**Model Performance Metric**

**Performance measurement** is an indispensable process in any machine-learning project. Our machine learning process may not always result in an optimum model with the expected accuracy. Hence, performance measurement is needed here to evaluate the effectiveness of a trained model in prediction.

There are a great variety of metrics which are used in performance measurement but in general; they can be categorised based on the model type, 1) Classifier or 2) Regressor. In this article, we will only focus on the classifier type measurement by introducing seven common performance metrics used in a classification project. The seven metrics are as below:

1. **Accuracy score**
2. **Confusion matrix**
3. **Precision**
4. **Recall**
5. **F1 Score**
6. **ROC Curve**
7. **AUROC**

However, there are other classification performance metrics out there that you will exposed to in this tutorial, you should look these up during your free time. While it may take a while to understand the underlying concept of some performance metrics above, the good news is that the implementation of those metrics has never been easier with [PyCaret](https://pycaret.gitbook.io/docs/) **a Python Machine Learning Library that actually optimizes the business objective and improves Return on Investment (ROI).**

PyCaret is an open-source, low-code machine learning library in Python that automates machine learning workflows. It is an end-to-end machine learning and model management tool that exponentially speeds up the experiment cycle and makes you more productive.

In the sections below, the concept of each performance metric will be explained by walking through a simple binary classification project based on the customer churn dataset. The aim of this classification is to predict if a customer is going to churn to a different service provider.

To ensure you can follow the material, you are recommended to update your Python package with

**# install pycaret**!pip install pycaret

Some functionalities presented here may not be supported in an earlier used package of Scikit-learn.

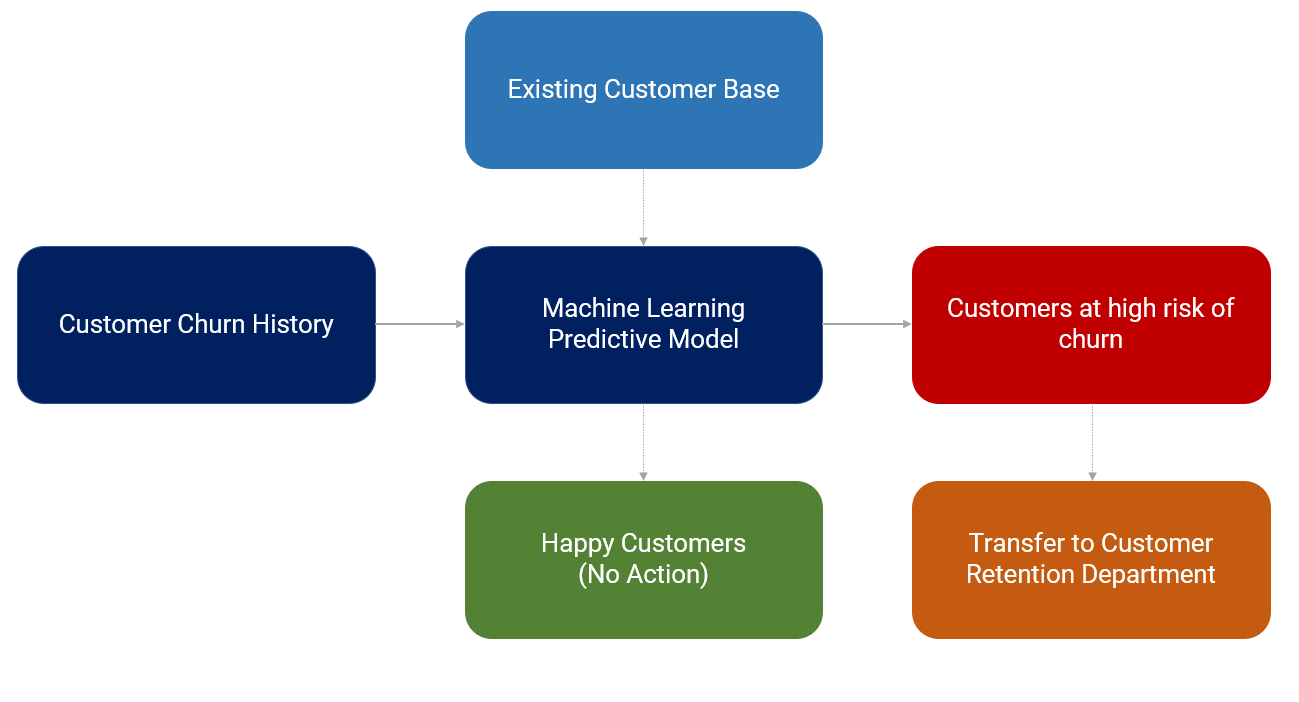
1. Understanding The Business Problem

The primary objective of the customer churn predictive model is to retain customers at the highest risk of churn by proactively engaging with them. For example: Offer a gift voucher or any promotional pricing and lock them in for an additional year or two to extend their lifetime value to the company.

