14.01.2024 | @pathakdarshan12
🚢 Titanic - Machine Learning from Disaster: A Survival Prediction using Various Machine Learning

1. | Introduction 👋

🎳 Probelm Statement

The sinking of the Titanic is one of the most infamous shipwrecks in history.On **April 15, 1912**, during her maiden voyage, the widely considered "unsinkable" RMS Titanic sank after colliding with an iceberg. Unfortunately, there weren't enough lifeboats for everyone onboard, resulting in the death of **1502 out of 2224** passengers and crew. While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others.In this notebook, we are going to build a **predictive model** that answers the question: "what sorts of people were more likely to survive?"using passenger data (ie name, age, gender, socio-economic class, etc).

🤔 Dataset Problems

This dataset is taken from the **Kaggle Website**. This dataset contains **passenger data (ie name, age, gender, socio-economic class, etc)** of whether the passenger has survived. With the help of Machine learning models based on the passenger information provided in the dataset. The **variables that most influence** a survival chances will also be **explored more deeply** in this notebook.

📌 Notebook Objectives

This notebook **aims** to:

- Perform dataset exploration using various types of data visualization.
- Build machine learning model that can predict survial.
- Export prediction result on test data into files.
- Save/dump the complete machine learing pipeline for later usage.
- Perform prediction on new example data given and export the prediction result.

🧑‍💻 Machine Learning Model

The **models** used in this notebook:

1. **Logistic Regression**,
2. **K-Nearest Neighbour (KNN)**,
3. **Support Vector Machine (SVM)**,
4. **Gaussian Naive Bayes**,
5. **Decision Tree**,
6. **Random Forest**,
7. **Extra Tree Classifier**,
8. **Gradient Boosting**, and
9. **AdaBoost**.

2. | Installing and Importing Libraries 📚

Installing and Importing libraries that will be used in this notebook.

In [1]:
```
## --- Installing Libraries ---
#!pip install ydata-profiling
#!pip install highlight-text
#!pip install Pillow
```

In [2]:
```
# --- Importing Libraries ---
from IPython.display import display, HTML, Javascript
import numpy as np
import pandas as pd
import ydata_profiling
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline
import seaborn as sns
import warnings
import os
import yellowbrick
import joblib
```

```
from ydata_profiling import ProfileReport
#from pywaffle import Waffle
from statsmodels.graphics.gofplots import qqplot
from PIL import Image
from highlight_text import fig_text
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import RobustScaler, OneHotEncoder
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, Ad
from sklearn.metrics import classification_report, accuracy_score
from yellowbrick.classifier import PrecisionRecallCurve, ROCAUC, ConfusionMatrix
from yellowbrick.model_selection import LearningCurve, FeatureImportances
from yellowbrick.contrib.wrapper import wrap
from yellowbrick.style import set_palette
warnings.filterwarnings("ignore")
```

3. | Reading Dataset 👓

After importing libraries, **the dataset that will be used will be imported**.

In [3]:
```
# --- Importing Dataset ---
df = pd.read_csv("titanic.csv")

# --- Reading Train Dataset ---
class Color:
    # Define color codes
    start = '\033[91m'
    end = '\033[0m'
    color = '\033[94m'

# Create an instance of the Color class
clr = Color()

# Reading Train Dataset
print(clr.start + '.: Imported Dataset :.' + clr.end)
print(clr.color + '*' * 23)
styled_df = df.head(10).reset_index(drop=True).style.background_gradient(cmap='Blue
styled_df
```

```
.: Imported Dataset :.
***********************
```

Out[3]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	3	Braund, Mr. Owen Harris	male	22.000000	1	0	A/5 21171	7.250000
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38.000000	1	0	PC 17599	71.283300
2	3	3	Heikkinen, Miss. Laina	female	26.000000	0	0	STON/O2. 3101282	7.925000
3	4	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000000	1	0	113803	53.100000
4	5	3	Allen, Mr. William Henry	male	35.000000	0	0	373450	8.050000
5	6	3	Moran, Mr. James	male	nan	0	0	330877	8.458300
6	7	1	McCarthy, Mr. Timothy J	male	54.000000	0	0	17463	51.862500
7	8	3	Palsson, Master. Gosta Leonard	male	2.000000	3	1	349909	21.075000
8	9	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.000000	0	2	347742	11.133300
9	10	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.000000	1	0	237736	30.070800

Dataset Description

The following is the **structure of the dataset**.

Variable Name	Description	Sample Data
PassengerId	Passenger's ID	3; 7; ...
pclass	Ticket Class	1 = 1st, 2 = 2nd, 3 = 3rd
Name	Name of the Passenger	Braund, Mr. Owen Harris
Sex	Gender of Passenger	0 = male 1 = female
Age	Passengers age (in years)	63; 37; ...
SibSp	No. of siblings / spouses aboard the Titanic	3; 1; 2; ...
Parch	No. of parents / children aboard the Titanic	5; 3; ...
Ticket	Ticket number	2649; 349909; ...
Fare	Passenger fare	31.3875; 263; ...
Cabin	Cabin number	C85;E46;; ...

/div>

4. | Initial Dataset Exploration 🔍

This section will focused on **initial data exploration on the dataset** with Pandas Profiling before pre-processing performed. In addition, **variables correlation** will be examined as well.

In [4]:
```
# --- Dataset Report ---
ProfileReport(df,
               title="Titanic Survival Prediction",
               minimal=True,
               progress_bar=False,
               samples=None,
               interactions=None,
               explorative=True,
               dark_mode=True,
               notebook={'iframe': {'height': '600px'},
                         'html': {'style': {'primary_color': clr}},
                         'missing_diagrams': {'heatmap': False, 'dendrogram': False
              ).to_notebook_iframe()
```

Overview

Dataset statistics

Number of variables	12
Number of observations	891
Missing cells	866
Missing cells (%)	8.1%
Total size in memory	315.0 KiB
Average record size in memory	362.1 B

Variable types

Numeric	7
Text	5

4. | Data Preprocessing 🔍

Since **Name** and **Ticket** columns are not directly related to the survival we can can drop them.

```
In [5]: df = df.drop(["Name","Ticket"],axis=1)
```

In the **Cabin** columns there are **687 (77.1%) missing values** hence removing the cabin column

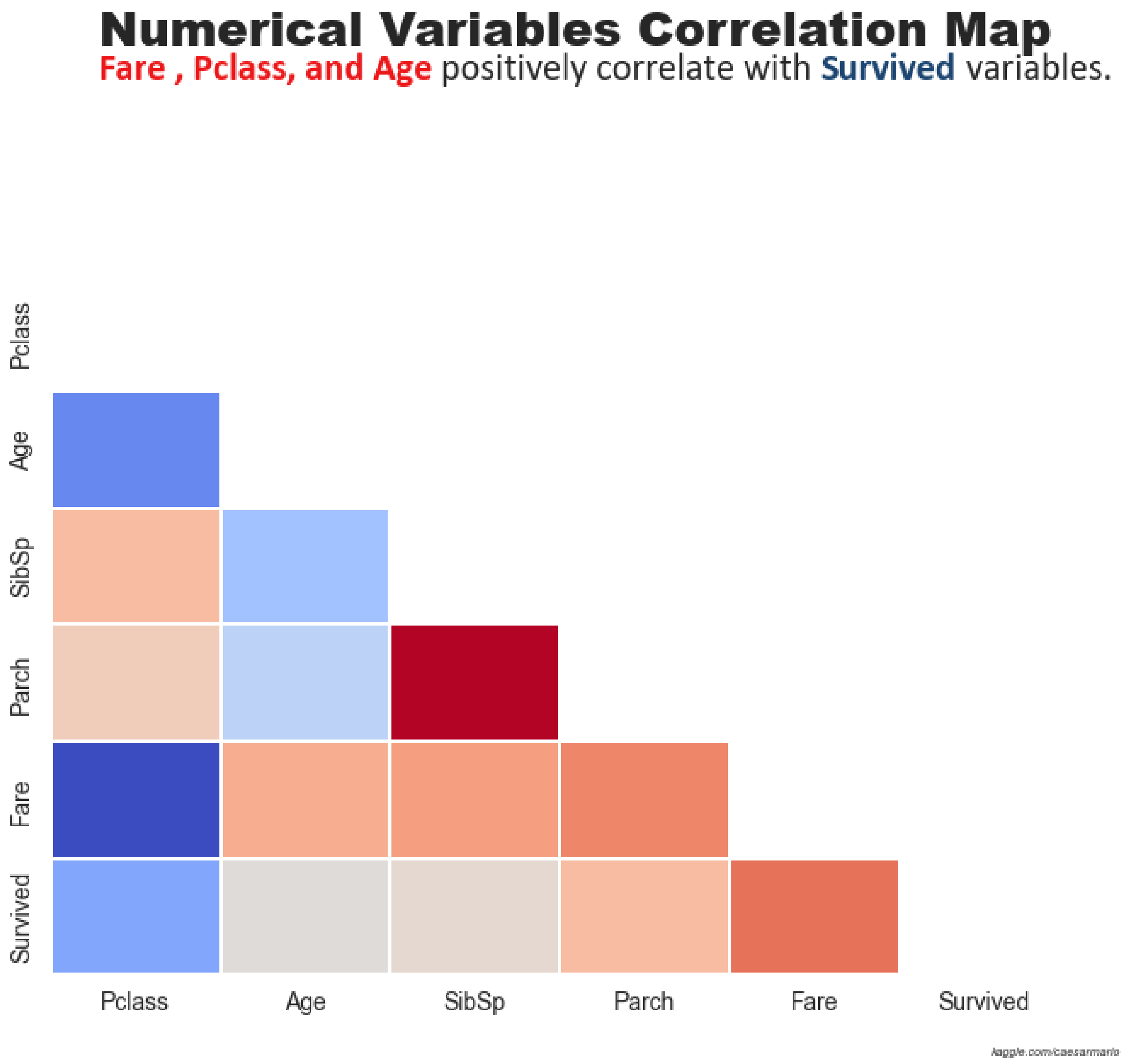
```
In [6]: df = df.drop("Cabin",axis=1)
```

```
In [7]: df["Embarked"] = df["Embarked"].fillna(df["Embarked"].mode()[0])
In [8]: df['Age'] = df.groupby(['Pclass', 'Sex'])['Age'].transform(lambda x: x.fillna(x.med
```

```
In [9]: df = df.drop(["PassengerId"],axis=1)
```

```
In [10]: corr = df.corr(numeric_only=True)
```

```
In [12]: mask = np.triu(np.ones_like(corr, dtype=bool))
         fig, ax = plt.subplots(figsize=(7, 6))
         sns.heatmap(df.corr(numeric_only=True), mask=mask, annot=True, cmap='coolwarm', lin
         yticks, ylabels = plt.yticks()
         xticks, xlabels = plt.xticks()
         ax.set_xticklabels(xlabels, rotation=0, **xy_label, fontsize=10)
         ax.set_yticklabels(ylabels, **xy_label, fontsize=10)
         ax.grid(False)

         # Assuming 'fig_text' is a custom function for adding text to the figure
         # Make sure it's defined properly in your code
         fig_text(s='Numerical Variables Correlation Map', **suptitle)
         fig_text(s='<Fare , Pclass, and Age> positively correlate with <Survived> variables
         plt.tight_layout(rect=[0, 0.04, 1, 1.01])
         plt.gcf().text(0.85, 0.03, 'kaggle.com/caesarmario', style='italic', fontsize=5)
         plt.show()
```



From **dataset report** and **correlation matrix**, it can be **concluded** that:

> - There are no missing values detected in the dataset. In addition, it also can be seen that **the number of categorical columns is more than the numerical columns**.
> - As can be seen from the profiling report, the number of male patients is greater than female patients. In addition, **chest pain type 0 (typical angina) is higher than other types**. Most of the patients in the dataset had fasting blood sugar that was less than 120 mg/dl. **The number of resting electrocardiographic types 1 (having ST-T wave abnormality) and 0 (normal) is more than type 2 (definite left ventricular hypertrophy)**. Moreover, patients who don't have exercise-induced angina have a higher number. **The number patients with flat and downsloping slopes is more** than upsloping slope.
> - Furthermore, patients with 0 major vessels are more numerous than those with major vessels. **Patients with fixed defect thalassemia have the**

highest distribution compared to others. The total number of patients with heart disease is higher than those without heart disease.

