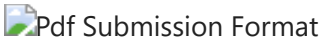


Employee Churn Prediction

1. | Introduction 👋

🎳 Problem Statement

As the global economy evolves, employee churn has become a critical challenge for organizations across various industries. The ability to predict employee churn accurately can provide valuable insights for HR departments and management teams to implement proactive measures aimed at retention and talent management. In this project, we aim to leverage various machine learning and deep learning techniques to develop a predictive model for employee churn, ultimately assisting organizations in identifying at-risk employees and devising effective retention strategies.

🤔 Dataset Problems

This dataset is taken from the **Kaggle Website**. This dataset contains **Employee data (i.e., name, age, department, income, etc.)** along with information on employechurnn. **With the help of machine learning models** based on the employee information provided in the dataset, we aim to **uncover the factors that lead to employechurnon** 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 employee attrition.
- **Export prediction result on test data** into files.

👨‍💻 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]:
```
# --- 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
import tensorflow as tf
from tensorflow import keras

from ydata_profiling import ProfileReport
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, A
from sklearn.metrics import classification_report, accuracy_score,f1_score, precis
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")
```

```
WARNING:tensorflow:From C:\Users\Darshan\AppData\Local\Packages\PythonSoftwareFoun
dation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages
\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is depr
ecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

3. | Reading Dataset 👓

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

In [2]:
```
# --- Importing Dataset ---
df = pd.read_csv("employee.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='Blu
styled_df
```

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

Out[2]:

	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Educati
0	41	Travel_Rarely	1102	Sales	1	2	Life S
1	49	Travel_Frequently	279	Research & Development	8	1	Life S
2	37	Travel_Rarely	1373	Research & Development	2	2	
3	33	Travel_Frequently	1392	Research & Development	3	4	Life S
4	27	Travel_Rarely	591	Research & Development	2	1	
5	32	Travel_Frequently	1005	Research & Development	2	2	Life S
6	59	Travel_Rarely	1324	Research & Development	3	3	
7	30	Travel_Rarely	1358	Research & Development	24	1	Life S
8	38	Travel_Frequently	216	Research & Development	23	3	Life S
9	36	Travel_Rarely	1299	Research & Development	27	3	

Dataset Description 📜

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

Variable Name	Description	Sample Data
Age	Employee age (in years)	[41, 49, 37]
Attrition	Employee attrition status	['Yes','No']
BusinessTravel	Frequency of business travel	['Travel_Rarely', 'Travel_Frequently', 'No Travel']
DailyRate	Daily rate of pay	[102, 279, 1373]
Department	Department where employee works	['Sales', 'Research & Development', 'Human Resources']
DistanceFromHome	Distance of employee's home from workplace (in miles)	[1, 8, 2]
Education	Level of education attained	[2, 1, 4]
EducationField	Field of education	['Life Sciences', 'Other', 'Medical']
HourlyRate	Hourly rate of pay	[94, 61, 92]
JobInvolvement	Level of involvement in job	[3, 2, 4, 1]

Variable Name	Description	Sample Data
JobLevel	Level of job position	[2,1, 3]
JobRole	Role of employee in the job	['Sales Executive', 'Research Scientist', 'Labora tory T echni cian']
JobSatisfaction	Marital status of employee	['Single', ' Married', 'Divorced']
MonthlyRate	Monthly rate of pay	[19479, 24907, 2396]
NumCompaniesWorked	Number of companies worked at previously	[8, 1, 6]
OverTime	Whether the employee works overtime	['Yes', 'No']
PercentSalaryHike	Percentage increase in salary	[11, 23, 15]
PerformanceRating	Performance rating of employee	[3, 4]
RelationshipSatisfaction	Level of satisfaction with relationships at work	[1, 4, 2, 3]
StandardHours	Standard hours of work per week	[80]
StockOptionLevel	Level of stock option available to employee< /td>	[0, 3, 2]
TotalWorkingears	Total number of years work ed	[8, 10]
TrainingTimesLastYear	Number of training time s last year	[0, 3,2]
WorkLifeBalance	Level of work-life balance	[1, 3, 2, 4]

---

4. | Data Preprocessing 🔍

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 [3]:
```
# --- Dataset Report ---
ProfileReport(df,
              title="Employee Attrition 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': Fals
             ).to_notebook_iframe()
```

Overview

Dataset statistics

Number of variables	35
Number of observations	1470
Missing cells	0
Missing cells (%)	0.0%
Total size in memory	1.0 MiB
Average record size in memory	745.7 B

Variable types

Numeric	27
Text	8

Some columns can be removed because their values do not affect the analysis results

- Over18: All values are 'Y'
- EmployeeCount: All values are 1
- StandardHours: All values are 80
- EmployeeNumber: Identifier for employees
- PercentSalaryHike: less correlated
- YearsSinceLastPromotion: less correlated
- DailyRate: Can be calculated from monthly income
- HourlyRate: Can be calculated from monthly income
- MonthlyRate: Can be calculated from monthly income
- PerformanceRating: Can be calculated from monthly income
- NumCompaniesWorked: Can be calculated from monthly income

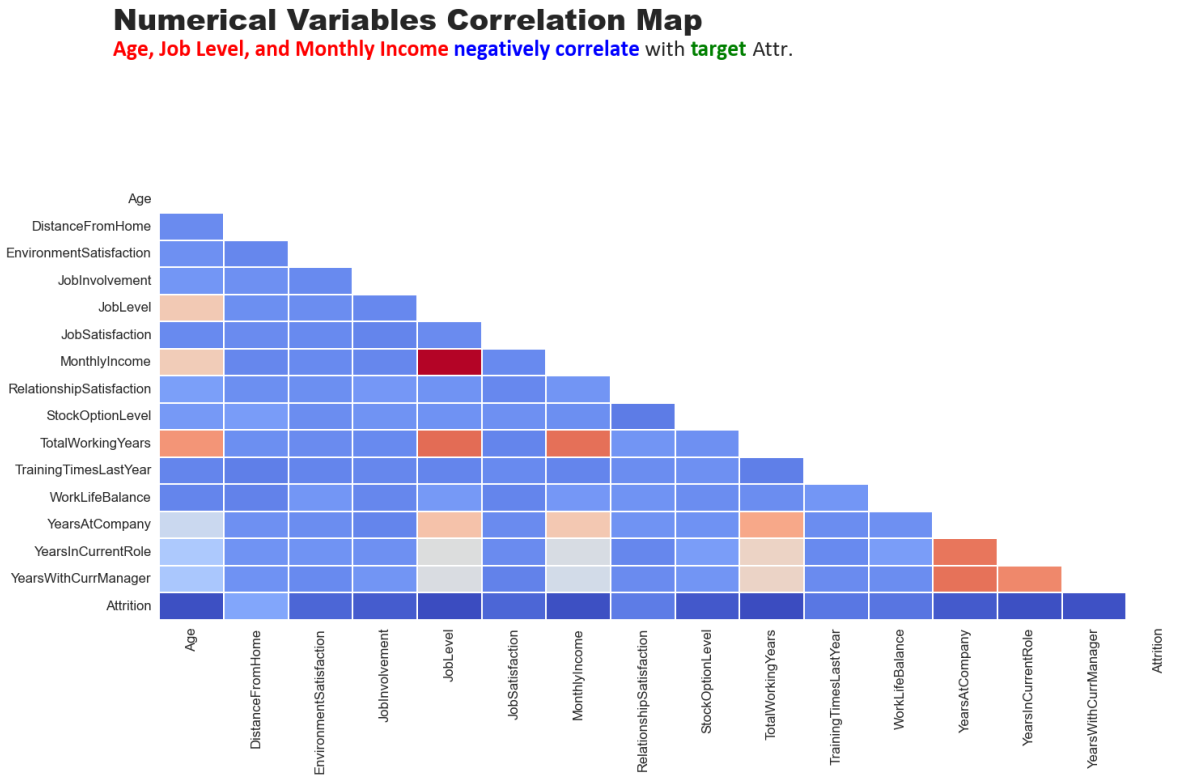
- EducationNuCan be calculated from monthly incomeloyees

In [4]:
```
df = df.drop(['Over18','EmployeeCount','StandardHours','EmployeeNumber','PercentSa
```

In [5]:
```
df = df.drop(['DailyRate' ,'HourlyRate','MonthlyRate','PerformanceRating'],axis=1)
```

In [6]:
```
df = df.drop(['NumCompaniesWorked','Education'],axis=1)
```

In [7]:
```
# --- Correlation Map Variables ---
palette = ["#4361EE", "#7209B7", "#3A0CA3", "#4CC9F0","#F72585"]
corr = df.corr(numeric_only=True)
colors = ['red', 'blue', 'green', 'yellow', 'purple', 'orange', 'cyan', 'magenta']
suptitle = dict(x=0.1, y=1.01, fontsize=26, weight='heavy', ha='left', va='bottom
title = dict(x=0.1, y=0.98, fontsize=20, weight='normal', ha='left', va='bottom',
xy_label = dict(size=12)
highlight_textprops = [{'weight':'bold', 'color': colors[0]},{'weight':'bold', 'co