We want a customer churn predictive model to predict the churn in advance (let’s say one month in advance, three months in advance, or even six months in advance — it all depends on the use-case). This means that you have to be extremely careful of the cut-off date i.e. You shouldn’t be using any information after the cut-off date as a feature in the machine learning model, otherwise it will be leakage. The period before the cut-off date is known as the **Event.**

2. Applied Machine Learning to The Business Problem

Now you understand the business problem, let’s discuss how this machine-learning model will be used in the business. Read the below diagram from left to right:



* A model is trained on customer churn history (event period for X features and performance window for target variable).
* Every month active customer base is passed onto **Machine Learning Predictive Model** to return the probability of churn for each customer (in business lingo, this is sometimes called a score of churn).
* The list will be sorted from highest to lowest probability value (or score as they say it) and the customer retention teams will start engaging with the customer to stop the churn, normally by offering some kind of promotion or gift card to lock in few more years.
* Customers that have a very low probability of churn (or essentially model predicts no-churn) are happy customers. No actions are taken on them.

3. Data Loading

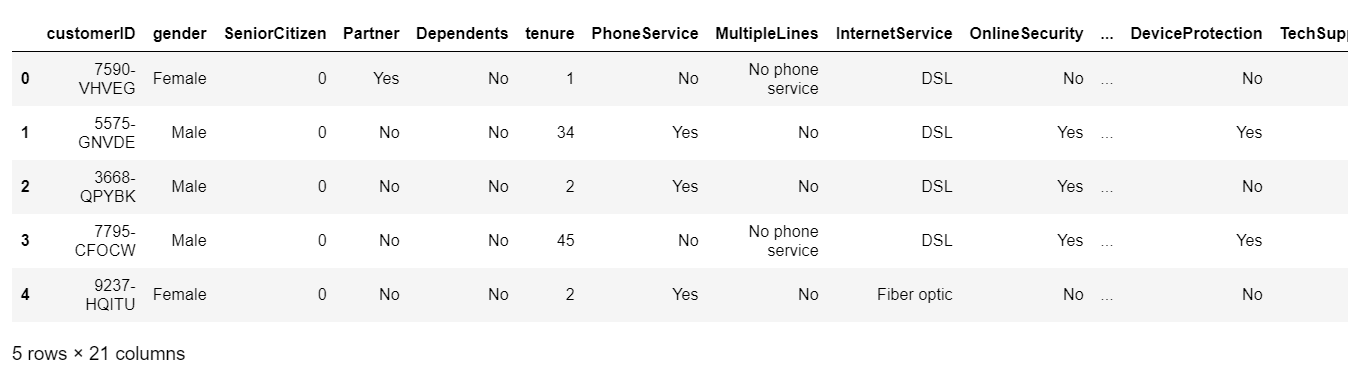
Install PyCaret in your google colab environment.

**# install pycaret**pip install pycaret

For this tutorial, we are using a [Telecom Customer Churn](https://www.kaggle.com/blastchar/telco-customer-churn) dataset from Kaggle. The dataset already contains the target column that we can use as is. You can read this dataset directly from this [GitHub](https://raw.githubusercontent.com/srees1988/predict-churn-py/main/customer_churn_data.csv) link. (*Shoutout to srees1988*) to load the dataset:

**# import libraries**  
import pandas as pd  
import numpy as np**# read csv data**data **=** pd.read\_csv('<https://raw.githubusercontent.com/srees1988/predict-churn-py/main/customer_churn_data.csv'>)

data.head()



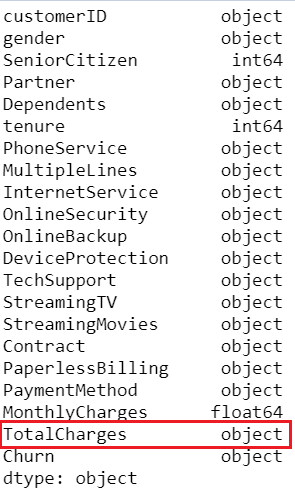
3. Understanding You Dataset

Check the types of your variables, to see which are numeric and which are categorical. Especially, to establish any variable that requires type conversion, use the following:

**# check data dimensions**data.shape

(7043, 21))

**# check data types**data.dtypes



Notice that **TotalCharges** is of an object type instead of float64. If you investigate the dataset, you will notice there are some blank spaces in this column which has caused Python to force the data type as object. To fix that, we will have to format the blank spaces before changing the data type to the correct one, float type.

**# replace blanks with np.nan**  
data['TotalCharges'] = data['TotalCharges'].replace(' ', np.nan)

**# convert to float64**  
data['TotalCharges'] = data['TotalCharges'].astype('float64')

Intuitively contract type, tenure (length of stay of the customer), and pricing plans typically provide very important information when it comes to customer churn or retention. Visualise the relationship:

import plotly.express as px

fig = px.scatter(x=data['tenure'], y=data['TotalCharges'],

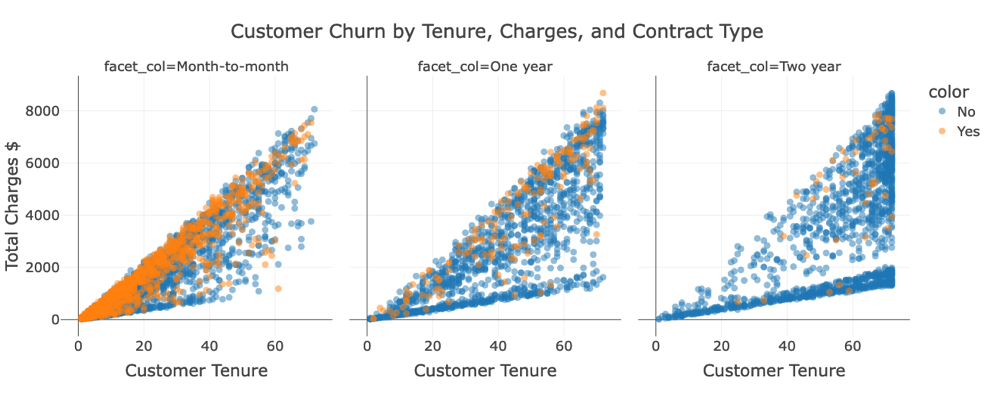
color = data['Churn'], template = 'presentation',

