

Topic : Term Deposit Subscription Prediction

Group : RSF3 Group 1

Members :

- Lau Chih Kei (18WMR00384)
- Tee Yu June (18WMR07982)
- Wong Kah Wai (18WMR04287)
- Yap Hui Wen (18WMR05135)

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler , LabelEncoder
from scipy import stats
import warnings
import keras
import random
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import (roc_auc_score, roc_curve)
from sklearn.linear_model import LogisticRegression
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn import metrics                    # Import scikit-learn metrics module for accuracy
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from scipy.stats import chi2_contingency
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, \
    recall_score, confusion_matrix, classification_report, \
    accuracy_score, f1_score
from sklearn.neighbors import KNeighborsClassifier

from google.colab import drive
drive.mount('/content/drive')
```



Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount('/content/drive')

## ▼ 1.0 Business Understanding

## 1.1 Business Background

Meybank, the largest financial institution in Malaysia that started in 1970. It provides different types of financial and banking services to the public which include deposit and investment, loan financing, wealth management, banking and so on. Due to an intense competition among the financial institutions or banks, Meybank has provided a broad range of products and services to attract the customers. During the marketing of those services and products, Meybank has spent a huge amount of money.

Therefore, Meybank is constantly organizing the marketing campaign to promote its term deposit plans to the public. This is because term deposits are important for banks, as the banks will use the fund from the term deposit to make loans and invest money in other financial products that will bring a higher rate of return. With this, the return from the investments will be higher than the interest paid for the term deposit and this is considered one of the main sources of income for the banks. In order to facilitate the sales and marketing on the term deposit plans, Meybank is planning to collaborate with a group of data scientists to integrate the knowledge of business and data mining. With this collaboration, a business intelligence system will be developed to predict the potential customers that will subscribe to the term deposit plans.

### Business Objectives

1. Classify and determine the target customer that has the high possibility of subscribing to the term deposit plans.
2. Increase the successful subscription rate among the target customers to the term deposit plan.

## ▼ 1.2 Business Analysis

### Inventory of Resources

Personnel Resource:

- Data Warehouse Administrator:

Manage and retrieve data related to the term deposit in Meybank.

- Business Analyst:

Experienced in the finance field and can give advice about domain knowledge to the project team. Besides, they help the project team to analyze the business process and extract the usable data that is suitable for analytical modeling.

- Legal Expert:

Provide legal advice to the project team about the regulation of confidential data used in the analytical model.

- Data Analysts:

The project team that applies data mining techniques to carry out actual data analysis.

Present the finding of analytical modeling in the readable form to the related stakeholders.

Data:

- Historical business data of Meybank from the warehouse.

Computing Resource:

- Z170 motherboard
- Intel Core i7-6770HQ processor
- 8GB RAM
- 128GB SSD storage
- GTX 1080 Ti GPU (Gokkulnath, 2017)

Software:

- Python
- SQL
- Julia
- Tableau
- Excel
- PSPP

## Requirements, Assumptions, Constraints

### Requirements

- The project shall be able to complete within 4 months.
- The confidential data of customers such as address, NRIC, phone number shall not be disclosed to third parties.
- A thorough descriptive data analysis shall be carried out to study the relationship between the data.
- The data pre-processing shall be performed to improve the quality of data used.
- The outlier, missing value and inconsistent data shall be removed during the data analysis stage.
- The result of prediction shall be more than 90% of accuracy.
- The built model shall be able to integrate with the banking deposit system.
- The built model shall be applicable in the real world environment.

## Assumptions

- The budget given by Meybank to complete the project is RM100,000.
- The sample chosen in the data sampling represents the true population of the term depositors.
- The data used for the analytical modeling is the unbiased data.
- The collected data is related to the term deposit analytic project.
- The historical data are always practical and applicable to the latest business environment.
- The attributes found in the extracted data are highly dependent on each other.

## Constraints

- The confidential data of customers should not be included in the analytical modeling project.
- There are software constraints in this analytical modeling project as only open source softwares will be used.
- The time constraints for this project is only 4 months, the project team needs to complete the data analytical modeling within the given duration.
- There are only 4 data analysts allocated for this project, with only 2 data analysts possessing 10 years of data mining experience, while the rest are junior staff.

## Risks and Contingencies

No	Possible Issue	Actions to be taken
1	Project Behind Schedule	<ul style="list-style-type: none"> <li>• Add more staff to the project team in order to ensure the sufficiency of human resources.</li> <li>• Reassign tasks to the project team according to their skills and strength.</li> </ul>
2	Unfamiliar with New Software Tools	<ul style="list-style-type: none"> <li>• Provide training to the project team to familiarize themselves with the new development tools.</li> <li>• Provide documentation such as lab manuals as references to the project team.</li> </ul>
3	Different Coding Standards among Programmers	<ul style="list-style-type: none"> <li>• Project team discusses and agrees on the coding standards in the beginning of the development process.</li> <li>• Project team always checks and reviews each other's code to ensure the adherence with coding standards.</li> </ul>

## Terminologies

1. age

2. job : type of jobs
3. marital : marital status
4. education : educational level
5. default : has credit in default?
6. balance : average yearly balance
7. housing : has housing loan?
8. loan : has personal loan?
9. contact : contact communication type
10. day : last contact day of the month
11. month : last contact month of the year
12. duration : last contact duration in seconds
13. campaign : number of contacts performed during this campaign and for this client
14. pdays : number of days that passed by after the client was last contacted from a previous campaign
15. previous : number of contacts performed before this campaign and for this client
16. poutcome : outcome of the previous marketing campaign
17. subscribed : has the client subscribed a term deposit?

## 1.3 Data Mining Goals

### Business Success Criteria

Based on the system prediction, the total number of actual customers that

- Subscribe to the term deposit plans should be more than 90%.
- The increase of the successful subscription rate among the target customers to the term deposit plan should be more than 50%.

### Data Mining Success Criteria

- The accuracy of the prediction result should be more than 90%.

## ▼ 1.4 Project Plan

	i	Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1			<b>Business Understanding Stage</b>	<b>7 days</b>	<b>Mon 6/1/20</b>	<b>Tue 6/9/20</b>		
2			Determine business objective	1 day	Mon 6/1/20	Mon 6/1/20		Lau Chih Kei
3			Business analysis	4 days	Tue 6/2/20	Fri 6/5/20	2	Tee Yu June
4			Determine data mining goal	1 day	Mon 6/8/20	Mon 6/8/20	3	Wong Kah Wai
5			Project and Achievement Review	1 day	Tue 6/9/20	Tue 6/9/20	4	
6			<b>Data Understanding Stage</b>	<b>11 days</b>	<b>Wed 6/10/20</b>	<b>Wed 6/24/20</b>		
7			Data description	3 days	Wed 6/10/20	Fri 6/12/20	5	Yap Hui Wen
8			Data exploratory	3 days	Mon 6/15/20	Wed 6/17/20	7	Lau Chih Kei
9			Data quality verification	3 days	Thu 6/18/20	Mon 6/22/20	8	Tee Yu June
10			Produce data quality report	1 day	Tue 6/23/20	Tue 6/23/20	9	Yap Hui Wen
11			Project and Achievement Review	1 day	Wed 6/24/20	Wed 6/24/20	10	
12			<b>Data Preparation Stage</b>	<b>38 days</b>	<b>Thu 6/25/20</b>	<b>Mon 8/17/20</b>		
13			Data selection	9 days	Thu 6/25/20	Tue 7/7/20	11	Lau Chih Kei,Tee Yu June
14			Data cleaning	9 days	Wed 7/8/20	Mon 7/20/20	13	Wong Kah Wai,Yap Hui Wen
15			Data construction	9 days	Tue 7/21/20	Fri 7/31/20	14	Tee Yu June,Wong Kah Wai
16			Data integration	10 days	Mon 8/3/20	Fri 8/14/20	15	Lau Chih Kei,Yap Hui Wen
17			Project and Achievement Review	1 day	Mon 8/17/20	Mon 8/17/20	16	

	i	Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names
18			<b>Modeling Stage</b>	<b>20 days</b>	<b>Tue 8/18/20</b>	<b>Mon 9/14/20</b>		
19			Modeling technique selection	2 days	Tue 8/18/20	Wed 8/19/20	17	Lau Chih Kei
20			Test design generation	1 day	Thu 8/20/20	Thu 8/20/20	19	Yap Hui Wen
21			Build model	12 days	Fri 8/21/20	Mon 9/7/20	20	Lau Chih Kei,Tee Yu June,Wong Kah Wai,Yap Hui Wen
22			Model assessment	4 days	Tue 9/8/20	Fri 9/11/20	21	Lau Chih Kei,Tee Yu June,Wong Kah Wai,Yap Hui Wen
23			Project and Achievement Review	1 day	Mon 9/14/20	Mon 9/14/20	22	
24			<b>Evaluation Stage</b>	<b>4 days</b>	<b>Tue 9/15/20</b>	<b>Fri 9/18/20</b>		
25			Result evaluation	2 days	Tue 9/15/20	Wed 9/16/20	23	Lau Chih Kei,Tee Yu June
26			Process review	2 days	Thu 9/17/20	Fri 9/18/20	25	Wong Kah Wai,Yap Hui Wen
27			<b>Deployment Stage</b>	<b>8 days</b>	<b>Mon 9/21/20</b>	<b>Wed 9/30/20</b>		
28			Deployment planning	2 days	Mon 9/21/20	Tue 9/22/20	26	Lau Chih Kei
29			Plan monitoring and maintenance	2 days	Wed 9/23/20	Thu 9/24/20	28	Tee Yu June
30			Produce final report	2 days	Fri 9/25/20	Mon 9/28/20	29	Wong Kah Wai
31			Project and Achievement Review	2 days	Tue 9/29/20	Wed 9/30/20	30	Yap Hui Wen

## ➤ 2.0 Data Understanding

```
#read train and test dataset
```

```
df1 = pd.read_csv("/content/drive/Shared drives/Data Science/train.csv")
```

```
#drop id column in order to prevent data leakage
```

```
df1 = df1.drop(['ID'],axis=1)
```

```
df2 = pd.read_csv("/content/drive/Shared drives/Data Science/bank.csv" , sep=';' , engine='py
df2.rename(columns={'y': 'subscribed'} , inplace = True)
```

```

#concat 2 datasets
X = pd.concat([df1,df2], ignore_index=False)
X = X.drop_duplicates()
X.head()

no_sub = X[X['subscribed'] == 'no'].index

count = 0;
sampling = []
for x in no_sub:
    if(count % 2 == 0):
        sampling.append(x)
    count = count + 1

X.drop(sampling,inplace=True)

no_sub = X[X['subscribed'] == 'no'].index

count = 0;
sampling = []
for x in no_sub:
    if(count % 2 == 0):
        sampling.append(x)
    count = count + 1

X.drop(sampling,inplace=True)

print(X['subscribed'].value_counts())


train , test = train_test_split(X,test_size=0.2, random_state=6)

#train dataset
Xtrain = train.iloc[:, :-1] #drop the last column
ytrain = train.iloc[:, -1] #choose the last column

#test dataset
Xtest = test.iloc[:, :-1] #drop the last column
ytest = test.iloc[:, -1] #choose the last column

print('There are {} samples in the training set and {} samples in the test set.'.format(
Xtrain.shape[0] , Xtest.shape[0]))
print()

```



```

no      6980
yes     4164
Name: subscribed, dtype: int64
There are 8915 samples in the training set and 2229 samples in the test set.

```

The table below shows the first 5 rows of the train dataset. Based on the table, we can briefly

```
Xtrain.head()
```



	age	job	marital	education	default	balance	housing	loan	contact
<b>16196</b>	22	student	single	secondary	no	948	no	no	telephone
<b>31317</b>	53	management	divorced	tertiary	no	3158	no	no	cellular
<b>7815</b>	33	technician	single	secondary	no	111	no	no	cellular
<b>11929</b>	59	retired	married	primary	no	866	no	yes	cellular
<b>5429</b>	40	self-employed	single	tertiary	no	10346	no	no	cellular

The datatype of the columns can be further studied by using info() function.

Based on the displayed result, the datatype of job, marital, education, default, housing, loan, contact, month, poutcome are object, they can be considered as categorical variables.

Besides, the datatype of age, balance, day, duration, campaign, pdays, previous are int64, it means that they are numerical variables.

