# <u>MACHINE LEARNING</u>

## (Human Resources Employee Attrition-Classifier)

*Summer Internship Report Submitted in partial fulfillment of the requirement for undergraduate degree of*

**Bachelor of Technology**

In

**Computer Science Engineering**

By

**T.Snithika Patel**

**221710304057**

*Under the Guidance of*

Professor name

(Assistant Professor)

GITAM (DEEMED TO BE UNIVERSITY)

VISAKHAPATNAM HYDERABAD BENGALURU

Department Of Computer Science and Engineering

GITAM School of Technology

GITAM (Deemed to be University), Hyderabad-502329

# DECLARATION

I submit this industrial training work entitled "HUMAN RESOURCE EMPLOYEE ATTRITION - CLASSIFIER" to GITAM (Deemed To Be University), Hyderabad in partial fulfilment of the requirements for the award of the degree of "Bachelor of Technology" in "Computer Science Engineering". I declare that it was carried out independently by me under the guidance of,Asst. Professor, GITAM (Deemed To Be University), Hyderabad, India.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

**Place: HYDERABAD** **T.Snithika Patel**

**Date:13-07-2020** **221710304057**

# CERTIFICATE

This is to certify that the Industrial Training Report entitled "HUMAN RESOURCE EMPLOYEE ATTRITION-CLASSIFIER" is being submitted by T.Snithika Patel (221710304057) in partial fulfilment of the requirement for the award of Bachelor of Technology in Computer Science Engineering at GITAM (Deemed To Be University), Hyderabad during the academic year 2018-19.

It is faithful record work carried out by her at the Computer Science Engineering Department, GITAM University Hyderabad Campus under my guidance and supervision.

| | |
|---|---|
| **Mr.** | **Dr.** |
| Assistant Professor | Professor and HOD |
| Department of ECE | Department of ECE |

# ACKNOWLEDGEMENT

Apart from my effort, the success of this internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful competition of this internship.

I would like to thank respected **Dr. N. Siva Prasad**, Pro Vice Chancellor,GITAM Hyderabad and **,** Principal, GITAM Hyderabad.

I would like to thank respected **Dr.** , Head of the Department of Computer Science Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a internship report. It helped me a lot to realize of what we study for.

I would like to thank the respected faculties **Mr.** who helped me to make this internship a successful accomplishment.

I would also like to thank my friends who helped me to make my work more organized and well-stacked till the end.

T.Snithika Patel

221710304057

# ABSTRACT

Employee Attrition is a process in which the workforce dwindles at a company, following a period in which a number of people retire or resign, and are not replaced. A reduction in staff due to attrition is often called a hiring freeze and is seen as a less disruptive way to trim the workforce and reduce payroll than layoffs. No common formula can be used by all the organizations. Attrition can also refer to a company losing its customer base, often as a result of older customers aging or moving on and fewer newer customers opting in.

This paper investigates the performance of logistic regression, random forest and naive bayes on highly skewed employee attrition data. Dataset of employee attrition contains 14999 rows and 10 columns. A hybrid technique of under-sampling and oversampling is carried out on the skewed data. The three techniques are applied on the raw and preprocessed data. The work is implemented in Python. The performance of the techniques is evaluated based on accuracy, sensitivity, specificity, precision, coefficient and balanced classification rate.

# Table of Contents

## List of Figures:

# 1. MACHINE LEARNING

## 1.1 INTRODUCTION:

Machine Learning (ML) is the scientific study of algorithms and statistical models that computer systems use in order to perform a specific task effectively without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of Artificial Intelligence (AI).

## 1.2 IMPORTANCE OF MACHINE LEARNING:

Consider some of the instances where machine learning is applied: the self-driving Google car, cyber fraud detection, online recommendation engines—like friend suggestions on Face book, Netflix showcasing the movies and shows you might like, and "more items to consider" and "get yourself a little something" on Amazon—are all examples of applied machine learning. All these examples echo the vital role machine learning has begun to take in today's data-rich world.

Machines can aid in filtering useful pieces of information that help in major advancements, and we are already seeing how this technology is being implemented in a wide variety of industries.

With the constant evolution of the field, there has been a subsequent rise in the uses, demands, and importance of machine learning. Big data has become quite a buzzword in the last few years; that's in part due to increased sophistication of machine learning, which helps analyze those big chunks of big data. Machine learning has also changed the way data extraction, and interpretation is done by involving automatic sets of generic methods that have replaced traditional statistical techniques.

The process flow depicted here represents how machine learning works

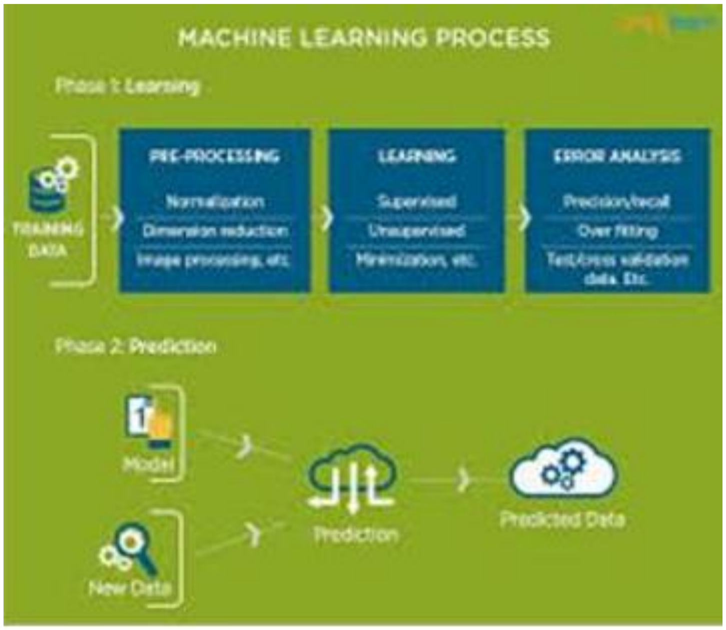



*Figure 1: The Process Flow*

## 1.3 USES OF MACHINE LEARNING:

Earlier in this article, we mentioned some applications of machine learning. To understand the concept of machine learning better, let's consider some more examples: web search results, real-time ads on web pages and mobile devices, email spam filtering, network intrusion detection, and pattern and image recognition. All these are by-products of applying machine learning to analyze huge volumes of data

Traditionally, data analysis was always being characterized by trial and error, an approach that becomes impossible when data sets are large and heterogeneous. Machine learning comes as the solution to all this chaos by proposing clever alternatives to analyzing huge volumes of data. By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning can produce accurate results and analysis.

## 1.4 TYPES OF LEARNING ALGORITHMS:

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

### 1.4.1 Supervised Learning:

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning. Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to "learn" how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data. Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign. Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification.

### 1.4.2 Unsupervised Learning:

When an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

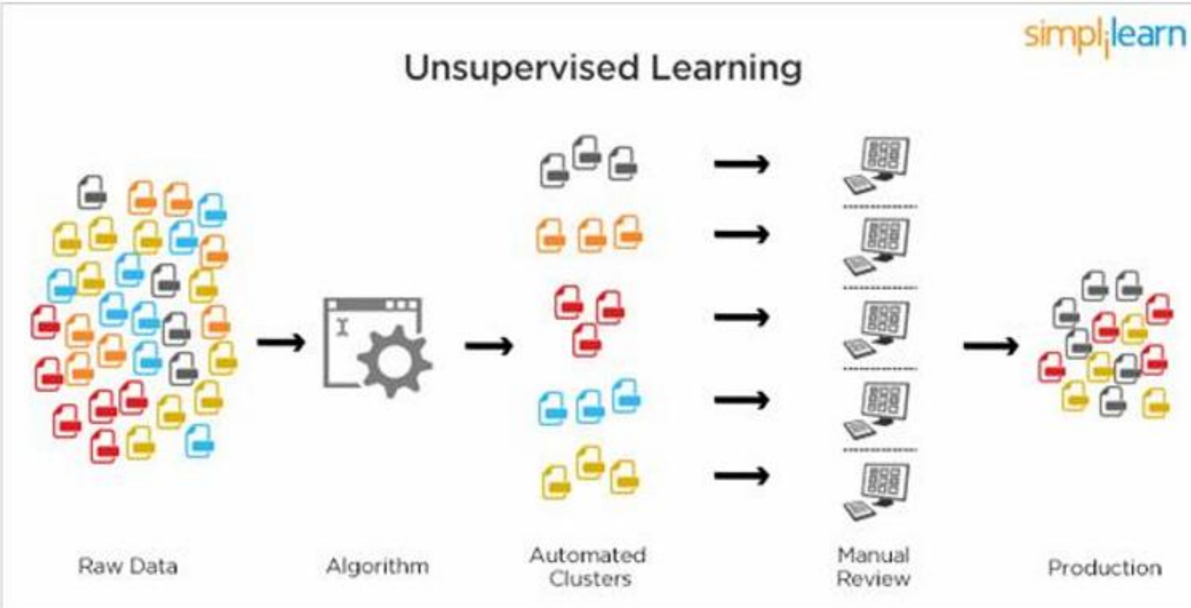



*Figure 2: Unsupervised Learning.*

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbour mapping, singular value decomposition, and k-means clustering. Basically, online recommendations, identification of data outliers, and segment text topics are all examples of unsupervised learning.

### 1.4.3 Semi Supervised Learning:

As the name suggests, semi-supervised learning is a bit of both supervised and unsupervised learning and uses both labelled and unlabeled data for training. In a typical scenario, the algorithm would use a small amount of labelled data with a large amount of unlabeled data.

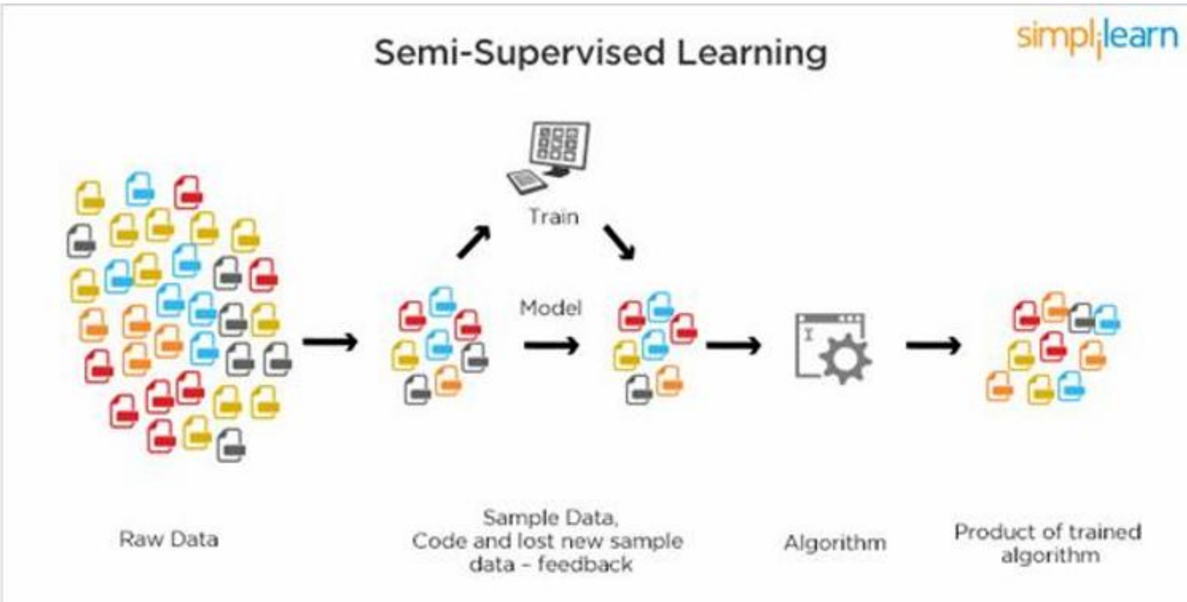



*Figure 3: Semi Supervised Learning*

## 1.5 RELATION BETWEEN DATA MINING, MACHINE LEARNING AND DEEP LEARNING:

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovers previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions. Deep learning, on the other hand, uses advanced computing power and special types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

# 2. PYTHON

Basic programming language used for machine learning is: PYTHON

## 2.1 INTRODUCTION TO PYHTON:

- Python is a high-level, interpreted, interactive and object-oriented scripting language.

- Python is a general purpose programming language that is often applied in scripting roles
- Python is interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.
- Python is Interactive: You can sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: Python supports the Object-Oriented style or technique of programming that encapsulates code within objects.

## 2.2 HISTORY OF PYTHON:

- Python was developed by GUIDO VAN ROSSUM in early 1990's.
- Its latest version is 3.7 , it is generally called as python3

## 2.3 FEATURES OF PYTHON:

- **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax: This allows the student to pick up the language quickly.
- **Easy-to-read:** Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintaining.
- **A broad standard library:** Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.
- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

## 2.4 HOW TO SETUP PYTHON:

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

## 2.4.1 Installation (using python IDLE):

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.
- Download python from www.python.org
- When the download is completed, double click the file and follow the instructions to install it.
- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.