- Age, resting blood pressure, cholestoral, and max. heart received columns are lack of variation since it has **low standard deviation**.
- The age column has a normal distribution based on the histogram and skewness value. However, **the resting blood pressure column has a moderately right-skewed distribution and the serum cholestoral and oldpeak columns have a highly right-skewed distribution**. On the other hand, the max. heart received column has a moderate left-skewed distribution. Since some columns are moderate to highly left or right-skewed, **some outliers are detected at the distribution tail**.
- The age, resting blood pressure, max. heart received and oldpeak columns have a kurtosis value of less than 3, which indicates that the column is platikurtic. Meanwhile, **the serum cholestoral column** has a kurtosis value of more than 3, which indicates that the column is **leptokurtic**.

> 📌 Low standard deviation means data are clustered around the mean (lack of variation), and high standard deviation indicates data are more spread out (more variation).

> 📌 If skewness is **less than -1 or greater than 1**, the distribution is highly skewed. If skewness is **between -1 and -0.5 or between 0.5 and 1**, the distribution is moderately skewed. If skewness is **between -0.5 and 0.5**, the distribution is approximately symmetric.

> 📌 Kurtosis values used to show tailedness of a column. The value of normal distribution (mesokurtotic) should be equal to 3. If kurtosis value is more than 3, it is called leptokurtic. Meanwhile, if kurtosis value is less than 3, then it is called platikurtic.

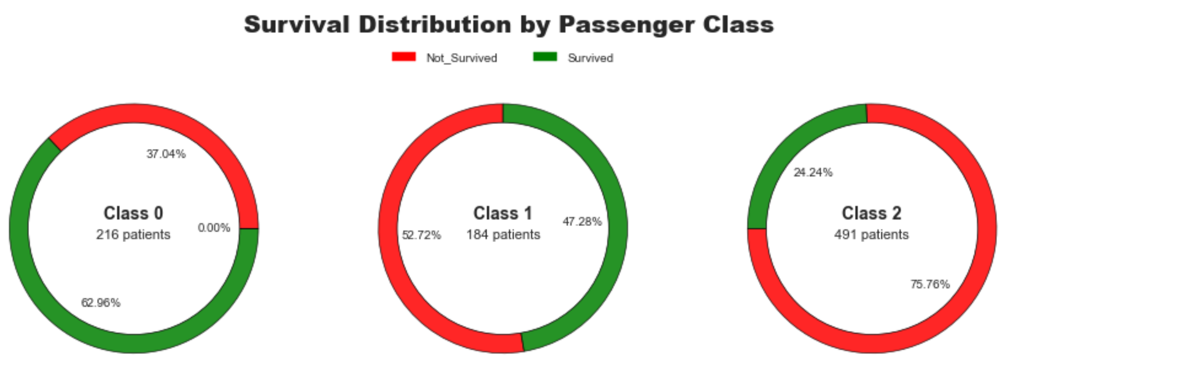
- **The mean age of the patients in the dataset was 54.36 years old**, with the most senior patient being 77 years old and the youngest being 29 years old. **The average resting blood pressure in the dataset is 131.62**, where the highest resting blood pressure is 200, and the minimum is 94 (generally, **the ideal blood pressure ranges from 90 to 120**).
- **The mean serum cholesteral was 246.26**, with a maximum of 564 and a minimum of 126. In addition, **the patient's average max. heart rate in the dataset was 149.64**, with a minimum of 71 and a maximum of 202. **The patient's mean oldpeak was 1.03**, with a minimum of 0 and a maximum of 6.2.
- According to the correlation between variables, it can be seen that **chest pain type, max. heart rate, and slope have a high positive correlation**

> **with the target variable**. However, **exang, oldpeak, and thalassemia negatively correlate with the target variable**.

5. | EDA 📈

This section will perform some **EDA** to get more insights about dataset.

In [13]:
```python
# --- EDA 3 Dataframes ---
df_eda3 = df[['Pclass', 'Survived']]
df_eda3 = pd.DataFrame(df_eda3.groupby(['Pclass', 'Survived']).size().reset_index(n
df_eda3.loc[len(df_eda3.index)] = [1, 2, 0]
df_eda3_0 = df_eda3.query('Pclass == 1').drop('Pclass', axis=1)
df_eda3_1 = df_eda3.query('Pclass == 2').drop('Pclass', axis=1)
df_eda3_2 = df_eda3.query('Pclass == 3').drop('Pclass', axis=1)

# --- EDA 3 Variables ---
total_list = [df_eda3_0['total'], df_eda3_1['total'], df_eda3_2['total']]
suptitle = dict(x=0.5, y=0.94, fontsize=14, weight='heavy', ha='center', va='center
exp_text = dict(x=0.5, y=-0.08, fontsize=8, weight='normal', ha='center', va='cente
highlight_explanation = [{'weight': 'bold', 'color': 'red'}, {'weight': 'bold', 'co
survived_0 = mpatches.Patch(color='red', label='Not_Survived')
survived_1 = mpatches.Patch(color='green', label='Survived')

def my_autopct(pct):
    return f'{pct:.2f}%' if pct != 0 else ''

# --- EDA 3 Functions ---
def display_eda3(subplot_num, pclass_type, total, colors, start_angle):
    centre = plt.Circle((0, 0), 0.85, fc='white', edgecolor='black', linewidth=0.5)
    total_passengers = total.sum()

    plt.subplot(1, 3, subplot_num)
    plt.tight_layout(rect=[0, 0, 1, 1.01])
    plt.pie(total, colors=colors, autopct='%.2f%%', pctdistance=0.65, startangle=st
    plt.text(0, 0.08, f"Class {pclass_type}", weight='bold', ha='center', fontsize=
    plt.text(0, -0.08, f"{total_passengers} patients", ha='center', fontsize=8)
    fig = plt.gcf()
    fig.gca().add_artist(centre)

# --- Display EDA 3 ---
plt.figure(figsize=(9, 4))
for idx, total in enumerate(total_list):
    display_eda3(idx + 1, idx, total, ['red', 'green', 'blue'], 90 * idx)
    if idx == 1:
        plt.legend(handles=[survived_0, survived_1], loc='upper center', bbox_to_an

plt.suptitle("Survival Distribution by Passenger Class", **suptitle)
plt.title("Explanation: Pclass 3 has the highest survival rate, Class 2 is moderate
plt.show()
```



Explanation: Pclass 3 has the highest survival rate, Class 2 is moderate, and Class 1 has the lowest.

6. | Data Preprocessing ⚙️

This section will **prepare the dataset** before building the machine learning models.

6.1 | Features Separating and Splitting 🪓

In this section, the 'Survived' (dependent) column will be seperated from independent columns. Also, the dataset will be splitted into 90:10 ratio (90% training and 10% testing).

In [14]:
```
# --- Seperating Dependent Features ---
x = df.drop(['Survived'], axis=1)
y = df['Survived']

# --- Splitting Dataset ---
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1, random_sta
```

6.2 | Processing Pipeline 🪠

This section will create a preprocessing pipeline for numerical and categorical columns and apply them to the `x_train` and `x_test` data. Not all columns will go through preprocessing. For all numerical columns, scaling will be carried out using a robust scaler since the dataset used is a **small dataset** where the presence of outliers dramatically affects the performance of a model. While for categorical columns with more than two categories, one-hot encoding will be carried out.

In [15]:
```
# --- Numerical Pipeline ---
num_column = ['Age', 'Fare']
num_pipeline = Pipeline([
    ('scaling', RobustScaler())
])

# --- Categorical Pipeline ---
cat_column = ['Pclass', 'Sex', 'Embarked']
cat_pipeline = Pipeline([
    ('onehot', OneHotEncoder(drop='first', sparse=False))
])

# --- Combine Both Pipelines into Transformer ---
preprocessor = ColumnTransformer([
    ('categorical', cat_pipeline, cat_column)
    , ('numerical', num_pipeline, num_column)]
    , remainder='passthrough')

# --- Apply Transformer to Pipeline ---
process_pipeline = Pipeline([
    ('preprocessor', preprocessor)
])

# --- Apply to Dataframe ---
x_train_process = process_pipeline.fit_transform(x_train)
x_test_process = process_pipeline.fit_transform(x_test)
```

7. | Machine Learning Model Implementation 🛠️

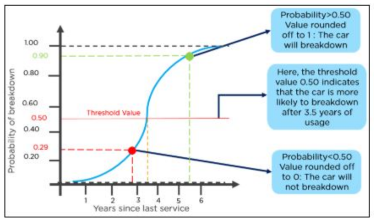
This section will **implement various machine learning models** as mentioned in Introduction section. In addition, explanation for each models also will be discussed.