```
#to identify numerical and categorical data
```

```
Xtrain.info()
```



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8915 entries, 16196 to 10366
Data columns (total 16 columns):
#   Column          Non-Null Count  Dtype
---  -
0   age             8915 non-null   int64
1   job             8915 non-null   object
2   marital         8915 non-null   object
3   education       8915 non-null   object
4   default         8915 non-null   object
5   balance         8915 non-null   int64
6   housing         8915 non-null   object
7   loan            8915 non-null   object
8   contact         8915 non-null   object
9   day             8915 non-null   int64
10  month           8915 non-null   object
11  duration        8915 non-null   int64
12  campaign        8915 non-null   int64
13  pdays           8915 non-null   int64
14  previous        8915 non-null   int64
15  poutcome        8915 non-null   object
dtypes: int64(7), object(9)
memory usage: 1.2+ MB
```

From the graph below, it seems like nothing highly correlated as most of the values is below 0.5.

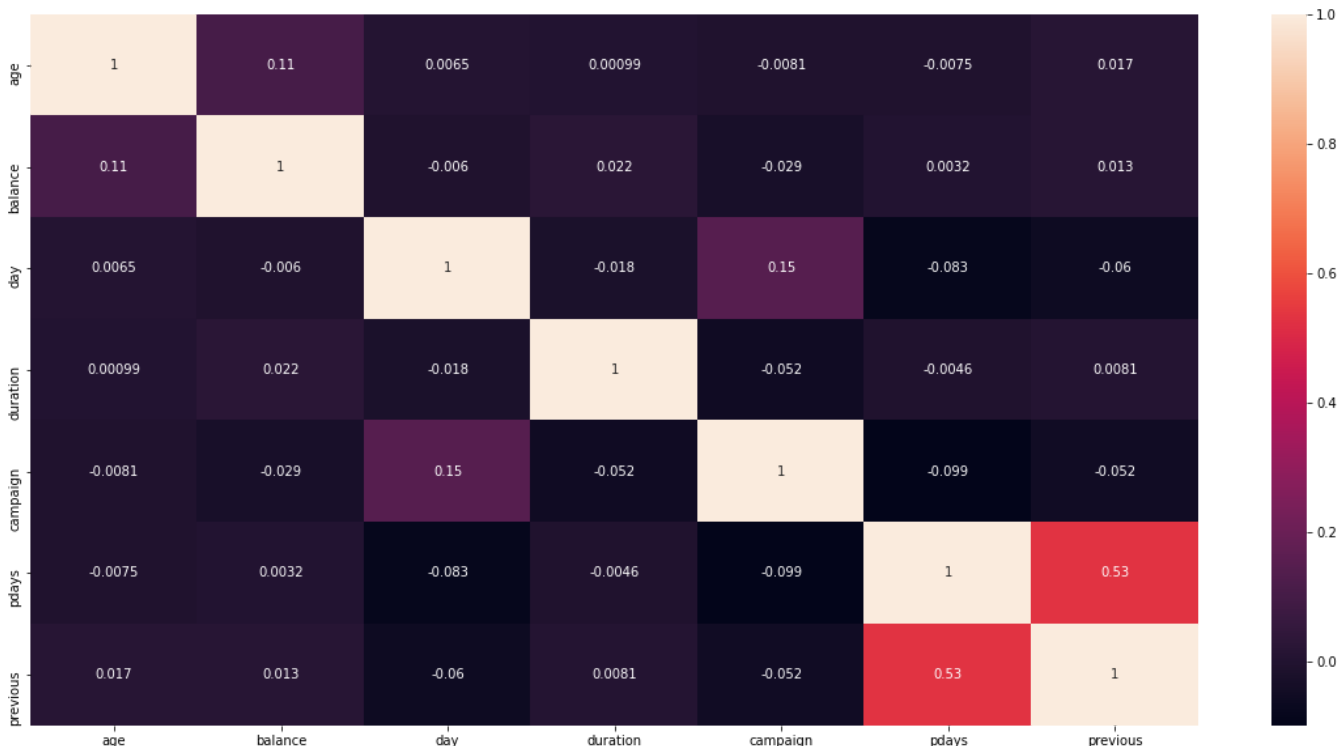


1 means is correlated while 0 means no correlation.

```
plt.figure(figsize=(20,10))
sns.heatmap(Xtrain.corr(),annot = True)
```



<matplotlib.axes.\_subplots.AxesSubplot at 0x7fcd13ca7240>



## 2.1 Exploring Numerical Columns

### Data Aggregation

The table belows shows the data aggregation of each numerical column which can brings insight about the data. As we observe through the table, there is no any missing value among the columns as the number of count for each column is same which is 8915. Besides, in the 'previous' column, noted there are 0 values in minimum, 25%, 50% row while there is a value of 58 in the max row.

However, in the 'previous' column, 0 is meaningful which indicates no contact before the campaign.

```
...
# descriptive analysis for numerical columns
```

```
Xtrain.describe()
```



	age	balance	day	duration	campaign	pdays	p
<b>count</b>	8915.000000	8915.000000	8915.000000	8915.000000	8915.000000	8915.000000	8915
<b>mean</b>	41.114077	1496.250589	15.456310	333.322378	2.567134	53.633202	0
<b>std</b>	11.735956	3358.039251	8.307851	320.752302	2.822801	112.184930	2
<b>min</b>	18.000000	-3058.000000	1.000000	1.000000	1.000000	-1.000000	0
<b>25%</b>	32.000000	111.000000	8.000000	128.000000	1.000000	-1.000000	0
<b>50%</b>	39.000000	512.000000	15.000000	229.000000	2.000000	-1.000000	0
<b>75%</b>	49.000000	1604.000000	21.000000	426.000000	3.000000	9.000000	1
<b>max</b>	93.000000	81204.000000	31.000000	4918.000000	55.000000	854.000000	58

## ▼ Data Distribution Visualization

Graph Age: The majority customers' age are between 20 to 40.

Graph Balance: The average yearly balance housing of customers are between 0 to 10000.

Graph Campaign: The average number of contacts performed by the customers during the campaign is between 1 to 20.

Graph Day: The last contact day of the month performed by the customers was mostly between 10 to 20.

Graph Duration: The last contact duration of the customers are averagely between 0 to 2000 seconds.

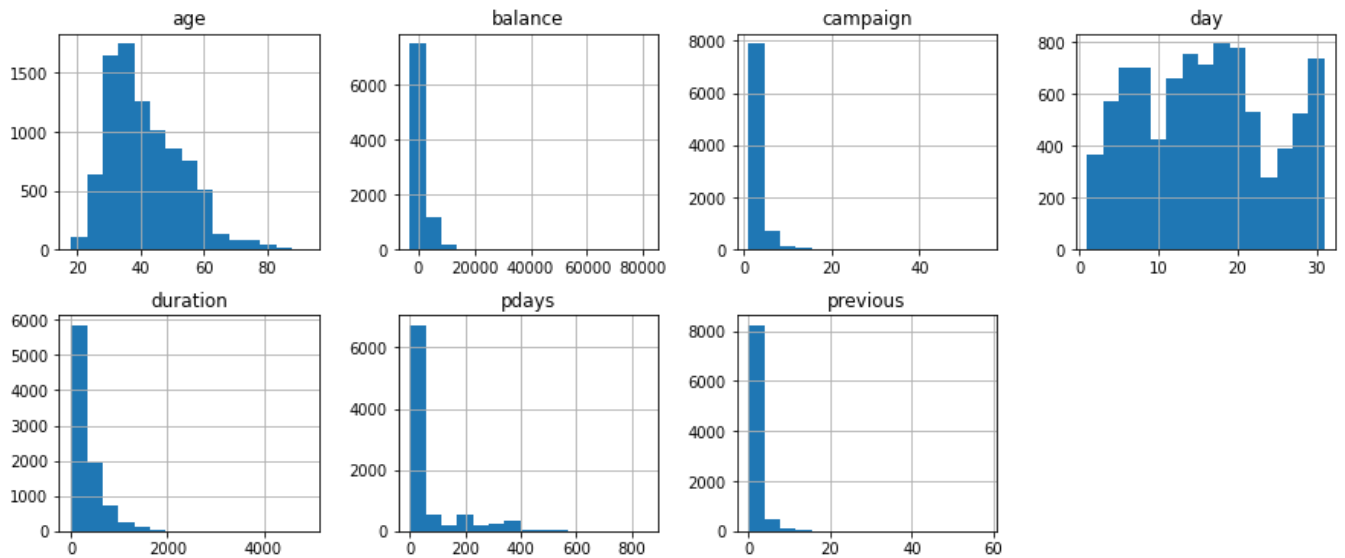
Graph Pdays: The number of days passed by after the customers were last contacted from previous campaign are mostly between 0 to 100 days.

Graph Previous: The number of contacts performed by each customers before the campaign are mostly between 0 to 10.

```
# visualize the data distribution of numerical data
```

```
Xtrain[['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']].hist(bins=15, figsize=
```





```
def bivariate_distribution(title):
    train = pd.concat([Xtrain,ytrain],axis=1)

    sns.FacetGrid(train,hue='subscribed' ,size=5 ).map(sns.distplot,title).add_legend()
```

The graph indicates a normal distribution of "yes" and "no" on subscriptions for age column which shows that the highest is 30 years old for both decisions.

Besides that, the "no" on subscriptions are linearly decreases after 60 years old and remain constant after 70 years old.