# --- Correlation Map (Heatmap) ---
mask = np.triu(np.ones_like(corr, dtype=bool))
fig, ax = plt.subplots(figsize=(15, 10))
sns.heatmap(corr, mask=mask, annot=True, cmap='coolwarm', linewidths=0.2, cbar=Fal
yticks, ylabels = plt.yticks()
xticks, xlabels = plt.xticks()
ax.set_xticklabels(xlabels, rotation=90, **xy_label)
ax.set_yticklabels(ylabels, **xy_label)
ax.grid(False)
fig_text(s='Numerical Variables Correlation Map', **suptitle)
fig_text(s='<Age, Job Level, and Monthly Income> <negatively correlate> with <targ
plt.tight_layout(rect=[0, 0.08, 1, 1.01])
```



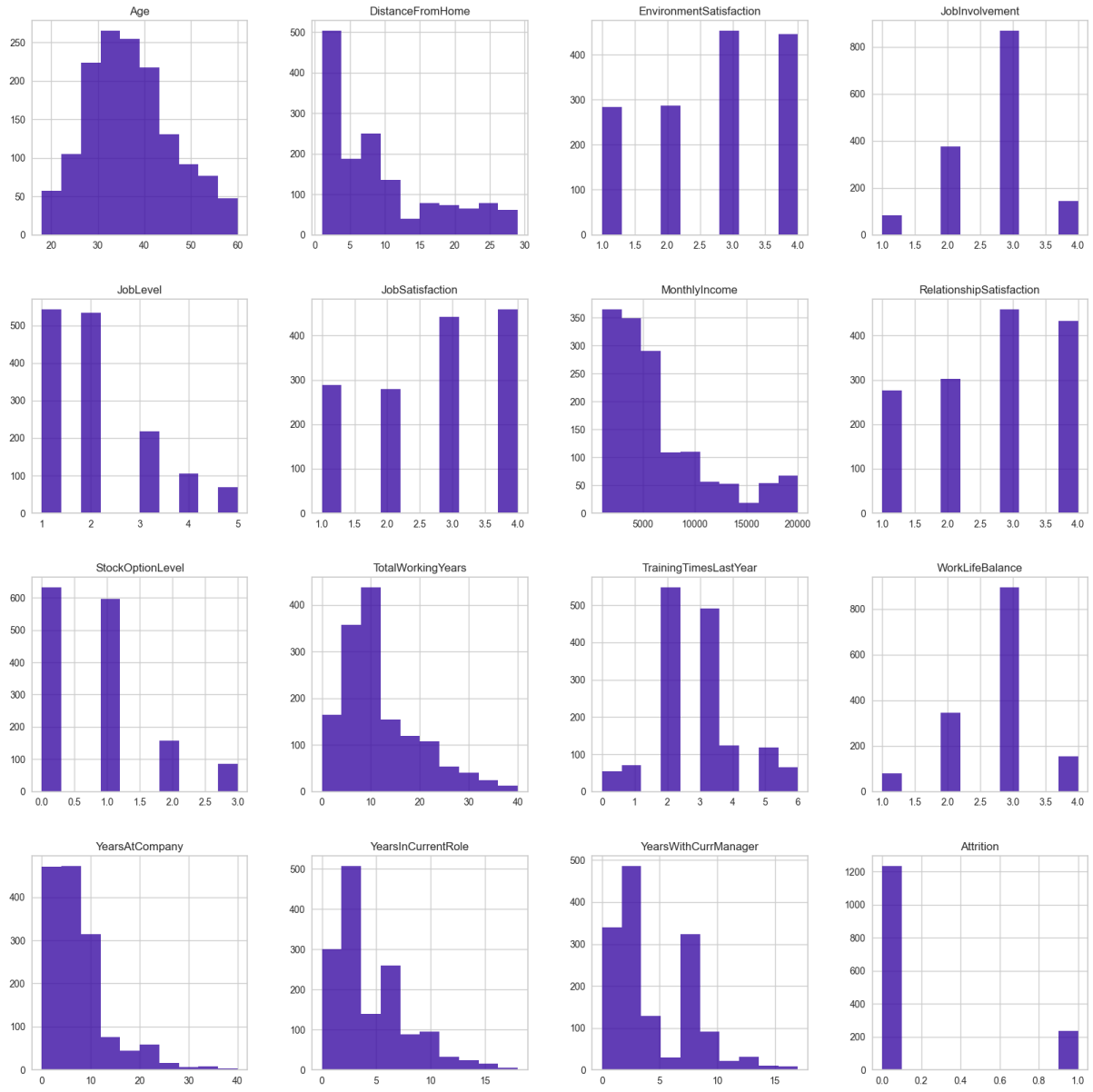
From **dataset report** and **correlation matrix**, it can be **concluded** that:

> - 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**.
> - Furthermore, Average Monthly Income is **6502.**
> - **The mean age of the employee in the dataset was 36 years old**, with the maximum being 60 years old and the youngest being 18 years old.

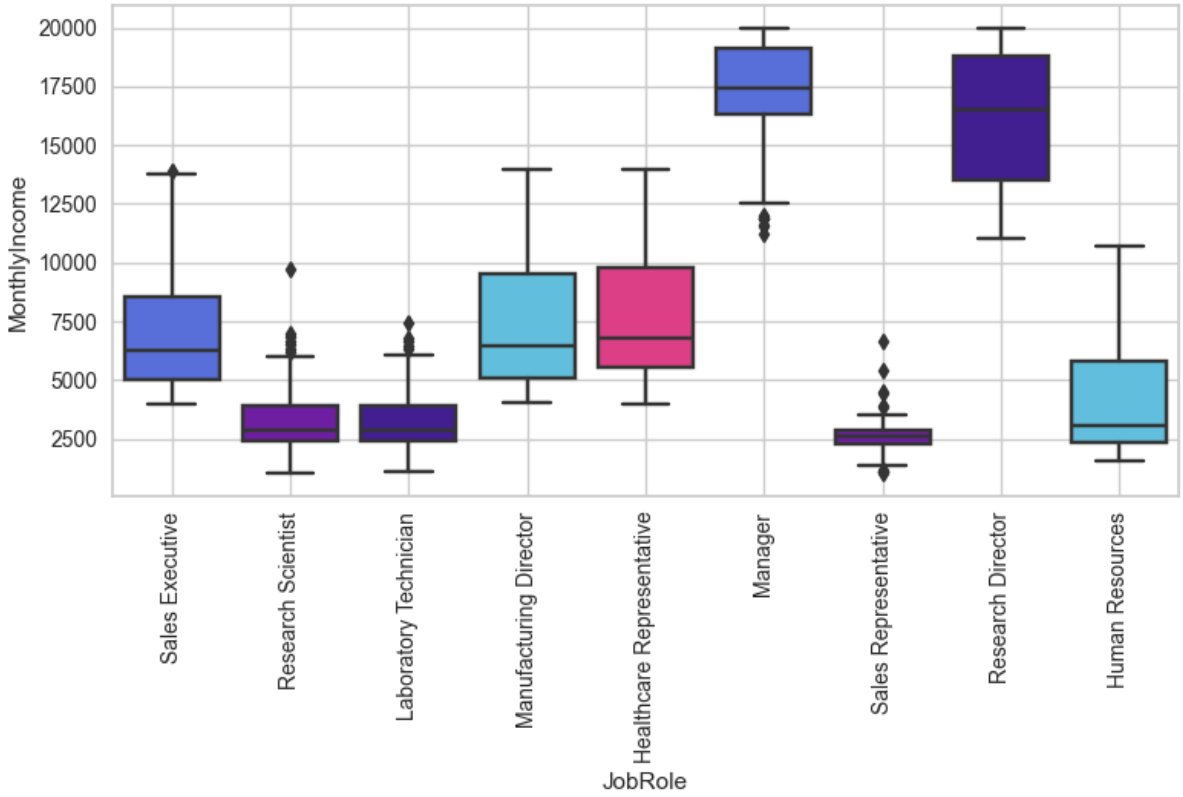
5. | EDA 📈

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

In [8]:
```
df.hist(figsize=(20, 20), color="#3A0CA3", alpha=0.8)
plt.show()
```



In [9]:
```
ax=sns.boxplot(y=df['MonthlyIncome'],x=df['JobRole'],palette=palette)
plt.setp(ax.get_xticklabels(), rotation=90)
plt.grid(True,alpha=1)
plt.tight_layout()
plt.show()
```



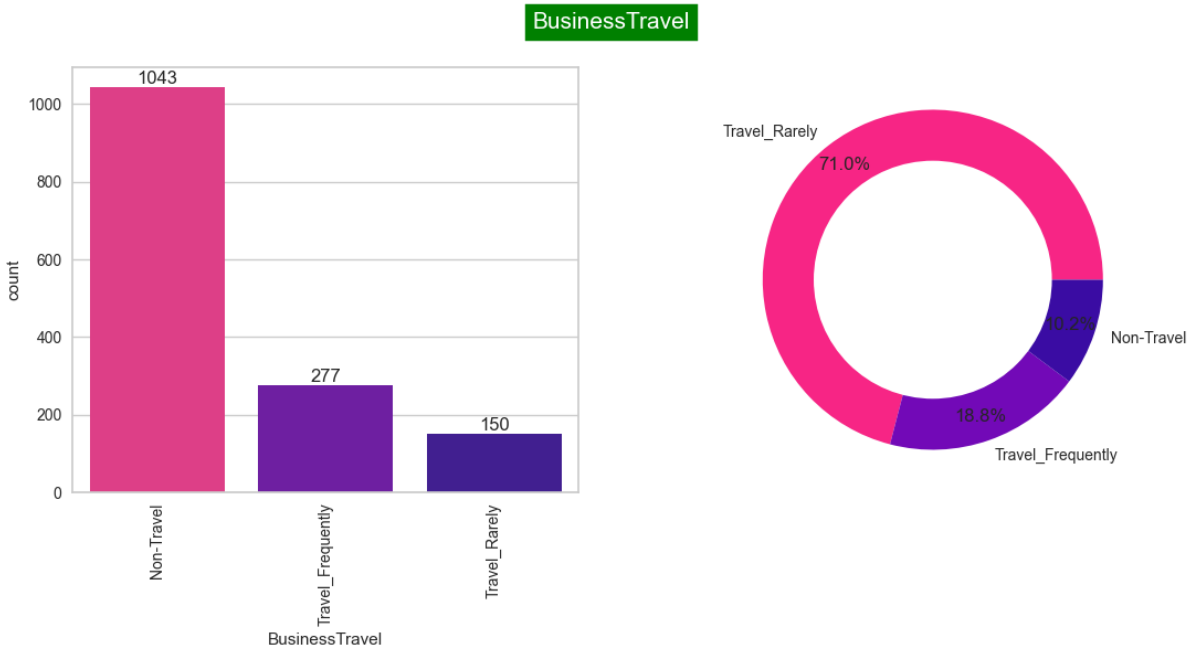
Separating Categorical and Numerical Columns

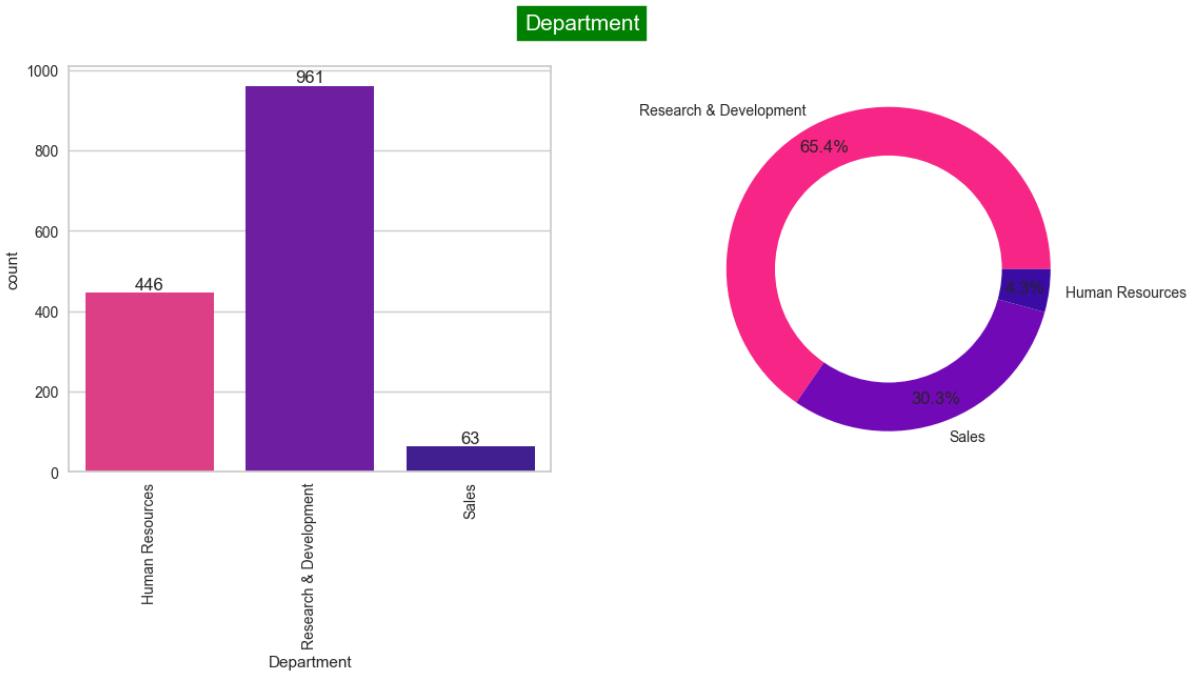
In [10]:
```
#Categorical Columns
cat_columns = df.select_dtypes(['object']).columns
print(f"Categorical Columns: {cat_columns}")
#Numerical Columns
num_columns = df.select_dtypes(['number']).columns
num_columns = num_columns.drop('Attrition', errors='ignore')
print(f"Numerical Columns: {num_columns}")
```

```
Categorical Columns: Index(['BusinessTravel', 'Department', 'EducationField', 'Gen
der', 'JobRole',
       'MaritalStatus', 'OverTime'],
      dtype='object')
Numerical Columns: Index(['Age', 'DistanceFromHome', 'EnvironmentSatisfaction', 'J
obInvolvement',
       'JobLevel', 'JobSatisfaction', 'MonthlyIncome',
       'RelationshipSatisfaction', 'StockOptionLevel', 'TotalWorkingYears',
       'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany',
       'YearsInCurrentRole', 'YearsWithCurrManager'],
      dtype='object')
```

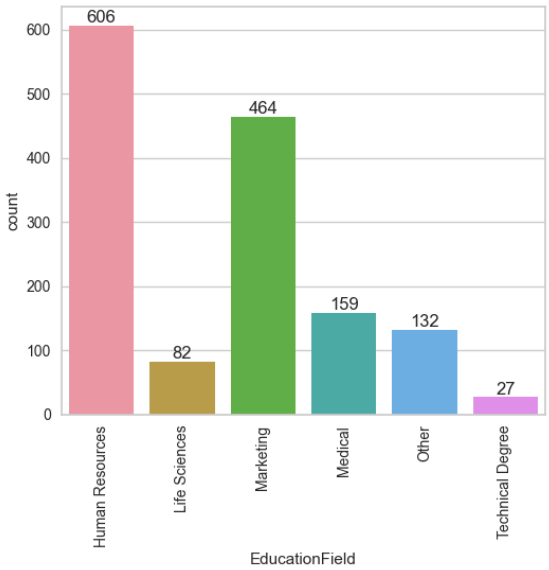
In [11]:
```
palette = ["#F72585", "#7209B7", "#3A0CA3", "#4361EE", "#4CC9F0"]
sns.set_palette(palette)
for i, col in enumerate(cat_columns):

    fig, axes = plt.subplots(1,2,figsize=(13,5))
    ax = sns.countplot(data=df,x=col , ax=axes[0])
    activities = [var for var in df[col].value_counts().sort_index().index]
    ax.set_xticklabels(activities,rotation=90)
```

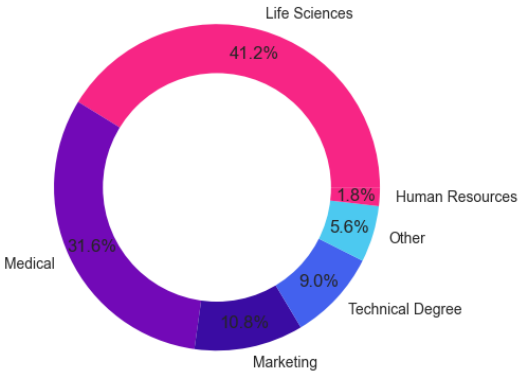
```
    for container in axes[0].containers:
        axes[0].bar_label(container)

    index = df[col].value_counts().index
    size = df[col].value_counts().values
    explode = (0.05,0.05)

    axes[1].pie(size , labels=index,autopct='%1.1f%%',pctdistance=0.85)
    centre_circle = plt.Circle((0,0),0.70,fc='white')
    fig = plt.gcf()

    fig.gca().add_artist(centre_circle)
    plt.suptitle(col,backgroundcolor='green',color='white',fontsize=15)

