4490 | 116.7370 | g |
| 4 | 5 | 5 | 75.1362 | 30.9205 | 3.1611 | 0.3168 | 0.1832 | -5.5277 | 28.5525 | 21.8393 | 4.6480 | 356.4620 | g |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 19015 | 19016 | 19016 | 21.3846 | 10.9170 | 2.6161 | 0.5857 | 0.3934 | 15.2618 | 11.5245 | 2.8766 | 2.4229 | 106.8258 | h |
| 19016 | 19017 | 19017 | 28.9452 | 6.7020 | 2.2672 | 0.5351 | 0.2784 | 37.0816 | 13.1853 | -2.9632 | 86.7975 | 247.4560 | h |
| 19017 | 19018 | 19018 | 75.4455 | 47.5305 | 3.4483 | 0.1417 | 0.0549 | -9.3561 | 41.0562 | -9.4662 | 30.2987 | 256.5166 | h |
| 19018 | 19019 | 19019 | 120.5135 | 76.9018 | 3.9939 | 0.0944 | 0.0683 | 5.8043 | -93.5224 | -63.8389 | 84.6874 | 408.3166 | h |
| 19019 | 19020 | 19020 | 187.1814 | 53.0014 | 3.2093 | 0.2876 | 0.1539 | -167.3125 | -168.4558 | 31.4755 | 52.7310 | 272.3174 | h |

19020 rows × 13 columns

### DESCRIPTION OF VARIABLES

```
1. fLength: It represents the major axis of an ellipse in millimeters.

          The major axis is the longest diameter of the elliptical shape formed by the detected particle.

          Unit of measurement = mm

2. fWidth: This variable indicates the minor axis of the ellipse in millimeters.

          The minor axis is the shortest diameter of the elliptical shape.

          Unit of measurement = millimeters , mm

3. fSize: It represents the logarithm (base 10) of the sum of the content of all the pixels in the image.

          It provides information about the overall magnitude or intensity of the detected signal.

          Unit of measurement = millimeters , mm

4. fConc: This variable measures the ratio of the sum of the two highest pixels' values over fSize.

          It provides insight into the concentration or distribution of pixel values within the image.

          Unit of measurement = dimensionless ratio

5. fAsym: This variable measures the distance from the highest pixel to the center of the ellipse, projected onto
          the major axis in millimeters.

          It provides information about the asymmetry or displacement of the signal with respect to the center.

          Unit of measurement = millimeters , mm

6. fConc1: It represents the ratio of the value of the highest pixel over fSize.

          This variable provides information about the dominance or intensity of the highest pixel in the image.

          Unit of measurement = dimensionless ratio

7. fM3Long: It represents the third root of the third moment along the major axis of the ellipse in millimeters.

          The third moment is a statistical measure that provides information about the shape or distribution of
          pixel values along the major axis.

          Unit of measurement = millimeters , mm

8. fM3Trans: This variable indicates the third root of the third moment along the minor axis of the ellipse.

          It provides information about the shape or distribution of pixel values along the minor axis.

          Unit of measurement = millimeters , mm

9. fAlpha: It represents the angle (in degrees) between the major axis of the ellipse and a vector connecting the
          origin (coordinate [0, 0]) to the ellipse's center.

          It provides information about the orientation or alignment of the detected signal.

          Unit of measurement = degrees (°)

10. fDist: This variable represents the distance from the origin to the center of the ellipse in millimeters.

          It provides information about the spatial position or radial distance of the detected signal.

          Unit of measurement = millimeters , mm

11. class: This variable indicates the class or type of the detected signal.

          It has two categories: "g" for gamma rays (signal) and "h" for hadron rays (background).

          This variable is used for classification purposes, distinguishing between gamma rays and hadron rays.

          TARGET VARIABLE > Class
```

## DATA DESCRIPTION

### DROPPING THE UNIQUE COLUMNS

In [3]:
```python
DF = DF.iloc [ : , 2 : ]
DF
```

Out[3]:

| | fLength: | fWidth: | fSize: | fConc: | fConc1: | fAsym: | fM3Long: | fM3Trans: | fAlpha: | fDist: | class: |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 28.7967 | 16.0021 | 2.6449 | 0.3918 | 0.1982 | 27.7004 | 22.0110 | -8.2027 | 40.0920 | 81.8828 | g |
| 1 | 31.6036 | 11.7235 | 2.5185 | 0.5303 | 0.3773 | 26.2722 | 23.8238 | -9.9574 | 6.3609 | 205.2610 | g |
| 2 | 162.0520 | 136.0310 | 4.0612 | 0.0374 | 0.0187 | 116.7410 | -64.8580 | -45.2160 | 76.9600 | 256.7880 | g |
| 3 | 23.8172 | 9.5728 | 2.3385 | 0.6147 | 0.3922 | 27.2107 | -6.4633 | -7.1513 | 10.4490 | 116.7370 | g |
| 4 | 75.1362 | 30.9205 | 3.1611 | 0.3168 | 0.1832 | -5.5277 | 28.5525 | 21.8393 | 4.6480 | 356.4620 | g |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 19015 | 21.3846 | 10.9170 | 2.6161 | 0.5857 | 0.3934 | 15.2618 | 11.5245 | 2.8766 | 2.4229 | 106.8258 | h |
| 19016 | 28.9452 | 6.7020 | 2.2672 | 0.5351 | 0.2784 | 37.0816 | 13.1853 | -2.9632 | 86.7975 | 247.4560 | h |
| 19017 | 75.4455 | 47.5305 | 3.4483 | 0.1417 | 0.0549 | -9.3561 | 41.0562 | -9.4662 | 30.2987 | 256.5166 | h |
| 19018 | 120.5135 | 76.9018 | 3.9939 | 0.0944 | 0.0683 | 5.8043 | -93.5224 | -63.8389 | 84.6874 | 408.3166 | h |
| 19019 | 187.1814 | 53.0014 | 3.2093 | 0.2876 | 0.1539 | -167.3125 | -168.4558 | 31.4755 | 52.7310 | 272.3174 | h |

19020 rows × 11 columns

### INFORMATION

In [4]:
```python
Information = pd.DataFrame ( { 'Columns' : DF.columns.tolist ( ) ,
                               'Null Count' : DF.count().tolist ( ) ,
                               'Data Type' : DF.dtypes.tolist ( )
                             } ).reset_index ( ).drop ( columns = [ 'index' ] )
Information
```

Out[4]:

| | Columns | Null Count | Data Type |
|---|---|---|---|
| 0 | fLength: | 19020 | float64 |
| 1 | fWidth: | 19020 | float64 |
| 2 | fSize: | 19020 | float64 |
| 3 | fConc: | 19020 | float64 |
| 4 | fConc1: | 19020 | float64 |
| 5 | fAsym: | 19020 | float64 |
| 6 | fM3Long: | 19020 | float64 |
| 7 | fM3Trans: | 19020 | float64 |
| 8 | fAlpha: | 19020 | float64 |
| 9 | fDist: | 19020 | float64 |
| 10 | class: | 19020 | object |

### STATISTICAL SUMMARY

In [5]:
```python
Description = pd.DataFrame ( DF.describe ( ) ).transpose ( )
Description
```

Out[5]:

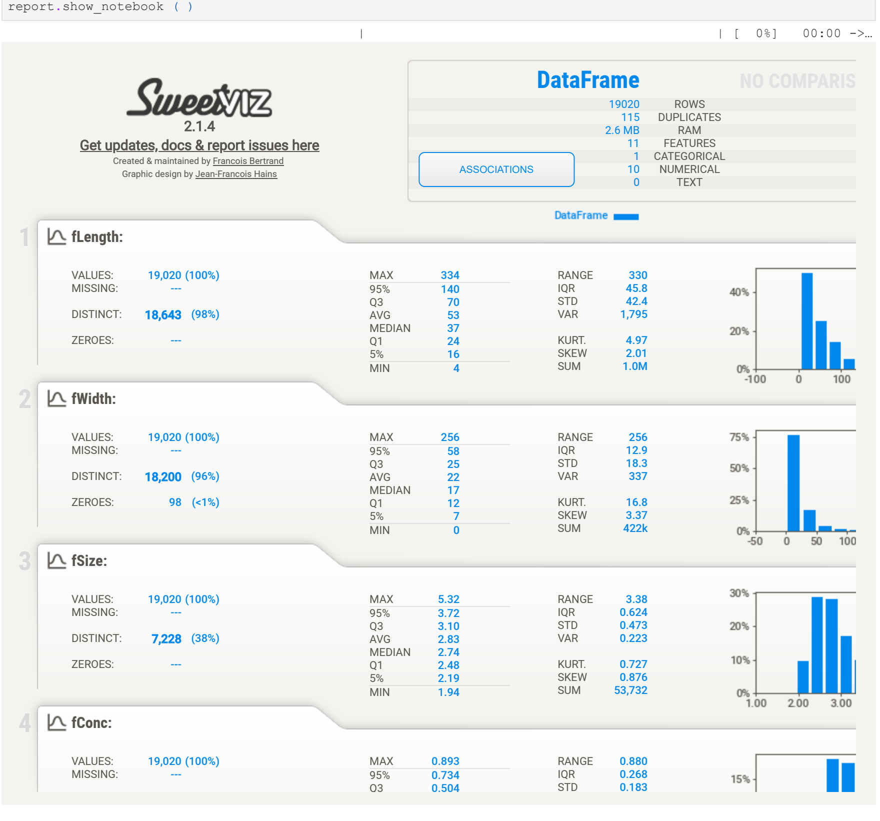
| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| fLength: | 19020.0 | 53.250154 | 42.364855 | 4.2835 | 24.336000 | 37.14770 | 70.122175 | 334.1770 |
| fWidth: | 19020.0 | 22.180966 | 18.346056 | 0.0000 | 11.863800 | 17.13990 | 24.739475 | 256.3820 |
| fSize: | 19020.0 | 2.825017 | 0.472599 | 1.9413 | 2.477100 | 2.73960 | 3.101600 | 5.3233 |
| fConc: | 19020.0 | 0.380327 | 0.182813 | 0.0131 | 0.235800 | 0.35415 | 0.503700 | 0.8930 |
| fConc1: | 19020.0 | 0.214657 | 0.110511 | 0.0003 | 0.128475 | 0.19650 | 0.285225 | 0.6752 |
| fAsym: | 19020.0 | -4.331745 | 59.206062 | -457.9161 | -20.586550 | 4.01305 | 24.063700 | 575.2407 |
| fM3Long: | 19020.0 | 10.545545 | 51.000118 | -331.7800 | -12.842775 | 15.31410 | 35.837800 | 238.3210 |
| fM3Trans: | 19020.0 | 0.249726 | 20.827439 | -205.8947 | -10.849375 | 0.66620 | 10.946425 | 179.8510 |
| fAlpha: | 19020.0 | 27.645707 | 26.103621 | 0.0000 | 5.547925 | 17.67950 | 45.883550 | 90.0000 |
| fDist: | 19020.0 | 193.818026 | 74.731787 | 1.2826 | 142.492250 | 191.85145 | 240.563825 | 495.5610 |

## EXPLORATORY DATA ANALYSIS

### VISUAL REPORT

In [6]:
```python
report = sv.analyze ( DF )
report.show_notebook ( )
```

| [ 0%] 00:00 ->...




### DATA PREPROCESSING

In [7]:
```python
DF.columns = DF.columns.str.replace (  ':' , ''  )
DF
```

Out[7]:

| | fLength | fWidth | fSize | fConc | fConc1 | fAsym | fM3Long | fM3Trans | fAlpha | fDist | class |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 28.7967 | 16.0021 | 2.6449 | 0.3918 | 0.1982 | 27.7004 | 22.0110 | -8.2027 | 40.0920 | 81.8828 | g |
| 1 | 31.6036 | 11.7235 | 2.5185 | 0.5303 | 0.3773 | 26.2722 | 23.8238 | -9.9574 | 6.3609 | 205.2610 | g |
| 2 | 162.0520 | 136.0310 | 4.0612 | 0.0374 | 0.0187 | 116.7410 | -64.8580 | -45.2160 | 76.9600 | 256.7880 | g |
| 3 | 23.8172 | 9.5728 | 2.3385 | 0.6147 | 0.3922 | 27.2107 | -6.4633 | -7.1513 | 10.4490 | 116.7370 | g |
| 4 | 75.1362 | 30.9205 | 3.1611 | 0.3168 | 0.1832 | -5.5277 | 28.5525 | 21.8393 | 4.6480 | 356.4620 | g |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 19015 | 21.3846 | 10.9170 | 2.6161 | 0.5857 | 0.3934 | 15.2618 | 11.5245 | 2.8766 | 2.4229 | 106.8258 | h |
| 19016 | 28.9452 | 6.7020 | 2.2672 | 0.5351 | 0.2784 | 37.0816 | 13.1853 | -2.9632 | 86.7975 | 247.4560 | h |
| 19017 | 75.4455 | 47.5305 | 3.4483 | 0.1417 | 0.0549 | -9.3561 | 41.0562 | -9.4662 | 30.2987 | 256.5166 | h |
| 19018 | 120.5135 | 76.9018 | 3.9939 | 0.0944 | 0.0683 | 5.8043 | -93.5224 | -63.8389 | 84.6874 | 408.3166 | h |
| 19019 | 187.1814 | 53.0014 | 3.2093 | 0.2876 | 0.1539 | -167.3125 | -168.4558 | 31.4755 | 52.7310 | 272.3174 | h |

19020 rows × 11 columns

In [8]:
```python
DF['class'].replace ( { 'g' : 0 , 'h' : 1 } , inplace = True )
DF['class'].value_counts ( )
```

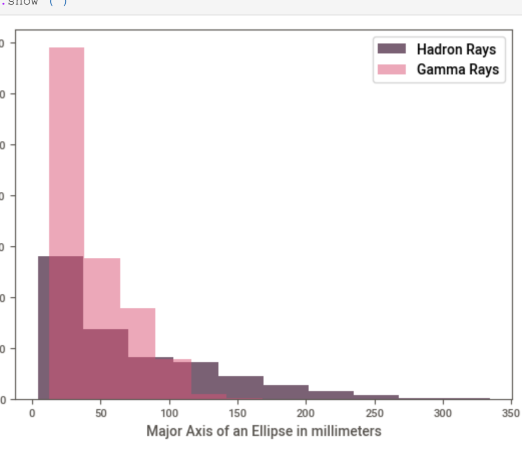
Out[8]:
```
0    12332
1     6688
Name: class, dtype: int64
```

### DATA VISUALIZATION

#### COMPARISON OF FLENGTH OF HADRON AND GAMMA RAYS

```
* 'fLength' specifically refers to the major axis of an ellipse that is fitted to the shower image detected by the
  telescope.

* This feature is measured in millimeters and provides information about the spatial extent of the shower image.

* A "shower image" refers to the pattern of Cherenkov light produced by the electromagnetic shower generated when
  high-energy gamma rays or cosmic rays interact with the Earth's atmosphere.
```

In [9]:
```python
plt.hist ( DF [ DF [ 'class' ] == 1 ] [ 'fLength' ] , color = '#7A6174' , bins = 10 , label = 'Hadron Rays'  );
plt.hist ( DF [ DF [ 'class' ] == 0 ] [ 'fLength' ] , color = '#DB5375' , bins = 10 , label = 'Gamma Rays' , al
plt.xlabel ( 'Major Axis of an Ellipse in millimeters' )
plt.legend ( )
plt.show ( )
```

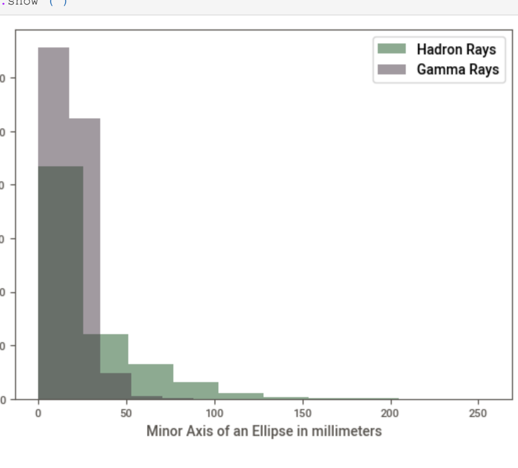



```
INFERENCE

    * The 'FLength' (Major Axis of an Ellipse in millimeters) is identified as a distinguishing feature between gamma
      rays and Hadron rays. The significant difference of nearly 4000 mm suggests that this feature could be critical
      in discriminating between the two types of particles.

    * It suggests that the spatial extent of the shower image produced by gamma rays is much larger than that of Hadron
      rays. This could be due to different interactions of gamma rays and Hadron rays with the Earth's atmosphere.

    *  Gamma rays are high-energy photons, whereas Hadron rays consist of particles like protons and neutrons.

    * When gamma rays enter the Earth's atmosphere, they interact with air nuclei, leading to the creation of an
      electromagnetic cascade.

    * On the other hand, Hadron rays, being composed of charged particles, undergo strong interactions with atomic
      nuclei, leading to a hadronic cascade.
```

