# PROJECT REPORT

## DATA ANALYSIS ON STROKE PREDICTION

Submitted towards the partial fulfilment of the criteria for award of
Post Graduation in Analytics by Imarticus

Submitted By:

**RASEEM AHAMED (IL025408)**

Course and Batch: PGA ONLINE01 **OCT 2021**

# ACKNOWLEDGEMENT

We are using this opportunity to express our gratitude to everyone who supported us throughout the project. We are thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the project work. We are sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.

Further, we were fortunate to have Thulasidass pounraj as our mentor. He has readily shared his immense knowledge in data analytics and guide us in a manner that the outcome resulted in enhancing our data skills.

We wish to thank, all the faculties, as this project utilized knowledge gained from the courses formulated by PGA program.

We certify that the work done by us for conceptualizing and completing this project is original and authentic.

Date:

Place: **Chennai** **RASEEM AHAMED A**

# CERTIFICATE OF COMPLETION

I hereby certify that the project titled **"DATA ANALYSIS ON STROKE PREDICTION"** was undertaken and completed under my supervision by **RASEEM AHAMED . A** from the batch of **PGA ONLINE 01 - OCT 21**

Mentor: **Thulasidass Pounraj**

Date:

Place: **Chennai** **RASEEM AHAMED A**

# ABSTRACT

According to the World Health Organization (WHO) stroke is the 2nd leading cause of death globally, responsible for approximately 11% of total deaths.Stroke is a cerebro-vascular ailment affecting the normal blood supply to the brain. Without proper supervision, it leads to death or long term disability.

Stroke can be either ischemic or hemorrhagic. There is a possibility for the cooccurrence of both ischemic and hemorrhagic strokes. Stroke caused due to a clot in the blood vessel is known as Ischemic stroke and that due to a rupture of blood vessel is referred to as Hemorrhagic stroke.

The deficiency of oxygen to the cerebral nervous system is referred to as ischemia which finally leads to their death. About 85 % of all strokes belong to ischemic category. Its frequency is accelerating in developing countries like India due to unhealthy lifestyles.

# TABLE OF CONTENTS

# List of Figures

## LIST OF TABLES

# CHAPTER 1: INTRODUCTION

## 1.1 TITLE & OBJECTIVE OF THE STUDY

The project titled "**DATA ANALYSIS ON STROKE PREDICTION**" is under category "Healthcare", which inspects the patient's medical information performed across various hospitals. The project primarily focuses on the causes that leads to stroke, which is a binary classification done by using ML-Supervised classification algorithms and predicting.

OBJECTIVE:

- Determine whether the patient has a Stroke or not
- Implementation of supervised ML classification Algorithms

## 1.2 NEED OF THE STUDY

29 October is observed as World Stroke Day. This day aims at spreading awareness about the fatal condition of stroke. This complication is caused due to the reduced or interrupted blood flow in the brain. This leads to insufficient nutrient and oxygen supply in the brain causing it to dysfunctional and damage. It is important to spread awareness about this condition as early detection and treatment is the only way of ensuring safe recovery and preventing fatality. In this Project, 11 clinical features like hypertension,heart disease,glucose level, BMI and so on are obtained for predicting stroke events.

## 1.3 PROBLEM STATEMENT

Predict whether a patient is likely to get stroke based on the input parameters like gender, age, various diseases, and smoking status. Each row in the data provides relevant information about the patient.Perform necessary exploratory data analysis before building the model and evaluate the model based on performance metrics other than model accuracy.

## 1.4 DATASET DESCRIPTION

The dataset consists of several predictor variables and one target variable, Outcome. Predictor variables includes the age,gender,hypertension,smoking status and so on.,

| Column | Description |
|---|---|
| Id | unique identifier |
| gender | "Male", "Female" or "Other" |
| age | age of the patient |
| hypertension | 0 if the patient doesn't have hypertension, 1 if the patient has hypertension |
| heart_disease | 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease |
| ever_married | "No" or "Yes" |
| work_type | "children", "Govt_jov", "Never_worked", "Private" or "Self-employed" |
| Residence_type | "Rural" or "Urban" |
| avg_glucose_level | average glucose level in blood |
| Bmi | body mass index |

| | |
|---|---|
| smoking_status | "formerly smoked", "never smoked", "smokes" or "Unknown"* |
| stroke | 1 if the patient had a stroke or 0 if not |

Table 1.1 - dataset description table

## 1.5 DATA SOURCES

The Dataset Stroke Prediction is taken in Kaggle.Kaggle is an AirBnB for Data Scientists. It's a crowd-sourced platform to attract, nurture, train and challenge data scientists from all around the world to solve data science, machine learning and predictive analytics problems. Kaggle is the number one stop for data science enthusiasts all around the world

Domain - Healthcare
Format - .CSV(Comma Separated value)
File - healthcare-dataset-stroke-data.csv
Source link -Stroke Prediction Dataset | Kaggle

## 1.6 ANALYTICS TOOLS

Python Notebook (Jupyter / google Collab)

## 1.7 ANALYTICS APPROACH

Model Build with Random Forest Classifier (RF)
Model Build with Support Vector Machine (SVM)
Model Build with Decision Tree Classifier (DT)
Model Build with K-Nearest Neighbour (KNN)
Model build with Gradient Boosting Method (GBM)
Model Build with Extreme Gradient Boosting (XGB)
Model Build with Adaboost (AB)

# CHAPTER 2 : LIBRARIES NEEDED IN THE PROJECT

A library is a collection of pre-combined codes that can be used iteratively to reduce the time required to code. They are particularly useful for accessing the pre-written frequently used codes, instead of writing them from scratch every single time.Python Libraries are a set of useful functions that eliminate the need for writing codes from scratch.Python libraries play a vital role in developing machine learning, data science, data visualization, image and data manipulation applications and more.For the visualization, the study will use both Seaborn and Plotly's interactive environment for making a better and meaningful comparison with related subjects.The libraries used in this project are:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import scipy.stats as stats
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV, cross_validate
from sklearn.metrics import plot_confusion_matrix,classification_report, confusion_matrix, accuracy_score
from sklearn.metrics import precision_score, plot_precision_recall_curve, plot_roc_curve, roc_auc_score, roc_curve, f1_score,recall_score
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier
from xgboost import XGBClassifier
from sklearn.tree import plot_tree
from sklearn.impute import SimpleImputer
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC


import cufflinks as cf
import plotly.offline
cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)

import warnings
warnings.filterwarnings('ignore')
warnings.warn("this will not show")
plt.rcParams["figure.figsize"] = (10,6)
pd.set_option('max_colwidth',200)


pd.set_option('display.max_columns', 200)
pd.set_option('display.float_format', lambda x: '%.3f' % x)

import colorama
from colorama import Fore, Style
from termcolor import colored
```

## 2.1 USER DEFINED FUNCTIONS

A user-defined function (UDF) is a function provided by the user of a program or environment,In this project, The function defined for determining the number of percentages of missing values,for comparing different approaches,for in sighting summary information about the column and for examining scores.

```
# Function for determining the number and percentages of missing values

def missing (df):
    missing_number = df.isnull().sum().sort_values(ascending=False)
    missing_percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)
    missing_values = pd.concat([missing_number, missing_percent], axis=1, keys=['Missing_Number', 'Missing_Percent'])
    return missing_values
```

```
# Function for comparing different approaches

def score_dataset(X_train, X_valid, y_train, y_valid):
    model = SVC(class_weight = "balanced", random_state=42)
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    return accuracy_score(y_valid, preds)
```

```python
# Function for insighting summary information about the column
def first_looking(col):
    print("column name    : ", col)
    print("-------------------------------")
    print("per_of_nulls   : ", "%", round(df[col].isnull().sum()/df.shape[0]*100, 2))
    print("num_of_nulls   : ", df[col].isnull().sum())
    print("num_of_uniques : ", df[col].nunique())
    print(df[col].value_counts(dropna = False))
```

```python
# Function for examining scores

def train_val(y_train, y_train_pred, y_test, y_pred):

    scores = {"train_set": {"Accuracy" : accuracy_score(y_train, y_train_pred),
                            "Precision" : precision_score(y_train, y_train_pred),
                            "Recall" : recall_score(y_train, y_train_pred),
                            "f1" : f1_score(y_train, y_train_pred)},

              "test_set": {"Accuracy" : accuracy_score(y_test, y_pred),
                           "Precision" : precision_score(y_test, y_pred),
                           "Recall" : recall_score(y_test, y_pred),
                           "f1" : f1_score(y_test, y_pred)}}

    return pd.DataFrame(scores)
```

## 2.2 WHAT THE PROBLEM IS

In the given Dataset, we have a binary classification problem.
We will make a prediction on the target variable stroke.
Lastly we will build a variety of Classification models and compare the models giving the best prediction on stroke.

## 2.3 TARGET VARIABLE

Target variable, in the machine learning context, is the variable that is or should be the output. For example it could be binary 0 or 1 if you are classifying or it could be a continuous variable if you are doing a regression. In statistics you also refer to it as the response variable.

In our Dataset our target variable is Stroke in the context of determining whether anybody is likely to get Stroke based on the input parameters like gender, age and various test results or not.

# CHAPTER 3 : ANALYZING THE DATA

## 3.1 READING THE DATA

The syntax for Pandas read file is by using a function called read_csv().

This function enables the program to read the data that is already created and saved by the program and implements it and produces the output.

```
df0=pd.read_csv('healthcare-dataset-stroke-data.csv')
```

```
df=df0.copy()
```

```
df
```

| | id | gender | age | hypertension | heart_disease | ever_married | work_type | Residence_type | avg_glucose_level | bmi | smoking_status | stroke |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 9046 | Male | 67.000 | 0 | 1 | Yes | Private | Urban | 228.690 | 36.600 | formerly smoked | 1 |
| **1** | 51676 | Female | 61.000 | 0 | 0 | Yes | Self-employed | Rural | 202.210 | NaN | never smoked | 1 |
| **2** | 31112 | Male | 80.000 | 0 | 1 | Yes | Private | Rural | 105.920 | 32.500 | never smoked | 1 |
| **3** | 60182 | Female | 49.000 | 0 | 0 | Yes | Private | Urban | 171.230 | 34.400 | smokes | 1 |
| **4** | 1665 | Female | 79.000 | 1 | 0 | Yes | Self-employed | Rural | 174.120 | 24.000 | never smoked | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **5105** | 18234 | Female | 80.000 | 1 | 0 | Yes | Private | Urban | 83.750 | NaN | never smoked | 0 |
| **5106** | 44873 | Female | 81.000 | 0 | 0 | Yes | Self-employed | Urban | 125.200 | 40.000 | never smoked | 0 |
| **5107** | 19723 | Female | 35.000 | 0 | 0 | Yes | Self-employed | Rural | 82.990 | 30.600 | never smoked | 0 |
| **5108** | 37544 | Male | 51.000 | 0 | 0 | Yes | Private | Rural | 166.290 | 25.600 | formerly smoked | 0 |
| **5109** | 44679 | Female | 44.000 | 0 | 0 | Yes | Govt_job | Urban | 85.280 | 26.200 | Unknown | 0 |

5110 rows × 12 columns

# CHAPTER 4 : EXPLORATORY DATA ANALYSIS (EDA) & VISUALIZATION

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns,to spot anomalies,to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.The main purpose of EDA is to help look at data before making any assumptions. It can help identify obvious errors, as well as better understand patterns within the data, detect outliers or anomalous events, find interesting relations among the variables.

## 4.1 GENERAL LOOKING AT THE DATA

For looking the Data's we have some functions that helps to check for the observations present in the data and the columns present, data types,statistical summary, missing data's,unique and check for duplicates.

.head() is used to view the top 5 observations

```
#Check for top 5 datas using .head()
```

```
df.head()
```

| | id | gender | age | hypertension | heart_disease | ever_married | work_type | Residence_type | avg_glucose_level | bmi | smoking_status | stroke |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 9046 | Male | 67.000 | 0 | 1 | Yes | Private | Urban | 228.690 | 36.600 | formerly smoked | 1 |
| 1 | 51676 | Female | 61.000 | 0 | 0 | Yes | Self-employed | Rural | 202.210 | NaN | never smoked | 1 |
| 2 | 31112 | Male | 80.000 | 0 | 1 | Yes | Private | Rural | 105.920 | 32.500 | never smoked | 1 |
| 3 | 60182 | Female | 49.000 | 0 | 0 | Yes | Private | Urban | 171.230 | 34.400 | smokes | 1 |
| 4 | 1665 | Female | 79.000 | 1 | 0 | Yes | Self-employed | Rural | 174.120 | 24.000 | never smoked | 1 |

.tail() is used to view the last 5 observations

```
#Check for last 5 datas using .tail()
```

```
df.tail()
```

| | id | gender | age | hypertension | heart_disease | ever_married | work_type | Residence_type | avg_glucose_level | bmi | smoking_status | stroke |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5105 | 18234 | Female | 80.000 | 1 | 0 | Yes | Private | Urban | 83.750 | NaN | never smoked | 0 |
| 5106 | 44873 | Female | 81.000 | 0 | 0 | Yes | Self-employed | Urban | 125.200 | 40.000 | never smoked | 0 |
| 5107 | 19723 | Female | 35.000 | 0 | 0 | Yes | Self-employed | Rural | 82.990 | 30.600 | never smoked | 0 |
| 5108 | 37544 | Male | 51.000 | 0 | 0 | Yes | Private | Rural | 166.290 | 25.600 | formerly smoked | 0 |
| 5109 | 44679 | Female | 44.000 | 0 | 0 | Yes | Govt_job | Urban | 85.280 | 26.200 | Unknown | 0 |

.sample() is used to view the random observations

```
#check for random datas using .sample()
```

```
df.sample(10)
```

| | id | gender | age | hypertension | heart_disease | ever_married | work_type | Residence_type | avg_glucose_level | bmi | smoking_status | stroke |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4895 | 28717 | Female | 56.000 | 1 | 0 | Yes | Private | Rural | 177.560 | 30.100 | never smoked | 0 |
| 1949 | 40371 | Female | 47.000 | 0 | 0 | Yes | Private | Urban | 62.470 | 26.500 | never smoked | 0 |
| 899 | 68438 | Female | 51.000 | 0 | 0 | Yes | Private | Rural | 90.780 | 32.300 | never smoked | 0 |
| 566 | 36942 | Male | 27.000 | 0 | 0 | No | Private | Urban | 114.790 | 32.000 | Unknown | 0 |
| 4742 | 5319 | Male | 48.000 | 0 | 0 | Yes | Private | Rural | 98.240 | 34.600 | never smoked | 0 |
| 2727 | 24832 | Female | 65.000 | 0 | 0 | Yes | Self-employed | Urban | 77.460 | 30.900 | formerly smoked | 0 |
| 5099 | 7293 | Male | 40.000 | 0 | 0 | Yes | Private | Rural | 83.940 | NaN | smokes | 0 |
| 4400 | 68059 | Male | 35.000 | 0 | 0 | Yes | Govt_job | Rural | 103.080 | 41.500 | smokes | 0 |
| 362 | 49916 | Male | 76.000 | 0 | 0 | Yes | Private | Rural | 110.990 | 29.800 | formerly smoked | 0 |
| 1489 | 30746 | Female | 30.000 | 0 | 0 | Yes | Private | Rural | 124.080 | 41.100 | Unknown | 0 |

.columns is used to view the columns present in the dataset

```
#Columns present in the Dataset .columns
```

```
df.columns
```

```
Index(['id', 'gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
       'work_type', 'Residence_type', 'avg_glucose_level', 'bmi',
       'smoking_status', 'stroke'],
      dtype='object')
```

.shape() is used to view the total observations of rows and columns

```
#Check for rows and columns in the dataset using .shape()
```

```
print("There is", df.shape[0], "observation and", df.shape[1], "columns in the dataset")
```

```
There is 5110 observation and 12 columns in the dataset
```

.info() is used to view the count and data-type present in each column

```
#Check for their Count and Datatype using .info()
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 5110 non-null   int64  
 1   gender             5110 non-null   object 
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64  
 4   heart_disease      5110 non-null   int64  
 5   ever_married       5110 non-null   object 
 6   work_type          5110 non-null   object 
 7   Residence_type     5110 non-null   object 
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object 
 11  stroke             5110 non-null   int64  
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

Our dataset demonstrates;

- 7 numeric variable including (4) int64 and (3) float64 data types out of 7.
- 5 non-numeric variable including (5) object types out of 5.
- In our dataset, we have both numerical and categorical variables.
- It is critical to determine if the columns are correctly designed.
- For the analysis, it is critical to determine our target (label) variable which is "stroke" in the given data.
- It is critical to determine if stroke is an integer/binary type or not.
- In this dataset, target variable is coded as 1 for positive cases (having stroke) and 0 for negative cases (not having stroke).
- Both Hypertension and heart disease have integer data types, not as an object.
- Like our Target variable (stroke), both hypertension and heart_disease are coded as 1 for the positive cases and 0 for negative cases.
- In addition, we have 5 categorical variables, which needs to be converted to Label Encoder.

.describe() is used to view the statistical summary for the given data

```
#Statistical summary using .describe()
```

```
#Numerical datatype
df.describe().T
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| id | 5110.000 | 36517.829 | 21161.722 | 67.000 | 17741.250 | 36932.000 | 54682.000 | 72940.000 |
| age | 5110.000 | 43.227 | 22.613 | 0.080 | 25.000 | 45.000 | 61.000 | 82.000 |
| hypertension | 5110.000 | 0.097 | 0.297 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 |
| heart_disease | 5110.000 | 0.054 | 0.226 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 |
| avg_glucose_level | 5110.000 | 106.148 | 45.284 | 55.120 | 77.245 | 91.885 | 114.090 | 271.740 |
| bmi | 4909.000 | 28.893 | 7.854 | 10.300 | 23.500 | 28.100 | 33.100 | 97.600 |
| stroke | 5110.000 | 0.049 | 0.215 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 |

```
#Object Datatype
df.describe(include='object').T
```

| | count | unique | top | freq |
|---|---|---|---|---|
| gender | 5110 | 3 | Female | 2994 |
| ever_married | 5110 | 2 | Yes | 3353 |
| work_type | 5110 | 5 | Private | 2925 |
| Residence_type | 5110 | 2 | Urban | 2596 |
| smoking_status | 5110 | 4 | never smoked | 1892 |

.nunique() is used to view the unique values present in each column for the given data

```
#Check for unique datas in each column using .nunique()
```

```
df.nunique()
```

```
id                   5110
gender                  3
age                   104
hypertension            2
heart_disease           2
ever_married            2
work_type               5
Residence_type          2
avg_glucose_level    3979
bmi                   418
smoking_status          4
stroke                  2
dtype: int64
```

```
# to find how many unique values numeric features have
```

```
for col in df.select_dtypes(include=[np.number]).columns:
    print(f"{col} has {df[col].nunique()} unique value")
```

```
id has 5110 unique value
age has 104 unique value
hypertension has 2 unique value
heart_disease has 2 unique value
avg_glucose_level has 3979 unique value
bmi has 418 unique value
stroke has 2 unique value
```

```
# to find how many unique values object features have
```

```
for col in df.select_dtypes(include=['object']).columns:
    print(f"{col} has {df[col].nunique()} unique value")
```

```
gender has 3 unique value
ever_married has 2 unique value
work_type has 5 unique value
Residence_type has 2 unique value
smoking_status has 4 unique value
```

.duplicated().value_counts is used to view the duplicated values present in the given data

```
df.duplicated().value_counts()
```

```
False    5110
dtype: int64
```

In the given dataset, we have no duplicated rows.

.isnull().sum() is used to view the missing values present in the given data

```
df.isnull().sum()
```

```
id                   0
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                201
smoking_status       0
stroke               0
dtype: int64
```

User defined function: (for missing data)

```
#Check for missing Datas
```

```
missing(df)
```

| | Missing_Number | Missing_Percent |
|---|---|---|
| bmi | 201 | 0,039 |
| id | 0 | 0,000 |
| gender | 0 | 0,000 |
| age | 0 | 0,000 |
| hypertension | 0 | 0,000 |
| heart_disease | 0 | 0,000 |
| ever_married | 0 | 0,000 |
| work_type | 0 | 0,000 |
| Residence_type | 0 | 0,000 |
| avg_glucose_level | 0 | 0,000 |
| smoking_status | 0 | 0,000 |
| stroke | 0 | 0,000 |

In the given dataset, there have been 201 missing values in the column of "bmi". These missing values will be handled with after Train & Split process for preventing data leakage.

## 4.2 DATA CLEANING.

Data cleansing is the process of identifying and correcting inaccurate records from a record set, table, or database. Data cleansing is a valuable process that helps to increase the quality of the data. As the key business decisions will be made based on the data, it is essential to have a strong data cleansing procedure is in place to deliver a good quality data.

The column of 'Residence_type' begins with uppercase while others are not. To make a standardize grammer to prevent mistake we will change all column names into lowercase

```
df.columns = df.columns.str.lower().str.replace('&', '_').str.replace(' ', '_')
```

```
df.columns
```

```
Index(['id', 'gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
       'work_type', 'residence_type', 'avg_glucose_level', 'bmi',
       'smoking_status', 'stroke'],
      dtype='object')
```

Before going deep into the analysis it would be beneficial to examine the correlation among variables using heatmap

```
sns.heatmap(df.corr(), annot=True,cmap='flare');
```

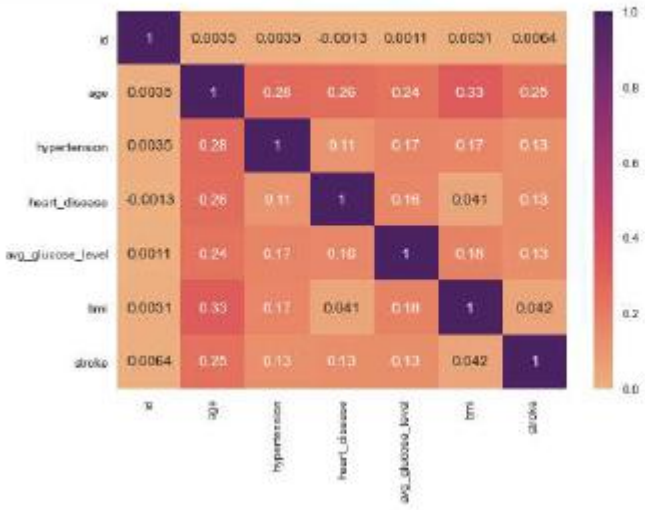



Fig 4.1 - Heat-map

Let's split our features into two part, numerical and categorical, for easing our further examination.

```
numerical= df.drop(['stroke'], axis=1).select_dtypes('number').columns

categorical = df.select_dtypes('object').columns

print("\033[1m", "Numerical Columns:", "\033[0;0m",  numerical)
print("-------------------------------------------------------------------------------------------------------------------")
print("\033[1m", "Categorical Columns:", "\033[0;0m", categorical)
```

```
Numerical Columns:  Index(['id', 'age', 'hypertension', 'heart_disease', 'avg_glucose_level',
       'bmi'],
      dtype='object')
-------------------------------------------------------------------------------------------------------------------
Categorical Columns:  Index(['gender', 'ever_married', 'work_type', 'residence_type',
       'smoking_status'],
      dtype='object')
```

```
df[numerical].shape
```

```
(5110, 6)
```

```
df[categorical].shape
```

```
(5110, 5)
```

## 4.3 EXAMINATION OF TARGET VARIABLE

The proportion of target variable is one of the most important things in a classification problem. So let's a close look at how its values are distributed.

```
df['stroke'].value_counts()
```

```
0    4861
1     249
Name: stroke, dtype: int64
```

```
df['stroke'].value_counts(normalize=True)*100
```

```
0   95.127
1    4.873
Name: stroke, dtype: float64
```

```
y = df['stroke']
print(f'Percentage of patient has a stroke: % {round(y.value_counts(normalize=True)[1]*100,2)} -->\
     ({y.value_counts()[1]} patient)\nPercentage of patient does not have a stroke: %
      {round(y.value_counts(normalize=True)[0]*100,2)} --> ({y.value_counts()[0]} patient)')
```

```
Percentage of patient has a stroke: % 4.87 -->     (249 patient)
Percentage of patient does not have a stroke: % 95.13 --> (4861 patient)
```

- With normalize set to True, we can obtain the relative frequencies by dividing all values by the sum of values.
- In this sense, almost %95 of the instances in our target variable haven't experienced with 'stroke' representing 4861 patients.
- On the other hand %5 of the instances in our target variable go through 'Stroke' representing 249 patient.
- Similarly, it's clear that the proportinate distribution for each class is not the case here. So we should assume an imbalanced data we have in the given case.

