# Lead_Scoring

May 16, 2023

## 1 Build lead scoring model for B2C sales

## 2 Introduction

A company that specializes in B2C sales (product: Data Science and Data Engineering related courses) is struggling to identify and prioritize high-quality leads efficiently. Currently, sales representatives are responsible for manually qualifying and scoring leads based on limited information, which often results in wasted time and resources on low-quality leads. This project aims to develop a lead scoring model using machine learning algorithms that can accurately predict the likelihood of a lead converting into a paying customer. This model will enable the sales team to prioritize high-quality leads and improve the efficiency of the sales process.

### 2.1 What is Lead Scoring?

Lead scoring is a methodology used by businesses to rank and prioritize their leads based on their level of interest and potential for conversion into customers. The purpose of lead scoring is to identify and focus on the most promising leads, allowing sales and marketing teams to allocate their time and resources more efficiently.

Lead scoring involves assigning a numerical value or score to each lead based on certain criteria and behaviors. These criteria typically include demographic information, such as job title and company size, as well as behavioral data, such as website visits, email engagement, and social media interactions. The specific criteria and their corresponding weights may vary depending on the business and its target audience.

#### 2.1.1 Business Impact of Lead Scoring

**Improve effectiveness of campaigns**: Lead scoring can enhance sales and marketing effectiveness by allowing teams to focus their efforts on leads with the highest potential, resulting in better lead conversion rates, increased revenue, and improved overall efficiency.

**Increase product conversions**: Lead scoring leads to righ user targeting through right channels/assets which leads to better conversions.

**Increase revenue**: Increased conversions(as mentioned in point above) will lead to more revenue or buyer engagement. For example, if the company is able to target a user who is more active on Instagram, chances are more that he/she will click on the Ad and add the product to cart. So overall probability of an order increases and hence the revenue.

**Cost Efficiency**: Lead scoring helps optimize resource allocation by focusing on leads that are more likely to convert. By avoiding wasteful spending on low-quality leads or unproductive marketing efforts, businesses can reduce costs and maximize their return on investment.

#### 2.1.2 Why Data Science?

Companies do digital and offline marketing through various channels such as social media, email, and search advertising. They have a fixed budget which they can spend across these channels to get more sales, acquire new customers or regain old customers.

Often they have vast majority of data on how efficient each of these channels have been for them and how much sales it is driving. If you do a simple vanilla analysis of channels vs revenue correlation you could get meaningful insights. Now imagine if a model is supplied with such rich and crucial data?

Data science enables these companies to build a model which can help them understand the likelihood of a conversion or in other words a customer buying their products if the latter is targeted through a specific channel. Model understands the historical nuances and builds the internal model parameters so as to tell the team where they should focus on spending their marketing budget!

## 3 Possible Solutions

### 3.1 Methods for Lead Scoring

There are various ways to build a model for lead scoring, and the choice depends on factors such as the data available, the complexity of the system, and the computational resources available. Some approaches are:

#### 3.1.1 1. Manual scoring:

This approach involves a subjective evaluation of leads by sales or marketing teams based on predefined criteria. Leads are manually assigned scores or labels based on their fit, interest level, and engagement. Although this method can be simple and cost-effective, it is prone to subjectivity and may lack consistency.

#### 3.1.2 2. Rule-based systems:

Rule-based lead scoring involves the creation of predefined rules and thresholds for assigning scores to leads based on specific criteria and behaviors. These rules can be based on demographic information (e.g., job title, company size) and engagement data (e.g., email opens, website visits). This method offers more consistency than manual scoring but may lack the flexibility and adaptability of automated approaches.

#### 3.1.3 3. Linear optimizatiom:

In this approach, we define the objective function(e.g. increase conversion, reduce total cost) and the constraints to come up with a linear equation. The linear equation is usually solved using a solver such as Gurobi, Pulp, etc. for the right solution which gives the confidence around the lead the company should pursue.

#### 3.1.4 4. Machine learning:

Machine learning systems combines strength of historical data and statistical techniques to explain the right lead for individual customers of the company. For example, a ML system can tell with a high confidence if a potentail customer will click on the ad or not.

#### 3.1.5 Assumptions

- We assume that Interest Level is our target variable which refers to the interest of a user for a lead id
- We assume that whenever the target variable is NA or Not called, the corresponding lead id is not meaningful and hence dropped
- If the model's prediction is very close to 1, it means that the user is very likely to engage with the lead id
- Columns with a lot of null values are not meaningful and imputation also won't be helpful

### 3.2 Approach

We are treating this problem as a supervised learning problem. So every data point will have a target variable for the model to learn the dependencies and predict on the unknown.

In real life, this model would tell the business whether a user is likely to engage with the ad or not and that would in turn help the company to pursue the lead.

Given our assumptions about the data, we will build a prediction model based on the historical data. Simplifying, here's the logic of what we'll build:

1. We'll build a model to identify if a customer will be interested in the lead;
2. We'll use various models and compare their performance on interest prediction;
3. We will then choose the most successful model to use in production;

**Supervised Machine Learning:**

In supervised machine learning, the algorithm is trained on an labeled dataset with a predefined target variable. The goal is to identify patterns, relationships, and structures of the data with the target variable, such as logistic regression, decision tree or boosting trees

## 4 Learning Outcomes

- How to load data from csv using pandas
- Exploratory Data Analysis
- Categorical Feature Encoding using Labelencoder
- Data Preprocessing and Feature Engineering
- Understanding missing values
- How do you define right target variable?
- How to build various supervised learning models?
- What are PR and AUC curves? And how do they help in chosing the right threshold?

### 4.1 Prerequisites

- Familiarity with Python programming language

- Familiarity with Pandas, sklearn, numpy libraries in Python, along with concepts like loops, lists, arrays and dataframe
- Basic knowledge of machine learning concepts such as supervised learning, tree models
- Understanding of data preprocessing techniques such as handling missing values, outliers, and categorical variables
- Knowledge of Jupyter Notebook or any other Python IDE.
- Understanding metrics such as precision, recall, PR curve, AUC curve

## 5 Package Requirements

[1]:
```
!pip install numpy==1.19.5
!pip install pandas==1.2.4
!pip install scikit-learn==0.23.2
!pip install xgboost==1.7.4
!pip install lightgbm==3.3.2
```

```
Looking in indexes: https://pypi.org/simple, https://udaan_reader:****@pypi-
read.ds.udaan.io/simple/
Requirement already satisfied: numpy==1.19.5 in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (1.19.5)
Looking in indexes: https://pypi.org/simple, https://udaan_reader:****@pypi-
read.ds.udaan.io/simple/
Requirement already satisfied: pandas==1.2.4 in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (1.2.4)
Requirement already satisfied: python-dateutil>=2.7.3 in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (from
pandas==1.2.4) (2.8.1)
Requirement already satisfied: pytz>=2017.3 in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (from
pandas==1.2.4) (2021.1)
Requirement already satisfied: numpy>=1.16.5 in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (from
pandas==1.2.4) (1.19.5)
Requirement already satisfied: six>=1.5 in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (from python-
dateutil>=2.7.3->pandas==1.2.4) (1.15.0)
Looking in indexes: https://pypi.org/simple, https://udaan_reader:****@pypi-
read.ds.udaan.io/simple/
Requirement already satisfied: scikit-learn==0.23.2 in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (0.23.2)
Requirement already satisfied: numpy>=1.13.3 in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (from scikit-
learn==0.23.2) (1.19.5)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (from scikit-
learn==0.23.2) (2.1.0)
Requirement already satisfied: scipy>=0.19.1 in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (from scikit-
learn==0.23.2) (1.5.4)
Requirement already satisfied: joblib>=0.11 in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (from scikit-
learn==0.23.2) (1.0.1)
Looking in indexes: https://pypi.org/simple, https://udaan_reader:****@pypi-
read.ds.udaan.io/simple/
Requirement already satisfied: xgboost==1.7.4 in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (1.7.4)
Requirement already satisfied: numpy in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (from
xgboost==1.7.4) (1.19.5)
Requirement already satisfied: scipy in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (from
xgboost==1.7.4) (1.5.4)
Looking in indexes: https://pypi.org/simple, https://udaan_reader:****@pypi-
read.ds.udaan.io/simple/
Requirement already satisfied: lightgbm==3.3.2 in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (3.3.2)
Requirement already satisfied: scipy in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (from
lightgbm==3.3.2) (1.5.4)
Requirement already satisfied: wheel in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (from
lightgbm==3.3.2) (0.37.1)
Requirement already satisfied: numpy in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (from
lightgbm==3.3.2) (1.19.5)
Requirement already satisfied: scikit-learn!=0.22.0 in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (from
lightgbm==3.3.2) (0.23.2)
Requirement already satisfied: joblib>=0.11 in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (from scikit-
learn!=0.22.0->lightgbm==3.3.2) (1.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/Users/pranjal.singh/opt/anaconda3/lib/python3.8/site-packages (from scikit-
learn!=0.22.0->lightgbm==3.3.2) (2.1.0)
```

[2]:
```python
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import preprocessing
# Suppress all warnings
import warnings
warnings.filterwarnings("ignore")
```

```
[3]: pd.set_option('display.max_columns', 200)
```

Its the maximum number of columns displayed when a frame is pretty-printed. By setting this limit we can see 200 columns at once without truncation.