In [16]:
```
# --- Functions: Model Fitting and Performance Evaluation ---
color_yb = sns.color_palette("Paired")
color_line = 'red'
color = 'red'
def fit_ml_models(algo, algo_param, algo_name):

    # --- Algorithm Pipeline ---
    algo = Pipeline([('algo', algo)])

    # --- Apply Grid Search ---
    model = GridSearchCV(algo, param_grid=algo_param, cv=10, n_jobs=-1, verbose=1)

    # --- Fitting Model ---
    print(clr.start+f".:. Fitting {algo_name} .:."+clr.end)
    fit_model = model.fit(x_train_process, y_train)

    # --- Model Best Parameters ---
    best_params = model.best_params_
    print("\n>> Best Parameters: "+clr.start+f"{best_params}"+clr.end)

    # --- Best & Final Estimators ---
    best_model = model.best_estimator_
    best_estimator = model.best_estimator_._final_estimator
    best_score = round(model.best_score_, 4)
    print(">> Best Score: "+clr.start+"{:.3f}".format(best_score)+clr.end)

    # --- Create Prediction for Train & Test ---
    y_pred_train = model.predict(x_train_process)
    y_pred_test = model.predict(x_test_process)
```

```
# --- Train & Test Accuracy Score ---
acc_score_train = round(accuracy_score(y_pred_train, y_train)*100, 3)
acc_score_test = round(accuracy_score(y_pred_test, y_test)*100, 3)
print("\n"+clr.start+f".:. Train and Test Accuracy Score for {algo_name} .:."+c
print("\t>> Train Accuracy: "+clr.start+"{:.2f}%".format(acc_score_train)+clr.e
print("\t>> Test Accuracy: "+clr.start+"{:.2f}%".format(acc_score_test)+clr.end

# --- Classification Report ---
print("\n"+clr.start+f".:. Classification Report for {algo_name} .:."+clr.end)
print(classification_report(y_test, y_pred_test))

# --- Figures Settings ---
xy_label = dict(fontweight='bold', fontsize=12)
grid_style = dict(color=color, linestyle='dotted', zorder=1)
title_style = dict(fontsize=14, fontweight='bold')
tick_params = dict(length=3, width=1, color='red')
bar_style = dict(zorder=3, edgecolor='black', linewidth=0.5, alpha=0.85)
set_palette(color_yb)
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(16, 14))

# --- Confusion Matrix ---
conf_matrix = ConfusionMatrix(best_estimator, ax=ax1, cmap='Reds')
conf_matrix.fit(x_train_process, y_train)
conf_matrix.score(x_test_process, y_test)
conf_matrix.finalize()
conf_matrix.ax.set_title('Confusion Matrix\n', **title_style)
conf_matrix.ax.tick_params(axis='both', labelsize=10, bottom='on', left='on', *
for spine in conf_matrix.ax.spines.values(): spine.set_color(color_line)
conf_matrix.ax.set_xlabel('\nPredicted Class', **xy_label)
conf_matrix.ax.set_ylabel('True Class\n', **xy_label)
conf_matrix.ax.xaxis.set_ticklabels(['False', 'True'], rotation=0)
conf_matrix.ax.yaxis.set_ticklabels(['True', 'False'])

# --- ROC AUC ---
logrocauc = ROCAUC(best_estimator, classes=['False', 'True'], ax=ax2, colors=co
logrocauc.fit(x_train_process, y_train)
logrocauc.score(x_test_process, y_test)
logrocauc.finalize()
logrocauc.ax.set_title('ROC AUC Curve\n', **title_style)
logrocauc.ax.tick_params(axis='both', labelsize=10, bottom='on', left='on', **t
logrocauc.ax.grid(axis='both', alpha=0.4, **grid_style)
for spine in logrocauc.ax.spines.values(): spine.set_color('None')
for spine in ['bottom', 'left']:
    logrocauc.ax.spines[spine].set_visible(True)
    logrocauc.ax.spines[spine].set_color(color_line)
logrocauc.ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.12), ncol=2, bo
logrocauc.ax.set_xlabel('\nFalse Positive Rate', **xy_label)
logrocauc.ax.set_ylabel('True Positive Rate\n', **xy_label)

# --- Learning Curve ---
lcurve = LearningCurve(best_estimator, scoring='f1_weighted', ax=ax3, colors=co
lcurve.fit(x_train_process, y_train)
lcurve.finalize()
lcurve.ax.set_title('Learning Curve\n', **title_style)
lcurve.ax.tick_params(axis='both', labelsize=10, bottom='on', left='on', **tick
    lcurve.ax.grid(axis='both', alpha=0.4, **grid_style)
    for spine in lcurve.ax.spines.values(): spine.set_color('None')
    for spine in ['bottom', 'left']:
        lcurve.ax.spines[spine].set_visible(True)
        lcurve.ax.spines[spine].set_color(color_line)
    lcurve.ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.12), ncol=2, borde
    lcurve.ax.set_xlabel('\nTraining Instances', **xy_label)
    lcurve.ax.set_ylabel('Scores\n', **xy_label)

    # --- Feature Importance or Precision Recall Curve ---
    try:
        feat_importance = FeatureImportances(best_estimator, labels=columns_list_on
        feat_importance.fit(x_train_process, y_train)
        feat_importance.finalize()
        feat_importance.ax.set_title('Feature Importances (Top 5 Features)\n', **ti
        feat_importance.ax.tick_params(axis='both', labelsize=10, bottom='on', left
        feat_importance.ax.grid(axis='x', alpha=0.4, **grid_style)
        feat_importance.ax.grid(axis='y', alpha=0, **grid_style)
        for spine in feat_importance.ax.spines.values(): spine.set_color('None')
        for spine in ['bottom']:
            feat_importance.ax.spines[spine].set_visible(True)
            feat_importance.ax.spines[spine].set_color(color_line)
        feat_importance.ax.set_xlabel('\nRelative Importance', **xy_label)
        feat_importance.ax.set_ylabel('Features\n', **xy_label)
    except:
        prec_curve = PrecisionRecallCurve(best_estimator, ax=ax4, ap_score=True, is
        prec_curve.fit(x_train_process, y_train)
        prec_curve.score(x_test_process, y_test)
        prec_curve.finalize()
        prec_curve.ax.set_title('Precision-Recall Curve\n', **title_style)
        prec_curve.ax.tick_params(axis='both', labelsize=10, bottom='on', left='on'
        for spine in prec_curve.ax.spines.values(): spine.set_color('None')
        for spine in ['bottom', 'left']:
            prec_curve.ax.spines[spine].set_visible(True)
            prec_curve.ax.spines[spine].set_color(color_line)
        prec_curve.ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.12), ncol=
        prec_curve.ax.set_xlabel('\nRecall', **xy_label)
        prec_curve.ax.set_ylabel('Precision\n', **xy_label)

    plt.suptitle(f'\n{algo_name} Performance Evaluation Report\n', fontsize=18, fon
    plt.gcf().text(0.88, 0.02, 'kaggle.com/caesarmario', style='italic', fontsize=1
    plt.tight_layout();

    return acc_score_train, acc_score_test, best_score
```

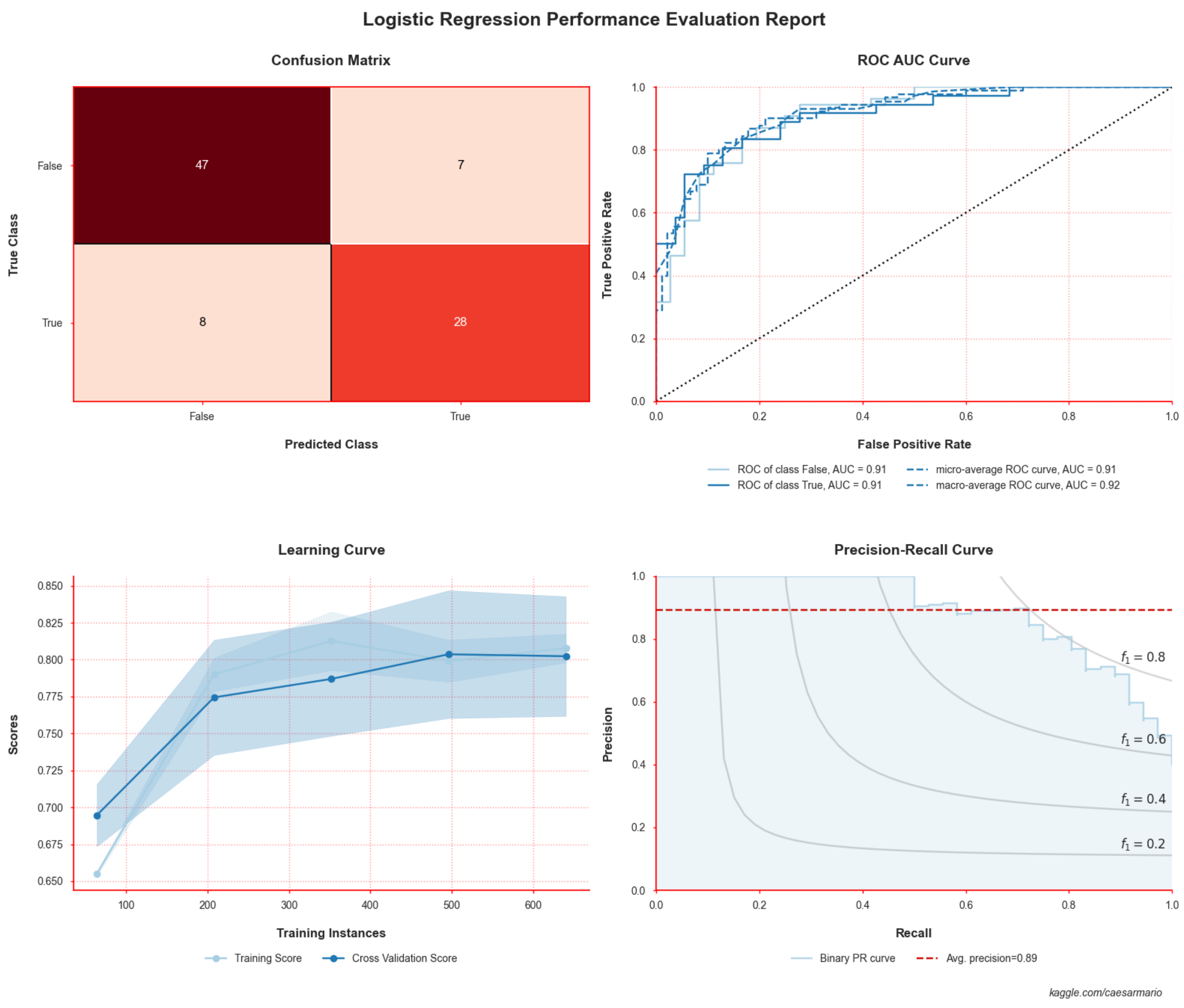
7.1 | Logistic Regression

> **Logistic regression** is a statistical method that is used for building machine learning models where **the dependent variable is dichotomous: i.e. binary**. Logistic regression is used to describe data and **the relationship between one dependent variable and one or more independent variables**. The independent variables can be nominal, ordinal, or of interval type.