    plt.show()
```



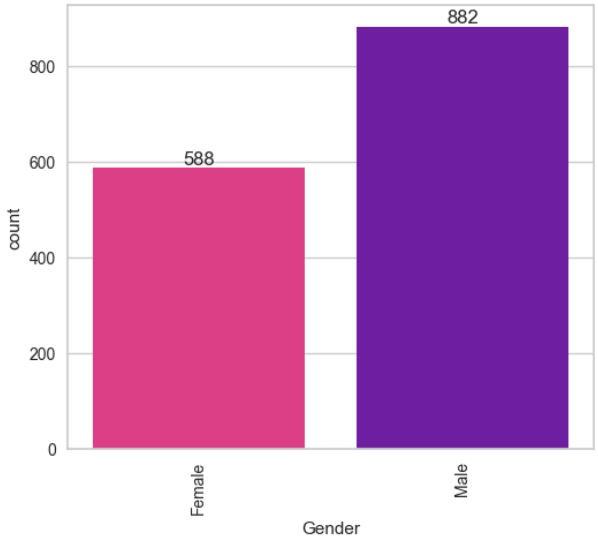


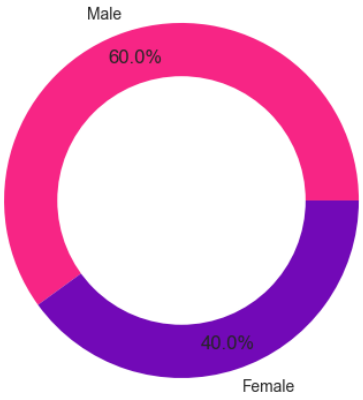
EducationField



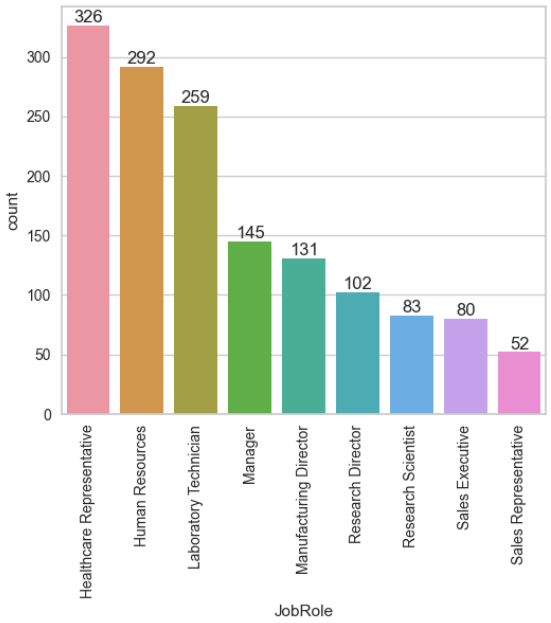


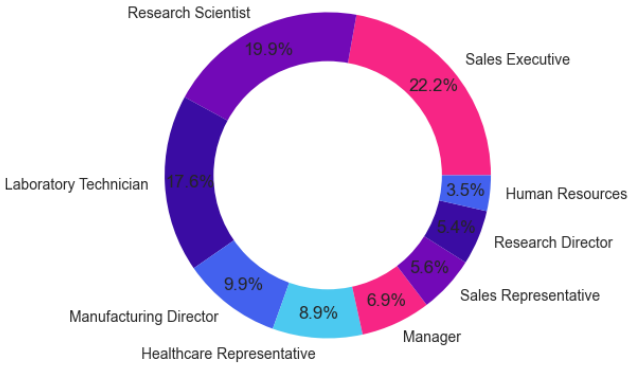
Gender



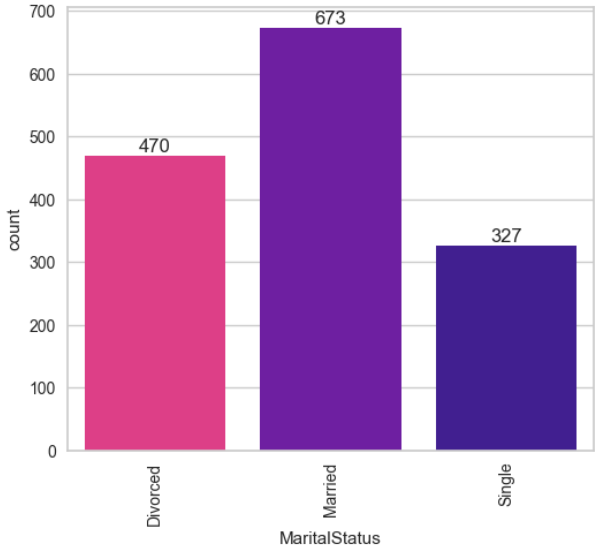


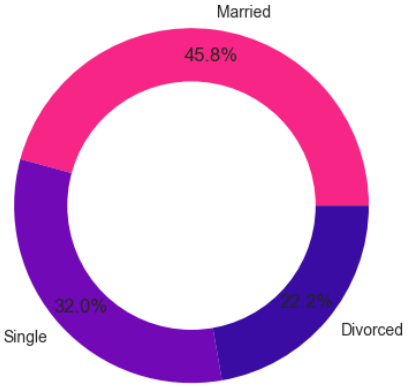
JobRole



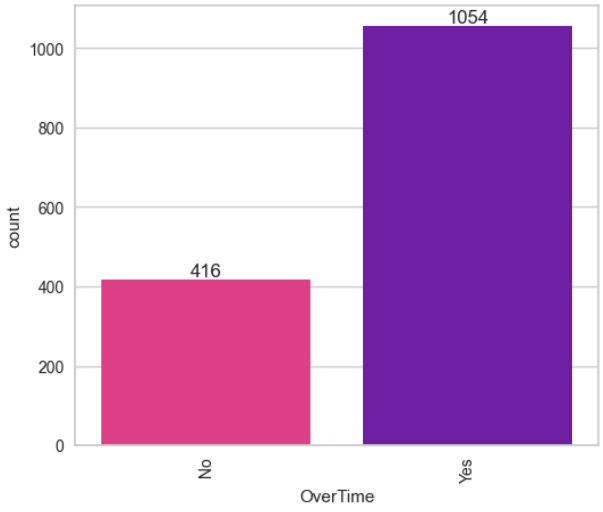


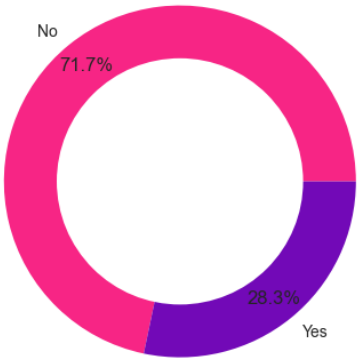
MaritalStatus





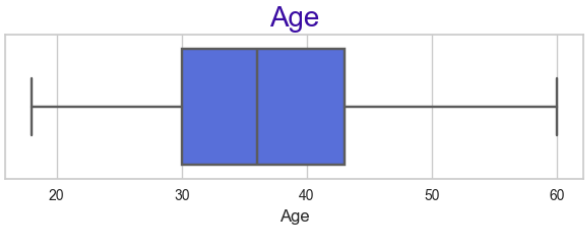
OverTime

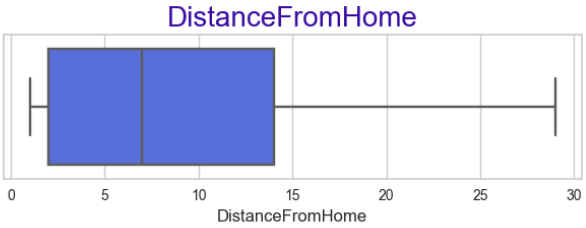




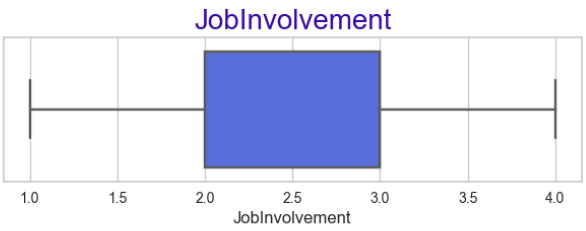
In [12]:

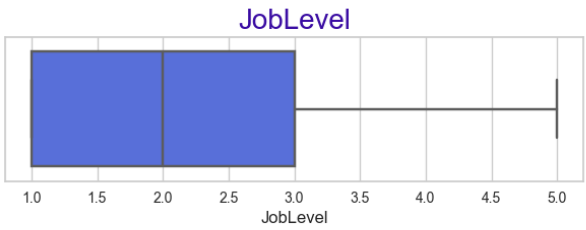
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(12, 30))
for idx, i in enumerate(num_columns):
    plt.subplot(13, 2, idx + 1)
    sns.boxplot(x=i, data=df,color="#4361EE")
    plt.title(i, color='#3A0CA3', fontsize=20)
    plt.xlabel(i, size=12)
plt.tight_layout()
plt.show()
```