## 6 The Data

```
[6]: csv_file_path = "Marketing Data.csv"
     df = pd.read_csv(csv_file_path)
```

Lets look at the first 10 records from the dataframe.

```
[7]: df.head(10)
```

```
[7]:                           Lead Id Lead Owner       Interest Level  \
0  5e502dcf828b8975a78e89f3e9aeac12     e14c3a       Not Interested
1  efe3f074c61959c2ea1906dd0346aa69     d16267  Slightly Interested
2  d26dc5cd5843622a203cf396b4ee4b1a     d138f9            No Answer
3  d50acaedc1e5b9c18f8ceb3c6cff345b     38e2a6       Not Interested
4  07758f3d12a23e68bb3b58b8009dd9a8     d130bb       Not Interested
5  665eb8f7c975b055afa58b5dda3a78bc     d5b5bd  Slightly Interested
6  a1ea99cba3b88f6c59fea8a84f051dec     d16267            No Answer
7  e69523450132baed2dd72836cdfc9778     d130bb       Not Interested
8  fe244887bc37b5f49311c750ce6b279f     d138f9            No Answer
9  3500a29dc4849a7166e98db2e44ddc53     38e2a6            No Answer

       Lead created Lead Location(Auto) Creation Source     Next activity  \
0  12-01-2023 16:42                  IN             API               NaN
1  04-12-2021 09:32                 NaN             API  12-01-2022 00:00
2  15-04-2022 10:16                 NaN             API  16-04-2022 00:00
3  21-10-2022 17:02                  IN             API  23-10-2022 00:00
4  25-10-2021 10:48                 NaN             API               NaN
5  24-11-2022 22:41                 NaN             API  26-11-2022 00:00
6  07-07-2022 14:50                 NaN             API               NaN
7  16-09-2021 23:37                 NaN             API               NaN
8  08-06-2022 13:30                 NaN             API  24-06-2022 00:00
9  21-10-2022 23:50                  IN             API  23-10-2022 00:00

  What do you do currently ? What are you looking for in Product ?  \
0                    Student                                   NaN
1                        NaN                                   NaN
2                        NaN                                   NaN
3                    fresher                                   NaN
4                        NaN                                   NaN
5                        NaN                  Big Data engineering
6                        NaN                                   NaN
7         Glass maker at home                                  NaN
```

```
8                       NaN                                    NaN
9                       NaN                                    NaN

  Website Source Lead Last Update time  Marketing Source  \
0            NaN      12-01-2023 19:27               NaN
1     Sales lead      12-01-2022 17:17  Paid - Instagram
2            NaN      16-04-2022 20:35      Paid-Adwords
3            NaN      02-12-2022 13:35      Paid-Adwords
4     Sales lead      13-11-2021 14:51         Affiliate
5            NaN      26-11-2022 19:49               NaN
6     Sales lead      08-07-2022 18:20            Medium
7     Sales lead      12-11-2021 04:49   Paid - Facebook
8            NaN      24-06-2022 10:44  Paid - Instagram
9            NaN      23-10-2022 11:09  Paid - Instagram

  Lead Location(Manual)         Demo Date Demo Status Closure date
0                 India               NaN         NaN          NaN
1                 India  05-12-2021 00:00     No Show          NaN
2                    In               NaN         NaN          NaN
3                    IN  22-11-2022 00:00   Scheduled          NaN
4                 India               NaN         NaN          NaN
5                    TR               NaN         NaN          NaN
6                 India               NaN         NaN          NaN
7                 India               NaN         NaN          NaN
8                    In               NaN         NaN          NaN
9                   NaN               NaN         NaN          NaN
```

If we look at the dataframe, we notice that there is an ID column and there are multiple columns with NA values.

What we will do here is: 1. Look for other such columns which don't serve value; 2. Treat columns having missing values either by imputation or by dropping them;

## 7 Exploratory Data Analysis

### 7.1 Data Exploration

Data exploration is a critical step in the data analysis process, where you examine the dataset to gain a preliminary understanding of the data, detect patterns, and identify potential issues that may need further investigation. Data exploration is important because it helps to provide a solid foundation for subsequent data analysis tasks, hypothesis testing and data visualization.

Data exploration is also important because it can help you to identify an appropriate approach for analyzing the data.

Here are the various functions that help us explore and understand the data.

- Shape: Shape is used to identify the dimensions of the dataset. It gives the number of rows and columns present in the dataset. Knowing the dimensions of the dataset is important to understand the amount of data available for analysis and to determine the feasibility of different methods of analysis.

- Head: The head function is used to display the top five rows of the dataset. It helps us to understand the structure and organization of the dataset. This function gives an idea of what data is present in the dataset, what the column headers are, and how the data is organized.
- Tail: The tail function is used to display the bottom five rows of the dataset. It provides the same information as the head function but for the bottom rows. The tail function is particularly useful when dealing with large datasets, as it can be time-consuming to scroll through all the rows.
- Describe: The describe function provides a summary of the numerical columns in the dataset. It includes the count, mean, standard deviation, minimum, and maximum values, as well as the quartiles. It helps to understand the distribution of the data, the presence of any outliers, and potential issues that can affect the model's accuracy.
- Isnull: The isnull function is used to identify missing values in the dataset. It returns a Boolean value for each cell, indicating whether it is null or not. This function is useful to identify the presence of missing data, which can be problematic for regression analysis.
- Dropna: The dropna function is used to remove rows or columns with missing data. It is used to remove any observations or variables with missing data, which can lead to biased results in the regression analysis. The dropna function is used after identifying the missing data with the isnull function.
- Columns: The .columns method is a built-in function that is used to display the column names of a pandas DataFrame or Series. It returns an array-like object that contains the names of the columns in the order in which they appear in the original DataFrame or Series. It can be used to obtain a quick overview of the variables in a dataset and their names.

### 7.1.1 What can we learn from the data?

[8]:
```
df.shape
```

[8]:
```
(38984, 16)
```

[9]:
```
# Checking the names of the columns
df.columns
```

[9]:
```
Index(['Lead Id', 'Lead Owner', 'Interest Level', 'Lead created',
       'Lead Location(Auto)', 'Creation Source', 'Next activity',
       'What do you do currently ?', 'What are you looking for in Product ?',
       'Website Source', 'Lead Last Update time', 'Marketing Source',
       'Lead Location(Manual)', 'Demo Date', 'Demo Status', 'Closure date'],
      dtype='object')
```

## 7.2 Data Dictionary

| Column name | Description |
|---|---|
| Lead Id | Unique Identifier |
| Lead Owner | Internal sales person associated with the lead |
| Interest Level | What is lead's interest level? (entered manually) |
| Lead created | Lead creation date |
| Lead Location(Auto) | Automatically detected location |
| Creation Source | Creation source of the lead |
| Next activity | Date for Next Activity |
| What do you do currently ? | Current profile of lead |
| What are you looking for in Product ? | Specific requirement from product |
| Website Source | Website Source of the Lead |
| Lead Last Update time | Last update time for Lead |
| Marketing Source | Marketing Source of the Lead |
| Lead Location(Manual) | Manually entered lead location |
| Demo Date | Date for Demo |
| Demo Status | Status of demo booked with lead |
| Closure date | Lead closing date |

[10]:
```
# Check the Information of the Dataframe, number of unique values and frequency
df.describe()
```

[10]:
```
                                 Lead Id Lead Owner       Interest Level  \
count                              38984      38984                38847
unique                             37450         23                    8
top     bcbcf737090f0a52c59237fb0ee921d5     2f6f7f  Slightly Interested
freq                                   6       5643                14572

             Lead created Lead Location(Auto) Creation Source  \
count                38984               10810           38984
unique               35951                 169               3
top     13-01-2022 14:05                    IN             API
freq                    17                6735           36291

            Next activity What do you do currently ?  \
count               14776                      16909
unique               2610                       6831
top     31-01-2023 00:00                    Student
freq                   74                       3406

       What are you looking for in Product ? Website Source  \
count                                   9970          24088
unique                                  4046             10
top                                       DS     Sales lead
freq                                     481          23121

       Lead Last Update time Marketing Source Lead Location(Manual)  \
count                  38984            28339               34974
unique                 32693               46                 415
top         06-03-2023 17:53              SEO                  IN
freq                     401            10127               14126

              Demo Date Demo Status      Closure date
count             10851       11423               629
unique              583           3               277
top    26-02-2022 00:00   Scheduled  01-05-2022 00:00
freq                 48        4000                 9
```

[11]:
```
# Check the Information of the Dataframe, datatypes and non-null counts
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38984 entries, 0 to 38983
Data columns (total 16 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Lead Id                               38984 non-null  object
 1   Lead Owner                            38984 non-null  object
 2   Interest Level                        38847 non-null  object
 3   Lead created                          38984 non-null  object
 4   Lead Location(Auto)                   10810 non-null  object
 5   Creation Source                       38984 non-null  object
 6   Next activity                         14776 non-null  object
 7   What do you do currently ?            16909 non-null  object
 8   What are you looking for in Product ? 9970 non-null   object
 9   Website Source                        24088 non-null  object
 10  Lead Last Update time                 38984 non-null  object
 11  Marketing Source                      28339 non-null  object
 12  Lead Location(Manual)                 34974 non-null  object
 13  Demo Date                             10851 non-null  object
 14  Demo Status                           11423 non-null  object
 15  Closure date                          629 non-null    object
dtypes: object(16)
memory usage: 4.8+ MB
```

**Observation:** * we can see some null values present in this data. We will treat them later * Lead created, Next activity, Lead Last Update time and Demo Date should be datetime datatype but it is object

[12]:
```
df['Lead created'] = pd.to_datetime(df['Lead created'], format="%d-%m-%Y %H:%M")
df['Lead Last Update time'] = pd.to_datetime(df['Lead Last Update time'],␣
 ↪format="%d-%m-%Y %H:%M")
df['Next activity'] = pd.to_datetime(df['Next activity'], format="%d-%m-%Y %H:
 ↪%M")
df['Demo Date'] = pd.to_datetime(df['Demo Date'], format="%d-%m-%Y %H:%M")
```

Lets see how many different Lead Owners we have

```
[13]: df['Lead Owner'].unique()
```

```
[13]: array(['e14c3a', 'd16267', 'd138f9', '38e2a6', 'd130bb', 'd5b5bd',
             '949886', 'fc348d', 'c18c01', '1eafbe', '2f6f7f', '5fe006',
             '8a10c8', '1a9b5d', 'c5837c', '64c0b2', '684149', '154755',
             'b89cfd', '8c20b0', '2c7db1', '65ed8c', '64347b'], dtype=object)
```

```
[14]: df['Lead Owner'].value_counts()
```

```
[14]: 2f6f7f    5643
      d16267    5313
      1eafbe    5226
      d5b5bd    4417
      fc348d    3405
      1a9b5d    2515
      d138f9    2023
      e14c3a    1695
      c5837c    1415
      d130bb    1195
      b89cfd    1120
      949886     986
      8c20b0     667
      38e2a6     594
      684149     559
      c18c01     526
      5fe006     525
      8a10c8     509
      64c0b2     317
      2c7db1     288
      154755      34
      65ed8c      10
      64347b       2
      Name: Lead Owner, dtype: int64
```

**Observation** * The data seems to be evenly distributed amongst lead owners

```
[15]: df['Interest Level'].unique()
```

```
[15]: array(['Not Interested', 'Slightly Interested', 'No Answer', 'Closed',
             'Not called', 'Invalid Number', 'Fairly Interested', nan,
             'Very Interested'], dtype=object)
```

```
[16]: df['Interest Level'].value_counts()
```

```
[16]: Slightly Interested     14572
      Not Interested          10545
```

```
No Answer              9254
Not called             1585
Fairly Interested      1320
Closed                  811
Invalid Number          636
Very Interested         124
Name: Interest Level, dtype: int64
```

**Observation** * We see that some of the interest levels are similar semantically * Since interest level is our target variable, it seems to be nicely distributed

**Points to ponder upon** * Should we formulate the problem as multi-class or binary classification problem? * In case, we want to do binary classification, how do we deal with many values in our target variable?