```
first_looking("stroke")
```

```
column name    :  stroke
--------------------------------
per_of_nulls   :  % 0.0
num_of_nulls   :  0
num_of_uniques :  2
0    4861
1     249
Name: stroke, dtype: int64
```

```
print(df["stroke"].value_counts())
df["stroke"].value_counts().plot(kind="pie", autopct='%1.1f%%', figsize=(10,10));
```

```
0    4861
1     249
Name: stroke, dtype: int64
```

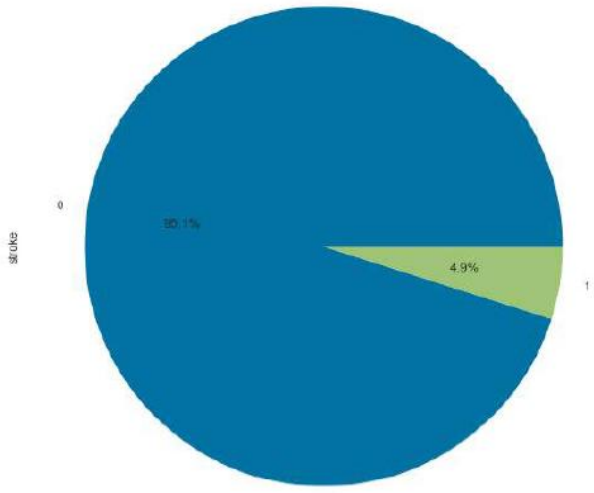



Fig 4.2 -Pie Chart for Target Variable

```
df['stroke'].describe()
```

```
count   5110.000
mean       0.049
std        0.215
min        0.000
25%        0.000
50%        0.000
75%        0.000
max        1.000
Name: stroke, dtype: float64
```

```
df[df['stroke']==0].describe().T.style.background_gradient(subset=['mean','std','50%','count'], cmap='GnBu')
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| id | 4861.000000 | 36487.236371 | 21120.133386 | 67.000000 | 17762.000000 | 36958.000000 | 54497.000000 | 72940.000000 |
| age | 4861.000000 | 41.971545 | 22.291940 | 0.080000 | 24.000000 | 43.000000 | 59.000000 | 82.000000 |
| hypertension | 4861.000000 | 0.088871 | 0.284586 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| heart_disease | 4861.000000 | 0.047110 | 0.211895 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 |
| avg_glucose_level | 4861.000000 | 104.795513 | 43.846069 | 55.120000 | 77.120000 | 91.470000 | 112.830000 | 267.760000 |
| bmi | 4700.000000 | 28.823064 | 7.908287 | 10.300000 | 23.400000 | 28.000000 | 33.100000 | 97.600000 |
| stroke | 4861.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |

## Visualization of Target variable: (plotly- iplot is used)

```
df['stroke'].iplot(kind='hist')
```

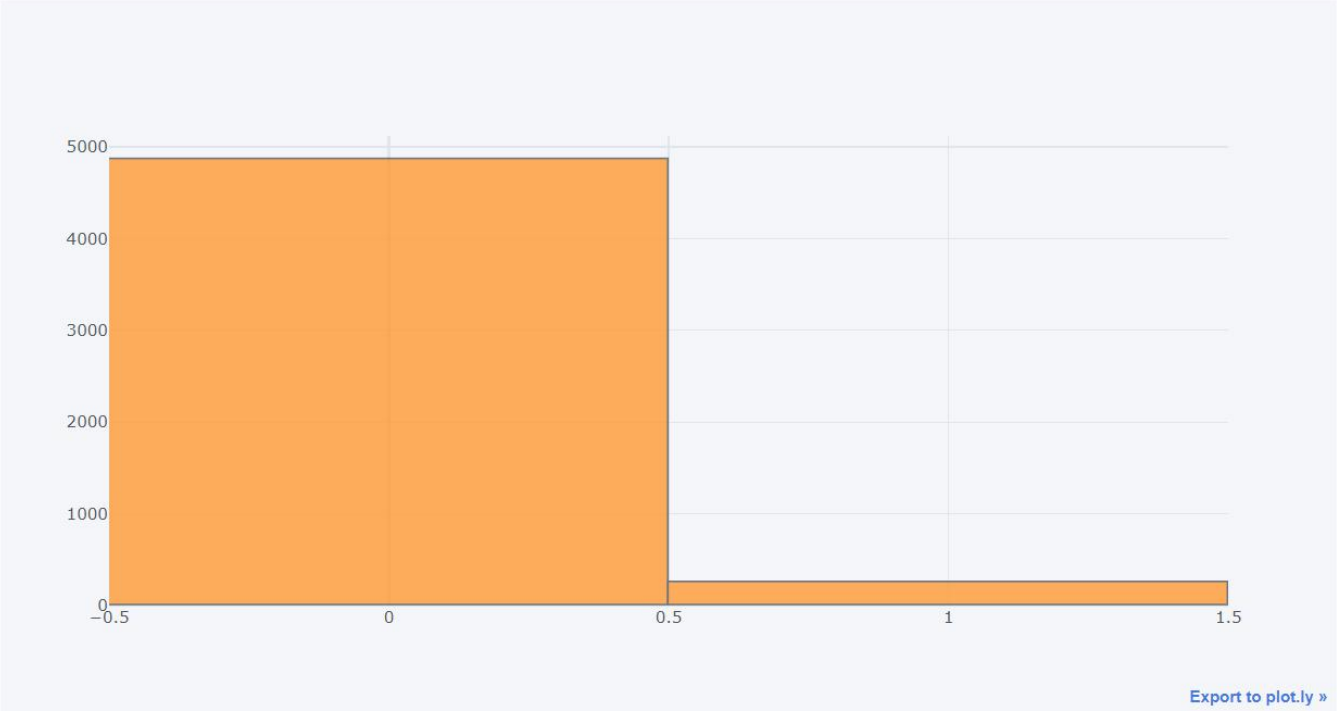



Fig 4.3 -Histogram plot for Target Variable

## 4.4 NUMERICAL FEATURES

In data, we have numerical datas and categorical datas, So lets take a close look on how numerical features are distributed by checking with their statistical summary and data visualization.

```
df[numerical].describe().T
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **id** | 5110.000 | 36517.829 | 21161.722 | 67.000 | 17741.250 | 36932.000 | 54682.000 | 72940.000 |
| **age** | 5110.000 | 43.227 | 22.613 | 0.080 | 25.000 | 45.000 | 61.000 | 82.000 |
| **hypertension** | 5110.000 | 0.097 | 0.297 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 |
| **heart_disease** | 5110.000 | 0.054 | 0.226 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 |
| **avg_glucose_level** | 5110.000 | 106.148 | 45.284 | 55.120 | 77.245 | 91.885 | 114.090 | 271.740 |
| **bmi** | 4909.000 | 28.893 | 7.854 | 10.300 | 23.500 | 28.100 | 33.100 | 97.600 |

### Visualization for Numerical features:

```
: df[numerical].iplot(kind='histogram',subplots=True,bins=50)
```


Fig 4.4 -Histogram plot for Numerical Features

**Visualization for Continuous Data:**

For ‘Id’ column:

```
for i in df.drop(columns=["stroke", 'gender', 'hypertension', 'heart_disease', 'ever_married',
       'work_type', 'residence_type', 'smoking_status']).columns:
    df[i].iplot(kind="box", title=i, boxpoints="all",color='Blue')
```

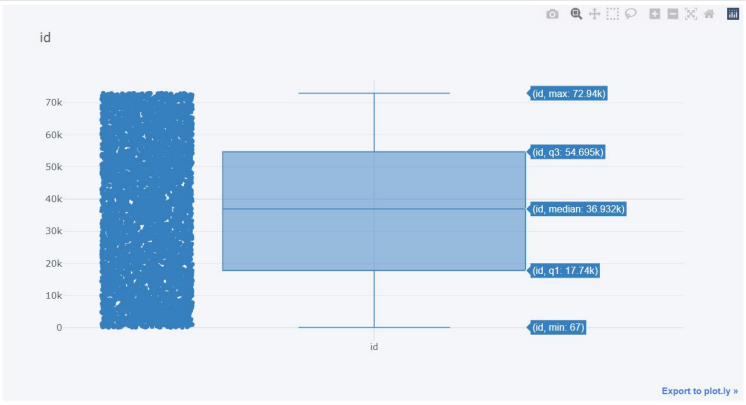



Fig 4.5 -Box plot for continuous data (id)

For ‘age’ column:

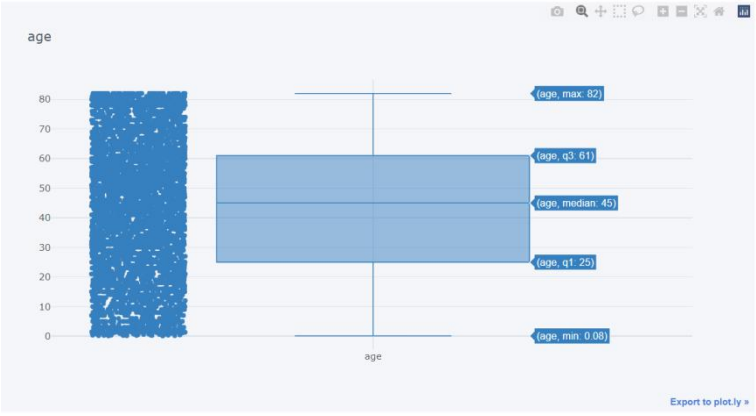



Fig 4.6 -Box plot for continuous data (age)

For ‘avg_glucose_level’ column:

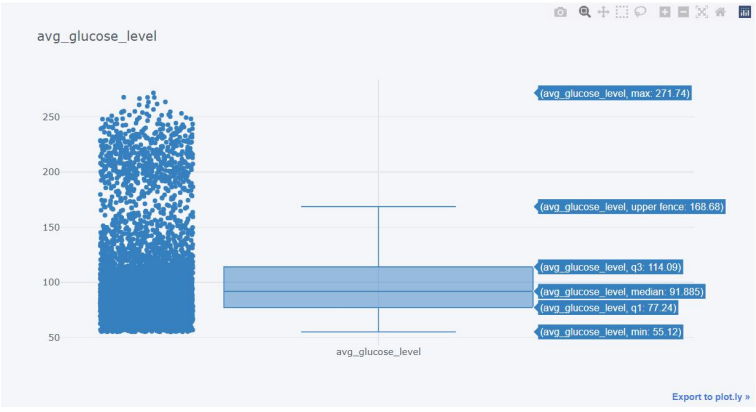



Fig 4.7 -Box plot for continuous data (avg_glucose_level)

For ‘bmi’ column:

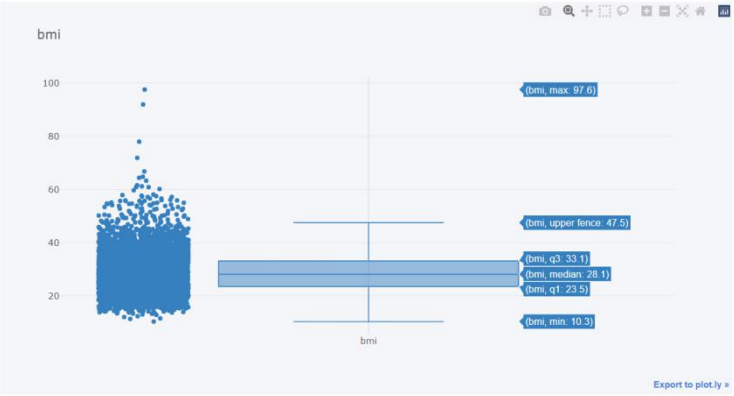



Fig 4.8 -Box plot for continuous data (bmi)

3d Visualization:

```
fig = px.scatter_3d(df,
                    x='avg_glucose_level',
                    y='age',
                    z='gender',
                    color='stroke')
fig.show();
```

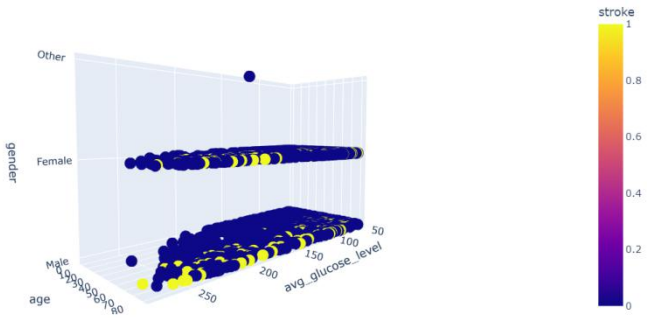



Fig 4.9 -3d Scatter plot for continuous data

By checking with 3d scatter plot (fig 1.8),In the "gender" column there has been an undefined classification which makes no contribution to understand stroke. So let's discard this row from the analysis

```
df.gender.value_counts()
```

```
Female    2994
Male      2115
Other        1
Name: gender, dtype: int64
```

```
df.drop(df[df['gender'] == 'Other'].index, inplace = True)
```

```
df.shape
```

```
(5109, 12)
```

```
df.gender.value_counts()
```

```
Female    2994
Male      2115
Name: gender, dtype: int64
```

## 4.5 CATEGORICAL FEATURES

In data, we have numerical data's and categorical data's, So lets take a close look on how categorical features are distributed by checking with their statistical summary and data visualization.

```
df[categorical].head().T
```

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| **gender** | Male | Female | Male | Female | Female |
| **ever_married** | Yes | Yes | Yes | Yes | Yes |
| **work_type** | Private | Self-employed | Private | Private | Self-employed |
| **residence_type** | Urban | Rural | Rural | Urban | Rural |
| **smoking_status** | formerly smoked | never smoked | never smoked | smokes | never smoked |

```
df[categorical].describe()
```

| | gender | ever_married | work_type | residence_type | smoking_status |
|---|---|---|---|---|---|
| **count** | 5109 | 5109 | 5109 | 5109 | 5109 |
| **unique** | 2 | 2 | 5 | 2 | 4 |
| **top** | Female | Yes | Private | Urban | never smoked |
| **freq** | 2994 | 3353 | 2924 | 2596 | 1892 |

## Visualization:

Each and every in-dependant categorical features is visualized with target variable and check how the data is distributed.

**gender & stroke**

```
first_looking("gender")
```

```
column name    :  gender
--------------------------------
per_of_nulls   :  % 0.0
num_of_nulls   :  0
num_of_uniques :  2
Female    2994
Male      2115
Name: gender, dtype: int64
```

```
print(df.groupby('gender')['stroke'].mean().sort_values())
print()
df.groupby('gender')['stroke'].mean().iplot(kind='histogram',subplots=True,bins=50)
```

```
gender
Female   0.047
Male     0.051
Name: stroke, dtype: float64
```

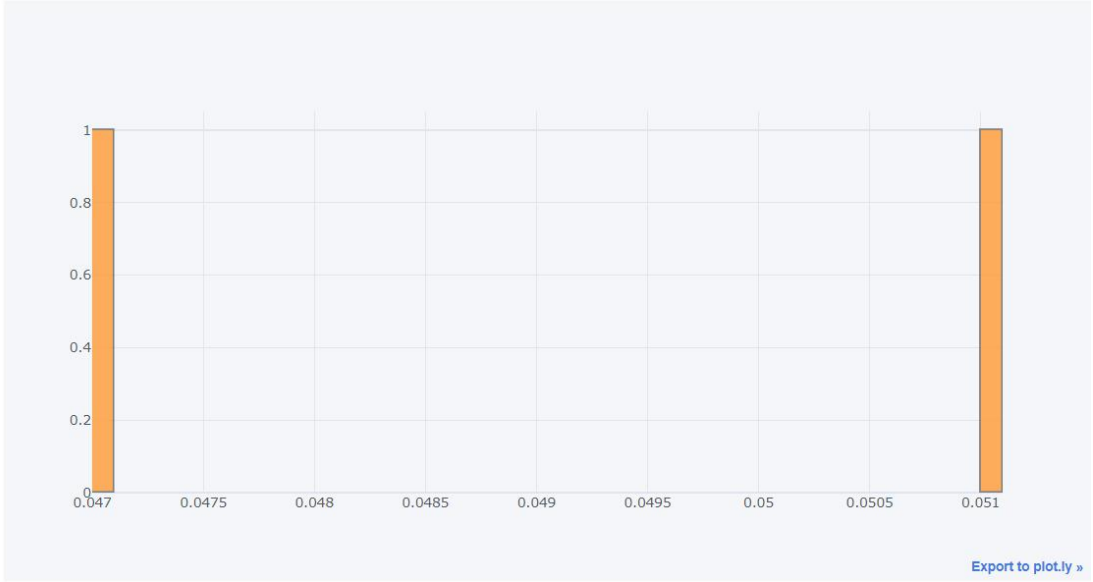



Fig 4.10 -histogram for gender & stroke data

**ever_married & stroke**

```
first_looking("ever_married")
```

```
column name    :  ever_married
--------------------------------
per_of_nulls   :  % 0.0
num_of_nulls   :  0
num_of_uniques :  2
Yes    3353
No     1756
Name: ever_married, dtype: int64
```

```
print(df.groupby('ever_married')['stroke'].mean().sort_values())
print()
df.groupby('ever_married')['stroke'].mean().iplot(kind='bar',subplots=True,bins=50)
```

```
ever_married
No    0.017
Yes   0.066
Name: stroke, dtype: float64
```

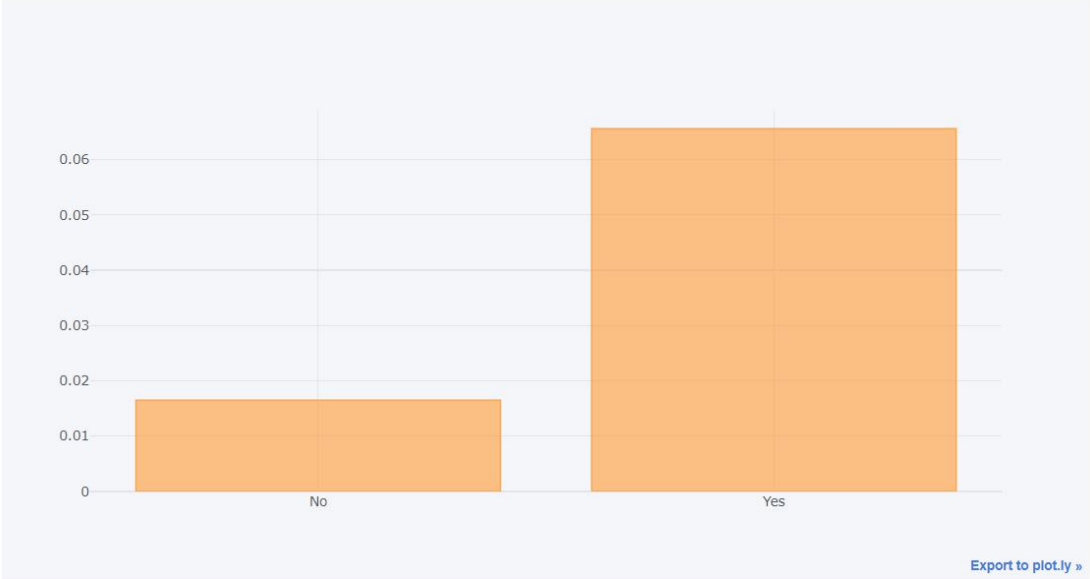



Fig 4.11 -Bar-plot for ever_married & stroke data

**work_type & stroke**

```
first_looking("work_type")
```

```
column name     :  work_type
---------------------------------
per_of_nulls    :  % 0.0
num_of_nulls    :  0
num_of_uniques  :  5
Private           2924
Self-employed      819
children           687
Govt_job           657
Never_worked        22
Name: work_type, dtype: int64
```

```
print(df.groupby('work_type')['stroke'].mean().sort_values())
print()
df.groupby('work_type')['stroke'].mean().iplot(kind='bar',subplots=True,bins=50)
```

```
work_type
Never_worked    0.000
children        0.003
Govt_job        0.050
Private         0.051
Self-employed   0.079
Name: stroke, dtype: float64
```

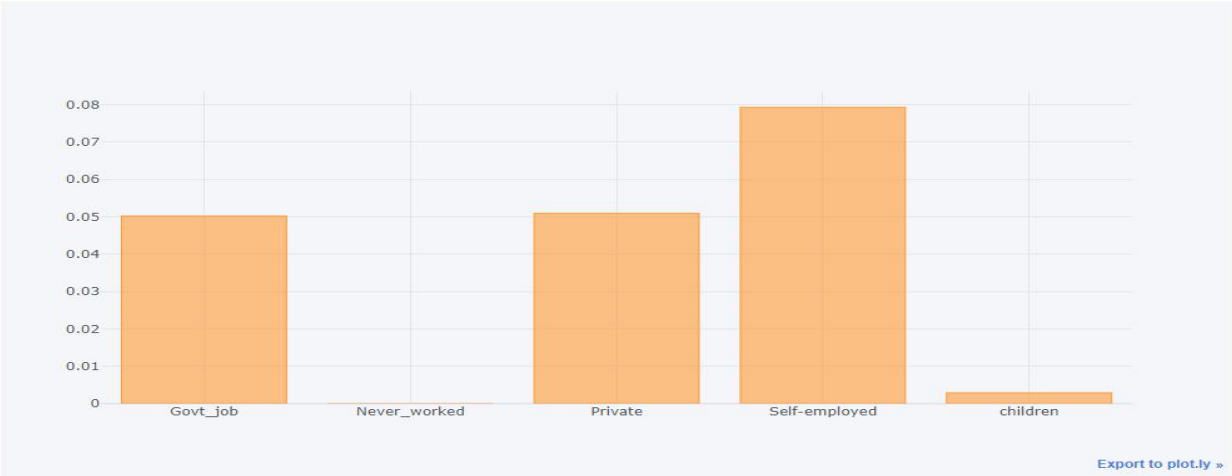



Fig 4.12 -Bar-plot for work_type & stroke data

**residence_type & stroke**

```
first_looking("residence_type")
```

```
column name    :  residence_type
---------------------------------
per_of_nulls   :  % 0.0
num_of_nulls   :  0
num_of_uniques :  2
Urban    2596
Rural    2513
Name: residence_type, dtype: int64
```

```
print(df.groupby('residence_type')['stroke'].mean().sort_values())
print()
df.groupby('residence_type')['stroke'].mean().iplot(kind='bar',subplots=True,bins=50)
```

```
residence_type
Rural   0.045
Urban   0.052
Name: stroke, dtype: float64
```


Fig 4.13 -Bar-plot for residence_type & stroke data

**smoking_status & stroke**

```
first_looking("smoking_status")
```

```
column name    :  smoking_status
---------------------------------
per_of_nulls   :  % 0.0
num_of_nulls   :  0
num_of_uniques :  4
never smoked       1892
Unknown            1544
formerly smoked     884
smokes              789
Name: smoking_status, dtype: int64
```

```
print(df.groupby('smoking_status')['stroke'].mean().sort_values())
print()
df.groupby('smoking_status')['stroke'].mean().iplot(kind='bar',subplots=True,bins=50)
```

```
smoking_status
Unknown           0.030
never smoked      0.048
smokes            0.053
formerly smoked   0.079
Name: stroke, dtype: float64
```

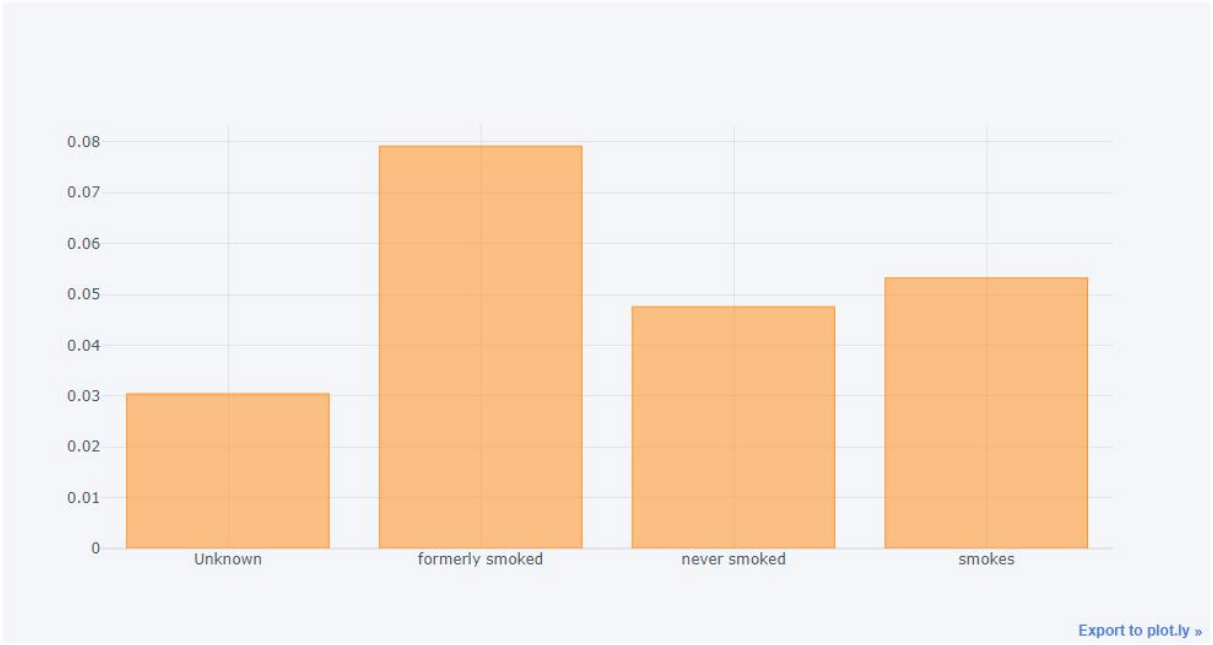



Fig 4.14 -Bar-plot for smoking_status & stroke data

Overall plot can be visualized by using Pairplot:

```
sns.pairplot(df, hue="stroke");
```

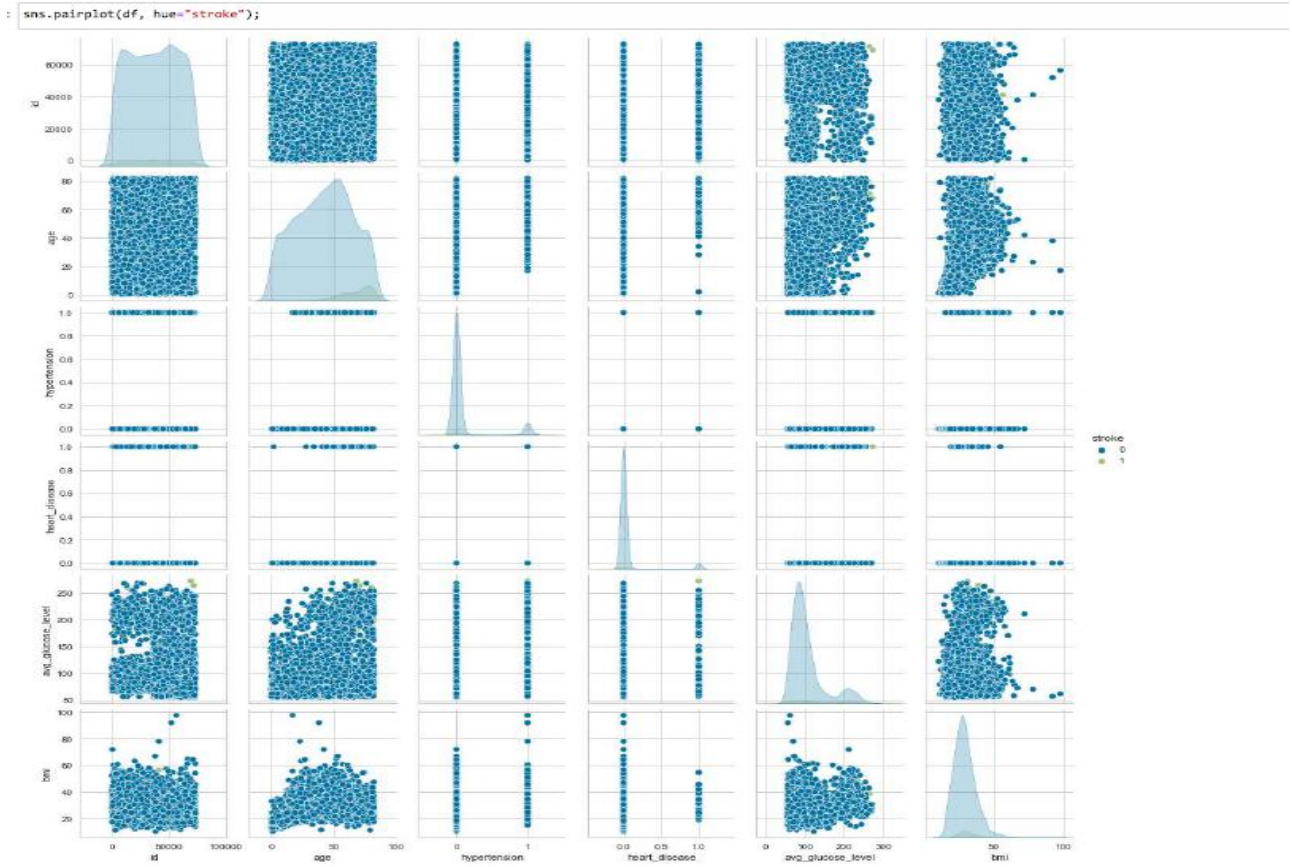



Fig 4.15 -pair-plot for categorical features

## 4.6 LABEL ENCODER OPERATION

Label Encoding refers to converting the labels into a numeric form so as to convert them into the machine-readable form. Machine learning algorithms can then decide in a better way how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning. For example, if a dataset contains a variable 'Gender' with labels 'Male' and 'Female', then the label encoder would convert these labels into a number format and the resultant outcome would be [0,1].