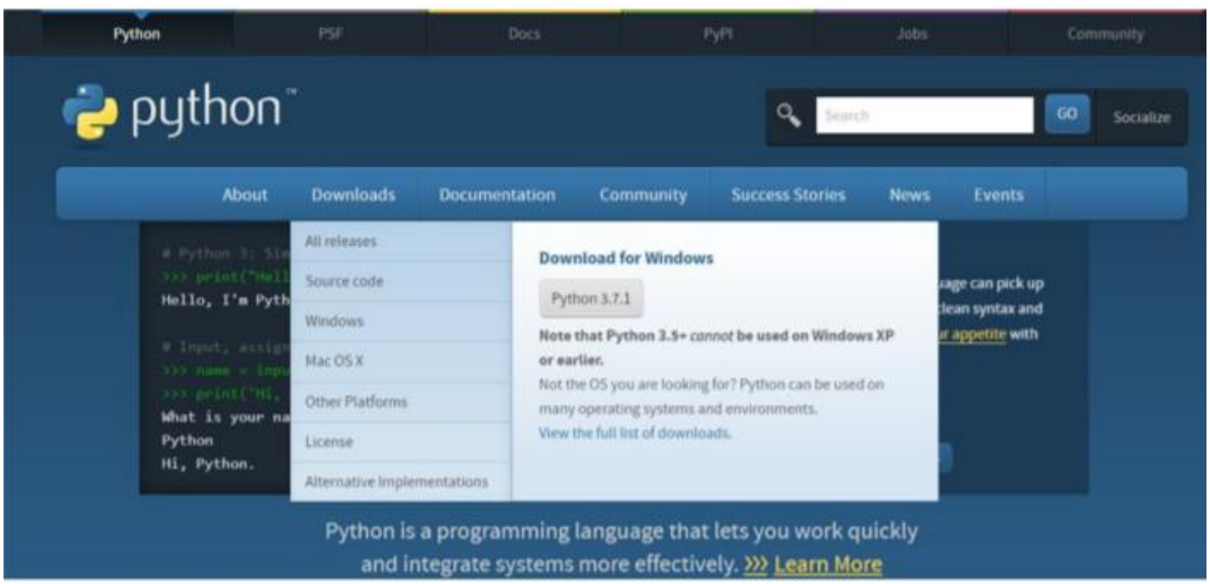



*Figure 4 : Python download*

## 2.4.2 Installation (using Anaconda):

- Python programs are also executed using Anaconda.

- Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.
- Conda is a package manager quickly installs and manages packages.
- In WINDOWS:
- Step 1: Open Anaconda.com/downloads in a web browser.
- Step 2: Download python 3.4 version for (32-bitgraphic installer/64 -bit graphic installer)
- Step 3: select installation type( all users)
- Step 4: Select path(i.e. add anaconda to path & register anaconda as default python 3.4) next click install and next click finish
- Step 5: Open jupyter notebook ( it opens in default browser)

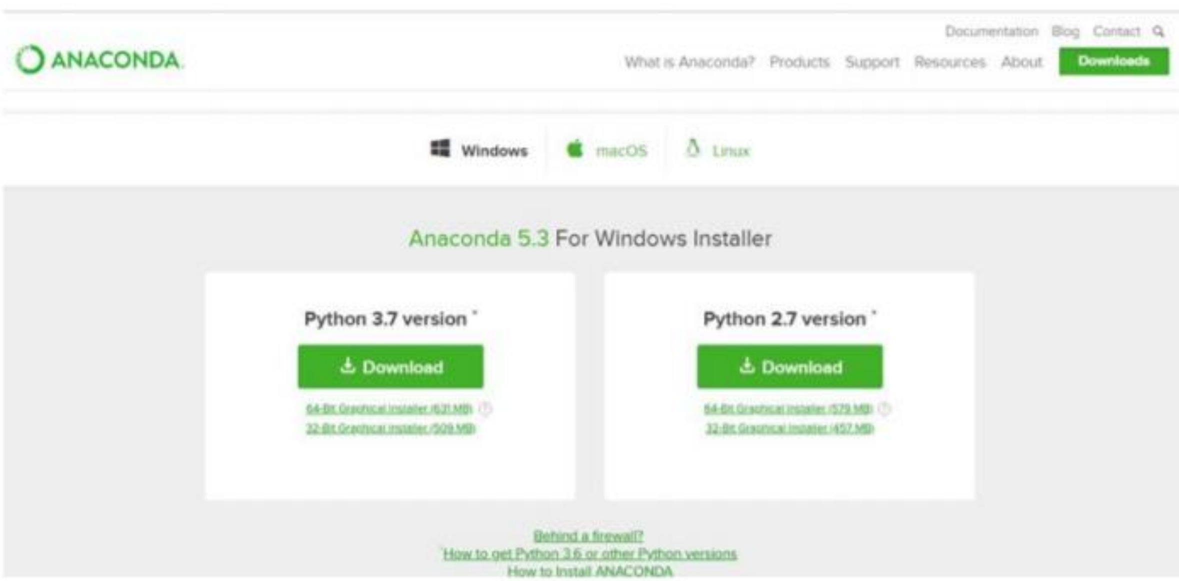



*Figure 5: Anaconda download*

*Figure 6: Jupyter notebook*

## 2.5 PYTHON VARIABLE TYPES:

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Variables are nothing but reserved memory locations to store values.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types –
    - Numbers
    - Strings
    - Lists
    - Tuples
    - Dictionary

### 2.5.1 Python Numbers:

- Number data types store numeric values. Number objects are created when you assign a value to them.
- Python supports four different numerical types − int (signed integers) long (long integers, they can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

### 2.5.2 Python Strings:

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.

- Subsets of strings can be taken using the slice operator ([ ] and [:] ) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

### 2.5.3 Python Lists:

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets ([]).
- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

### 2.5.4 Python Tuples:

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed, while tuples are enclosed in parentheses ( ( ) ) and cannot be updated.
- Tuples can be thought of as read-only lists.
- For example − Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add elements to a tuple. Tuples have no append or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

### 2.5.5 Python Dictionary:

- Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).
- You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of a list.
- What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

## 2.6 PYTHON FUNCTION:

### 2.6.1 Defining a Function:

You can define functions to provide the required functionality. Here are simple rules to define a function in Python. Function blocks begin with the keyword def followed by the function name and parentheses (i.e.()).

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses The code block within every function starts with a colon (:) and is indented. The statement returns [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

### 2.6.2 Calling a Function:

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

## 2.7 PYTHON USING OOP's CONCEPTS:

### 2.7.1 Class:

- Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- Data member: A class variable or instance variable that holds data associated with a class and its objects.
- Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.
- Defining a Class: o We define a class in a very similar way how we define a function. o Just like a function, we use parentheses and a colon after the class name(i.e. ():) when we define a class. Similarly, the body of our class is indented like a function body is.

```python
def my_function():
    # the details of the
    # function go here
```

```python
class MyClass():
    # the details of the
    # class go here
```

*Figure 7: Defining a Class*

### 2.7.2 __init__ method in Class:

- The init method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.
- The init method has a special name that starts and ends with two underscores:__init__().

# 3. CASE STUDY

## 3.1 PROBLEM STATEMENT:

To understand the reason behind employee leaving the company .

## 3.2 DATA SET:

The given dataset contains following parameters:

- satisfaction_level (0–1)
- last_evaluation (Time since last evaluation in years)
- number_of_projects (Number of projects completed while at work)
- average_monthly_hours (Average monthly hours at workplace)
- years_spend_company (Time spent at the company in years)
- Work_accident (Whether the employee had a workplace accident)
- left (Whether the employee left the workplace or not (1 or 0))
- promotion_last_5years (Whether the employee was promoted in the last five years)
- Department (Department in which they work for)
- salary (Relative level of salary)

## 3.3 OBJECTIVE OF THE CASE STUDY:

The company wants to understand what factors contributed most to employee and to create a model that can predict if a certain employee will leave the company or not. The goal is to create or improve different retention strategies on targeted employees. Overall, the implementation of this model will allow management to create better decision-making actions.

# 4. MODEL BUILDING

## 4.1 PREPROCESSING OF THE DATA:

Preprocessing of the data actually involves the following steps:

### 4.1.1 GETTING THE DATASET:

We can get the data set from the database or we can get the data from the client.

## 4.1.2 IMPORTING THE LIBRARIES:

We have to import the libraries as per the requirement of the algorithm.

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

*Figure 8: Importing Libraries*

## 4.1.3 IMPORTING THE DATA-SET:

Pandas in python provide an interesting method read_csv(). The read_csv function reads the entire dataset from a comma separated values file and we can assign it to a DataFrame to which all the operations can be performed. It helps us to access each and every row as well as columns and each and every value can be access using the dataframe. Any missing value or NaN value have to be cleaned.

```
df = pd.read_csv("Human_Resources_Employee_Attrition.csv")
df
```

| tisfaction_level | last_evaluation | number_of_projects | average_monthly_hours | years_at_company | work_accident | left | promotion_last_5years | department | salary |
|---|---|---|---|---|---|---|---|---|---|
| 0.38 | 0.53 | 2 | 157 | 3 | 0 | 1 | 0 | sales | low |
| 0.80 | 0.86 | 5 | 262 | 6 | 0 | 1 | 0 | sales | medium |
| 0.11 | 0.88 | 7 | 272 | 4 | 0 | 1 | 0 | sales | medium |
| 0.72 | 0.87 | 5 | 223 | 5 | 0 | 1 | 0 | sales | low |
| 0.37 | 0.52 | 2 | 159 | 3 | 0 | 1 | 0 | sales | low |
| 0.41 | 0.50 | 2 | 153 | 3 | 0 | 1 | 0 | sales | low |
| 0.10 | 0.77 | 6 | 247 | 4 | 0 | 1 | 0 | sales | low |
| 0.92 | 0.85 | 5 | 259 | 5 | 0 | 1 | 0 | sales | low |
| 0.89 | 1.00 | 5 | 224 | 5 | 0 | 1 | 0 | sales | low |
| 0.42 | 0.53 | 2 | 142 | 3 | 0 | 1 | 0 | sales | low |
| 0.45 | 0.54 | 2 | 135 | 3 | 0 | 1 | 0 | sales | low |
| 0.11 | 0.81 | 6 | 305 | 4 | 0 | 1 | 0 | sales | low |
| 0.84 | 0.92 | 4 | 234 | 5 | 0 | 1 | 0 | sales | low |
| 0.41 | 0.55 | 2 | 148 | 3 | 0 | 1 | 0 | sales | low |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **14983** | 0.72 | 0.84 | 5 | 257 | 5 | 0 | 1 | 0 | technic |
| **14984** | 0.40 | 0.56 | 2 | 148 | 3 | 0 | 1 | 0 | technic |
| **14985** | 0.91 | 0.99 | 5 | 254 | 5 | 0 | 1 | 0 | technic |
| **14986** | 0.85 | 0.85 | 4 | 247 | 6 | 0 | 1 | 0 | technic |
| **14987** | 0.90 | 0.70 | 5 | 206 | 4 | 0 | 1 | 0 | technic |
| **14988** | 0.46 | 0.55 | 2 | 145 | 3 | 0 | 1 | 0 | technic |
| **14989** | 0.43 | 0.57 | 2 | 159 | 3 | 1 | 1 | 0 | technic |
| **14990** | 0.89 | 0.88 | 5 | 228 | 5 | 1 | 1 | 0 | suppc |
| **14991** | 0.09 | 0.81 | 6 | 257 | 4 | 0 | 1 | 0 | suppc |
| **14992** | 0.40 | 0.48 | 2 | 155 | 3 | 0 | 1 | 0 | suppc |
| **14993** | 0.76 | 0.83 | 6 | 293 | 6 | 0 | 1 | 0 | suppc |
| **14994** | 0.40 | 0.57 | 2 | 151 | 3 | 0 | 1 | 0 | suppc |
| **14995** | 0.37 | 0.48 | 2 | 160 | 3 | 0 | 1 | 0 | suppc |
| **14996** | 0.37 | 0.53 | 2 | 143 | 3 | 0 | 1 | 0 | suppc |
| **14997** | 0.11 | 0.96 | 6 | 280 | 4 | 0 | 1 | 0 | suppc |
| **14998** | 0.37 | 0.52 | 2 | 158 | 3 | 0 | 1 | 0 | suppc |

14999 rows × 10 columns

*Figure 9: Reading the dataset*

## 4.1.4 HANDLING MISSING VALUES:

Missing values can be handled in many ways using some inbuilt methods:

1. dropna()
2. fillna()
3. interpolate()
4. mean imputation and median imputation .

**<u>1. dropna():</u>**

dropna() is a function which drops all the rows and columns which are having the missing values(i.e. NaN).

dropna() function has a parameter called how which works as follows:

- if how = ’all’ is passed then it drops the rows where all the columns of the particular row are missing.
- if how = ’any’ is passed then it drops the rows where all the columns of the particular row are missing.

**<u>2. fillna():</u>**

fillna() is a function which replaces all the missing values using different ways

fillna() also have parameters called method and axis.

- if we use method = ’ffill’ where ffill is a method called forward fill, which carry forwards the previous row’s value .
- if we use method = ‘bfill’ where bfill is a method called backward fill, which carry backward the next row’s value .
- if we use method = ‘ffill’ , axis = ‘columns’ then it carry forwards the previous column’s value .
- if we use method = ‘bfill’ , axis = ‘columns’ then it carry backward the next column’s value .

**3. interpolate():**

- interpolate() is a function which comes up with a guess value based on the other values in the dataset and fills those guess values in the place of missing values .

**4. mean and median imputation**

- mean and median imputation can be performed by using fillna().
- mean imputation calculates the mean for the entire column and replaces the missing values in that column with the calculated mean.
- median imputation calculates the median for the entire column and replaces the missing values in that column with the calculated median.

Missing values can be checked using isna() or isnull() functions which returns the output in a boolean format.