Age
20
30
40
50
60
Age

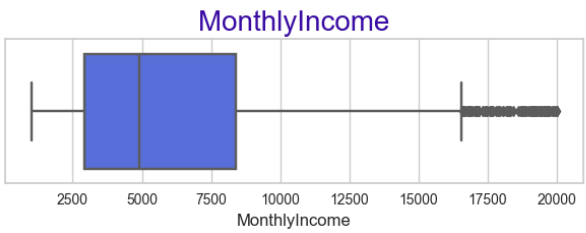

DistanceFromHome
0
5
10
15
20
25
30
DistanceFromHome


EnvironmentSatisfaction
1.0
1.5
2.0
2.5
3.0
3.5
4.0
EnvironmentSatisfaction

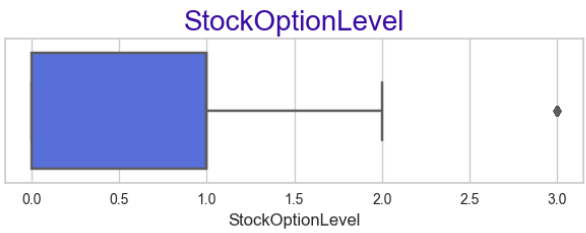

JobInvolvement
1.0
1.5
2.0
2.5
3.0
3.5
4.0
JobInvolvement

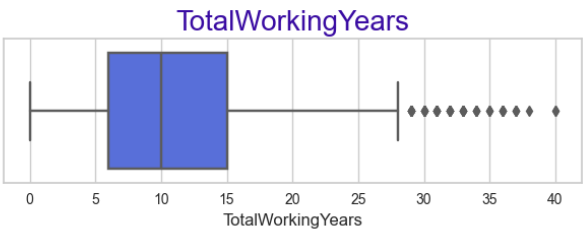

JobLevel
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
JobLevel


JobSatisfaction
1.0
1.5
2.0
2.5
3.0
3.5
4.0
JobSatisfaction

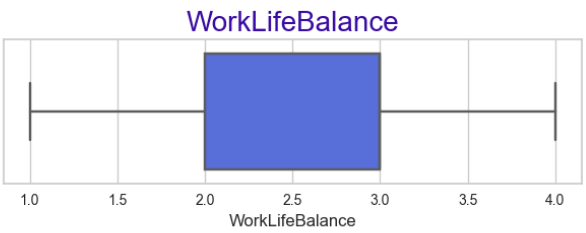

MonthlyIncome
2500
5000
7500
10000
12500
15000
17500
20000
MonthlyIncome


RelationshipSatisfaction
1.0
1.5
2.0
2.5
3.0
3.5
4.0
RelationshipSatisfaction

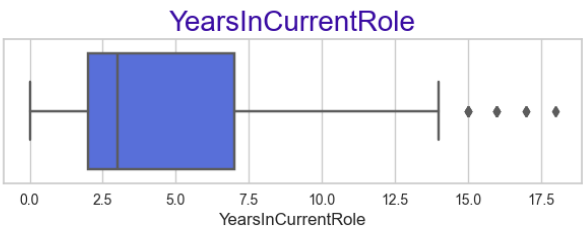

StockOptionLevel
0.0
0.5
1.0
1.5
2.0
2.5
3.0
StockOptionLevel

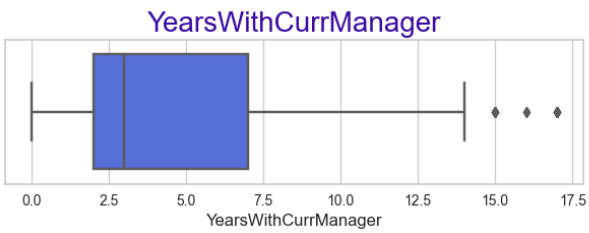

TotalWorkingYears
0
5
10
15
20
25
30
35
40
TotalWorkingYears


TrainingTimesLastYear
0
1
2
3
4
5
6
TrainingTimesLastYear


WorkLifeBalance
1.0
1.5
2.0
2.5
3.0
3.5
4.0
WorkLifeBalance

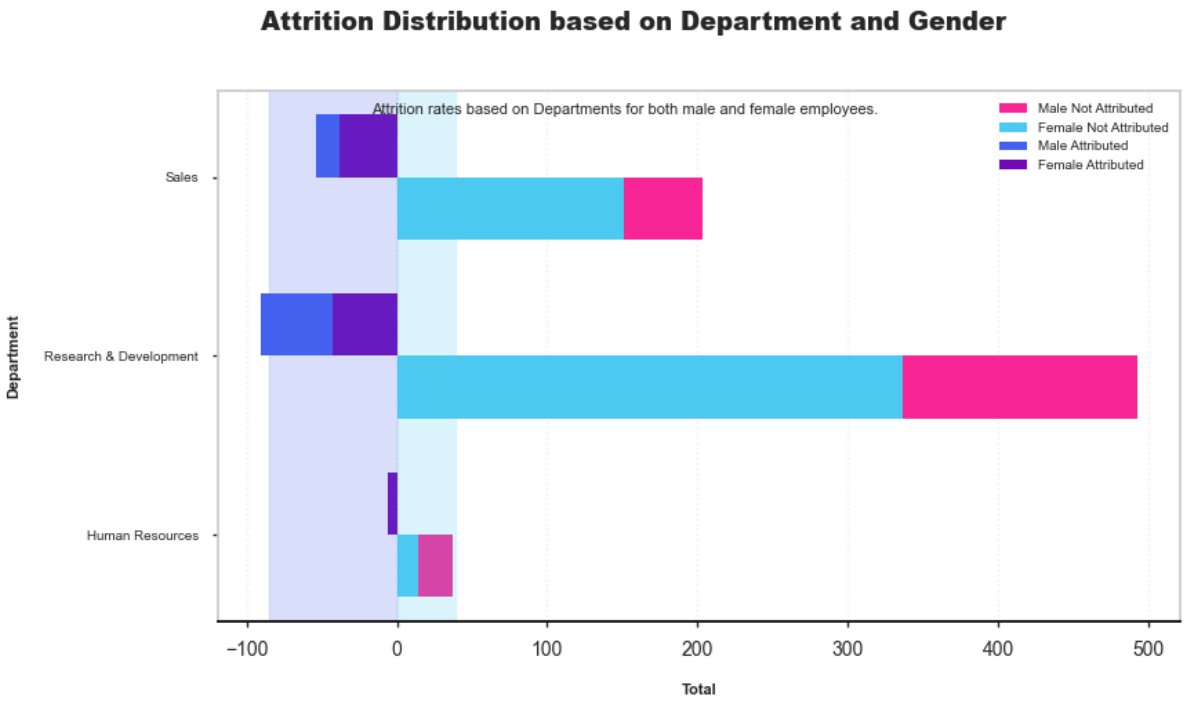

YearsAtCompany
0
5
10
15
20
25
30
35
40
YearsAtCompany


YearsInCurrentRole
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
YearsInCurrentRole


YearsWithCurrManager
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
YearsWithCurrManager

In [13]:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# EDA 1 Dataframes
df_eda1 = df[['Gender', 'Department', 'Attrition']]
df_eda1 = pd.DataFrame(df_eda1.groupby(['Gender', 'Attrition', 'Department']).size
df_eda1['total'] *= np.where(df_eda1['Attrition'] == 0, 1, -1)
df_eda1_m = df_eda1.query('Gender == "Male" & Attrition == 0')
df_eda1_f = df_eda1.query('Gender == "Female" & Attrition == 0')
df_eda1_ms = df_eda1.query('Gender == "Male" & Attrition == 1')
df_eda1_fs = df_eda1.query('Gender == "Female" & Attrition == 1')

# Plotting
fig, ax = plt.subplots(figsize=(9, 5))
bar_mns = plt.barh(np.arange(len(df_eda1_m)), df_eda1_m['total'], color='#F72595',
bar_fns = plt.barh(np.arange(len(df_eda1_f)), df_eda1_f['total'], color='#4CC9F0',
bar_ms = plt.barh(np.arange(len(df_eda1_ms)) + 0.35, df_eda1_ms['total'], color='#
bar_fs = plt.barh(np.arange(len(df_eda1_fs)) + 0.35, df_eda1_fs['total'], color='#

ax.set_yticks(np.arange(len(df_eda1.Department.unique())) + 0.35 / 2)
ax.set_yticklabels(df_eda1.Department.unique(), fontsize=7)
plt.xlabel('\nTotal', fontweight='bold', fontsize=8)
plt.ylabel('Department\n', fontweight='bold', fontsize=8)
plt.grid(axis='y', alpha=0, zorder=2)
plt.grid(axis='x', which='major', alpha=0.3, linestyle='dotted', zorder=1)

plt.axvspan(-85, 0, color='#4361EE', alpha=0.2)
plt.axvspan(40, 0, color='#4CC9F0', alpha=0.2)

plt.legend(fontsize=7)
plt.tick_params(bottom='on', length=3, width=1)
ax.spines['bottom'].set_color('black')

plt.suptitle('Attrition Distribution based on Department and Gender', x=0.16, y=0.
plt.title("Attrition rates based on Departments for both male and female employees

plt.show()
```



💡 Analysis of graphs

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

> - Attrition is the highest for both men and women from 18 to 35 years of age and gradually decreases.
> - As income increases, attrition decreases.
> - Attrition is much, much less in divorced women.
> - Attrition is higher for employees who usually travel than others, and this rate is higher for women than for men.
> - Attrition is the highest for those in level 1 jobs.
> - Women with the job position of manager, research director, and laboratory technician have almost no attrition.
> - Men with the position of sales expert have more attrition as compare to female.
> - Department of Research & Devlopment has highest attrition in terms both male and female as compared to other deparemnt

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 'Attrition' (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(['Attrition'], axis=1)
y = df['Attrition']

# --- Splitting Dataset ---
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_st
```

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 **MinMax 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]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.compose import ColumnTransformer

# --- Numerical Pipeline ---
num_pipeline = Pipeline([
    ('scaling', MinMaxScaler())
])

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

# --- Combine Both Pipelines into Transformer ---
preprocessor = ColumnTransformer([
    ('categorical', cat_pipeline, cat_columns),
    ('numerical', num_pipeline, num_columns)
], 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)
```

6.3 | Treating Dataset Imbalance ⚖️

Dataset Imbalance: Only **16%** represent churn employees, impacting model accuracy.

Solution: Synthetic Minority Over-sampling Technique (SMOTE)
SMOTE mitigates class imbalance by oversampling the minority class.

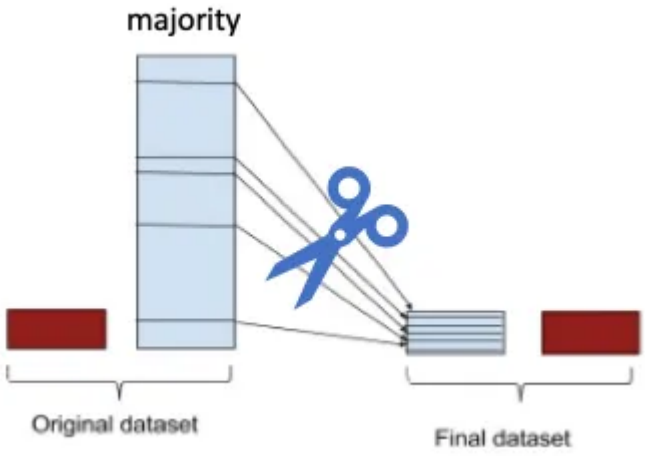
It generates synthetic samples, increasing minority class representation.

Benefits:
1. *Improved Learning:* Better understanding of minority class characteristics.

2. *Enhanced Predictions:* Reduces bias, improving predictive performance.

Outcome:
1. *Balanced Dataset:* Improves model generalization and accuracy.

2. *Business Impact:* Informed decisions on potential churn employees.



In [16]:
```
from imblearn.over_sampling import SMOTE
oversampler = SMOTE(random_state=0)
X_sm_train , y_sm_train = oversampler.fit_resample(X_train_process,y_train)
X_sm_test , y_sm_test = oversampler.fit_resample(X_test_process,y_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 [17]:
```
from sklearn.metrics import f1_score, precision_score, recall_score

color_yb = sns.color_palette("Paired")
color_line = 'red'
color = 'red'

def fit_ml_models(algo, algo_param, algo_name):
    # --- Algorithm Pipeline ---
```

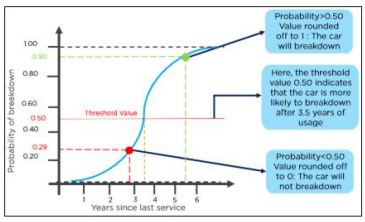
```
    algo = Pipeline([('algo', algo)])

    # --- 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_sm_train, y_sm_train)

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

    # --- Create Prediction for Train & Test ---
    y_pred_train = model.predict(X_sm_train)
    y_pred_test = model.predict(X_sm_test)

    # --- Calculate F1 score ---
    f1_score_train = f1_score(y_sm_train, y_pred_train)
    f1_score_test = f1_score(y_sm_test, y_pred_test)

    # --- Calculate Precision ---
    precision_train = precision_score(y_sm_train, y_pred_train)
    precision_test = precision_score(y_sm_test, y_pred_test)

    # --- Calculate Recall ---
    recall_train = recall_score(y_sm_train, y_pred_train)
    recall_test = recall_score(y_sm_test, y_pred_test)

    # --- 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 ---
    print(">> Best Score: " + clr.start + "{:.3f}".format(best_score) + clr.end)

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

    # --- Classification Report ---
    print("\n" + clr.start + f".:. Classification Report for {algo_name} .:." + cl
    print(classification_report(y_sm_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_sm_train, y_sm_train)
conf_matrix.score(X_sm_test, y_sm_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=c
logrocauc.fit(X_sm_train, y_sm_train)
logrocauc.score(X_sm_test, y_sm_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', **
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, b
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=c
lcurve.fit(X_sm_train, y_sm_train)
lcurve.finalize()
lcurve.ax.set_title('Learning Curve\n', **title_style)
lcurve.ax.tick_params(axis='both', labelsize=10, bottom='on', left='on', **tic
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, borc
lcurve.ax.set_xlabel('\nTraining Instances', **xy_label)
lcurve.ax.set_ylabel('Scores\n', **xy_label)

try:
    feat_importance = FeatureImportances(best_estimator, labels=columns_list_c
    feat_importance.fit(X_sm_train, y_sm_train)
    feat_importance.finalize()
    feat_importance.ax.set_title('Feature Importances (Top 5 Features)\n', **t
    feat_importance.ax.tick_params(axis='both', labelsize=10, bottom='on', lef
    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, i
        prec_curve.fit(X_sm_train, y_sm_train)
        prec_curve.score(X_sm_test, y_sm_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='or
        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, fo
    plt.gcf().text(0.88, 0.02, 'kaggle.com/darshanpathak12', style='italic', fonts
    plt.tight_layout()

    return acc_score_train, acc_score_test, best_score, f1_score_train, f1_score_t
```

7.1 | Logistic Regression

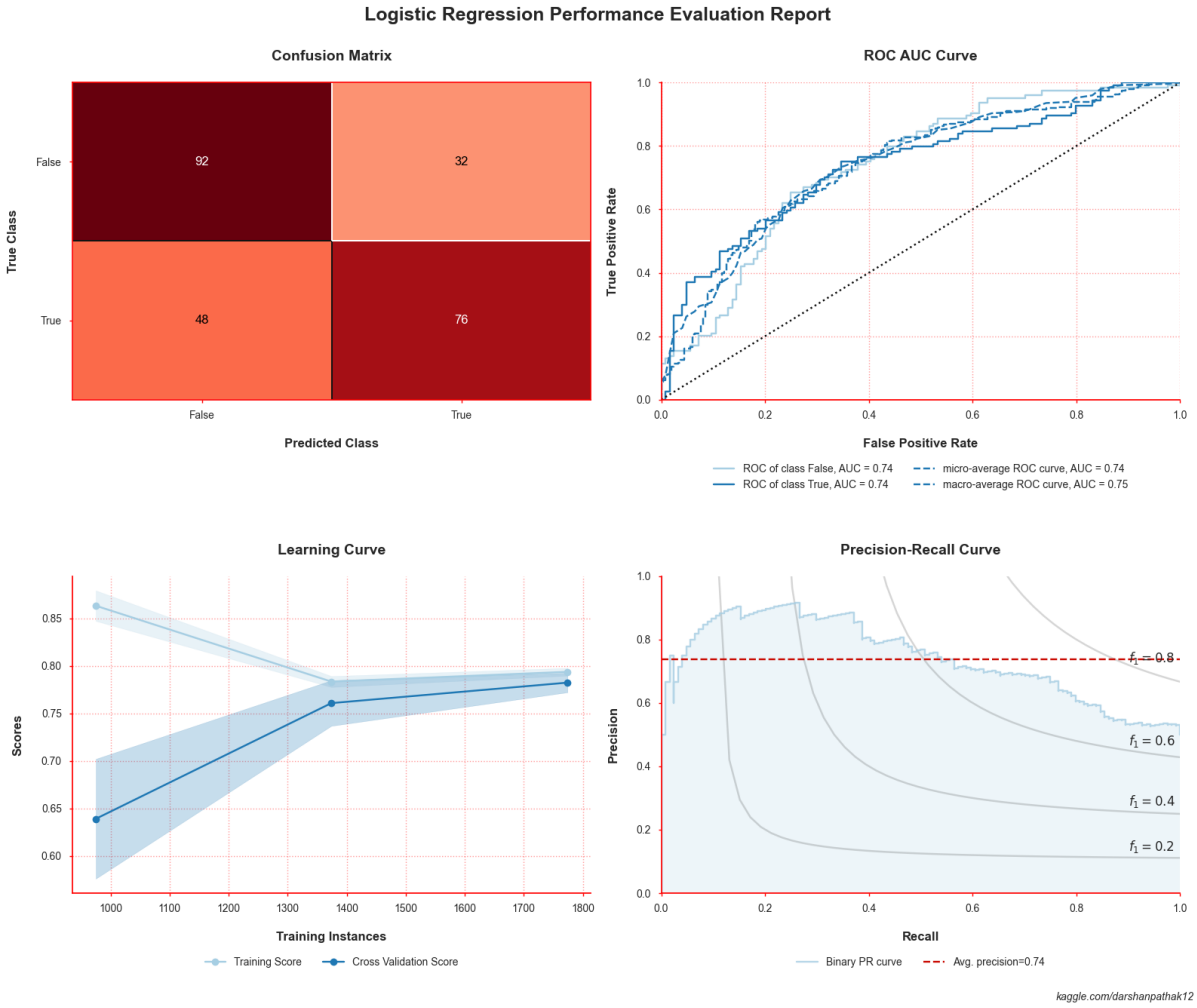
> **Logistic regression** is a statistical method that is used for building machine learning models where **the dependent variable is dichotomous: i.e. binary**. Logistic regression is used to describe data and **the relationship between one dependent variable and one or more independent variables**.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