#### COMPARISON OF FWIDTH OF HADRON AND GAMMA RAYS

```
* This feature represents the minor axis length of an ellipse that is fitted to the shower image detected by the MAGIC
  Telescope during the observation of high-energy gamma-ray and background (non-gamma-ray) events.
```

In [10]:
```python
plt.hist ( DF [ DF [ 'class' ] == 1 ] [ 'fWidth' ] , color =  '#8DAA91' , bins = 10 , label = 'Hadron Rays' );
plt.hist ( DF [ DF [ 'class' ] == 0 ] [ 'fWidth' ] , color = '#453643' , bins = 10 , label = 'Gamma Rays' , alp
plt.xlabel ( 'Minor Axis of an Ellipse in millimeters' )
plt.legend ( )
plt.show ( )
```

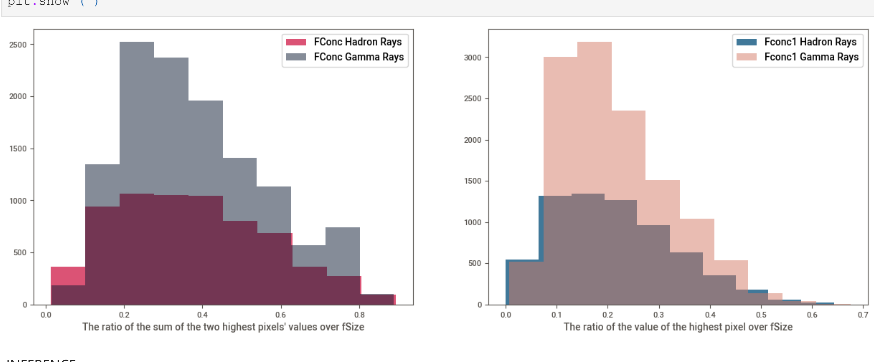



```
INFERENCE

    * The difference in 'Fwidth' is a result of the distinct energy deposition and interactions of gamma rays and
      Hadron rays with the Earth's atmosphere.

    * The wider 'Fwidth' for gamma rays suggests a more extensive and energetic electromagnetic cascade compared to the
      more confined hadronic cascade for Hadron rays.

    * When gamma rays enter the Earth's atmosphere, they interact with air nuclei, initiating an electromagnetic
      cascade.

    * This cascade involves the production of high-energy electrons and positrons, which travel at nearly the speed of
      light.

    * These charged particles emit Cherenkov radiation and create an extensive and widespread shower.

    * The Cherenkov radiation emitted by high-energy electrons and positrons in the electromagnetic cascade produces a
      wide and elongated shower image.

    * As a result, the fitted ellipse to this shower image has a larger minor axis  ('Fwidth').
```

#### COMPARISON OF FConc (PIXEL RATIO) OF HADRON AND GAMMA RAYS

```
FConc

* Each pixel in the camera records the intensity of the Cherenkov light it receives.

* The higher the intensity, the higher the energy of the particles detected in that region.

* 'FSize' is another feature in the dataset and represents the "size of the fitted ellipse in the shower image."

* It is related to the spatial extent of the shower image, typically measured in number of pixels.

* 'Fconc' is calculated by taking the sum of the two highest pixel values in the shower image and then dividing that
  sum by 'FSize.'

* It represents the concentration of the Cherenkov light within the shower image.

* A higher value of 'Fconc' indicates that a significant portion of the Cherenkov light is concentrated within a
  smaller area of the shower image.

* Conversely, a lower value of 'Fconc' suggests a more spread-out distribution of light intensity.

FConc1

* 'Fconc1' is calculated by dividing the value of the highest pixel intensity by 'FSize.'

* It represents the proportion of the Cherenkov light concentration contributed by the pixel with the highest
  intensity.

* A higher value of 'Fconc1' suggests that a significant portion of the Cherenkov light is concentrated in the pixel
  with the highest intensity, indicating a highly localized and intense light emission in the shower image.

* 'Fconc1' can be important in discriminating between different types of particle events. It may help differentiate
  between gamma-ray events and background events (non-gamma-ray events).

* Gamma-ray showers, being more focused and intense, might have higher 'Fconc1' values, while background showers, which
  are more diffuse, might have lower 'Fconc1' values.
```

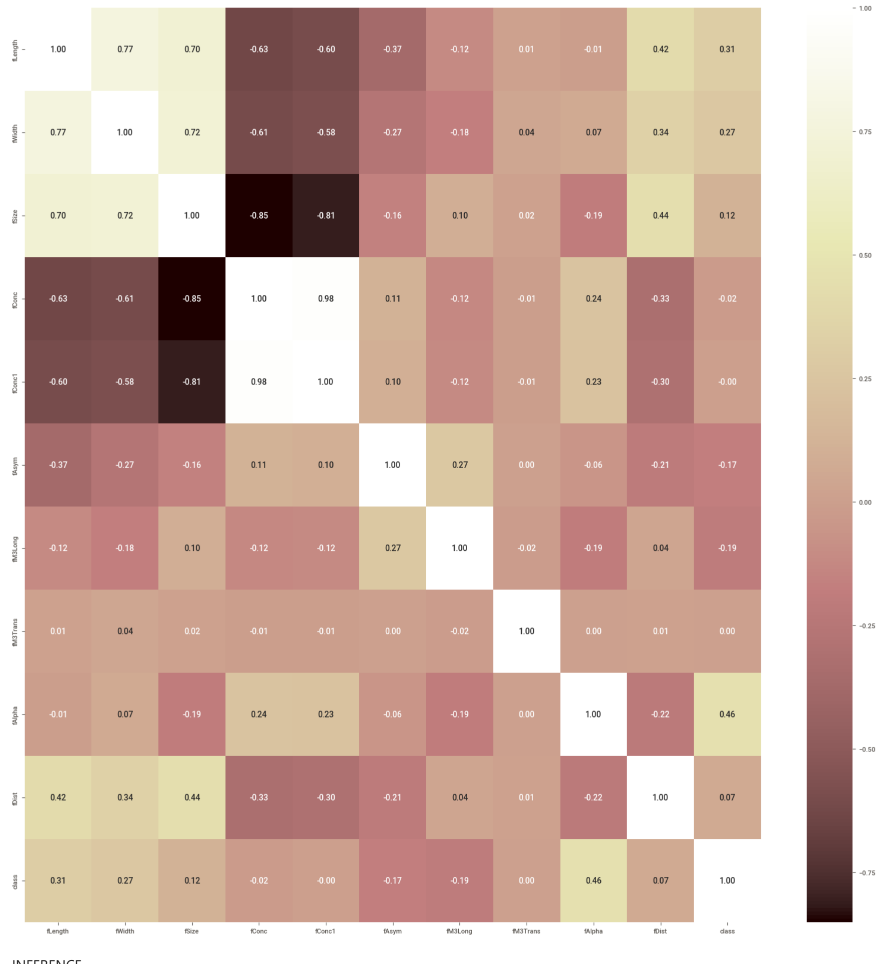
In [11]:
```python
plt.figure ( figsize = ( 15 , 5 ) )
plt.subplot(1, 2, 1)
plt.hist ( DF [ DF [ 'class' ] == 1 ] [ 'fConc' ] , color = '#DB5375' , bins = 10 , label = 'FConc Hadron Rays'
plt.hist ( DF [ DF [ 'class' ] == 0 ] [ 'fConc' ] , color = '#0C1B33' , bins = 10 , label = 'FConc Gamma Rays'
plt.xlabel ( "The ratio of the sum of the two highest pixels' values over fSize" )
plt.legend ( )
plt.subplot(1, 2, 2)
plt.hist ( DF [ DF [ 'class' ] == 1 ] [ 'fConc1' ] , color = '#407899' , bins = 10 , label = 'Fconc1 Hadron Ray
plt.hist ( DF [ DF [ 'class' ] == 0 ] [ 'fConc1' ] , color = '#D57A66' , bins = 10 , label = 'Fconc1 Gamma Rays
plt.xlabel ( "The ratio of the value of the highest pixel over fSize" )
plt.legend ( )
plt.show ( )
```




```
INFERENCE

FCONC

    * This implies that the Cherenkov light emitted by gamma rays tends to be more concentrated in a smaller area
      within the shower image, while Hadron rays exhibit a more spread-out distribution.

    * The Cherenkov light emitted by the high-energy electrons and positrons in the electromagnetic cascade tends to be
      more concentrated within a smaller spatial area. This concentration of Cherenkov light results in a higher
      'Fconc' value for gamma rays.

    * The higher energy of gamma rays compared to most Hadron rays contributes to more extensive and energetic
      electromagnetic cascades. The higher energy deposition in the atmosphere leads to a more concentrated shower for
      gamma rays, thus increasing 'Fconc.'

    * Gamma rays have no rest mass, whereas Hadron rays (e.g., protons, neutrons) have rest masses.

    * This mass difference affects the energy deposition and interactions of the particles in the atmosphere,
      influencing the characteristics of the generated showers and, consequently, the 'Fconc' measurements.


FCONC1

    * The higher 'FConc1' values for gamma rays suggest that gamma-ray showers tend to have a more intense, localized,
      and focused light emission, with the highest pixel being a major contributor to the total Cherenkov light
      observed by the telescope.

    * The lower 'FConc1' values for Hadron rays indicate that the distribution of Cherenkov light in Hadron shower
      images is more evenly spread out, with no single pixel dominating the emission.

    * The difference in 'FConc1' can be attributed to the varying energy and particle interactions of gamma rays and
      Hadron rays with the Earth's atmosphere.

    * Higher-energy gamma rays might produce more intense showers, leading to higher 'FConc1' values.
```