Total number of missing values in each column can be calculated using isna().sum() or isnull().sum().

```
#To check missing values in the each column.
df.isnull()
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **14988** | False | False | False | False | False | False | False | False |
| **14989** | False | False | False | False | False | False | False | False |
| **14990** | False | False | False | False | False | False | False | False |
| **14991** | False | False | False | False | False | False | False | False |
| **14992** | False | False | False | False | False | False | False | False |
| **14993** | False | False | False | False | False | False | False | False |
| **14994** | False | False | False | False | False | False | False | False |
| **14995** | False | False | False | False | False | False | False | False |
| **14996** | False | False | False | False | False | False | False | False |
| **14997** | False | False | False | False | False | False | False | False |
| **14998** | False | False | False | False | False | False | False | False |

14999 rows × 10 columns

*Figure 10: Checking missing values.*

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

```
satisfaction_level       0
last_evaluation          0
number_of_projects       0
average_monthly_hours    0
years_at_company         0
work_accident            0
left                     0
promotion_last_5years    0
department               0
salary                   0
dtype: int64
```

*Figure 11: Total number of missing values in each column.*

**From the above output we can observe that the given dataset do not contain any missing values.**

```
sns.heatmap(df.isna())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xd617e9cd30>
```

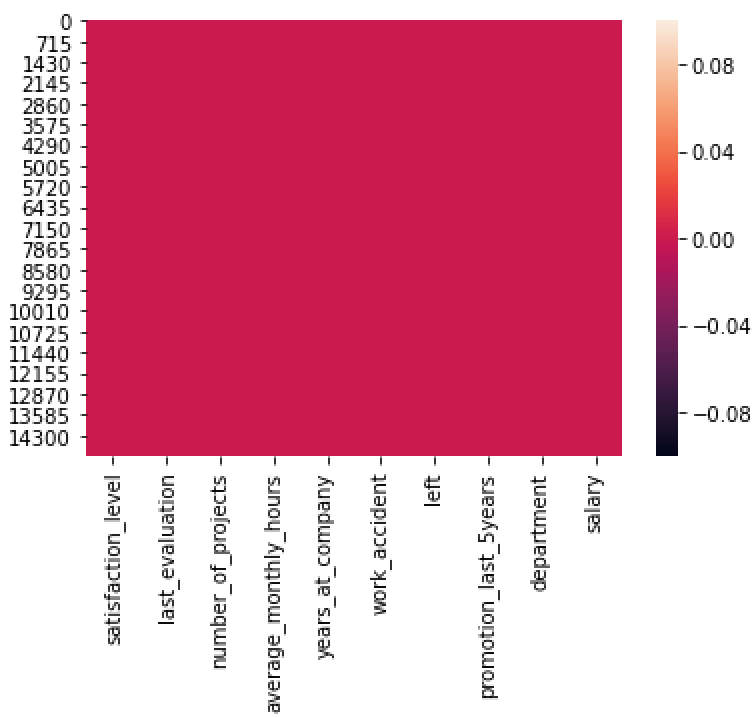



*Figure 12: Visualising the missing values.*

## 4.1.5 OUTLIERS:

An outlier is a data point in a data set that is distant from all other observations. A data point that lies outside the overall distribution of the dataset.

```
df.boxplot(column='number_of_projects')
```

<matplotlib.axes._subplots.AxesSubplot at 0xd61a38c160>

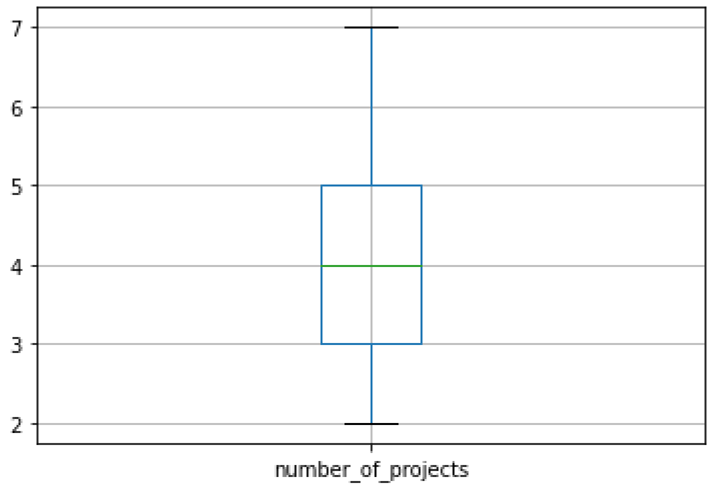



*Figure 13: Box plot for number of projects*

```
df.boxplot(column='years_at_company')
```

<matplotlib.axes._subplots.AxesSubplot at 0xd23724a6a0>

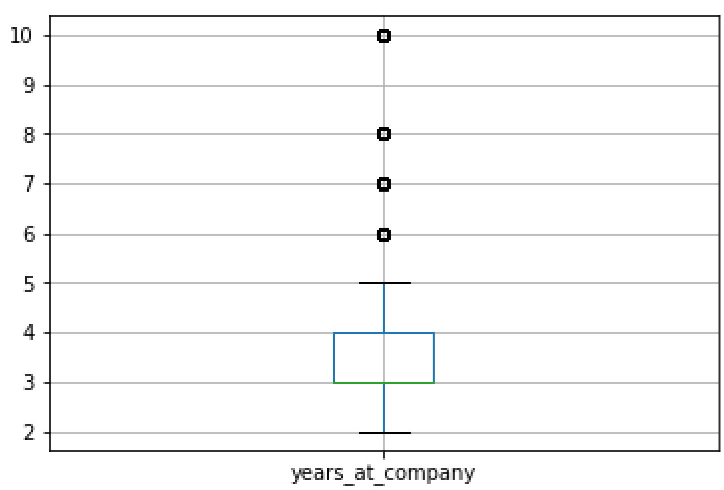


*Figure 14: Box plot for years at company*

## Observations of the Figure 13 and Figure 14:

In figure 13 the box plot has no outliers for number of projects and Figure 14 the box plot has outliers for years at company.

```
# Getting the count of people who leave and not leave
leftcounts=df['left'].value_counts()
print(leftcounts)
# Using matplotlib pie chart and label the pie chart
plt.pie(leftcounts,labels=['not leave','leave']);
```

```
0    11428
1     3571
Name: left, dtype: int64
```

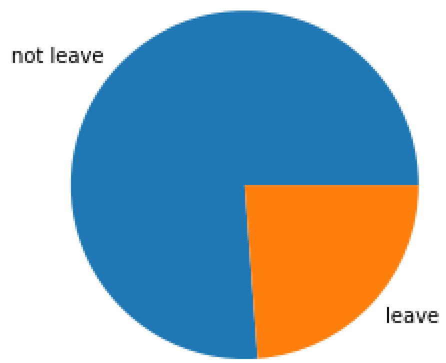



*Figure 15: Pie chart plot for getting the count of people who leave and not left*

```
# Getting data of employee who leave and do not leave
leftdata=df[df['left']==1]
notleftdata=df[df['left']==0]
# Getting the shapes and number of these people
print("shape of leftdata",leftdata.shape)
print("shape of notleftdata",notleftdata.shape)
```

```
shape of leftdata (3571, 10)
shape of notleftdata (11428, 10)
```

```
# Getting the distribution of satisfaction_level
sns.distplot(leftdata['satisfaction_level'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x6fd47a17b8>
```

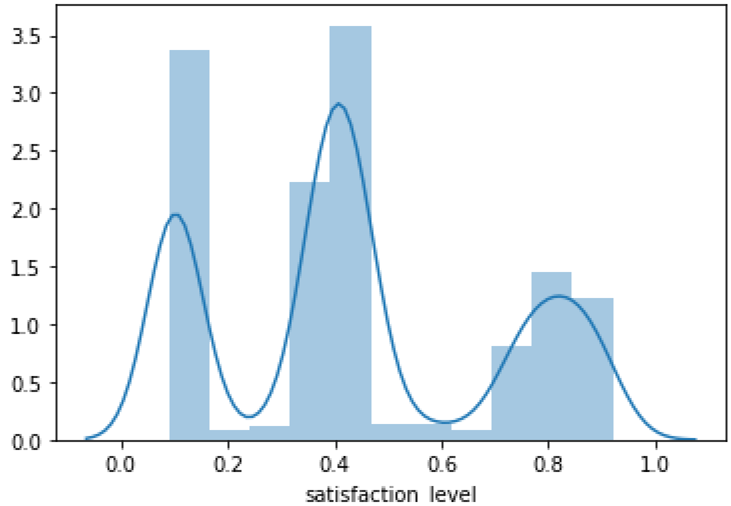



*Figure 16: Sub plot for getting the distribution of satisfaction level*

```
#Creating a figure instance and the two subplots
fig=plt.figure(figsize=(15,10))
x1=fig.add_subplot(221)
x2=fig.add_subplot(222)
sns.distplot(leftdata['average_monthly_hours'],kde=True,ax=x1)
sns.distplot(notleftdata['average_monthly_hours'],kde=True,ax=x2)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x6fdf0ce3c8>
```

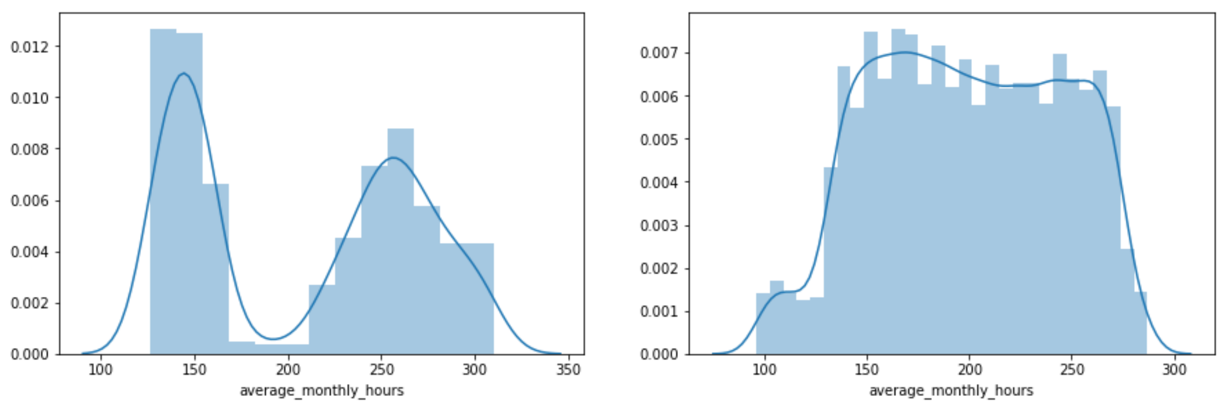



*Figure 17: Creating a figure instance and the two Subplots of distplots of left data and not left data, they vary from each other based on average monthly hours*

```
fig=plt.figure(figsize=(15,10))
x1=fig.add_subplot(221)
x2=fig.add_subplot(222)
sns.distplot(leftdata['number_of_projects'],kde=True,ax=x1)
sns.distplot(notleftdata['number_of_projects'],kde=True,ax=x2)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x6fdf4a9780>
```

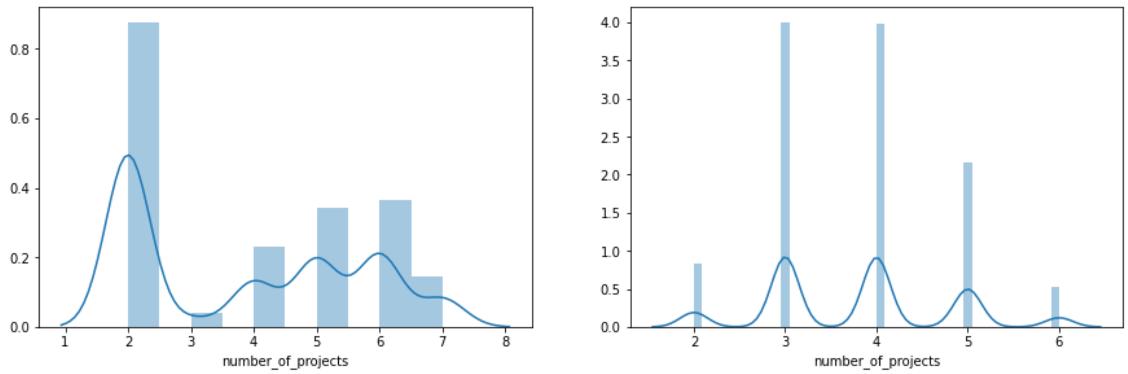



*Figure 18: Creating a figure instance and the two Subplots of distplots of left data and not left data ,they vary from each other based on number of projects.*

```
#Creating a figure instance and the two subplots
fig=plt.figure(figsize=(15,10))
x1=fig.add_subplot(221)
x2=fig.add_subplot(222)
sns.distplot(leftdata['last_evaluation'],kde=True,ax=x1)
sns.distplot(notleftdata['last_evaluation'],kde=True,ax=x2)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x6fe21f69b0>
```

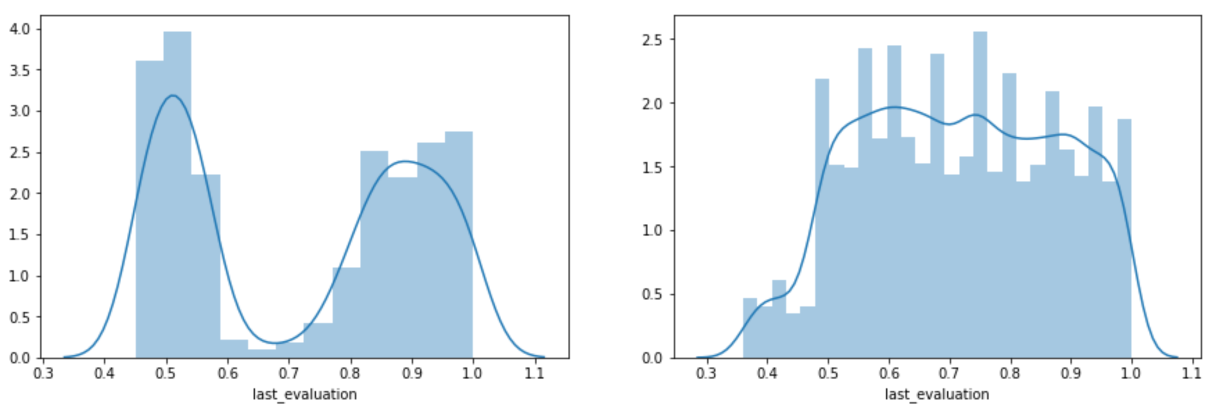



*Figure 19: Creating a figure instance and the two Subplots of distplots of left data and not left data, they vary from each other based on last Evaluation.*