> The name "logistic regression" is derived from the concept of the logistic function that it uses. **The logistic function is also known as the sigmoid function**. The value of this logistic function lies between zero and one.
>
> 
>
>
> 🖼️ *Logistic Function by Simplilearn*

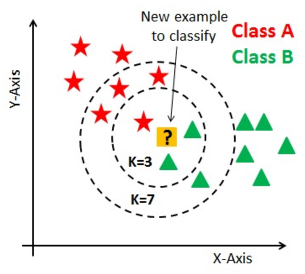
In [17]:
```
# --- Logistic Regression Parameters ---
parameter_lr = {"algo__solver": ["lbfgs", "saga", "newton-cg"]
                , "algo__C": [0.1, 0.2, 0.5, 0.8]}

# --- Logistic Regression Algorithm ---
algo_lr = LogisticRegression(penalty="l2", random_state=42, n_jobs=-1)

# --- Applying Logistic Regression ---
acc_score_train_lr, acc_score_test_lr, best_score_lr = fit_ml_models(algo_lr, param
```

```
.:. Fitting Logistic Regression .:.
Fitting 10 folds for each of 12 candidates, totalling 120 fits

>> Best Parameters: {'algo__C': 0.1, 'algo__solver': 'lbfgs'}
>> Best Score: 0.808

.:. Train and Test Accuracy Score for Logistic Regression .:.
        >> Train Accuracy: 81.15%
        >> Test Accuracy: 83.33%

.:. Classification Report for Logistic Regression .:.
              precision    recall  f1-score   support

           0       0.85      0.87      0.86        54
           1       0.80      0.78      0.79        36

    accuracy                           0.83        90
   macro avg       0.83      0.82      0.83        90
weighted avg       0.83      0.83      0.83        90
```



7.2 | K-Nearest Neighbour (KNN)

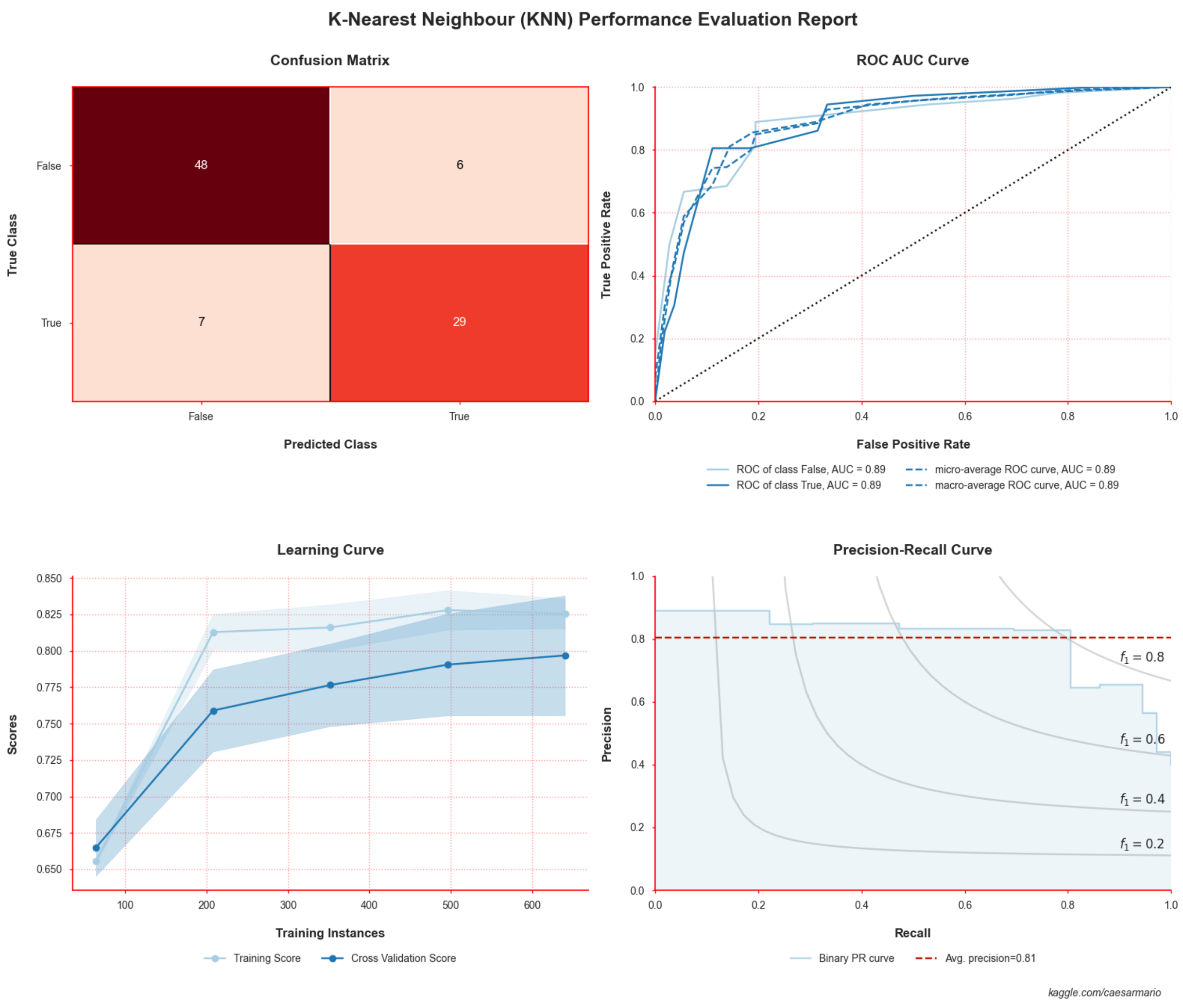
> **The k-nearest neighbors (KNN)** algorithm is a data classification method **for estimating the likelihood that a data point will become a member of one group or another** based on what group the data points nearest to it belong to. The k-nearest neighbor algorithm is a type of supervised machine learning algorithm used **to solve classification and regression problems**.
>
> It's called a **lazy learning algorithm or lazy learner** because it doesn't perform any training when you supply the training data. Instead, it just stores the data during the training time and doesn't perform any calculations. It doesn't build a model until a query is performed on the dataset. This makes KNN ideal for data mining.



🖼️ *KNN by Kita Informatika*

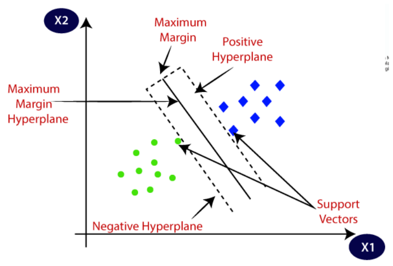
```
In [18]: # --- KNN Parameters ---
         parameter_knn = {"algo__n_neighbors": [2, 5, 10, 17]
                          , "algo__leaf_size": [1, 10, 11, 30]}

         # --- KNN Algorithm ---
         algo_knn = KNeighborsClassifier(n_jobs=-1)

         # --- Applying KNN ---
         acc_score_train_knn, acc_score_test_knn, best_score_knn = fit_ml_models(algo_knn, p
```

```
.:. Fitting K-Nearest Neighbour (KNN) .:.
Fitting 10 folds for each of 16 candidates, totalling 160 fits

>> Best Parameters: {'algo__leaf_size': 10, 'algo__n_neighbors': 10}
>> Best Score: 0.805

.:. Train and Test Accuracy Score for K-Nearest Neighbour (KNN) .:.
        >> Train Accuracy: 82.77%
        >> Test Accuracy: 85.56%

.:. Classification Report for K-Nearest Neighbour (KNN) .:.
              precision    recall  f1-score   support

           0       0.87      0.89      0.88        54
           1       0.83      0.81      0.82        36

    accuracy                           0.86        90
   macro avg       0.85      0.85      0.85        90
weighted avg       0.86      0.86      0.86        90
```



7.3 | Support Vector Machine (SVM)

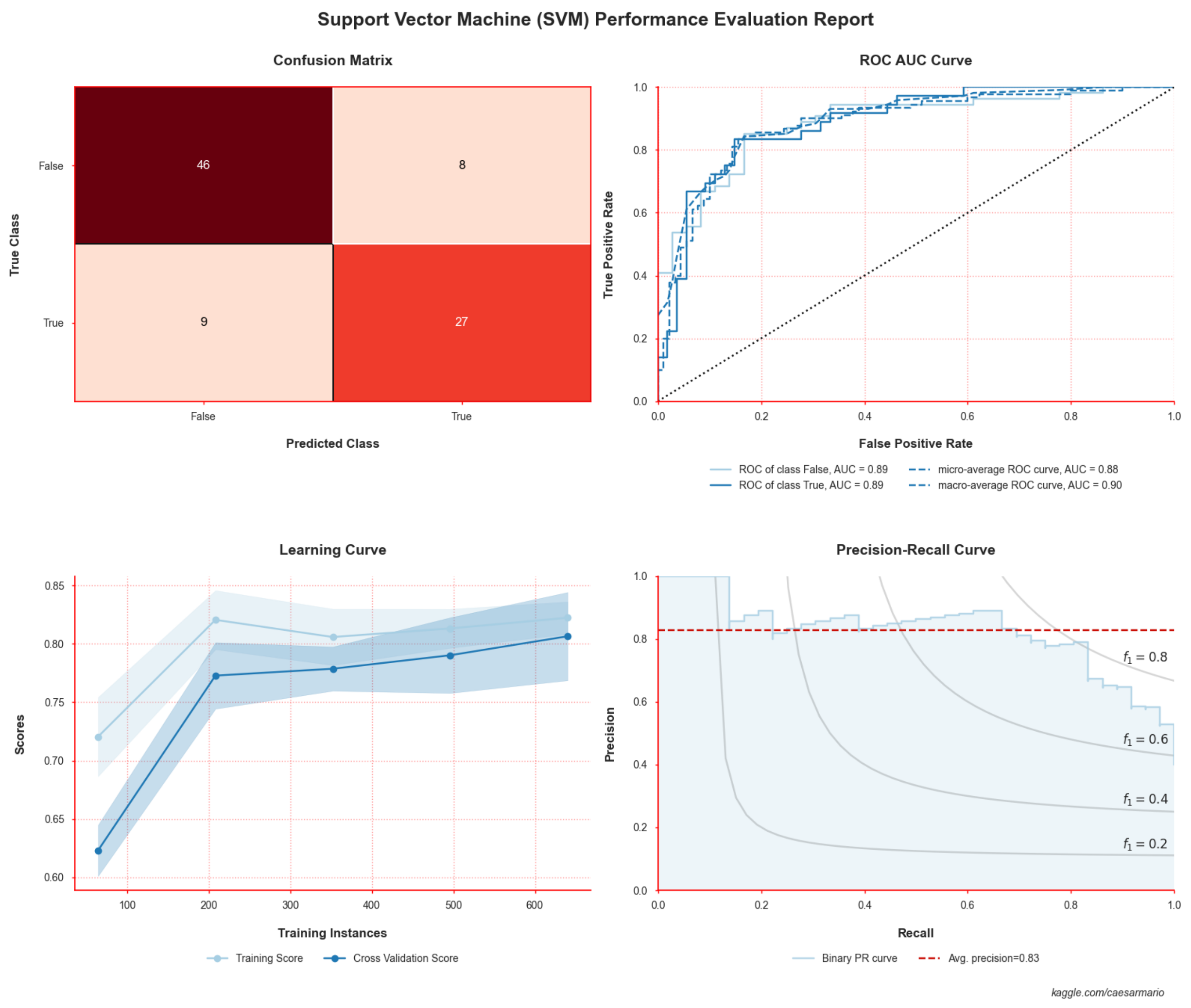
> **Support Vector Machine (SVM)** is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. The goal of the SVM algorithm is **to create the best line or decision boundary that can segregate n-dimensional space into classes** so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.
>
> SVM chooses the **extreme points/vectors** that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.