```
from sklearn.preprocessing import LabelEncoder
LE=LabelEncoder()
```

```
objList1 = df.select_dtypes(include = "object").columns # only columns with object datatypes
print (objList1)
```

```
Index(['gender', 'ever_married', 'work_type', 'Residence_type',
       'smoking_status'],
      dtype='object')
```

```
for i in objList1:
    df[i] = LE.fit_transform(df[i].astype(str))

print (df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 5110 non-null   int64  
 1   gender             5110 non-null   int32  
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64  
 4   heart_disease      5110 non-null   int64  
 5   ever_married       5110 non-null   int32  
 6   work_type          5110 non-null   int32  
 7   Residence_type     5110 non-null   int32  
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   int32  
 11  stroke             5110 non-null   int64  
dtypes: float64(3), int32(5), int64(4)
memory usage: 379.4 KB
None
```

Check with heatmap:

```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True,cmap='cool')
plt.xticks(rotation=45);
```

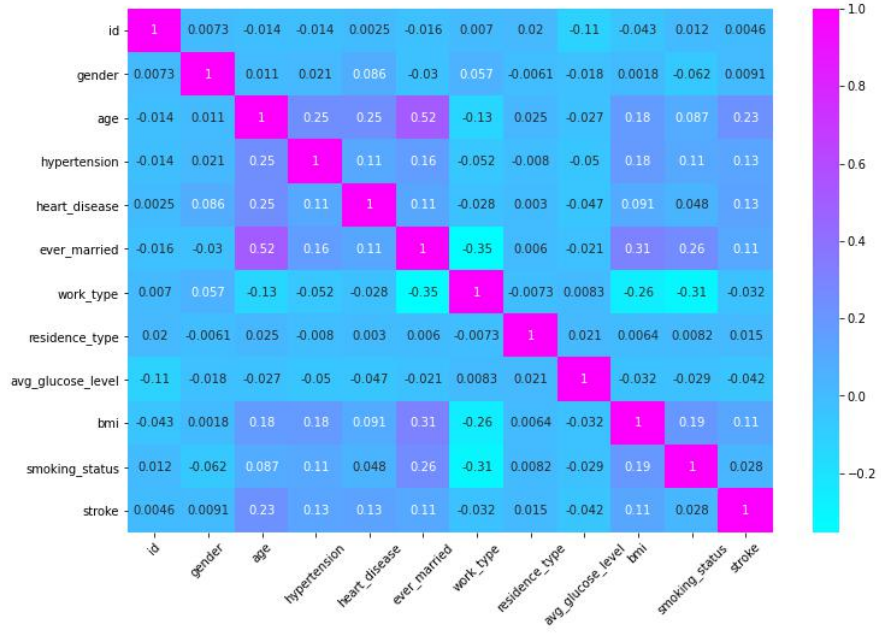



Fig 4.16 -Heat-map after Label Encoder

# CHAPTER 5 : TRAIN | TEST SPLIT & HANDLING MISSING VALUES

## 5.1 TRAIN AND TEST SPLIT

In machine learning, it is a common practice to split your data into two different sets. These two sets are the training set and the testing set. As the name suggests, the training set is used for training the model and the testing set is used for testing the accuracy of the model.

While training a machine learning model we are trying to find a pattern that best represents all the data points with minimum error. While doing so, two common errors come up. These are overfitting and underfitting.

**Underfitting:**Underfitting is when the model is not even able to represent the data points in the training dataset. In the case of under-fitting, you will get a low accuracy even when testing on the training dataset.Under fitting usually means that your model is too simple to capture the complexities of the dataset.

**Overfitting:** Overfitting is the case when your model represents the training dataset a little too accurately. This means that your model fits too closely. In the case of overfitting, your model will not be able to perform well on new unseen data. Overfitting is usually a sign of model being too complex.

**Random state:** Random state ensures that the splits that you generate are reproducible. Scikit-learn uses random permutations to generate the splits. The random state that you provide is used as a seed to the random number generator. This ensures that the random numbers are generated in the same order.

We must separate the columns (attributes or features) of the dataset into input patterns (X) and output patterns (y), after splitting of train and test data with 80% as train data and 20% as test data.(test_size=0.2)

```
X = df.drop(["stroke"], axis=1)
y = df["stroke"]
```

Finally, we can split the X and Y data into a training and test dataset. The training set will be used to prepare the models used in this dataset, the test set will be used to make new predictions, from which we can evaluate the performance of the model.For this we will use the train_test_split() function from the scikit-learn library.

**Train/Test and split**

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)
```

**Test_size -** 0.2 (assigning test size as 20%)
**X_train -** consists of 80% of independent trained data
**X_test -** consists of 20% of independent test data
**Y_train -** consists of 80% of dependant trained data
**Y_test -** consists of 20% of dependant test data

## 5.2 HANDLING MISSING VALUES

To prevent data leakage we prefer to handle with 201 missing values stored in "bmi" column after train-test split.It would be more sensible to fill the missing values in "bmi" column with the median by taking the effects of outliers on mean into consideration. For handling missing values Simple Imputer is used

Simple Imputer is a class found in package sklearn.impute. It is used to impute / replace the numerical or categorical missing data related to one or more features with appropriate values.Having imputed train and test sets separately so that we did not have any data leakage. The strategy used in our data is median.

```
from sklearn.impute import SimpleImputer

# Imputation
my_imputer = SimpleImputer(missing_values=np.nan, strategy="median")
imputed_X_train = pd.DataFrame(my_imputer.fit_transform(X_train))
imputed_X_test = pd.DataFrame(my_imputer.transform(X_test))

# Imputation removed column names; put them back
imputed_X_train.columns = X_train.columns
imputed_X_test.columns = X_test.columns
```

```
X_test = imputed_X_test
X_train = imputed_X_train
```

Cross check whether their is any missing values left in train and test datas

```
missing(X_train)
```

| | Missing_Number | Missing_Percent |
|---|---|---|
| id | 0 | 0.000 |
| gender | 0 | 0.000 |
| age | 0 | 0.000 |
| hypertension | 0 | 0.000 |
| heart_disease | 0 | 0.000 |
| ever_married | 0 | 0.000 |
| work_type | 0 | 0.000 |
| residence_type | 0 | 0.000 |
| avg_glucose_level | 0 | 0.000 |
| bmi | 0 | 0.000 |
| smoking_status | 0 | 0.000 |

```
missing(X_test)
```

| | Missing_Number | Missing_Percent |
|---|---|---|
| id | 0 | 0.000 |
| gender | 0 | 0.000 |
| age | 0 | 0.000 |
| hypertension | 0 | 0.000 |
| heart_disease | 0 | 0.000 |
| ever_married | 0 | 0.000 |
| work_type | 0 | 0.000 |
| residence_type | 0 | 0.000 |
| avg_glucose_level | 0 | 0.000 |
| bmi | 0 | 0.000 |
| smoking_status | 0 | 0.000 |

By checking with user defined function on missing values, nulls are removed by the use of simple imputer with strategy=’median’

## 5.3 SCALING

Scaling is a pre-processing step. This technique used to normalize the range of independent variables. Variables that are used to determine the target variable are known as features.Raw data contains a variety of values. Some values have a small range (age) while some have a very large range (salary). And this wide range can lead to wrong results.The techniques for scaling are:

**StandardScaler:** Assumes that data has normally distributed features and will scale them to zero mean and 1 standard deviation. Use StandardScaler() if you know the data distribution is normal. For most cases StandardScaler would do no harm. Especially when dealing with variance (PCA, clustering, logistic regression, SVM, perceptron, neural networks) in fact Standard Scaler would be very important. On the other hand it will not make much of a difference if you are using tree based classifiers or regressor.

$$z = \frac{x - \mu}{\sigma}$$

**MinMaxScaler :** This will transform each value in the column proportionally within the range [0,1]. This is quite acceptable in cases where we are not concerned about the standardization along the variance axes. e.g. image processing or neural networks expecting values between 0 to 1.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

In our project we are using the technique, MinMaxScaler()

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

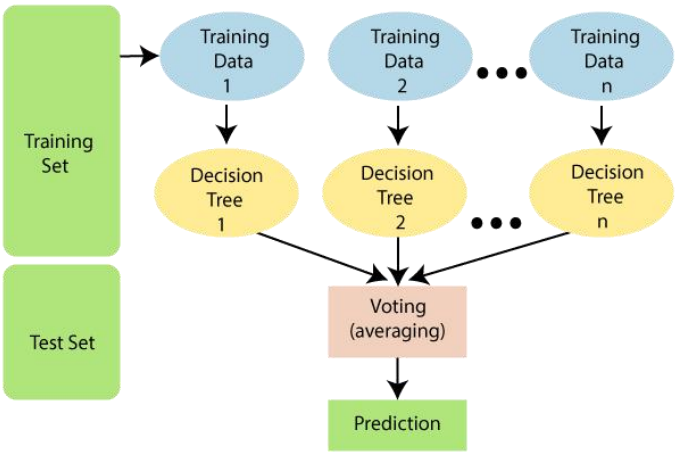
# CHAPTER 6 : MODEL IMPLEMENTATION

## 6.1 IMPLEMENTATION OF RANDOM FOREST (RF)

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.




Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:

- ✧ There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- ✧ The predictions from each tree must have very low correlations.

Below are some points that explain why we should use the Random Forest algorithm:

- ✧ It takes less training time as compared to other algorithms.
- ✧ It predicts output with high accuracy, even for the large dataset it runs efficiently.
- ✧ It can also maintain accuracy when a large proportion of data is missing.

**Advantages of Random Forest**

- ✧ Random Forest is capable of performing both Classification and Regression tasks.
- ✧ It is capable of handling large datasets with high dimensionality.
- ✧ It enhances the accuracy of the model and prevents the overfitting issue.

**Disadvantages of Random Forest**

- ✧ Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

## 6.1.A MODELLING RANDOM FOREST WITH DEFAULT PARAMETERS.

The model is build with default parameters by assigning the class weights as balanced and by assigning the random state and check its performance by accuracy,F1_score,recall_score,confusion_matrix,classification _report and roc_auc_score.

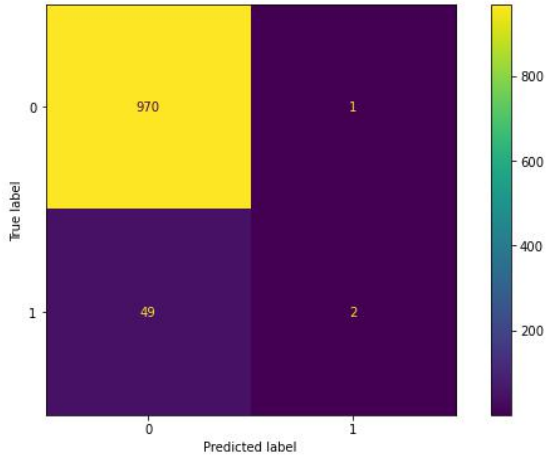
```
RF_model = RandomForestClassifier(class_weight="balanced", random_state=101)
RF_model.fit(x_train_sm, y_train_sm)
y_pred = RF_model.predict(X_test_scaled)
y_train_pred = RF_model.predict(X_train_scaled)

rf_f1 = f1_score(y_test, y_pred)
rf_acc = accuracy_score(y_test, y_pred)
rf_recall = recall_score(y_test, y_pred)
rf_auc = roc_auc_score(y_test, y_pred)

print(confusion_matrix(y_test, y_pred))
print("\033[1m-------------------------------------------------------\033[0m")
print(classification_report(y_test, y_pred))
print("\033[1m-------------------------------------------------------\033[0m")

plot_confusion_matrix(RF_model, X_test_scaled, y_test)

train_val(y_train, y_train_pred, y_test, y_pred)
```

```
[[970   1]
 [ 49   2]]
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       971
           1       0.67      0.04      0.07        51

    accuracy                           0.95      1022
   macro avg       0.81      0.52      0.52      1022
weighted avg       0.94      0.95      0.93      1022

-------------------------------------------------------
```

### Accuracy and Confusion Matrix:

:

| | train_set | test_set |
|---|---|---|
| Accuracy | 1.000 | 0.951 |
| Precision | 1.000 | 0.667 |
| Recall | 1.000 | 0.039 |
| f1 | 1.000 | 0.074 |



## 6.1.B CROSS-VALIDATING RANDOM FOREST ALGORITHM

Cross-validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of the input data. We can also say that it is a technique to check how a statistical model generalizes to an independent dataset.

```
rf_xvalid_model = RandomForestClassifier(max_depth=None, random_state=101)

rf_xvalid_model_scores = cross_validate(rf_xvalid_model, X_train_scaled,
                                        y_train,scoring=["accuracy", "precision", "recall", "f1"],cv=10)
rf_xvalid_model_scores = pd.DataFrame(rf_xvalid_model_scores, index = range(1, 11))

rf_xvalid_model_scores
```

| | fit_time | score_time | test_accuracy | test_precision | test_recall | test_f1 |
|---|---|---|---|---|---|---|
| 1 | 0.401 | 0.018 | 0.951 | 0.000 | 0.000 | 0.000 |
| 2 | 0.396 | 0.018 | 0.951 | 0.500 | 0.050 | 0.091 |
| 3 | 0.375 | 0.017 | 0.951 | 0.000 | 0.000 | 0.000 |
| 4 | 0.369 | 0.016 | 0.949 | 0.000 | 0.000 | 0.000 |
| 5 | 0.371 | 0.017 | 0.949 | 0.333 | 0.050 | 0.087 |
| 6 | 0.379 | 0.019 | 0.951 | 0.000 | 0.000 | 0.000 |
| 7 | 0.410 | 0.018 | 0.949 | 0.000 | 0.000 | 0.000 |
| 8 | 0.422 | 0.017 | 0.956 | 1.000 | 0.053 | 0.100 |
| 9 | 0.400 | 0.019 | 0.956 | 1.000 | 0.053 | 0.100 |
| 10 | 0.387 | 0.018 | 0.951 | 0.000 | 0.000 | 0.000 |

```
rf_xvalid_model_scores.mean()[2:]
```

```
test_accuracy    0.951
test_precision   0.283
test_recall      0.021
test_f1          0.038
dtype: float64
```

## 6.1.C MODELLING RANDOM FOREST WITH BEST PARAMETERS USING GRIDSEARCHCV

The model is further proceed by checking the best parameters of the given data, by tuning the parameters of the Random forest classifier using GridSearchCV().GridSearchCV tries all the combinations of the values passed in the dictionary and evaluates the model for each combination. Hence after using this function we get accuracy/loss for every combination of hyperparameters and we can choose the one with the best performance.The parameters used in our model is:

**For param_grid (parameters)**
n_estimators - the number of trees in the forest
max_features - number of trees to consider for best split
max_depth - maximum depth of the tree
min_samples_split - minimum number of samples required to split an internal node

**For GridSearchCV**
Estimator - pass the model instance for which you want to check the hyperparameters
Param_grid - the dictionary object that holds the hyperparameters you want to try
Scoring - evaluation metric (‘recall’)
N_jobs - number of processes you wish to run in parallel (-1 for using all available processors)
Verbose - set it to 1 to get the detailed printout while you fit the data to GridSearchCV

```
param_grid = {'n_estimators':[50, 100, 300],
              'max_features':[2, 3, 4],
              'max_depth':[3, 5, 7, 9],
              'min_samples_split':[2, 5, 8]}
```

```
RF_grid_model = RandomForestClassifier(random_state=101)

RF_grid_model = GridSearchCV(estimator=RF_grid_model,
                             param_grid=param_grid,
                             scoring = "recall",
                             n_jobs = -1, verbose = 2).fit(X_train_scaled, y_train)
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
```

**Let's look at the best parameters & estimator found by GridSearchCV.**

```
print(colored('\033[1mBest Parameters of GridSearchCV for Random Forest Model:\033[0m', 'blue'),
      colored(RF_grid_model.best_params_, 'cyan'))
print("----------------------------------------------------------------------------------------------------------------------------------")
print(colored('\033[1mBest Estimator of GridSearchCV for Random Forest Model:\033[0m', 'blue'),
      colored(RF_grid_model.best_estimator_, 'cyan'))
```

```
Best Parameters of GridSearchCV for Random Forest Model: {'max_depth': 9, 'max_features': 4, 'min_samples_split': 2, 'n_estimat
ors': 50}
----------------------------------------------------------------------------------------------------------------------------------
Best Estimator of GridSearchCV for Random Forest Model: RandomForestClassifier(max_depth=9, max_features=4, n_estimators=50,
                       random_state=101)
```

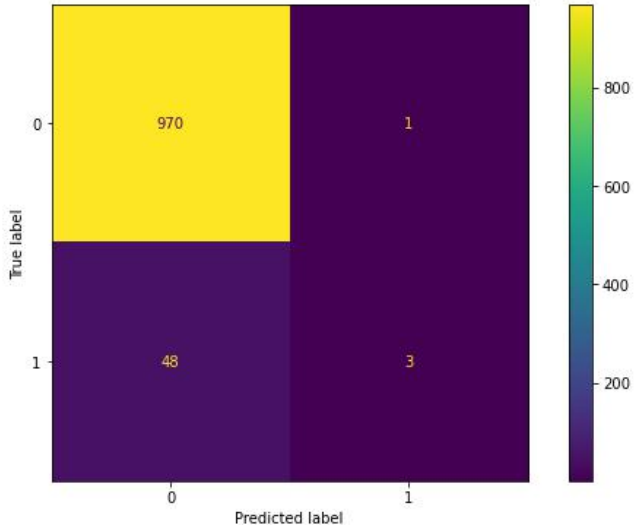
## Fit the model with Best parameters and check the performance:

```
y_pred = RF_grid_model.predict(X_test_scaled)
y_train_pred = RF_grid_model.predict(X_train_scaled)

rf_grid_f1 = f1_score(y_test, y_pred)
rf_grid_acc = accuracy_score(y_test, y_pred)
rf_grid_recall = recall_score(y_test, y_pred)
rf_grid_auc = roc_auc_score(y_test, y_pred)

print(confusion_matrix(y_test, y_pred))
print("\033[1m-------------------------------------------------------------\033[0m")
print(classification_report(y_test, y_pred))
print("\033[1m-------------------------------------------------------------\033[0m")

plot_confusion_matrix(RF_grid_model, X_test_scaled, y_test)

train_val(y_train, y_train_pred, y_test, y_pred)
```

```
[[970   1]
 [ 48   3]]
-----------------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      1.00      0.98       971
           1       0.75      0.06      0.11        51

    accuracy                           0.95      1022
   macro avg       0.85      0.53      0.54      1022
weighted avg       0.94      0.95      0.93      1022

-----------------------------------------------------------
```

**Accuracy and Confusion Matrix:**

| | train_set | test_set |
|---|---|---|
| Accuracy | 0.965 | 0.952 |
| Precision | 1.000 | 0.750 |
| Recall | 0.278 | 0.059 |
| f1 | 0.435 | 0.109 |




## 6.1.D FEATURE IMPORTANCE FOR RANDOM FOREST

The feature importance (variable importance) describes which features are relevant. It can help with better understanding of the solved problem and sometimes lead to model improvements by employing the feature selection.

```
RF_model.feature_importances_
```

```
array([0.12864406, 0.01925905, 0.35099212, 0.02625058, 0.01839643,
       0.03518394, 0.04955707, 0.01843739, 0.14332351, 0.16855393,
       0.04140191])
```

```
RF_feature_imp = pd.DataFrame(index = X.columns, data = RF_model.feature_importances_,
                              columns = ["Feature Importance"]).sort_values("Feature Importance", ascending = False)
RF_feature_imp
```

| | Feature Importance |
|---|---|
| age | 0.351 |
| bmi | 0.169 |
| avg_glucose_level | 0.143 |
| id | 0.129 |
| work_type | 0.050 |
| smoking_status | 0.041 |
| ever_married | 0.035 |
| hypertension | 0.026 |
| gender | 0.019 |
| residence_type | 0.018 |
| heart_disease | 0.018 |

**Visualizing Feature Importance in RF:**

```
sns.barplot(x=RF_feature_imp["Feature Importance"], y=RF_feature_imp.index)
plt.title("Feature Importance")
plt.show()
```

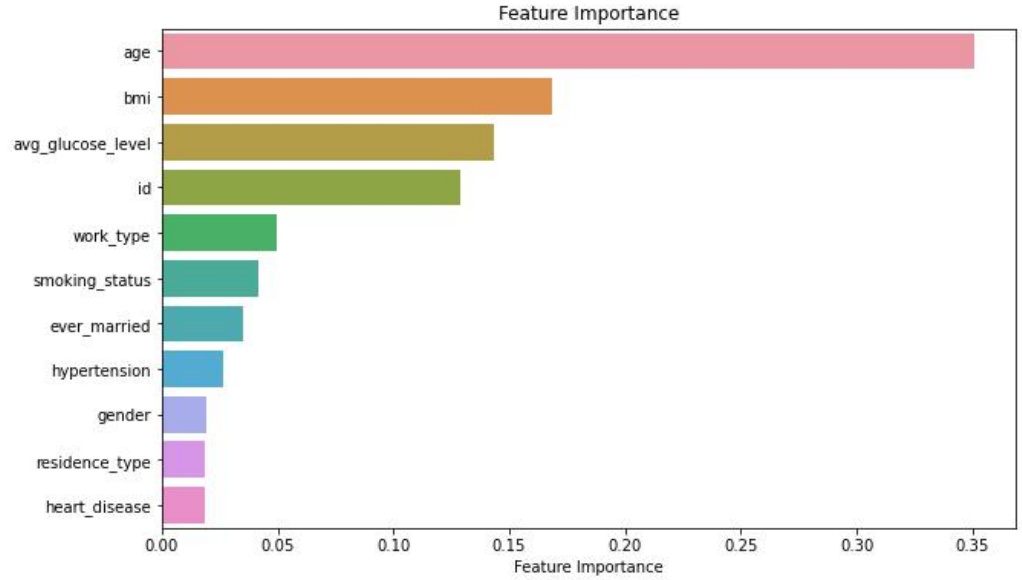



Fig 6.1 -Bar-plot for Feature Importance in RF

## 6.1.E ROC (Receiver Operating Curve) AND AUC (Area Under Curve).

The ROC curve stands for Receiver Operating Characteristic curve. ROC curves display the performance of a classification model.ROC tells us how good the model is for distinguishing between the given classes, in terms of the predicted probability.

The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.

```
plot_roc_curve(RF_grid_model, X_test_scaled, y_test);
```

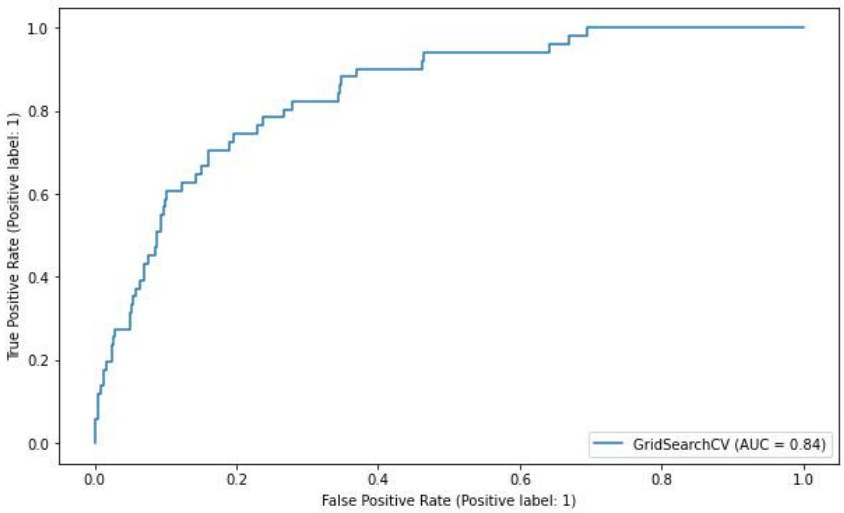



Fig 6.2 -Roc Curve for Random Forest

```
plot_precision_recall_curve(RF_grid_model, X_test_scaled, y_test);
```

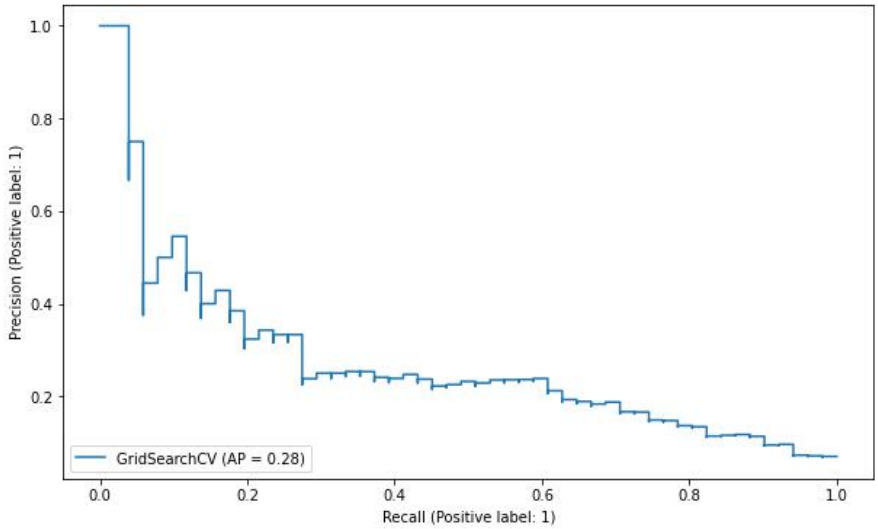



Fig 6.3 -Recall Curve for Random Forest

By checking with the Random Forest Classification Model

We get the output as,

**Accuracy of the model:**
For Train data - 96.5%
For Test data - 95.2%

**Precision of the model:**
For Train data - 1.0
For Test data - 0.750

**Recall of the model:**
For Train data - 0.278
For Test data - 0.059

**F1 score of the model:**
For Train data - 0.435
For Test data - 0.109

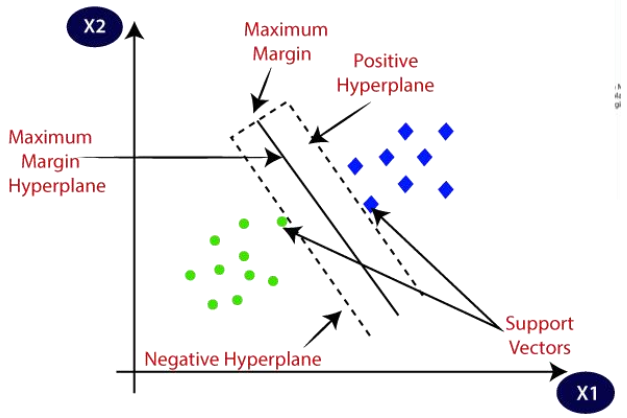
**AUC score of the Model in Random Forest** - 0.84

## 6.2 IMPLEMENTATION OF SUPPORT VECTOR MACHINE (SVM)

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

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. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:




**Hyperplane and Support Vectors in the SVM algorithm:**

**Hyperplane:** There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

**Support Vectors:** The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

**Types of SVM**

**SVM can be of two types:**

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

**Advantages of Support Vector Machine (SVM)**

- ✧ Regularization Capabilities
- ✧ Handles non-linear data efficiently
- ✧ Solves both classification and regression problems
- ✧ Stability

**Disadvantages of Support Vector Machine (SVM)**

- ✧ Choosing an appropriate kernel function is difficult
- ✧ Extensive memory requirement
- ✧ Difficult to interpret
- ✧ Long training time

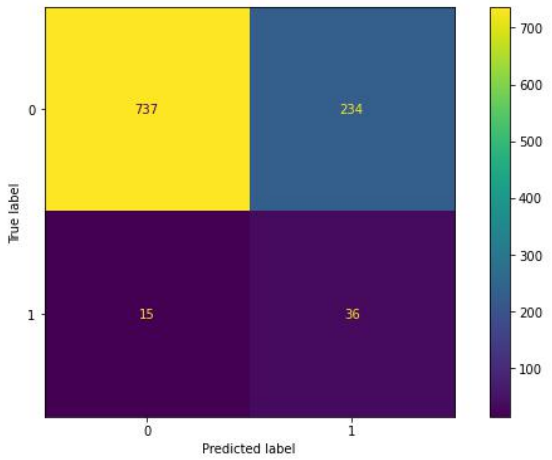
## 6.2.A MODELLING SUPPORT VECTOR MACHINE (SVM) WITH DEFAULT PARAMETERS.