opacity = 0.5, facet\_col = data['Contract'],

title = 'Customer Churn by Tenure, Charges, and Contract Type',

labels = {'x' : 'Customer Tenure', 'y' : 'Total Charges $'})

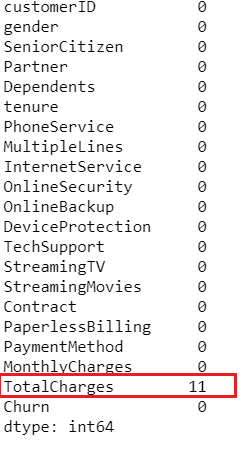
fig.show()



Notice that most churn can be seen in the contracts that are “Month-to-Month”. Also, I can see that as the tenure increases and so are the total charges.

4. Data Preparation   
**Missing Data:** It is time to check for missing data values and deal with them.

**# check missing values**data.isnull().sum()



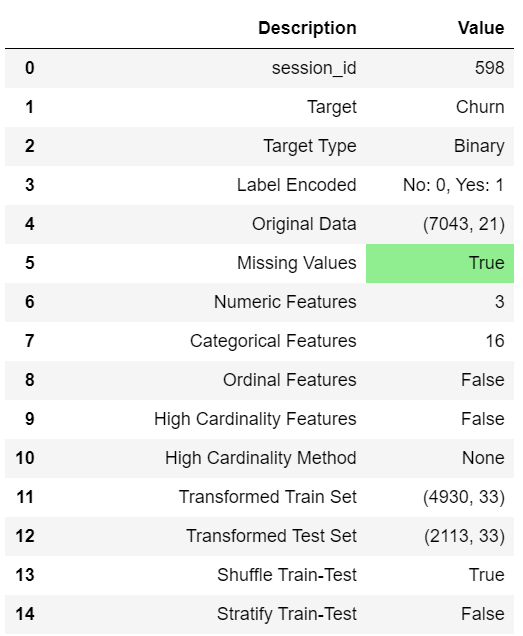
Notice that because we replaced blank values with np.nan there are now 11 rows with missing values in TotalCharges. No problem — Let’s leave it with PyCaret to impute it automatically!

**Ignoring useless variables and defining your target variable:**In PyCaret, the setup is the first and the only mandatory step in any machine learning experiment performed in PyCaret. This function takes care of all the data preparation required prior to training models. Besides performing some basic default processing tasks, PyCaret also offers a wide array of pre-processing features. To learn more about all the preprocessing functionalities in PyCaret.

**# init setup**  
from pycaret.classification import \*  
s = setup(data, target = 'Churn', ignore\_features = ['customerID'])

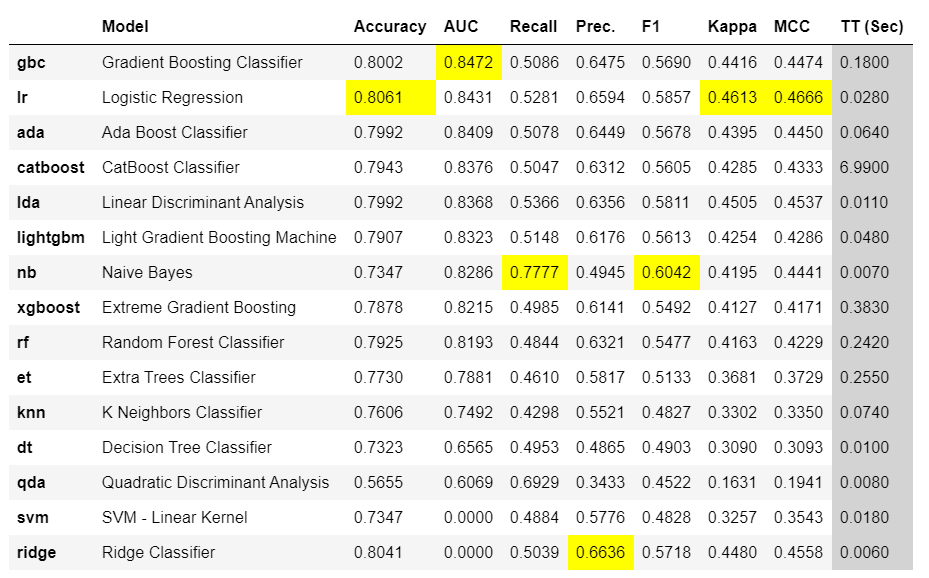
Whenever you initialize the setup function in PyCaret, it profiles the dataset and infers the data types for all input features.