In [18]:
```
# --- Logistic Regression Parameters ---
parameter_lr = {"algo__solver": ["lbfgs", "saga", "newton-cg"]
                , "algo__C": [0.1, 0.2, 0.5, 0.8]}
```

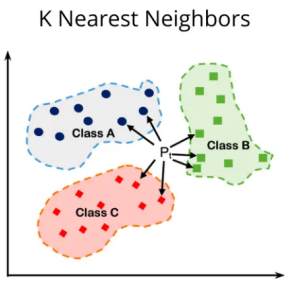
```python
# --- Logistic Regression Algorithm ---
algo_lr = LogisticRegression(penalty="l2", random_state=42, n_jobs=-1)

# --- Applying Logistic Regression ---
acc_score_train_lr, acc_score_test_lr, best_score_lr,f1_score_train_lr,f1_score_te
```

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

>> Best Parameters: {'algo__C': 0.8, 'algo__solver': 'saga'}
>> Best Score: 0.785

.:. Train and Test Accuracy Score for Logistic Regression .:.
        >> Train Accuracy: 79.31%
        >> Test Accuracy: 67.74%

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

           0       0.66      0.74      0.70       124
           1       0.70      0.61      0.66       124

    accuracy                           0.68       248
   macro avg       0.68      0.68      0.68       248
weighted avg       0.68      0.68      0.68       248
```



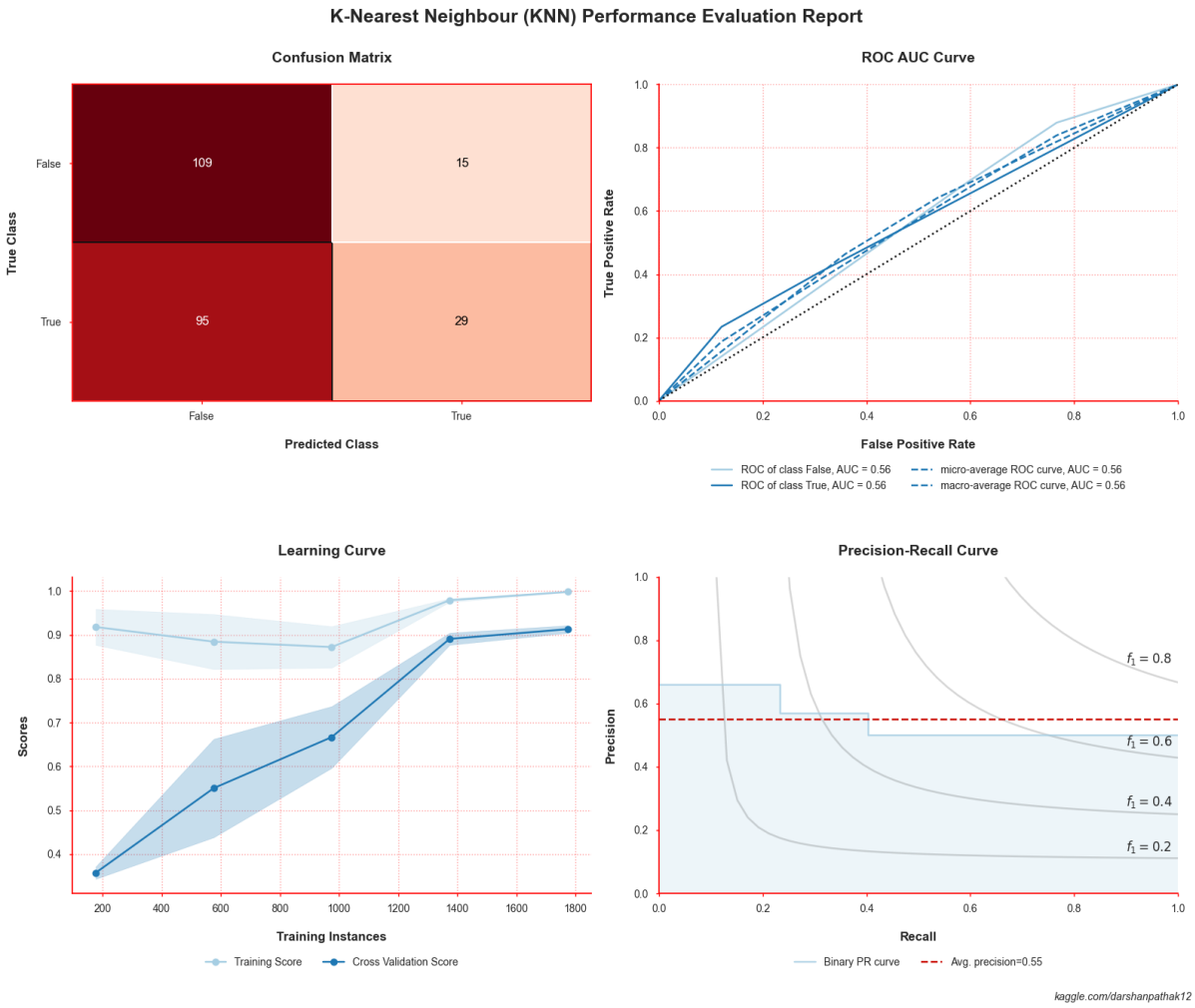
7.2 | K-Nearest Neighbour (KNN)

> **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.



In [19]:
```
# --- 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,f1_score_train_knn,f1_scor
```

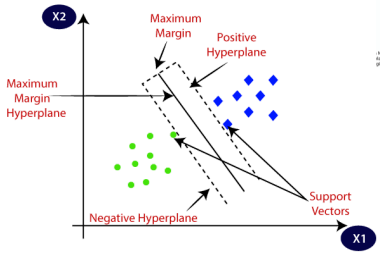
```
.:. Fitting K-Nearest Neighbour (KNN) .:.
Fitting 10 folds for each of 16 candidates, totalling 160 fits

>> Best Parameters: {'algo__leaf_size': 1, 'algo__n_neighbors': 2}
>> Best Score: 0.923

.:. Train and Test Accuracy Score for K-Nearest Neighbour (KNN) .:.
        >> Train Accuracy: 100.00%
        >> Test Accuracy: 55.65%

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

           0       0.53      0.88      0.66       124
           1       0.66      0.23      0.35       124

    accuracy                           0.56       248
   macro avg       0.60      0.56      0.50       248
weighted avg       0.60      0.56      0.50       248
```



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.

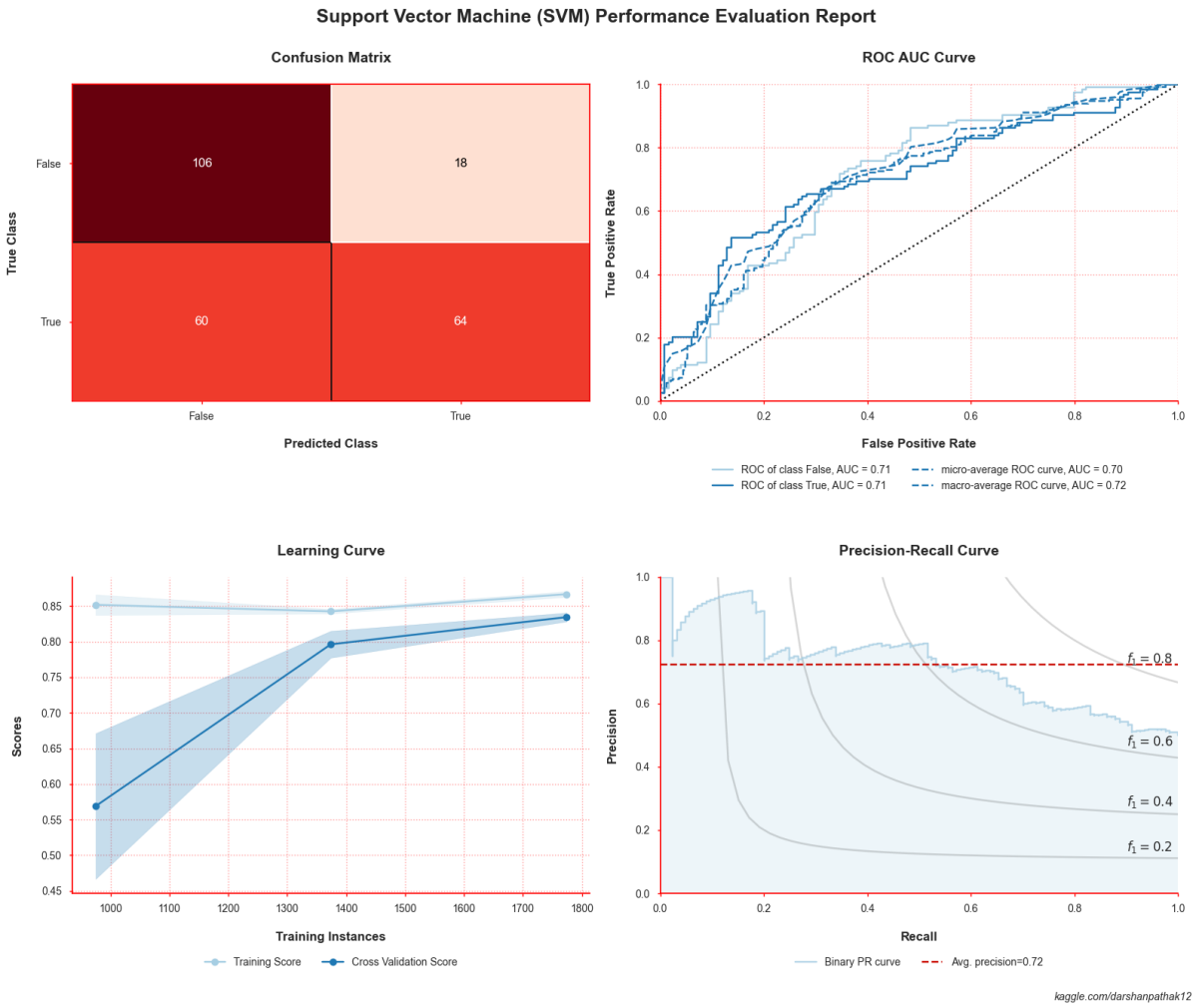
SVM chooses the **extreme points/vectors** that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.



SVM by JavaTPoint

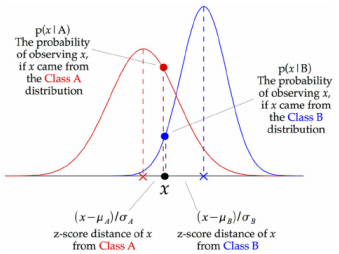
In [20]:
```
# --- SVM Parameters ---
parameter_svc = [
     {'algo__C': [0.6], 'algo__degree': [2], 'algo__kernel': ['poly']}
]

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

# --- Applying SVM ---
acc_score_train_svc, acc_score_test_svc, best_score_svc,f1_score_train_svc,f1_scor
```

```
.:. Fitting Support Vector Machine (SVM) .:.
Fitting 10 folds for each of 1 candidates, totalling 10 fits

>> Best Parameters: {'algo__C': 0.6, 'algo__degree': 2, 'algo__kernel': 'poly'}
>> Best Score: 0.842

.:. Train and Test Accuracy Score for Support Vector Machine (SVM) .:.
        >> Train Accuracy: 86.97%
        >> Test Accuracy: 68.55%