#### CORRELATION HEAT MAP

In [12]:
```python
Correlation_Matrix = DF.corr ( )
plt.figure ( figsize = ( 20 , 20 ) )
sns.heatmap ( Correlation_Matrix , annot = True , fmt = '0.2f' , cmap = 'pink' );
```




INFERENCE

* The correlation between Fconc and FLength is -0.63

```
    - This negative correlation suggests that as the value of FLength (major axis of the ellipse) increases, the
      value of Fconc (ratio of the sum of the two highest pixels' values over fSize) tends to decrease.

    - In other words, as the elongation of the ellipse increases (larger FLength), the concentration of pixel values
      in the image tends to be lower (smaller Fconc).

    - This relationship could potentially indicate that Gamma and Hadron rays may have different patterns of
      elongation and pixel concentration in the images.
```

* The correlation between Fconc and FWidth is -0.61

```
     - Similarly, this negative correlation indicates that as the value of FWidth (minor axis of the ellipse)
       increases, the value of Fconc decreases.

     - It means that more elongated ellipses (larger FWidth) tend to have lower pixel concentration (smaller Fconc).

     - Once again, this relationship might have implications for differentiating between Gamma and Hadron rays, as
       they could exhibit distinct patterns of elongation and pixel distribution in the images.
```

* The correlation between Fconc and FSize is -0.85

```
     - The strong negative correlation between Fconc and FSize (logarithm of the sum of the content of all pixels in the
       image) indicates that as the overall size of the image increases (larger FSize), the pixel concentration (Fconc)
       tends to decrease.

     - Larger images may have a more spread-out distribution of pixel values, resulting in a lower concentration.

     - This correlation could be valuable in understanding the relationship between the size of the image and the
       concentration of pixel values in differentiating between Gamma and Hadron rays.
```

* The correlation between Fconc and FConc1 is 0.98

```
     - This very high positive correlation between Fconc and FConc1 (ratio of the value of the highest pixel over FSize)
       suggests that Fconc1 and Fconc are almost perfectly linearly related.

     - When FConc1 increases (highest pixel value relative to FSize), Fconc tends to increase proportionally.

     - This relationship implies that the highest pixel value in the image has a significant impact on the overall
       concentration of pixel values.

     - This relationship could be crucial in distinguishing between Gamma and Hadron rays, as they may exhibit distinct
       patterns of the highest pixel concentration relative to the overall image size.
```

* The correlation between FLength and FWidth is 0.77

```
     - The positive correlation of 0.77 between FLength and FWidth suggests that there is a strong linear relationship
       between the major axis (FLength) and the minor axis (FWidth) of the ellipse in millimeters.

     - This could mean that as the major axis of the ellipse increases, the
       minor  axis tends to increase as well.

     - It implies that the shapes of the detected images tend to be elongated along both axes when observing Gamma and
       Hadron rays.
```

* The correlation between FLength and FSize is 0.70

```
     - The positive correlation of 0.70 between FLength and FSize indicates a moderate linear relationship between the
       major axis (FLength) and the logarithm (base 10) of the sum of the content of all the pixels in the image
       (FSize).

     - This correlation suggests that as the major axis of the ellipse increases, the sum of pixel values tends to
       increase logarithmically.

     - This could imply that larger and more extended images are associated with higher pixel values, which might be
       expected as more energetic rays are detected, leading to larger images with higher pixel content.
```

* The correlation between FLength and FConc1 is -0.60

```
     - The negative correlation of -0.60 between FLength and FConc1 suggests a moderate inverse linear relationship
       between the major axis (FLength) and the ratio of the value of the highest pixel over FSize (FConc1).

     - This means that as the major axis of the ellipse increases, the ratio of the highest pixel value to the sum of
       all pixel values tends to decrease.

     - In the context of Gamma and Hadron rays, this might imply that elongated images are associated with a more
       uniform distribution of pixel values, with less reliance on a single highest pixel value.
```

* The correlation between FWidth and FSize is 0.72

```
     -  This relationship might be due to the fact that when the minor axis of an ellipse is larger, it covers more
        area, leading to a higher sum of pixel contents in the image.

     - This correlation could be significant for understanding the characteristics of the detected particles.

     - For instance, it could imply that images with larger minor axes of the ellipse (indicating broader spatial
       distributions) are more likely to correspond to Gamma rays, as Gamma rays tend to spread out more due to their
       electromagnetic nature.

     - On the other hand, Hadron rays, being more massive and charged, might have more concentrated energy depositions
       with smaller minor axes.
```

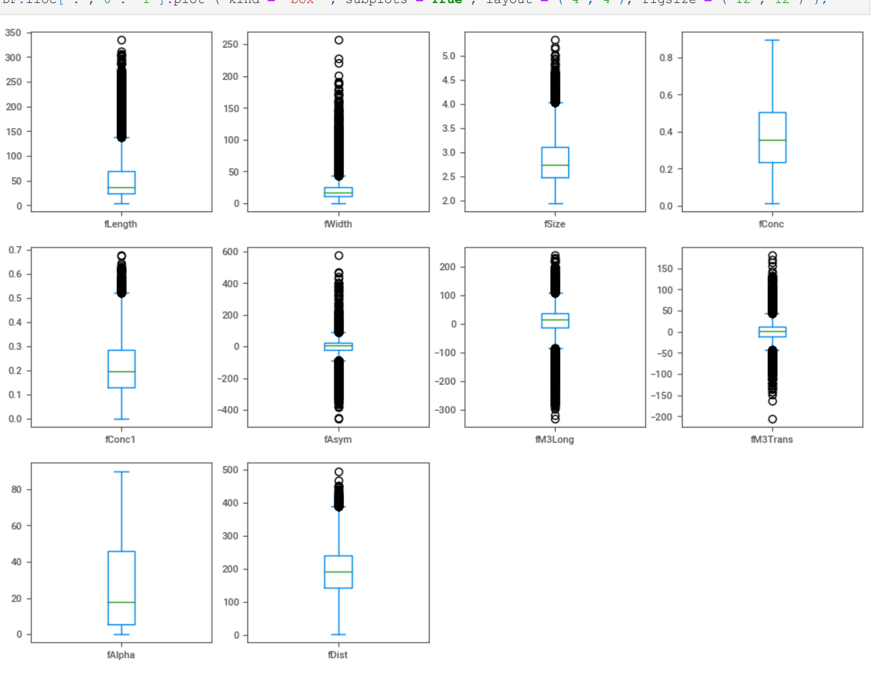
* The correlation between FWidth and FConc1 is -0.58

```
     - This correlation suggests that when the minor axis of the ellipse (FWidth) increases, the ratio of the value of
       the highest pixel over FSize (FConc1) tends to decrease.

     - Conversely, when the minor axis decreases, the ratio tends to increase.

     - This correlation could be linked to the characteristics of energy deposition in the images.

     - A higher FConc1 value means that the highest pixel value contributes significantly to the overall sum of pixel
       contents, indicating a more focused and localized energy deposition.

     - Therefore, a negative correlation with FWidth could imply that when the minor axis (FWidth) of the ellipse is
       larger, the energy deposition tends to be more spread out (lower FConc1), possibly corresponding to Gamma rays.

     - Conversely, when the minor axis is smaller, the energy deposition is more concentrated (higher FConc1), which
       could be indicative of Hadron rays.
```