Also, notice that I have passed ignore\_features = ['customerID'] in the setup function so that it is not considered when training the models. The good thing about this is PyCaret will not remove the column from the dataset, it will just ignore it behind the scene for model training. As such when you generate predictions at the end, you don’t need to worry about joining IDs back by yourself.



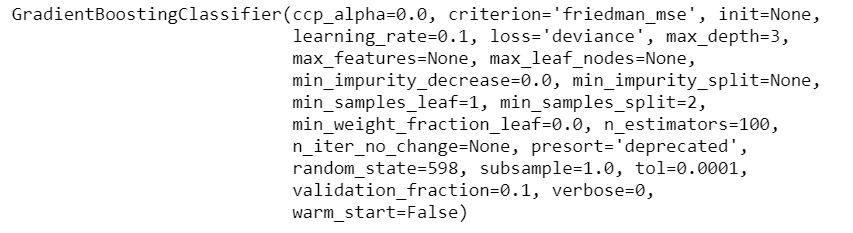
5. Machine Learning Algorithms Training   
Now that data preparation is done, let’s start the training process by using compare\_models functionality. This function trains all the algorithms available in the model library and evaluates multiple performance metrics using cross-validation. Then test them on a 70:30 testset split ratio.

**# compare all models**  
best\_model = compare\_models(sort='AUC')



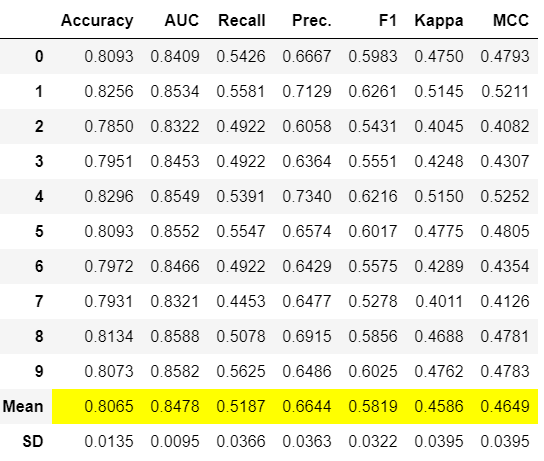
The best model based on **AUC** is Gradient Boosting Classifier. AUC using 10-fold cross-validation is 0.8472.

**# print best\_model parameters**  
print(best\_model)



6. Models Hyper-Parameters Tuning   
  
You can use the tune\_model function from PyCaret to automatically tune the hyperparameters of the model.

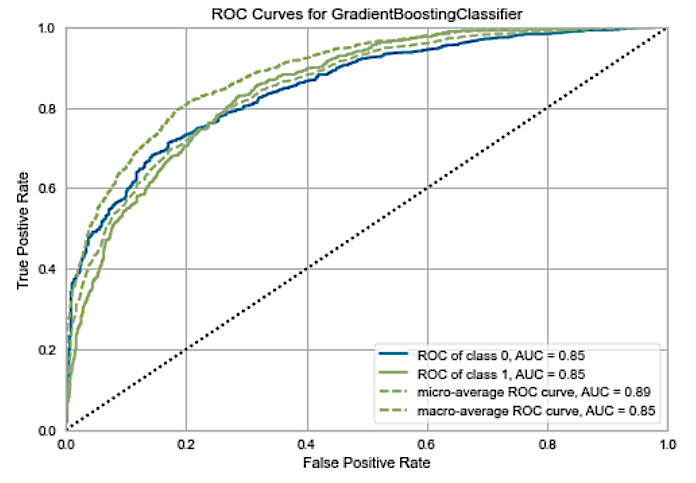
**# tune the best model**  
tuned\_best\_model = tune\_model(best\_model)



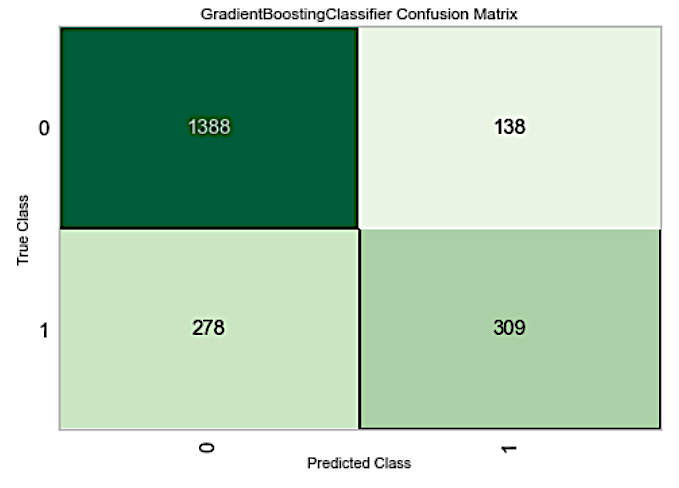
Notice that AUC has slightly increased from 0.8472 to 0.8478

7. Data Scientist Model’s Analysis   
  
Here you analyse the performance of your model based on your selected performance metric of choice, the AUC. This is the one which you thought is relevant to interpret the success criteria!