Overall, the distribution of "yes" on subscriptions are lower than the distribution of "no" on subscriptions.

```
bivariate_distribution('age')
```



```
/usr/local/lib/python3.6/dist-packages/seaborn/axisgrid.py:243: UserWarning: The `size`  
warnings.warn(msg, UserWarning)
```



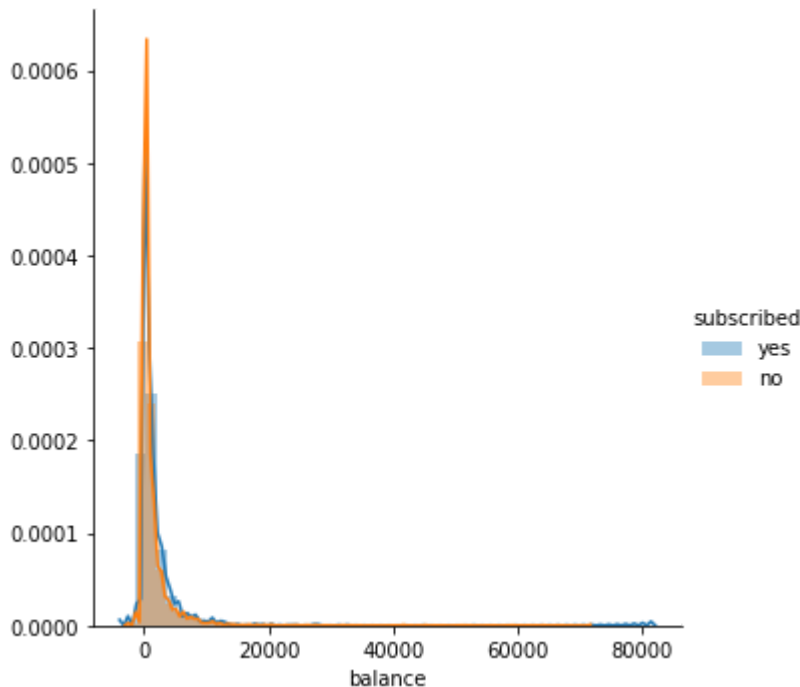
According to the graph shown below, the average yearly balance housing for both subscribed customers and not subscribe customers are in between of 0 to 10000.



```
bivariate_distribution('balance')
```



```
/usr/local/lib/python3.6/dist-packages/seaborn/axisgrid.py:243: UserWarning: The `size`  
warnings.warn(msg, UserWarning)
```

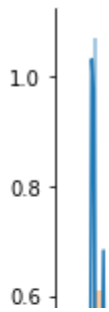


According to the graph below, the average number of contacts performed by subscribed and not subscribe customers are mostly in between of 0 to 10.

```
bivariate_distribution('campaign')
```



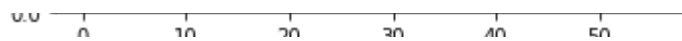
```
/usr/local/lib/python3.6/dist-packages/seaborn/axisgrid.py:243: UserWarning: The `size`
warnings.warn(msg, UserWarning)
```



The graph indicates a distribution of "yes" and "no" on subscriptions for day column which shows that the highest is 18th for both decisions.

Besides that, the "no" on subscriptions are instantly decreases after 20th and 28th.

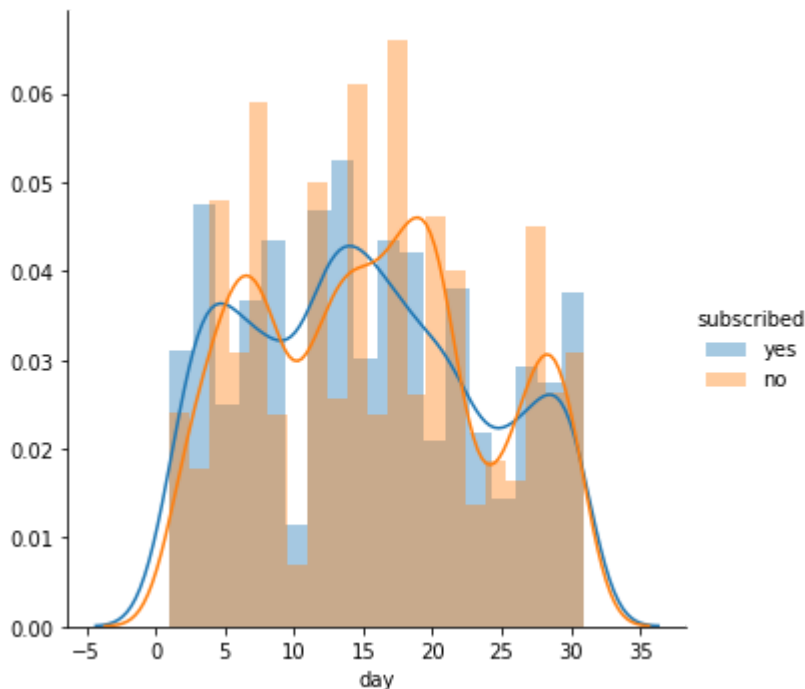
Overall, the distribution of "yes" on subscriptions are lower than the distribution of "no" on subscriptions.



```
bivariate_distribution('day')
```



```
/usr/local/lib/python3.6/dist-packages/seaborn/axisgrid.py:243: UserWarning: The `size`
warnings.warn(msg, UserWarning)
```

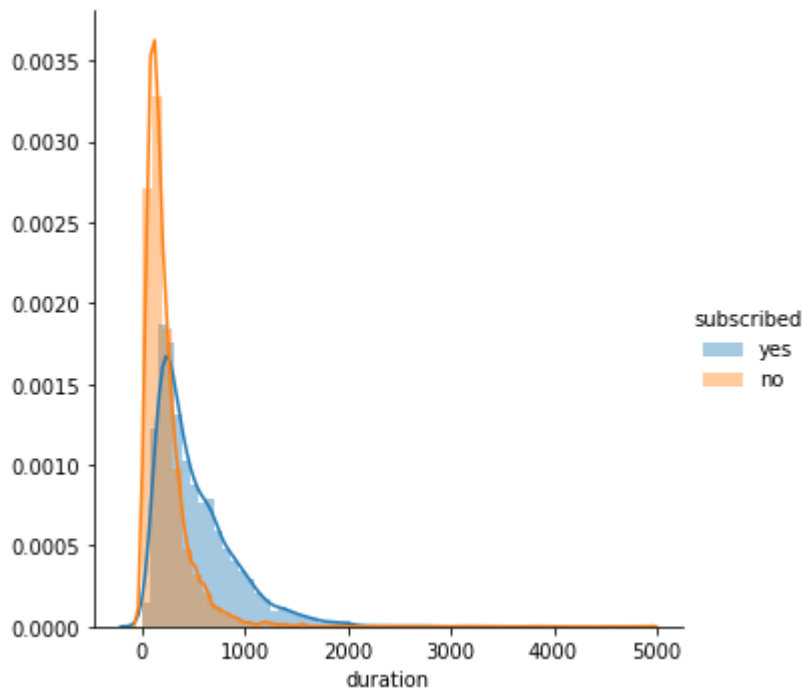


According to graph below, the last contact duration of not subscribe customers are mostly between 0 to 1000 seconds while for subscribed customers, the last contact duration is between 0 to 2000 seconds.

```
bivariate_distribution('duration')
```



```
/usr/local/lib/python3.6/dist-packages/seaborn/axisgrid.py:243: UserWarning: The `size`  
warnings.warn(msg, UserWarning)
```



The graph indicates a normal distribution of "yes" on subscriptions which shows that the highest is 0 day.

Besides that, there is a flat distribution of "no" on subscriptions because people who do not subscribe will not contact the clients.

Overall, the distribution of "yes" on subscriptions determines that the highest passed day after the last contacted to clients is 0.

```
bivariate_distribution('pdays')
```



```
/usr/local/lib/python3.6/dist-packages/seaborn/axisgrid.py:243: UserWarning: The `size`
warnings.warn(msg, UserWarning)
```



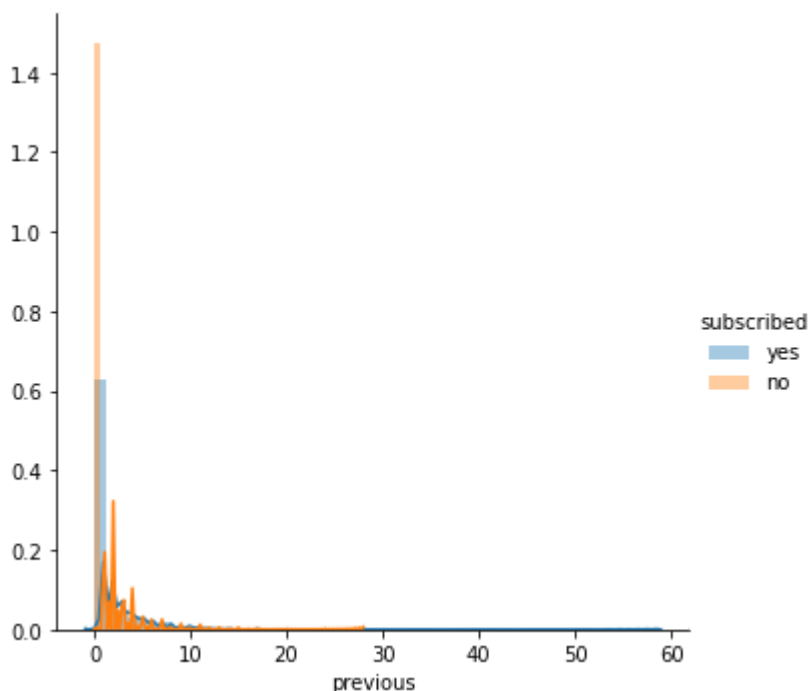
According to the graph shown below, the number of contacts performed by subscribed and not subscribe customers are mostly in between of 0 to 10.



```
bivariate_distribution('previous')
```



```
/usr/local/lib/python3.6/dist-packages/seaborn/axisgrid.py:243: UserWarning: The `size`
warnings.warn(msg, UserWarning)
```



## ▼ 2.2 Explore Categorical Columns

```
def explore_categorical_column(title):
    train = pd.concat([Xtrain,ytrain],axis=1)

    sns.catplot(x=title,kind='count', hue="subscribed", palette='pastel', data=train)
```

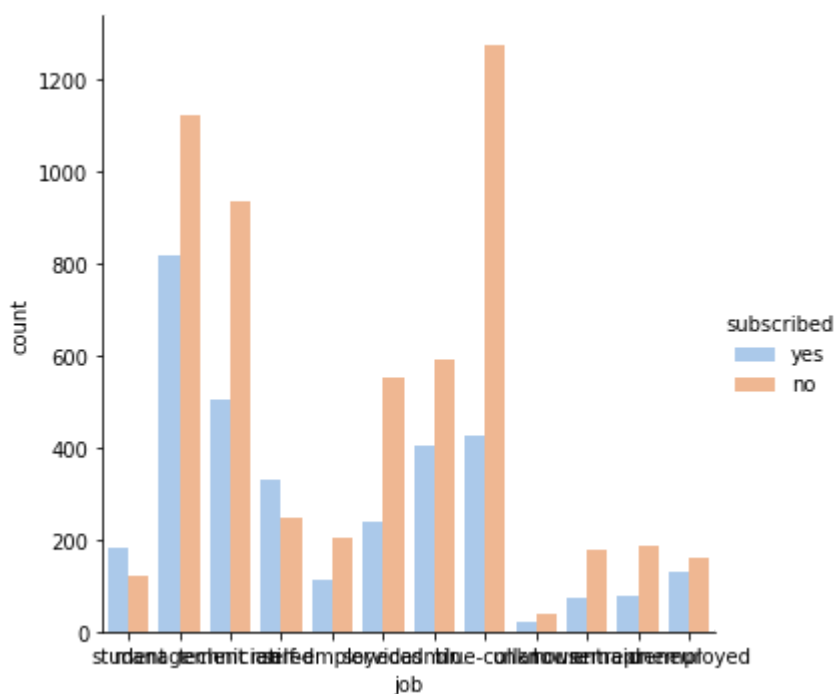
The subscribed and not subscribe customers will be categorized according to their job. According to the graph below, the highest number of customers who do not subscribe the term deposit are from 'blue-collar' job (1274) followed by management job (1122).