## OUTLIERS

### OUTLIERS DETECTION

In [13]:
```
DF.iloc[ : , 0 : -1 ].plot ( kind = 'box' , subplots = True , layout = ( 4 , 4 ), figsize = ( 12 , 12 ) );
```




```
INFERENCE
    * We can see the presence of outliers in the following columns or features
        - FLength
        - FWidth
        - FSize
        - FConc1
        - FAsym
        - FM3Long
        - FM3Trans
        - FDist

    1. Outliers in FLength, FWidth, and FSize:

        * FLength, FWidth, and FSize likely represent measurements related to the size and shape of the detected
          objects in the telescope images.

        * Presence of outliers in these columns might indicate rare or extreme cases of objects with significantly
          different sizes or shapes compared to the majority of observations.

    2. Outliers in FConc1:

        * FConc1 may represent a measure of concentration or compactness of the detected objects.

        * The presence of outliers in FConc1 suggests the existence of objects that are exceptionally compact or
         dispersed in the telescope images compared to the rest of the dataset.

    3. Outliers in FAsym:

        * The presence of outliers in this column suggests the existence of highly asymmetric objects in the data,
          which might be rare or unusual cases.

    4. Outliers in FM3Long and FM3Trans:

        * FM3Long and FM3Trans probably represent features related to the third moment of the distribution of detected
          objects' lengths and their transformation.

        * The presence of outliers in these columns implies significant deviations from the typical length distribution
          of objects in the data.

    5. Outliers in FDist:

       * FDist likely represents the distance of the detected objects from the telescope.

       * Outliers in this column could indicate objects that are much closer or much farther away from the telescope
         than the majority of observations.
```

### OUTLIERS TREATMENT

```
The threshold of 3 for Z-Score is based on the empirical rule for normal distributions:

    * Approximately 68% of the data falls within 1 standard deviation from the mean (Z-Score within ±1).

    * Approximately 95% of the data falls within 2 standard deviations from the mean (Z-Score within ±2).

    * Approximately 99.7% of the data falls within 3 standard deviations from the mean (Z-Score within ±3).
```

In [14]:
```
Z_Score = np.abs ( ( DF - DF.mean ( ) ) / DF.std ( ) )

Threshold = 3

DF = DF.where ( ~ ( ( Z_Score >= Threshold ) | ( Z_Score <= -Threshold ) ) )

DF
```

Out[14]:

| | fLength | fWidth | fSize | fConc | fConc1 | fAsym | fM3Long | fM3Trans | fAlpha | fDist | class |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 28.7967 | 16.0021 | 2.6449 | 0.3918 | 0.1982 | 27.7004 | 22.0110 | -8.2027 | 40.0920 | 81.8828 | 0 |
| 1 | 31.6036 | 11.7235 | 2.5185 | 0.5303 | 0.3773 | 26.2722 | 23.8238 | -9.9574 | 6.3609 | 205.2610 | 0 |
| 2 | 162.0520 | NaN | 4.0612 | 0.0374 | 0.0187 | 116.7410 | -64.8580 | -45.2160 | 76.9600 | 256.7880 | 0 |
| 3 | 23.8172 | 9.5728 | 2.3385 | 0.6147 | 0.3922 | 27.2107 | -6.4633 | -7.1513 | 10.4490 | 116.7370 | 0 |
| 4 | 75.1362 | 30.9205 | 3.1611 | 0.3168 | 0.1832 | -5.5277 | 28.5525 | 21.8393 | 4.6480 | 356.4620 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 19015 | 21.3846 | 10.9170 | 2.6161 | 0.5857 | 0.3934 | 15.2618 | 11.5245 | 2.8766 | 2.4229 | 106.8258 | 1 |
| 19016 | 28.9452 | 6.7020 | 2.2672 | 0.5351 | 0.2784 | 37.0816 | 13.1853 | -2.9632 | 86.7975 | 247.4560 | 1 |
| 19017 | 75.4455 | 47.5305 | 3.4483 | 0.1417 | 0.0549 | -9.3561 | 41.0562 | -9.4662 | 30.2987 | 256.5166 | 1 |
| 19018 | 120.5135 | 76.9018 | 3.9939 | 0.0944 | 0.0683 | 5.8043 | -93.5224 | NaN | 84.6874 | 408.3166 | 1 |
| 19019 | NaN | 53.0014 | 3.2093 | 0.2876 | 0.1539 | -167.3125 | NaN | 31.4755 | 52.7310 | 272.3174 | 1 |

19020 rows × 11 columns

### CHECKING FOR NULL VALUES

In [15]:
```
DF.isnull ( ).sum ( ) [ DF.isnull ( ).sum ( ) > 0 ]
```

Out[15]:
```
fLength     431
fWidth      487
fSize       159
fConc1       99
fAsym       421
fM3Long     398
fM3Trans    398
fDist        27
dtype: int64
```

### HANDLING NULL VALUES

In [16]:
```
Imputer = SimpleImputer ( strategy = 'median' )

DF_Imputed_Array = Imputer.fit_transform ( DF )

DF = pd.DataFrame ( DF_Imputed_Array , columns = DF.columns )

DF [ 'class' ] = DF [ 'class' ].astype ( int )

DF
```

Out[16]:

| | fLength | fWidth | fSize | fConc | fConc1 | fAsym | fM3Long | fM3Trans | fAlpha | fDist | class |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 28.7967 | 16.0021 | 2.6449 | 0.3918 | 0.1982 | 27.7004 | 22.0110 | -8.20270 | 40.0920 | 81.8828 | 0 |
| 1 | 31.6036 | 11.7235 | 2.5185 | 0.5303 | 0.3773 | 26.2722 | 23.8238 | -9.95740 | 6.3609 | 205.2610 | 0 |
| 2 | 162.0520 | 16.8769 | 4.0612 | 0.0374 | 0.0187 | 116.7410 | -64.8580 | -45.21600 | 76.9600 | 256.7880 | 0 |
| 3 | 23.8172 | 9.5728 | 2.3385 | 0.6147 | 0.3922 | 27.2107 | -6.4633 | -7.15130 | 10.4490 | 116.7370 | 0 |
| 4 | 75.1362 | 30.9205 | 3.1611 | 0.3168 | 0.1832 | -5.5277 | 28.5525 | 21.83930 | 4.6480 | 356.4620 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 19015 | 21.3846 | 10.9170 | 2.6161 | 0.5857 | 0.3934 | 15.2618 | 11.5245 | 2.87660 | 2.4229 | 106.8258 | 1 |
| 19016 | 28.9452 | 6.7020 | 2.2672 | 0.5351 | 0.2784 | 37.0816 | 13.1853 | -2.96320 | 86.7975 | 247.4560 | 1 |
| 19017 | 75.4455 | 47.5305 | 3.4483 | 0.1417 | 0.0549 | -9.3561 | 41.0562 | -9.46620 | 30.2987 | 256.5166 | 1 |
| 19018 | 120.5135 | 76.9018 | 3.9939 | 0.0944 | 0.0683 | 5.8043 | -93.5224 | 0.49995 | 84.6874 | 408.3166 | 1 |
| 19019 | 36.3453 | 53.0014 | 3.2093 | 0.2876 | 0.1539 | -167.3125 | 15.7910 | 31.47550 | 52.7310 | 272.3174 | 1 |

19020 rows × 11 columns

## DATA TRANSFORMATION

### STANDARDIZATION

In [17]:
```
DF_1 = DF.iloc [ : , 0 : - 1 ]

DF_2 = ( DF_1 - DF_1.mean ( ) ) / DF_1.std ( )

DF_2['class'] = DF['class']

DF = DF_2

DF
```

Out[17]:

| | fLength | fWidth | fSize | fConc | fConc1 | fAsym | fM3Long | fM3Trans | fAlpha | fDist | class |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -0.599491 | -0.325117 | -0.370011 | 0.062758 | -0.134719 | 0.622872 | 0.193350 | -0.517870 | 0.476803 | -1.504435 | 0 |
| 1 | -0.516776 | -0.674224 | -0.652222 | 0.820362 | 1.534857 | 0.591745 | 0.236370 | -0.626472 | -0.815397 | 0.158928 | 0 |
| 2 | 3.327319 | -0.253739 | 2.792134 | -1.875834 | -1.808023 | 2.563484 | -1.868171 | -2.808701 | 1.889174 | 0.853605 | 0 |
| 3 | -0.746228 | -0.849708 | -1.054104 | 1.282035 | 1.673755 | 0.612199 | -0.482385 | -0.452797 | -0.658786 | -1.034537 | 0 |
| 4 | 0.766056 | 0.892131 | 0.782499 | -0.347497 | -0.274549 | -0.101324 | 0.348588 | 1.341492 | -0.881016 | 2.197392 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 19015 | -0.817913 | -0.740029 | -0.434312 | 1.123403 | 1.684941 | 0.351777 | -0.055509 | 0.167851 | -0.966257 | -1.168158 | 1 |
| 19016 | -0.595115 | -1.083947 | -1.213294 | 0.846618 | 0.612908 | 0.827332 | -0.016096 | -0.193587 | 2.266038 | 0.727793 | 1 |
| 19017 | 0.775171 | 2.247402 | 1.423724 | -1.305306 | -1.470566 | -0.184762 | 0.645318 | -0.596071 | 0.101633 | 0.849946 | 1 |
| 19018 | 2.103249 | 4.643916 | 2.641875 | -1.564040 | -1.345651 | 0.145654 | -2.548417 | 0.020755 | 2.185202 | 2.896486 | 1 |
| 19019 | -0.377046 | 2.693794 | 0.890114 | -0.507223 | -0.547685 | -3.627372 | 0.045740 | 1.937897 | 0.960989 | 1.062970 | 1 |