SVM by JavaTPoint

```
In [19]: # --- SVM Parameters ---
         parameter_svc = [
             {'algo__kernel': ['rbf'], 'algo__gamma': np.arange(0.1, 1, 0.1), 'algo__C': np.
             , {'algo__kernel': ['linear'], 'algo__C': np.arange(0.1, 1, 0.1)}
             , {'algo__kernel': ['poly'], 'algo__degree' : np.arange(1, 10, 1), 'algo__C': n
         ]

         # --- SVM Algorithm ---
         algo_svc = SVC(random_state=1, probability=True)

         # --- Applying SVM ---
         acc_score_train_svc, acc_score_test_svc, best_score_svc = fit_ml_models(algo_svc, p
```

```
.:. Fitting Support Vector Machine (SVM) .:.
Fitting 10 folds for each of 171 candidates, totalling 1710 fits

>> Best Parameters: {'algo__C': 0.6, 'algo__degree': 2, 'algo__kernel': 'poly'}
>> Best Score: 0.825

.:. Train and Test Accuracy Score for Support Vector Machine (SVM) .:.
        >> Train Accuracy: 83.15%
        >> Test Accuracy: 81.11%

.:. Classification Report for Support Vector Machine (SVM) .:.
              precision    recall  f1-score   support

           0       0.84      0.85      0.84        54
           1       0.77      0.75      0.76        36

    accuracy                           0.81        90
   macro avg       0.80      0.80      0.80        90
weighted avg       0.81      0.81      0.81        90
```



7.4 | Gaussian Naive Bayes

Naive Bayes Classifiers are based on the Bayes Theorem, which **one assumption taken is the strong independence assumptions between the features**. These classifiers assume that the value of a particular feature is independent of the value of any other feature. In a supervised learning situation, Naive Bayes Classifiers are trained very efficiently. Naive Bayes classifiers **need a small training data to estimate the parameters needed for classification**. Naive Bayes Classifiers have simple design and implementation and they can applied to many real life situations.

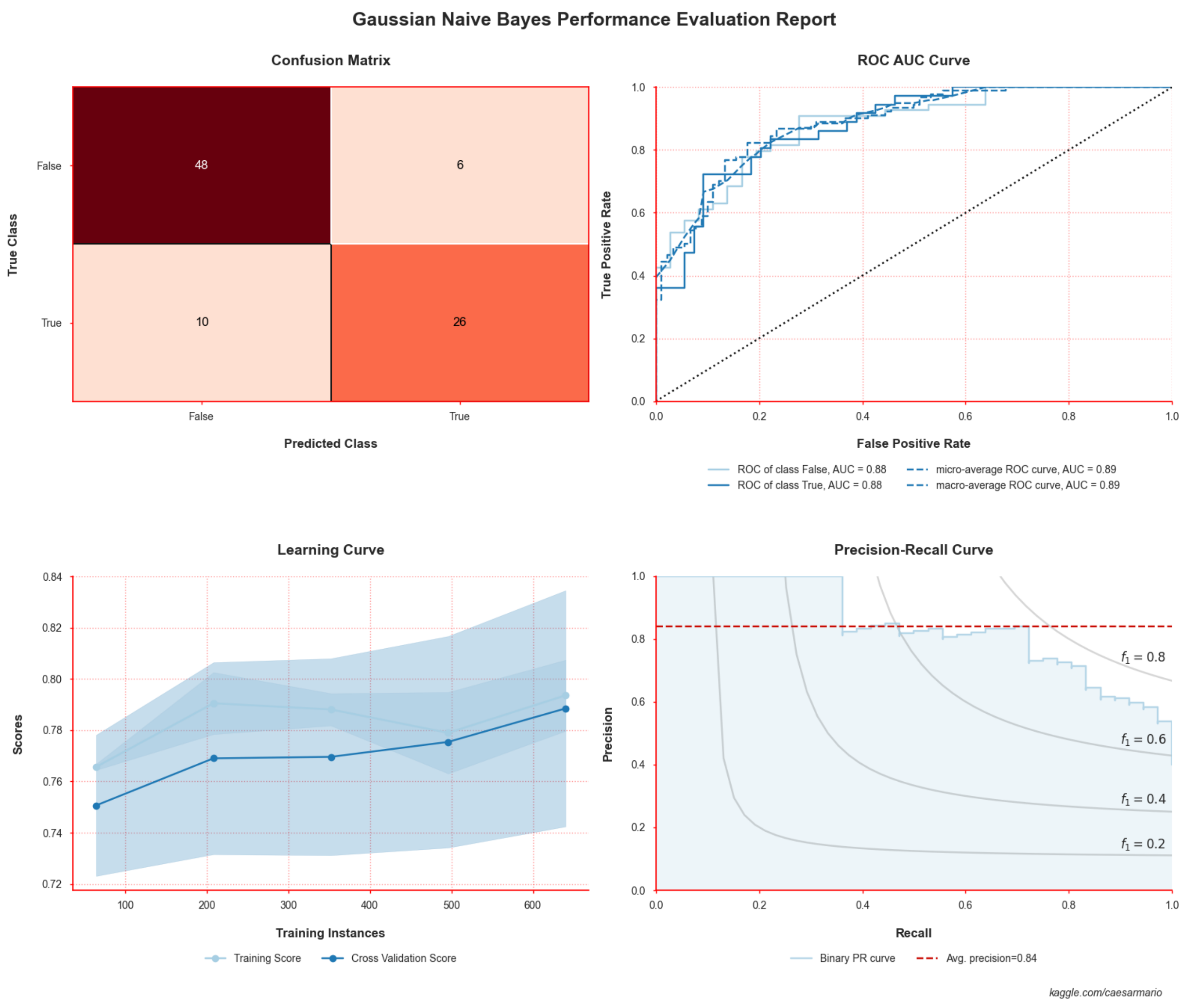
Gaussian Naive Bayes is a **variant of Naive Bayes that follows Gaussian normal distribution and supports continuous data**. When working with continuous data, an assumption often taken is that the continuous values associated with each class are distributed according to a normal (or Gaussian) distribution.

GNB

Gaussian Naive Bayes by OpenGenus

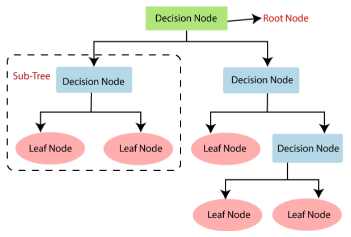
In [20]:
```
# --- Gaussian NB Parameters ---
parameter_gnb = {"algo__var_smoothing": [1e-2, 1e-3, 1e-4, 1e-6]}

# --- Gaussian NB Algorithm ---
algo_gnb = GaussianNB()

# --- Applying Gaussian NB ---
acc_score_train_gnb, acc_score_test_gnb, best_score_gnb = fit_ml_models(algo_gnb, p
```

```
.:. Fitting Gaussian Naive Bayes .:.
Fitting 10 folds for each of 4 candidates, totalling 40 fits

>> Best Parameters: {'algo__var_smoothing': 0.01}
>> Best Score: 0.790

.:. Train and Test Accuracy Score for Gaussian Naive Bayes .:.
        >> Train Accuracy: 80.40%
        >> Test Accuracy: 82.22%

.:. Classification Report for Gaussian Naive Bayes .:.
              precision    recall  f1-score   support

           0       0.83      0.89      0.86        54
           1       0.81      0.72      0.76        36

    accuracy                           0.82        90
   macro avg       0.82      0.81      0.81        90
weighted avg       0.82      0.82      0.82        90
```



7.5 | Decision Tree

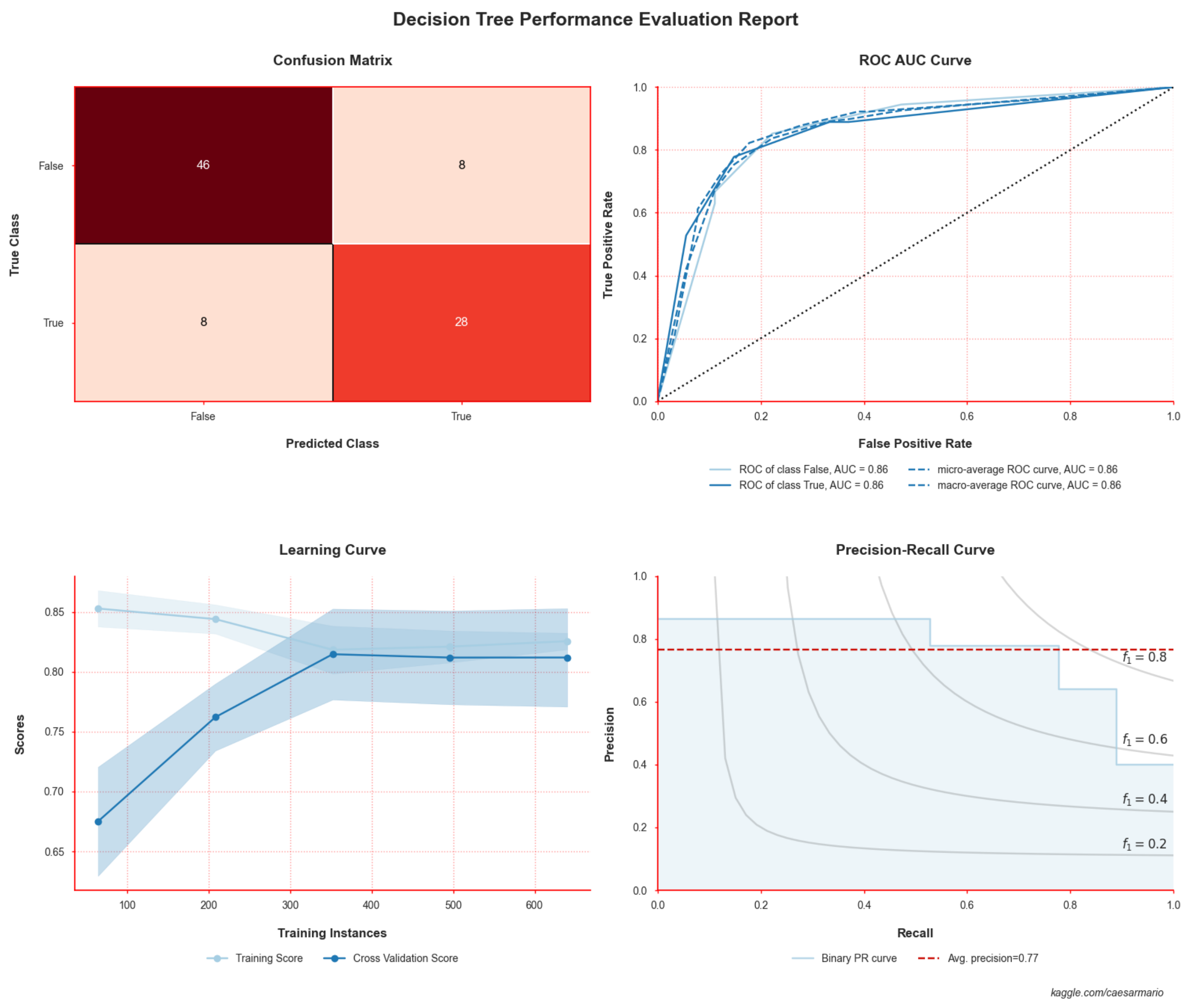
> **Decision Tree** is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome**.
>
> In a Decision tree, there are **two nodes**, which are the **Decision Node and Leaf Node**. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.