**# Analyse the AUC Plot**  
plot\_model(tuned\_best\_model, plot = 'auc')



**# Analyse the Confusion Matrix**  
plot\_model(tuned\_best\_model, plot = 'confusion\_matrix')



This confusion matrix is on the test set which includes 30% of our data (2,113 rows) We have 309 **True Positives** (15%) — these are the customers for which are likely to churn, retention team will be able to extend the lifetime value. If we wouldn’t have predicted them, then there was no opportunity for intervention by retention team to keep them.  
  
There are 138 (7%) **False Positives** where we will lose money because the promotion offered to these customers will just be an extra cost.  
  
1,388 (66%) are True Negatives (good customers) and 278 (13%) are **False Negative** (this is a missed opportunity).

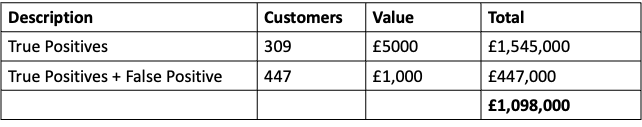
So far, you have trained multiple models to select the best model giving the highest AUC, followed by tuning the hyperparameters of the best model to squeeze a little more performance in terms of AUC. However, the best AUC doesn’t necessarily translate into the best model for business**. This is because there was not a clear firm success criterion to start with!**

8. Marketing Experts' Analysis of The Model

In a churn model, often, the reward of **true positives** is way different from the cost of **false positives**. Let’s use the following assumptions:

* £1,000 vouchers will be offered to all the customers identified as churn (**True Positive + False Positive**);
* If we are able to stop the churn, we will gain £5,000 in customer lifetime value.

Using these assumptions and the confusion matrix above, we can calculate the $ impact of this model:

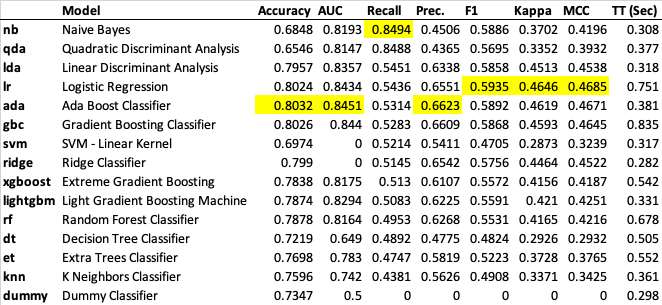


It’s a good model, but the problem is it’s not a business-smart model. It is doing a pretty good job compared to if you have no model, but **how can we train and select a model that maximises the business value? (This is your true success criteria)**   
  
In order to achieve that, we have to train, select, and optimise models using business metrics instead of any conventional metric like AUC or Accuracy.

9. Selecting A Performance Metric to Interpret The Success Criteria

From the above cost table, we notice that maximising the return on retaining True Positives will compensate for the loss of retention vouchers by £4000. Should the metric of choice perhaps be the Recall, AKA True Positive Rate? Let’s compare the models with the Recall in mind.

**# compare all models**  
best\_model = compare\_models(sort='Recall')



Surprisingly Naive Bayes which is a pretty bad model in terms of AUC is the best model when it comes to profit. Let’s see how:

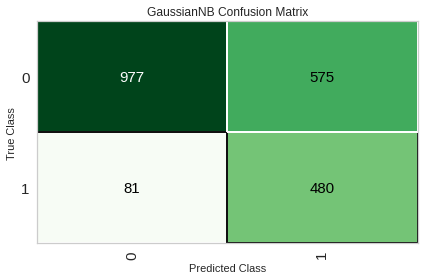
**# print best\_model parameters**

print(best\_model)

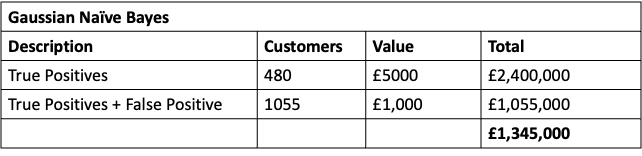
Output: GaussianNB(priors=None, var\_smoothing=1e-09)

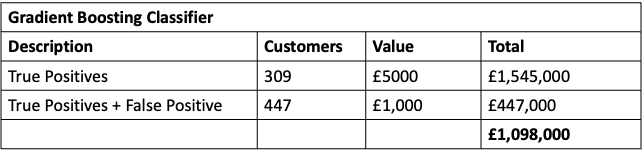
**# Analyse the Confusion Matrix of the best model**

plot\_model(best\_model, plot = 'confusion\_matrix')



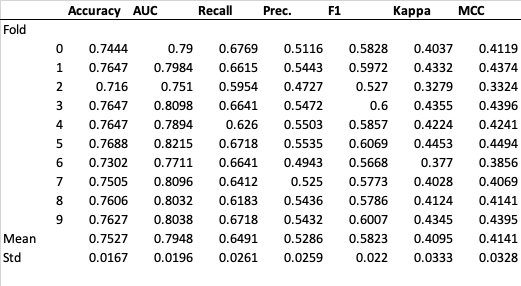
You can clearly see an increase in the TP from the above confusion matrix, however, this was accompanied by an increase in FP compared to the Gradient Boosting Classifier! Did NB Generate more profit? At what Cost?