19020 rows × 11 columns

## RANDOM SAMPLING

In [18]:
```
Train , Test = train_test_split ( DF , random_state = 42 )

Train_X = Train.iloc [ : , 0 : - 1 ]

Train_Y = Train.iloc [ : , - 1 ]

Test_X = Test.iloc [ : , 0 : - 1 ]

Test_Y = Test.iloc [ : , - 1 ]

print ( "\nThe shapes of the test and train data frames are : " )

print ( '\nTrain = ' , Train.shape , '\n\nTest = ' , Test.shape   )

print ( '\nTrain_X = ' , Train_X.shape  ,  '\n\nTrain_Y = ' , Train_Y.shape )

print ( '\nTest_X = ' , Test_X.shape  ,  '\n\nTest_Y = ' , Test_Y.shape )
```

```
The shapes of the test and train data frames are :

Train =  (14265, 11)

Test =  (4755, 11)

Train_X =  (14265, 10)

Train_Y =  (14265,)

Test_X =  (4755, 10)

Test_Y =  (4755,)
```

### CHECKING FOR CLASS IMBALANCE

In [19]:
```
print ( "The class distribution in Train Data is : \n\n" , pd.DataFrame ( Train_Y.value_counts ( ) ) )

print ( "\nThe class distribution in Test Data is : \n\n" , pd.DataFrame ( Test_Y.value_counts ( ) ) )
```

```
The class distribution in Train Data is :

    class
0   9240
1   5025

The class distribution in Test Data is :

    class
0   3092
1   1663
```

### OVER SAMPLING

In [20]:
```
Sample = DF [ DF['class'] == 1 ]

DF = pd.concat ( [DF , Sample] )

DF
```

Out[20]:

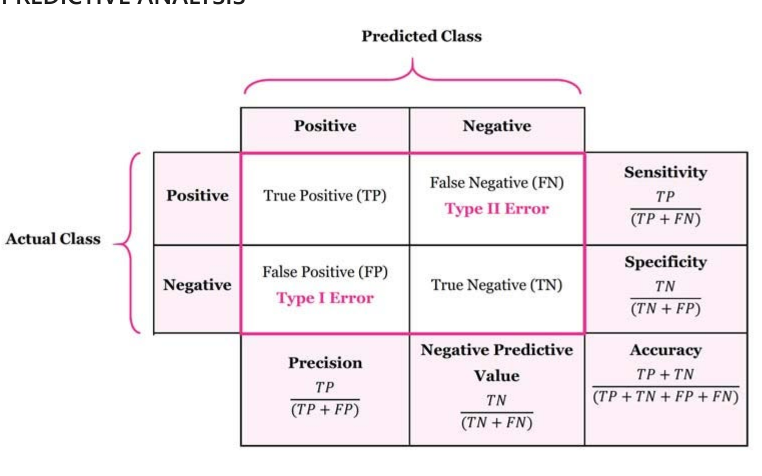
| | fLength | fWidth | fSize | fConc | fConc1 | fAsym | fM3Long | fM3Trans | fAlpha | fDist | class |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -0.599491 | -0.325117 | -0.370011 | 0.062758 | -0.134719 | 0.622872 | 0.193350 | -0.517870 | 0.476803 | -1.504435 | 0 |
| 1 | -0.516776 | -0.674224 | -0.652222 | 0.820362 | 1.534857 | 0.591745 | 0.236370 | -0.626472 | -0.815397 | 0.158928 | 0 |
| 2 | 3.327319 | -0.253739 | 2.792134 | -1.875834 | -1.808023 | 2.563484 | -1.868171 | -2.808701 | 1.889174 | 0.853605 | 0 |
| 3 | -0.746228 | -0.849708 | -1.054104 | 1.282035 | 1.673755 | 0.612199 | -0.482385 | -0.452797 | -0.658786 | -1.034537 | 0 |
| 4 | 0.766056 | 0.892131 | 0.782499 | -0.347497 | -0.274549 | -0.101324 | 0.348588 | 1.341492 | -0.881016 | 2.197392 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 19015 | -0.817913 | -0.740029 | -0.434312 | 1.123403 | 1.684941 | 0.351777 | -0.055509 | 0.167851 | -0.966257 | -1.168158 | 1 |
| 19016 | -0.595115 | -1.083947 | -1.213294 | 0.846618 | 0.612908 | 0.827332 | -0.016096 | -0.193587 | 2.266038 | 0.727793 | 1 |
| 19017 | 0.775171 | 2.247402 | 1.423724 | -1.305306 | -1.470566 | -0.184762 | 0.645318 | -0.596071 | 0.101633 | 0.849946 | 1 |
| 19018 | 2.103249 | 4.643916 | 2.641875 | -1.564040 | -1.345651 | 0.145654 | -2.548417 | 0.020755 | 2.185202 | 2.896486 | 1 |
| 19019 | -0.377046 | 2.693794 | 0.890114 | -0.507223 | -0.547685 | -3.627372 | 0.045740 | 1.937897 | 0.960989 | 1.062970 | 1 |

25708 rows × 11 columns

In [21]:
```
Train , Test = train_test_split ( DF , random_state = 42 )

Train_X = Train.iloc [ : , 0 : - 1 ]

Train_Y = Train.iloc [ : , - 1 ]

Test_X = Test.iloc [ : , 0 : - 1 ]

Test_Y = Test.iloc [ : , - 1 ]

print ( "The class distribution in Train Data after oversampling is : \n\n" , pd.DataFrame ( Train_Y.value_coun

print ( "\nThe class distribution in Test Data after oversampling is : \n\n" , pd.DataFrame ( Test_Y.value_coun
```

```
The class distribution in Train Data after oversampling is :

    class
1  10057
0   9224

The class distribution in Test Data after oversampling is :

    class
1   3319
0   3108
```

## PREDICTIVE ANALYSIS

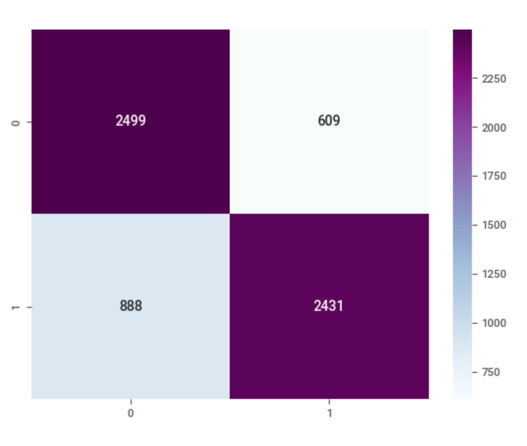



### LOGISTIC REGRESSION

In [22]:
```
LR = LogisticRegression ( )

LR.fit ( Train_X , Train_Y )

Y_LR = LR.predict ( Test_X )

Accuracy_LR = accuracy_score ( Test_Y , Y_LR ) * 100

Precision_LR = precision_score ( Test_Y , Y_LR ) * 100

Recall_LR = recall_score ( Test_Y , Y_LR ) * 100

F1_Score_LR = f1_score ( Test_Y , Y_LR ) * 100

Confusion_LR = confusion_matrix ( Test_Y , Y_LR )

FPR_LR = Confusion_LR[0][1] / ( Confusion_LR [0][0] + Confusion_LR [0][1] )

LR_Performance_Metrics = { 'Accuracy ' : Accuracy_LR , 'Precision ' : Precision_LR , 'Recall ' : Recall_LR ,

                           'F1_Score ' : F1_Score_LR , 'FPR ' : FPR_LR }

LR_Performance_Metrics = pd.DataFrame ( np.round ( list(LR_Performance_Metrics.values()), 2 ) , index = list(LR

display ( sns.heatmap ( pd.DataFrame ( Confusion_LR ) , annot = True , fmt = '0.0f' , cmap = 'BuPu' ) );

print ( "The performance parameters of the logistic regression model : \n" )

display ( LR_Performance_Metrics )
```

```
<AxesSubplot:>
The performance parameters of the logistic regression model :
```

| | Values (%) |
|---|---|
| Accuracy | 76.71 |
| Precision | 79.97 |
| Recall | 73.24 |
| F1_Score | 76.46 |
| FPR | 0.20 |