[17]:
```
df['What do you do currently ?'].value_counts()
```

[17]:
```
Student
3406
student
1282
Fresher
298
Working
194
Working pro
148
   ...
Final year MCA
1
Dhanashree Pawar
1
Azure enginneer
1
Studying - Final year engineering
1
Working professional purchase dept - cost planner - // mechanical engineer 2017
1
Name: What do you do currently ?, Length: 6831, dtype: int64
```

[18]:
```
df['What do you do currently ?'].unique().shape
```

[18]:
```
(6832,)
```

**Observation** * There are many unique values in the above column * If we process the string we can reduce these

**Think about it** * Do we need to focus on so many values and confuse the model? * What can we do to reduce these?

```
[19]: df['Creation Source'].unique()
```

```
[19]: array(['API', 'Manually created', 'Deal'], dtype=object)
```

```
[20]: df['Creation Source'].value_counts()
```

```
[20]: API                 36291
      Manually created     2533
      Deal                  160
      Name: Creation Source, dtype: int64
```

**Observation** * This feature looks well balanced in terms of unique values

```
[21]: df['What are you looking for in Product ?'].unique().shape
```

```
[21]: (4047,)
```

```
[22]: df['What are you looking for in Product ?'].value_counts()
```

```
[22]: DS                                                481
      ML                                                325
      DS projects                                       254
      ML projects                                       221
      BD                                                158
                                                       ...
      ML / DS / Cloud computing                           1
      ML n DL projects                                    1
      2nd year student:::Wants to pursue ds               1
      Projects in ML ops and with Hypothesis testing      1
      Big data spark                                      1
      Name: What are you looking for in Product ?, Length: 4046, dtype: int64
```

**Observation** * This feature has many unqiue values and processing it will take a lot of time

**Think about it** * Should we still work with this column? * If yes, what can we do?

```
[23]: df['Website Source'].unique()
```

```
[23]: array([nan, 'Sales lead', 'Start Project', 'Demo button lead',
             'Chat lead', 'Cashback lead', 'eBook',
             'Demo button lead, Chat lead', 'Sales lead, Demo button lead',
             'Sales lead, Chat lead', 'Sales lead, eBook'], dtype=object)
```

```
[24]: df['Website Source'].value_counts()
```

```
[24]: Sales lead                     23121
      Start Project                    560
      Demo button lead                 267
      Chat lead                        114
```

```
Cashback lead                    10
eBook                             5
Sales lead, Demo button lead      5
Sales lead, Chat lead             3
Sales lead, eBook                 2
Demo button lead, Chat lead       1
Name: Website Source, dtype: int64
```

**Observation** * Column has very less variance in terms of frequency * Most of the values are concentrated around 1 or 2 enums

**Think about it** * Almost no variance in data, can model learn something important?

[25]:
```
df['Marketing Source'].value_counts()
```

[25]:
```
SEO                                              10127
Paid - Instagram                                  3895
Paid-Adwords                                      3514
Paid-YouTube                                      2652
Affiliate                                         2531
Medium                                            2215
Paid - Facebook                                   1528
Email Campaign                                    1050
Paid - Linkedin                                    154
Naukri                                             102
Medium, Paid-Adwords                                70
SEO, Medium, Paid-Adwords                           57
Referral                                            50
SEO, Affiliate                                      48
Linkedin jobs                                       46
SEO, Paid-Adwords                                   44
SEO, Paid - Instagram                               34
Affiliate, Medium                                   29
SEO, Medium                                         26
Paid - Instagram, Paid-Adwords                      24
Paid-Adwords, Paid-YouTube                          20
SEO, Paid-YouTube                                   16
SEO, Paid - Facebook                                15
Affiliate, Paid-Adwords                             14
SEO, Paid - Instagram, Paid-Adwords                 11
SEO, Linkedin jobs                                   8
Paid - Facebook, Paid-Adwords                        8
SEO, Paid - Facebook, Paid-Adwords                   7
SEO, Naukri                                          6
SEO, Paid - Linkedin                                 6
SEO, Paid - Instagram, Medium                        4
Affiliate, Email Campaign                            4
Paid - Linkedin, Paid-Adwords                        3
Paid - Instagram, Medium                        3
Affiliate, Paid-YouTube                         3
Paid - Facebook, Paid - Instagram               2
Medium, Paid-YouTube                            2
SEO, Email Campaign                             2
Paid-Adwords, Email Campaign                    2
SEO, Paid - Facebook, Medium                    1
SEO, Affiliate, Medium                          1
Paid - Linkedin, Affiliate                      1
SEO, Paid - Instagram, Medium, Paid-Adwords     1
Paid-YouTube, Email Campaign                    1
Paid - Linkedin, Medium                         1
Medium, Paid-Adwords, Paid-YouTube              1
Name: Marketing Source, dtype: int64
```

**Observation** * There is a long tail of values * The 1st half looks really interesting in terms of distribution

[26]:
```
df['Demo Status'].value_counts()
```

[26]:
```
Scheduled     4000
Done          3956
No Show       3467
Name: Demo Status, dtype: int64
```

**Observation** * The column is nicely dsitributed * Has very less unique values

**Think about it** * If we use this column, are we doing a feature leak?

[27]:
```
df['Lead Location(Manual)'].value_counts(normalize=1)
```

[27]:
```
IN                         0.403900
India                      0.322354
In                         0.059444
US                         0.046635
in                         0.018128
                             ...
USA / Netherlands numb     0.000029
Maharashtra, India         0.000029
United KIngdom             0.000029
ON                         0.000029
Indiia                     0.000029
Name: Lead Location(Manual), Length: 415, dtype: float64
```

**Observation** * This feature again has a very long tail

**Think about it** * What if we just create 2 enums; India and Non India

# 8 Data Processing & Feature engineering

### 8.0.1 Data Preprocessing and Leakage

Data leakage is a situation where information from the test or prediction data is inadvertently used during the training process of a machine learning model. This can occur when information from the test or prediction data is leaked into the training data, and the model uses this information to improve its performance during the training process.

Data leakage can occur during the preprocessing phase of machine learning when information from the test or prediction data is used to preprocess the training data, inadvertently leaking information from the test or prediction data into the training data.

For example, consider a scenario where the preprocessing step involves imputing missing values in the dataset. If the missing values are imputed using the mean or median values of the entire dataset, including the test and prediction data, then the imputed values in the training data may be influenced by the values in the test and prediction data. This can lead to data leakage, as the model may learn to recognize patterns in the test and prediction data during the training process, leading to overfitting and poor generalization performance.

To avoid data leakage, it's important to perform the data preprocessing steps on the training data only, and then apply the same preprocessing steps to the test and prediction data separately. This ensures that the test and prediction data remain unseen by the model during the training process, and helps to prevent overfitting and improve the accuracy of the model.

In the context of this problem, we will perform data preprocessing steps together for the sake of simplicity, which could potentially lead to data leakage. However, in real-world scenarios, it's important to treat the test and prediction data separately and apply the necessary preprocessing steps separately, based on the characteristics of the data.

### 8.0.2 Missing Value Detection and Imputation

Real world datasets are never friendly to data scientists. They always pose great challenges to those who are dealing with them due to many different reasons and one of them is "missing values"

Missing values can be imputed with a provided constant value, or using the statistics (mean, median or most frequent) of each column in which the missing values are located

We previously saw there are some missing values in the data. Lets have a look into that now.

[31]:
```
# Lead Owner
```

[32]:
```
df['Lead Owner'].isna().sum()
```

[32]:
```
0
```

[33]:
```
# Interest Level
```

[34]:
```
df['Interest Level'].isna().sum()
```

[34]:
```
137
```

Since target variable has missing values, we will drop such rows

[35]:
```
df = df[df['Interest Level'].notna()]
```

[36]:
```
df['Interest Level'].value_counts()
```

[36]:
```
Slightly Interested    14572
Not Interested         10545
No Answer               9254
Not called              1585
Fairly Interested       1320
Closed                   811
Invalid Number           636
Very Interested          124
Name: Interest Level, dtype: int64
```

With this mean, we will fill the NaN values.

**Now we will handle our target variable** Since there are multiple values in target variable and we want to formulate our problem as a binary classification problem, we will do the following assignments

**Label assignment:** * Slightly Interested = 1 * Not Interested=0 * No Answer=0 * Fairly Interested=1 * Very Interested=1

- we will drop rows where value is Not called, Closed or Invalid Number

[37]:
```
df = df[~df['Interest Level'].isin(["Not called", "Closed", "Invalid Number"])]
```

[38]:
```
df['Interest Level'].value_counts()
```

[38]:
```
Slightly Interested    14572
Not Interested         10545
No Answer               9254
Fairly Interested       1320
Very Interested          124
Name: Interest Level, dtype: int64
```

[39]:
```
df['Interest Level'] = df['Interest Level'].apply(lambda x: 1 if x in ["Slightly␣
 ↪Interested", "Fairly Interested", "Very Interested"] else 0)
```

[40]:
```
df['Interest Level'].value_counts()
```

[40]:
```
0    19799
1    16016
Name: Interest Level, dtype: int64
```

**Drop not imporant columns**

```
[41]: df.columns
```

```
[41]: Index(['Lead Id', 'Lead Owner', 'Interest Level', 'Lead created',
             'Lead Location(Auto)', 'Creation Source', 'Next activity',
             'What do you do currently ?', 'What are you looking for in Product ?',
             'Website Source', 'Lead Last Update time', 'Marketing Source',
             'Lead Location(Manual)', 'Demo Date', 'Demo Status', 'Closure date'],
            dtype='object')
```

```
[42]: df = df.drop(["Lead Id", "Lead Location(Auto)", "Next activity", "What are you␣
       ↪looking for in Product ?",
                     "Lead Last Update time", "Lead Location(Manual)", "Demo Date",␣
       ↪"Demo Status", "Closure date"], axis=1)
```

### Lead creation time

- We will create 2 features here from our lead creation time column
    1. hour of day
    2. day of week

```
[43]: df['hour_of_day'] = df['Lead created'].dt.hour
      df['day_of_week'] = df['Lead created'].dt.weekday
```

```
[44]: df = df.drop(["Lead created"], axis=1)
```

Now lead created column is not useful and we drop it

### Creation source

```
[45]: df['Creation Source'].value_counts()
```

```
[45]: API                   33317
      Manually created       2346
      Deal                    152
      Name: Creation Source, dtype: int64
```

```
[46]: from pandas import factorize
```

```
[47]: labels, categories = factorize(df["Creation Source"])
```

```
[48]: df["labels"] = labels
      abs(df["Interest Level"].corr(df["labels"]))
```

```
[48]: 0.008490292073158531
```

There is a positive correlation with the target variable

```
[49]: df = df.drop(["labels"], axis=1)
```

**What do you do currently?**

- student = 1
- others = 0

As we saw earlier, this feature has a large number of values of which students are a dominating part.