🖼️ *Decision Tree by Javatpoint*

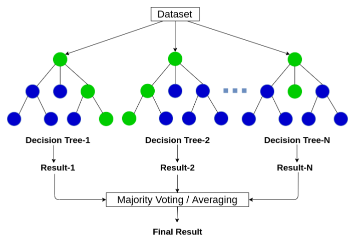
In [21]:
```
# --- Decision Tree Parameters ---
parameter_dt = {"algo__max_depth": [1, 2, 3]}

# --- Decision Tree Algorithm ---
algo_dt = DecisionTreeClassifier(random_state=42)

# --- Applying Decision Tree ---
acc_score_train_dt, acc_score_test_dt, best_score_dt = fit_ml_models(algo_dt, param
```

```
.:. Fitting Decision Tree .:.
Fitting 10 folds for each of 3 candidates, totalling 30 fits

>> Best Parameters: {'algo__max_depth': 3}
>> Best Score: 0.815

.:. Train and Test Accuracy Score for Decision Tree .:.
        >> Train Accuracy: 82.77%
        >> Test Accuracy: 82.22%

.:. Classification Report for Decision Tree .:.
              precision    recall  f1-score   support

           0       0.85      0.85      0.85        54
           1       0.78      0.78      0.78        36

    accuracy                           0.82        90
   macro avg       0.81      0.81      0.81        90
weighted avg       0.82      0.82      0.82        90
```



7.6 | Random Forest

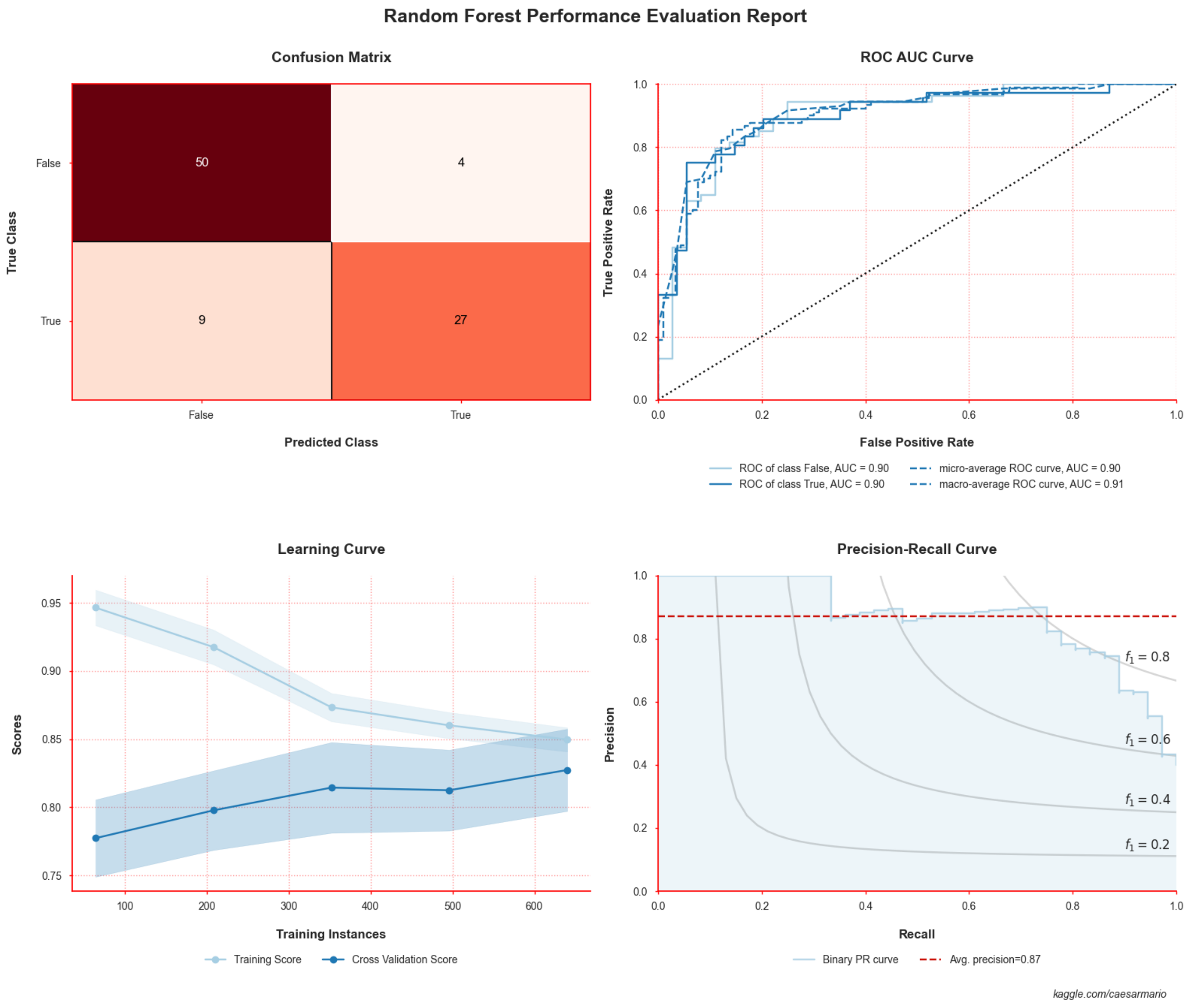
> **Random Forest** is a tree-based machine learning algorithm that **leverages the power of multiple decision trees for making decisions**. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction. **A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models**.



In [22]:
```
# --- Random Forest Parameters ---
parameter_rf = {"algo__max_depth": np.arange(1, 6, 1)}

# --- Random Forest Algorithm ---
```

```python
algo_rf = RandomForestClassifier(random_state=99, n_jobs=-1)

# --- Applying Random Forest ---
acc_score_train_rf, acc_score_test_rf, best_score_rf = fit_ml_models(algo_rf, param
```

```
.:. Fitting Random Forest .:.
Fitting 10 folds for each of 5 candidates, totalling 50 fits

>> Best Parameters: {'algo__max_depth': 5}
>> Best Score: 0.837

.:. Train and Test Accuracy Score for Random Forest .:.
        >> Train Accuracy: 85.52%
        >> Test Accuracy: 85.56%

.:. Classification Report for Random Forest .:.
              precision    recall  f1-score   support

           0       0.85      0.93      0.88        54
           1       0.87      0.75      0.81        36

    accuracy                           0.86        90
   macro avg       0.86      0.84      0.85        90
weighted avg       0.86      0.86      0.85        90
```

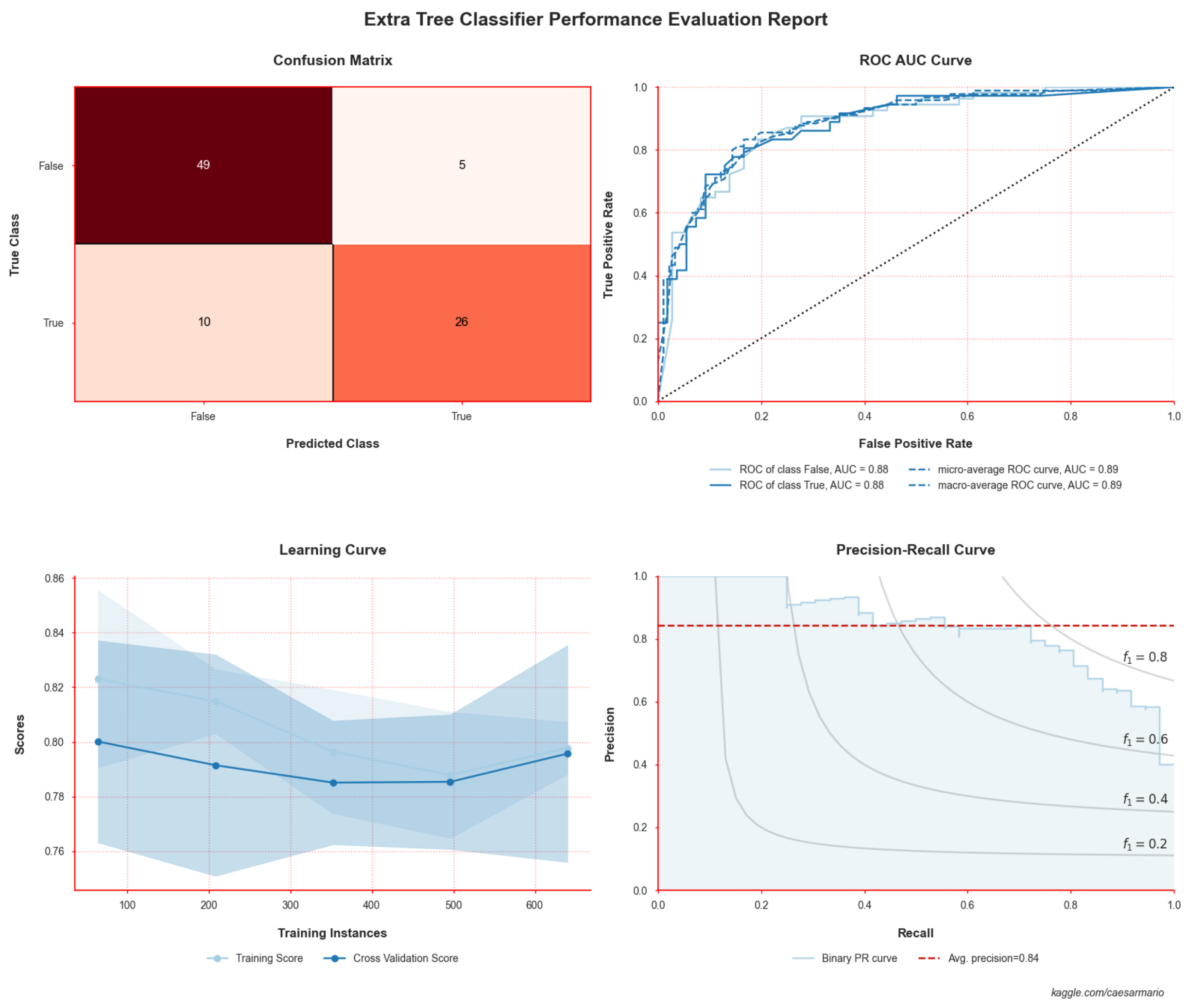


7.7 | Extra Tree Classifier

> **Extra Trees Classifier** is a type of ensemble learning technique which **aggregates the results of multiple de-correlated decision trees collected in a "forest" to output it’s classification result**. In concept, it is very similar to a Random Forest Classifier and only differs from it in the manner of construction of the decision trees in the forest.
>
> Each Decision Tree in the Extra Trees Forest is **constructed from the original training sample**. Then, at each test node, each tree is provided with a **random sample of k features** from the feature-set from which each decision tree must select the best feature to split the data based on some mathematical criteria (typically the Gini Index). This random sample of features leads to the creation of multiple de-correlated decision trees.