The model is build with default parameters by assigning the class weights as balanced and check its performance by accuracy,F1_score,recall_score,confusion_matrix,classification _report and roc_auc_score.

```
svm_model_scaled = SVC(class_weight = "balanced")
svm_model_scaled.fit(X_train_scaled, y_train)
y_pred = svm_model_scaled.predict(X_test_scaled)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[737 234]
 [ 15  36]]
              precision    recall  f1-score   support

           0       0.98      0.76      0.86       971
           1       0.13      0.71      0.22        51

    accuracy                           0.76      1022
   macro avg       0.56      0.73      0.54      1022
weighted avg       0.94      0.76      0.82      1022
```

```
plot_confusion_matrix(svm_model_scaled, X_test_scaled, y_test);
```



## 6.2.B CROSS-VALIDATING SUPPORT VECTOR MACHINE (SVM) ALGORITHM

Cross-validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of the input data. We can also say that it is a technique to check how a statistical model generalizes to an independent dataset.

```
svm_xvalid_model = SVC()

svm_xvalid_model_scores = cross_validate(svm_xvalid_model, X_train_scaled, y_train, scoring = ['accuracy', 'precision','recall',
                                                                                                'f1'], cv = 10)
svm_xvalid_model_scores = pd.DataFrame(svm_xvalid_model_scores, index = range(1, 11))

svm_xvalid_model_scores
```

| | fit_time | score_time | test_accuracy | test_precision | test_recall | test_f1 |
|---|---|---|---|---|---|---|
| 1 | 0.194 | 0.031 | 0.951 | 0.000 | 0.000 | 0.000 |
| 2 | 0.191 | 0.031 | 0.951 | 0.000 | 0.000 | 0.000 |
| 3 | 0.184 | 0.032 | 0.951 | 0.000 | 0.000 | 0.000 |
| 4 | 0.178 | 0.031 | 0.951 | 0.000 | 0.000 | 0.000 |
| 5 | 0.176 | 0.030 | 0.951 | 0.000 | 0.000 | 0.000 |
| 6 | 0.172 | 0.032 | 0.951 | 0.000 | 0.000 | 0.000 |
| 7 | 0.201 | 0.034 | 0.951 | 0.000 | 0.000 | 0.000 |
| 8 | 0.189 | 0.032 | 0.953 | 0.000 | 0.000 | 0.000 |
| 9 | 0.216 | 0.034 | 0.953 | 0.000 | 0.000 | 0.000 |
| 10 | 0.212 | 0.033 | 0.951 | 0.000 | 0.000 | 0.000 |

```
model = SVC(class_weight = "balanced")

scores = cross_validate(model, X_train_scaled, y_train, scoring = ['accuracy', 'precision','recall','f1'], cv = 10)
df_scores = pd.DataFrame(scores, index = range(1, 11))
df_scores.mean()[2:]
```

```
test_accuracy    0.754
test_precision   0.125
test_recall      0.678
test_f1          0.211
dtype: float64
```

## 6.2.C MODELLING SUPPORT VECTOR MACHINE (SVM) WITH BEST PARAMETERS USING GRIDSEARCHCV

A machine learning algorithm requires certain hyperparameters that must be tuned before training. An optimal subset of these hyperparameters must be selected, which is called hyperparameter optimization. Grid-Search is a sci-kit learn package that provides for hyperparameter tuning. A grid search space is generated by taking the initial set of values given to each hyperparameter. Each cell in the grid is searched for the optimal solution.

**For param_grid (parameters)**

C - C value adds a penalty each time an item is misclassified

gamma - If the gamma value is low, all training points have an influence on the decision line
If the gamma value is high, the radius of similarity is low and influence is limited only to the nearby points

Kernel - rbf and linear

**For GridSearchCV**

Estimator - pass the model instance for which you want to check the hyperparameters
Param_grid - the dictionary object that holds the hyperparameters you want to try
Verbose - set it to 1 to get the detailed printout while you fit the data to GridSearchCV
Refit - True, the GridSearchCV will be refitted with the best scoring parameter combination on the whole data that is passed in fit ().

```
from sklearn.model_selection import GridSearchCV
```

```
param_grid = {'C': [0.1,1, 10, 100, 1000],
              'gamma': ["scale", "auto", 1,0.1,0.01,0.001,0.0001],
              'kernel': ['rbf', 'linear']}
```

```
model = SVC(class_weight = "balanced")
svm_model_grid = GridSearchCV(model, param_grid, verbose=3, refit=True)
```

```
svm_model_grid.fit(X_train_scaled, y_train)
```

```
Fitting 5 folds for each of 70 candidates, totalling 350 fits
[CV 1/5] END .................C=0.1, gamma=scale, kernel=rbf; total time=   0.6s
[CV 2/5] END .................C=0.1, gamma=scale, kernel=rbf; total time=   0.5s
[CV 3/5] END .................C=0.1, gamma=scale, kernel=rbf; total time=   0.5s
[CV 4/5] END .................C=0.1, gamma=scale, kernel=rbf; total time=   0.5s
[CV 5/5] END .................C=0.1, gamma=scale, kernel=rbf; total time=   0.5s
[CV 1/5] END ..............C=0.1, gamma=scale, kernel=linear; total time=   0.2s
[CV 2/5] END ..............C=0.1, gamma=scale, kernel=linear; total time=   0.2s
[CV 3/5] END ..............C=0.1, gamma=scale, kernel=linear; total time=   0.2s
[CV 4/5] END ..............C=0.1, gamma=scale, kernel=linear; total time=   0.2s
[CV 5/5] END ..............C=0.1, gamma=scale, kernel=linear; total time=   0.2s
[CV 1/5] END ..................C=0.1, gamma=auto, kernel=rbf; total time=   0.6s
[CV 2/5] END ..................C=0.1, gamma=auto, kernel=rbf; total time=   0.6s
[CV 3/5] END ..................C=0.1, gamma=auto, kernel=rbf; total time=   0.6s
[CV 4/5] END ..................C=0.1, gamma=auto, kernel=rbf; total time=   0.6s
[CV 5/5] END ..................C=0.1, gamma=auto, kernel=rbf; total time=   0.6s
[CV 1/5] END ................C=0.1, gamma=auto, kernel=linear; total time=   0.2s
[CV 2/5] END ................C=0.1, gamma=auto, kernel=linear; total time=   0.2s
[CV 3/5] END ................C=0.1, gamma=auto, kernel=linear; total time=   0.2s
[CV 4/5] END ................C=0.1, gamma=auto, kernel=linear; total time=   0.2s
[CV 5/5] END ................C=0.1, gamma=auto, kernel=linear; total time=   0.2s
[CV 1/5] END ....................C=0.1, gamma=1, kernel=rbf; total time=   0.5s
[CV 2/5] END ....................C=0.1, gamma=1, kernel=rbf; total time=   0.5s
[CV 3/5] END ....................C=0.1, gamma=1, kernel=rbf; total time=   0.5s
[CV 4/5] END ....................C=0.1, gamma=1, kernel=rbf; total time=   0.6s
[CV 5/5] END ....................C=0.1, gamma=1, kernel=rbf; total time=   0.5s
[CV 1/5] END ................C=0.1, gamma=1, kernel=linear; total time=   0.3s
[CV 2/5] END ................C=0.1, gamma=1, kernel=linear; total time=   0.2s
[CV 3/5] END ................C=0.1, gamma=1, kernel=linear; total time=   0.2s
[CV 4/5] END ................C=0.1, gamma=1, kernel=linear; total time=   0.2s
[CV 5/5] END ................C=0.1, gamma=1, kernel=linear; total time=   0.2s
[CV 1/5] END ................C=1000, gamma=0.1, kernel=rbf; total time=   1.8s
[CV 2/5] END ................C=1000, gamma=0.1, kernel=rbf; total time=   1.4s
[CV 3/5] END ................C=1000, gamma=0.1, kernel=rbf; total time=   1.3s
[CV 4/5] END ................C=1000, gamma=0.1, kernel=rbf; total time=   1.0s
[CV 5/5] END ................C=1000, gamma=0.1, kernel=rbf; total time=   1.0s
[CV 1/5] END ..............C=1000, gamma=0.1, kernel=linear; total time=   4.7s
[CV 2/5] END ..............C=1000, gamma=0.1, kernel=linear; total time=   5.7s
[CV 3/5] END ..............C=1000, gamma=0.1, kernel=linear; total time=   5.6s
[CV 4/5] END ..............C=1000, gamma=0.1, kernel=linear; total time=   5.0s
[CV 5/5] END ..............C=1000, gamma=0.1, kernel=linear; total time=   4.5s
[CV 1/5] END ...............C=1000, gamma=0.01, kernel=rbf; total time=   0.6s
[CV 2/5] END ...............C=1000, gamma=0.01, kernel=rbf; total time=   0.6s
[CV 3/5] END ...............C=1000, gamma=0.01, kernel=rbf; total time=   0.6s
[CV 4/5] END ...............C=1000, gamma=0.01, kernel=rbf; total time=   0.6s
[CV 5/5] END ...............C=1000, gamma=0.01, kernel=rbf; total time=   0.6s
[CV 1/5] END ............C=1000, gamma=0.01, kernel=linear; total time=   4.6s
[CV 2/5] END ............C=1000, gamma=0.01, kernel=linear; total time=   5.1s
[CV 3/5] END ............C=1000, gamma=0.01, kernel=linear; total time=   5.2s
[CV 4/5] END ............C=1000, gamma=0.01, kernel=linear; total time=   4.1s
[CV 5/5] END ............C=1000, gamma=0.01, kernel=linear; total time=   4.1s
[CV 1/5] END ..............C=1000, gamma=0.001, kernel=rbf; total time=   0.5s
[CV 2/5] END ..............C=1000, gamma=0.001, kernel=rbf; total time=   0.5s
[CV 3/5] END ..............C=1000, gamma=0.001, kernel=rbf; total time=   0.4s
[CV 4/5] END ..............C=1000, gamma=0.001, kernel=rbf; total time=   0.4s
[CV 5/5] END ..............C=1000, gamma=0.001, kernel=rbf; total time=   0.4s
[CV 1/5] END ............C=1000, gamma=0.001, kernel=linear; total time=   4.2s
[CV 2/5] END ............C=1000, gamma=0.001, kernel=linear; total time=   5.0s
[CV 3/5] END ............C=1000, gamma=0.001, kernel=linear; total time=   4.9s
[CV 4/5] END ............C=1000, gamma=0.001, kernel=linear; total time=   3.8s
[CV 5/5] END ............C=1000, gamma=0.001, kernel=linear; total time=   4.0s
[CV 1/5] END ..............C=1000, gamma=0.0001, kernel=rbf; total time=   0.5s
[CV 2/5] END ..............C=1000, gamma=0.0001, kernel=rbf; total time=   0.5s
[CV 3/5] END ..............C=1000, gamma=0.0001, kernel=rbf; total time=   0.5s
[CV 4/5] END ..............C=1000, gamma=0.0001, kernel=rbf; total time=   0.5s
[CV 5/5] END ..............C=1000, gamma=0.0001, kernel=rbf; total time=   0.5s
[CV 1/5] END ............C=1000, gamma=0.0001, kernel=linear; total time=   4.4s
[CV 2/5] END ............C=1000, gamma=0.0001, kernel=linear; total time=   5.3s
[CV 3/5] END ............C=1000, gamma=0.0001, kernel=linear; total time=   4.8s
[CV 4/5] END ............C=1000, gamma=0.0001, kernel=linear; total time=   3.8s
[CV 5/5] END ............C=1000, gamma=0.0001, kernel=linear; total time=   3.9s
GridSearchCV(estimator=SVC(class_weight='balanced'),
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': ['scale', 'auto', 1, 0.1, 0.01, 0.001,
                                   0.0001],
                         'kernel': ['rbf', 'linear']},
             verbose=3)
```

```
from termcolor import colored
print(colored('\033[1mBest Parameters of GridSearchCV for SVM Model:\033[0m', 'blue'), colored(svm_model_grid.best_params_,
                                                                                               'cyan'))
print("----------------------------------------------------------------------------------------------------------------------")
print(colored('\033[1mBest Estimator of GridSearchCV for SVM Model:\033[0m', 'blue'), colored(svm_model_grid.best_estimator_,
                                                                                             'cyan'))
```

```
Best Parameters of GridSearchCV for SVM Model: {'C': 1000, 'gamma': 1, 'kernel': 'rbf'}
----------------------------------------------------------------------------------------------------------------------
Best Estimator of GridSearchCV for SVM Model: SVC(C=1000, class_weight='balanced', gamma=1)
```

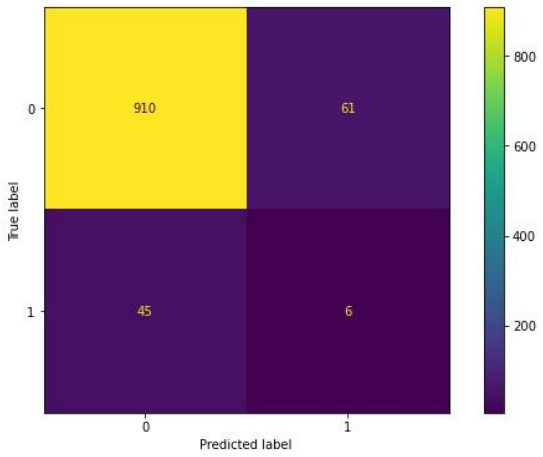
## Fit the model with Best parameters and check the performance:

```
y_pred =svm_model_grid.predict(X_test_scaled)
y_train_pred = svm_model_grid.predict(X_train_scaled)

svm_grid_f1 = f1_score(y_test, y_pred)
svm_grid_acc = accuracy_score(y_test, y_pred)
svm_grid_recall = recall_score(y_test, y_pred)
svm_grid_auc = roc_auc_score(y_test, y_pred)

print('-----------CONFUSION MATRIX------------')
print(confusion_matrix(y_test, y_pred))
print("\033[1m-------------------------------------------------------\033[0m")
print('-----------CLASSIFICATION REPORT---------')
print(classification_report(y_test, y_pred))
print("\033[1m-------------------------------------------------------\033[0m")

plot_confusion_matrix(svm_model_grid, X_test_scaled, y_test)

train_val(y_train, y_train_pred, y_test, y_pred)
```

```
-----------CONFUSION MATRIX------------
[[910  61]
 [ 45   6]]
---------------------------------------------------------
-----------CLASSIFICATION REPORT---------
              precision    recall  f1-score   support

           0       0.95      0.94      0.94       971
           1       0.09      0.12      0.10        51

    accuracy                           0.90      1022
   macro avg       0.52      0.53      0.52      1022
weighted avg       0.91      0.90      0.90      1022

---------------------------------------------------------
```

## Accuracy and Confusion Matrix:

| | train_set | test_set |
|---|---|---|
| Accuracy | 0.982 | 0.896 |
| Precision | 0.733 | 0.090 |
| Recall | 1.000 | 0.118 |
| f1 | 0.846 | 0.102 |



## 6.2.D ROC (Receiver Operating Curve) AND AUC (Area Under Curve).

The ROC curve stands for Receiver Operating Characteristic curve. ROC curves display the performance of a classification model.ROC tells us how good the model is for distinguishing between the given classes, in terms of the predicted probability.

The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.

```
plot_roc_curve(svm_model_grid, X_test_scaled, y_test);
```

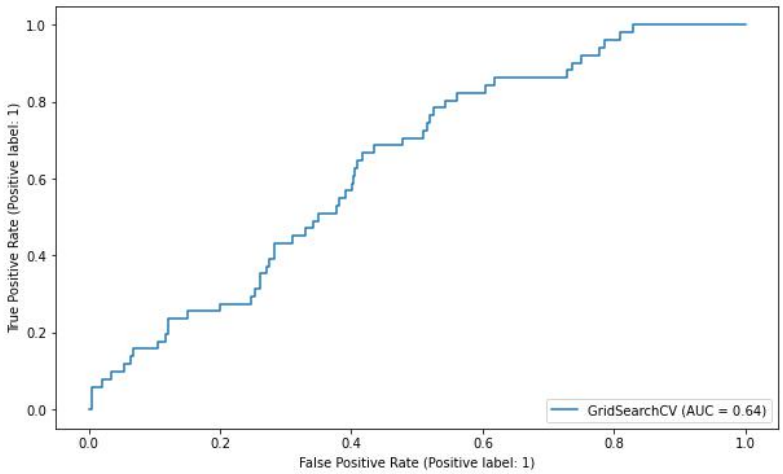



Fig 6.4 -Roc Curve for SVM

```
plot_precision_recall_curve(svm_model_grid, X_test_scaled, y_test);
```

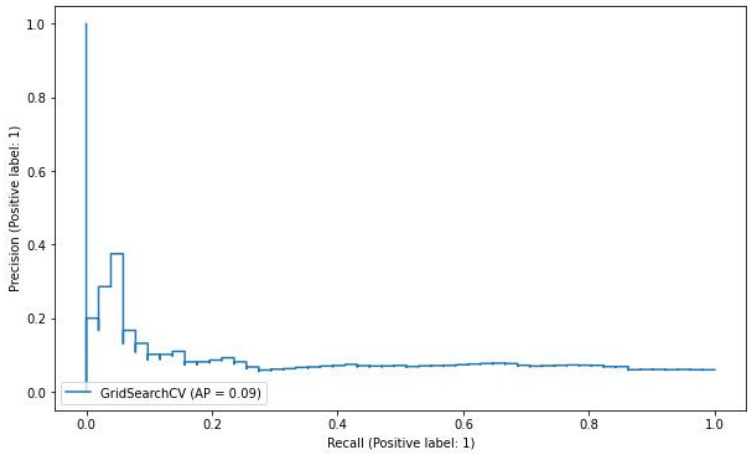



Fig 6.5 -Recall_Curve for SVM

By checking with the Support Vector Classification Model

We get the output as,

**Accuracy of the model:**
For Train data - 98.2%
For Test data - 89.6%

**Precision of the model:**
For Train data - 0.733
For Test data - 0.090

**Recall of the model:**
For Train data - 1.0
For Test data - 0.118

**F1 score of the model:**
For Train data - 0.846
For Test data - 0.102

**AUC score of the Model in Support Vector Machine** - 0.64

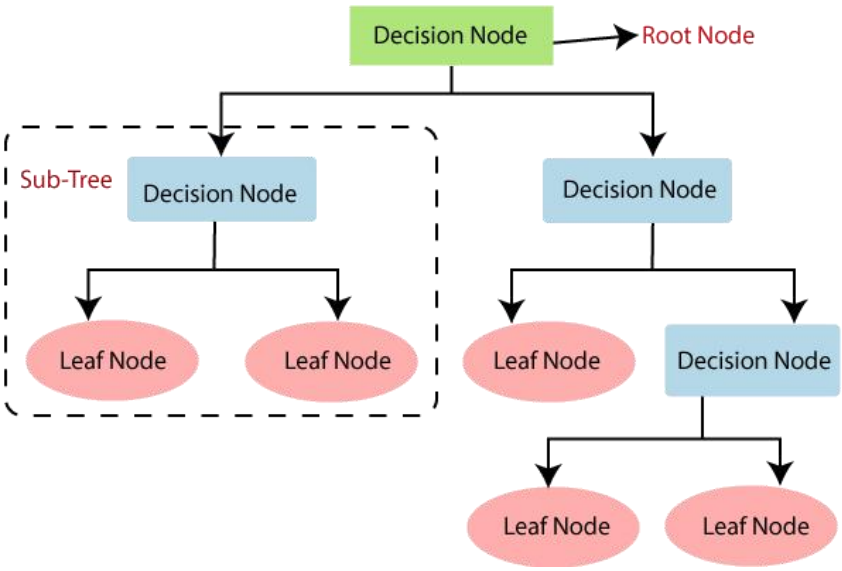
## 6.3 IMPLEMENTATION OF DECISION TREE (DT) ALGORITHM

- 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.
- The decisions or the test are performed on the basis of features of the given dataset.
- It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.

**Steps of Decision Tree:**

- Step-1: Begin the tree with the root node, says S, which contains the complete dataset.
- Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).
- Step-3: Divide the S into subsets that contains possible values for the best attributes.
- Step-4: Generate the decision tree node, which contains the best attribute.

- Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.




**Decision Tree Terminologies:**

· Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.

· Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.

· Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.

· Branch/Sub Tree: A tree formed by splitting the tree.

· Pruning: Pruning is the process of removing the unwanted branches from the tree.

· Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

**Advantages of the Decision Tree**

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

**Disadvantages of the Decision Tree**

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the Random Forest algorithm.
- For more class labels, the computational complexity of the decision tree may increase.

## 6.3.A MODELLING DECISION TREE (DT) WITH DEFAULT PARAMETERS.

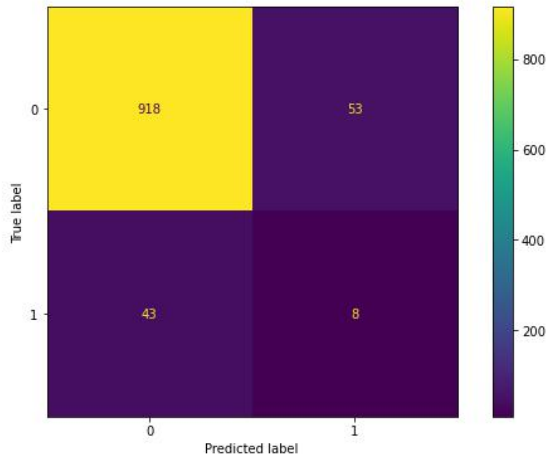
The model is build with default parameters and random state by assigning the class weights as balanced and check its performance by accuracy,F1_score,recall_score,confusion_matrix,classification _report and roc_auc_score.

```
DT_model = DecisionTreeClassifier(class_weight="balanced", random_state=42)
DT_model.fit(X_train_scaled, y_train)
y_pred = DT_model.predict(X_test_scaled)
y_train_pred = DT_model.predict(X_train_scaled)

dt_f1 = f1_score(y_test, y_pred)
dt_acc = accuracy_score(y_test, y_pred)
dt_recall = recall_score(y_test, y_pred)
dt_auc = roc_auc_score(y_test, y_pred)

print(confusion_matrix(y_test, y_pred))
print("\033[1m-------------------------------------------------------\033[0m")
print(classification_report(y_test, y_pred))
print("\033[1m-------------------------------------------------------\033[0m")

plot_confusion_matrix(DT_model, X_test_scaled, y_test)

train_val(y_train, y_train_pred, y_test, y_pred)
```

```
[[918  53]
 [ 43   8]]
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.96      0.95      0.95       971
           1       0.13      0.16      0.14        51

    accuracy                           0.91      1022
   macro avg       0.54      0.55      0.55      1022
weighted avg       0.91      0.91      0.91      1022

-------------------------------------------------------
```

### Accuracy and Confusion matrix

| | train_set | test_set |
|---|---|---|
| Accuracy | 1.000 | 0.906 |
| Precision | 1.000 | 0.131 |
| Recall | 1.000 | 0.157 |
| f1 | 1.000 | 0.143 |



## 6.3.B CROSS-VALIDATING DECISION TREE (DT) ALGORITHM

Cross-validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of the input data. We can also say that it is a technique to check how a statistical model generalizes to an independent dataset.

```
dt_xvalid_model = DecisionTreeClassifier(max_depth=None, random_state=42)

dt_xvalid_model_scores = cross_validate(dt_xvalid_model, X_train_scaled, y_train, scoring=["accuracy", "precision",
                                                                                           "recall", "f1"],cv = 10)
dt_xvalid_model_scores = pd.DataFrame(dt_xvalid_model_scores, index = range(1, 11))

dt_xvalid_model_scores
```

| | fit_time | score_time | test_accuracy | test_precision | test_recall | test_f1 |
|---|---|---|---|---|---|---|
| 1 | 0.015 | 0.004 | 0.917 | 0.111 | 0.100 | 0.105 |
| 2 | 0.012 | 0.003 | 0.914 | 0.174 | 0.200 | 0.186 |
| 3 | 0.013 | 0.003 | 0.912 | 0.192 | 0.250 | 0.217 |
| 4 | 0.013 | 0.003 | 0.919 | 0.118 | 0.100 | 0.108 |
| 5 | 0.013 | 0.004 | 0.905 | 0.087 | 0.100 | 0.093 |
| 6 | 0.013 | 0.003 | 0.914 | 0.105 | 0.100 | 0.103 |
| 7 | 0.013 | 0.003 | 0.900 | 0.138 | 0.200 | 0.163 |
| 8 | 0.013 | 0.004 | 0.900 | 0.077 | 0.105 | 0.089 |
| 9 | 0.013 | 0.004 | 0.912 | 0.095 | 0.105 | 0.100 |
| 10 | 0.013 | 0.004 | 0.902 | 0.045 | 0.050 | 0.048 |

```
dt_xvalid_model_scores.mean()[2:]
```

```
test_accuracy    0.909
test_precision   0.114
test_recall      0.131
test_f1          0.121
dtype: float64
```

## 6.3.C MODELLING DECISION TREE (DT) WITH BEST PARAMETERS USING RANDOMIZEDSEARCHCV

Random search is a method in which random combinations of hyperparameters are selected and used to train a model. The best random hyperparameter combinations are used. Random search bears some similarity to grid search.A key difference is that it does not test all parameters. Instead, the search is done at random. As we shall note later, a practical difference between the two is that RandomizedSearchCV allows us to specify the number of parameter values we seek to test.