```
fig=plt.figure(figsize=(15,10))
x1=fig.add_subplot(221)
x2=fig.add_subplot(222)
leftpromotion_last_5yearscounts=leftdata['promotion_last_5years'].value_counts()
notleftpromotion_last_5yearscounts=notleftdata['promotion_last_5years'].value_counts()
# Plot each pie chart in a separate subplot
x1.pie(leftpromotion_last_5yearscounts,labels=leftpromotion_last_5yearscounts.index)
x2.pie(notleftpromotion_last_5yearscounts,labels=notleftpromotion_last_5yearscounts.index)
```

```
([<matplotlib.patches.Wedge at 0x6fe26a0518>,
  <matplotlib.patches.Wedge at 0x6fe26a0ac8>],
 [Text(-1.0962613188355268, 0.09061523506006805, '0'),
  Text(1.0962613177750256, -0.09061524788999208, '1')])
```


*Figure 20: pie chart plot for employees who left with promotion last 5 years and who did not leave*

```
#Create a figure with  two subplots
fig=plt.figure(figsize=(15,10))
x1=fig.add_subplot(221)
x2=fig.add_subplot(222)
leftdepartmentcounts=leftdata['salary'].value_counts()
notleftdepartmentcounts=notleftdata['salary'].value_counts()
# Plot each pie chart in a separate subplot
x1.pie(leftdepartmentcounts,labels=leftdepartmentcounts.index)
x2.pie(notleftdepartmentcounts,labels=notleftdepartmentcounts.index)
```

```
([<matplotlib.patches.Wedge at 0x6fe2727748>,
  <matplotlib.patches.Wedge at 0x6fe2727c88>,
  <matplotlib.patches.Wedge at 0x6fe27331d0>],
 [Text(0.17165976256904342, 1.0865233204652074, 'low'),
  Text(-0.5022972074318767, -0.978620209992691, 'medium'),
  Text(1.0450162802225114, -0.3434253544366018, 'high')])
```

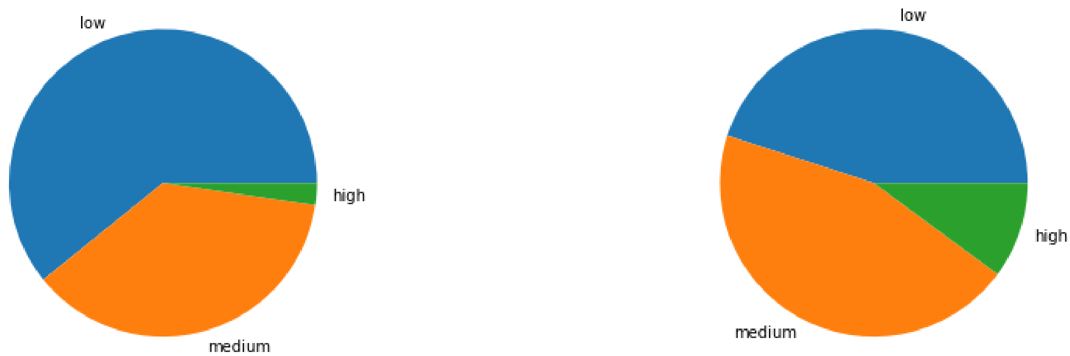



*Figure 21: Pie chart plot for employees who left with how much salary and and who did not leave*

## 4.1.6 CATEGORICAL DATA:

- Machine Learning models are based on equations, we need to replace the text by numbers. So that we can include the numbers in the equations.

  Categorical Variables are of two types: Nominal and Ordinal

- **Nominal:**

  The categories do not have any numeric ordering in between them. They don't have any ordered relationship between each of them. Examples: Male or Female, any colour

- **Ordinal:**

The categories have a numerical ordering in between them. Example: Graduate is less than Post Graduate, Post Graduate is less than Ph.D. customer satisfaction survey, high low medium

- Categorical data can be handled by using dummy variables, which are also called as indicator variables.

Handling categorical data using dummies: In pandas library we have a method called get_dummies() which creates dummy variables for those categorical data in the form of 0's and 1's. Once these dummies got created we have to concat this dummy set to our dataframe or we can add that dummy set to the dataframe

```
: df.dtypes

: satisfaction_level       float64
  last_evaluation          float64
  number_of_projects         int64
  average_monthly_hours      int64
  years_at_company           int64
  work_accident              int64
  left                       int64
  promotion_last_5years      int64
  department                object
  salary                     int64
  dtype: object
```

*Figure 22: Description about the type of each feature in the dataset.(Categorical or Numerical).*

## 4.2 TRAINING THE MODEL:

```
# Getting data of employee who leave and do not leave
leftdata=df[df['left']==1]
notleftdata=df[df['left']==0]
# Getting the shapes and number of these people
print("shape of leftdata",leftdata.shape)
print("shape of notleftdata",notleftdata.shape)

shape of leftdata (3571, 10)
shape of notleftdata (11428, 10)
```

*Figure 23: Imbalanced data*

**Since the dataset is imbalanced, it is balanced using SMOTE.**

In Machine Learning and Data Science we often come across a term called Imbalanced Data Distribution, generally happens when observations in one of the class are much higher or lower than the other classes. As Machine Learning algorithms tend to increase accuracy by reducing the error, they do not consider the class distribution. This problem is prevalent in examples such as Fraud Detection, Anomaly Detection, Facial recognition etc.

Standard ML techniques such as Decision Tree and Logistic Regression have a bias towards the majority class, and they tend to ignore the minority class. They tend only to predict the majority class, hence, having major misclassification of the minority class in comparison with the majority class. In more technical words, if we have imbalanced data distribution in our dataset then our model becomes more prone to the case when minority class has negligible or very lesser recall.

Imbalanced Data Handling Techniques: There are mainly 2 mainly algorithms that are widely used for handling imbalanced class distribution.

1. SMOTE
2. Near Miss Algorithm

### Balancing the dataset

```
from imblearn.combine import SMOTETomek
smk = SMOTETomek(random_state=120)
X,y = smk.fit_sample(df.drop(['left','department'],axis=1),df['left'])
```

```
y.value_counts()
```

```
1    11385
0    11385
Name: left, dtype: int64
```

*Figure 24: Balancing the dataset*

## 4.2.1 Method 1:

- **Splitting the data :** after the preprocessing is done then the data is split into train and test sets.
- In Machine Learning in order to access the performance of the classifier. You train the classifier using 'training set' and then test the performance of your classifier on unseen 'test set'. An important point to note is that during training the classifier only uses the training set . The test set must not be used during training the classifier. The test set will only be available during testing the classifier.
- training set - a subset to train a model.(Model learns patterns between Input and Output)
- test set - a subset to test the trained model.(To test whether the model has correctly learnt)
- The amount or percentage of Splitting can be taken as specified (i.e. train data = 75% , test data =25% or train data = 80% , test data= 20%) .
- First we need to identify the input and output variables and we need to separate the input set and output set.
- In scikit learn library we have a package called model_selection in which train_test_split method is available .we need to import this method.
- This method splits the input and output data to train and test based on the percentage specified by the user and assigns them to four different variables(we need to mention the variables).

```
#Splitting the dataset into training and test data.
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(18216, 8)
(4554, 8)
(18216,)
(4554,)
```

*Figure 25 : importing train_test_split and splitting the data.*

- Then we need to import logistic regression method from linear_model package from scikit learn library

- We need to train the model based on our train set (that we have obtained from splitting)
- Then we have to test the model for the test set ,that is done as follows
    - We have a method called predict , using this method we need to predict the output for the input test set and we need to compare the output with the output test data.
    - If the predicted values and the original values are close then we can say that model is trained with good accuracy .

## 4.2.2 Method 2:

```
corr=df.corr()
corr
```

| | satisfaction_level | last_evaluation | number_of_projects | average_monthly_hours | years_at_company | work_accident | left | promotion_ |
|---|---|---|---|---|---|---|---|---|
| **satisfaction_level** | 1.000000 | 0.105021 | -0.142970 | -0.020048 | -0.100866 | 0.058697 | -0.388375 | |
| **last_evaluation** | 0.105021 | 1.000000 | 0.349333 | 0.339742 | 0.131591 | -0.007104 | 0.006567 | |
| **number_of_projects** | -0.142970 | 0.349333 | 1.000000 | 0.417211 | 0.196786 | -0.004741 | 0.023787 | |
| **average_monthly_hours** | -0.020048 | 0.339742 | 0.417211 | 1.000000 | 0.127755 | -0.010143 | 0.071287 | |
| **years_at_company** | -0.100866 | 0.131591 | 0.196786 | 0.127755 | 1.000000 | 0.002120 | 0.144822 | |
| **work_accident** | 0.058697 | -0.007104 | -0.004741 | -0.010143 | 0.002120 | 1.000000 | -0.154622 | |
| **left** | -0.388375 | 0.006567 | 0.023787 | 0.071287 | 0.144822 | -0.154622 | 1.000000 | |
| **promotion_last_5years** | 0.025605 | -0.008684 | -0.006064 | -0.003544 | 0.067433 | 0.039245 | -0.061788 | |

*Figure 26 : Correlation*