**# Lets see if tuning the best model (Gaussian NB) improves**

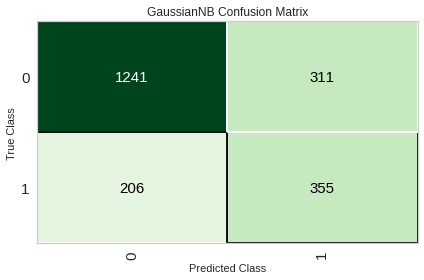
tuned\_best\_model = tune\_model(best\_model)



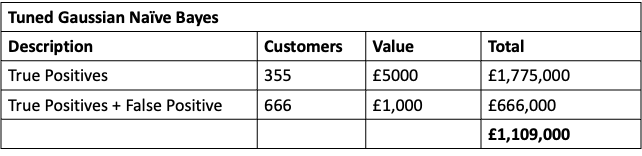
After tuning the model wrapped with the Recall, the model’s TPR, Recall has worsened to 0.64 compared to the untuned NB model!

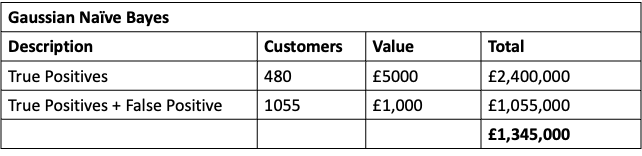
**# Analyse the Confusion Matrix of the tuned NB model**

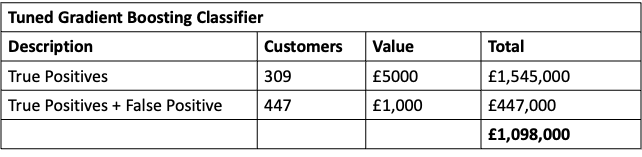
plot\_model(tuned\_best\_model, plot = 'confusion\_matrix')

****

You can clearly see when tuning NB, a decrease occurred to the TP from the above confusion matrix; however, this was accompanied by a decrease in FP compared to the untuned NB Classifier! Did tuning NB improve the profit situation? At what Cost?





****

Appendix.

**Performance Measurement**

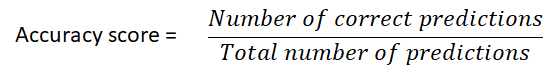
This is the main topic of this article. Here we are going to cover seven performance metrics for classifier by delving into their underlying concept and their implementation using Scikit-Learn.

Let’s start with the simplest one, *accuracy score*.

**(A) Accuracy Score**

Concept

Accuracy score is probably the most straightforward metric in measuring a classifier’s performance. It is a metric that shows us a fraction of the correct prediction. The formula is as below:



The formula of accuracy score

Let’s say if there are 100 records in our test set and our classifier manages to make an accurate prediction for 92 of them, the accuracy score would be 0.92.

**(B)Confusion matrix**

Concept

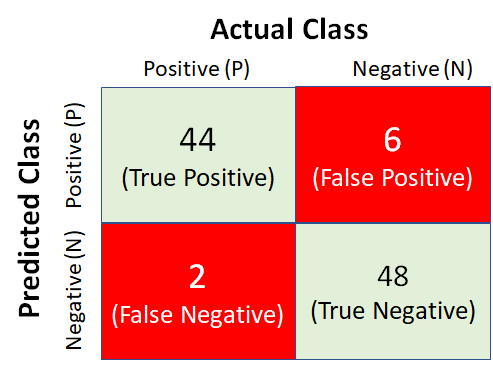
Accuracy scores only give us a fraction of correct prediction in overall. What if we would like to know more about our classifier in the following aspects:

* how many malignant tumours in our samples are correctly and incorrectly predicted?
* how many benign tumours in our samples are correctly and incorrectly predicted?

These questions are essential as no single patient will be happy to receive a misreported medical result. Ideally, the prediction of both malignant and benign classes should achieve high accuracy to minimize the false report. If our classifier shows an obvious bias to either malignant or benign class, we might need to retrain our classifiers by adjusting some hyperparameters or fetching additional data into the training pipeline.

Hence, a better way to evaluate the performance of a classifier is to use **confusion matrix**. A confusion matrix (also known as error matrix)is a two-dimensional table that permits visualization of correctly labelled and mislabeled instances by a classifier.

Given a sample of 100 instances in a test set where each instance belongs to either positive or negative class. The performance of a classifier on that test set can be represented as the confusion matrix below:



A sample of the confusion matrix

Each column in the confusion matrix above represents an actual class while each row represents a predicted class. The top-left quadrant shows that there are 44 positive instances in the test set which are predicted accurately because the actual class (positive) is matched with the predicted class (positive). This type of outcome is known as **True Positive (TP)**. On another hand, the top -right quadrant shows 6 negative instances which are wrongly predicted as the positive class and these are known as **False Positive(FP)**.

At the bottom-right quadrant, there are 48 negative instances which are predicted correctly and are known as **True Negative (TN)**. Lastly, the bottom-left quadrant displays 2 positive instances which are wrongly predicted as a negative class and these are called **False Negative (FN)**.

From the sample confusion matrix above, we can observe that the classifier shows a slightly better performance in identifying negative class than a positive class. Ideally, a good classifier would have an as least possible number of false positive and false negative as possible. This means the higher the non-zero number in the main diagonal of a confusion matrix, the better performance of a classifier.