**For Parameters:**
Criterion - supported criteria are Gini and Entropy
max_features - number of trees to consider for best split
max_depth - maximum depth of the tree
min_samples_split - minimum number of samples required to split an internal node

**For RandomizedSearchCV:**
Estimator - pass the model instance for which you want to check the hyperparameters
Param_distributions - the dictionary object that holds the hyperparameters you want to try
Cv - how many times to iterate or Cross Validate the data

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn import tree
```

```
parameters={'max_depth':(10,20,30,40,50),'criterion':('gini','entropy'),'max_features':('log2','sqrt','auto'),
            'min_samples_split':(2,4,6),'random_state':(0,1,2,3,4,5)}
```

```
DT_hp=RandomizedSearchCV(tree.DecisionTreeClassifier(),param_distributions=parameters,cv=5,random_state=2)
```

```
DT_fit=DT_hp.fit(X_train_scaled,y_train)
```

**Let's look at the best parameters & estimator found by RandomizedSearchCV.**

```
from termcolor import colored
print(colored('\033[1mBest Parameters of RandomizedSearchCV for Decision Tree Model:\033[0m', 'blue'),
      colored(DT_hp.best_params_, 'green'))
print("----------------------------------------------------------------------------------------------------------------------------------")
print(colored('\033[1mBest Estimator of RandomizedSearchCV for Decision Tree Model:\033[0m', 'blue'),
      colored(DT_hp.best_estimator_, 'green'))
```

```
Best Parameters of RandomizedSearchCV for Decision Tree Model: {'random_state': 2, 'min_samples_split': 6, 'max_features': 'aut
o', 'max_depth': 10, 'criterion': 'entropy'}
----------------------------------------------------------------------------------------------------------------------------------
Best Estimator of RandomizedSearchCV for Decision Tree Model: DecisionTreeClassifier(criterion='entropy', max_depth=10, max_fea
tures='auto',
                       min_samples_split=6, random_state=2)
```

## Fit the model with Best parameters and check the performance:

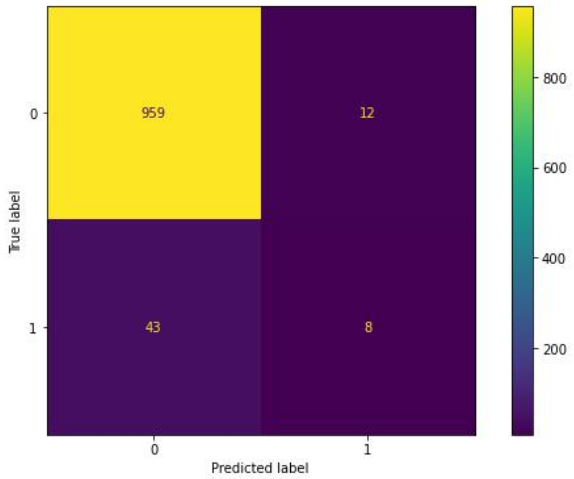
```
model_ht=model_after_ht.fit(X_train_scaled,y_train)
```

```
y_pred_DT =model_ht.predict(X_test_scaled)
y_train_pred_DT = model_ht.predict(X_train_scaled)

DT_grid_f1 = f1_score(y_test, y_pred_DT)
DT_grid_acc = accuracy_score(y_test, y_pred_DT)
DT_grid_recall = recall_score(y_test, y_pred_DT)
DT_grid_auc = roc_auc_score(y_test, y_pred_DT)

print('-----------CONFUSION MATRIX------------')
print(confusion_matrix(y_test, y_pred_DT))
print("\033[1m-------------------------------------------------------------\033[0m")
print('-----------CLASSIFICATION REPORT---------')
print(classification_report(y_test, y_pred_DT))
print("\033[1m-------------------------------------------------------------\033[0m")

print('--------------Plot for Confusion_matrix------------')
plot_confusion_matrix(model_ht, X_test_scaled, y_test)

print('---------------Train vs Test Report----------------')
train_val(y_train, y_train_pred_DT, y_test, y_pred_DT)
```

```
-----------CONFUSION MATRIX------------
[[959  12]
 [ 43   8]]
---------------------------------------------------------
-----------CLASSIFICATION REPORT---------
              precision    recall  f1-score   support

           0       0.96      0.99      0.97       971
           1       0.40      0.16      0.23        51

    accuracy                           0.95      1022
   macro avg       0.68      0.57      0.60      1022
weighted avg       0.93      0.95      0.93      1022

---------------------------------------------------------
```

**Accuracy and Confusion matrix:**

| | train_set | test_set |
|---|---|---|
| Accuracy | 0.963 | 0.946 |
| Precision | 0.836 | 0.400 |
| Recall | 0.283 | 0.157 |
| f1 | 0.423 | 0.225 |




## 6.3.D FEATURE IMPORTANCE FOR DECISION TREE (DT)

The feature importance (variable importance) describes which features are relevant. It can help with better understanding of the solved problem and sometimes lead to model improvements by employing the feature selection.

```
model_ht.feature_importances_
```

```
array([0.09670637, 0.01712398, 0.27383689, 0.05256498, 0.05963227,
       0.00167716, 0.08658213, 0.00348682, 0.13978366, 0.20698523,
       0.0616205 ])
```

```
DT_feature_imp = pd.DataFrame(index=X.columns, data = DT_model.feature_importances_,
                      columns = ["Feature Importance"]).sort_values("Feature Importance")
DT_feature_imp
```

| | Feature Importance |
|---|---|
| ever_married | 0.002 |
| residence_type | 0.003 |
| heart_disease | 0.014 |
| hypertension | 0.016 |
| gender | 0.019 |
| smoking_status | 0.030 |
| work_type | 0.056 |
| id | 0.141 |
| avg_glucose_level | 0.145 |
| bmi | 0.154 |
| age | 0.419 |

**Visualizing Feature Importance in DT:**

```
sns.barplot(x=DT_feature_imp["Feature Importance"], y=DT_feature_imp.index)
plt.title("Feature Importance")
plt.show()
```

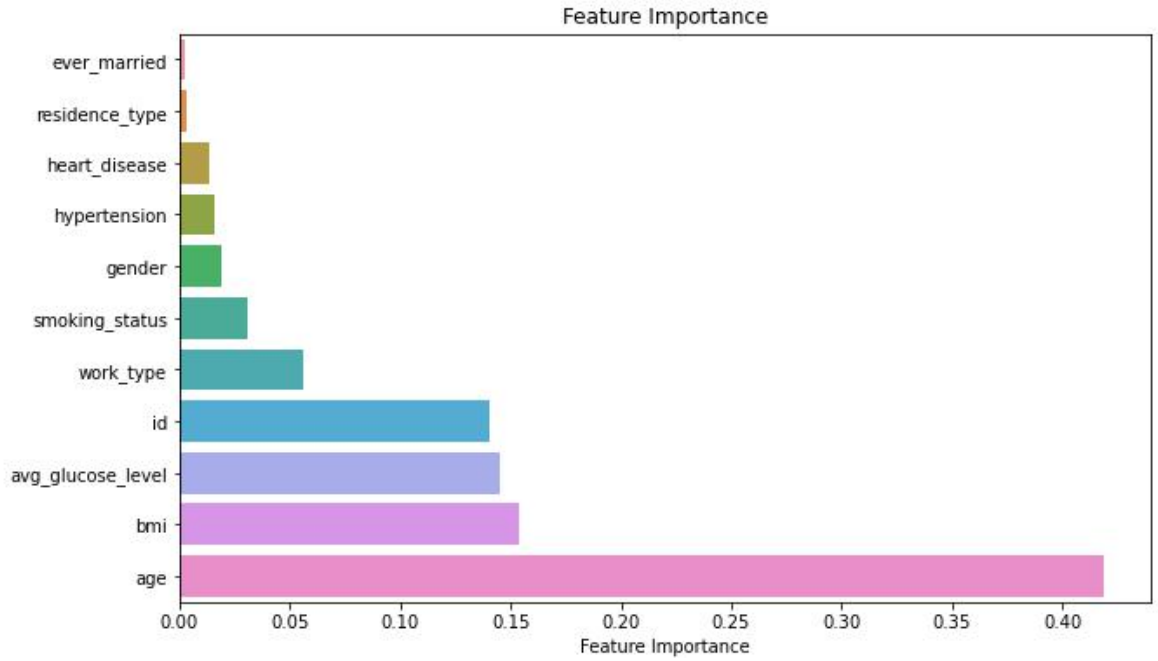



Fig 6.6 -Bar-plot for Feature Importance in DT

## 6.3.E ROC (Receiver Operating Curve) AND AUC (Area Under Curve).

The ROC curve stands for Receiver Operating Characteristic curve. ROC curves display the performance of a classification model.ROC tells us how good the model is for distinguishing between the given classes, in terms of the predicted probability.

The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.

```
plot_roc_curve(model_ht, X_test_scaled, y_test);
```

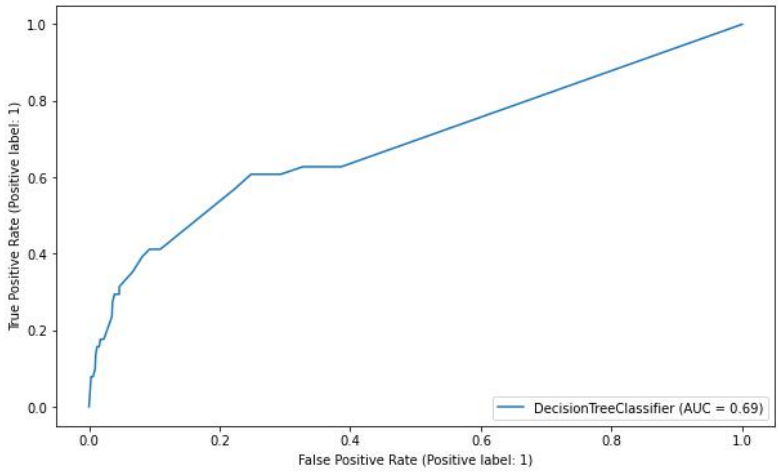



Fig 6.7 -Roc Curve for DT

```
plot_precision_recall_curve(model_ht, X_test_scaled, y_test);
```

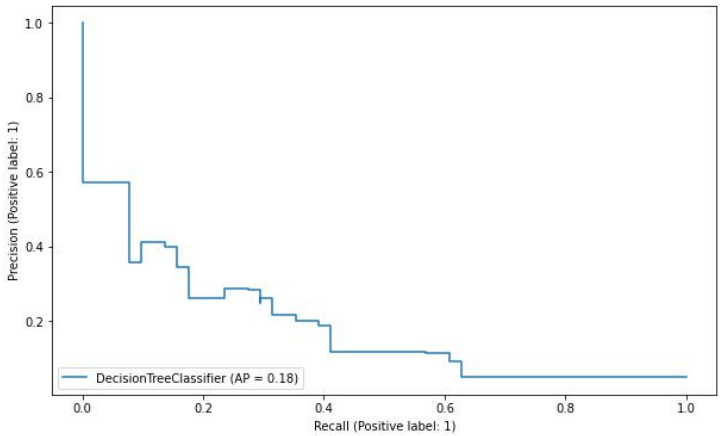



Fig 6.8 -Recall_Curve for DT

By checking with the Decision Tree Model

We get the output as,

**Accuracy of the model:**
For Train data - 96.3%
For Test data - 94.6%

**Precision of the model:**
For Train data - 0.836
For Test data - 0.400

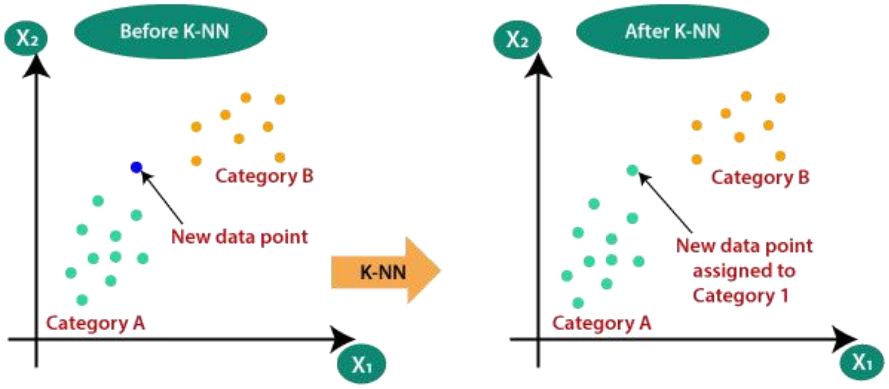
**Recall of the model:**
For Train data - 0.283
For Test data - 0.157

**F1 score of the model:**
For Train data - 0.423
For Test data - 0.225

**AUC score of the Model in Decision Tree Algorithm** - 0.69

## 6.4 IMPLEMENTATION OF K NEAREST NEIGHBOUR (KNN)

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.




Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the above diagram:

- Step-1: Select the number K of the neighbors
- Step-2: Calculate the Euclidean distance of K number of neighbors
- Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.
- Step-4: Among these k neighbors, count the number of the data points in each category.
- Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.
- Step-6: Our model is ready.

**Advantages of KNN Algorithm:**

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

**Disadvantages of KNN Algorithm:**

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

## 6.4.A MODELLING KNN WITH DEFAULT PARAMETERS

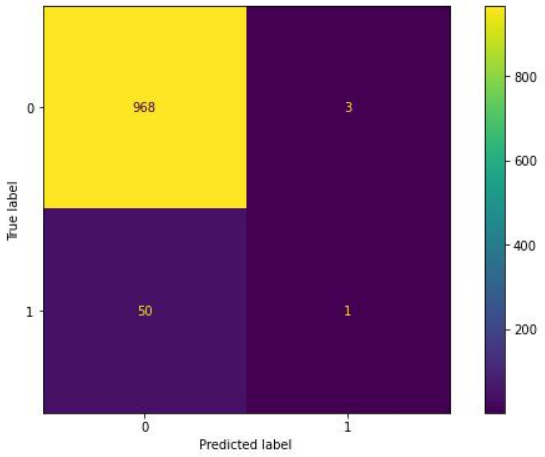
The model is build with default parameters by assigning the n_neighbors , metric and check its performance by accuracy,F1_score,recall_score,confusion_matrix,classification _report and roc_auc_score.

```
KNN_model = KNeighborsClassifier(n_neighbors=5,p=2,metric='euclidean')
KNN_model.fit(X_train_scaled, y_train)
y_pred_KNN = KNN_model.predict(X_test_scaled)
y_train_pred_KNN = KNN_model.predict(X_train_scaled)

knn_f1 = f1_score(y_test, y_pred_KNN)
knn_acc = accuracy_score(y_test, y_pred_KNN)
knn_recall = recall_score(y_test, y_pred_KNN)
knn_auc = roc_auc_score(y_test, y_pred_KNN)

print(confusion_matrix(y_test, y_pred_KNN))
print("\033[1m------------------------------------------------------------\033[0m")
print(classification_report(y_test, y_pred_KNN))
print("\033[1m------------------------------------------------------------\033[0m")

plot_confusion_matrix(KNN_model, X_test_scaled, y_test)

train_val(y_train, y_train_pred_KNN, y_test, y_pred_KNN)
```

```
[[968   3]
 [ 50   1]]
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       971
           1       0.25      0.02      0.04        51

    accuracy                           0.95      1022
   macro avg       0.60      0.51      0.50      1022
weighted avg       0.92      0.95      0.93      1022

------------------------------------------------------------
```

Accuracy and confusion matrix

| | train_set | test_set |
|---|---|---|
| Accuracy | 0.955 | 0.948 |
| Precision | 0.741 | 0.250 |
| Recall | 0.101 | 0.020 |
| f1 | 0.178 | 0.036 |



## 6.4.B CROSS-VALIDATING K-NEAREST NEIGHBOUR (KNN)

Cross-validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of the input data. We can also say that it is a technique to check how a statistical model generalizes to an independent dataset.

```
knn_xvalid_model = KNeighborsClassifier(n_neighbors=5)

knn_xvalid_model_scores = cross_validate(knn_xvalid_model, X_train_scaled, y_train,
                                         scoring = ["accuracy", "precision", "recall", "f1"], cv = 10)
knn_xvalid_model_scores = pd.DataFrame(knn_xvalid_model_scores, index = range(1, 11))

knn_xvalid_model_scores
```

| | fit_time | score_time | test_accuracy | test_precision | test_recall | test_f1 |
|---|---|---|---|---|---|---|
| 1 | 0.011 | 0.028 | 0.949 | 0.000 | 0.000 | 0.000 |
| 2 | 0.013 | 0.027 | 0.941 | 0.000 | 0.000 | 0.000 |
| 3 | 0.008 | 0.028 | 0.946 | 0.250 | 0.050 | 0.083 |
| 4 | 0.015 | 0.031 | 0.951 | 0.500 | 0.050 | 0.091 |
| 5 | 0.010 | 0.032 | 0.949 | 0.333 | 0.050 | 0.087 |
| 6 | 0.008 | 0.027 | 0.954 | 1.000 | 0.050 | 0.095 |
| 7 | 0.008 | 0.028 | 0.949 | 0.000 | 0.000 | 0.000 |
| 8 | 0.009 | 0.028 | 0.951 | 0.000 | 0.000 | 0.000 |
| 9 | 0.008 | 0.027 | 0.946 | 0.000 | 0.000 | 0.000 |
| 10 | 0.008 | 0.027 | 0.941 | 0.000 | 0.000 | 0.000 |

```
knn_xvalid_model_scores.mean()[2:]
```

```
test_accuracy    0.948
test_precision   0.208
test_recall      0.020
test_f1          0.036
dtype: float64
```

## 6.4.C ERROR RATE METHOD FOR CHOOSING REASONABLE K VALUES

Let's go ahead and use the error rate method to pick a good K Value. We will basically check the error rate for k=1 to say k=30. For every value of k we will call KNN classifier and then choose the value of k which has the least error rate.

```
test_error_rates = []


for k in range(1, 30):
    KNN_model = KNeighborsClassifier(n_neighbors=k)
    KNN_model.fit(X_train_scaled, y_train)

    y_pred_Er = KNN_model.predict(X_test_scaled)
    test_error_rates.append(np.mean(y_pred_Er!=y_test))
```

We are setting the k value at a range of 1 to 30, for each and every k value, the model is fitted and the output is predicted for each k value, where the error reduces becomes the best k value

**Plot for Error Rate Method**

```
plt.figure(figsize=(20,8))
plt.plot(range(1,30),test_error_rates,color='blue',linestyle='dashed',marker='o',markerfacecolor='red')
plt.title("Error Rate Method")
plt.xlabel('Kth values')
plt.ylabel('Error_rate')
```

```
Text(0, 0.5, 'Error_rate')
```

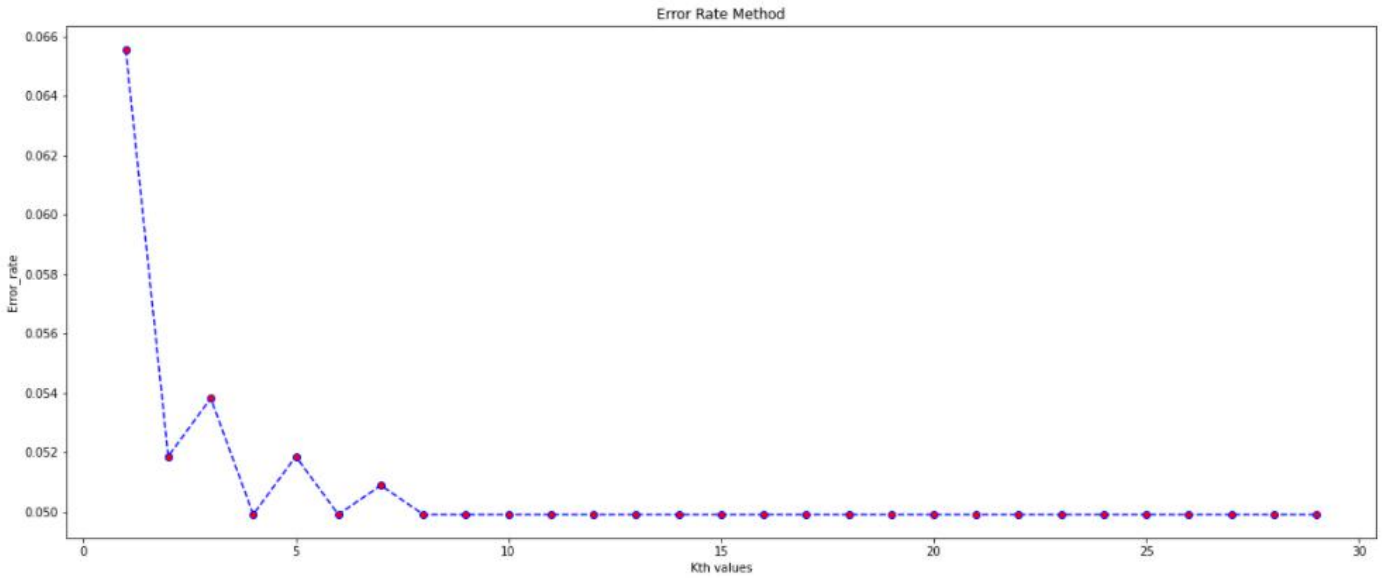



Fig 6.9 -Error Rate Graph for KNN

While checking with error rate method, from k=7 we get less error , lets cross check by using GridSearch CV for choosing reasonable k-values.

## 6.4.D GRID SEARCH CV FOR CHOOSING REASONABLE K VALUES

The model is build with Grid Search CV for best parameters that suits the reasonable k-values, in GridSearchCV by assigning the n_neighbors and weights and p value.

```
k_values= range(1, 30)
param_grid = {"n_neighbors": k_values, "p": [1, 2], "weights": ['uniform', "distance"]}
```

```
KNN_grid = KNeighborsClassifier()
KNN_grid_model = GridSearchCV(KNN_grid, param_grid, cv=10, scoring='accuracy')
KNN_grid_model.fit(X_train_scaled, y_train)
```

```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': range(1, 30), 'p': [1, 2],
                         'weights': ['uniform', 'distance']},
             scoring='accuracy')
```

**Let's look at the best parameters & estimator found by GridSearchCV.**

```
print(colored('\033[1mBest Parameters of GridSearchCV for KNN Model:\033[0m', 'blue'),
      colored(KNN_grid_model.best_params_, 'cyan'))
print("----------------------------------------------------------------------------------------------------------------------------------")
print(colored('\033[1mBest Estimator of GridSearchCV for KNN Model:\033[0m', 'blue'),
      colored(KNN_grid_model.best_estimator_, 'cyan'))
```

```
Best Parameters of GridSearchCV for KNN Model: {'n_neighbors': 9, 'p': 1, 'weights': 'distance'}
----------------------------------------------------------------------------------------------------------------------------------
Best Estimator of GridSearchCV for KNN Model: KNeighborsClassifier(n_neighbors=9, p=1, weights='distance')
```

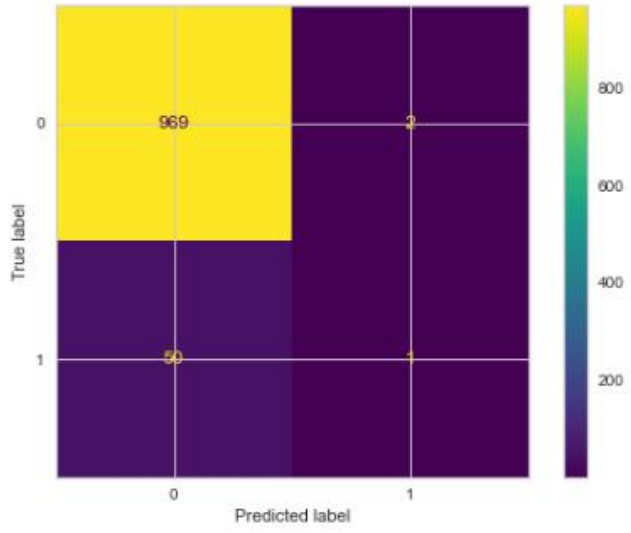
**Fit the model with Best parameters and check the performance:**

```
# NOW WITH K=7

KNN_model = KNeighborsClassifier(n_neighbors=9)
KNN_model.fit(X_train_scaled, y_train)
y_pred_ER = KNN_model.predict(X_test_scaled)
y_train_pred_ER = KNN_model.predict(X_train_scaled)

knn7_f1 = f1_score(y_test, y_pred_ER)
knn7_acc = accuracy_score(y_test, y_pred_ER)
knn7_recall = recall_score(y_test, y_pred_ER)
knn7_auc = roc_auc_score(y_test, y_pred_ER)

print('WITH K=7')
print('------------------')
print(confusion_matrix(y_test,y_pred_ER))
print("\033[1m------------------------------------------------------------\033[0m")
print(classification_report(y_test,y_pred_ER))
print("\033[1m------------------------------------------------------------\033[0m")

plot_confusion_matrix(KNN_model, X_test_scaled, y_test)

train_val(y_train, y_train_pred_ER, y_test, y_pred_ER)
```

```
WITH K=7
------------------
[[971   0]
 [ 51   0]]
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       971
           1       0.00      0.00      0.00        51

    accuracy                           0.95      1022
   macro avg       0.48      0.50      0.49      1022
weighted avg       0.90      0.95      0.93      1022

------------------------------------------------------------
```

Accuracy and Confusion Matrix

| | train_set | test_set |
|---|---|---|
| **Accuracy** | 0.955 | 0.949 |
| **Precision** | 0.826 | 0.333 |
| **Recall** | 0.096 | 0.020 |
| **f1** | 0.172 | 0.037 |



## 6.4.E ROC (Receiver Operating Curve) AND AUC (Area Under Curve).

The ROC curve stands for Receiver Operating Characteristic curve. ROC curves display the performance of a classification model.ROC tells us how good the model is for distinguishing between the given classes, in terms of the predicted probability.

The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.