While for subscribed customers, the highest number of job is management (816) followed by technician (505).

```
explore_categorical_column("job")
```

```
job_train = pd.crosstab(Xtrain.job,ytrain)
print(job_train)
```

subscribed	no	yes
job		
admin.	591	404
blue-collar	1274	426
entrepreneur	186	79
housemaid	176	71
management	1122	816
retired	246	331
self-employed	203	111
services	550	240
student	119	180
technician	935	505
unemployed	159	129
unknown	40	22



According to the graph below, most of the customers (3398) who has married are not subscribe the term deposit plan. On the other hand, the number of the customers who are married that subscribed the term deposit also is the highest (1730) compared to single customers (1204) and divorced customers (380).

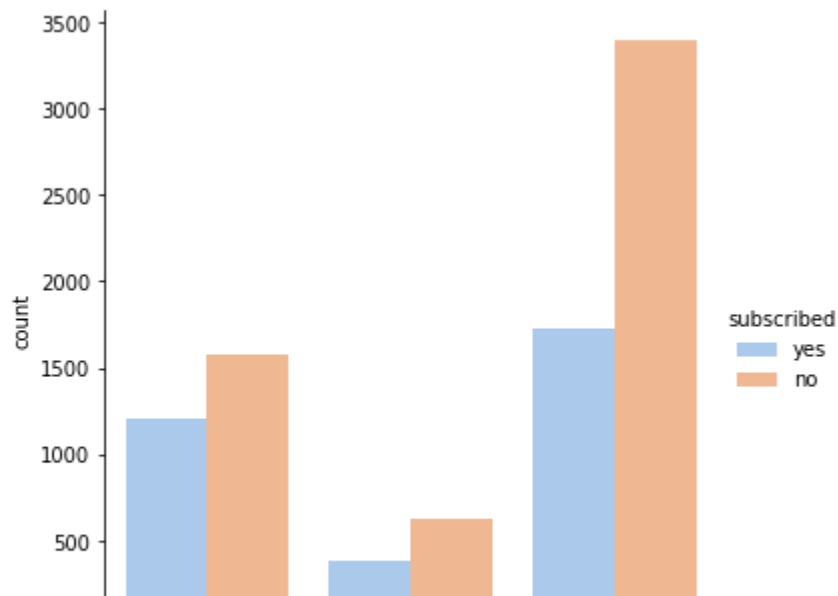
```
explore_categorical_column("marital")
```

```
marital_train = pd.crosstab(Xtrain.marital,ytrain)
print(marital_train)
```





subscribed	no	yes
marital		
divorced	622	380
married	3398	1730
single	1581	1204



The customers are then categorized according to their presence of credit in default. According to the graph, most of the customers (8785) do not have credit in default. 5504 of them are not subscribing the term deposit while 3281 of them have subscribed the term deposit.

```
explore_categorical_column("default")
```

```
default_train = pd.crosstab(Xtrain.default,ytrain)
print(default_train)
```



```

subscribed    no    yes
default
no            5504  3281

```

The subscribed and not subscribe customers will be categorized according to their housing loan.

According to the graph below, 2127 of customers that do not have housing loan are the person that subscribed the term deposit which are higher than customers that have housing loan (1187).

```

explore_categorical_column("housing")

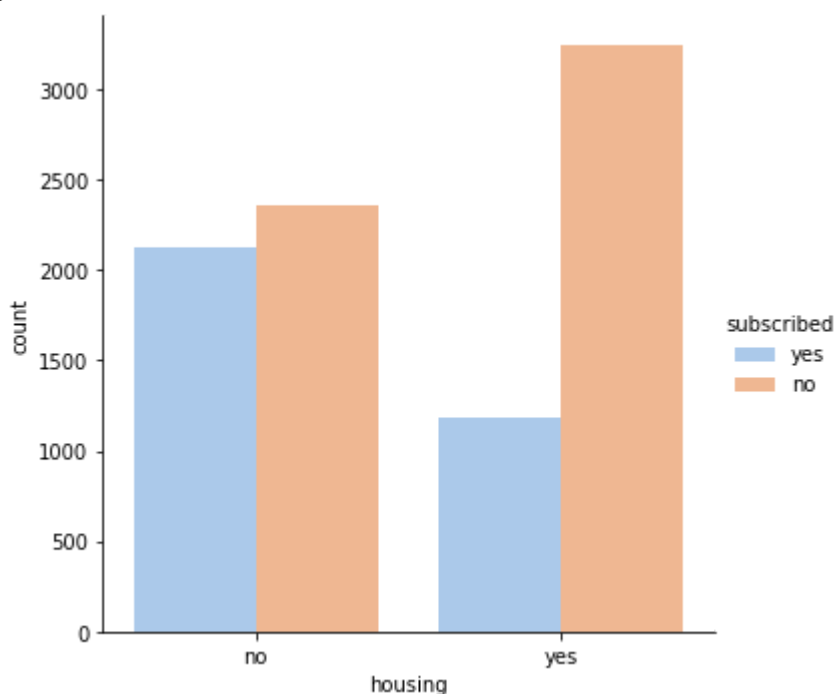
housing_train = pd.crosstab(Xtrain.housing,ytrain)
print(housing_train)

```

```

subscribed    no    yes
housing
no            2357  2127
yes           3244  1187

```



The customers are categorized according to the presence of personal loan. According to the graph, based on the customer that have subscribed the term deposit, most of the customers (3045) do not have housing loan. Only 269 customers have housing loan.

```

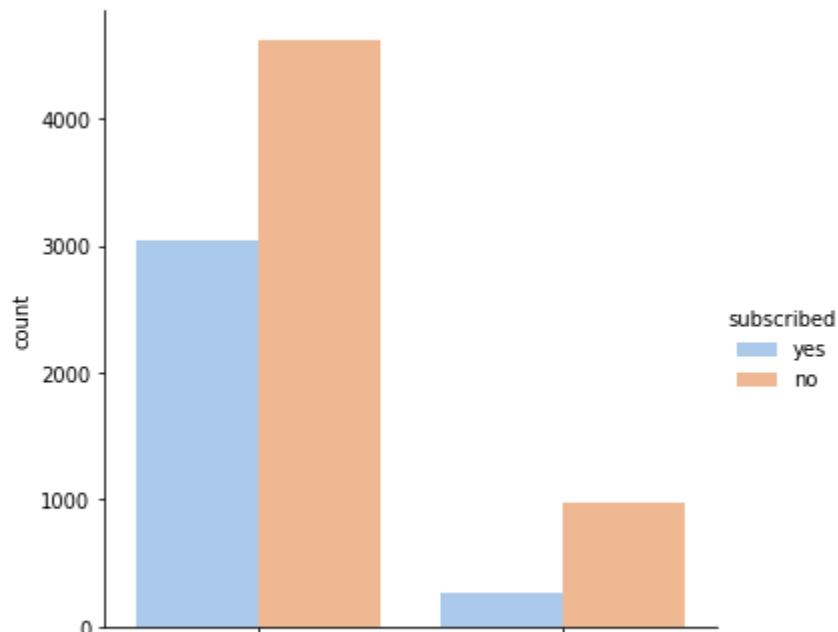
explore_categorical_column("loan")

loan_train = pd.crosstab(Xtrain.loan,ytrain)
print(loan_train)

```



	no	yes
loan		
no	4619	3045
yes	982	269



The customers are then categorized according to their contact communication type. According to the graph, the communication type will be classified into 3 types, which are telephone, cellular and unknown. From the total number of customers who subscribed the term deposit (3314), 2783 of customers who use cellular have subscribed the term deposit. While 236 and 295 customers use telephone and unknown communication gadget.

```
explore_categorical_column("contact")
```

```
contact_train = pd.crosstab(Xtrain.contact,ytrain)
print(contact_train)
```



subscribed	no	yes
contact		
cellular	3568	2783
telephone	385	236
unknown	1648	295



The subscribed and not subscribe customers will be categorized according to their last contact month of the year.

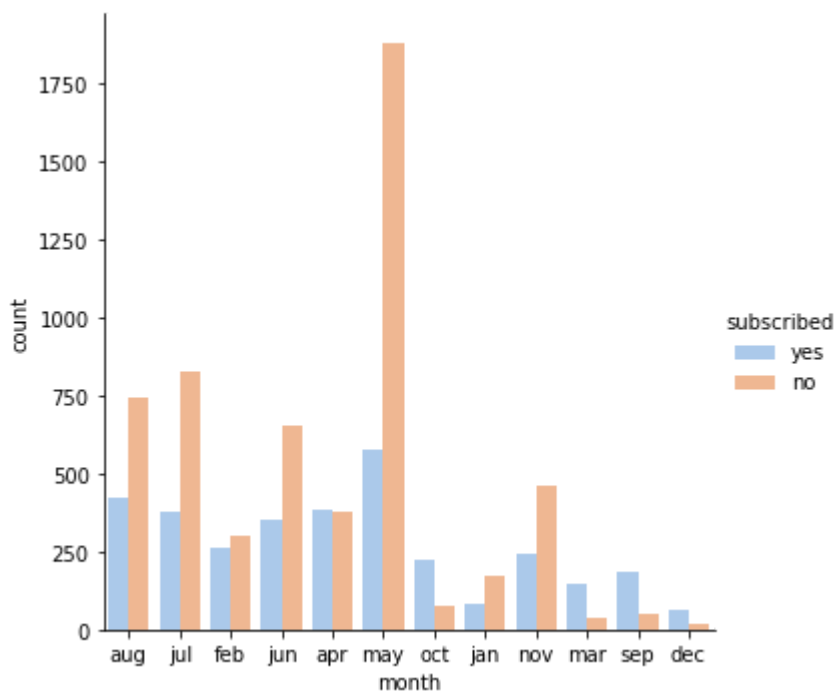
According to the graph below, the highest number of subscribed customers is from May of the year followed by August.



```
explore_categorical_column("month")
```

```
month_train = pd.crosstab(Xtrain.month,ytrain)
print(month_train)
```

subscribed	no	yes
month		
apr	381	385
aug	743	424
dec	19	64
feb	298	259
jan	173	80
jul	829	380
jun	656	349
mar	36	148
may	1881	574
nov	459	240
oct	77	225
sep	49	186



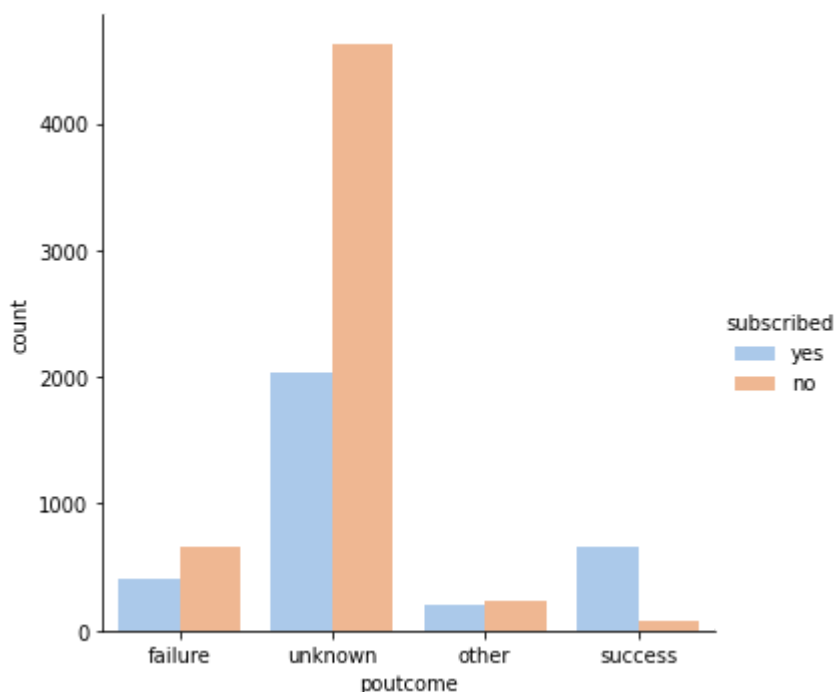
The subscribed and not subscribe customers will be categorized according to their outcome of the previous marketing campaign which are failure, success, unknown and other.