```
fig = plt.subplots (figsize = (10, 10))
sns.heatmap(df.corr (), square = True, cbar = True, annot = True, cmap="GnBu", annot_kws = {'size': 8})
plt.title('Correlations between Attributes')
plt.show()
```

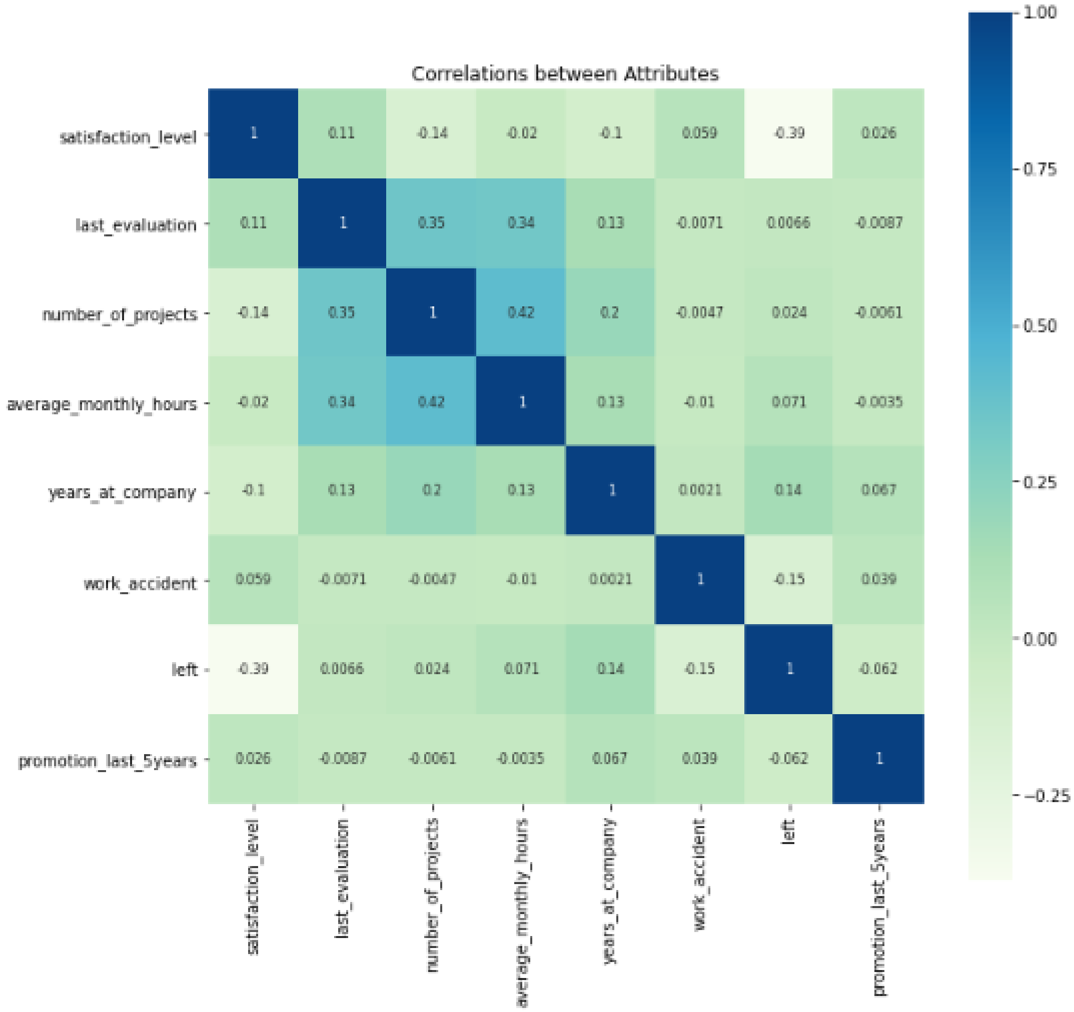



*Figure 27: Correlations between Attributes using heatmap*

- **Correlation:** Correlation is a statistical technique that can show whether and how strongly pairs of variables are related. Correlation is described as the analysis which lets us know the association or the absence of the relationship between two variables 'x' and 'y'. It is a statistical measure that represents the strength of the connection between pairs of variables.
- We can also find the column which is effecting the R-Squared value so that we can perform operations on that specific column or we can remove that column, this can be done using pair plot.
- In pair plot we need to find the correlation between two variables and we can do some 33 operations on that variables.

- pairplot is a method which is available in seaborn library, so to use this pairplot method we have to import seaborn library.

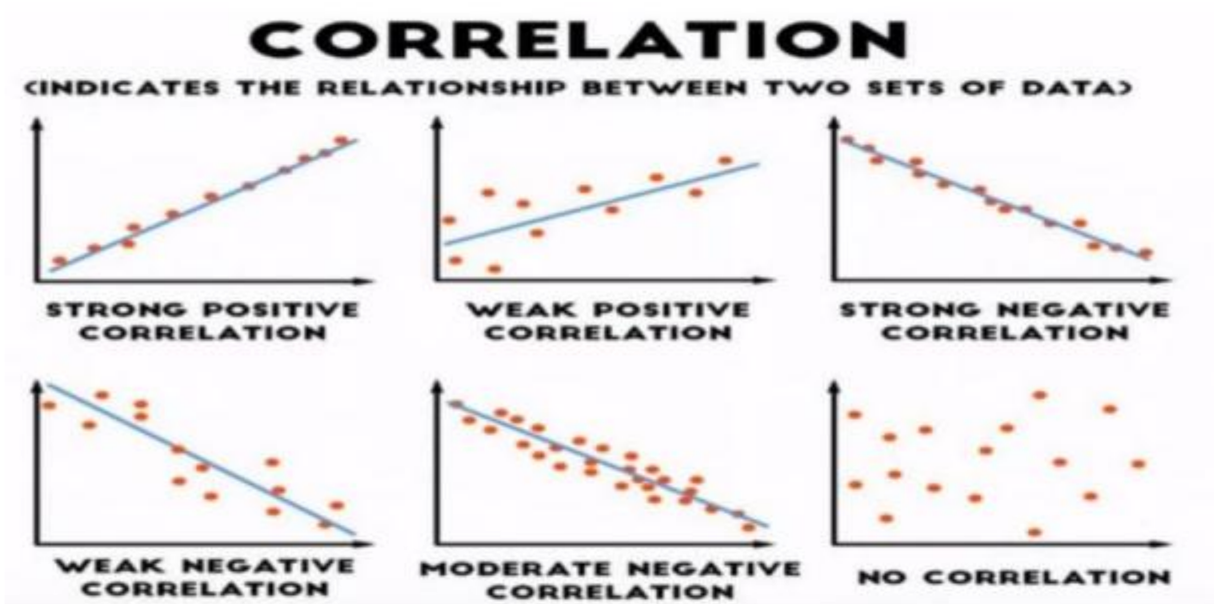



*Figure 28: Correlation between two sets of data*

## Classification Report:

A Classification report is used to measure the quality of predictions from a classification algorithm. How many predictions are True and how many are False. More specifically, True Positives, False Positives, True negatives and False Negatives are used to predict the metrics of a classification report as shown below.

- **Precision:**

Precision is the ability of a classifier not to label an instance positive that is actually negative. For each class it is defined as the ratio of true positives to the sum of true and false positives.

TP – True Positives
FP – False Positives

Precision – Accuracy of positive predictions.
Precision = TP/(TP + FP)

- **Recall:**

Recall is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives.

FN – False Negatives

Recall: Fraction of positives that were correctly identified.
Recall = TP/(TP+FN)

- **F1 score:**

The $F_1$ score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. Generally speaking, $F_1$ scores are lower than accuracy measures as they embed precision and recall into their computation. As a rule of thumb, the weighted average of $F_1$ should be used to compare classifier models, not global accuracy.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

## Confusion Matrix:

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.For a binary classification problem, we would have a 2 x 2 matrix as shown below with 4 values:

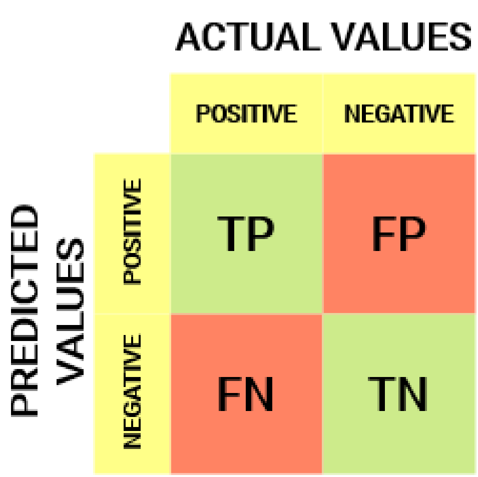



*Figure 29: Confusion matrix*

- TP: True Positive: Predicted values correctly predicted as actual positive
- FP: Predicted values incorrectly predicted an actual positive. i.e., Negative values predicted as positive
- FN: False Negative: Positive values predicted as negative
- TN: True Negative: Predicted values correctly predicted as an actual negative

**ACCURACY:**

It is a metric used to predict the correctness of a machine learning model. The model is trained using the train data and a classifier is built. The test data is used to cross validate the classifier model. The percentage of correctly classified instances is termed as **accuracy**.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

## 4.3 Model Building and Evaluation

### 4.3.1 Logistic regression

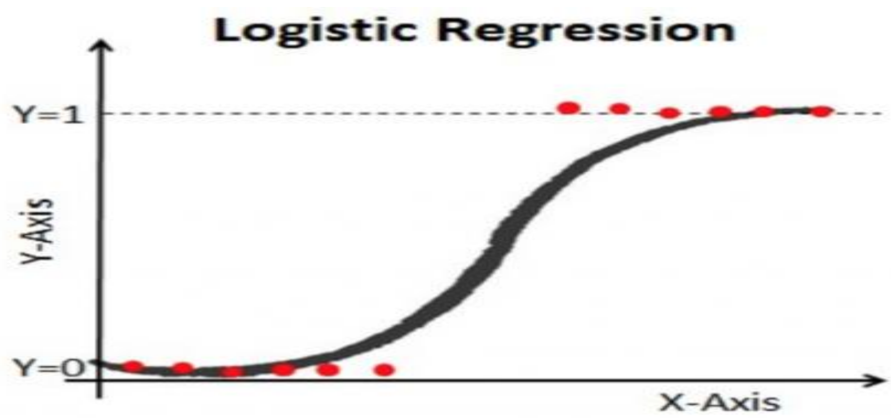

Logistic Regression is used when the dependent variable (target) is categorical.

Logistic Regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis and it predicts the probability

Example: Yes or No, get a disease or not, pass or fail, defective or non-defective, etc.,

Also called a classification algorithm, because we are classifying the data. It predicts the probability associated with each dependent variable category.

## Training and Testing the logistic regression without scaling




```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression() # creating an object for Logistic Regression
# we have to apply this object(Log_reg) to the training data
final_model1 = log_reg.fit(X_train,y_train) # with the help of fit method we are fitting logistic regression with training data
##objectName.fit(InputData, outputData)
```

*Figure 30: Applying logistic regression on training data.*

**Instead of directly predicting on test data, let us see how well the model predicts the training data.**

**<u>Predicting on training data</u>**

```
y_train_pred = log_reg.predict(X_train)
y_train_pred
```

```
array([1, 1, 0, ..., 0, 1, 1], dtype=int64)
```

*Figure 31: Predicting on train data*

```
y_train == y_train_pred # comparing original data o/p and model predicted o/p
```

```
5496     False
20638     True
11889     True
16256    False
4665      True
13958     True
10341     True
3261      True
14200    False
12033     True
1177      True
10133     True
20159    False
4204      True
10583    False
16638     True
17985     True
14731     True
19492    False
6890      True
16553     True
7310      True
19766     True
3950      True
2913     False
10622     True
14524    False
17861     True
28        True
15510     True
         ...
22505     True
6285      True
1110      True
18272     True
11742     True
17137     True
19433     True
16946     True
4764      True
19946    False
8444      True
18900    False
2962      True
12645     True
21758     True
21780    False
3462      True
10989     True
7751      True
16332     True
20609     True
144       True
21440     True
19279     True
7813      True
10955     True
17289     True
5192      True
12172     True
235       True
Name: left, Length: 18216, dtype: bool
```

*Figure 32:  comparing the predicted value with the original one .*

```
from sklearn.metrics import classification_report,confusion_matrix
confusion_matrix(y_train,y_train_pred)
```

```
array([[6812, 2292],
       [1698, 7414]], dtype=int64)
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_train,y_train_pred)
```

```
0.780961791831357
```

*Figure 33: Applying the metrics on training data.*

**<u>Predicting on test data</u>**

```python
# Predicting the model on test data
y_test_pred = log_reg.predict(X_test)
```

```python
y_test_pred
```

```
array([1, 1, 1, ..., 0, 1, 1], dtype=int64)
```

*Figure 34: Predicting on test data.*

```
11342    False
14859     True
16285     True
16632     True
15100     True
5699      True
19960     True
9345      True
22463     True
3455     False
7853      True
22473     True
7759      True
6420      True
7397      True
11957     True
246       True
16050    False
7364     False
20417     True
8937      True
9248      True
15456     True
153       True
6830      True
565       True
17770     True
16098     True
18950     True
10766     True
          ...
8523      True
9767      True
4492     False
4765      True
7502      True
1967      True
10595     True
283       True
14315    False
20377    False
4901     False
21207     True
6005      True
3419      True
12083     True
21928    False
19092     True
13281     True
4530      True
4173      True
9373      True
863       True
5221     False
21748     True
23        True
16105     True
9761      True
5829      True
21883     True
12338     True
Name: left, Length: 4554, dtype: bool
```

*Figure 35: comparing the predicted value with the original test data*

```
confusion_matrix(y_test,y_test_pred)
```

```
array([[1692,  589],
       [ 378, 1895]], dtype=int64)
```

```
accuracy_score(y_test,y_test_pred)
```

```
0.7876592007026789
```

*Figure 36: Applying metrics on test data*

```
#classification report on training and test data
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_train,y_train_pred))
print("-----------------------------------------------------")
print(classification_report(y_test,y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.75      0.77      9104
           1       0.76      0.81      0.79      9112

    accuracy                           0.78     18216
   macro avg       0.78      0.78      0.78     18216
weighted avg       0.78      0.78      0.78     18216

-----------------------------------------------------
              precision    recall  f1-score   support

           0       0.82      0.74      0.78      2281
           1       0.76      0.83      0.80      2273

    accuracy                           0.79      4554
   macro avg       0.79      0.79      0.79      4554
weighted avg       0.79      0.79      0.79      4554
```

*Figure 37: Overall performance of the logistic regression model based on training and test data.*

## Scaling the data:

It is a step of Data Pre Processing which is applied to independent variables or features of data. It basically helps to normalise the data within a particular range. Sometimes, it also helps in speeding up the calculations in an algorithm.

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
scaler = StandardScaler().fit(X_train)
```