```
plot_roc_curve(KNN_model, X_test_scaled, y_test);
```

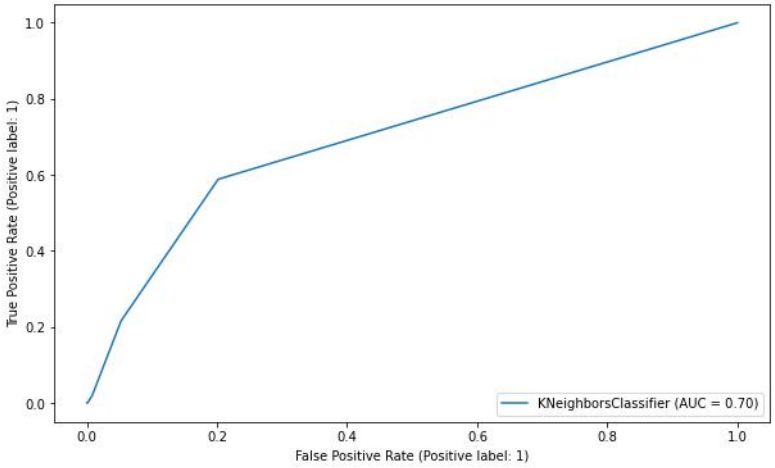



Fig 6.10 -ROC curve for KNN

```
plot_precision_recall_curve(KNN_model, X_test_scaled, y_test);
```

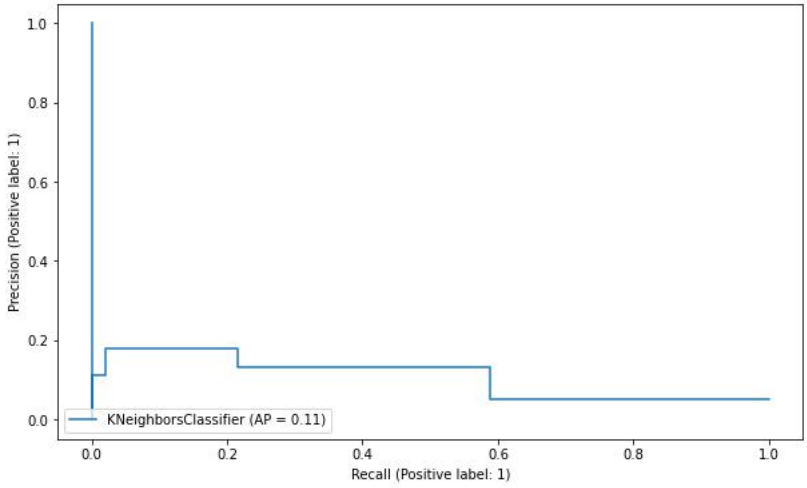



Fig 6.11 -Recall_Curve for KNN

By checking with the K-Nearest Neighbors Model

We get the output as,

**Accuracy of the model:**
For Train data - 95.5%
For Test data - 94.9%

**Precision of the model:**
For Train data - 0.826
For Test data - 0.333

**Recall of the model:**
For Train data - 0.096
For Test data - 0.020

**F1 score of the model:**
For Train data - 0.172
For Test data - 0.037

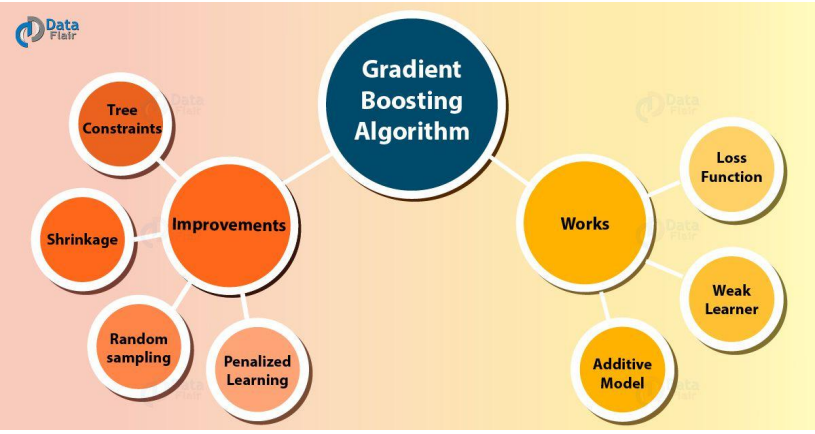
**AUC score of the Model in K-Nearest Neighbour Algorithm** - 0.70

## 6.5 IMPLEMENTATION OF GRADIENT BOOSTING (GB) ALGORITHM.

Gradient boosting machines are a family of powerful boosting machine learning algorithms with various practical applications that have demonstrated tremendous success in the form of accuracy.Which can be tailored to the application's unique needs since they learned about different loss functions in a better way.

The loss function is an indicator of the model's stability when the underlying data of the model is updated. A rational interpretation of the loss function is what we strive to minimize the error.Likewise, the loss function will calculate how strong our predictive trend is in detecting bad credit defaults.

Gradient boosting is a naive algorithm that can easily bypass a training data collection. The regulatory methods that penalize different parts of the algorithm will benefit from increasing the algorithm's efficiency by minimizing over fitness.In way it handles the model overfitting.



## 6.5.A MODELLING GRADIENT BOOSTING (GB) WITH DEFAULT PARAMETERS.

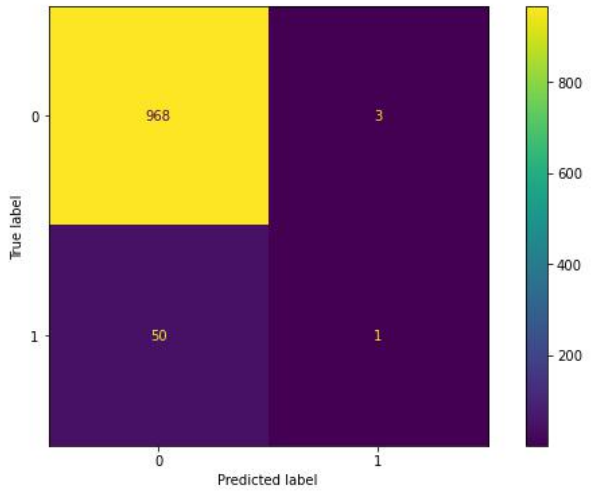
The model is build with default parameters by assigning the random state check its performance by accuracy,F1_score,recall_score,confusion_matrix,classification _report and roc_auc_score.

```
GB_model = GradientBoostingClassifier(random_state=42)
GB_model.fit(X_train_scaled, y_train)
y_pred = GB_model.predict(X_test_scaled)
y_train_pred = GB_model.predict(X_train_scaled)

gb_f1 = f1_score(y_test, y_pred)
gb_acc = accuracy_score(y_test, y_pred)
gb_recall = recall_score(y_test, y_pred)
gb_auc = roc_auc_score(y_test, y_pred)

print(confusion_matrix(y_test, y_pred))
print("\033[1m-------------------------------------------------------\033[0m")
print(classification_report(y_test, y_pred))
print("\033[1m-------------------------------------------------------\033[0m")

plot_confusion_matrix(GB_model, X_test_scaled, y_test)

train_val(y_train, y_train_pred, y_test, y_pred)
```

```
<IPython.core.display.Javascript object>

[[968   3]
 [ 50   1]]
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       971
           1       0.25      0.02      0.04        51

    accuracy                           0.95      1022
   macro avg       0.60      0.51      0.50      1022
weighted avg       0.92      0.95      0.93      1022

-------------------------------------------------------
```

Accuracy and confusion matrix

| | train_set | test_set |
|---|---|---|
| Accuracy | 0.961 | 0.948 |
| Precision | 0.951 | 0.250 |
| Recall | 0.197 | 0.020 |
| f1 | 0.326 | 0.036 |



## 6.5.B CROSS-VALIDATING GRADIENT BOOSTING (GB) ALGORITHM

Cross-validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of the input data. We can also say that it is a technique to check how a statistical model generalizes to an independent dataset.

```
gb_xvalid_model = GradientBoostingClassifier(random_state=42)

gb_xvalid_model_scores = cross_validate(gb_xvalid_model, X_train_scaled, y_train,
                                        scoring = ["accuracy", "precision_macro", "recall_macro", "f1_macro"], cv = 10)
gb_xvalid_model_scores = pd.DataFrame(gb_xvalid_model_scores, index = range(1, 11))

gb_xvalid_model_scores
```

```
<IPython.core.display.Javascript object>
```

| | fit_time | score_time | test_accuracy | test_precision_macro | test_recall_macro | test_f1_macro |
|---|---|---|---|---|---|---|
| 1 | 0.443 | 0.003 | 0.954 | 0.977 | 0.525 | 0.536 |
| 2 | 0.441 | 0.005 | 0.956 | 0.978 | 0.550 | 0.580 |
| 3 | 0.460 | 0.004 | 0.956 | 0.978 | 0.550 | 0.580 |
| 4 | 0.459 | 0.004 | 0.949 | 0.475 | 0.499 | 0.487 |
| 5 | 0.451 | 0.004 | 0.956 | 0.978 | 0.550 | 0.580 |
| 6 | 0.451 | 0.004 | 0.949 | 0.475 | 0.499 | 0.487 |
| 7 | 0.450 | 0.003 | 0.949 | 0.643 | 0.522 | 0.530 |
| 8 | 0.499 | 0.004 | 0.949 | 0.477 | 0.497 | 0.487 |
| 9 | 0.498 | 0.004 | 0.953 | 0.477 | 0.500 | 0.488 |
| 10 | 0.491 | 0.004 | 0.944 | 0.475 | 0.496 | 0.485 |

```
gb_xvalid_model_scores.mean()
```

```
fit_time              0.464
score_time            0.004
test_accuracy         0.951
test_precision_macro  0.693
test_recall_macro     0.519
test_f1_macro         0.524
dtype: float64
```

## 6.5.C MODELLING GRADIENT BOOSTING (GB) WITH BEST PARAMETERS USING GRIDSEARCHCV

A machine learning algorithm requires certain hyperparameters that must be tuned before training. An optimal subset of these hyperparameters must be selected, which is called hyperparameter optimization. Grid-Search is a sci-kit learn package that provides for hyperparameter tuning. A grid search space is generated by taking the initial set of values given to each hyperparameter. Each cell in the grid is searched for the optimal solution.

**Computing the accuracy scores on train and validation sets when training with different learning rates**

```
learning_rates = [0.05, 0.1, 0.15, 0.25, 0.5, 0.6, 0.75, 0.85, 1]

for learning_rate in learning_rates:
    gb = GradientBoostingClassifier(n_estimators=20, learning_rate = learning_rate, random_state=42)
    gb.fit(X_train, y_train)
    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb.score(X_train, y_train)))
    print("Accuracy score (test): {0:.3f}".format(gb.score(X_test, y_test)))
    print()
```

```
<IPython.core.display.Javascript object>

Learning rate:  0.05
Accuracy score (training): 0.952
Accuracy score (test): 0.950


<IPython.core.display.Javascript object>

Learning rate:  0.1
Accuracy score (training): 0.953
Accuracy score (test): 0.951


<IPython.core.display.Javascript object>

Learning rate:  0.15
Accuracy score (training): 0.954
Accuracy score (test): 0.950


<IPython.core.display.Javascript object>

Learning rate:  0.25
Accuracy score (training): 0.957
Accuracy score (test): 0.950


<IPython.core.display.Javascript object>

Learning rate:  0.5
Accuracy score (training): 0.962
Accuracy score (test): 0.948


<IPython.core.display.Javascript object>

Learning rate:  0.6
Accuracy score (training): 0.962
Accuracy score (test): 0.940


<IPython.core.display.Javascript object>

Learning rate:  0.75
Accuracy score (training): 0.964
Accuracy score (test): 0.942


<IPython.core.display.Javascript object>

Learning rate:  0.85
Accuracy score (training): 0.956
Accuracy score (test): 0.942


<IPython.core.display.Javascript object>

Learning rate:  1
Accuracy score (training): 0.962
Accuracy score (test): 0.942
```

```
param_grid = {"n_estimators":[100, 200, 300],
              "subsample":[0.5, 1], "max_features" : [None, 2, 3, 4],
               "learning_rate": [0.2, 0.5, 0.6, 0.75, 0.85, 1.0, 1.25, 1.5]}
```

```
GB_grid_model = GradientBoostingClassifier(random_state=42)

GB_grid_model = GridSearchCV(GB_grid_model, param_grid, scoring = "f1", verbose=2, n_jobs = -1).fit(X_train, y_train)
```

```
<IPython.core.display.Javascript object>

Fitting 5 folds for each of 192 candidates, totalling 960 fits
```

**Let's look at the best parameters & estimator found by GridSearchCV.**

```
print(colored('\033[1mBest Parameters of GridSearchCV for Gradient Boosting Model:\033[0m', 'blue'),
      colored(GB_grid_model.best_params_, 'cyan'))
print("-----------------------------------------------------------------------------------------------------------------------------")
print(colored('\033[1mBest Estimator of GridSearchCV for Gradient Boosting Model:\033[0m', 'blue'),
      colored(GB_grid_model.best_estimator_, 'cyan'))
```

```
Best Parameters of GridSearchCV for Gradient Boosting Model: {'learning_rate': 1.5, 'max_features': None, 'n_estimators': 300,
'subsample': 1}
-----------------------------------------------------------------------------------------------------------------------------
Best Estimator of GridSearchCV for Gradient Boosting Model: GradientBoostingClassifier(learning_rate=1.5, n_estimators=300, ran
dom_state=42,
                           subsample=1)
```

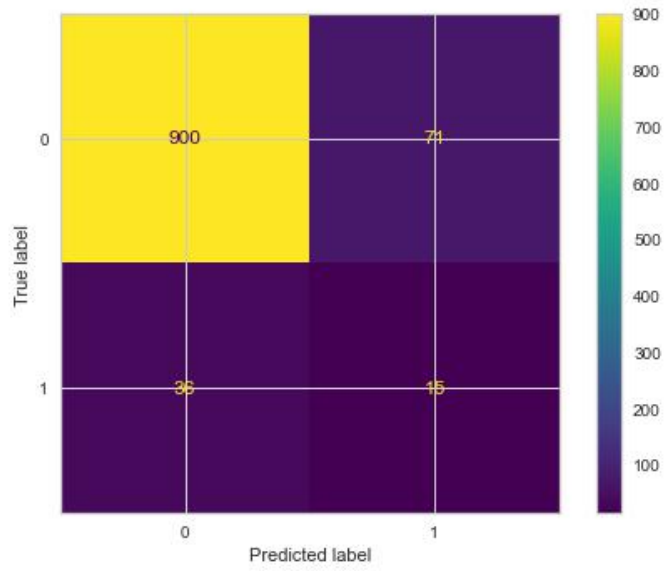
**Fit the model with Best parameters and check the performance:**

```
y_pred = GB_grid_model.predict(X_test_scaled)
y_train_pred = GB_grid_model.predict(X_train_scaled)

gb_grid_f1 = f1_score(y_test, y_pred)
gb_grid_acc = accuracy_score(y_test, y_pred)
gb_grid_recall = recall_score(y_test, y_pred)
gb_grid_auc = roc_auc_score(y_test, y_pred)

print(confusion_matrix(y_test, y_pred))
print("\033[1m-------------------------------------------------------\033[0m")
print(classification_report(y_test, y_pred))
print("\033[1m-------------------------------------------------------\033[0m")

plot_confusion_matrix(GB_grid_model, X_test_scaled, y_test)

train_val(y_train, y_train_pred, y_test, y_pred)
```

```
[[900  71]
 [ 36  15]]
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.96      0.93      0.94       971
           1       0.17      0.29      0.22        51

    accuracy                           0.90      1022
   macro avg       0.57      0.61      0.58      1022
weighted avg       0.92      0.90      0.91      1022

-------------------------------------------------------
```

Accuracy and confusion matrix:

| | train_set | test_set |
|---|---|---|
| **Accuracy** | 0.893 | 0.895 |
| **Precision** | 0.123 | 0.174 |
| **Recall** | 0.197 | 0.294 |
| **f1** | 0.151 | 0.219 |



## 6.5.D FEATURE IMPORTANCE FOR GRADIENT BOOSTING (GB)

The feature importance (variable importance) describes which features are relevant. It can help with better understanding of the solved problem and sometimes lead to model improvements by employing the feature selection.

```
GB_model.feature_importances_
```

```
array([0.20001838, 0.00384661, 0.35317492, 0.02595992, 0.02873132,
       0.00608496, 0.06448584, 0.00375002, 0.12379522, 0.17451035,
       0.01564246])
```

```
GB_feature_imp = pd.DataFrame(index = X.columns, data = GB_model.feature_importances_,
                              columns = ["Feature Importance"]).sort_values("Feature Importance", ascending = False)
GB_feature_imp
```

| | Feature Importance |
|---|---|
| age | 0.353 |
| id | 0.200 |
| bmi | 0.175 |
| avg_glucose_level | 0.124 |
| work_type | 0.064 |
| heart_disease | 0.029 |
| hypertension | 0.026 |
| smoking_status | 0.016 |
| ever_married | 0.006 |
| gender | 0.004 |
| Residence_type | 0.004 |

### Visualizing Feature Importance in Gradient Boosting Algorithm:

```
sns.barplot(y=GB_feature_imp["Feature Importance"], x=GB_feature_imp.index)
plt.title("Feature Importance")
plt.xticks(rotation=45)
plt.show()
```

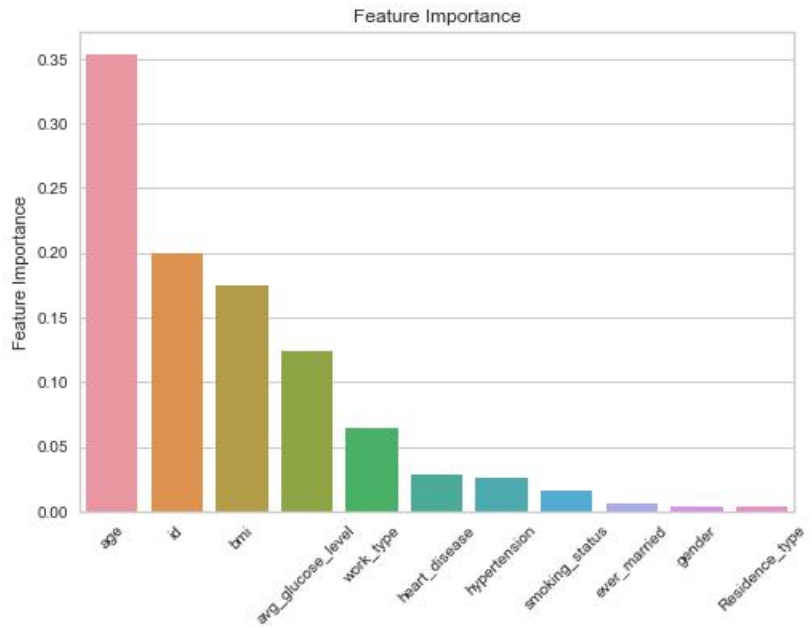



Fig 6.12 -Bar-plot for Feature Importance in GB

## 6.5.E ROC (Receiver Operating Curve) AND AUC (Area Under Curve).

The ROC curve stands for Receiver Operating Characteristic curve. ROC curves display the performance of a classification model.ROC tells us how good the model is for distinguishing between the given classes, in terms of the predicted probability.

The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.

```
plot_roc_curve(GB_model, X_test, y_test);
```

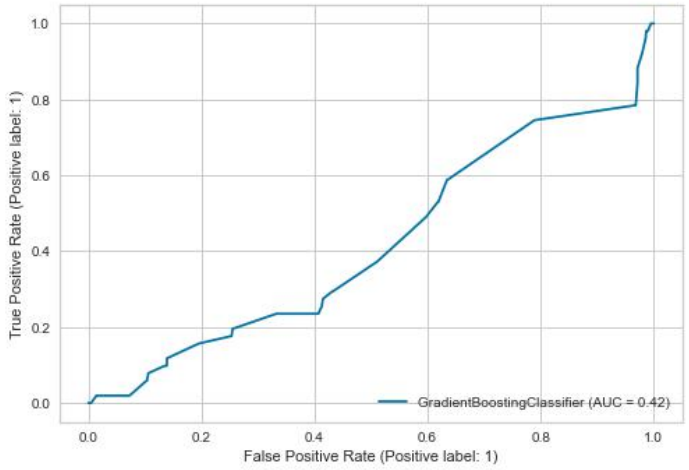



Fig 6.13 -ROC curve for GB

```
plot_precision_recall_curve(GB_model, X_test, y_test);
```

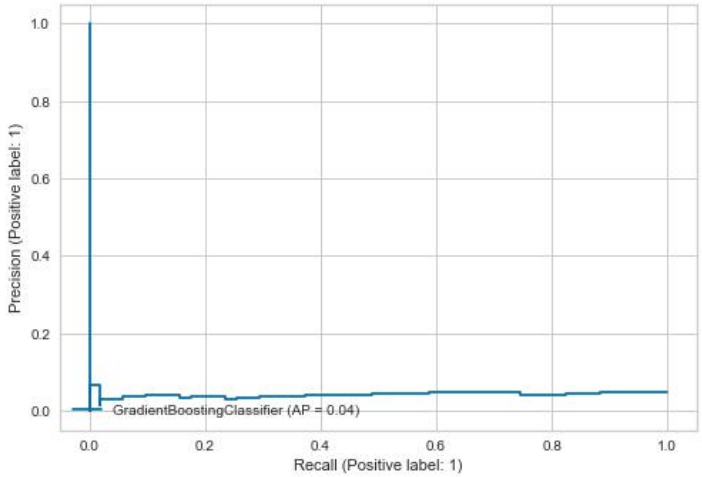



Fig 6.14 -Recall_curve for GB

By checking with the Gradient Boosting Model

We get the output as,

**Accuracy of the model:**
For Train data - 89.3%
For Test data - 89.5%

**Precision of the model:**
For Train data - 0.123
For Test data - 0.174

**Recall of the model:**
For Train data - 0.197
For Test data - 0.294

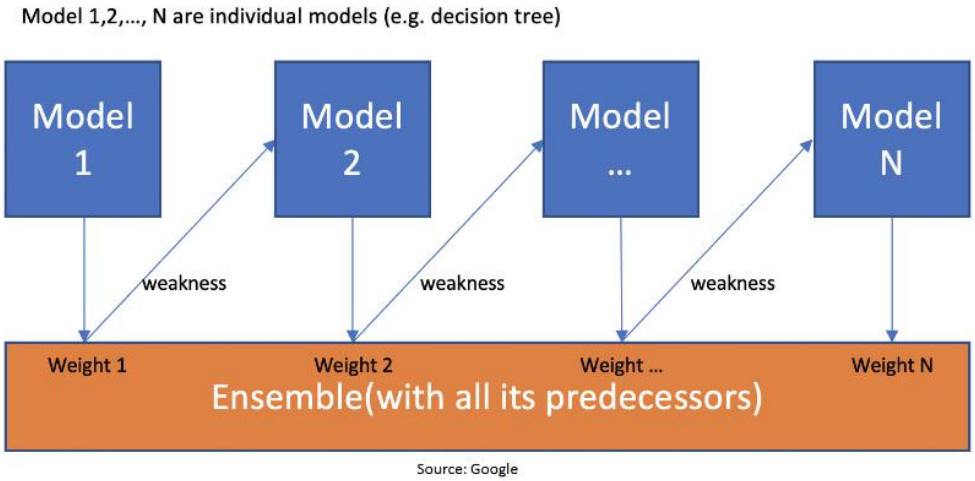
**F1 score of the model:**
For Train data - 0.151
For Test data - 0.219

**AUC score of the Model in Gradient Boosting Algorithm** - 0.42

## 6.6 IMPLEMENTATION OF ADABOOST (AB) ALGORITHM

AdaBoost algorithm, short for Adaptive Boosting, is a Boosting technique used as an Ensemble Method in Machine Learning. It is called Adaptive Boosting as the weights are re-assigned to each instance, with higher weights assigned to incorrectly classified instances. Boosting is used to reduce bias as well as variance for supervised learning. It works on the principle of learners growing sequentially. Except for the first, each subsequent learner is grown from previously grown learners. In simple words, weak learners are converted into strong ones. The AdaBoost algorithm works on the same principle as boosting with a slight difference.

It makes 'n' number of decision trees during the data training period. As the first decision tree/model is made, the incorrectly classified record in the first model is given priority. Only these records are sent as input for the second model. The process goes on until we specify a number of base learners we want to create. Remember, repetition of records is allowed with all boosting techniques.



Source: Google

**WORKING STEPS:**

Step 1 – Creating the First Base Learner

Step 2 – Calculating the Total Error (TE)

Step 3 – Calculating Performance of the Stump

Step 4 – Updating Weights

Step 5 – Creating a New Dataset

## 6.6.A MODELLINGADABOOST (AB) WITH DEFAULT PARAMETERS

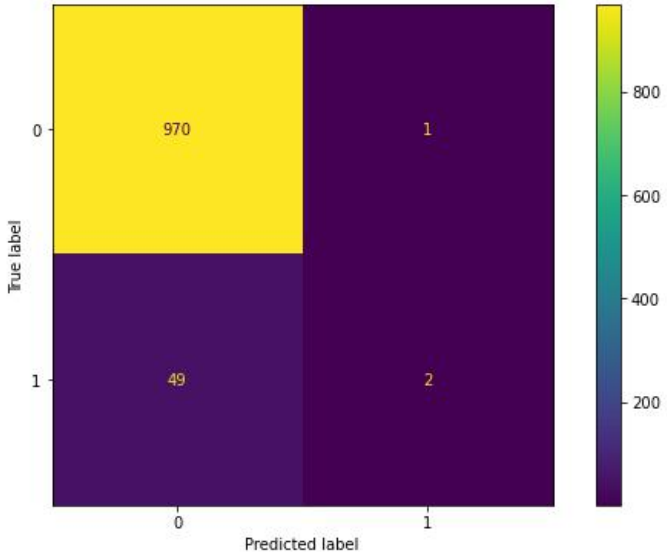
The model is build with default parameters by assigning the n_estimators ,random state and check its performance by accuracy,F1_score,recall_score,confusion_matrix,classification _report and roc_auc_score.

```
AB_model = AdaBoostClassifier(n_estimators=50, random_state=101)
AB_model.fit(X_train, y_train)
y_pred = AB_model.predict(X_test)
y_train_pred = AB_model.predict(X_train)

ab_f1 = f1_score(y_test, y_pred)
ab_acc = accuracy_score(y_test, y_pred)
ab_recall = recall_score(y_test, y_pred)
ab_auc = roc_auc_score(y_test, y_pred)

print(confusion_matrix(y_test, y_pred))
print("\033[1m-------------------------------------------------------\033[0m")
print(classification_report(y_test, y_pred))
print("\033[1m-------------------------------------------------------\033[0m")

plot_confusion_matrix(AB_model, X_test, y_test)

train_val(y_train, y_train_pred, y_test, y_pred)
```

```
[[970   1]
 [ 49   2]]
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       971
           1       0.67      0.04      0.07        51

    accuracy                           0.95      1022
   macro avg       0.81      0.52      0.52      1022
weighted avg       0.94      0.95      0.93      1022

-------------------------------------------------------
```

Accuracy and Confusion matrix

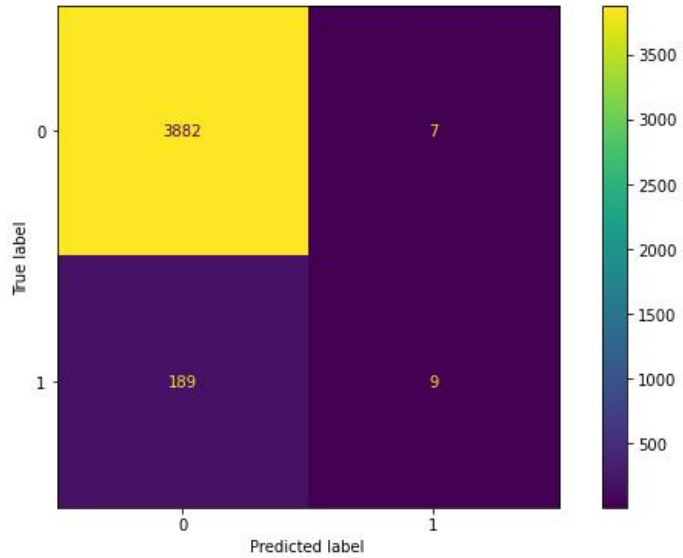
| | train_set | test_set |
|---|---|---|
| Accuracy | 0.952 | 0.951 |
| Precision | 0.562 | 0.667 |
| Recall | 0.045 | 0.039 |
| f1 | 0.084 | 0.074 |




**Cross-checking the model by predictions in Train Set for consistency**

```
y_train_pred = AB_model.predict(X_train)

print(confusion_matrix(y_train, y_train_pred))
print("\033[1m-------------------------------------------------------\033[0m")
print(classification_report(y_train, y_train_pred))
print("\033[1m-------------------------------------------------------\033[0m")

plot_confusion_matrix(AB_model, X_train, y_train);
```

```
[[3882    7]
 [ 189    9]]
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      1.00      0.98      3889
           1       0.56      0.05      0.08       198

    accuracy                           0.95      4087
   macro avg       0.76      0.52      0.53      4087
weighted avg       0.93      0.95      0.93      4087

-------------------------------------------------------
```



## 6.6.B CROSS-VALIDATING ADABOOST (AB) ALGORITHM

Cross-validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of the input data. We can also say that it is a technique to check how a statistical model generalizes to an independent dataset.