According to the graph below, from the 3314 customers who subscribed the term deposit, 2041 of customers are from the "unknown" category which is the highest number followed by "success" category (663).

```
explore_categorical_column("poutcome")
```

```
poutcome_train = pd.crosstab(Xtrain.poutcome,ytrain)
print(poutcome_train)
```

subscribed	no	yes
poutcome		
failure	664	409
other	236	201
success	77	663
unknown	4624	2041



## ▼ Chi-Square Test

Chi-Square Test is carried out to investigate the dependency between the both categorical x and y.

Based on the results below, the attribute job, marital, education, default, housing, loan, contact, month and poutcome are dependent to "subscribed" column.

```
job_train = pd.crosstab(Xtrain.job,ytrain)
marital_train = pd.crosstab(Xtrain.marital,ytrain)
```

```

edu_train = pd.crosstab(Xtrain.education,ytrain)
default_train = pd.crosstab(Xtrain.default,ytrain)
house_train = pd.crosstab(Xtrain.housing,ytrain)
loan_train = pd.crosstab(Xtrain.loan,ytrain)
contact_train = pd.crosstab(Xtrain.contact,ytrain)
month_train = pd.crosstab(Xtrain.month,ytrain)
poutcome_train = pd.crosstab(Xtrain.poutcome,ytrain)

# returns four values,  $\chi^2$  value, p-value, degree of freedom and expected values.

a = [job_train,marital_train,edu_train,default_train,house_train,loan_train,contact_train,mon

print("P values of every column")
n=1
for x in a:

    chi, pval, dof, exp = chi2_contingency(x)
    significance = 0.05
    print(n,'. -----',x.index.name,'-----

    print('p-value=%.6f, significance=%.2f\n' % (pval, significance))
    if pval < significance:
        print("""At %.2f level of significance, we reject the null hypotheses and accept H1.
        They are not independent.""" % (significance))
    else:
        print("""At %.2f level of significance, we accept the null hypotheses.
        They are independent.""" % (significance))

    # print(x.index.name," = " ,chi2_contingency(x)[1]) # print p values

    print(' -----\n
    n+=1

```



```
1 . ----- job -----
p-value=0.000000, significance=0.05
```

At 0.05 level of significance, we reject the null hypotheses and accept H1.  
They are not independent.

-----

```
2 . ----- marital -----
p-value=0.000000, significance=0.05
```

At 0.05 level of significance, we reject the null hypotheses and accept H1.  
They are not independent.

-----

```
3 . ----- education -----
p-value=0.000000, significance=0.05
```

At 0.05 level of significance, we reject the null hypotheses and accept H1.  
They are not independent.

-----

```
4 . ----- default -----
p-value=0.006720, significance=0.05
```

At 0.05 level of significance, we reject the null hypotheses and accept H1.  
They are not independent.

-----

```
5 . ----- housing -----
p-value=0.000000, significance=0.05
```

At 0.05 level of significance, we reject the null hypotheses and accept H1.  
They are not independent.

-----

```
6 . ----- loan -----
p-value=0.000000, significance=0.05
```

At 0.05 level of significance, we reject the null hypotheses and accept H1.  
They are not independent.

-----

```
7 . ----- contact -----
p-value=0.000000, significance=0.05
```

At 0.05 level of significance, we reject the null hypotheses and accept H1.

## ▼ 3.0 Data Preparation

```
8 . ----- month -----
```

## ▼ 3.1 Encode Variables

they are not independent.

The sklearn LabelEncoder function is utilized to encode the categorical variables such as job, marital, education, contact, poutcome, default, housing, loan, subscribed and month.

p-value=0.000000, significance=0.05

```
def column_encoding(df_x , df_y):

    df = pd.concat([df_x,df_y],axis=1)

    label_encoder = preprocessing.LabelEncoder()

    nominal_cols = ['job', 'marital','education' , 'contact', 'poutcome']
    for name in nominal_cols:
        df[name] = label_encoder.fit_transform(df[name])
        df[name].value_counts()

    # encoding 'default' , 'housing', 'loan' attributes
    # 1 is yes , 0 is no
    mapping_dictionary = {"default" : {"yes" : 1 , "no" : 0},
                          "housing" : {"yes" : 1 , "no" : 0},
                          "loan" : {"yes" : 1 , "no" : 0} ,
                          "subscribed" : {"yes" : 1 , "no" : 0}}

    df = df.replace(mapping_dictionary)

    # month
    replace_dictionary = { "month" : {"jan" : 1 ,
                                       "feb" : 2,
                                       "mar" : 3,
                                       "apr" : 4,
                                       "may" : 5,
                                       "jun" : 6,
                                       "jul" : 7,
                                       "aug" : 8,
                                       "sep" : 9,
                                       "oct" : 10,
                                       "nov" : 11,
                                       "dec" : 12}}

    df.replace(replace_dictionary , inplace=True)

    df_y = df.subscribed
    df_x = df.drop('subscribed', axis=1)
    return df_x , df_y
```

The train and test datasets are encoded separately.



```
Xtrain , ytrain = column_encoding(Xtrain , ytrain)
Xtest , ytest = column_encoding(Xtest , ytest)
```

After performing label encoding operation for both train and test datasets, the categorical values are converted to numerical values.

```
Xtrain.head()
```



	age	job	marital	education	default	balance	housing	loan	contact	day	mont
<b>16196</b>	22	8	2	1	0	948	0	0	1	18	
<b>31317</b>	53	4	0	2	0	3158	0	0	0	31	
<b>7815</b>	33	9	2	1	0	111	0	0	0	28	
<b>11929</b>	59	5	1	0	0	866	0	1	0	2	
<b>5429</b>	40	6	2	2	0	10346	0	0	0	15	

## ▼ 3.2 Data Cleaning (For Train Dataset)

### ▼ Missing Value

From the result shown in below, there is no missing value as the number of null values in each column is 0.

```
# handle missing value
null_counts = Xtrain.isnull().sum()
print("Number of null values in each column:\n{}".format(null_counts))

# conclusion : no missing value
```



Number of null values in each column:

```
age          0
job          0
marital      0
education    0
default      0
```

## ▼ Remove Duplicated Value

```
contact      0
```

From the result shown below, there is no duplicated data as the data size is the same.

```
duration     0
```

```
# Detect duplicate data
Xtrain_dedupped = Xtrain.drop_duplicates()

print(Xtrain.shape)
print(Xtrain_dedupped.shape)
```

```
(8915, 16)
(8915, 16)
```

## ▼ Remove Outliers

The method used to remove the outliers is called Z-score. In this particular case, a threshold of 3 is used and if the Z-score value is greater than 3, that data point will be identified as outliers. From the table below, there are a total of 7989 rows that have the Z-score which is lower than the threshold value which is 3. Hence, all the selected rows will be used in the following step.

```
# remove outliers

from scipy import stats

X = pd.concat([Xtrain,ytrain],axis=1)

X = X[(np.abs(stats.zscore(X)) < 3).all(axis=1)]

Xtrain = X.iloc[:, :-1] #drop the last column
ytrain = X.iloc[:, -1] #choose the last column
print(X)
print(X.shape)
```



	age	job	marital	education	...	pdays	previous	poutcome	subscribed
16196	22	8	2	1	...	197	2	0	1
11929	59	5	1	0	...	208	1	1	0
5429	40	6	2	2	...	207	1	1	1
2435	31	4	2	2	...	-1	0	3	1
12294	51	7	1	1	...	-1	0	3	0
...	...	...	...	...	...	...	...	...	...
4497	39	1	1	1	...	-1	0	3	0
-----	--	-	-	-	-	-	-	-	-

The code below is used to split the Xtrain into 2 different categories which consists of categorical data and numerical data.

```
categorical_var_train = Xtrain[['contact','education','default','housing','loan','job','poutc
'month']]

numerical_var_train = Xtrain.drop(['contact','education','default','housing','loan','job','po
'month'],axis=1)

categorical_var_test = Xtest[['contact','education','default','housing','loan','job','poutcom
'month']]

numerical_var_test = Xtest.drop(['contact','education','default','housing','loan','job','pout
'month'],axis=1)
```

Since there are numerical data (continuous) in the table, StandardScaler is used to scale each input variable separately by subtracting the mean and dividing by the standard deviation, in order to have a distribution of mean of 0 and a standard deviation of 1.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_Xtrain = scaler.fit_transform(numerical_var_train)
scaled_Xtest = scaler.transform(numerical_var_test)
```

After the continuous data is scaled, it is then joined back with the categorical dataframe for later model prediction.

```
scaled_Xtrain = pd.DataFrame(scaled_Xtrain)
scaled_Xtrain = scaled_Xtrain.reset_index()
categorical_var_train = categorical_var_train.reset_index()

scaled_Xtest = pd.DataFrame(scaled_Xtest)
scaled_Xtest = scaled_Xtest.reset_index()
categorical_var_test = categorical_var_test.reset_index()
```

```
Xtrain = pd.concat([scaled_Xtrain,categorical_var_train],axis=1)
```

```
Xtrain = Xtrain.drop(['index','index'],axis=1)

Xtest = pd.concat([scaled_Xtest,categorical_var_test],axis=1)
Xtest = Xtest.drop(['index','index'],axis=1)

Xtrain.head(-20)
```



	0	1	2	3	4	5	6	contact	€
0	-1.675779	-0.150581	0.316734	-0.363216	-0.728189	1.522156	1.039378	1	
1	1.628902	-0.193918	-1.613019	-0.857210	-0.728189	1.633486	0.295262	0	
2	-0.068096	4.816268	-0.045095	0.710347	-0.162504	1.623365	0.295262	0	
3	-0.871938	0.204043	1.764049	0.072433	-0.162504	-0.481787	-0.448854	0	
4	0.914376	-1.251976	1.161001	-0.596599	-0.728189	-0.481787	-0.448854	2	
...	...	...	...	...	...	...	...	...	...
7964	-0.514675	-0.328686	-1.733628	-0.013141	-0.162504	0.752965	0.295262	0	
7965	-0.693306	-0.456583	1.161001	-0.876659	-0.728189	0.459459	1.783494	0	
7966	-0.157412	-0.651599	-1.251190	1.955057	-0.162504	-0.481787	-0.448854	0	
7967	-0.157412	0.545983	0.316734	-0.390444	0.968865	-0.481787	-0.448854	0	
7968	-0.425359	1.990904	0.075515	-0.511025	1.534549	-0.481787	-0.448854	2	

7969 rows × 16 columns

## ▼ 4.0 Modeling

### ▼ 4.1 Naive Bayes

Naive Bayes is a simple "probabilistic classifiers" which based on applying Bayes' theorem with strong (naïve) independence assumptions between the features (X) and it is useful for very large dataset. In this section, GaussianNB is imported from sklearn and Xtrain, ytrain are being fitted into the model in order to do prediction.