*Figure 38: importing standard scaler and minmaxscaler for applying scaling*

```
scaled_X_train = pd.DataFrame(scaler.fit_transform(X_train))
scaled_X_train
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | -0.430421 | -1.317760 | -1.247261 | 0.425560 | -0.464701 | -0.337325 | -0.119626 | -0.851365 |
| 1 | -1.716510 | 1.208426 | 1.513216 | 1.203719 | 0.294832 | -0.337325 | -0.119626 | 0.798823 |
| 2 | 0.996152 | -0.260605 | -0.557142 | -0.093213 | -0.464701 | 2.964504 | -0.119626 | -0.851365 |
| 3 | 1.258942 | 1.130389 | 0.823097 | 0.629363 | 1.054365 | -0.337325 | -0.119626 | 0.798823 |
| 4 | 0.020076 | 0.240153 | -0.557142 | -1.278978 | -1.224234 | -0.337325 | -0.119626 | 2.449011 |
| 5 | 0.395490 | -0.650083 | -0.557142 | 0.129118 | -1.224234 | -0.337325 | -0.119626 | -0.851365 |
| 6 | 0.207783 | 0.518351 | 0.823097 | -1.241923 | -0.464701 | -0.337325 | -0.119626 | -0.851365 |
| 7 | -1.293874 | -0.817002 | 0.823097 | 0.666419 | -0.464701 | -0.337325 | -0.119626 | 2.449011 |
| 8 | 1.108776 | 1.074749 | 0.823097 | 0.425560 | 1.054365 | -0.337325 | -0.119626 | 0.798823 |
| 9 | -0.467963 | -1.095201 | -1.247261 | -0.834316 | -0.464701 | -0.337325 | -0.119626 | 0.798823 |
| 10 | -0.730753 | -1.150841 | -1.247261 | -0.889899 | -0.464701 | -0.337325 | -0.119626 | -0.851365 |
| 11 | 0.057617 | 1.408587 | 0.823097 | -0.130268 | -1.224234 | -0.337325 | -0.119626 | 0.798823 |
| 12 | 1.106981 | 1.571342 | 0.823097 | 0.999915 | 1.054365 | -0.337325 | -0.119626 | 0.798823 |
| 13 | 1.596815 | -1.206480 | 0.132977 | 1.129608 | -0.464701 | -0.337325 | -0.119626 | 2.449011 |
| 14 | -0.092549 | 1.352948 | 0.132977 | 1.277829 | -1.224234 | -0.337325 | -0.119626 | -0.851365 |
| 15 | -0.587782 | -1.328424 | -1.247261 | -1.130758 | -0.464701 | -0.337325 | -0.119626 | -0.851365 |
| 16 | -0.399970 | -1.241104 | -1.247261 | -1.278978 | -0.464701 | -0.337325 | -0.119626 | 0.798823 |
| 17 | 0.395490 | -0.260605 | -1.247261 | 0.962860 | -0.464701 | -0.337325 | -0.119626 | -0.851365 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **18200** | -0.618128 | -1.484679 | -1.247261 | -1.278978 | -0.464701 | -0.337325 | -0.119626 | 0.798823 |
| **18201** | 0.089849 | 0.339078 | -0.557142 | -0.500820 | -0.464701 | -0.337325 | -0.119626 | 0.798823 |
| **18202** | 1.371566 | 1.186028 | 0.132977 | 0.814639 | -1.224234 | -0.337325 | -0.119626 | 0.798823 |
| **18203** | 0.545655 | 0.240153 | 0.132977 | -0.556402 | -1.224234 | -0.337325 | -0.119626 | 0.798823 |
| **18204** | 0.545655 | 0.351432 | 0.823097 | 1.129608 | -1.224234 | -0.337325 | -0.119626 | 0.798823 |
| **18205** | -1.677697 | 0.678597 | 1.513216 | 1.074025 | 0.294832 | -0.337325 | -0.119626 | -0.851365 |
| **18206** | 1.055206 | 1.384706 | 0.132977 | 0.240284 | 1.054365 | -0.337325 | -0.119626 | -0.851365 |
| **18207** | 1.071235 | 1.519867 | 0.132977 | 1.185191 | 1.054365 | -0.337325 | -0.119626 | -0.851365 |
| **18208** | -0.678815 | -1.052529 | -1.247261 | -1.390144 | -0.464701 | -0.337325 | -0.119626 | -0.851365 |
| **18209** | -0.665350 | -1.113164 | -1.247261 | -1.241923 | -0.464701 | -0.337325 | -0.119626 | -0.851365 |
| **18210** | -1.368957 | -1.317760 | 1.513216 | -0.463764 | -0.464701 | 2.964504 | -0.119626 | 0.798823 |
| **18211** | 1.146318 | 0.573991 | 0.132977 | 1.314884 | -1.224234 | 2.964504 | -0.119626 | -0.851365 |
| **18212** | -1.740674 | 0.785592 | 1.513216 | 1.407522 | 0.294832 | -0.337325 | -0.119626 | -0.851365 |
| **18213** | 1.183859 | -1.150841 | 0.132977 | -0.556402 | -0.464701 | -0.337325 | -0.119626 | -0.851365 |
| **18214** | -0.580587 | -1.206480 | -1.247261 | -1.390144 | -0.464701 | -0.337325 | -0.119626 | -0.851365 |
| **18215** | 1.033694 | 0.740910 | 0.132977 | 0.962860 | 1.054365 | -0.337325 | -0.119626 | -0.851365 |

18216 rows × 8 columns

*Figure 39: scaling for X_train*

```
scaled_X_test = pd.DataFrame(scaler.fit_transform(X_test))
scaled_X_test
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | -1.444883 | -0.016880 | -0.601140 | -0.045439 | -0.487079 | -0.325387 | -0.134568 | 0.818920 |
| 1 | -1.630040 | 0.937433 | 1.472270 | 1.145501 | 0.284186 | -0.325387 | -0.134568 | 0.818920 |
| 2 | -0.509743 | -1.420282 | -1.292277 | -1.291346 | -0.487079 | -0.325387 | -0.134568 | -0.842266 |
| 3 | -1.667071 | 0.712889 | 2.163406 | 1.347044 | 0.284186 | -0.325387 | -0.134568 | 0.818920 |
| 4 | -1.667071 | 0.488344 | 1.472270 | 1.548588 | 0.284186 | -0.325387 | -0.134568 | 0.818920 |
| 5 | 0.739962 | -0.185289 | -0.601140 | 1.255434 | -1.258344 | -0.325387 | -0.134568 | 0.818920 |
| 6 | 1.143977 | 0.661799 | 0.089996 | 0.742413 | 1.826716 | -0.325387 | -0.134568 | -0.842266 |
| 7 | 1.443557 | 0.656753 | 0.781133 | 0.632480 | -0.487079 | -0.325387 | -0.134568 | -0.842266 |
| 8 | -1.651139 | 1.033713 | 1.472270 | 1.493622 | 0.284186 | -0.325387 | -0.134568 | -0.842266 |
| 9 | -1.518946 | -1.476418 | -1.292277 | 1.127178 | 1.826716 | 3.073264 | -0.134568 | -0.842266 |
| 10 | 1.369494 | 0.319936 | -0.601140 | -0.210339 | -0.487079 | -0.325387 | -0.134568 | 0.818920 |
| 11 | -1.674296 | 0.892250 | 1.472270 | 1.402011 | 0.284186 | -0.325387 | -0.134568 | -0.842266 |
| 12 | 1.591682 | -0.185289 | -0.601140 | 1.255434 | -0.487079 | -0.325387 | -0.134568 | -0.842266 |
| 13 | -0.519101 | 1.330386 | 0.781133 | -0.705037 | -0.487079 | 3.073264 | -0.134568 | -0.842266 |
| 14 | -0.074726 | -1.195738 | 0.089996 | -0.576782 | -1.258344 | -0.325387 | -0.134568 | 0.818920 |
| 15 | -0.519101 | -1.476418 | -1.292277 | -1.419601 | -0.487079 | -0.325387 | -0.134568 | -0.842266 |
| 16 | -0.667227 | -1.364146 | -1.292277 | -0.851614 | -0.487079 | -0.325387 | -0.134568 | -0.842266 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **4543** | 1.517619 | 0.656753 | 0.089996 | 1.218789 | 0.284186 | 3.073264 | -0.134568 | 0.818920 |
| **4544** | -0.111757 | 0.432208 | 0.781133 | 0.064494 | -1.258344 | -0.325387 | -0.134568 | -0.842266 |
| **4545** | 0.702931 | 0.881297 | 0.089996 | 0.779058 | 1.826716 | -0.325387 | -0.134568 | -0.842266 |
| **4546** | 0.110430 | 0.039256 | -0.601140 | 1.090534 | 1.826716 | -0.325387 | -0.134568 | -0.842266 |
| **4547** | -1.646585 | 1.118382 | 1.472270 | 0.907313 | 0.284186 | -0.325387 | -0.134568 | 0.818920 |
| **4548** | -0.333945 | -0.858922 | -1.292277 | -1.218057 | -0.487079 | -0.325387 | -0.134568 | -0.842266 |
| **4549** | -0.632799 | -1.247927 | -1.292277 | -1.199735 | -0.487079 | -0.325387 | -0.134568 | 0.818920 |
| **4550** | 0.925119 | -0.465969 | 0.781133 | -0.466849 | -1.258344 | -0.325387 | -0.134568 | -0.842266 |
| **4551** | 0.702931 | -0.522105 | -0.601140 | -0.466849 | -1.258344 | -0.325387 | -0.134568 | -0.842266 |
| **4552** | -0.482070 | -1.391228 | -1.292277 | -1.401279 | -0.487079 | -0.325387 | -0.134568 | -0.842266 |
| **4553** | 0.999181 | 1.442658 | 0.089996 | 0.504225 | 1.055451 | -0.325387 | -0.134568 | -0.842266 |

4554 rows × 8 columns

*Figure 40: Scaling for X_test*

```
from sklearn.linear_model import LogisticRegression
```

```
log_reg1 = LogisticRegression(multi_class = 'ovr', solver = 'sag',  max_iter = 10000)
log_reg1.fit(scaled_X_train, y_train)
```

```
LogisticRegression(max_iter=10000, multi_class='ovr', solver='sag')
```

```
y_train_pred_lr = log_reg1.predict(scaled_X_train)
```

```
y_train_pred_lr
```

```
array([1, 1, 0, ..., 0, 1, 1], dtype=int64)
```

*Figure 41: Applying scaling for Logistic Regression*

```
confusion_matrix = metrics.confusion_matrix(y_train, y_train_pred_lr) #train
confusion_matrix
```

```
array([[6820, 2284],
       [1633, 7479]], dtype=int64)
```

```
print(classification_report(y_train, y_train_pred_lr))
```

```
              precision    recall  f1-score   support

           0       0.81      0.75      0.78      9104
           1       0.77      0.82      0.79      9112

    accuracy                           0.78     18216
   macro avg       0.79      0.78      0.78     18216
weighted avg       0.79      0.78      0.78     18216
```

*Figure 42: Confusion matrix for y_train*

```
y_test_pred_lr = log_reg1.predict(scaled_X_test)
```

```
confusion_matrix = metrics.confusion_matrix(y_test, y_test_pred_lr)
confusion_matrix
```

```
array([[1710,  571],
       [ 369, 1904]], dtype=int64)
```

```
print(classification_report(y_test, y_test_pred_lr))
```

```
              precision    recall  f1-score   support

           0       0.82      0.75      0.78      2281
           1       0.77      0.84      0.80      2273

    accuracy                           0.79      4554
   macro avg       0.80      0.79      0.79      4554
weighted avg       0.80      0.79      0.79      4554
```

*Figure 43: Confusion matrix for y_test*

```
acc_lr = metrics.accuracy_score(y_train, y_train_pred_lr) #train
acc_lr
```

```
0.7849692577953448
```

```
acc_lr = metrics.accuracy_score(y_test, y_test_pred_lr) #test
acc_lr
```

```
0.7935880544576197
```

*Figure 44: Accuracy scores for y_train ,y_test*

```
y_test_prob1 = final_model1.predict_proba(X_test)
y_test_prob1 = pd.DataFrame(y_test_prob1)
y_test_prob1
```

| | 0 | 1 |
|---|---|---|
| **0** | 0.165851 | 0.834149 |
| **1** | 0.186385 | 0.813615 |
| **2** | 0.241841 | 0.758159 |
| **3** | 0.254097 | 0.745903 |
| **4** | 0.178016 | 0.821984 |
| **5** | 0.764116 | 0.235884 |
| **6** | 0.361766 | 0.638234 |
| **7** | 0.832655 | 0.167345 |
| **8** | 0.077252 | 0.922748 |
| **9** | 0.091992 | 0.908008 |
| **10** | 0.856575 | 0.143425 |
| **11** | 0.079242 | 0.920758 |
| **12** | 0.712102 | 0.287898 |
| **13** | 0.815028 | 0.184972 |
| **14** | 0.793783 | 0.206217 |
| **15** | 0.248443 | 0.751557 |
| **16** | 0.187517 | 0.812483 |

| | | |
|---|---|---|
| **4535** | 0.243078 | 0.756922 |
| **4536** | 0.809384 | 0.190616 |
| **4537** | 0.536904 | 0.463096 |
| **4538** | 0.200865 | 0.799135 |
| **4539** | 0.591453 | 0.408547 |
| **4540** | 0.058272 | 0.941728 |
| **4541** | 0.958592 | 0.041408 |
| **4542** | 0.598546 | 0.401454 |
| **4543** | 0.972532 | 0.027468 |
| **4544** | 0.587678 | 0.412322 |
| **4545** | 0.238038 | 0.761962 |
| **4546** | 0.096786 | 0.903214 |
| **4547** | 0.187119 | 0.812881 |
| **4548** | 0.255199 | 0.744801 |
| **4549** | 0.379919 | 0.620081 |
| **4550** | 0.876471 | 0.123529 |
| **4551** | 0.691237 | 0.308763 |
| **4552** | 0.252484 | 0.747516 |
| **4553** | 0.399455 | 0.600545 |

4554 rows × 2 columns

*Figure 45:*

```
y_test
```

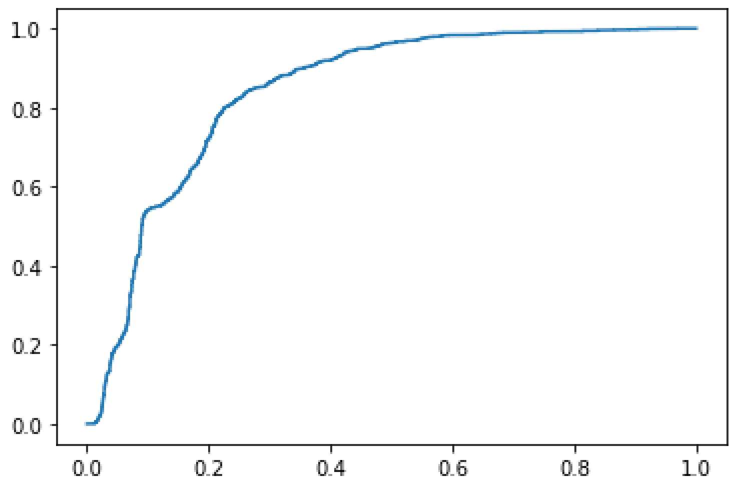
```
11342    0
14859    1
16285    1
16632    1
15100    1
5699     0
19960    1
9345     0
22463    1
3455     0
7853     0
22473    1
7759     0
6420     0
7397     0
11957    1
246      1
16050    1
7364     0
20417    1
8937     0
9248     0
15456    1
153      1
6830     0
565      1
17770    1
16098    1
18950    1
10766    0
        ..
8523     0
9767     0
4492     0
4765     0
7502     0
1967     1
10595    0
203      1
14315    1
20377    1
4901     0
21207    1
6005     0
3419     0
12083    1
21928    1
19092    1
13281    0
4530     0
4173     0
9373     0
863      1
5221     0
21748    1
23       1
16105    1
9761     0
5029     0
21883    1
12338    1
Name: left, Length: 4554, dtype: int64
```

*Figure 46:*

```
from sklearn.metrics import roc_auc_score, roc_curve
left_prob1 = final_model1.predict_proba(X_test)[:,1]
lpr1, tpr1, threshold1 = roc_curve(y_test, left_prob1)
```

```
plt.plot(lpr1, tpr1)
```

```
[<matplotlib.lines.Line2D at 0x6fe3952320>]
```




```
roc_auc_score(y_test, left_prob1)
```

```
0.8424445094646511
```

*Figure 47:* Measuring the accuracy of logistic regression model using the Area under the Precision-Recall Curve (AUPRC).