In [23]:
```
# --- Extra Tree Parameters ---
parameter_et = {"algo__max_depth": [2, 3]
    , "algo__max_leaf_nodes": [3, 5, 7]}

# --- Extra Tree Algorithm ---
algo_et = ExtraTreesClassifier(random_state=42, n_jobs=-1)

# --- Applying Extra Tree ---
acc_score_train_et, acc_score_test_et, best_score_et = fit_ml_models(algo_et, param
```

```
.:. Fitting Extra Tree Classifier .:.
Fitting 10 folds for each of 6 candidates, totalling 60 fits

>> Best Parameters: {'algo__max_depth': 3, 'algo__max_leaf_nodes': 5}
>> Best Score: 0.809

.:. Train and Test Accuracy Score for Extra Tree Classifier .:.
        >> Train Accuracy: 80.90%
        >> Test Accuracy: 83.33%

.:. Classification Report for Extra Tree Classifier .:.
              precision    recall  f1-score   support

           0       0.83      0.91      0.87        54
           1       0.84      0.72      0.78        36

    accuracy                           0.83        90
   macro avg       0.83      0.81      0.82        90
weighted avg       0.83      0.83      0.83        90
```



7.8 | Gradient Boosting

Boosting is a method of **converting weak learners into strong learners**. In boosting, **each new tree is a fit on a modified version** of the original data set. It strongly relies on the prediction that the next model will reduce prediction errors when blended with previous ones. The main idea is **to establish target outcomes for this upcoming model to minimize errors**.

Gradient Boosting trains many models in **a gradual, additive and sequential manner**. The term gradient boosting emerged because every case's target outcomes are based on the gradient's error with regards to the predictions. Every model reduces prediction errors by taking a step in the correct direction.

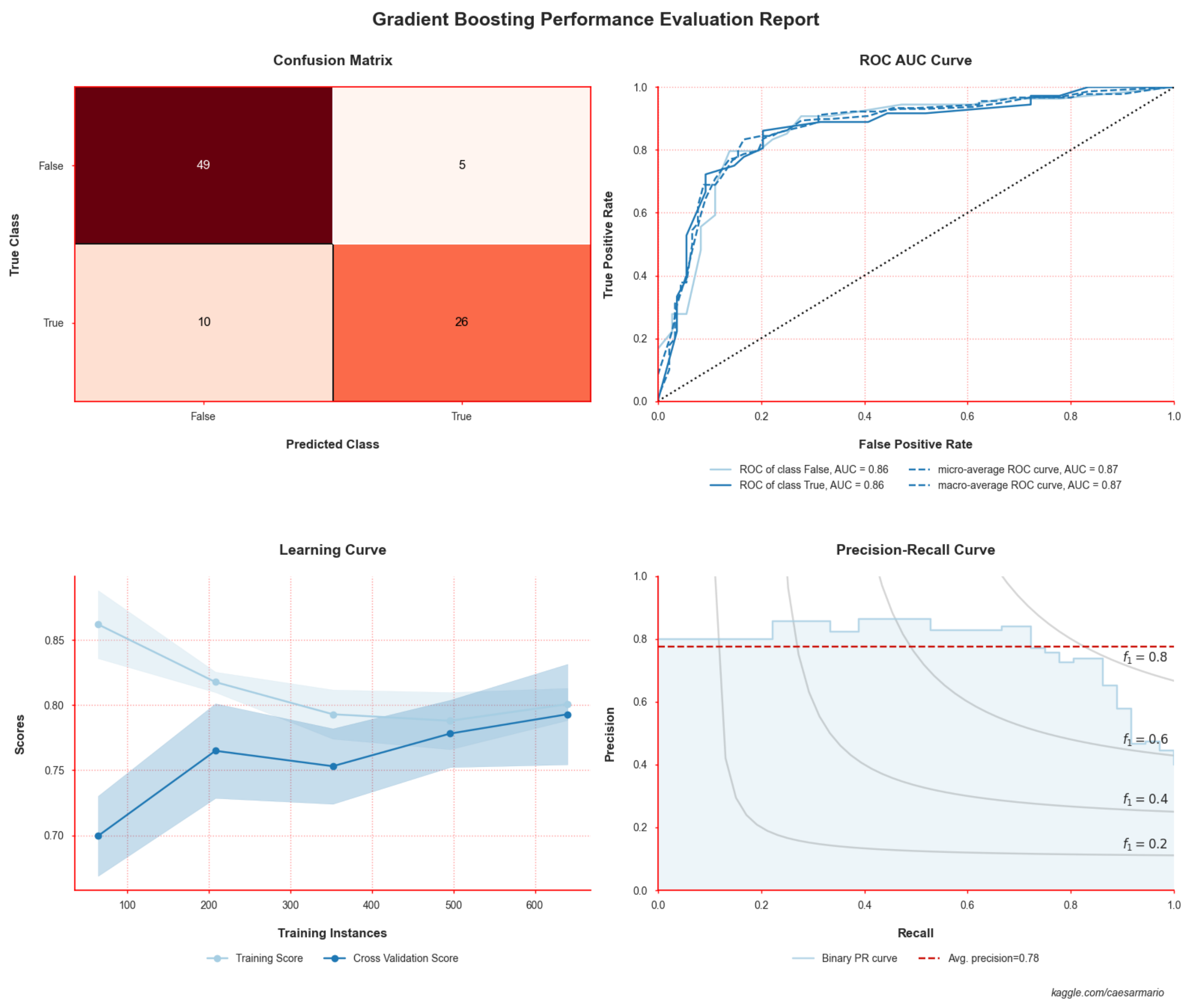
GB

Boosting Algorithm by Rui Guo et al.

```
In [24]: # --- Gradient Boosting Parameters ---
         parameter_gb = {
```

```
    "algo__learning_rate": [0.1, 0.3, 0.5]
    , "algo__n_estimators": [2, 4, 6]
    , "algo__min_weight_fraction_leaf": [0.1, 0.2, 0.5]
}

# --- Gradient Boosting Algorithm ---
algo_gb = GradientBoostingClassifier(loss="exponential", random_state=2)

# --- Applying Gradient Boosting ---
acc_score_train_gb, acc_score_test_gb, best_score_gb = fit_ml_models(algo_gb, param
```

```
.:. Fitting Gradient Boosting .:.
Fitting 10 folds for each of 27 candidates, totalling 270 fits

>> Best Parameters: {'algo__learning_rate': 0.3, 'algo__min_weight_fraction_leaf':
0.1, 'algo__n_estimators': 6}
>> Best Score: 0.803

.:. Train and Test Accuracy Score for Gradient Boosting .:.
        >> Train Accuracy: 80.90%
        >> Test Accuracy: 83.33%

.:. Classification Report for Gradient Boosting .:.
              precision    recall  f1-score   support

           0       0.83      0.91      0.87        54
           1       0.84      0.72      0.78        36

    accuracy                           0.83        90
   macro avg       0.83      0.81      0.82        90
weighted avg       0.83      0.83      0.83        90
```



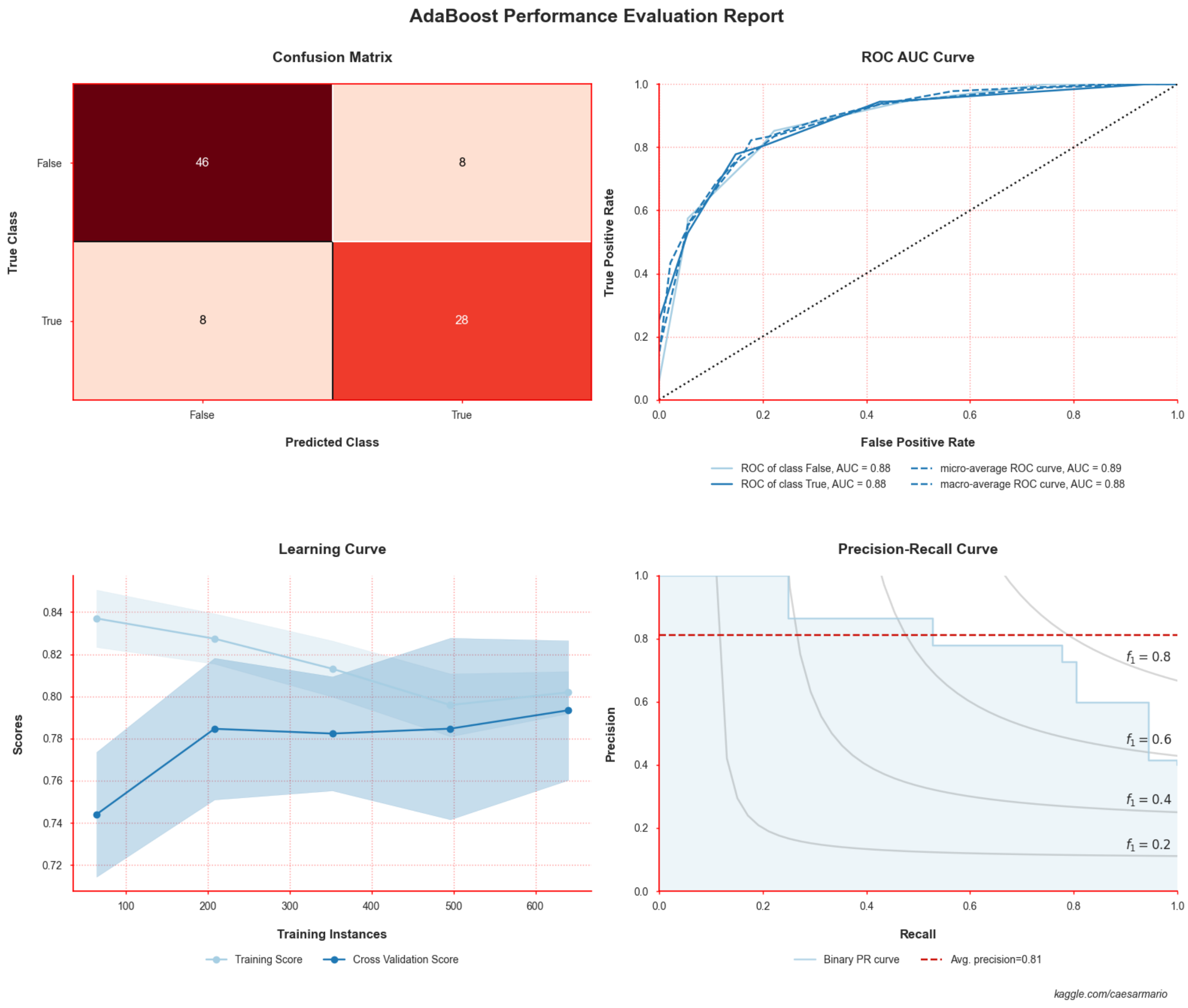
7.9 | AdaBoost

> **AdaBoost** also called **Adaptive Boosting** is a technique in Machine Learning used as an Ensemble Method. The most common algorithm used with AdaBoost is **decision trees with one level** that means with Decision trees with only 1 split. These trees are also called **Decision Stumps**. **AdaBoost builds a model and gives equal weights to all the data points**. It then assigns higher weights to points that are wrongly classified. Now, all the points which have higher weights are given more importance in the next model. It will keep training models until and unless a lowe error is received.