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

           0       0.64      0.85      0.73       124
           1       0.78      0.52      0.62       124

    accuracy                           0.69       248
   macro avg       0.71      0.69      0.68       248
weighted avg       0.71      0.69      0.68       248
```



7.4 | Gaussian Naive Bayes

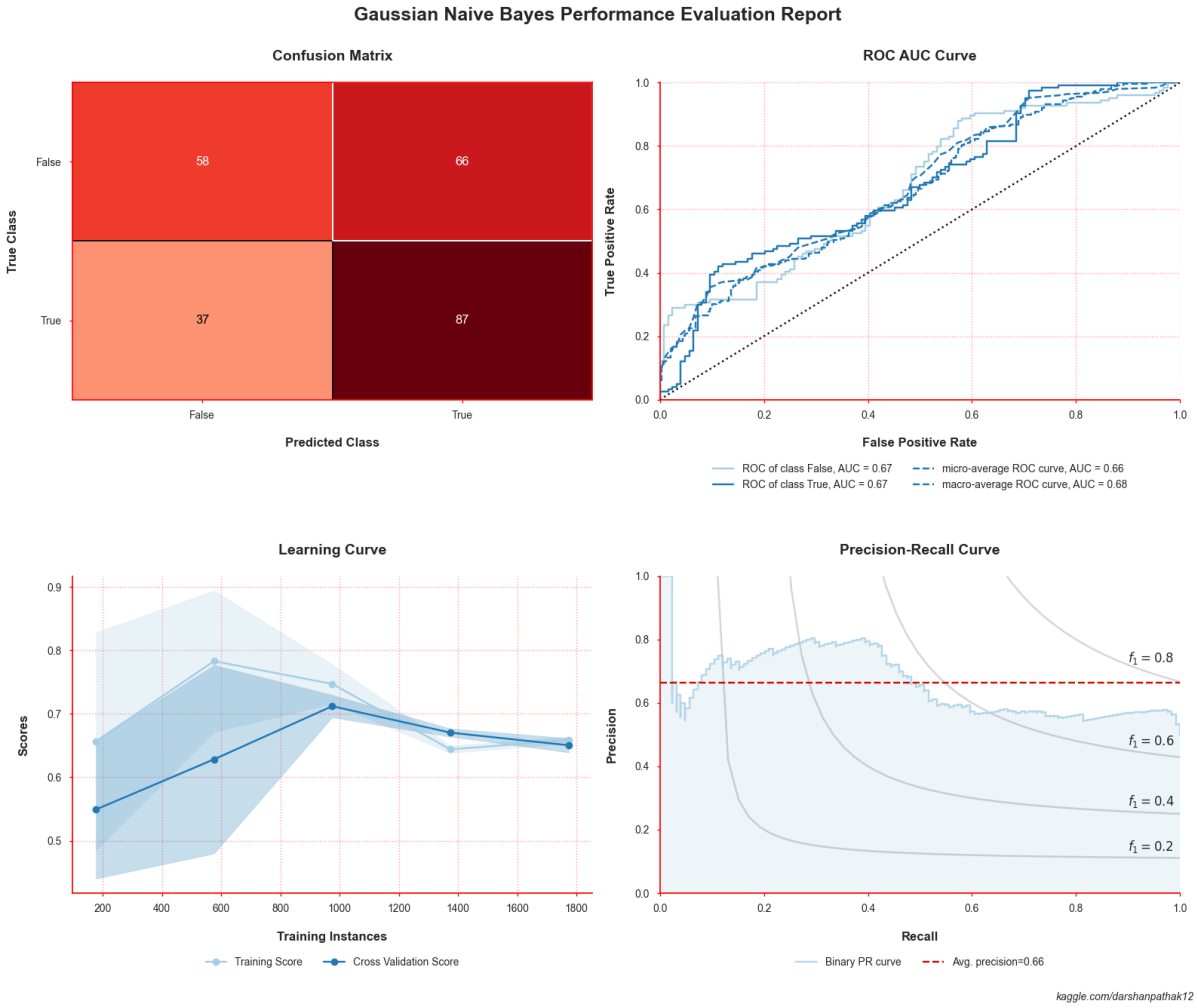
> **Naive Bayes Classifiers** are based on the Bayes Theorem, which **one assumption taken is the strong independence assumptions between the features**. These classifiers assume that the value of a particular feature is independent of the value of any other feature. In a supervised learning situation, 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.



Gaussian Naive Bayes by OpenGenus

In [21]:
```
# --- 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,f1_score_train_gnb,f1_scor
```

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

>> Best Parameters: {'algo__var_smoothing': 0.01}
>> Best Score: 0.662

.:. Train and Test Accuracy Score for Gaussian Naive Bayes .:.
        >> Train Accuracy: 66.91%
        >> Test Accuracy: 58.47%

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

           0       0.61      0.47      0.53       124
           1       0.57      0.70      0.63       124

    accuracy                           0.58       248
   macro avg       0.59      0.58      0.58       248
weighted avg       0.59      0.58      0.58       248
```

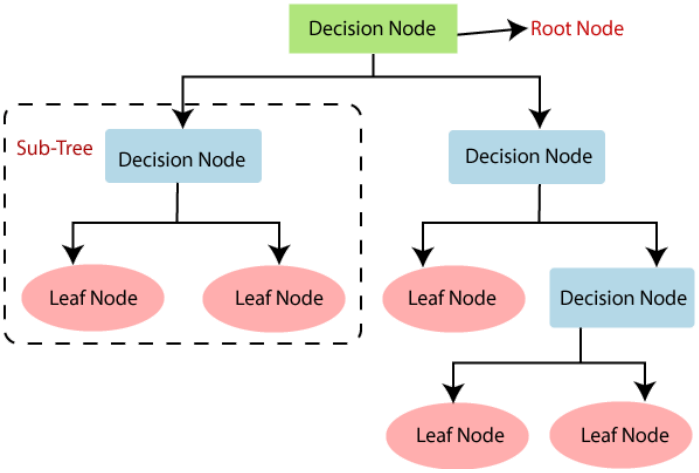


7.5 | Decision Tree

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.**

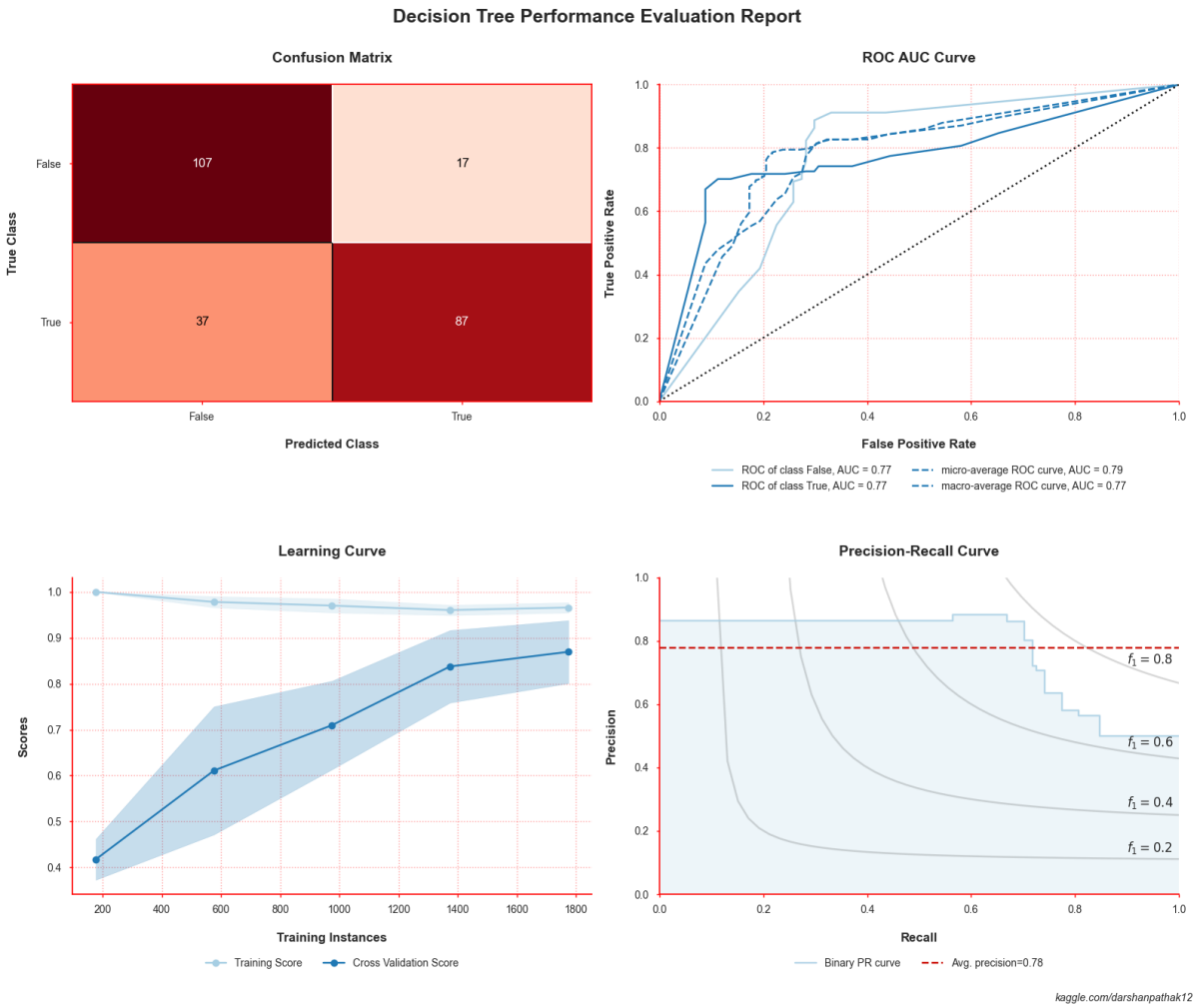
In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those

decisions and do not contain any further branches.



In [22]:
```
# Decision Tree Parameters
parameter_dt = {
    "algo__max_depth": np.arange(5,15,1),
}

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

# Applying Decision Tree
acc_score_train_dt, acc_score_test_dt, best_score_dt,f1_score_train_dt,f1_score_te
```

```
.:. Fitting Decision Tree .:.
Fitting 10 folds for each of 10 candidates, totalling 100 fits

>> Best Parameters: {'algo__max_depth': 9}
>> Best Score: 0.881

.:. Train and Test Accuracy Score for Decision Tree .:.
        >> Train Accuracy: 96.26%
        >> Test Accuracy: 78.23%

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

           0       0.74      0.86      0.80       124
           1       0.84      0.70      0.76       124

    accuracy                           0.78       248
   macro avg       0.79      0.78      0.78       248
weighted avg       0.79      0.78      0.78       248
```

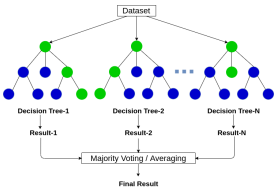


7.6 | Random Forest

> **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.

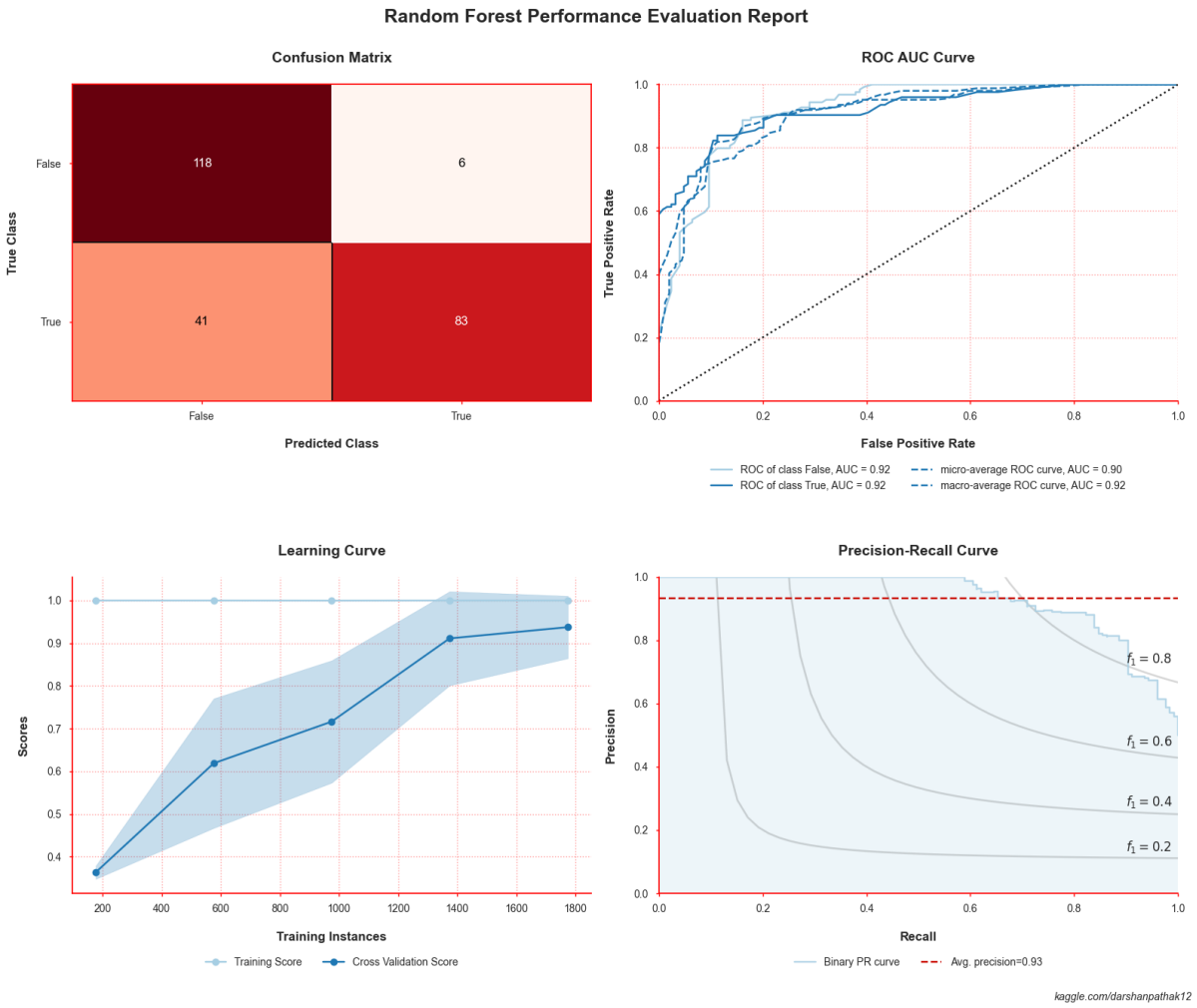
> **Random Forest** is a tree-based machine learning algorithm that **leverages the power of multiple decision trees for making decisions**. Each individual tree in the random forest spits out a class prediction and the class with the most votes