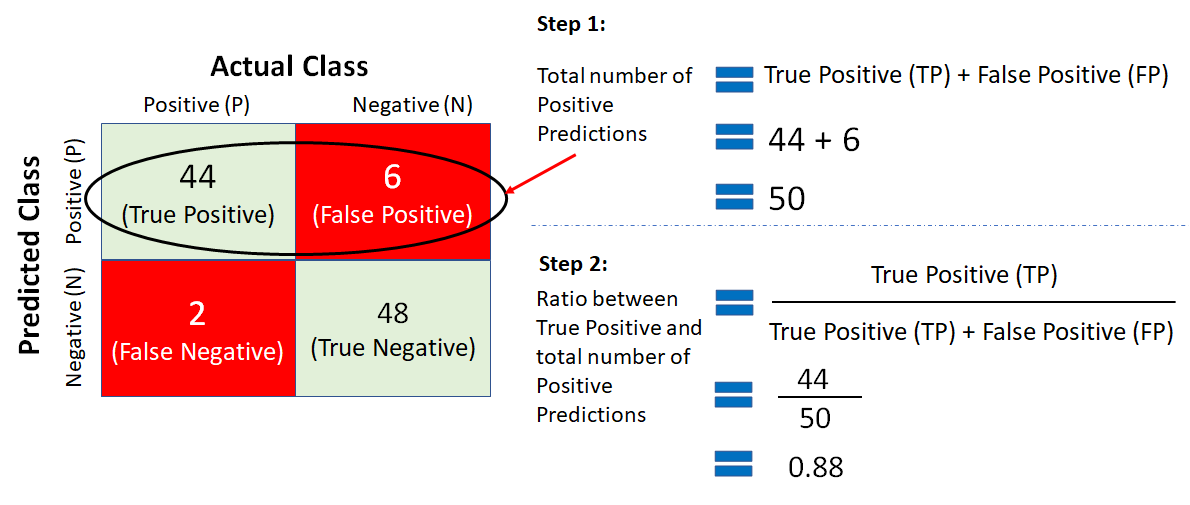
**(C) Precision, Recall and F1 Score**

Concept

The confusion matrix shows us the exact number of true positive, false positive, true negative and false negative in a classification result. In fact, we can go further to derive several useful metrics from the confusion matrix.

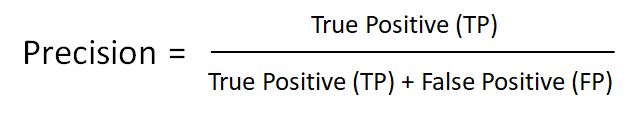
Firstly, we may ask ourselves: **What is the fraction of positive predictions which are true positive?**

To address the question, let us walk through a simple mathematic calculation logic as presented below.



Calculation of precision

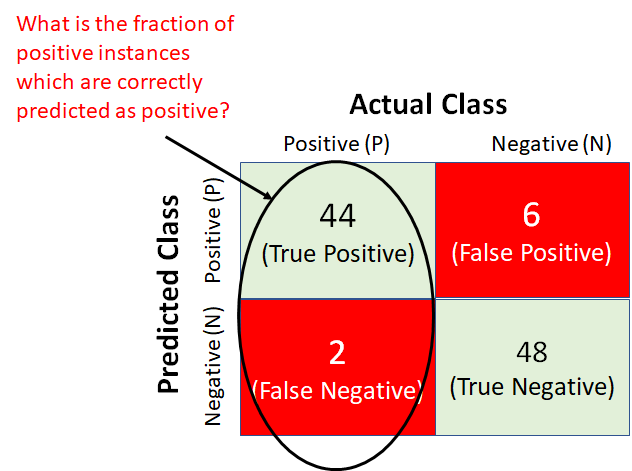
The ratio between the number of True Positives and the total number of Positives are the metric known as **precision.**The formula of precision is given below:



Precision formula

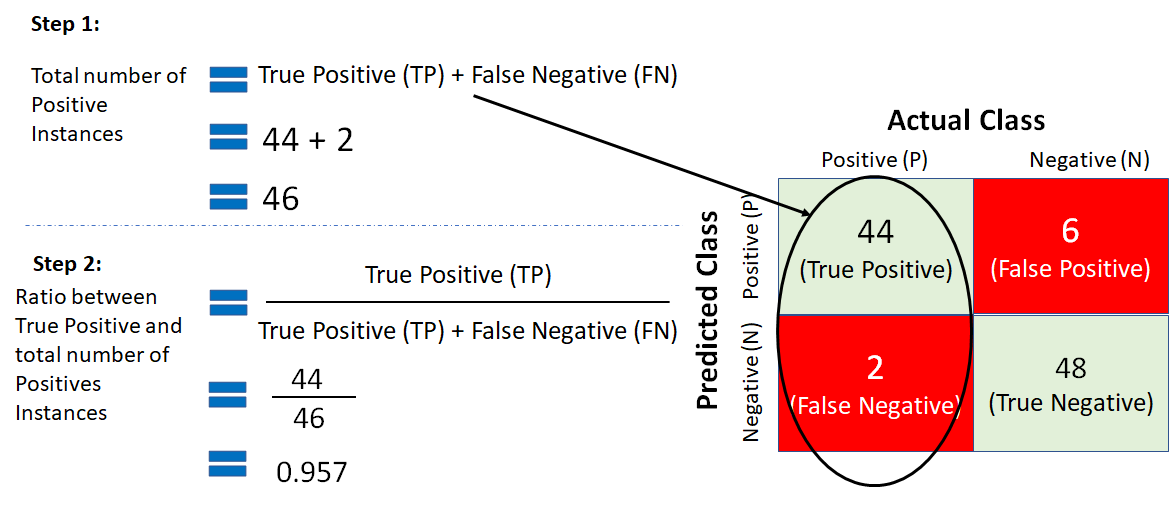
Precision gives us a quantifier to reveal the fraction of the predicted positive instances which are true positive. The sample classifier above achieves the precision score of 0.88 and this also means 88% of predicted positives are true positive.