**Even After Applying Scaling also there is not much change in Accuracy.**

### 4.3.2 Random forest classification

Random forest is a type of supervised machine learning algorithm based on ensemble learning. Ensemble learning is a type of learning where you join different types of algorithms or the same algorithm multiple times to form a more powerful prediction model. The random forest algorithm combines multiple algorithms of the same type i.e. multiple decision trees, resulting in a forest of trees, hence the name "Random Forest". The random forest algorithm can be used for both regression and classification tasks.

The following are the basic steps involved in performing the random forest algorithm:

1. Pick N random records from the dataset.
2. Build a decision tree based on these N records.
3. Choose the number of trees you want in your algorithm and repeat steps 1 and 2.
4. In case of a regression problem, for a new record, each tree in the forest predicts a value for Y (output). The final value can be calculated by taking the average of all the values predicted by all the trees in forest. Or, in case of a classification problem, each tree in the forest predicts the category to which the new record belongs. Finally, the new record is assigned to the category that wins the majority vote.

```
#import initialize and fit
#import the RFC from sklearn
from sklearn.ensemble import RandomForestClassifier

#initialize the object for RFC
rfc = RandomForestClassifier()

#fit RFC to dataset
final_model2 = rfc.fit(X_train,y_train)
```

*Figure 48: Applying random forest classifier on the training data.*

**<u>Predicting on training data</u>**

```
y_train_pred1 = rfc.predict(X_train) #Predicting on training data
```

```
confusion_matrix(y_train,y_train_pred1)
```

```
array([[9104,    0],
       [   0, 9112]], dtype=int64)
```

```
accuracy_score(y_train,y_train_pred1)
```

```
1.0
```

*Figure 49: Prediction and applying the metrics on train data.*

**Predicting on test data**

```
y_test_pred1 = rfc.predict(X_test) #Predicting on test data
```

```
confusion_matrix(y_test,y_test_pred1)
```

```
array([[2268,   13],
       [  39, 2234]], dtype=int64)
```

```
accuracy_score(y_test,y_test_pred1)
```

```
0.9885814668423364
```

*Figure 50: Prediction and applying the metrics on test data.*

```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_train,y_train_pred1))
print("------------------------------------------------------")
print(classification_report(y_test,y_test_pred1))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      9104
           1       1.00      1.00      1.00      9112

    accuracy                           1.00     18216
   macro avg       1.00      1.00      1.00     18216
weighted avg       1.00      1.00      1.00     18216

------------------------------------------------------
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      2281
           1       0.99      0.98      0.99      2273

    accuracy                           0.99      4554
   macro avg       0.99      0.99      0.99      4554
weighted avg       0.99      0.99      0.99      4554
```

*Figure 51: Overall performance of the random forest classifier model based on training and test data.*

```
y_test_prob2 = final_model2.predict_proba(X_test)
y_test_prob2 = pd.DataFrame(y_test_prob2)
y_test_prob2
```

| | 0 | 1 |
|---|---|---|
| 0 | 0.86 | 0.14 |
| 1 | 0.00 | 1.00 |
| 2 | 0.00 | 1.00 |
| 3 | 0.00 | 1.00 |
| 4 | 0.00 | 1.00 |
| 5 | 0.96 | 0.04 |
| 6 | 0.00 | 1.00 |
| 7 | 1.00 | 0.00 |
| 8 | 0.00 | 1.00 |
| 9 | 0.92 | 0.08 |
| 10 | 1.00 | 0.00 |
| 11 | 0.00 | 1.00 |
| 12 | 1.00 | 0.00 |
| 13 | 0.88 | 0.12 |
| 14 | 1.00 | 0.00 |
| 15 | 0.00 | 1.00 |
| 16 | 0.00 | 1.00 |
| 17 | 0.00 | 1.00 |
| 18 | 0.98 | 0.02 |
| 19 | 0.00 | 1.00 |
| 20 | 1.00 | 0.00 |
| 21 | 0.99 | 0.01 |
| 22 | 0.00 | 1.00 |
| 23 | 0.00 | 1.00 |
| 24 | 0.98 | 0.02 |
| 25 | 0.04 | 0.96 |
| 26 | 0.00 | 1.00 |
| 27 | 0.00 | 1.00 |

| ... | ... | ... |
|---|---|---|
| 4524 | 1.00 | 0.00 |
| 4525 | 1.00 | 0.00 |
| 4526 | 1.00 | 0.00 |
| 4527 | 1.00 | 0.00 |
| 4528 | 1.00 | 0.00 |
| 4529 | 0.00 | 1.00 |
| 4530 | 1.00 | 0.00 |
| 4531 | 0.00 | 1.00 |
| 4532 | 0.00 | 1.00 |
| 4533 | 0.05 | 0.95 |
| 4534 | 0.95 | 0.05 |
| 4535 | 0.00 | 1.00 |
| 4536 | 1.00 | 0.00 |
| 4537 | 1.00 | 0.00 |
| 4538 | 0.00 | 1.00 |
| 4539 | 0.29 | 0.71 |
| 4540 | 0.00 | 1.00 |
| 4541 | 1.00 | 0.00 |
| 4542 | 1.00 | 0.00 |
| 4543 | 0.89 | 0.11 |
| 4544 | 0.97 | 0.03 |
| 4545 | 0.00 | 1.00 |
| 4546 | 0.72 | 0.28 |
| 4547 | 0.01 | 0.99 |
| 4548 | 0.00 | 1.00 |
| 4549 | 0.00 | 1.00 |
| 4550 | 1.00 | 0.00 |
| 4551 | 1.00 | 0.00 |
| 4552 | 0.00 | 1.00 |
| 4553 | 0.00 | 1.00 |

4554 rows × 2 columns

*Figure 52:*

```
y_test
```

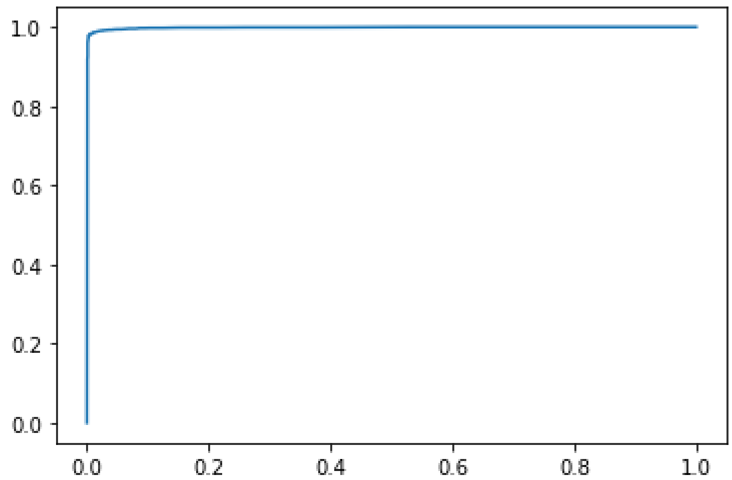
```
11342    0
14859    1
16285    1
16632    1
15100    1
5699     0
19960    1
9345     0
22463    1
3455     0
7053     0
22473    1
7759     0
6420     0
7397     0
11957    1
246      1
16050    1
7364     0
20417    1
8937     0
9248     0
15456    1
153      1
6830     0
565      1
17770    1
16098    1
18950    1
10766    0
        ..
8523     0
9767     0
4492     0
4765     0
7502     0
1967     1
10595    0
203      1
14315    1
20377    1
4901     0
21207    1
6005     0
3419     0
12083    1
21928    1
19092    1
13281    0
4530     0
4173     0
9373     0
863      1
5221     0
21748    1
23       1
16105    1
9761     0
5029     0
21883    1
12338    1
Name: left, Length: 4554, dtype: int64
```

*Figure 53:*

```
from sklearn.metrics import roc_auc_score, roc_curve
left_prob2 = final_model2.predict_proba(X_test)[:,1]
lpr2, tpr2, threshold2 = roc_curve(y_test, left_prob2)
```

```
plt.plot(lpr2, tpr2)
```

```
[<matplotlib.lines.Line2D at 0x12d4056780>]
```




```
roc_auc_score(y_test, left_prob2)
```

```
0.998330476537467
```

*Figure 54: Measuring the accuracy of a random forest classifier model using the Area Under the Precision-Recall Curve (AUPRC).*

### 4.3.3 Naive Bayes

Naive Bayes is the most straightforward and fast classification algorithm, which is suitable for a large chunk of data. Naive Bayes classifier is successfully used in various applications such as spam filtering, text classification, sentiment analysis, and recommender systems. It uses Bayes theorem of probability for prediction of unknown class.

Naive Bayes is a statistical classification technique based on Bayes Theorem. It is one of the simplest supervised learning algorithms. Naive Bayes classifier is the fast, accurate and reliable algorithm. Naive Bayes classifiers have high accuracy and speed on large datasets.

Naive Bayes classifier assumes that the effect of a particular feature in a class is independent of other features. For example, a loan applicant is desirable or not depending on his/her income, previous loan and transaction history, age, and location. Even if these features are interdependent, these features are still considered independently. This assumption simplifies computation, and that's why it is considered as naive. This assumption is called class conditional independence.

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- P(h): the probability of hypothesis h being true (regardless of the data). This is known as the prior probability of h.
- P(D): the probability of the data (regardless of the hypothesis). This is known as the prior probability.
- P(h|D): the probability of hypothesis h given the data D. This is known as posterior probability.
- P(D|h): the probability of data d given that the hypothesis h was true. This is known as posterior probability.

```
from sklearn.naive_bayes import GaussianNB
gn = GaussianNB()
final_model3 = gn.fit(X_train,y_train)
y_train_pred2 = gn.predict(X_train)
```

*Figure 55: Applying naive bayes algorithm  on training data.*

**Predicting on training data**

```
confusion_matrix(y_train,y_train_pred2)
```

```
array([[4004, 5100],
       [ 383, 8729]], dtype=int64)
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_train,y_train_pred2)
```

```
0.6990008783487044
```

*Figure 56: Applying metrics on training data.*

**<u>Predicting on training data</u>**

```
y_test_pred2 = gn.predict(X_test)
```

```
confusion_matrix(y_test,y_test_pred2)
```

```
array([[1012, 1269],
       [  90, 2183]], dtype=int64)
```

```
accuracy_score(y_test,y_test_pred2)
```

```
0.7015810276679841
```

*Figure 57: Applying metrics on test data.*

```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_train,y_train_pred2))
print("-------------------------------------------------------")
print(classification_report(y_test,y_test_pred2))
```

```
              precision    recall  f1-score   support

           0       0.91      0.44      0.59      9104
           1       0.63      0.96      0.76      9112

    accuracy                           0.70     18216
   macro avg       0.77      0.70      0.68     18216
weighted avg       0.77      0.70      0.68     18216

-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.92      0.44      0.60      2281
           1       0.63      0.96      0.76      2273

    accuracy                           0.70      4554
   macro avg       0.78      0.70      0.68      4554
weighted avg       0.78      0.70      0.68      4554
```

*Figure 58: Overall performance of the naive bayes model based on training and test data.*

```
y_test_prob3 = final_model3.predict_proba(X_test)
y_test_prob3 = pd.DataFrame(y_test_prob3)
y_test_prob3
```

| | 0 | 1 |
|---|---|---|
| 0 | 0.055783 | 9.442174e-01 |
| 1 | 0.002256 | 9.977440e-01 |
| 2 | 0.024549 | 9.754510e-01 |
| 3 | 0.000249 | 9.997508e-01 |
| 4 | 0.001576 | 9.984241e-01 |
| 5 | 0.580186 | 4.198137e-01 |
| 6 | 0.491490 | 5.085100e-01 |
| 7 | 0.231119 | 7.688813e-01 |
| 8 | 0.000654 | 9.993456e-01 |
| 9 | 0.999981 | 1.910478e-05 |
| 10 | 0.569948 | 4.300517e-01 |
| 11 | 0.000733 | 9.992671e-01 |
| 12 | 0.233703 | 7.662974e-01 |
| 13 | 0.999997 | 2.786642e-06 |
| 14 | 0.571989 | 4.280115e-01 |
| 15 | 0.021725 | 9.782748e-01 |

| | | |
|---|---|---|
| **4536** | 0.513389 | 4.866108e-01 |
| **4537** | 0.237636 | 7.623636e-01 |
| **4538** | 0.027830 | 9.721698e-01 |
| **4539** | 0.319408 | 6.805923e-01 |
| **4540** | 0.002904 | 9.970958e-01 |
| **4541** | 0.923712 | 7.628782e-02 |
| **4542** | 0.335544 | 6.644558e-01 |
| **4543** | 1.000000 | 3.080457e-07 |
| **4544** | 0.279723 | 7.202767e-01 |
| **4545** | 0.399396 | 6.006036e-01 |
| **4546** | 0.232374 | 7.676256e-01 |
| **4547** | 0.002480 | 9.975198e-01 |
| **4548** | 0.039532 | 9.604685e-01 |
| **4549** | 0.051328 | 9.486718e-01 |
| **4550** | 0.509599 | 4.904015e-01 |
| **4551** | 0.558666 | 4.413337e-01 |
| **4552** | 0.024072 | 9.759281e-01 |
| **4553** | 0.215401 | 7.845992e-01 |