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 [23]:
```
# --- Random Forest Parameters ---
parameter_rf = {"algo__max_depth": np.arange(30,32, 1)}

# --- Random Forest Algorithm ---
algo_rf = RandomForestClassifier(random_state=99, n_jobs=-1)

# --- Applying Random Forest ---
acc_score_train_rf, acc_score_test_rf, best_score_rf,f1_score_train_rf,f1_score_te
```

```
.:. Fitting Random Forest .:.
Fitting 10 folds for each of 2 candidates, totalling 20 fits

>> Best Parameters: {'algo__max_depth': 30}
>> Best Score: 0.939

.:. Train and Test Accuracy Score for Random Forest .:.
        >> Train Accuracy: 100.00%
        >> Test Accuracy: 81.05%

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

           0       0.74      0.95      0.83       124
           1       0.93      0.67      0.78       124

    accuracy                           0.81       248
   macro avg       0.84      0.81      0.81       248
weighted avg       0.84      0.81      0.81       248
```



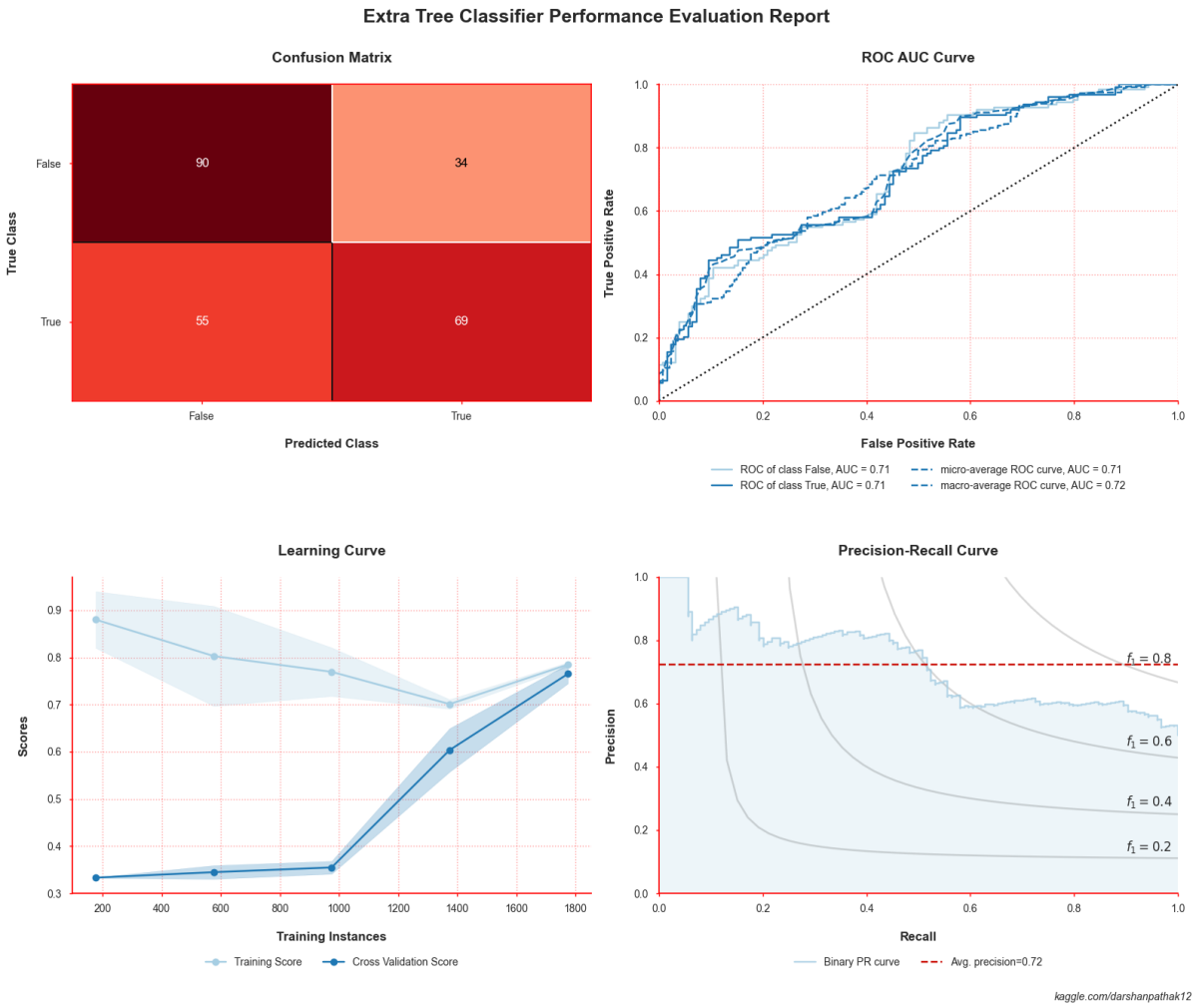
7.7 | Extra Tree Classifier

> **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 decisions.
>
>  ETC

In [24]:
```
# --- Extra Tree Parameters ---
parameter_et = {"algo__max_depth": [2, 3]
    , "algo__max_leaf_nodes": [3, 5, 7]}
```

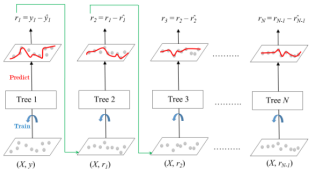
```python
# --- Extra Tree Algorithm ---
algo_et = ExtraTreesClassifier(random_state=42, n_jobs=-1)

# --- Applying Extra Tree ---
acc_score_train_et, acc_score_test_et, best_score_et,f1_score_train_et,f1_score_te
```

```
.:. 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': 7}
>> Best Score: 0.770

.:. Train and Test Accuracy Score for Extra Tree Classifier .:.
        >> Train Accuracy: 78.09%
        >> Test Accuracy: 64.11%

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

           0       0.62      0.73      0.67       124
           1       0.67      0.56      0.61       124

    accuracy                           0.64       248
   macro avg       0.65      0.64      0.64       248
weighted avg       0.65      0.64      0.64       248
```



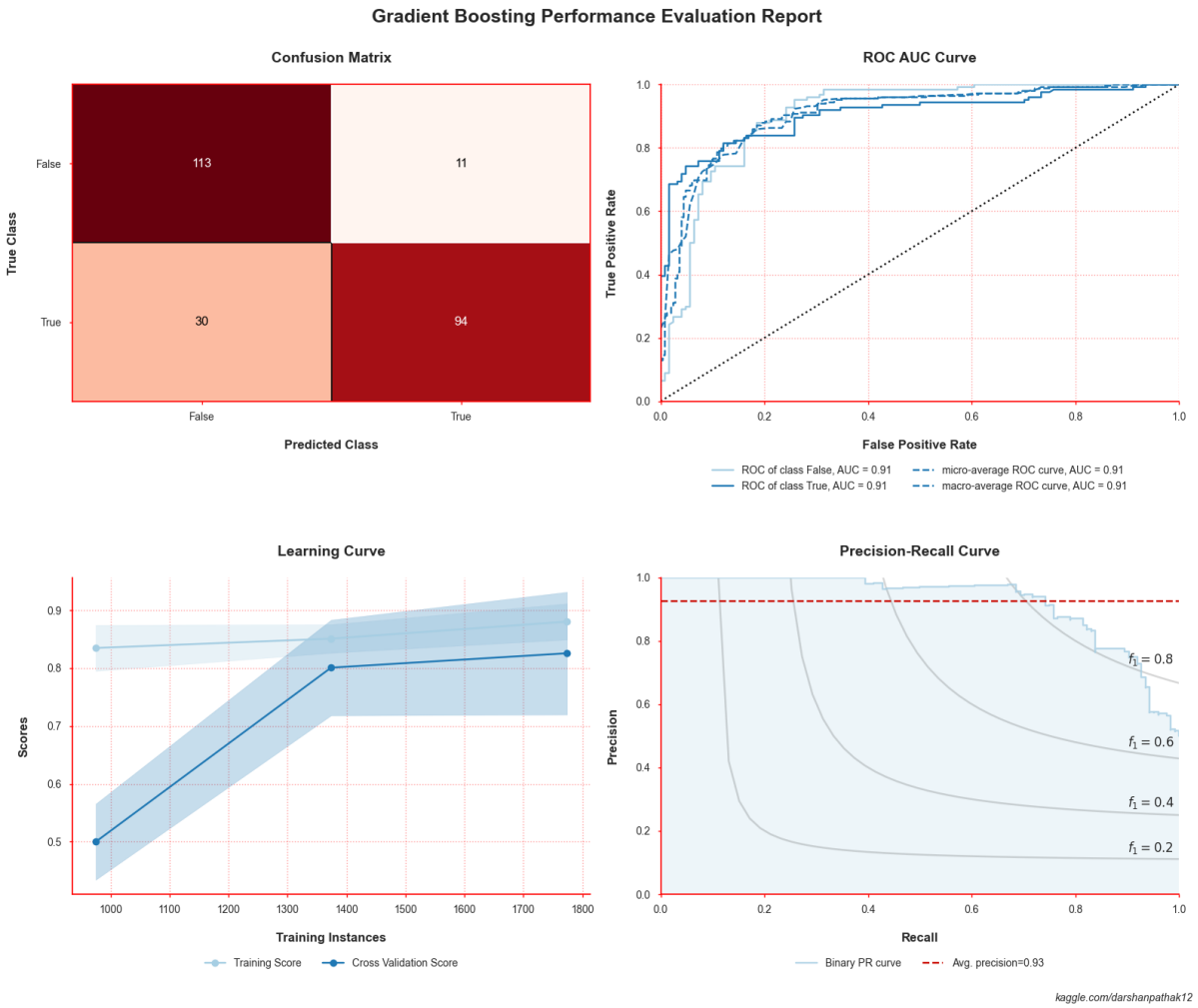
7.8 | Gradient Boosting

> **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.



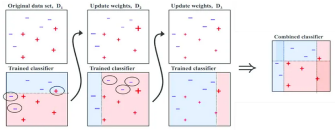
In [25]:
```
# --- Gradient Boosting Parameters ---
parameter_gb = {
    "algo__learning_rate": [0.1, 0.3, 0.5]
    , "algo__n_estimators": [2, 4, 6]
    , "algo__min_weight_fraction_leaf": [0.1, 0.2, 0.5]
}

# --- 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,f1_score_train_gb,f1_score_te
```

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

>> Best Parameters: {'algo__learning_rate': 0.5, 'algo__min_weight_fraction_leaf':
0.1, 'algo__n_estimators': 6}
>> Best Score: 0.855

.:. Train and Test Accuracy Score for Gradient Boosting .:.
        >> Train Accuracy: 87.65%
        >> Test Accuracy: 83.47%