We will binarize this column into students and non-students

**Facts**

Binarization is the process of dividing data into two groups and assigning one out. of two values to all the members of the same group. This is usually accomplished. by defining a threshold t and assigning the value 0 to all the data points below. the threshold and 1 to those above it.

```
[50]: df['What do you do currently ?'].isna().sum()
```

```
[50]: 19419
```

```
[51]: df['What do you do currently ?'].value_counts(normalize=1)
```

```
[51]: Student
0.205904
student
0.076482
Fresher
0.018053
Working
0.011710
Data Engineer
0.008295
     ...
MTECH - projects
0.000061
Mtech AIML Final year
0.000061
Government sector
0.000061
Works in Finance - Risk Management
0.000061
Working professional purchase dept - cost planner - // mechanical engineer 2017
0.000061
Name: What do you do currently ?, Length: 6592, dtype: float64
```

```
[52]: df['What do you do currently ?'] = df['What do you do currently ?'].apply(lambda␣
      ↪x: 1 if 'student' in str(x).strip().lower() else 0)
```

**Website Source**

```
[53]: df['Website Source'].isna().sum()
```

[53]:
```
13828
```

[54]:
```
df['Website Source'].value_counts()
```

[54]:
```
Sales lead                      21133
Start Project                     517
Demo button lead                  240
Chat lead                          78
Cashback lead                       6
eBook                               5
Sales lead, Demo button lead        4
Sales lead, Chat lead               2
Sales lead, eBook                   2
Name: Website Source, dtype: int64
```

[55]:
```
df = df.drop(["Website Source"], axis=1)
```

Dropping the Website Source column as there is not enough variance

**Marketing Source**

[56]:
```
df['Marketing Source'].value_counts()
```

[56]:
```
SEO                                           9751
Paid - Instagram                              3738
Paid-Adwords                                  3258
Paid-YouTube                                  2376
Affiliate                                     2215
Medium                                        2045
Paid - Facebook                               1436
Email Campaign                                 813
Paid - Linkedin                                136
Naukri                                          99
Medium, Paid-Adwords                            63
SEO, Medium, Paid-Adwords                       51
Linkedin jobs                                   42
SEO, Paid-Adwords                               39
Referral                                        38
SEO, Affiliate                                  38
SEO, Paid - Instagram                           30
Affiliate, Medium                               27
SEO, Medium                                     23
Paid - Instagram, Paid-Adwords                  23
Paid-Adwords, Paid-YouTube                      19
Affiliate, Paid-Adwords                         14
SEO, Paid - Facebook                            12
SEO, Paid-YouTube                               12
SEO, Paid - Instagram, Paid-Adwords             11
```

```
Paid - Facebook, Paid-Adwords                      8
SEO, Paid - Facebook, Paid-Adwords                 6
SEO, Linkedin jobs                                 5
SEO, Paid - Linkedin                               4
SEO, Naukri                                        4
SEO, Paid - Instagram, Medium                      4
Paid - Linkedin, Paid-Adwords                      3
Paid - Instagram, Medium                           3
SEO, Email Campaign                                2
Affiliate, Email Campaign                          2
Medium, Paid-YouTube                               2
Paid - Facebook, Paid - Instagram                  2
SEO, Paid - Instagram, Medium, Paid-Adwords        1
SEO, Paid - Facebook, Medium                       1
SEO, Affiliate, Medium                             1
Paid-Adwords, Email Campaign                       1
Paid - Linkedin, Medium                            1
Name: Marketing Source, dtype: int64
```

[57]:
```
df['Marketing Source'].isna().sum()
```

[57]:
```
9456
```

Marketing Source has a large number of missing value and it will be noisy if we do an imputation here.

Rather, let's create a new value Unknown which will be substituted for NA values

[58]:
```
df['Marketing Source'].fillna("Unknown", inplace=True)
```

PS: Imputation with Unknown led to improvements that dropping these rows

### 8.0.3 Label Encoding

**Transforming Categorical Variables**

Transforming variables is an important step in the data preprocessing pipeline of machine learning, as it helps to convert the data into a format that is suitable for analysis and modeling. There are several ways to transform variables, depending on the type and nature of the data.

Categorical variables, for example, are variables that take on discrete values from a finite set of categories, such as colors, gender, or occupation. One common way to transform categorical variables is through one-hot encoding. One-hot encoding involves creating a new binary variable for each category in the original variable, where the value is 1 if the observation belongs to that category and 0 otherwise. This approach is useful when the categories have no natural order or ranking.

Another way to transform categorical variables is through label encoding. Label encoding involves assigning a unique integer value to each category in the variable. This approach is useful when the categories have a natural order or ranking, such as low, medium, and high. Transforming categorical features into numerical labels:

**Note:** We are NOT using dummies here to minimize the explosion of columns because of the distance methods we are using.

[59]:
```
label_encoder1 = preprocessing.LabelEncoder()
```

[60]:
```
df['Marketing Source']= label_encoder1.fit_transform(df['Marketing Source'])
```

[61]:
```
label_encoder2 = preprocessing.LabelEncoder()
```

[62]:
```
df['Lead Owner']= label_encoder2.fit_transform(df['Lead Owner'])
```

[63]:
```
label_encoder3 = preprocessing.LabelEncoder()
df['Creation Source']= label_encoder3.fit_transform(df['Creation Source'])
```

[64]:
```
df.head()
```

[64]:
```
   Lead Owner  Interest Level  Creation Source  What do you do currently ?  \
0          20               0                0                           1
1          18               1                0                           0
2          17               0                0                           0
3           5               0                0                           0
4          16               0                0                           0

   Marketing Source  hour_of_day  day_of_week
0                42           16            3
1                13            9            5
2                19           10            4
3                19           17            4
4                 0           10            0
```

We transformed 3 columns using label encoding

Remember one thing, you should always use the same label encoding variable for test dataset. Since here we are handling train/test together, we are not worrying about it

## 9 Model Building and Testing

[80]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_recall_curve, roc_curve,␣
 ↪plot_roc_curve, plot_precision_recall_curve
from sklearn.model_selection import train_test_split

from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
```

Identify the right features for the model

```
[66]: X = df[["Lead Owner", "What do you do currently ?", "Marketing Source",␣
       ↪"Creation Source", "hour_of_day", "day_of_week"]]
      y = df["Interest Level"]
```

**Splitting the dataset into a training and production dataset:**

- Training: Part of data used for training our supervised models
- Test: Part of the dataset used for testing our models performance

```
[67]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
       ↪random_state=42)
```

```
[68]: X_train.head()
```

```
[68]:        Lead Owner  What do you do currently ?  Marketing Source  \
      2201           19                           1                42
      29205           4                           1                24
      24325          18                           0                 6
      20499           1                           0                42
      34645          17                           0                24