4554 rows × 2 columns

*Figure 59:*

```
y_test
```

```
11342    0
14859    1
16285    1
16632    1
15100    1
5699     0
19960    1
9345     0
22463    1
3455     0
7053     0
22473    1
7759     0
6420     0
7397     0
11957    1
246      1
16050    1
7364     0
20417    1
8937     0
9248     0
15456    1
153      1
6830     0
565      1
17770    1
16098    1
```

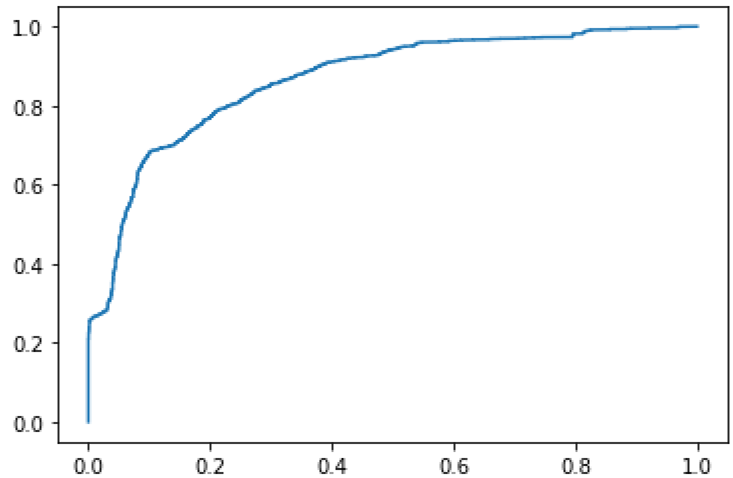
```
9767     0
4492     0
4765     0
7502     0
1967     1
10595    0
203      1
14315    1
20377    1
4901     0
21207    1
6005     0
3419     0
12083    1
21928    1
19092    1
13281    0
4530     0
4173     0
9373     0
863      1
5221     0
21748    1
23       1
16105    1
9761     0
5029     0
21883    1
12338    1
Name: left, Length: 4554, dtype: int64
```

*Figure 60:*

```
# Roc curve
## TPR, LPR, Threshold
from sklearn.metrics import roc_auc_score, roc_curve
left_prob3 = final_model3.predict_proba(X_test)[:,1]
lpr3, tpr3, threshold3 = roc_curve(y_test, left_prob3)
```

```
plt.plot(lpr3, tpr3)
```

```
[<matplotlib.lines.Line2D at 0x12d50a93c8>]
```




```
roc_auc_score(y_test, left_prob3)
```

```
0.86648827042114
```

*Figure 61: Measuring the accuracy of naive bayes model using the Area Under the Precision-Recall Curve (AUPRC).*

## 4.4 Visualising the best model among logistic regression, Random forest and NaiveBayes

```
models = ['Logistic Regression','Random forest','NaiveBayes']
accuracy_scores = [0.78,1.00,0.69]
plt.bar(models,accuracy_scores,color=['lightblue','pink','lightgrey'])
plt.ylabel("accuracy scores")
plt.title("which model has high accuracy")
plt.show()
```

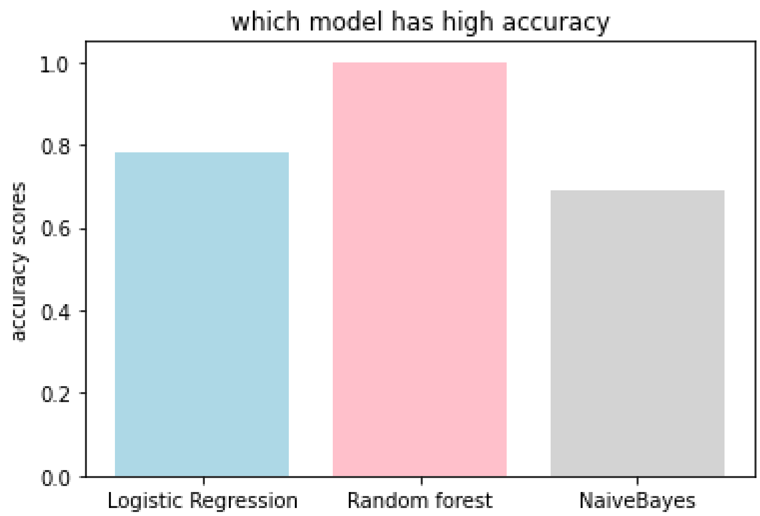



*Figure 62: Comparison of the applied models.*

## 5. Predicting The Model With Unknown Data

```
# Predict Test Set
fav_clf = RandomForestClassifier()
fav_clf.fit(X_train, y_train)
sub = pd.DataFrame(fav_clf.predict(X_test),index=X_test.index,columns={'Prediction'})
sub
```

| | Prediction |
|---|---|
| 11342 | 0 |
| 14859 | 1 |
| 16285 | 1 |
| 16632 | 1 |
| 15100 | 1 |
| 5699 | 0 |
| 19960 | 1 |
| 9345 | 0 |
| 22463 | 1 |
| 3455 | 0 |
| 7053 | 0 |
| 22473 | 1 |
| 7759 | 0 |
| 6420 | 0 |
| 9373 | 0 |
| 863 | 1 |
| 5221 | 0 |
| 21748 | 1 |
| 23 | 1 |
| 16105 | 1 |
| 9761 | 0 |
| 5029 | 0 |
| 21883 | 1 |
| 12338 | 1 |

4554 rows × 1 columns

*Figure 63: Predict The Test Data*

```
# See the prediction result
result = pd.concat([X_test, y_test,sub], axis=1)
print(result.shape)
result[result.left==result.Prediction]
```

```
(4554, 10)
```

| | satisfaction_level | last_evaluation | number_of_projects | average_monthly_hours | years_at_company | work_accident | promotion_last_5years | salary | left | Prediction |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0.160000 | 0.720000 | 3 | 203 | 3 | 0 | 0 | 1 | 0 | 0 |
| | 0.110000 | 0.890000 | 6 | 268 | 4 | 0 | 0 | 1 | 1 | 1 |
| | 0.412527 | 0.470000 | 2 | 135 | 3 | 0 | 0 | 0 | 1 | 1 |
| | 0.100000 | 0.850000 | 7 | 279 | 4 | 0 | 0 | 1 | 1 | 1 |
| | 0.100000 | 0.810000 | 6 | 290 | 4 | 0 | 0 | 1 | 1 | 1 |
| | 0.750000 | 0.690000 | 3 | 274 | 2 | 0 | 0 | 1 | 0 | 0 |
| | 0.859101 | 0.840899 | 4 | 246 | 6 | 0 | 0 | 0 | 1 | 1 |
| | 0.940000 | 0.840000 | 5 | 240 | 3 | 0 | 0 | 0 | 0 | 0 |
| | 0.104302 | 0.907151 | 6 | 287 | 4 | 0 | 0 | 0 | 1 | 1 |
| | 0.140000 | 0.460000 | 2 | 267 | 6 | 1 | 0 | 0 | 0 | 0 |
| | 0.920000 | 0.780000 | 3 | 194 | 3 | 0 | 0 | 1 | 0 | 0 |
| | 0.098049 | 0.881951 | 6 | 282 | 4 | 0 | 0 | 0 | 1 | 1 |
| | 0.980000 | 0.690000 | 3 | 274 | 3 | 0 | 0 | 0 | 0 | 0 |
| | 0.410000 | 0.960000 | 5 | 167 | 3 | 1 | 0 | 0 | 0 | 0 |
| | 0.530000 | 0.510000 | 4 | 174 | 2 | 0 | 0 | 1 | 0 | 0 |
| **21748** | 0.105532 | 0.922234 | 6 | 255 | 4 | 0 | 0 | 1 | 1 | |
| **23** | 0.460000 | 0.570000 | 2 | 139 | 3 | 0 | 0 | 0 | 1 | |
| **16105** | 0.379297 | 0.500703 | 2 | 140 | 3 | 0 | 0 | 1 | 1 | |
| **9761** | 0.800000 | 0.640000 | 5 | 180 | 2 | 0 | 0 | 0 | 0 | |
| **5029** | 0.740000 | 0.630000 | 3 | 180 | 2 | 0 | 0 | 0 | 0 | |
| **21883** | 0.420000 | 0.475176 | 2 | 129 | 3 | 0 | 0 | 0 | 1 | |
| **12338** | 0.820000 | 0.980000 | 4 | 233 | 5 | 0 | 0 | 0 | 1 | |

4493 rows × 10 columns

*Figure 64: Seeing the prediction result*

```
result = pd.concat([X_test, y_test,sub], axis=1)
print(result.shape)
result[result.left!=result.Prediction].count()
```

```
(4554, 10)

satisfaction_level       61
last_evaluation          61
number_of_projects       61
average_monthly_hours    61
years_at_company         61
work_accident            61
promotion_last_5years    61
salary                   61
left                     61
Prediction               61
dtype: int64
```

*Figure 65: prediction of  result count*

```
result = pd.concat([X_test, y_test,sub], axis=1)
print(result.shape)
result[result.left!=result.Prediction].shape[0]
```

```
(4554, 10)

61
```

```
result = pd.concat([X_test, y_test,sub], axis=1)
print(result.shape)
result[result.Prediction==1]
```

```
(4554, 10)
```

*Figure 66: Checking the shape Of  The Result*

```
result = pd.concat([X_test, y_test,sub], axis=1)
print(result.shape)
result[result.Prediction==1]
```

```
(4554, 10)
```

| | satisfaction_level | last_evaluation | number_of_projects | average_monthly_hours | years_at_company | work_accident | promotion_last_5years | salary | left | Prediction |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0.110000 | 0.890000 | 6 | 268 | 4 | 0 | 0 | 1 | 1 | 1 |
| | 0.412527 | 0.470000 | 2 | 135 | 3 | 0 | 0 | 0 | 1 | 1 |
| | 0.100000 | 0.850000 | 7 | 279 | 4 | 0 | 0 | 1 | 1 | 1 |
| | 0.100000 | 0.810000 | 6 | 290 | 4 | 0 | 0 | 1 | 1 | 1 |
| | 0.859101 | 0.840899 | 4 | 246 | 6 | 0 | 0 | 0 | 1 | 1 |
| | 0.104302 | 0.907151 | 6 | 287 | 4 | 0 | 0 | 0 | 1 | 1 |
| | 0.098049 | 0.881951 | 6 | 282 | 4 | 0 | 0 | 0 | 1 | 1 |
| | 0.410000 | 0.460000 | 2 | 128 | 3 | 0 | 0 | 0 | 1 | 1 |
| | 0.370000 | 0.480000 | 2 | 159 | 3 | 0 | 0 | 0 | 1 | 1 |
| | 0.883897 | 0.975931 | 5 | 273 | 5 | 0 | 0 | 1 | 1 | 1 |
| | 0.867567 | 0.953154 | 4 | 225 | 5 | 0 | 0 | 0 | 1 | 1 |
| | 0.107821 | 0.843971 | 6 | 270 | 4 | 0 | 0 | 0 | 1 | 1 |
| | 0.400000 | 0.520000 | 2 | 155 | 3 | 0 | 0 | 0 | 1 | 1 |
| | 0.790000 | 0.580000 | 3 | 294 | 4 | 0 | 0 | 0 | 1 | 1 |
| | 0.094410 | 0.842048 | 6 | 250 | 4 | 0 | 0 | 0 | 1 | 1 |
| **12083** | 0.390000 | 0.480000 | 2 | 160 | 3 | 0 | 0 | 0 | 1 | |
| **21928** | 0.710000 | 0.870000 | 3 | 177 | 4 | 0 | 0 | 1 | 1 | |
| **19092** | 0.110000 | 0.810000 | 5 | 287 | 4 | 0 | 0 | 0 | 1 | |
| **863** | 0.740000 | 0.880000 | 4 | 248 | 6 | 0 | 0 | 0 | 1 | |
| **21748** | 0.105532 | 0.922234 | 6 | 255 | 4 | 0 | 0 | 1 | 1 | |
| **23** | 0.460000 | 0.570000 | 2 | 139 | 3 | 0 | 0 | 0 | 1 | |
| **16105** | 0.379297 | 0.500703 | 2 | 140 | 3 | 0 | 0 | 1 | 1 | |
| **21883** | 0.420000 | 0.475176 | 2 | 129 | 3 | 0 | 0 | 0 | 1 | |
| **12338** | 0.820000 | 0.980000 | 4 | 233 | 5 | 0 | 0 | 0 | 1 | |

2246 rows × 10 columns

*Figure 67: Checking the result of Prediction*

# 6. Conclusion

It is concluded after performing thorough Exploratory Data analysis which include Stats models which are computed to get accuracy and also Heat maps which are computed to get a clear understanding of the data set (which parameter has most abundant effect on the study case).From the above model building and evaluation we can predict that random forest classifier is best for predicting the fraud and normal transaction

# 7. References


- https://en.wikipedia.org/wiki/Machine_learning
- https://towardsdatascience.com/supervised-machine-learning-model-validation-a-step-by-step-approach-771109ae0253
- https://builtin.com/data-science/random-forest-algorithm