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

           0       0.79      0.91      0.85       124
           1       0.90      0.76      0.82       124

    accuracy                           0.83       248
   macro avg       0.84      0.83      0.83       248
weighted avg       0.84      0.83      0.83       248
```



7.9 | AdaBoost

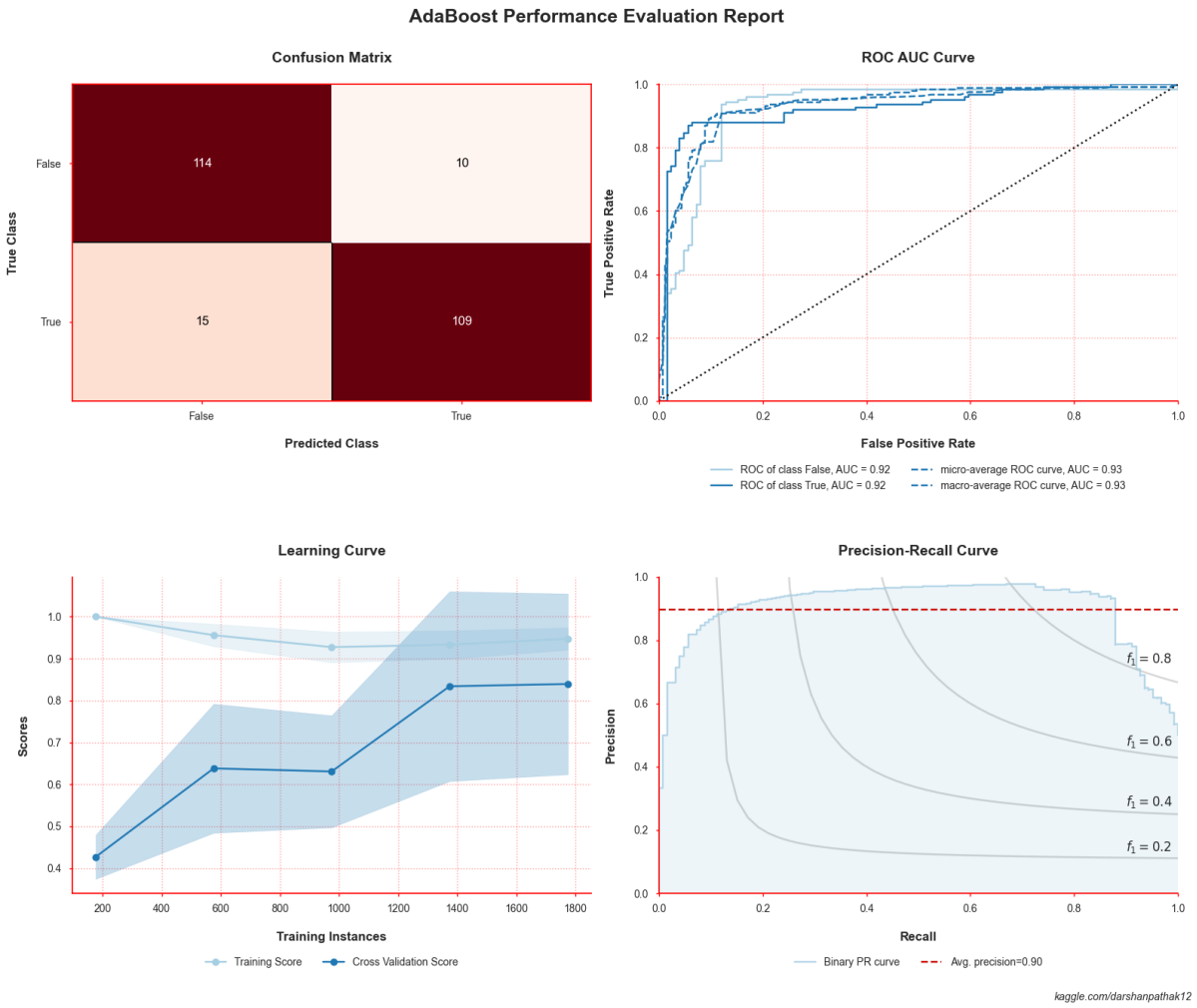
> **AdaBoost** also called **Adaptive Boosting** is a technique in Machine Learning used as an Ensemble Method. The most common algorithm used with AdaBoost is **decision trees with one level** that means with Decision trees with only 1 split. These trees are also called **Decision Stumps**.
> **AdaBoost 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 [26]:
```
# --- AdaBoost Parameters ---
parameter_ab = {
    "algo__n_estimators": [100,200,300]
    , "algo__learning_rate": [0.1,0.75,0.1]
}
```

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

# --- Applying AdaBoost ---
acc_score_train_ab, acc_score_test_ab, best_score_ab,f1_score_train_ab,f1_score_te
```

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

>> Best Parameters: {'algo__learning_rate': 0.75, 'algo__n_estimators': 200}
>> Best Score: 0.906

.:. Train and Test Accuracy Score for AdaBoost .:.
        >> Train Accuracy: 93.91%
        >> Test Accuracy: 89.92%

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

           0       0.88      0.92      0.90       124
           1       0.92      0.88      0.90       124

    accuracy                           0.90       248
   macro avg       0.90      0.90      0.90       248
weighted avg       0.90      0.90      0.90       248
```



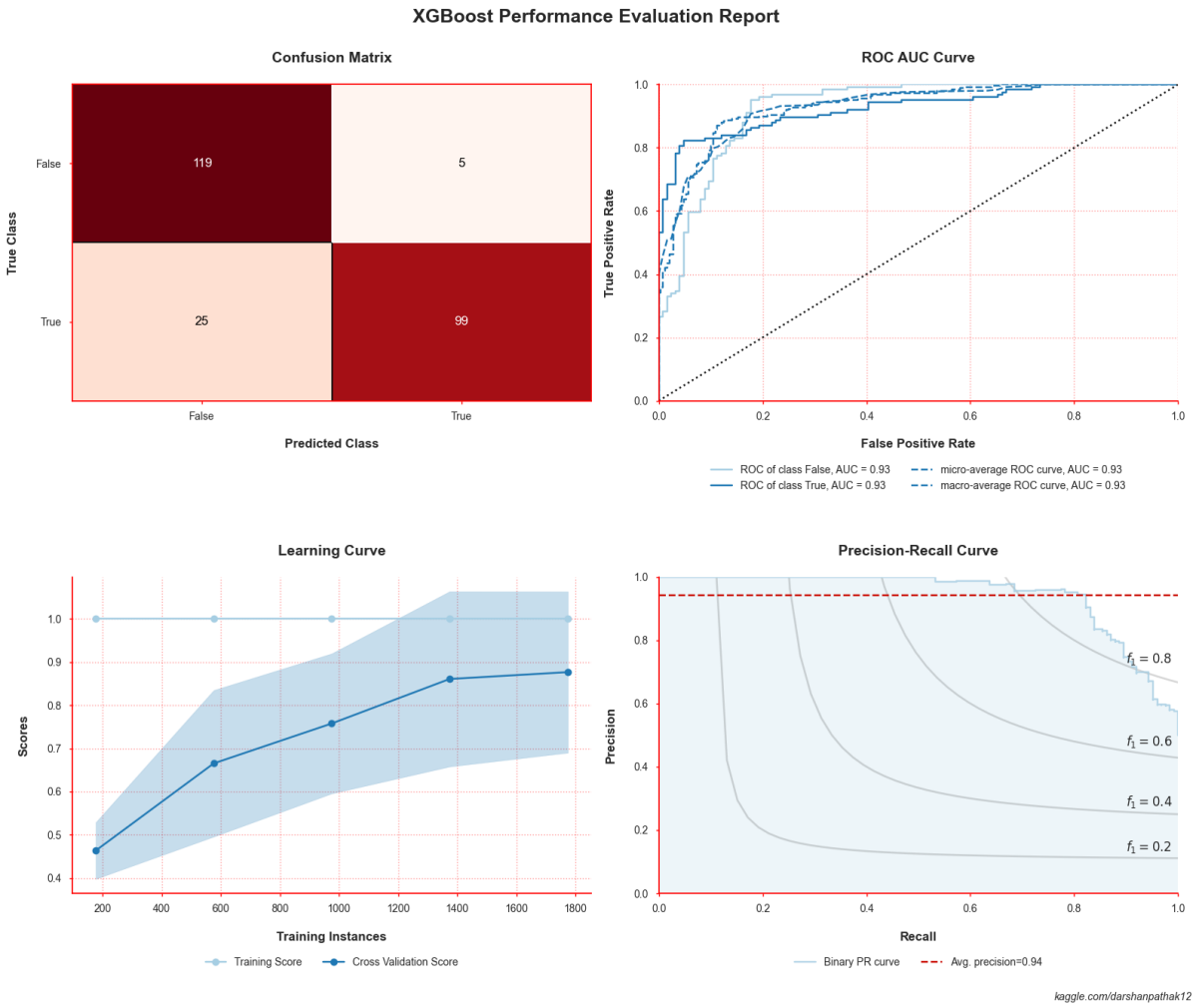
7.10 | XgBoost

> **XGBoost** stands for **Extreme Gradient Boosting** and is a powerful and efficient implementation of the gradient boosting framework. It is widely used for supervised learning tasks, particularly in structured/tabular data problems, and excels in both speed and performance. XGBoost builds a series of decision trees sequentially, with each tree aiming to correct the errors of its predecessor. Unlike traditional gradient boosting methods, XGBoost employs a more regularized model structure, which helps prevent overfitting and enhances generalization. It also incorporates advanced features like parallel computing, tree pruning, and handling missing values, making it a popular choice in machine learning competitions and real-world applications.
>
> 
>

In [27]:
```
# --- XGBoost Parameters ---
import xgboost as xgb
parameter_xgb = {
    "algo__n_estimators": [100],
```

```
    "algo__learning_rate": [0.75]
}

# --- XGBoost Algorithm ---
algo_xgb = xgb.XGBClassifier(random_state=1)

# --- Applying XGBoost ---
acc_score_train_xgb, acc_score_test_xgb, best_score_xgb,f1_score_train_xgb,f1_scor
```

```
.:. Fitting XGBoost .:.
Fitting 10 folds for each of 1 candidates, totalling 10 fits

>> Best Parameters: {'algo__learning_rate': 0.75, 'algo__n_estimators': 100}
>> Best Score: 0.915

.:. Train and Test Accuracy Score for XGBoost .:.
        >> Train Accuracy: 100.00%
        >> Test Accuracy: 87.90%

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

           0       0.83      0.96      0.89       124
           1       0.95      0.80      0.87       124

    accuracy                           0.88       248
   macro avg       0.89      0.88      0.88       248
weighted avg       0.89      0.88      0.88       248
```



7.11 | Model Comparison 👀

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

In [28]:
```
# --- 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_trai
                                               acc_score_train_dt, acc_score_trai
                          , 'Accuracy Test': [acc_score_test_lr, acc_score_test_k
                                              acc_score_test_dt, acc_score_test_r
                          , 'Best Score': [best_score_lr, best_score_knn, best_sc
                                           best_score_et, best_score_gb, best_scc
                          , 'F1 Score Train': [f1_score_train_lr, f1_score_train_
                                               f1_score_train_dt, f1_score_train_
                          , 'F1 Score Test': [f1_score_test_lr, f1_score_test_knr
                                              f1_score_test_dt, f1_score_test_rf
                          , 'Precision Score Train': [precision_train_xgb, precis
                                                      precision_train_dt, precision_trai
                          , 'Precision Score Test': [precision_test_lr, precisior
```

```
                                                precision_test_dt, precision_test_
                                , 'Recall Score Train': [recall_train_lr, recall_train_
                                                recall_train_dt, recall_train_rf,
                                , 'Recall Score Test': [recall_test_lr, recall_test_knr
                                                recall_test_dt, recall_test_rf, re
                                })

# --- Create Comparison Table ---
print(clr.start+f".:. Models Comparison .:."+clr.end)
print(clr.color+'*' * 26)
df_compare.sort_values(by=['Accuracy Test','F1 Score Test','Best Score'], ascendir
```

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

Out[28]:

	Model	Accuracy Train	Accuracy Test	Best Score	F1 Score Train	F1 Score Test	Precision Score Train	Precision Score Test
0	AdaBoost	93.913000	89.919000	0.905800	0.938328	0.897119	0.950926	0.915966
1	XGBoost	100.000000	87.903000	0.915300	1.000000	0.868421	1.000000	0.951923
2	Gradient Boosting	87.647000	83.468000	0.855300	0.873500	0.820961	0.894986	0.895238
3	Random Forest	100.000000	81.048000	0.938700	1.000000	0.779343	1.000000	0.932584
4	Decision Tree	96.258000	78.226000	0.880600	0.961909	0.763158	0.979439	0.836538
5	Support Vector Machine	86.970000	68.548000	0.842200	0.873854	0.621359	0.846870	0.780488
6	Logistic Regression	79.306000	67.742000	0.785400	0.795363	0.655172	1.000000	0.703704
7	Extra Tree Classifier	78.088000	64.113000	0.770500	0.782648	0.607930	0.776398	0.669903
8	Gaussian NB	66.907000	58.468000	0.662300	0.715504	0.628159	0.627464	0.568627
9	K-Nearest Neighbour	100.000000	55.645000	0.923400	1.000000	0.345238	1.000000	0.659091

8. | Miscellaneous 🧪

This section focuses onbrk>creating a complete pipelinebrk>, starting from data processing to a machine learning pipeline, using the best model concluded in the previous section andbrk>exporting it to `joblib` and `pickle (.pkl)` filesbrk>. Besides that,brk>test dataset predicted results would also be exportedbrk> along with actual results in CSV and JSON files. Moreover, this section will alsobrk>make predictions on

dummy databrk> (data generated using Python functions) andbrk>export them to CSV and JSON filesbrk>.

In [29]:
```
# --- Complete Pipeline: Preprocessor & RF ---
xgb_pipeline = Pipeline([
    ('preprocessor', preprocessor)
    , ('algo', xgb.XGBClassifier(random_state=1))
])

# --- Save Complete Pipeline (joblib and pickle) ---
file_name = 'pipeline_employee_churn_xgboost_pathakdarshan'
for ext in ['joblib', 'pkl']:
    joblib.dump(xgb_pipeline, f'Pipeline/{file_name}.{ext}')
```

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**:

> - **XGBoost is the best model** out of 10 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).
> - **The prediction results on test 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 is performing advanced hyperparameter tuning experiments to obtain higher accuracy.

Thank You