```
from sklearn.naive_bayes import GaussianNB      # 1. choose model class
naive_model = GaussianNB()                     # 2. instantiate model
```

Fit the training sets to the model.

```
naive_model.fit(xtrain, ytrain) # 3. fit model to data
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

Predict the testing sets.

```
y_naive_model_model = naive_model.predict(Xtest)
```

The accuracy score of the Gaussian Naive Bayes is 76.22% which is consider acceptable.

```
from sklearn.metrics import accuracy_score
#accuracy_score(ytest, y_naive_model_model)
print(f"Accuracy : {accuracy_score(ytest, y_naive_model_model)*100} %" )
```

```
Accuracy : 76.22252131000448 %
```

## ▼ 4.2 K-Nearest Neighbours (KNN)

KNN is an algorithm which it is non-parametric and lazy (instance based) because it doesn't have a specialized training phase. In this section, the grid search algorithm is used to find the best parameters for the k values in order to have the best accuracy. In this case, k value of 14 provides the highest accuracy score.

```
from sklearn.neighbors import KNeighborsClassifier

def grid_search_knn():

    # Grid Search to find the best parameters
    k_range = list(range(1,31))
    weight_options = ["uniform", "distance"]
    param_grid = dict(n_neighbors = k_range, weights = weight_options)

    knn = KNeighborsClassifier()
    grid = GridSearchCV(knn, param_grid, cv = 10, scoring = 'accuracy')
    grid.fit(Xtrain,ytrain)

    print (grid.best_score_)
    print (grid.best_params_)
    print (grid.best_estimator_)

grid_search_knn()
```



```
0.8068589809944134
{'n_neighbors': 10, 'weights': 'distance'}
```

Fit the training sets to the model.

```
knn = KNeighborsClassifier(n_neighbors = 14)
```

```
knn.fit(Xtrain, ytrain)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                      metric_params=None, n_jobs=None, n_neighbors=14, p=2,
                      weights='uniform')
```

Predict the testing sets.

```
knn_model = knn.predict(Xtest)
```

The accuracy score for KNN model is 79.45% which is consider acceptable.

```
print(f"Accuracy : {accuracy_score(ytest, knn_model)*100} %" )
```

```
Accuracy : 79.4526693584567 %
```

## ▼ 4.3 Decision Tree

Randomized search is carried out to find the optimal hyperparameter value for Decision Tree model.

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

```
def randomize_search_decision_tree():
```

```
    X_train , X_test , y_train , y_test = Xtrain , Xtest , ytrain , ytest
```

```
    max_depth = list(range(1, 50))
    min_samples_leaf = list(range(1, 60))
    min_samples_split = list(range(2,50))
    max_features = list(range(1, X_train.shape[1]))
    criterion = ['entropy' , 'gini']
```

```
    decision_tree_model = DecisionTreeClassifier()
```

```
    #carry out randomized search
    parameter_grid = dict(criterion=criterion,
```

```

parameter_grid = {
    'max_features':max_features,
    'min_samples_leaf':min_samples_leaf,
    'max_depth':max_depth,
    'min_samples_split':min_samples_split)

grid = RandomizedSearchCV(estimator=decision_tree_model, param_distributions=parameter_grid)
grid.fit(X_train,y_train)

print("Best criterion : " , grid.best_estimator_.criterion)
print("Best max_features" , grid.best_estimator_.max_features)
print("Best min_samples_leaf : " , grid.best_estimator_.min_samples_leaf)
print("Best max_depth : " , grid.best_estimator_.max_depth )
print("Best min_samples_split : " , grid.best_estimator_.min_samples_split)

```

The result of randomized search is applied to the Decision Tree model.

```

criterion = "gini"
max_features = 10
min_samples_leaf = 41
max_depth = 35
min_samples_split = 41

# result of randomized search


X_train , X_test , y_train , y_test = Xtrain , Xtest , ytrain , ytest

decision_tree_model = DecisionTreeClassifier(criterion=criterion,
    max_features=max_features,
    min_samples_leaf=min_samples_leaf,
    max_depth=max_depth,
    min_samples_split=min_samples_split,
    random_state = 50
)
decision_tree_model.fit(X_train, y_train)

# Prediction on test dataset
y_pred_decision_tree = decision_tree_model.predict(X_test)

print(f"Accuracy : {decision_tree_model.score(X_test,y_test)*100} %" )

```

 Accuracy : 83.22117541498429 %

## ▼ 4.4 Random Forest

Carried out randomized search to find the optimal hyperparameter value for Random Forest model.

```

from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier

def randomize_search_random_forest():

    X_train , X_test , y_train , y_test = Xtrain , Xtest , ytrain , ytest

    n_estimators = [100,200,300,400,500,600,700,800,900]
    max_features = ['auto', 'sqrt']
    max_depth = [10,20,30,40,50,60,70,80,90,100]
    min_samples_split = [10,20,30,40,50]
    min_samples_leaf = [10,20,30,40,50]
    bootstrap = [True, False]
    max_leaf_nodes = [2,4,6,8,10,20,30,40,50]

    random_forest_model = RandomForestClassifier()

    # carry out randomized search
    parameter_grid = dict(n_estimators=n_estimators,
                          max_features=max_features,
                          min_samples_leaf=min_samples_leaf,
                          max_depth=max_depth,
                          min_samples_split=min_samples_split,
                          bootstrap=bootstrap,
                          max_leaf_nodes=max_leaf_nodes)

    grid = RandomizedSearchCV(estimator=random_forest_model, param_distributions=parameter_grid
    grid.fit(X_train,y_train)

    # print hyperparameter values
    print("Best n_estimators : " , grid.best_estimator_.n_estimators)
    print("Best max_features : " , grid.best_estimator_.max_features)
    print("Best min_samples_leaf : " , grid.best_estimator_.min_samples_leaf)
    print("Best max_depth : " , grid.best_estimator_.max_depth)
    print("Best min_samples_split : " , grid.best_estimator_.min_samples_split)
    print("Best bootstrap : " , grid.best_estimator_.bootstrap)
    print("Best max_leaf_nodes : " , grid.best_estimator_.max_leaf_nodes)

```

The optimal hyperparameter values are shown below. The will be applied to the Random Forest model.

```

n_estimators = 500
max_features = "auto"
min_samples_leaf = 30
max_depth = 30
min_samples_split = 40
bootstrap = "True"
max_leaf_nodes = 50

```




Performing training for the Random Forest model.

```
# result of randomized search
random_forest_model = RandomForestClassifier(n_estimators=n_estimators ,
      max_features=max_features ,
      min_samples_leaf=min_samples_leaf,
      max_depth=max_depth,
      min_samples_split=min_samples_split ,
      bootstrap=bootstrap,
      max_leaf_nodes=max_leaf_nodes,
      random_state = 50)
random_forest_model.fit(X_train, y_train)

# Prediction on test with giniIndex
y_pred_random_forest = random_forest_model.predict(X_test)

print(f"Accuracy : {random_forest_model.score(X_test,y_test)*100} %" )
```

 Accuracy : 83.80439659039928 %

## ▼ 4.5 Logistic Regression

For Logistic Regression, a logistic regression classifier is implemented. Hyperparameters are tuned using GridSearchCV and model is then fit to training data.

```
# Instantiate classifier
logistic_regression = LogisticRegression(random_state = 30)

# Set up hyperparameter grid for tuning
logistic_regression_param_grid = {'C' : [0.0001, 0.001, 0.01, 0.05, 0.1] }

# Tune hyperparameters
logistic_regression_model = GridSearchCV(logistic_regression, param_grid = logistic_regressio

# Fit model to training data
logistic_regression_model.fit(Xtrain, ytrain)
```




```
GridSearchCV(cv=5, error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
             fit_intercept=True,
```

Predict the test set results and calculate the accuracy.

```

n_jobs=None, n_estimators=100,
# Predict test data on logistic regression
print(f"Accuracy : {logistic_regression_model.score(Xtest, ytest)*100} %" )

# Obtain model performance metrics
lr_pred_prob = logistic_regression_model.predict_proba(Xtest)[:,-1]
lr_aucroc = roc_auc_score(ytest, lr_pred_prob)
```

 Accuracy : 79.58725886047556 %

## ▼ 4.6 Artificial Neural Network (ANN)

The ANN architecture is set up with 2 hidden layers (9 neurons, ReLU activation) and output layer (1 neuron, Sigmoid activation).

```
# Create an object of sequential model
ann_classifier = Sequential()
# Add the first hidden layer
ann_classifier.add(Dense(9, activation = 'relu', input_dim = 16))
# Adding the second hidden layer
ann_classifier.add(Dense(9, activation= 'relu'))
# Adding the output layer
ann_classifier.add(Dense(1, activation = 'sigmoid'))
```