             Creation Source  hour_of_day  day_of_week
      2201                 0            4            2
      29205                0           19            2
      24325                0            3            1
      20499                0           16            1
      34645                0           12            0
```

We finally have prepared model ready data.

Lets look into model building now.

## 9.1 Supervised learning

Supervised learning uses a training set to teach models to yield the desired output. This training dataset includes inputs and correct outputs, which allow the model to learn over time. The algorithm measures its accuracy through the loss function, adjusting until the error has been sufficiently minimized.

Supervised learning can be separated into two types of problems when data mining—classification and regression:

1. Classification uses an algorithm to accurately assign test data into specific categories. It recognizes specific entities within the dataset and attempts to draw some conclusions on how those entities should be labeled or defined. Common classification algorithms are linear classifiers, support vector machines (SVM), decision trees, k-nearest neighbor, and random forest, which are described in more detail below.

2. Regression is used to understand the relationship between dependent and independent variables. It is commonly used to make projections, such as for sales revenue for a given business. Linear regression, logistical regression, and polynomial regression are popular regression algorithms.

## 9.2 Logistic Regression

Logistic regression is a statistical model used for binary classification tasks. Despite its name, it is primarily used for classification rather than regression. The goal of logistic regression is to estimate the probability that a given instance belongs to a particular class based on its features.

Here's a step-by-step explanation of the logistic regression algorithm:

1. Data Preparation: Logistic regression requires a labeled dataset for training. Each instance in the dataset consists of a set of features and a binary class label.
2. Linear Combination: Logistic regression assumes a linear relationship between the features and the log-odds (also known as the logit) of the probability of the positive class. The log-odds is defined as the logarithm of the odds, where odds is the ratio of the probability of the positive class to the probability of the negative class.
3. Logistic Function (Sigmoid): To convert the linear combination into a valid probability value between 0 and 1, logistic regression uses the logistic function, also known as the sigmoid function. The sigmoid function transforms any real-valued number into a value between 0 and 1.
4. Model Training (Maximum Likelihood Estimation): The logistic regression model is trained by finding the optimal values for the model parameters that maximize the likelihood of the observed data. This is typically done using an optimization algorithm such as gradient descent.
5. Cost Function (Log Loss): The cost function in logistic regression is often defined as the negative log-likelihood, or log loss. The goal is to minimize the log loss by adjusting the model parameters during training.
6. Classification Decision: Once the model parameters are learned, logistic regression uses a threshold (often 0.5) to classify instances. If the predicted probability is above the threshold, the instance is assigned to the positive class; otherwise, it is assigned to the negative class.
7. Model Evaluation: After training the logistic regression model, its performance is evaluated using a separate test dataset. Common evaluation metrics include accuracy, precision, recall, and F1 score.

Logistic regression is a widely used algorithm due to its simplicity and interpretability. It can handle both categorical and continuous features, and it can be extended to handle multiclass classification problems (e.g., using one-vs-rest or softmax regression). However, logistic regression assumes a linear relationship between the features and the log-odds, which may limit its performance on complex, nonlinear datasets. In such cases, more advanced techniques like decision trees, support vector machines, or neural networks may be more appropriate.

## 9.3 Naive Bayes

Naive Bayes is a simple and popular algorithm for classification tasks. It is based on Bayes' theorem, which calculates the probability of a certain event occurring given prior knowledge or evidence. Naive Bayes assumes independence between features, meaning that the presence or absence of one feature does not affect the presence or absence of another feature.

Here's a step-by-step explanation of the Naive Bayes algorithm:

1. Data Preparation: The algorithm requires a labeled dataset for training. Each instance in the dataset consists of a set of features and a corresponding class label.
2. Feature Independence Assumption: Naive Bayes assumes that all features in the dataset are independent of each other, given the class label. Although this assumption may not hold in real-world scenarios, Naive Bayes can still perform well in practice.
3. Calculating Class Priors: The algorithm begins by calculating the prior probability of each class label. It determines the relative frequency of each class in the training dataset.
4. Calculating Feature Likelihoods: For each feature, the algorithm calculates the likelihood of that feature occurring in each class. It does this by estimating the conditional probability of a feature given a class label.
5. Combining Likelihoods: Using the calculated likelihoods and the prior probabilities, Naive Bayes combines the probabilities using Bayes' theorem to compute the posterior probability of each class given the features of an instance.
6. Classification Decision: Once the posterior probabilities have been calculated for each class, the algorithm selects the class with the highest probability as the predicted class for the given instance.
7. Model Evaluation: After training the Naive Bayes model, its performance is evaluated using a separate test dataset. Common evaluation metrics include accuracy, precision, recall, and F1 score.

Naive Bayes is efficient and performs well even with a small amount of training data. However, it may struggle when faced with features that are not truly independent or when there are missing combinations of feature values in the training data.

There are different variations of Naive Bayes, such as Gaussian Naive Bayes for continuous features, Multinomial Naive Bayes for discrete features, and Bernoulli Naive Bayes for binary features. These variations accommodate different types of data and make appropriate probability assumptions based on the feature distributions.

Overall, Naive Bayes is a straightforward yet effective algorithm for classification tasks, particularly in text classification, spam filtering, and sentiment analysis.

## 9.4 Linear SVM

Linear SVM (Support Vector Machine) is a popular algorithm for binary classification. It aims to find an optimal hyperplane that separates the data into different classes by maximizing the margin between the classes.

Here's a step-by-step explanation of the linear SVM algorithm:

1. Data Preparation: Linear SVM requires a labeled dataset for training. Each instance in the dataset consists of a set of features and a binary class label.
2. Hyperplane Definition: A hyperplane in a feature space is a subspace of one dimension less than the input space. In the case of linear SVM, the hyperplane is defined as a linear combination of the input features.
3. Margin Maximization: The key idea in linear SVM is to find the hyperplane that maximizes the margin between the classes. The margin is the distance between the hyperplane and the nearest data points of each class. SVM aims to find the hyperplane that maximizes this margin, as it provides better generalization and reduces the risk of misclassification.
4. Optimization Objective: To find the optimal hyperplane, SVM formulates an optimization problem. The objective is to minimize the norm of the weight vector (||w||) while satisfying the constraint that all data points are correctly classified, i.e., they lie on the correct side of the hyperplane. This is often referred to as the primal problem.
5. Dual Problem: The primal problem can be transformed into a dual problem that involves maximizing a function subject to constraints. The dual problem is typically solved using optimization techniques, such as the quadratic programming algorithm.
6. Classification Decision: Once the optimal hyperplane is found, the linear SVM classifies new instances based on which side of the hyperplane they fall on. Instances on one side are assigned to one class, while instances on the other side are assigned to the other class.
7. Model Evaluation: After training the linear SVM model, its performance is evaluated using a separate test dataset. Common evaluation metrics include accuracy, precision, recall, and F1 score.

Linear SVM is widely used due to its ability to handle high-dimensional data and its effectiveness in scenarios where the classes are separable by a hyperplane. It is often used in text classification, image classification, and other applications. However, linear SVM may not perform well when the classes are not linearly separable. In such cases, nonlinear SVM kernels, such as polynomial or Gaussian (RBF) kernels, can be used to handle more complex decision boundaries.

## 9.5 Decision Tree

A decision tree is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. It has a hierarchical, tree structure, which consists of a root node, branches, internal nodes and leaf nodes.

Decision tree learning employs a divide and conquer strategy by conducting a greedy search to identify the optimal split points within a tree. This process of splitting is then repeated in a top-down, recursive manner until all, or the majority of records have been classified under specific class labels. Whether or not all data points are classified as homogenous sets is largely dependent on the complexity of the decision tree. Smaller trees are more easily able to attain pure leaf nodes—i.e. data points in a single class.

However, as a tree grows in size, it becomes increasingly difficult to maintain this purity, and it usually results in too little data falling within a given subtree. When this occurs, it is known as data fragmentation, and it can often lead to overfitting. As a result, decision trees have preference for small trees, which is consistent with the principle of parsimony in Occam's Razor; that is, "entities should not be multiplied beyond necessity." Said differently, decision trees should add complexity only if necessary, as the simplest explanation is often the best. To reduce complexity and prevent overfitting, pruning is usually employed; this is a process, which removes branches that split on features with low importance. The model's fit can then be evaluated through the process of cross-validation.

## 9.6 Random Forest

Another way that decision trees can maintain their accuracy is by forming an ensemble via a random forest algorithm; this classifier predicts more accurate results, particularly when the individual trees are uncorrelated with each other.

## 9.7 Gradient Boosting Trees

Gradient boosting is a machine learning technique used in regression and classification tasks, among others. It gives a prediction model in the form of an ensemble of weak prediction models, which are typically decision trees.

When a decision tree is the weak learner, the resulting algorithm is called gradient-boosted trees; it usually outperforms random forest. A gradient-boosted trees model is built in a stage-wise fashion as in other boosting methods, but it generalizes the other methods by allowing optimization of an arbitrary differentiable loss function.

Few examples of gradient boosting trees are Xgboost, LightGBM, etc

## 9.8 Bagging vs Boosting

1. Bagging: It is a homogeneous weak learners' model that learns from each other independently in parallel and combines them for determining the model average.

   https://media.geeksforgeeks.org/wp-content/uploads/202107071

2. Boosting: It is also a homogeneous weak learners' model but works differently from Bagging. In this model, learners learn sequentially and adaptively to improve model predictions of a learning algorithm.

   https://media.geeksforgeeks.org/wp-content/uploads/202107071

Credit: geeksforgeeks

```
[90]: lr = LogisticRegression()
gnb = GaussianNB()
svm = LinearSVC()
rf = RandomForestClassifier(n_estimators=300)
xgb = XGBClassifier(n_estimators=300, objective='binary:logistic',␣
 ↪tree_method='hist', eta=0.1, max_depth=3)
lgb = LGBMClassifier(n_estimators=300)
nn =  MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(10, 10),␣
 ↪random_state=123)
```

**Training multiple models together**

```
[91]: print("Training Linear Regression Model")
lr.fit(X_train, y_train)

print("Training Gaussian Naive Bayes Model")
gnb.fit(X_train, y_train)

print("Training Linear SVM Model")
svm.fit(X_train, y_train)

print("Training Random Forest Model")
rf.fit(X_train, y_train)

print("Training XGBoost Model")
xgb.fit(X_train, y_train)

print("Training Light GBM Model")
lgb.fit(X_train, y_train)

print("Training Neural Network Model")
nn.fit(X_train, y_train)
```

```
Training Linear Regression Model
Training Gaussian Naive Bayes Model
Training Linear SVM Model
Training Random Forest Model
Training XGBoost Model
Training Light GBM Model
Training Neural Network Model
```