### DECISION TREE

#### HYPER PARAMETER TUNING

In [23]:
```
DT = DecisionTreeClassifier()

Parameters = {
    'criterion': ['gini', 'entropy'],

    'max_depth': [None, 5, 10, 15],

    'min_samples_split': [2, 5, 10],

    'min_samples_leaf': [1, 2, 4]

}

GS = GridSearchCV ( estimator = DT , param_grid = Parameters , cv = 5 )

GS.fit ( Train_X, Train_Y )

print("Best Hyperparameters : ", GS.best_params_ )

Best_Model = GS.best_estimator_

Validation_Accuracy = Best_Model.score ( Test_X, Test_Y )

print( "\nValidation Set Accuracy with Best Model : ", Validation_Accuracy)
```

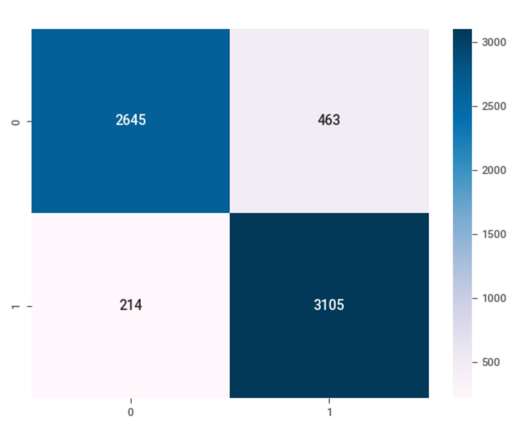
```
Best Hyperparameters :  {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}

Validation Set Accuracy with Best Model :  0.8929516103936518
```

#### PREDICTION

In [24]:
```
DT = DecisionTreeClassifier ( **GS.best_params_ )

DT.fit ( Train_X , Train_Y )

Y_DT = DT.predict ( Test_X )

Accuracy_DT = accuracy_score ( Test_Y , Y_DT ) * 100

Precision_DT = precision_score ( Test_Y , Y_DT ) * 100

Recall_DT = recall_score ( Test_Y , Y_DT ) * 100

F1_Score_DT = f1_score ( Test_Y , Y_DT ) * 100

Confusion_DT = confusion_matrix ( Test_Y , Y_DT )

FPR_DT = Confusion_DT[0][1] / ( Confusion_DT [0][0] + Confusion_DT [0][1] )

DT_Performance_Metrics = { 'Accuracy ' : Accuracy_DT , 'Precision ' : Precision_DT , 'Recall ' : Recall_DT ,

                           'F1_Score ' : F1_Score_DT , 'FPR ' : FPR_DT }

DT_Performance_Metrics = pd.DataFrame ( np.round ( list(DT_Performance_Metrics.values()), 2 ) , index = list(DT

display ( sns.heatmap ( pd.DataFrame ( Confusion_DT ) , annot = True , fmt = '0.0f' , cmap = 'PuBu' ) );

print ( "The performance parameters of the logistic regression model : \n" )

display ( DT_Performance_Metrics )
```

```
<AxesSubplot:>
The performance parameters of the logistic regression model :
```

| | Values (%) |
|---|---|
| Accuracy | 89.47 |
| Precision | 87.02 |
| Recall | 93.55 |
| F1_Score | 90.17 |
| FPR | 0.15 |




### RANDOM FOREST

#### HYPER PARAMETER TUNING

In [25]:
```
RF = RandomForestClassifier ( )

Parameters = {

    'max_depth': [None, 10, 20],

    'min_samples_split': [2, 5, 10]
}

GS = GridSearchCV ( estimator = RF , param_grid = Parameters , cv = 5 )

GS.fit ( Train_X, Train_Y )

print("Best Hyperparameters : ", GS.best_params_ )

Best_Model = GS.best_estimator_

Validation_Accuracy = Best_Model.score ( Test_X, Test_Y )

print( "\nValidation Set Accuracy with Best Model : ", Validation_Accuracy)
```

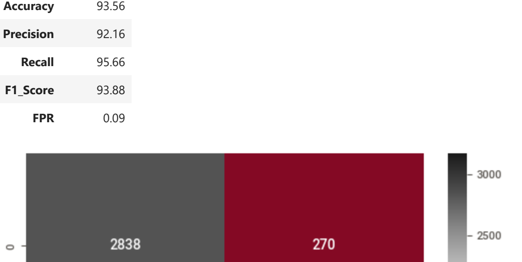
```
Best Hyperparameters :  {'max_depth': None, 'min_samples_split': 2}

Validation Set Accuracy with Best Model :  0.9358954411078264
```

#### PREDICTION

In [26]:
```
RF = RandomForestClassifier ( **GS.best_params_ )

RF.fit ( Train_X , Train_Y )

Y_RF = RF.predict ( Test_X )

Accuracy_RF = accuracy_score ( Test_Y , Y_RF ) * 100

Precision_RF = precision_score ( Test_Y , Y_RF ) * 100

Recall_RF = recall_score ( Test_Y , Y_RF ) * 100

F1_Score_RF = f1_score ( Test_Y , Y_RF ) * 100

Confusion_RF = confusion_matrix ( Test_Y , Y_RF )

FPR_RF = Confusion_RF[0][1] / ( Confusion_RF [0][0] + Confusion_RF [0][1] )

RF_Performance_Metrics = { 'Accuracy ' : Accuracy_RF , 'Precision ' : Precision_RF , 'Recall ' : Recall_RF ,

                           'F1_Score ' : F1_Score_RF , 'FPR ' : FPR_RF }

RF_Performance_Metrics = pd.DataFrame ( np.round ( list(RF_Performance_Metrics.values()), 2 ) , index = list(RF

display ( sns.heatmap ( pd.DataFrame ( Confusion_RF ) , annot = True , fmt = '0.0f' , cmap = 'RdGy' ) );

print ( "The performance parameters of the logistic regression model : \n" )

display ( RF_Performance_Metrics )
```

```
<AxesSubplot:>
The performance parameters of the logistic regression model :
```

| | Values (%) |
|---|---|
| Accuracy | 93.56 |
| Precision | 92.16 |
| Recall | 95.66 |
| F1_Score | 93.88 |
| FPR | 0.09 |




# ENSEMBLE TECHNIQUES : BOOSTING

## GRADIENT BOOSTING

### HYPER PARAMETER TUNING

In [27]:
```
GB = GradientBoostingClassifier( )

Parameters = {

    'learning_rate': [0.1 , 0.01 ],

    'n_estimators': [ 150 , 200 ],

    'max_depth': [ 7 , 9 ]

}

GS = GridSearchCV ( estimator = GB , param_grid = Parameters , cv = 5 )

GS.fit ( Train_X, Train_Y )

print("Best Hyperparameters : ", GS.best_params_ )

Best_Model = GS.best_estimator_

Validation_Accuracy = Best_Model.score ( Test_X, Test_Y )

print( "\nValidation Set Accuracy with Best Model : ", Validation_Accuracy)
```

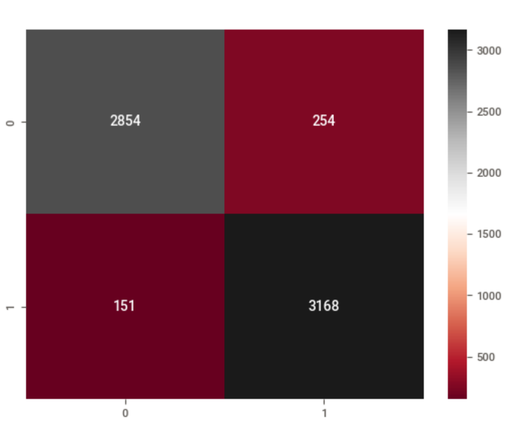
```
Best Hyperparameters :  {'learning_rate': 0.1, 'max_depth': 9, 'n_estimators': 200}

Validation Set Accuracy with Best Model :  0.9349618795705616
```