```
ab_xvalid_model = AdaBoostClassifier(n_estimators=50, random_state=101)

ab_xvalid_model_scores = cross_validate(ab_xvalid_model, X_train, y_train,
                                        scoring = ['accuracy', 'precision','recall', 'f1'], cv = 10)
ab_xvalid_model_scores = pd.DataFrame(ab_xvalid_model_scores, index = range(1, 11))

ab_xvalid_model_scores
```

| | fit_time | score_time | test_accuracy | test_precision | test_recall | test_f1 |
|---|---|---|---|---|---|---|
| 1 | 0.200 | 0.014 | 0.951 | 0.500 | 0.100 | 0.167 |
| 2 | 0.205 | 0.014 | 0.951 | 0.500 | 0.050 | 0.091 |
| 3 | 0.194 | 0.019 | 0.949 | 0.000 | 0.000 | 0.000 |
| 4 | 0.192 | 0.014 | 0.951 | 0.000 | 0.000 | 0.000 |
| 5 | 0.177 | 0.013 | 0.956 | 1.000 | 0.100 | 0.182 |
| 6 | 0.206 | 0.014 | 0.949 | 0.000 | 0.000 | 0.000 |
| 7 | 0.187 | 0.013 | 0.954 | 0.667 | 0.100 | 0.174 |
| 8 | 0.176 | 0.014 | 0.944 | 0.000 | 0.000 | 0.000 |
| 9 | 0.186 | 0.013 | 0.953 | 0.500 | 0.053 | 0.095 |
| 10 | 0.183 | 0.014 | 0.946 | 0.000 | 0.000 | 0.000 |

```
ab_xvalid_model_scores.mean()
```

```
fit_time          0.191
score_time        0.014
test_accuracy     0.950
test_precision    0.317
test_recall       0.040
test_f1           0.071
dtype: float64
```

## 6.6.C VISUALIZATION OF TREE

The Tree is visualized by using plot_tree library of AdaBoost Classifier

```
AB_model = AdaBoostClassifier(n_estimators=3, random_state=42)
AB_model.fit(X_train, y_train)
```

```
AdaBoostClassifier(n_estimators=3, random_state=42)
```

```
features = list(X.columns)
targets = df["stroke"].astype("str")

plt.figure(figsize=(12, 8),dpi=150)
plot_tree(AB_model.estimators_[0], filled=True, feature_names=features, class_names=targets.unique(), proportion=True);
```

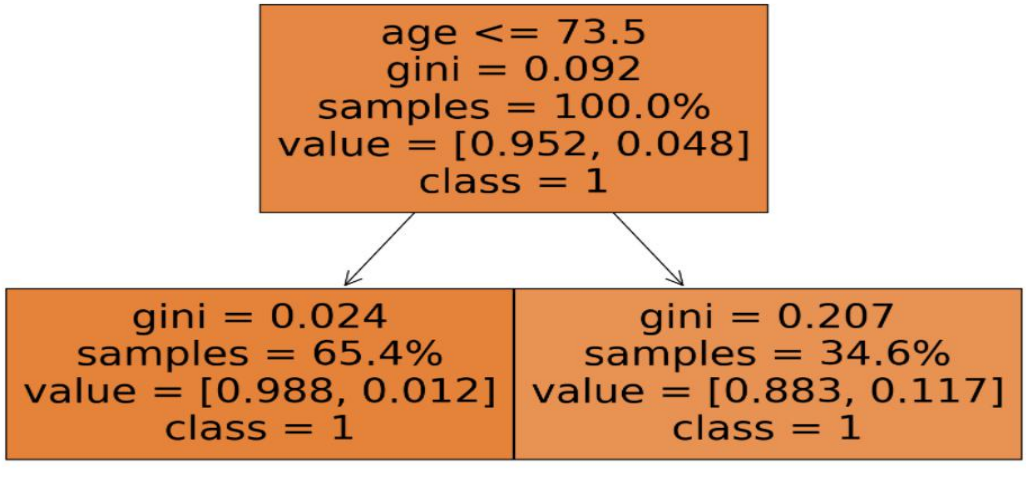



Fig 6.15 -Visualization of Tree in Adaboost

## 6.6.D ANALYZING PERFORMANCE WHILE WEAK LEARNERS ARE ADDED

The Range is set between 1 to 100 and check its performance by adding the weak learners and visualized the error rate graph.

```
error_rates = []

for n in range(1, 100):

    AB_model = AdaBoostClassifier(n_estimators=n)
    AB_model.fit(X_train, y_train)
    preds = AB_model.predict(X_test)
    err = 1 - f1_score(y_test, preds)

    error_rates.append(err)
```

```
plt.figure(figsize=(14, 8))
plt.plot(range(1, 100), error_rates);
```

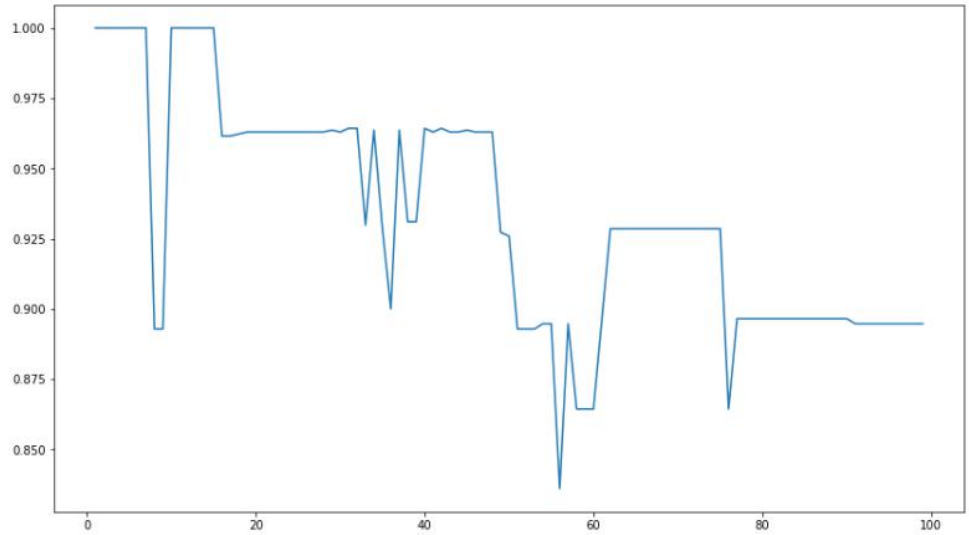



Fig 6.16 -Error rate plot for Adaboost

## 6.6.E FEATURE IMPORTANCE OF ADABOOST(AB)

The feature importance (variable importance) describes which features are relevant. It can help with better understanding of the solved problem and sometimes lead to model improvements by employing the feature selection.

```
AB_model.feature_importances_
```

```
array([0.23232323, 0.01010101, 0.26262626, 0.01010101, 0.        ,
       0.        , 0.01010101, 0.        , 0.31313131, 0.14141414,
       0.02020202])
```

```
AB_feature_imp = pd.DataFrame(index = X.columns, data = AB_model.feature_importances_,
                              columns = ["Feature Importance"]).sort_values("Feature Importance", ascending = False)
AB_feature_imp
```

| | Feature Importance |
|---|---|
| avg_glucose_level | 0.313 |
| age | 0.263 |
| id | 0.232 |
| bmi | 0.141 |
| smoking_status | 0.020 |
| gender | 0.010 |
| hypertension | 0.010 |
| work_type | 0.010 |
| heart_disease | 0.000 |
| ever_married | 0.000 |
| residence_type | 0.000 |

**Visualizing Feature Importance in Adaboost Algorithm:**

```
imp_feats = AB_feature_imp.sort_values("Feature Importance")
```

```
plt.figure(figsize=(12,6))

sns.barplot(y=AB_feature_imp["Feature Importance"], x=AB_feature_imp.index)

plt.title("Feature Importance")
plt.xticks(rotation=45)
plt.show()
```

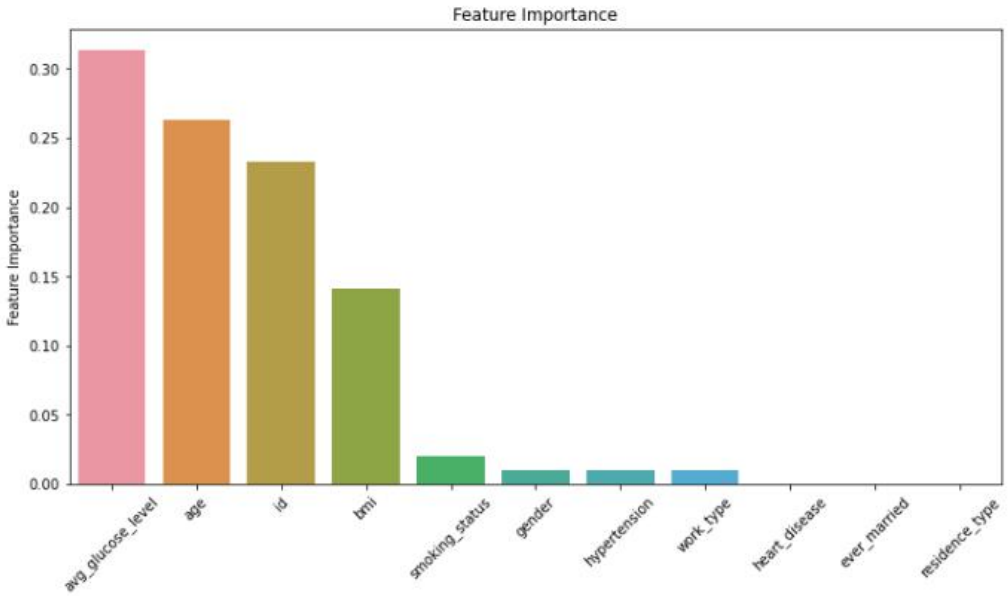



Fig 6.17 -Bar-plot for Feature Importance in Adaboost

## 6.6.F MODELLING ADABOOST(AB) WITH BEST PARAMETERS USING GRID SEARCH CV

A machine learning algorithm requires certain hyperparameters that must be tuned before training. An optimal subset of these hyperparameters must be selected, which is called hyperparameter optimization. Grid-Search is a sci-kit learn package that provides for hyperparameter tuning. A grid search space is generated by taking the initial set of values given to each hyperparameter. Each cell in the grid is searched for the optimal solution.

Computing the accuracy scores on train and validation sets when training with different learning rates

```
learning_rates = [0.05, 0.1, 0.15, 0.25, 0.5, 0.6, 0.75, 0.85, 1]

for learning_rate in learning_rates:
    ab = AdaBoostClassifier(n_estimators=20, learning_rate = learning_rate, random_state=42)
    ab.fit(X_train, y_train)
    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(ab.score(X_train, y_train)))
    print("Accuracy score (test): {0:.3f}".format(ab.score(X_test, y_test)))
    print()
```

```
Learning rate:  0.05
Accuracy score (training): 0.952
Accuracy score (test): 0.950

Learning rate:  0.1
Accuracy score (training): 0.952
Accuracy score (test): 0.950

Learning rate:  0.15
Accuracy score (training): 0.952
Accuracy score (test): 0.950
```

```
Learning rate:  0.25
Accuracy score (training): 0.952
Accuracy score (test): 0.950

Learning rate:  0.5
Accuracy score (training): 0.952
Accuracy score (test): 0.950

Learning rate:  0.6
Accuracy score (training): 0.951
Accuracy score (test): 0.950

Learning rate:  0.75
Accuracy score (training): 0.951
Accuracy score (test): 0.950

Learning rate:  0.85
Accuracy score (training): 0.952
Accuracy score (test): 0.950

Learning rate:  1
Accuracy score (training): 0.951
Accuracy score (test): 0.949
```

```
param_grid = {"n_estimators": [15, 20, 100, 500], "learning_rate": [0.2, 0.5, 0.6, 0.75, 0.85, 1.0, 1.25, 1.5]}
```

```
AB_grid_model = AdaBoostClassifier(random_state=42)
AB_grid_model = GridSearchCV(AB_grid_model, param_grid, cv=5, scoring= 'f1')
```

```
AB_grid_model.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=AdaBoostClassifier(random_state=42),
             param_grid={'learning_rate': [0.2, 0.5, 0.6, 0.75, 0.85, 1.0, 1.25,
                                           1.5],
                         'n_estimators': [15, 20, 100, 500]},
             scoring='f1')
```

**Let's look at the best parameters & estimator found by GridSearchCV.**

```
print(colored('\033[1mBest Parameters of GridSearchCV for AdaBoosting Model:\033[0m', 'blue'), colored(AB_grid_model.best_params_
print("----------------------------------------------------------------------------------------------------------------------------")
print(colored('\033[1mBest Estimator of GridSearchCV for AdaBoosting Model:\033[0m', 'blue'), colored(AB_grid_model.best_estimato
```

```
Best Parameters of GridSearchCV for AdaBoosting Model: {'learning_rate': 1.5, 'n_estimators': 100}
----------
Best Estimator of GridSearchCV for AdaBoosting Model: AdaBoostClassifier(learning_rate=1.5, n_estimators=100, random_state=42)
```

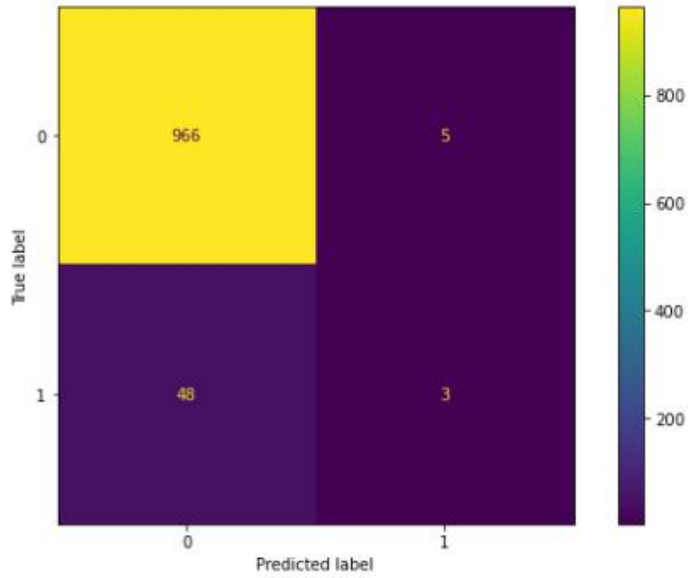
## Fit the model with Best parameters and check the performance:

```
y_pred = AB_grid_model.predict(X_test)
y_train_pred = AB_grid_model.predict(X_train)

ab_grid_f1 = f1_score(y_test, y_pred)
ab_grid_acc = accuracy_score(y_test, y_pred)
ab_grid_recall = recall_score(y_test, y_pred)
ab_grid_auc = roc_auc_score(y_test, y_pred)

print(confusion_matrix(y_test, y_pred))
print("\033[1m-------------------------------------------------------\033[0m")
print(classification_report(y_test, y_pred))
print("\033[1m-------------------------------------------------------\033[0m")

plot_confusion_matrix(AB_grid_model, X_test, y_test)

train_val(y_train, y_train_pred, y_test, y_pred)
```

```
[[966   5]
 [ 48   3]]
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       971
           1       0.38      0.06      0.10        51

    accuracy                           0.95      1022
   macro avg       0.66      0.53      0.54      1022
weighted avg       0.92      0.95      0.93      1022

-------------------------------------------------------
```

Accuracy and confusion matrix

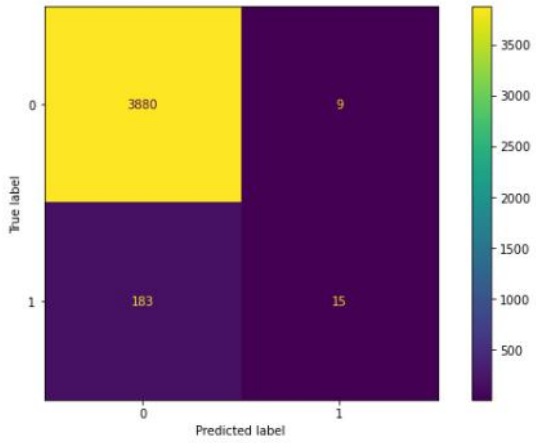
| | train_set | test_set |
|---|---|---|
| Accuracy | 0.950 | 0.948 |
| Precision | 0.444 | 0.375 |
| Recall | 0.101 | 0.059 |
| f1 | 0.165 | 0.102 |




Cross-checking the model by predictions in Train Set for consistency

```
y_train_pred = AB_grid_model.predict(X_train)

print(confusion_matrix(y_train, y_train_pred))
print("\033[1m-------------------------------------------------------\033[0m")
print(classification_report(y_train, y_train_pred))
print("\033[1m-------------------------------------------------------\033[0m")

plot_confusion_matrix(AB_model, X_train, y_train);
```

```
[[3864   25]
 [ 178   20]]
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.96      0.99      0.97      3889
           1       0.44      0.10      0.16       198

    accuracy                           0.95      4087
   macro avg       0.70      0.55      0.57      4087
weighted avg       0.93      0.95      0.94      4087

-------------------------------------------------------
```



## 6.6.G ROC (Receiver Operating Curve) AND AUC (Area Under Curve)

The ROC curve stands for Receiver Operating Characteristic curve. ROC curves display the performance of a classification model.ROC tells us how good the model is for distinguishing between the given classes, in terms of the predicted probability.

The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.

```
plot_roc_curve(AB_grid_model, X_test, y_test);
```

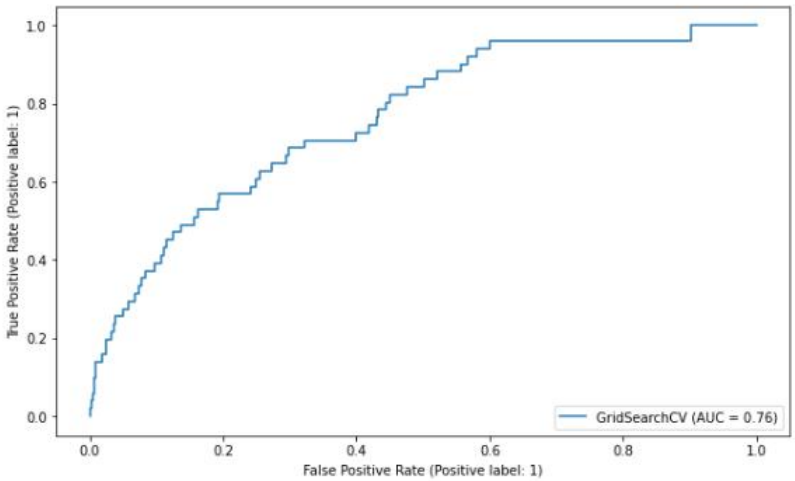



Fig 6.18 -ROC curve for AdaBoost

```
plot_precision_recall_curve(AB_grid_model, X_test, y_test);
```

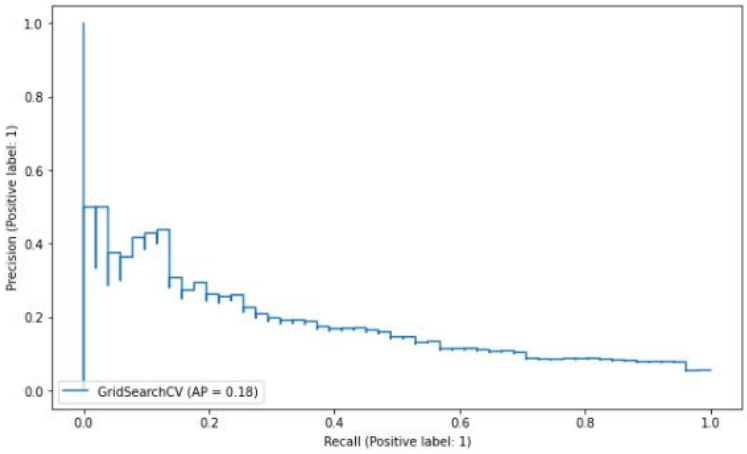



Fig 6.19 -Recall_curve for Adaboost

By checking with the Adaboost Model

We get the output as,

**Accuracy of the model:**
For Train data - 95.0%
For Test data - 94.8%

**Precision of the model:**
For Train data - 0.444
For Test data - 0.375

**Recall of the model:**
For Train data - 0.101
For Test data - 0.059

**F1 score of the model:**
For Train data - 0.165
For Test data - 0.102

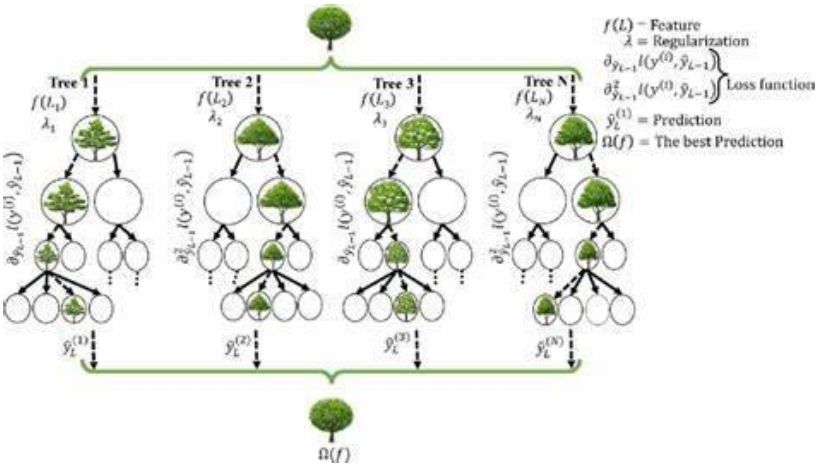
**AUC score of the Model in AdaBoost Algorithm** - 0.76

## 6.7 IMPLEMENTATION OF XGBOOST (XGB) ALGORITHM

XGBoost stands for eXtreme Gradient Boosting. It became popular in the recent days and is dominating applied machine learning and Kaggle competitions for structured data because of its scalability.XGBoost is an extension to gradient boosted decision trees (GBM) and specially designed to improve speed and performance.

- Execution Speed: XGBoost was almost always faster than the other benchmarked implementations from R, Python Spark and H2O and it is really faster when compared to the other algorithms.

- Model Performance: XGBoost dominates structured or tabular datasets on classification and regression predictive modelling problems.




XGBoost is a faster algorithm when compared to other algorithms because of its parallel and distributed computing. XGBoost is developed with both deep considerations in terms of systems optimization and principles in machine learning. The goal of this library is to push the extreme of the computation limits of machines to provide a scalable, portable and accurate library.

## 6.7.A MODELLING WITH XGBOOST (XGB) DEFAULT PARAMETERS.

The model is build with default parameters by assigning the random state and check its performance by accuracy,F1_score,recall_score,confusion_matrix,classification _report and roc_auc_score.

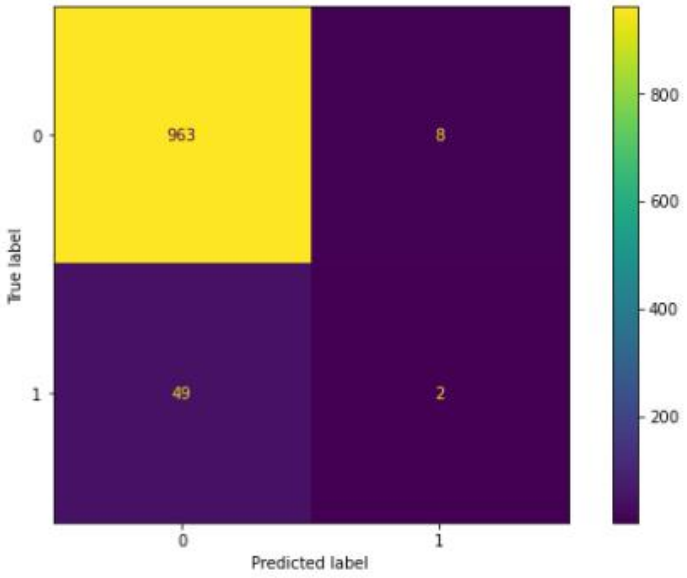
```
XGB_model = XGBClassifier(random_state=101)
XGB_model.fit(X_train_scaled, y_train)
y_pred = XGB_model.predict(X_test_scaled)
y_train_pred = XGB_model.predict(X_train_scaled)

xgb_f1 = f1_score(y_test, y_pred)
xgb_acc = accuracy_score(y_test, y_pred)
xgb_recall = recall_score(y_test, y_pred)
xgb_auc = roc_auc_score(y_test, y_pred)

print(confusion_matrix(y_test, y_pred))
print("\033[1m-------------------------------------------------------\033[0m")
print(classification_report(y_test, y_pred))
print("\033[1m-------------------------------------------------------\033[0m")

plot_confusion_matrix(XGB_model, X_test_scaled, y_test)

train_val(y_train, y_train_pred, y_test, y_pred)
```

```
[20:56:50] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/learner.cc:1115: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
[[963   8]
 [ 49   2]]
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       971
           1       0.20      0.04      0.07        51

    accuracy                           0.94      1022
   macro avg       0.58      0.52      0.52      1022
weighted avg       0.91      0.94      0.93      1022

-------------------------------------------------------
```

Accuracy and Confusion Matrix:

:

| | train_set | test_set |
|---|---|---|
| Accuracy | 1.000 | 0.944 |
| Precision | 1.000 | 0.200 |
| Recall | 1.000 | 0.039 |
| f1 | 1.000 | 0.066 |



## 6.7.B CROSS-VALIDATING XGBOOST (XGB) ALGORITHM

Cross-validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of the input data. We can also say that it is a technique to check how a statistical model generalizes to an independent dataset.

```
xgb_xvalid_model = XGBClassifier(random_state=101)

xgb_xvalid_model_scores = cross_validate(xgb_xvalid_model, X_train_scaled, y_train,
                                         scoring = ["accuracy", "precision", "recall", "f1"], cv = 10)
xgb_xvalid_model_scores = pd.DataFrame(xgb_xvalid_model_scores, index = range(1, 11))

xgb_xvalid_model_scores
```

| | fit_time | score_time | test_accuracy | test_precision | test_recall | test_f1 |
|---|---|---|---|---|---|---|
| 1 | 0.292 | 0.008 | 0.944 | 0.286 | 0.100 | 0.148 |
| 2 | 0.256 | 0.009 | 0.944 | 0.333 | 0.150 | 0.207 |
| 3 | 0.286 | 0.008 | 0.949 | 0.333 | 0.050 | 0.087 |
| 4 | 0.268 | 0.008 | 0.936 | 0.000 | 0.000 | 0.000 |
| 5 | 0.290 | 0.008 | 0.946 | 0.333 | 0.100 | 0.154 |
| 6 | 0.259 | 0.008 | 0.946 | 0.000 | 0.000 | 0.000 |
| 7 | 0.284 | 0.008 | 0.941 | 0.250 | 0.100 | 0.143 |
| 8 | 0.272 | 0.009 | 0.949 | 0.250 | 0.053 | 0.087 |
| 9 | 0.259 | 0.009 | 0.944 | 0.167 | 0.053 | 0.080 |
| 10 | 0.264 | 0.009 | 0.939 | 0.000 | 0.000 | 0.000 |

```
xgb_xvalid_model_scores.mean()
```

```
fit_time          0.273
score_time        0.008
test_accuracy     0.944
test_precision    0.195
test_recall       0.061
test_f1           0.091
dtype: float64
```

## 6.7.C FEATURE IMPORTANCE FOR XGBOOST (XGB) ALGORITHM

The feature importance (variable importance) describes which features are relevant. It can help with better understanding of the solved problem and sometimes lead to model improvements by employing the feature selection.