```
[91]: MLPClassifier(alpha=1e-05, hidden_layer_sizes=(10, 10), random_state=123,
              solver='lbfgs')
```

All models are trained very qucikly here.

Scitkit-learn provides an additional parameter n_jobs=-1 which parallelize some of the models using cpu threads

## 10 Model Evaluation

#### 10.0.1 Evaluation metrics

https://blog.paperspace.com/content/images/2020/09/Fig01.jpg

1. Accuracy

$$\text{Accuracy} = \frac{TruePositive + TrueNegative}{TruePositive + FalsePositive + TrueNegative + FalseNegative} \tag{1}$$

2. Precision

$$\text{Precision} = \frac{TruePositive}{TruePositive + FalsePositive} \tag{2}$$

3. Recall

$$\text{Recall} = \frac{TruePositive}{TruePositive + FalseNegative} \tag{3}$$

4. F1-score

$$\text{F1-score} = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4}$$

[92]:
```python
def get_evaluation_metrics(model_name, model, pred, actual):
    print("Accuracy of %s: " % model_name, accuracy_score(pred, actual))
```

[93]:
```python
get_evaluation_metrics("Logistic Regression", lr, lr.predict(X_test), y_test)
get_evaluation_metrics("Gaussian Naive Bayes", gnb, gnb.predict(X_test), y_test)
get_evaluation_metrics("Linear SVM", svm, svm.predict(X_test), y_test)
get_evaluation_metrics("Random Forest", rf, rf.predict(X_test), y_test)
get_evaluation_metrics("XGBoost", xgb, xgb.predict(X_test), y_test)
get_evaluation_metrics("Light GBM", lgb, lgb.predict(X_test), y_test)
get_evaluation_metrics("Neural Network", nn, nn.predict(X_test), y_test)
```

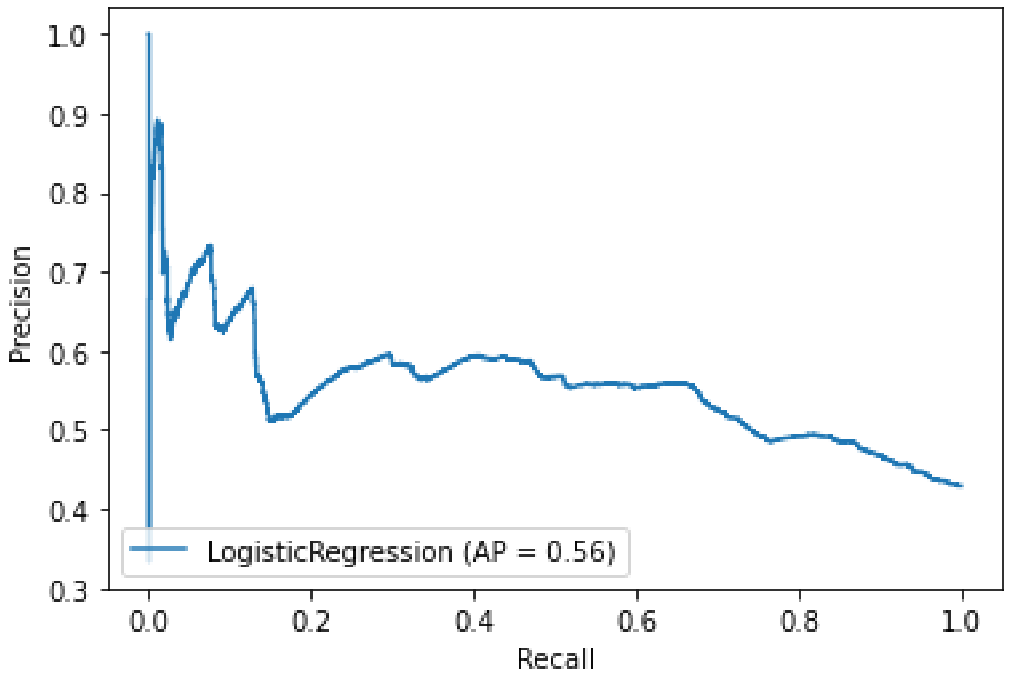
```
Accuracy of Logistic Regression:  0.6040765042579924
Accuracy of Gaussian Naive Bayes:  0.6056121736702499
Accuracy of Linear SVM:  0.5786681558006422
Accuracy of Random Forest:  0.6950998185117967
Accuracy of XGBoost:  0.7320954907161804
Accuracy of Light GBM:  0.7300013960631021
Accuracy of Neural Network:  0.637861231327656
```

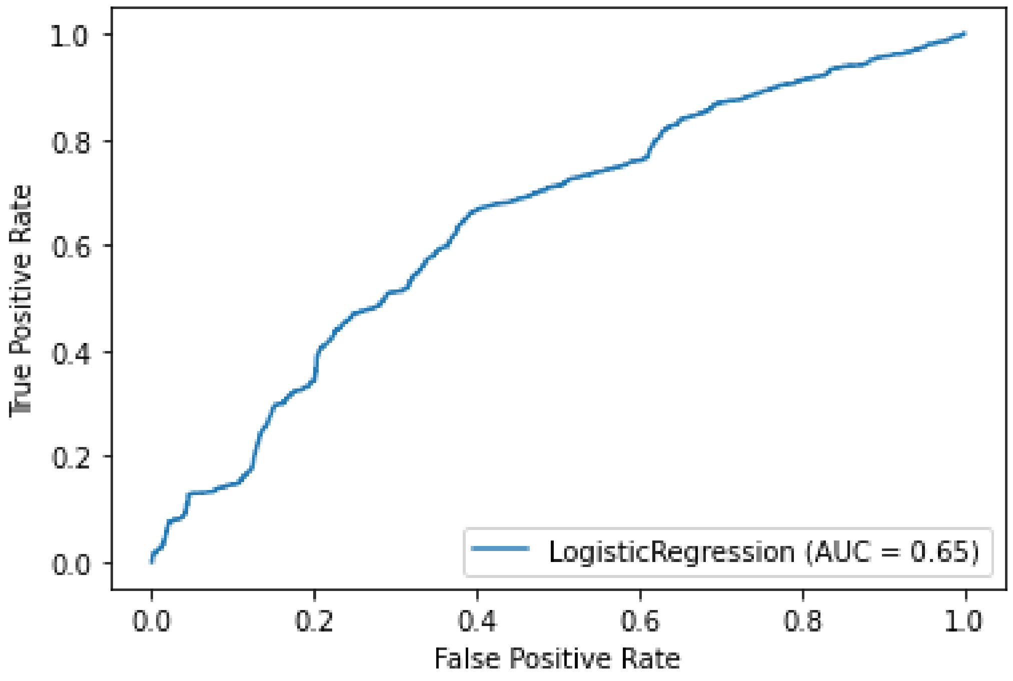
#### 10.0.2 PR-Curves

- We will plot PR curves to understand what is the trade-off between Precision and Recall and what threshold we should select
- One example is that we want higher precision, so we can pick a threshold where we get 70% precision but recall is 20%

[74]:
```
plot_precision_recall_curve(lr, X_test, y_test)
plot_roc_curve(lr, X_test, y_test)
```

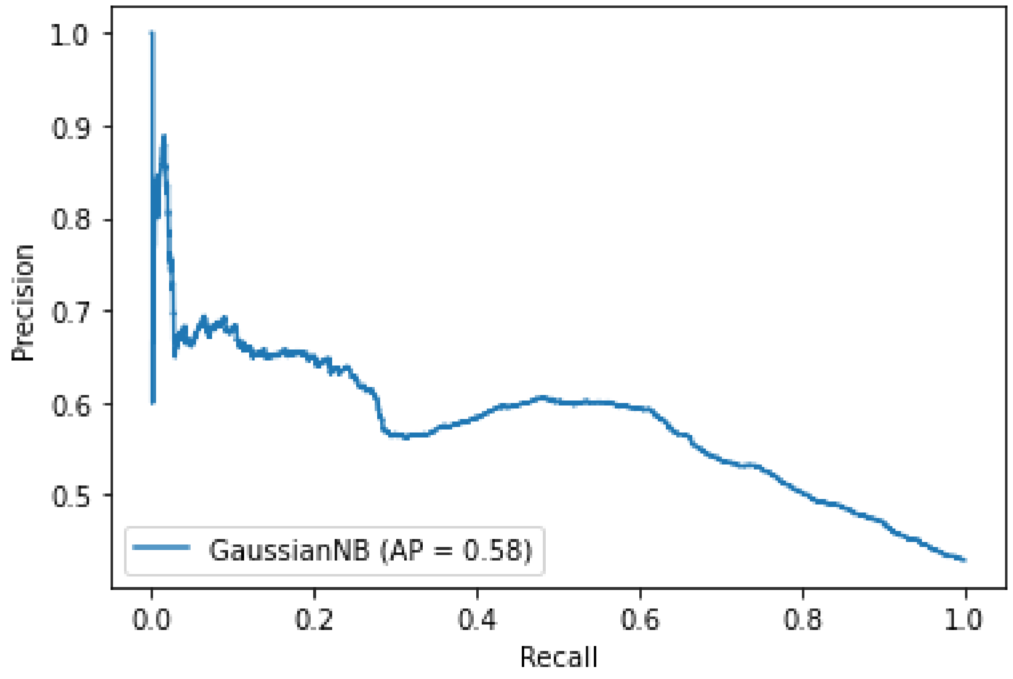
[74]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f7c4fc01be0>
```

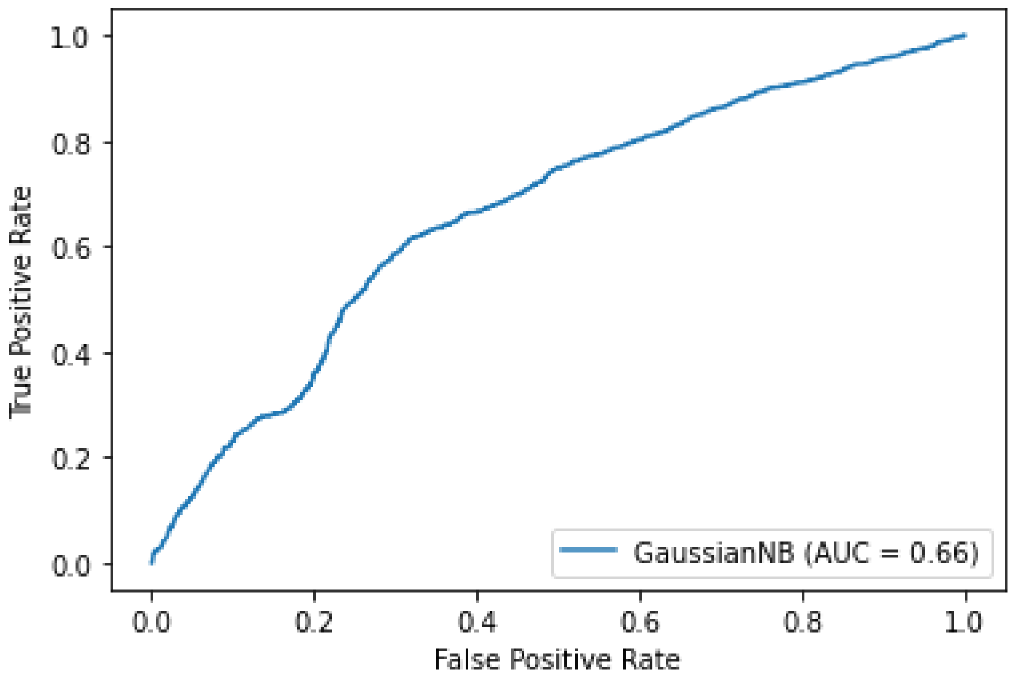




```
[75]: plot_precision_recall_curve(gnb, X_test, y_test)
      plot_roc_curve(gnb, X_test, y_test)
```

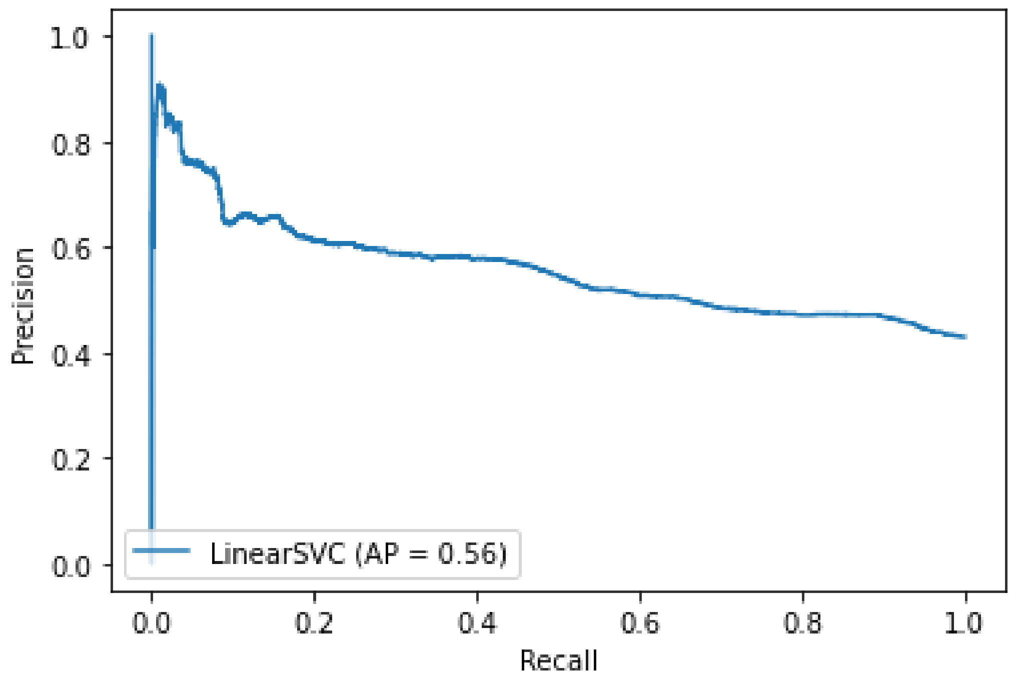
```
[75]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f7c489d2fd0>
```

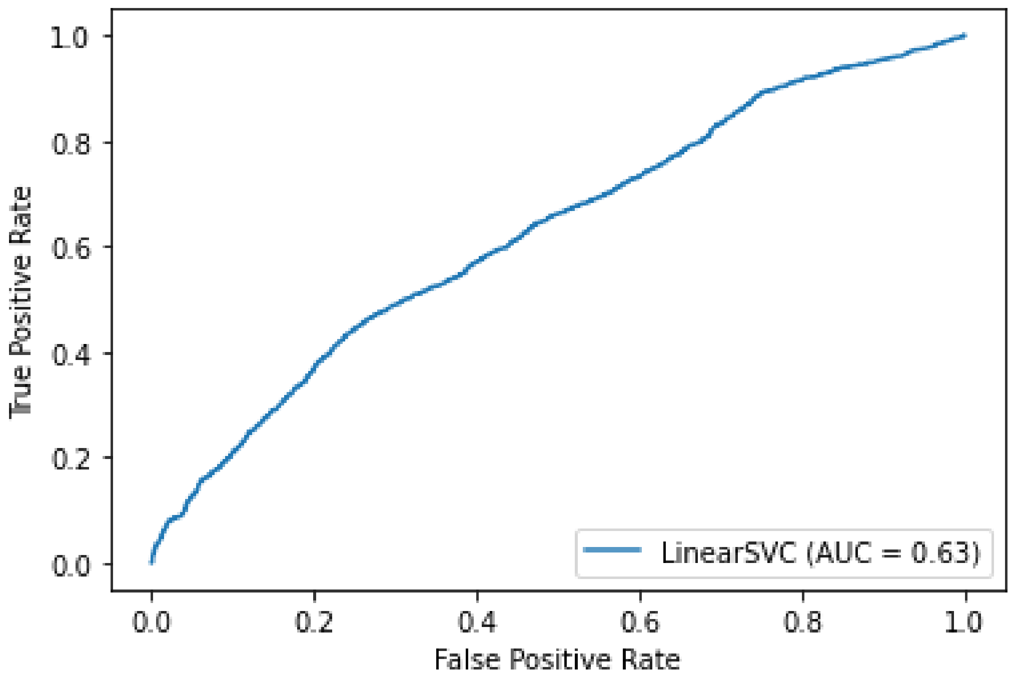




[76]:
```
plot_precision_recall_curve(svm, X_test, y_test)
plot_roc_curve(svm, X_test, y_test)
```