### PREDICTION

In [28]:
```
GB = GradientBoostingClassifier ( **GS.best_params_ )

GB.fit ( Train_X , Train_Y )

Y_GB = GB.predict ( Test_X )

Accuracy_GB = accuracy_score ( Test_Y , Y_GB ) * 100

Precision_GB = precision_score ( Test_Y , Y_GB ) * 100

Recall_GB = recall_score ( Test_Y , Y_GB ) * 100

F1_Score_GB = f1_score ( Test_Y , Y_GB ) * 100

Confusion_GB = confusion_matrix ( Test_Y , Y_GB )

FPR_GB = Confusion_GB[0][1] / ( Confusion_GB [0][0] + Confusion_GB [0][1] )

GB_Performance_Metrics = { 'Accuracy ' : Accuracy_GB , 'Precision ' : Precision_GB , 'Recall ' : Recall_GB ,

                           'F1_Score ' : F1_Score_GB , 'FPR ' : FPR_GB }

GB_Performance_Metrics = pd.DataFrame ( np.round ( list(GB_Performance_Metrics.values()), 2 ) , index = list(GB

display ( sns.heatmap ( pd.DataFrame ( Confusion_GB ) , annot = True , fmt = '0.0f' , cmap = 'RdGy' ) );

print ( "The performance parameters of the logistic regression model : \n" )

display ( GB_Performance_Metrics )
```

```
<AxesSubplot:>
The performance parameters of the logistic regression model :
```

| | Values (%) |
|---|---|
| Accuracy | 93.70 |
| Precision | 92.58 |
| Recall | 95.45 |
| F1_Score | 93.99 |
| FPR | 0.08 |




## EXTREME GRADIENT BOOSTING

### HYPER PARAMETER TUNING

In [29]:
```
XGB = XGBClassifier ( )

Parameters = {
    'learning_rate': [0.1, 0.01],

    'max_depth': [3, 5, 7],

    'n_estimators': [100, 200, 300],

    'gamma': [0, 0.1, 0.2],
}

GS = GridSearchCV ( estimator = XGB , param_grid = Parameters , cv = 5 )

GS.fit ( Train_X, Train_Y )

print("Best Hyperparameters : ", GS.best_params_ )

Best_Model = GS.best_estimator_

Validation_Accuracy = Best_Model.score ( Test_X, Test_Y )

print( "\nValidation Set Accuracy with Best Model : ", Validation_Accuracy)
```

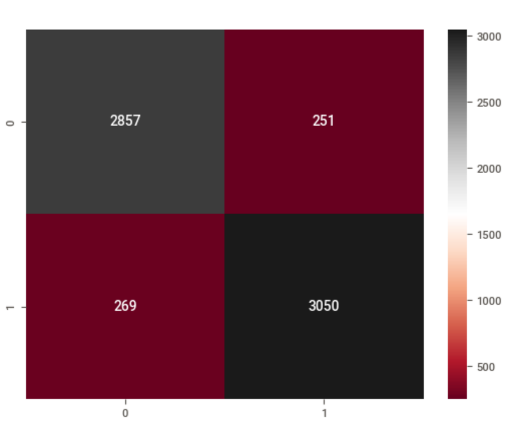
```
Best Hyperparameters :  {'gamma': 0.2, 'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 300}

Validation Set Accuracy with Best Model :  0.9190913334370624
```

### PREDICTION

In [30]:
```
XGB = XGBClassifier ( **GS.best_params_ )

XGB.fit ( Train_X , Train_Y )

Y_XGB = XGB.predict ( Test_X )

Accuracy_XGB = accuracy_score ( Test_Y , Y_XGB ) * 100

Precision_XGB = precision_score ( Test_Y , Y_XGB ) * 100

Recall_XGB = recall_score ( Test_Y , Y_XGB ) * 100

F1_Score_XGB = f1_score ( Test_Y , Y_XGB ) * 100

Confusion_XGB = confusion_matrix ( Test_Y , Y_XGB )

FPR_XGB = Confusion_XGB[0][1] / ( Confusion_XGB [0][0] + Confusion_XGB [0][1] )

XGB_Performance_Metrics = { 'Accuracy ' : Accuracy_XGB , 'Precision ' : Precision_XGB , 'Recall ' : Recall_XGB

                            'F1_Score ' : F1_Score_XGB , 'FPR ' : FPR_XGB }

XGB_Performance_Metrics = pd.DataFrame ( np.round ( list(XGB_Performance_Metrics.values()), 2 ) , index = list(

display ( sns.heatmap ( pd.DataFrame ( Confusion_XGB ) , annot = True , fmt = '0.0f' , cmap = 'RdGy' ) );

print ( "The performance parameters of the logistic regression model : \n" )

display ( XGB_Performance_Metrics )
```

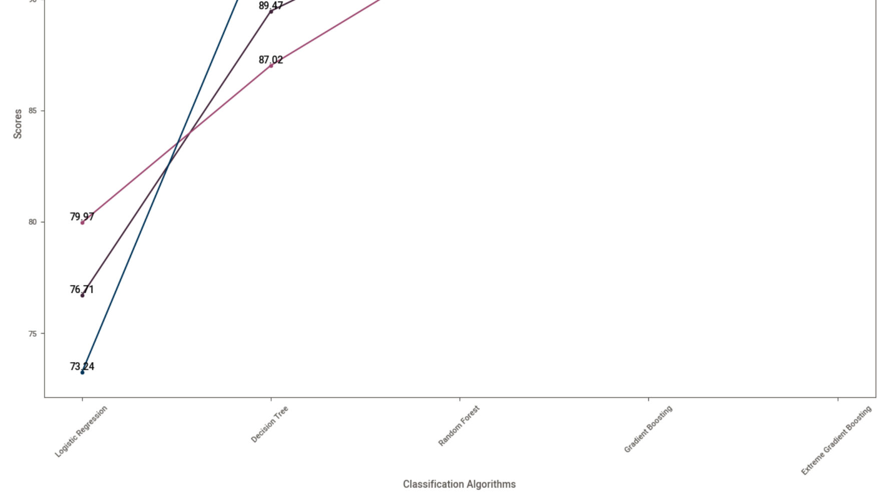
```
<AxesSubplot:>
The performance parameters of the logistic regression model :
```

| | Values (%) |
|---|---|
| Accuracy | 91.91 |
| Precision | 92.40 |
| Recall | 91.90 |
| F1_Score | 92.15 |
| FPR | 0.08 |




# COMPARING THE MODELS

In [33]:
```
# Classification algorithm names

algorithm_names = ['Logistic Regression', 'Decision Tree', 'Random Forest', 'Gradient Boosting' , 'Extreme Grad

# Accuracy scores

accuracy_scores = [ Accuracy_LR , Accuracy_DT , Accuracy_RF , Accuracy_GB , Accuracy_XGB ]

# Precision scores

precision_scores = [ Precision_LR , Precision_DT , Precision_RF , Precision_GB , Precision_XGB ]

# Recall scores

recall_scores = [ Recall_LR , Recall_DT , Recall_RF , Recall_GB , Recall_XGB ]

# Plotting the scores

plt.figure( figsize = ( 15 , 10 ) )

# Accuracy scores plot

plt.plot(algorithm_names, accuracy_scores, label = 'Accuracy', marker = 'o', color = '#42253B')

for i in range(len(algorithm_names)):

    plt.text(algorithm_names[i], accuracy_scores[i], f'{accuracy_scores[i]:.2f}', ha ='center', va = 'bottom')

# Precision scores plot

plt.plot( algorithm_names, precision_scores, label='Precision', marker='o', color = '#9E4770')

for i in range ( len( algorithm_names ) ):

    plt.text( algorithm_names [ i ] , precision_scores[ i ], f'{precision_scores[i]:.2f}', ha='center', va='bot

# Recall scores plot

plt.plot(algorithm_names, recall_scores, label='Recall', marker='o', color='#003459')

for i in range(len(algorithm_names)):

    plt.text(algorithm_names[i], recall_scores[i], f'{recall_scores[i]:.2f}', ha='center', va='bottom')



# Labeling the axes and title

plt.xlabel( 'Classification Algorithms' )

plt.ylabel( 'Scores' )

plt.title( 'Performance Comparison' )

# Adding a legend

plt.legend()

# Rotating the x-axis labels for better visibility

plt.xticks( rotation = 45 )

# Displaying the plot

plt.show ( )
```




In [34]:
```
Performance = pd.DataFrame ( index = algorithm_names )

Performance [ "Accuracy" ] = accuracy_scores

Performance [ "Precision" ] = precision_scores

Performance [ "Recall" ] = recall_scores

Performance
```

Out[34]:

| | Accuracy | Precision | Recall |
|---|---|---|---|
| Logistic Regression | 76.707640 | 79.967105 | 73.244953 |
| Decision Tree | 89.466314 | 87.023543 | 93.552275 |
| Random Forest | 93.558425 | 92.162554 | 95.661344 |
| Gradient Boosting | 93.698460 | 92.577440 | 95.450437 |
| Extreme Gradient Boosting | 91.909133 | 92.396244 | 91.895149 |