In [25]:
```
# --- AdaBoost Parameters ---
parameter_ab = {
    "algo__n_estimators": [6, 7, 10]
    , "algo__learning_rate": [0.2, 0.4, 0.8]
}

# --- AdaBoost Algorithm ---
algo_ab = AdaBoostClassifier(random_state=1)
```

```python
# --- Applying AdaBoost ---
acc_score_train_ab, acc_score_test_ab, best_score_ab = fit_ml_models(algo_ab, param
```

```
.:. Fitting AdaBoost .:.
Fitting 10 folds for each of 9 candidates, totalling 90 fits

>> Best Parameters: {'algo__learning_rate': 0.8, 'algo__n_estimators': 6}
>> Best Score: 0.803

.:. Train and Test Accuracy Score for AdaBoost .:.
        >> Train Accuracy: 80.65%
        >> Test Accuracy: 82.22%

.:. Classification Report for AdaBoost .:.
              precision    recall  f1-score   support

           0       0.85      0.85      0.85        54
           1       0.78      0.78      0.78        36

    accuracy                           0.82        90
   macro avg       0.81      0.81      0.81        90
weighted avg       0.82      0.82      0.82        90
```



7.10 | Model Comparison 👀

After implementing and tuning 9 models, this section will compare all machine learning models accuracy and best score.

```
In [26]: # --- Create Accuracy Comparison Table ---
df_compare = pd.DataFrame({'Model': ['Logistic Regression', 'K-Nearest Neighbour',
                                     'Decision Tree', 'Random Forest', 'Extra Tree
                          , 'Accuracy Train': [acc_score_train_lr, acc_score_train
                                              acc_score_train_dt, acc_score_train
                          , 'Accuracy Test': [acc_score_test_lr, acc_score_test_kn
                                             acc_score_test_dt, acc_score_test_rf
                          , 'Best Score': [best_score_lr, best_score_knn, best_sco
                                          best_score_et, best_score_gb, best_scor

# --- Create Comparison Table ---
print(clr.start+f".:. Models Comparison .:."+clr.end)
print(clr.color+'*' * 26)
df_compare.sort_values(by='Best Score', ascending=False).reset_index(drop=True).sty
```

```
.:. Models Comparison .:.
**************************
```

Out[26]:

	Model	Accuracy Train	Accuracy Test	Best Score
0	Random Forest	85.518000	85.556000	0.836500
1	Support Vector Machine	83.146000	81.111000	0.825200
2	Decision Tree	82.772000	82.222000	0.815300
3	Extra Tree Classifier	80.899000	83.333000	0.809000
4	Logistic Regression	81.149000	83.333000	0.807800
5	K-Nearest Neighbour	82.772000	85.556000	0.805200
6	Gradient Boosting	80.899000	83.333000	0.802800
7	AdaBoost	80.649000	82.222000	0.802800
8	Gaussian NB	80.400000	82.222000	0.790300

From the results of the accuracy of the train and test above, **most models experienced overfitting or underfitting**. However, **several models have a good fit**, where the difference between train and test accuracy or vice versa is a little. These models are **random forest, AdaBoost, Gaussian Naive Bayes, and extra tree classifier**. As seen in the data frame above, of the four models, random forest and Gaussian NB have the highest accuracy compared to the other models. This is also supported by the ROC AUC curve figure for random forest and Gaussian NB, where the AUC value for both models is close to 1, which means that both models can predict well whether patients have heart disease. The **confusion matrix** shows that the prediction results between the actual target and the predicted target for the random forest and Gaussian NB models in each class in the test data are better than those of other models.

Judging from the F1 scores of both models, both models do a very good job differentiating sick patients from those who are not (scores above **0.85**). If seen from the precision value for Gaussian NB, 93% of all the patients that the model predicted have heart disease. Whereas in the random forest precision value, only 88% out of all the patients that the model predicted have heart disease, slightly lower than the Gaussian NB precision value. At the **Gaussian NB recall value, this model only correctly predicts 81% of all heart disease patients. However, in the random forest recall value, this model can predict better than Gaussian NB, where 88% of patients are predicted to have heart disease out of all patients who do have heart disease**.

Furthermore, in the learning curve between Gaussian NB and random forest, the learning curve for the random forest is more ideal than Gaussian NB. This is because **both training and validation scores of Gaussian NB stay too close together** (indicates low variance and high bias). This will more likely result in poor fit and especially poor generalization of the data (towards the data it has not seen before). Whereas in a **random forest**, the validation score constantly improves as the number of training set sizes gets larger (notice the difference in the x-axis scale from the previous two curves). Both the training and validation scores also converge to nearly similar values. This is a model that can generalize very well. From the analysis above, it can be concluded that the random forest model best predicts whether a person has heart disease.

In the random forest feature importance plot, the **five following features to be the most important** are major vessels number (ca), fixed defect thalassemia (thal_2), ST depression induced by exercise relative to rest (oldpeak), reversable defect thalassemia (thal_3), and exercise-induced angina (exang). Major vessel number might be significant since a lower number of major vessels can lead to a reduced blood supply to the heart muscle, which can result in ischemia (lack of oxygen and nutrients) and potentially lead to heart disease. For example, individuals with single-vessel disease have a greater risk of adverse cardiac events than those with multi-vessel disease. Fixed defects and reversible defects of thalassemia can affect the heart by reducing the number of red blood cells and oxygen delivery to the heart muscles, which can lead to the development of myocardial ischemia, coronary artery disease, and other forms of heart disease. Early detection and appropriate treatment of thalassemia can help prevent these complications and improve cardiovascular health.

ST depression induced by exercise relative to rest can be important finding which can suggest underlying heart disease and may warrant further evaluation and management to reduce the risk of adverse cardiovascular events. Early detection and appropriate treatment of underlying heart disease can help prevent complications and improve cardiovascular health. Finally, exercise-induced angina is also important because it can suggest underlying CAD and may warrant further evaluation and management to reduce the risk of adverse cardiovascular events. Individuals with exercise-induced angina may need further testing, such as stress testing or coronary angiography, to assess the extent of their coronary artery disease and determine the most appropriate treatment options.

8. | Miscellaneous 🧪

This section focuses on creating a complete pipeline, starting from data processing to a machine learning pipeline, using the best model concluded in the previous section and exporting it to `joblib` and `pickle (.pkl)` files. Besides that, test dataset predicted results would also be exported along with actual results in CSV and JSON files. Moreover, this section will also make predictions on dummy data (data generated using Python functions) and export them to CSV and JSON files.

8.1 | Creating Outputs 📤

The complete pipeline will be exported in this section. The pipeline will be stored using the joblib library into `joblib` and `pickle (.pkl)` files. This section will also show the test data frame before exporting the predicted results and the actual results to the CSV and JSON files.

In [27]:
```python
# --- Complete Pipeline: Preprocessor & RF ---
rf_pipeline = Pipeline([
    ('preprocessor', preprocessor)
    , ('algo', RandomForestClassifier(max_depth=3, random_state=99, n_jobs=-1))
])

# --- Save Complete Pipeline (joblib and pickle) ---
directory = r'D:\Data_Science\Machine_Learning\Titanic_Classification'
file_name = 'pipeline_titanic_survival_random_forest_darshanpathak'

# Save pipeline using joblib
for ext in ['joblib', 'pkl']:
    file_path = os.path.join(directory, f'{file_name}.{ext}')
    joblib.dump(rf_pipeline, file_path)
```

In [28]:
```python
# --- Dataframes to Create Test Output Dataframe ---
rf_pipeline.fit(x_train, y_train)
y_pred_rf = rf_pipeline.predict(x_test)
pred_target = pd.DataFrame(y_pred_rf, columns=['pred_target'])

x_test_output = x_test.reset_index()
actual_target = y_test.to_frame(name='actual_target').reset_index()

# --- Combining and Creating Test Output Dataframe ---
df_test_output = pd.concat([x_test_output, actual_target, pred_target], axis=1).dro

# --- Showing Sample Test Output Dataframe ---
print(clr.start+'.: Sample Test Dataframe :.'+clr.end)
print(clr.color+'*' * 28)
df_test_output.sample(n=10, random_state=0).reset_index(drop=True).style.background
```

```
.: Sample Test Dataframe :.
****************************
```

Out[28]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	actual_target	pred_targ
0	3	male	20.000000	0	0	7.925000	S	0	
1	3	male	30.000000	0	0	7.225000	C	0	
2	1	male	11.000000	1	2	120.000000	S	1	
3	3	male	29.000000	1	0	7.045800	S	0	
4	2	male	21.000000	0	0	73.500000	S	0	
5	3	male	19.000000	0	0	7.895800	S	0	
6	3	male	16.000000	1	3	34.375000	S	0	
7	3	male	25.000000	0	0	7.750000	Q	0	
8	2	female	38.000000	0	0	13.000000	S	0	
9	3	male	25.000000	0	0	7.250000	S	0	

In [29]:

```
# --- Export to CSV and JSON Files ---
output_name = 'titanic_survival_random_forest_darshanpathak'
df_test_output.to_csv(f'D:\Data_Science\Machine_Learning\Titanic_Classification.csv
df_test_output.to_json(f'D:\Data_Science\Machine_Learning\Titanic_Classification.js
```

9. | Conclusions and Future Improvements 🧐

From the results of dataset analysis and implementation of machine learning models in the previous section, it can be concluded as follows:

> - Random forest is the best model out of 9 machine-learning models implemented in this notebook. This is because **this model fits well with train and test data**. In addition, **this model also performs better than other models when predicting the test data** (can be seen from the performance evaluation graph and classification report of each model).
> - Based on previous findings, medical workers can focus more on examining the five variables previously mentioned. This is because these five variables most influence whether a patient has heart disease.
> - The prediction results on test data, dummy data, and the complete machine learning pipeline have been successfully exported for other purposes. In addition, data exploration has also been successfully carried

> out using the `ydata-profiling`, `seaborn`, and `matplotlib` libraries.
> - Several improvements can be implemented in the following research/notebook. For example, by carrying out A/B Testing on patients with the same major vessel number in one group. Another example is performing advanced hyperparameter tuning experiments to obtain higher accuracy (~90%).

In [ ]:

```

```

In [ ]:

```

```