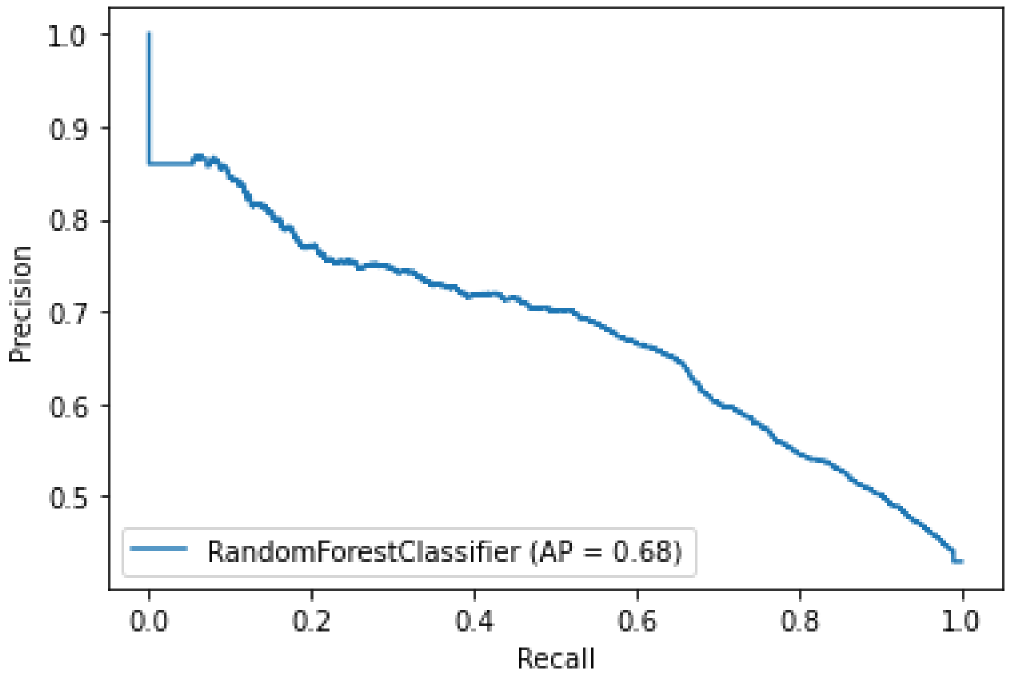
[76]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f7c48ae2850>
```

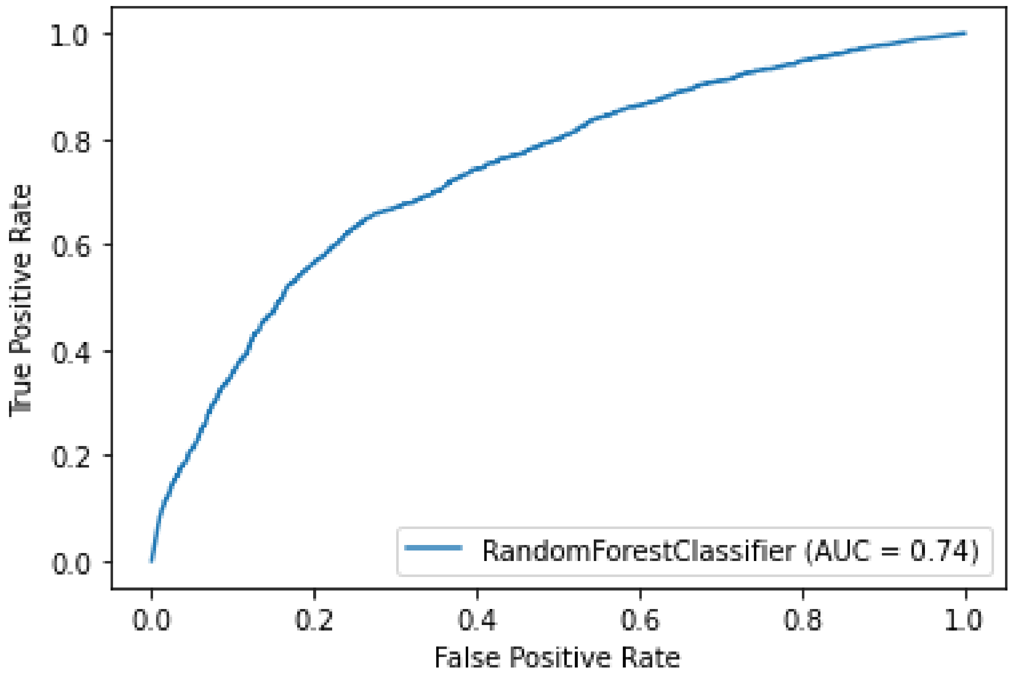




[77]:
```
plot_precision_recall_curve(rf, X_test, y_test)
plot_roc_curve(rf, X_test, y_test)
```

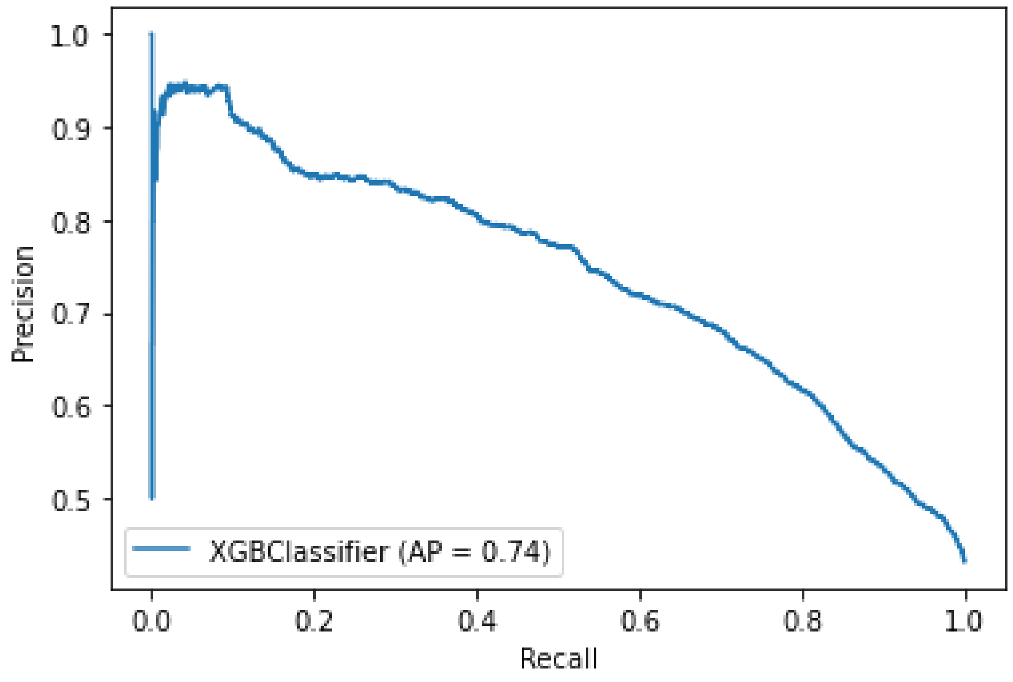
```
[77]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f7c489a5eb0>
```

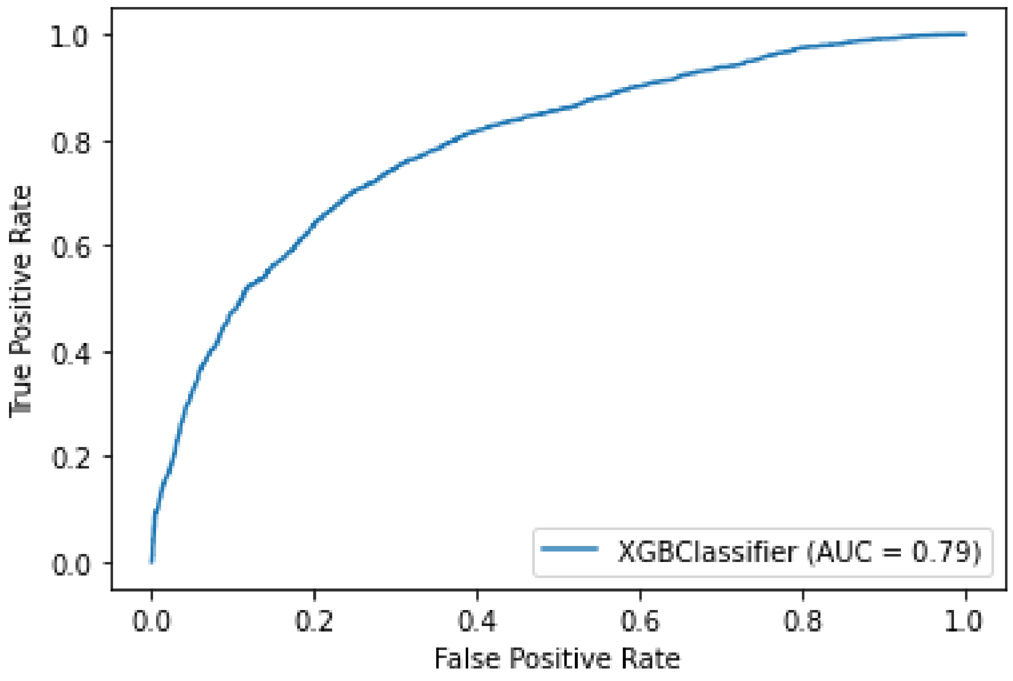

1.0
0.9
0.8
0.7
0.6
0.5
Precision
RandomForestClassifier (AP = 0.68)
0.0
0.2
0.4
0.6
0.8
1.0
Recall


1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
RandomForestClassifier (AUC = 0.74)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