```
XGB_model.feature_importances_
```

```
array([0.08048734, 0.06615134, 0.15699951, 0.09775037, 0.0786995 ,
       0.09952894, 0.09796058, 0.06960252, 0.07558236, 0.0899193 ,
       0.08731826], dtype=float32)
```

```
feats = pd.DataFrame(index=X.columns, data=XGB_model.feature_importances_, columns=["Feature Importance"])
XGB_feature_imp = feats.sort_values("Feature Importance", ascending=False)

XGB_feature_imp
```

| | Feature Importance |
|---|---|
| age | 0.157 |
| ever_married | 0.100 |
| work_type | 0.098 |
| hypertension | 0.098 |
| bmi | 0.090 |
| smoking_status | 0.087 |
| id | 0.080 |
| heart_disease | 0.079 |
| avg_glucose_level | 0.076 |
| residence_type | 0.070 |
| gender | 0.066 |

**Visualizing Feature Importance in XGBoost Algorithm:**

```
plt.figure(figsize=(12,6))
sns.barplot(y=XGB_feature_imp["Feature Importance"], x=XGB_feature_imp.index)

plt.title("Feature Importance")
plt.xticks(rotation=45)
plt.show()
```

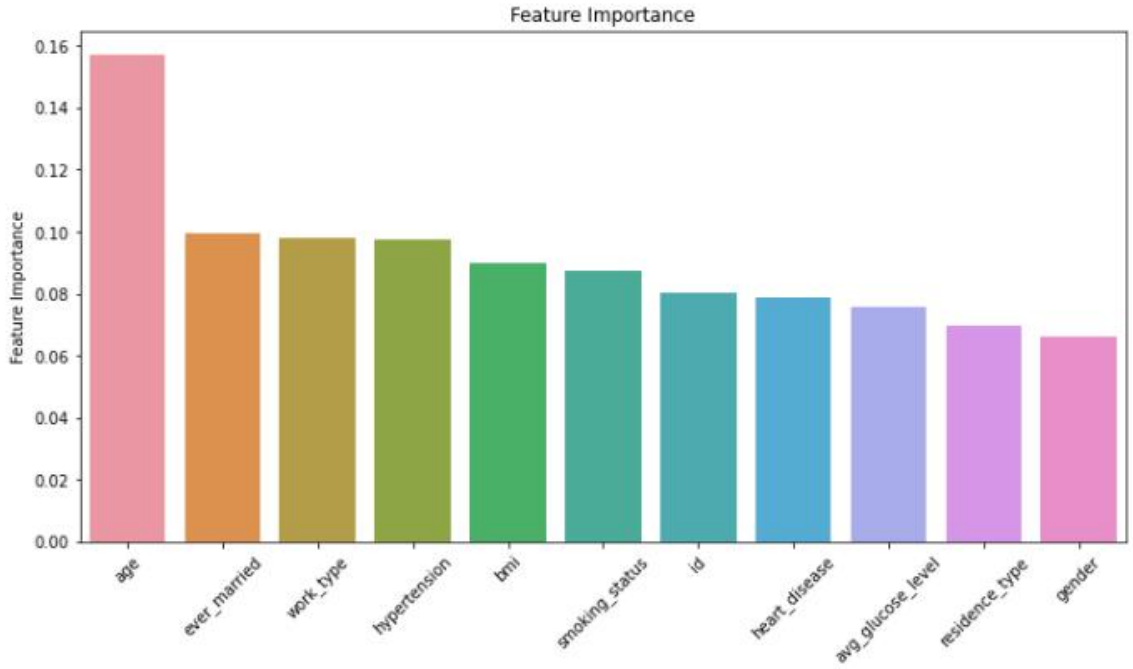



Fig 6.20 -Bar-plot for feature importance in XGBoost

## 6.7.D MODELLING XGBOOST (XGB) WITH BEST PARAMETERS USING GRIDSEARCHCV

A machine learning algorithm requires certain hyperparameters that must be tuned before training. An optimal subset of these hyperparameters must be selected, which is called hyperparameter optimization. Grid-Search is a sci-kit learn package that provides for hyperparameter tuning. A grid search space is generated by taking the initial set of values given to each hyperparameter. Each cell in the grid is searched for the optimal solution.

```
param_grid = {"n_estimators":[100, 300],
              "max_depth":[3,5,6],
              "learning_rate": [0.1, 0.3],
              "subsample":[0.5, 1],
              "colsample_bytree":[0.5, 1]}
```

```
XGB_grid_model = XGBClassifier(random_state=42)
XGB_grid_model = GridSearchCV(XGB_grid_model, param_grid, scoring = "f1", verbose=2, n_jobs = -1)
```

```
XGB_grid_model.fit(X_train_scaled, y_train)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[20:57:49] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/learner.cc:1115: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.

GridSearchCV(estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None,
                                     enable_categorical=False, gamma=None,
                                     gpu_id=None, importance_type=None,
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=None,
                                     max_depth=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_est...=None,
                                     num_parallel_tree=None, predictor=None,
                                     random_state=42, reg_alpha=None,
                                     reg_lambda=None, scale_pos_weight=None,
                                     subsample=None, tree_method=None,
                                     validate_parameters=None, verbosity=None),
             n_jobs=-1,
             param_grid={'colsample_bytree': [0.5, 1],
                         'learning_rate': [0.1, 0.3], 'max_depth': [3, 5, 6],
                         'n_estimators': [100, 300], 'subsample': [0.5, 1]},
             scoring='f1', verbose=2)
```

Let's look at the best parameters & estimator found by GridSearchCV.

```
print(colored('\033[1mBest Parameters of GridSearchCV for RF Model:\033[0m', 'blue'),
      colored(XGB_grid_model.best_params_, 'cyan'))
print("----------------------------------------------------------------------------------------------------------------------------------")
print(colored('\033[1mBest Estimator of GridSearchCV for RF Model:\033[0m', 'blue'),
      colored(XGB_grid_model.best_estimator_, 'cyan'))
```

```
Best Parameters of GridSearchCV for RF Model: {'colsample_bytree': 0.5, 'learning_rate': 0.3, 'max_depth': 6, 'n_estimators': 3
00, 'subsample': 0.5}
----------------------------------------------------------------------------------------------------------------------------------
Best Estimator of GridSearchCV for RF Model: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.5,
              enable_categorical=False, gamma=0, gpu_id=-1,
              importance_type=None, interaction_constraints='',
              learning_rate=0.3, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=300, n_jobs=8, num_parallel_tree=1, predictor='auto',
              random_state=42, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              subsample=0.5, tree_method='exact', validate_parameters=1,
              verbosity=None)
```

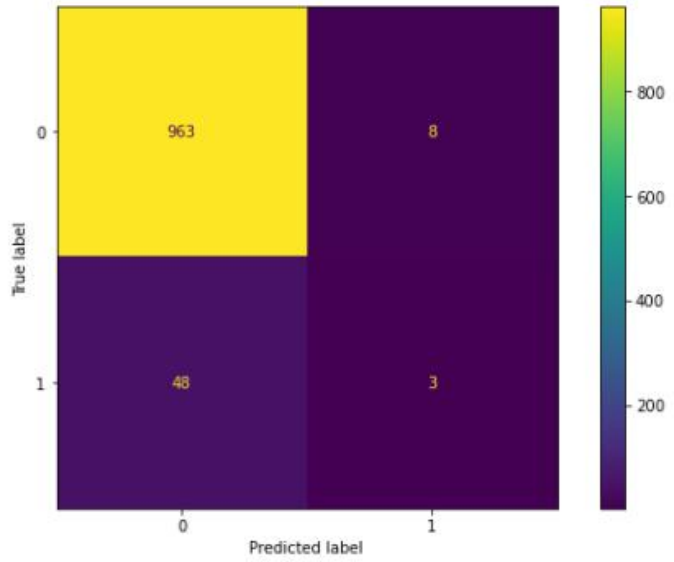
**Fit the model with Best parameters and check the performance:**

```
y_pred = XGB_grid_model.predict(X_test_scaled)
y_train_pred = XGB_grid_model.predict(X_train_scaled)

xgb_grid_f1 = f1_score(y_test, y_pred)
xgb_grid_acc = accuracy_score(y_test, y_pred)
xgb_grid_recall = recall_score(y_test, y_pred)
xgb_grid_auc = roc_auc_score(y_test, y_pred)

print(confusion_matrix(y_test, y_pred))
print("\033[1m-------------------------------------------------------------\033[0m")
print(classification_report(y_test, y_pred))
print("\033[1m-------------------------------------------------------------\033[0m")

plot_confusion_matrix(XGB_grid_model, X_test_scaled, y_test)

train_val(y_train, y_train_pred, y_test, y_pred)
```

```
[[963   8]
 [ 48   3]]
-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       971
           1       0.27      0.06      0.10        51

    accuracy                           0.95      1022
   macro avg       0.61      0.53      0.53      1022
weighted avg       0.92      0.95      0.93      1022

-------------------------------------------------------
```

Accuracy and Confusion Matrix

| | train_set | test_set |
|---|---|---|
| Accuracy | 1.000 | 0.945 |
| Precision | 1.000 | 0.273 |
| Recall | 1.000 | 0.059 |
| f1 | 1.000 | 0.097 |




## 6.7.E ROC (Receiver Operating Curve) AND AUC (Area Under Curve).

The ROC curve stands for Receiver Operating Characteristic curve. ROC curves display the performance of a classification model.ROC tells us how good the model is for distinguishing between the given classes, in terms of the predicted probability.

The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.

```
plot_roc_curve(XGB_grid_model, X_test_scaled, y_test);
```

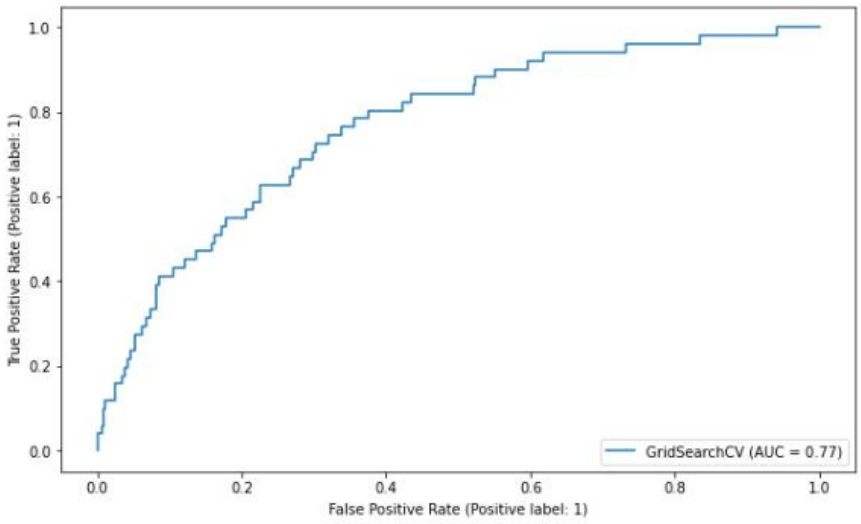



Fig 6.21 -ROC curve for XGBoost

```
plot_precision_recall_curve(XGB_grid_model, X_test_scaled, y_test);
```

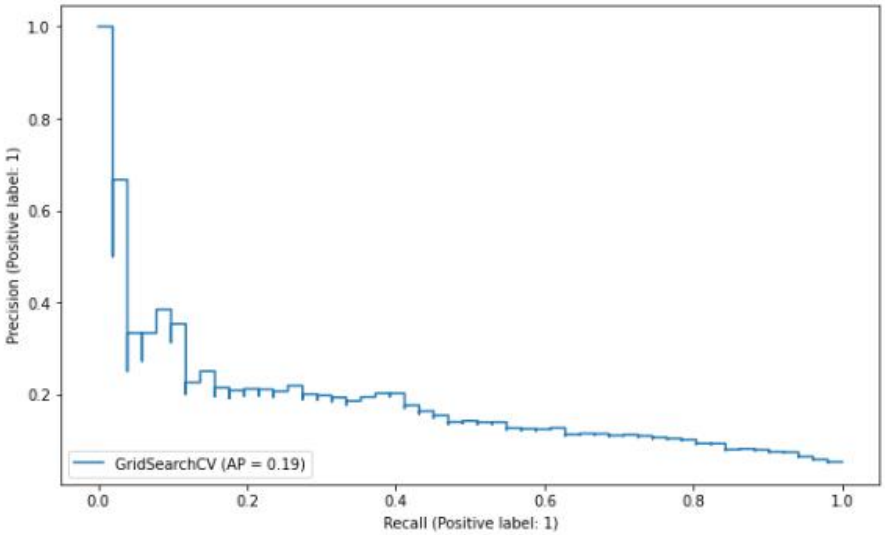



Fig 6.22 -Recall_curve for XGBoost

By checking with the XGBoost Model

We get the output as,

**Accuracy of the model:**
For Train data - 100%
For Test data - 94.5%

**Precision of the model:**
For Train data - 1.0
For Test data - 0.273

**Recall of the model:**
For Train data - 1.0
For Test data - 0.059

**F1 score of the model:**
For Train data - 1.0
For Test data - 0.097

**AUC score of the Model in XGBoost Algorithm** - 0.77

# CHAPTER 7 : COMPARISON OF MODELS

Every classification model in the project, Decision Tree, Random Forest, Support Vector Machine, K-Nearest Neighbors, Gradient Boosting Method, Adaboosting Method, Extreme Gradient Boosting method is compared by its accuracy, F1 score, precision and recall values.

## Comparison Table for the Performance of the Classification Models:

**For Train Data:**

| Model | Method | Accuracy | F1 score | Precision | Recall |
|---|---|---|---|---|---|
| Random Forest (RF) | Default Parameter | 100% | 1.0 | 1.0 | 1.0 |
| | Best parameter | 96.5% | 0.435 | 1.0 | 0.278 |
| Support Vector Machine (SVM) | Default Parameter | 98.6% | 0.458 | 0.625 | 1.0 |
| | Best parameter | 98.2% | 0.846 | 0.733 | 1.0 |
| Decision Tree (DT) | Default Parameter | 100% | 1.0 | 1.0 | 1.0 |
| | Best parameter | 96.3% | 0.423 | 0.836 | 0.283 |
| K-Nearest Neighbor(KNN) | Default Parameter | 95.5% | 0.178 | 0.741 | 0.101 |
| | Best parameter | 95.5% | 0.172 | 0.826 | 0.096 |
| Gradient Boosting Method (GBM) | Default Parameter | 96.1% | 0.326 | 0.951 | 0.197 |
| | Best parameter | 89.3% | 0.151 | 0.123 | 0.197 |
| Adaboost Method (AD) | Default Parameter | 95.2% | 0.084 | 0.562 | 0.045 |
| | Best parameter | 95.0% | 0.165 | 0.444 | 0.101 |
| Extreme Gradient Boosting (XGB) | Default Parameter | 100% | 1.0 | 1.0 | 1.0 |
| | Best parameter | 100% | 1.0 | 1.0 | 1.0 |

Table 7.1 -Comparison Train data Table for all Models

**For Test Data:**

| Model | Method | Accuracy | F1 score | Precision | Recall |
|---|---|---|---|---|---|
| Random Forest (RF) | Default Parameter | 95.1% | 0.074 | 0.667 | 0.039 |
| | Best parameter | 95.2% | 0.109 | 0.750 | 0.059 |
| Support Vector Machine (SVM) | Default Parameter | 88.8% | 0.078 | 0.045 | 0.102 |
| | Best parameter | 89.6% | 0.102 | 0.090 | 0.118 |
| Decision Tree (DT) | Default Parameter | 90.6% | 0.143 | 0.131 | 0.157 |
| | Best parameter | 94.6% | 0.225 | 0.400 | 0.157 |
| K-Nearest Neighbor(KNN) | Default Parameter | 94.8% | 0.036 | 0.250 | 0.020 |
| | Best parameter | 94.9% | 0.037 | 0.333 | 0.020 |
| Gradient Boosting Method (GBM) | Default Parameter | 94.8% | 0.036 | 0.250 | 0.020 |
| | Best parameter | 89.5% | 0.219 | 0.174 | 0.294 |
| Adaboost Method (AD) | Default Parameter | 95.1% | 0.074 | 0.667 | 0.039 |
| | Best parameter | 94.8% | 0.102 | 0.375 | 0.059 |
| Extreme Gradient Boosting (XGB) | Default Parameter | 94.4 | 0.066 | 0.200 | 0.039 |
| | Best parameter | 94.5% | 0.097 | 0.273 | 0.059 |

Table 7.2 -Comparison Test data Table for all Models

## Comparison plot for All classification Models by using Bar plot:

```
compare = pd.DataFrame({"Model": [ "SVM", "KNN", "Decision Tree", "Random Forest", "AdaBoost", "GradientBoost", "XGBoost"],
                        "F1": [ svm_grid_f1, knn_f1, DT_grid_f1, rf_grid_f1, ab_grid_f1, gb_f1, xgb_grid_f1],
                        "Recall": [ svm_grid_recall, knn_recall, DT_grid_recall, rf_grid_recall, ab_grid_recall, gb_recall,
                                   xgb_grid_recall],
                        "Accuracy": [ svm_grid_acc, knn_acc, DT_grid_acc, rf_grid_acc, ab_grid_acc, gb_acc, xgb_grid_acc],
                        "ROC_AUC": [ svm_grid_auc, knn_auc, DT_grid_auc, rf_grid_auc, ab_grid_auc, gb_auc, xgb_grid_auc]})

def labels(ax):
    for p in ax.patches:
        width = p.get_width()                          # get bar length
        ax.text(width,                                 # set the text at 1 unit right of the bar
                p.get_y() + p.get_height() / 2,        # get Y coordinate + X coordinate / 2
                '{:1.3f}'.format(width),               # set variable to display, 2 decimals
                ha = 'left',                           # horizontal alignment
                va = 'center')                         # vertical alignment

plt.figure(figsize=(14,14))
plt.subplot(411)
compare = compare.sort_values(by="F1", ascending=False)
ax=sns.barplot(x="F1", y="Model", data=compare, palette="Blues_d")
labels(ax)

plt.subplot(412)
compare = compare.sort_values(by="Recall", ascending=False)
ax=sns.barplot(x="Recall", y="Model", data=compare, palette="Blues_d")
labels(ax)

plt.subplot(413)
compare = compare.sort_values(by="Accuracy", ascending=False)
ax=sns.barplot(x="Accuracy", y="Model", data=compare, palette="Blues_d")
labels(ax)

plt.subplot(414)
compare = compare.sort_values(by="ROC_AUC", ascending=False)
ax=sns.barplot(x="ROC_AUC", y="Model", data=compare, palette="Blues_d")
labels(ax)

plt.show()
```

### F1 Score for all Classification Models:

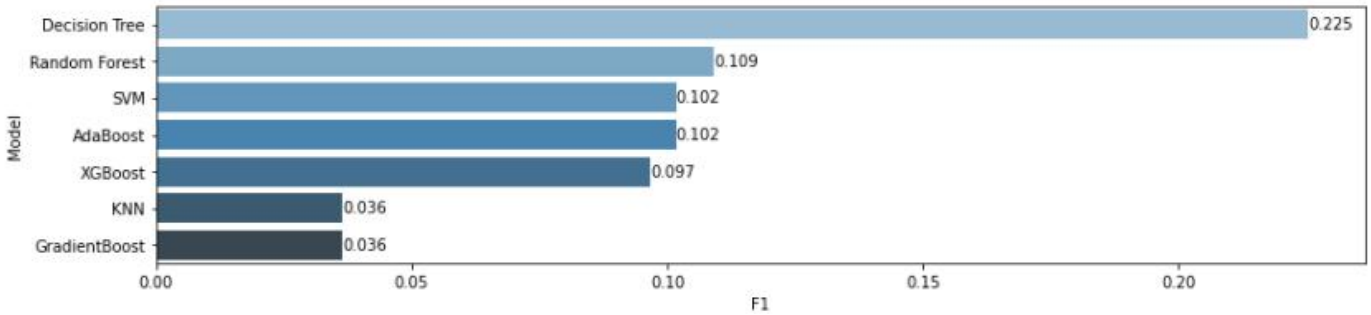



Fig 7.1 -Comparison Bar-Plot on F1_score for all Models

### Recall Score for all classification Models:

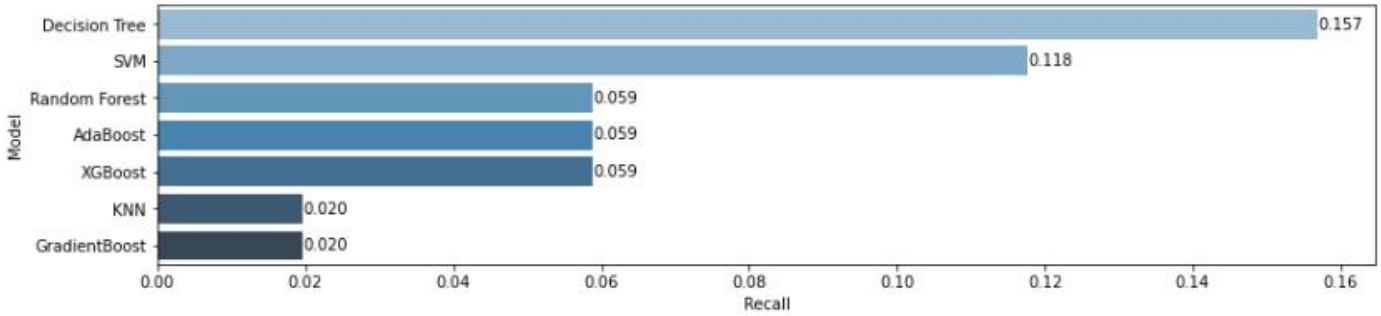



Fig 7.2 -Comparison Bar-Plot on Recall_score for all Models

**ROC_AUC Score for all Classification Models:**

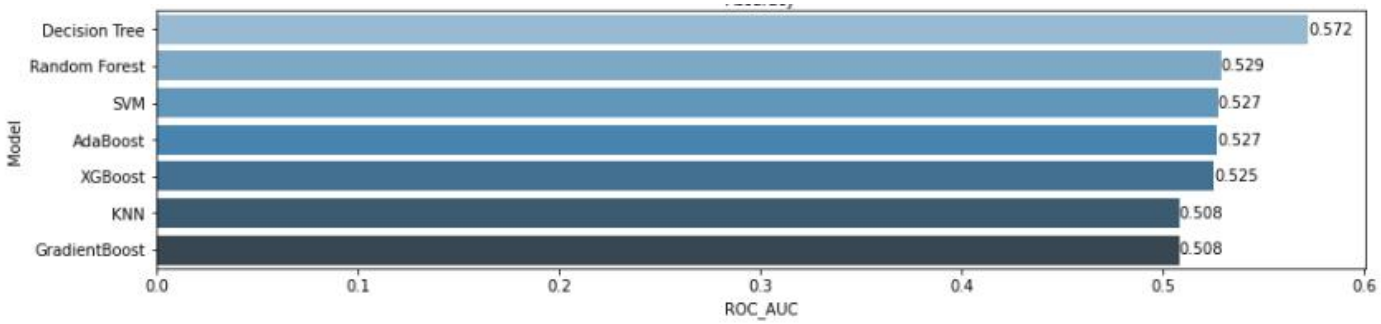



Fig 7.3 -Comparison Bar-Plot on ROC_AUC_score for all Models

**Accuracy Score for all Classification Models:**

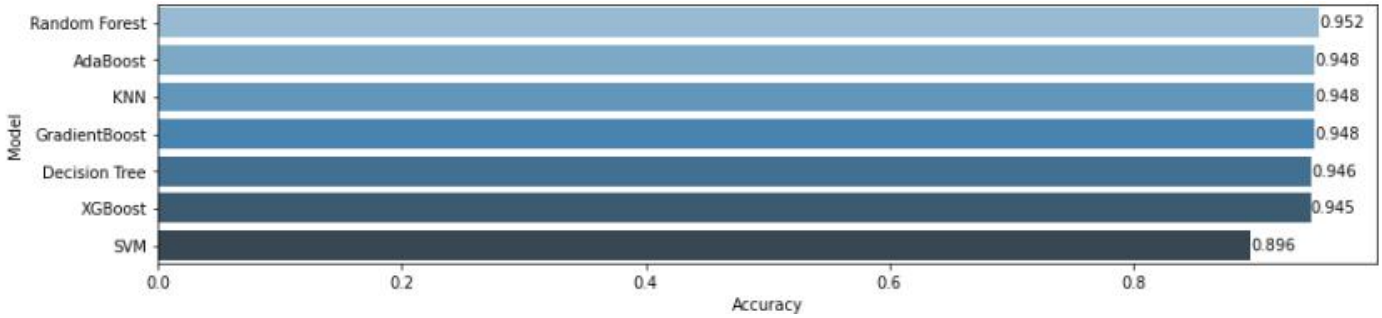



Fig 7.4 -Comparison Bar-Plot on Accuracy for all Models

# CHAPTER 8 : CONCLUSION

- In this Project Respectively,
- We have tried to a predict classification problem in Stroke Dataset by a variety of models to classify Stroke predictions in the context of determining whether anybody is likely to get Stroke based on the input parameters like gender, age and various test results or not
- We have made the detailed exploratory analysis (EDA).
- missing values are removed in the Dataset by using Simple Imputer with Median
- We have decided which metrics will be used.
- We have analysed both target and features in detail.
- We have transformed categorical variables into integer by using Label Encoder, so we can use them in the models.
- We have cross-checked the models obtained from train sets by applying cross validation for each model performance.
- We have examined the feature importance of some models.
- Lastly we have examined the results of all models visually with respect to select the best one for the problem in hand.
- By checking with all the scores like F1_score, Precision, Recall and Accuracy, The Decision Tree and Random Forest gives the best results while comparing with other models.
- For Respective Dataset the **Decision Tree** and **Random Forest** is the Best model for Future Predictions.

# CHAPTER 9 : FUTURE WORKS


- sklearn.pipeline.Pipeline — scikit-learn 1.0.1 documentation
- How to Use Power Transforms for Machine Learning (machinelearningmastery.com)
- Deep Learning Algorithms - Javatpoint
- Keras Sequential Class - Javatpoint
- TensorFlow - Installation (tutorialspoint.com)
- Visualizing Artificial Neural Networks (ANNs) with just One Line of Code | by Adesh Shah | Towards Data Science
- A Brief Overview of Outlier Detection Techniques | by Sergio Santoyo | Towards Data Science
- 11 different ways for Outlier Detection in Python - Machine Learning HD

# CHAPTER 10 : REFERENCES


- Stroke Prediction Dataset | Kaggle
- pandas.read_csv — pandas 1.3.4 documentation (pydata.org)
- Classification Algorithm in Machine Learning - Javatpoint
- sklearn.model_selection.GridSearchCV — scikit-learn 1.0.1 documentation
- Machine Learning Random Forest Algorithm - Javatpoint
- Machine Learning Decision Tree Classification Algorithm - Javatpoint
- Support Vector Machine (SVM) Algorithm - Javatpoint
- Cross-Validation in Machine Learning - Javatpoint
- K-Nearest Neighbor(KNN) Algorithm for Machine Learning - Javatpoint
- Gradient Boosting Algorithm: A Complete Guide for Beginners (analyticsvidhya.com)
- Extreme Gradient Boosting (XGBoost) Ensemble in Python (machinelearningmastery.com)
- AdaBoost Algorithm for Machine Learning in Python - CodeSpeedy
- sklearn.preprocessing.LabelEncoder — scikit-learn 1.0.1 documentation
- Feature scaling - Wikipedia
- Comparing supervised learning algorithms (dataschool.io)
- How to Handle Missing Data with Python (machinelearningmastery.com)
- The Most Used Methods to Deal with MISSING VALUES | Kaggle
- Python data visualization - Numpy, Pandas, Matplotlib and Seaborn (techlearn.live)
- Plotly Python Graphing Library | Python | Plotly