Now, let us think of another subtle question: **What is the fraction of positive instances which are correctly predicted as positive?**



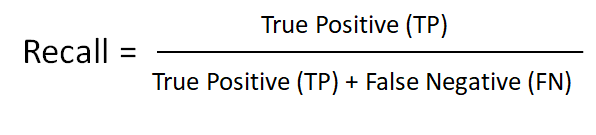
The fraction of positive instances which are predicted as positive

Once again, we are going to use a simple math logics to address the question.



Calculation of recall

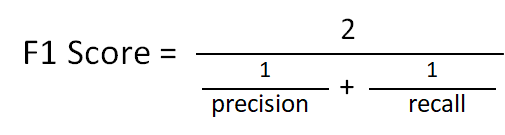
The calculation above result in a ratio between true positive and the total number of positive instances and this ratio is known as **recall**. Recall is also called sensitivity or true positive rate (TPR). The formula of recall is given below:



Recall formula

The sample classifier above hits a recall score of 0.957 which is higher than its precision. This means among all the 46 positive instances, 95.7% of them are correctly predicted as positive.

Precision and recall can also be combined into a single metric called the **F1 Score**. The F1 Score is the harmonic mean of precision and recall.



F1 Score formula

The F1 Score formula may seem a little daunting at the first glance but it is just a specific mean that gives much more weight to low values (in comparison to the regular mean that treats all values equally). A high F1 score indicates a similar precision and recall. For example, if we apply the F1 Score formula to our sample classifier, we will get a score of approximately 0.917. This shows both the precision and recall rates of the sample classifier are quite high.

This is important to learn that precision and recall may not be always similar and in some cases, we would prefer a higher precision than recall or vice versa. For example, we would expect a higher recall rate in medical use cases such as infection detection. It is always better to enable our classifier to accurately identify as many true positive infected instances as possible rather than letting go of some positive instances unnoticed. On another hand, bank analysts may prefer their loan classifier performs with higher precision so that they won’t accidentally reject a potential customer and lose the business.

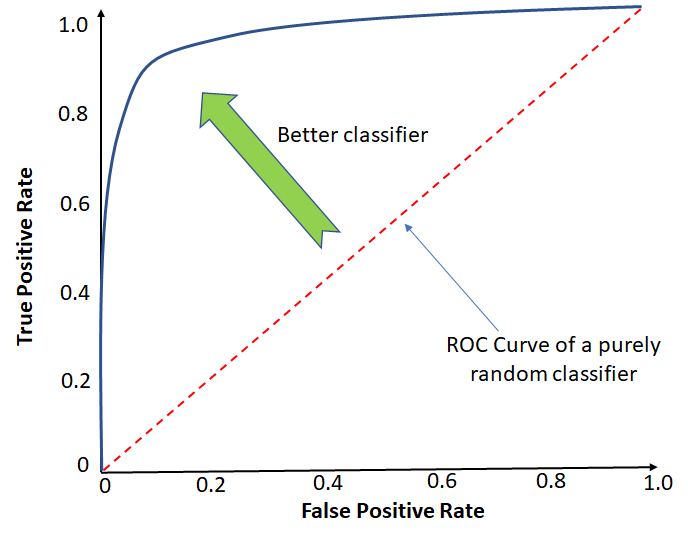
Unfortunately, we can’t always tune our classifier to achieve almost equally high precision and recall rate. Increasing precision results in a reduction of recall and vice versa. This is known as the **precision/recall tradeoff.**

**(D) The ROC Curve and AUROC**

Concept

**The receiver operating characteristic (ROC) curve** offers us a visual approach to examine the performance of our trained classifier. It is a curve that plots the true positive rate (TPR)of a classifier against its false positive rate (FPR). As highlighted in the earlier section, the TPR is also known as recall. On another hand, the FPR is the ratio of negative instances that are wrongly predicted as positive.

A sample of the ROC curve is given below:



ROC Curve

There is a tradeoff between the TPR and FPR — the higher the TPR, the more FPR will be produced by the classifier. The diagonal dotted line in the middle represents the ROC curve which is a random classifier.

The ROC curve of a good classifier should stay as far away from the diagonal line as possible by heading towards the top left corner. Based on this rationale, we can measure and compare our classifiers’ performance by calculating the area under the curve (AUC) which will result in a score called **AUROC (**Area Under the Receiver Operating Characteristics**)**. A perfect AUROC should have a score of 1 whereas a random classifier will have a score of 0.5.