```
[78]: plot_precision_recall_curve(xgb, X_test, y_test)
      plot_roc_curve(xgb, X_test, y_test)
```

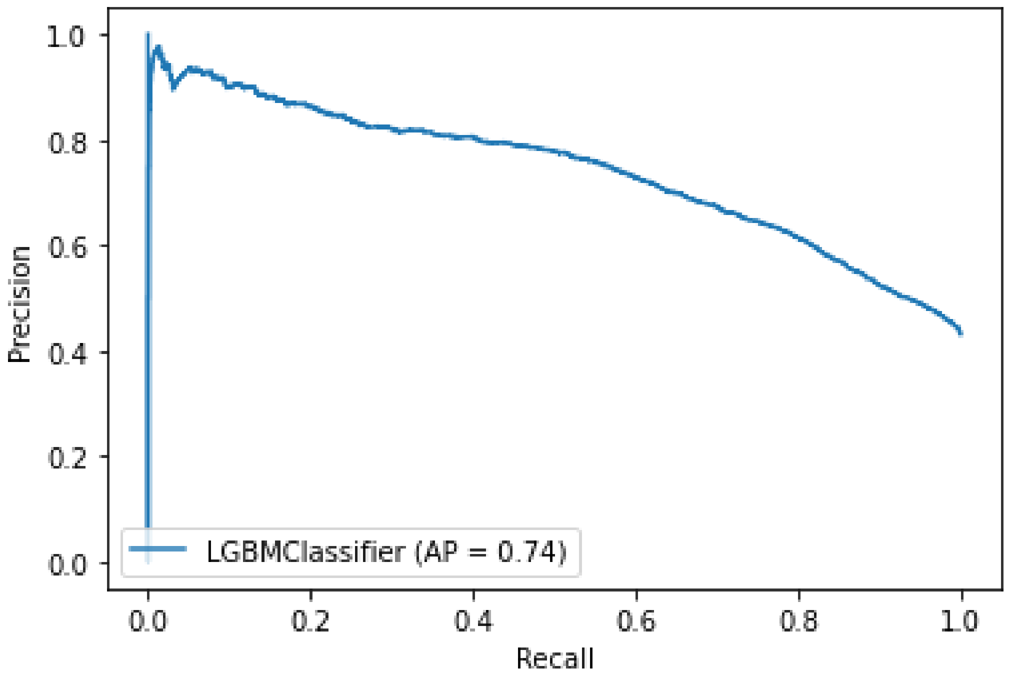
```
[78]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f7c54c1c760>
```

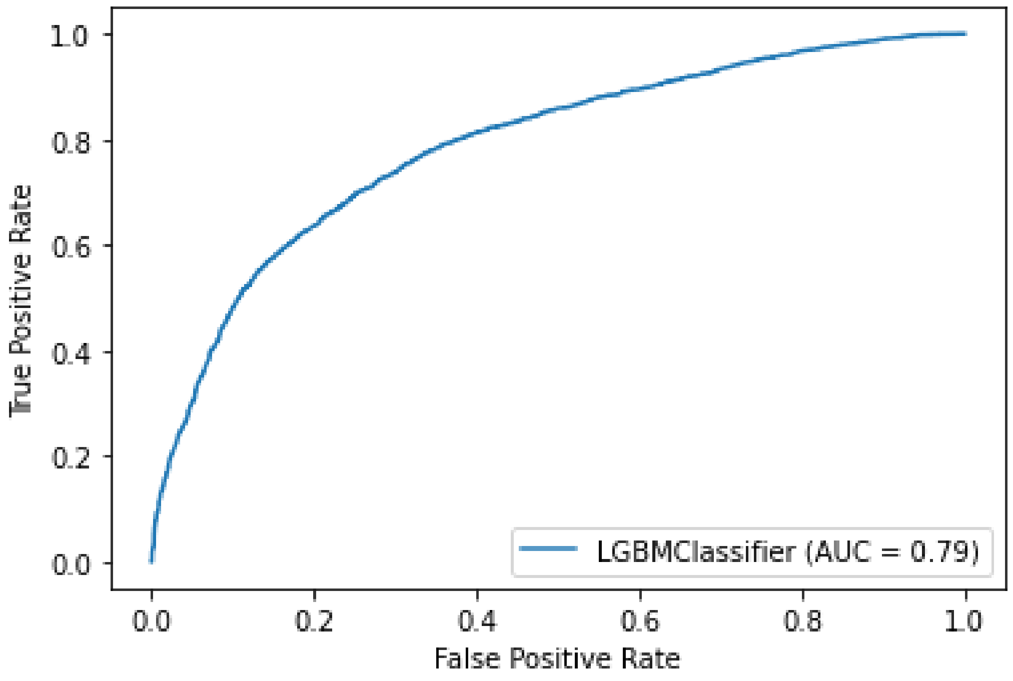




```
[79]: plot_precision_recall_curve(lgb, X_test, y_test)
      plot_roc_curve(lgb, X_test, y_test)
```

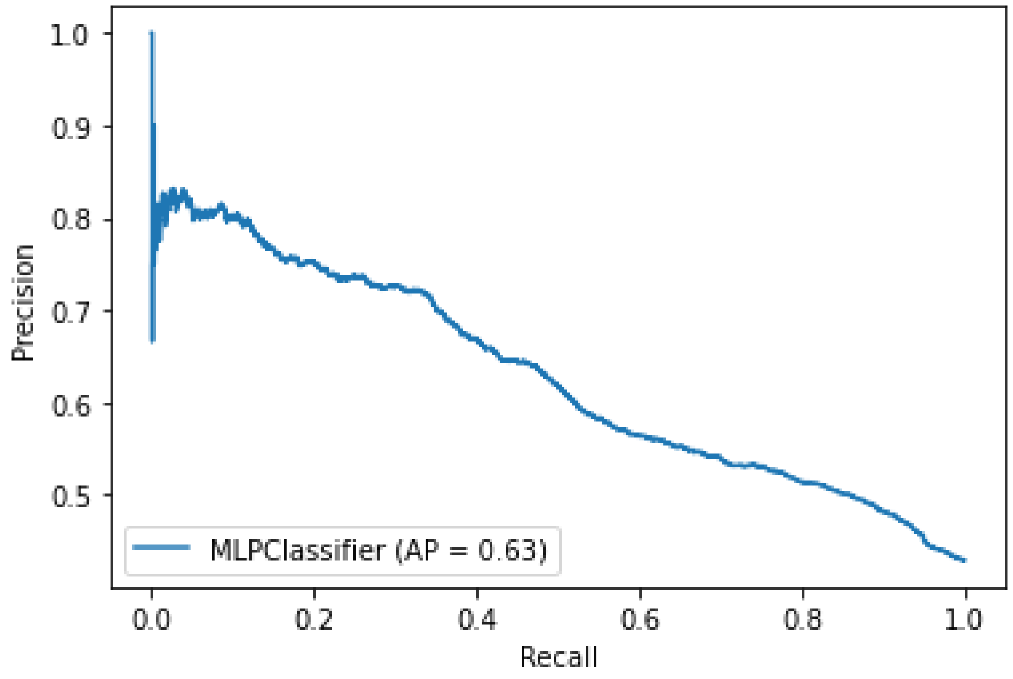
```
[79]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f7c48b66c40>
```

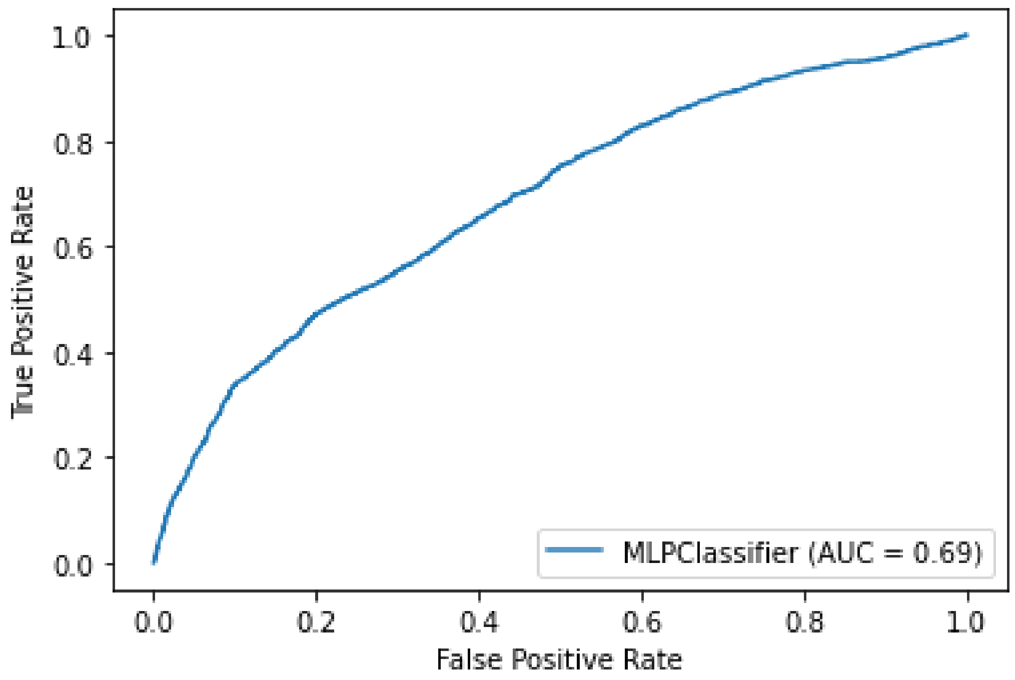




```
[94]: plot_precision_recall_curve(nn, X_test, y_test)
      plot_roc_curve(nn, X_test, y_test)
```

```
[94]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f7c4fc26fa0>
```





## 11 Conclusion

#### 11.0.1 Think about it

- Although numerically results are similar for XGBoost and LightGBM, which one do you think is better?
- Why does PR curve matter here? Think in terms of business justfication!

## 12 Try it out

- Can you try out grid-search to tune hyperparameters for these models?
- Can you come up with the right thresholds for these models depending on what do you feel is more important here? Precision or recall?
- Can you train a multi-layer perceptron and check the performance?

### 12.1 Conclusion

In this project we used a bunch of supervised models to predict if the customer would be interested in a lead.

A successful data science project requires a clear understanding of the business problem and the data available, as well as the ability to select and apply appropriate data preprocessing techniques, feature engineering methods, and machine learning algorithms. It is also important to assess and optimize the performance of the model and communicate the results effectively to stakeholders.

After looking at the PR and ROC curves above, we can conclude that LightGBM is giving us the best possible results.

## 12.2 Interview Questions

### 12.2.1 Supervised Learning:

- What are decision trees? How does the model decide on the split?
- What is boostrap aggregation?
- Explain bias and variance in context of boosting and bagging?
- How can you use decision tree for missing value imputation?
- How does regularization work in gradient boosting models?

### 12.2.2 Code Implementation:

- Is bagging or boosting computationally faster?
- Can you write your own code to calculate precision and recall?
- How would you handle dataset if the target variable would have been imbalanced?

[ ]:
```
```