Configure the model and train it over 100 epochs.

```
ann_classifier.compile(optimizer = 'sgd', loss = 'binary_crossentropy', metrics = ['accuracy']

# Model is trained over 100 epochs
ann_model = ann_classifier.fit(Xtrain, ytrain, validation_split = 0.33, batch_size = 10, epochs = 100)
```



```
Epoch 1/100
536/536 [=====] - 1s 2ms/step - loss: 0.6472 - accuracy: 0.6401
Epoch 2/100
536/536 [=====] - 1s 1ms/step - loss: 0.5882 - accuracy: 0.6766
Epoch 3/100
536/536 [=====] - 1s 1ms/step - loss: 0.5217 - accuracy: 0.7345
Epoch 4/100
536/536 [=====] - 1s 1ms/step - loss: 0.4708 - accuracy: 0.7668
Epoch 5/100
536/536 [=====] - 1s 1ms/step - loss: 0.4480 - accuracy: 0.7846
Epoch 6/100
536/536 [=====] - 1s 1ms/step - loss: 0.4379 - accuracy: 0.7935
Epoch 7/100
536/536 [=====] - 1s 1ms/step - loss: 0.4346 - accuracy: 0.7958
Epoch 8/100
536/536 [=====] - 1s 1ms/step - loss: 0.4282 - accuracy: 0.8018
Epoch 9/100
536/536 [=====] - 1s 1ms/step - loss: 0.4262 - accuracy: 0.8021
Epoch 10/100
536/536 [=====] - 1s 1ms/step - loss: 0.4264 - accuracy: 0.8006
Epoch 11/100
536/536 [=====] - 1s 1ms/step - loss: 0.4247 - accuracy: 0.8015
Epoch 12/100
536/536 [=====] - 1s 1ms/step - loss: 0.4206 - accuracy: 0.8096
Epoch 13/100
536/536 [=====] - 1s 1ms/step - loss: 0.4215 - accuracy: 0.8061
Epoch 14/100
536/536 [=====] - 1s 1ms/step - loss: 0.4207 - accuracy: 0.8059
Epoch 15/100
536/536 [=====] - 1s 1ms/step - loss: 0.4187 - accuracy: 0.8059
Epoch 16/100
536/536 [=====] - 1s 1ms/step - loss: 0.4168 - accuracy: 0.8074
Epoch 17/100
536/536 [=====] - 1s 1ms/step - loss: 0.4185 - accuracy: 0.8089
Epoch 18/100
536/536 [=====] - 1s 1ms/step - loss: 0.4182 - accuracy: 0.8075
Epoch 19/100
536/536 [=====] - 1s 1ms/step - loss: 0.4167 - accuracy: 0.8081
Epoch 20/100
536/536 [=====] - 1s 1ms/step - loss: 0.4164 - accuracy: 0.8077
Epoch 21/100
536/536 [=====] - 1s 1ms/step - loss: 0.4160 - accuracy: 0.8094
Epoch 22/100
536/536 [=====] - 1s 1ms/step - loss: 0.4139 - accuracy: 0.8102
Epoch 23/100
536/536 [=====] - 1s 1ms/step - loss: 0.4137 - accuracy: 0.8066
Epoch 24/100
536/536 [=====] - 1s 1ms/step - loss: 0.4135 - accuracy: 0.8096
Epoch 25/100
536/536 [=====] - 1s 1ms/step - loss: 0.4131 - accuracy: 0.8092
Epoch 26/100
536/536 [=====] - 1s 1ms/step - loss: 0.4117 - accuracy: 0.8085
Epoch 27/100
536/536 [=====] - 1s 1ms/step - loss: 0.4109 - accuracy: 0.8083
Epoch 28/100
536/536 [=====] - 1s 1ms/step - loss: 0.4121 - accuracy: 0.8102
Epoch 29/100
```

```
536/536 [=====] - 1s 1ms/step - loss: 0.4107 - accuracy: 0.8098
Epoch 30/100
536/536 [=====] - 1s 1ms/step - loss: 0.4111 - accuracy: 0.8096
Epoch 31/100
536/536 [=====] - 1s 1ms/step - loss: 0.4108 - accuracy: 0.8083
Epoch 32/100
536/536 [=====] - 1s 1ms/step - loss: 0.4104 - accuracy: 0.8081
Epoch 33/100
536/536 [=====] - 1s 1ms/step - loss: 0.4094 - accuracy: 0.8074
Epoch 34/100
536/536 [=====] - 1s 1ms/step - loss: 0.4092 - accuracy: 0.8079
Epoch 35/100
536/536 [=====] - 1s 1ms/step - loss: 0.4080 - accuracy: 0.8113
Epoch 36/100
536/536 [=====] - 1s 1ms/step - loss: 0.4084 - accuracy: 0.8064
Epoch 37/100
536/536 [=====] - 1s 1ms/step - loss: 0.4080 - accuracy: 0.8089
Epoch 38/100
536/536 [=====] - 1s 1ms/step - loss: 0.4080 - accuracy: 0.8081
Epoch 39/100
536/536 [=====] - 1s 1ms/step - loss: 0.4088 - accuracy: 0.8076
Epoch 40/100
536/536 [=====] - 1s 1ms/step - loss: 0.4053 - accuracy: 0.8092
Epoch 41/100
536/536 [=====] - 1s 1ms/step - loss: 0.4060 - accuracy: 0.8096
Epoch 42/100
536/536 [=====] - 1s 1ms/step - loss: 0.4083 - accuracy: 0.8092
Epoch 43/100
536/536 [=====] - 1s 1ms/step - loss: 0.4069 - accuracy: 0.8062
Epoch 44/100
536/536 [=====] - 1s 1ms/step - loss: 0.4064 - accuracy: 0.8089
Epoch 45/100
536/536 [=====] - 1s 1ms/step - loss: 0.4063 - accuracy: 0.8104
Epoch 46/100
536/536 [=====] - 1s 1ms/step - loss: 0.4059 - accuracy: 0.8096
Epoch 47/100
536/536 [=====] - 1s 1ms/step - loss: 0.4052 - accuracy: 0.8089
Epoch 48/100
536/536 [=====] - 1s 1ms/step - loss: 0.4059 - accuracy: 0.8057
Epoch 49/100
536/536 [=====] - 1s 1ms/step - loss: 0.4053 - accuracy: 0.8113
Epoch 50/100
536/536 [=====] - 1s 1ms/step - loss: 0.4053 - accuracy: 0.8061
Epoch 51/100
536/536 [=====] - 1s 1ms/step - loss: 0.4056 - accuracy: 0.8102
Epoch 52/100
536/536 [=====] - 1s 1ms/step - loss: 0.4050 - accuracy: 0.8122
Epoch 53/100
536/536 [=====] - 1s 1ms/step - loss: 0.4043 - accuracy: 0.8106
Epoch 54/100
536/536 [=====] - 1s 1ms/step - loss: 0.4046 - accuracy: 0.8092
Epoch 55/100
536/536 [=====] - 1s 1ms/step - loss: 0.4042 - accuracy: 0.8092
Epoch 56/100
536/536 [=====] - 1s 1ms/step - loss: 0.4024 - accuracy: 0.8111
Epoch 57/100
536/536 [=====] - 1s 1ms/step - loss: 0.4036 - accuracy: 0.8102
Epoch 58/100
```

```
536/536 [=====] - 1s 1ms/step - loss: 0.4023 - accuracy: 0.8081
Epoch 59/100
536/536 [=====] - 1s 1ms/step - loss: 0.4020 - accuracy: 0.8105
Epoch 60/100
536/536 [=====] - 1s 1ms/step - loss: 0.4025 - accuracy: 0.8094
Epoch 61/100
536/536 [=====] - 1s 1ms/step - loss: 0.4015 - accuracy: 0.8083
Epoch 62/100
536/536 [=====] - 1s 1ms/step - loss: 0.4022 - accuracy: 0.8074
Epoch 63/100
536/536 [=====] - 1s 1ms/step - loss: 0.4007 - accuracy: 0.8087
Epoch 64/100
536/536 [=====] - 1s 1ms/step - loss: 0.4008 - accuracy: 0.8096
Epoch 65/100
536/536 [=====] - 1s 1ms/step - loss: 0.3999 - accuracy: 0.8094
Epoch 66/100
536/536 [=====] - 1s 1ms/step - loss: 0.4012 - accuracy: 0.8105
Epoch 67/100
536/536 [=====] - 1s 1ms/step - loss: 0.4000 - accuracy: 0.8104
Epoch 68/100
536/536 [=====] - 1s 1ms/step - loss: 0.4009 - accuracy: 0.8105
Epoch 69/100
536/536 [=====] - 1s 1ms/step - loss: 0.4002 - accuracy: 0.8085
Epoch 70/100
536/536 [=====] - 1s 1ms/step - loss: 0.3984 - accuracy: 0.8081
Epoch 71/100
536/536 [=====] - 1s 1ms/step - loss: 0.3984 - accuracy: 0.8111
Epoch 72/100
536/536 [=====] - 1s 1ms/step - loss: 0.3991 - accuracy: 0.8102
Epoch 73/100
536/536 [=====] - 1s 1ms/step - loss: 0.3961 - accuracy: 0.8128
Epoch 74/100
536/536 [=====] - 1s 1ms/step - loss: 0.3988 - accuracy: 0.8104
Epoch 75/100
536/536 [=====] - 1s 1ms/step - loss: 0.3998 - accuracy: 0.8122
Epoch 76/100
536/536 [=====] - 1s 1ms/step - loss: 0.3990 - accuracy: 0.8136
Epoch 77/100
536/536 [=====] - 1s 1ms/step - loss: 0.3976 - accuracy: 0.8128
Epoch 78/100
536/536 [=====] - 1s 1ms/step - loss: 0.3986 - accuracy: 0.8098
Epoch 79/100
536/536 [=====] - 1s 1ms/step - loss: 0.3982 - accuracy: 0.8085
Epoch 80/100
536/536 [=====] - 1s 1ms/step - loss: 0.3964 - accuracy: 0.8117
Epoch 81/100
536/536 [=====] - 1s 1ms/step - loss: 0.3971 - accuracy: 0.8117
Epoch 82/100
536/536 [=====] - 1s 1ms/step - loss: 0.3959 - accuracy: 0.8107
Epoch 83/100
536/536 [=====] - 1s 1ms/step - loss: 0.3964 - accuracy: 0.8135
Epoch 84/100
536/536 [=====] - 1s 1ms/step - loss: 0.3967 - accuracy: 0.8128
Epoch 85/100
536/536 [=====] - 1s 1ms/step - loss: 0.3973 - accuracy: 0.8126
Epoch 86/100
536/536 [=====] - 1s 1ms/step - loss: 0.3955 - accuracy: 0.8113
Epoch 87/100
```

```

Epoch 87/100
536/536 [=====] - 1s 1ms/step - loss: 0.3967 - accuracy: 0.8139
Epoch 88/100
536/536 [=====] - 1s 1ms/step - loss: 0.3953 - accuracy: 0.8154
Epoch 89/100
536/536 [=====] - 1s 1ms/step - loss: 0.3945 - accuracy: 0.8154
Epoch 90/100
536/536 [=====] - 1s 1ms/step - loss: 0.3945 - accuracy: 0.8141
Epoch 91/100
536/536 [=====] - 1s 1ms/step - loss: 0.3933 - accuracy: 0.8154
Epoch 92/100
536/536 [=====] - 1s 1ms/step - loss: 0.3942 - accuracy: 0.8186
Epoch 93/100

```

Predict the Xtest and ytest and calculate the ANN model score.

```
536/536 [=====] - 1s 1ms/step - loss: 0.3957 - accuracy: 0.8139
```

```

# Predict probabilities for test set
ann_y_predict = ann_classifier.predict(Xtest)
# Predict crisp classes for test set
ann_y_classes = ann_classifier.predict_classes(Xtest)
# Reduce to 1d array
ann_y_classes = ann_y_classes[:, 0]
# Model score calculation
ann_score = accuracy_score(ann_y_predict.astype('int'), ytest.astype('int'))
print(f"Accuracy : {ann_score*100} %" )

```



WARNING:tensorflow:From <ipython-input-57-6c9887e466a0>:4: Sequential.predict\_classes (1 Instructions for updating:  
Please use instead: \*`np.argmax(model.predict(x), axis=-1)`\*, if your model does multi-  
Accuracy : 61.86630776132795 %

## ▼ 4.7 Gradient Boosting Classification

The Gradient Boosting classifier is implemented to carry out gradient boosting classification. The best learning rate is selected by setting different rate to achieve the best performance.

```

# Import necessary libraries
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import GradientBoostingClassifier

# Scalling the data
scaler = MinMaxScaler()
X_train = scaler.fit_transform(Xtrain)
X_test = scaler.transform(Xtest)

# Set different learning rates to retrieve the best rate on performance
lr_list = [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]

for learning rate in lr_list:


```

```

gb_clf = GradientBoostingClassifier(n_estimators=20, learning_rate=learning_rate, max_fea
gb_clf.fit(Xtrain, ytrain)

print("Learning rate: ", learning_rate)
print("Accuracy score (training): {0:.3f}".format(gb_clf.score(Xtrain, ytrain)))
print("Accuracy score (validation): {0:.3f}".format(gb_clf.score(Xtest, ytest)))

```

 Learning rate: 0.05  
 Accuracy score (training): 0.675  
 Accuracy score (validation): 0.658  
 Learning rate: 0.075  
 Accuracy score (training): 0.716  
 Accuracy score (validation): 0.696  
 Learning rate: 0.1  
 Accuracy score (training): 0.748  
 Accuracy score (validation): 0.731  
 Learning rate: 0.25  
 Accuracy score (training): 0.799  
 Accuracy score (validation): 0.785  
 Learning rate: 0.5  
 Accuracy score (training): 0.820  
 Accuracy score (validation): 0.810  
 Learning rate: 0.75  
 Accuracy score (training): 0.829  
 Accuracy score (validation): 0.813  
 Learning rate: 1  
 Accuracy score (training): 0.824  
 Accuracy score (validation): 0.819

The learning rate, 1 is selected as a parameter of the gradient boosting model created. After that, the Xtrain and ytrain datasets are fit into the model created and make prediction on Xtest dataset.

```


# The best learning rate is 1 to fit into the classifier
gradient_boosting_model = GradientBoostingClassifier(n_estimators=20, learning_rate=1, max_fe
gradient_boosting_model.fit(Xtrain, ytrain)
y_pred_gradient_boosting = gradient_boosting_model.predict(Xtest)
gbScore = gradient_boosting_model.score(Xtest, ytest)
gbMatrix = confusion_matrix(ytest, y_pred_gradient_boosting)

```

```

# Output of accuracy
print(f"Accuracy : {gbScore*100} %")

```

 Accuracy : 81.87528039479587 %

## ▼ 4.8 Support Vector Machine (SVM)

A linear SVM is chosen by using the SVC classifier to make prediction on Xtest dataset by fitting the Xtrain and ytrain datasets into the SVM model. Accuracy is calculated and displayed by using score

syntax.

```
# Import necessary library
from sklearn.svm import SVC

# Create a linear SVM classifier
svm_model = SVC(kernel='linear', probability=True)

# Train classifier
svm_model.fit(Xtrain, ytrain)

# Take the model that was trained on the Xtrain data and apply it to the Xtest
y_pred_svm = svm_model.predict(Xtest)

# Calculation of accuracy
svmScore = svm_model.score(Xtest, ytest)

# Calculation of confusion matrix
svmMatrix = confusion_matrix(ytest, y_pred_svm)

# Print output
print(f"Accuracy : {svmScore*100} %")
```



Accuracy : 79.31807985643786 %

## ▼ 5.0 Evaluation

The evaluation method used involving confusion matrix, precision-recall curve and also learning curve.

### ▼ 5.1 Learning Curve

Learning curves are plots that show changes in learning performance over time and it can be used to diagnose an underfit, overfit, or well-fit model based on the training and validation datasets.

The section below shows the learning curve, scalability and performance graphs of different models.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.datasets import load_digits
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit
```



```
def plot_learning_curve(estimator, title, X, y, axes=None, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Generate 3 plots: the test and training learning curve, the training
    samples vs fit times curve, the fit times vs score curve.

    Parameters
    -----
    estimator : object type that implements the "fit" and "predict" methods
        An object of that type which is cloned for each validation.

    title : string
        Title for the chart.

    X : array-like, shape (n_samples, n_features)
        Training vector, where n_samples is the number of samples and
        n_features is the number of features.

    y : array-like, shape (n_samples) or (n_samples, n_features), optional
        Target relative to X for classification or regression;
        None for unsupervised learning.

    axes : array of 3 axes, optional (default=None)
        Axes to use for plotting the curves.

    ylim : tuple, shape (ymin, ymax), optional
        Defines minimum and maximum yvalues plotted.

    cv : int, cross-validation generator or an iterable, optional
        Determines the cross-validation splitting strategy.
        Possible inputs for cv are:

        - None, to use the default 5-fold cross-validation,
        - integer, to specify the number of folds.
        - :term:`CV splitter`,
        - An iterable yielding (train, test) splits as arrays of indices.

    For integer/None inputs, if ``y`` is binary or multiclass,
    :class:`StratifiedKFold` used. If the estimator is not a classifier
    or if ``y`` is neither binary nor multiclass, :class:`KFold` is used.

    Refer :ref:`User Guide <cross_validation>` for the various
    cross-validators that can be used here.

    n_jobs : int or None, optional (default=None)
        Number of jobs to run in parallel.
        ``None`` means 1 unless in a :obj:`joblib.parallel_backend` context.
        ``-1`` means using all processors. See :term:`Glossary <n_jobs>`
        for more details.
```

```

train_sizes : array-like, shape (n_ticks,), dtype float or int
    Relative or absolute numbers of training examples that will be used to
    generate the learning curve. If the dtype is float, it is regarded as a
    fraction of the maximum size of the training set (that is determined
    by the selected validation method), i.e. it has to be within (0, 1].
    Otherwise it is interpreted as absolute sizes of the training sets.
    Note that for classification the number of samples usually have to
    be big enough to contain at least one sample from each class.
    (default: np.linspace(0.1, 1.0, 5))
"""
if axes is None:
    _, axes = plt.subplots(1, 3, figsize=(20, 5))

axes[0].set_title(title)
if ylim is not None:
    axes[0].set_ylim(*ylim)
axes[0].set_xlabel("Training examples")
axes[0].set_ylabel("Score")

train_sizes, train_scores, test_scores, fit_times, _ = \
    learning_curve(estimator, X, y, cv=cv, n_jobs=n_jobs,
                   train_sizes=train_sizes,
                   return_times=True)
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)
fit_times_mean = np.mean(fit_times, axis=1)
fit_times_std = np.std(fit_times, axis=1)

# Plot learning curve
axes[0].grid()
axes[0].fill_between(train_sizes, train_scores_mean - train_scores_std,
                    train_scores_mean + train_scores_std, alpha=0.1,
                    color="r")
axes[0].fill_between(train_sizes, test_scores_mean - test_scores_std,
                    test_scores_mean + test_scores_std, alpha=0.1,
                    color="g")
axes[0].plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
axes[0].plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")
axes[0].legend(loc="best")

# Plot n_samples vs fit_times
axes[1].grid()
axes[1].plot(train_sizes, fit_times_mean, 'o-')
axes[1].fill_between(train_sizes, fit_times_mean - fit_times_std,
                    fit_times_mean + fit_times_std, alpha=0.1)
axes[1].set_xlabel("Training examples")
axes[1].set_ylabel("fit_times")

```

```

axes[1].set_title("Scalability of the model")

# Plot fit_time vs score
axes[2].grid()
axes[2].plot(fit_times_mean, test_scores_mean, 'o-')
axes[2].fill_between(fit_times_mean, test_scores_mean - test_scores_std,
                    test_scores_mean + test_scores_std, alpha=0.1)
axes[2].set_xlabel("fit_times")
axes[2].set_ylabel("Score")
axes[2].set_title("Performance of the model")

return plt

```

As we observed from the learning curve of Naive Bayes, the training score decreased in the beginning and then flat after 2000th training examples while the validation score increase until 2000th training examples and flat after that. Hence, it is a well fit model.

Besides, as we can see from the second graph (Scalability of the model), Naive Bayes requires more time to train when the training datasets increase.

From the third graph (Performance of the model), Naive Bayes model has a greater score when fit times increased.

```

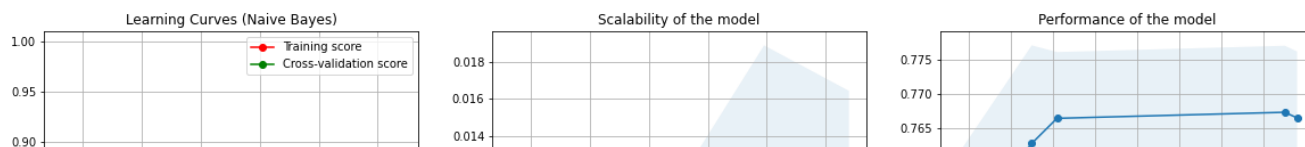
title = "Learning Curves (Naive Bayes)"
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)

estimator = GaussianNB()
plot_learning_curve(naive_model, title, Xtrain, ytrain, ylim=(0.7, 1.01),
                  cv=cv, n_jobs=4)

plt.show()

```





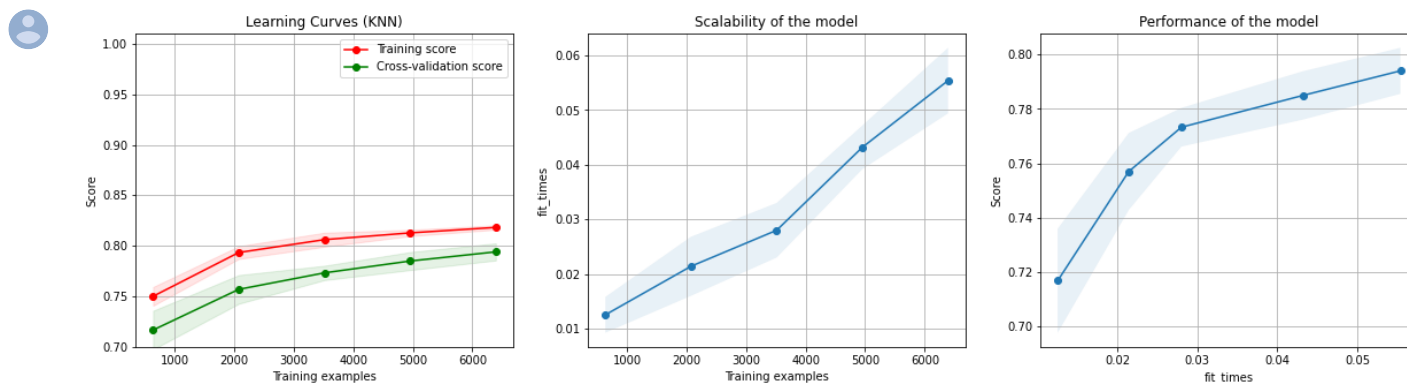
As we observed from the learning curve of KNN, the training score gradually increase to the end while the validation score also the same but it has a score lower than training score. Hence, it is a well fit model.

Besides, as we can see from the second graph (Scalability of the model), KNN requires more time to train when the training datasets increase.

From the third graph (Performance of the model), KNN model has a greater score when fit times increased.

```
title = "Learning Curves (KNN)"
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)

estimator = KNeighborsClassifier()
plot_learning_curve(knn, title, Xtrain, ytrain, ylim=(0.7, 1.01),
                    cv=cv, n_jobs=4)
plt.show()
```



According to the learning curve of Decision Tree, the training score and cross-validation score are not very good at beginning. However, the both scores show a slightly enhancement at the end. When comparing the training score with cross-validation score, the cross-validation score is lower than the training score, which means that there is overfitting in the model.

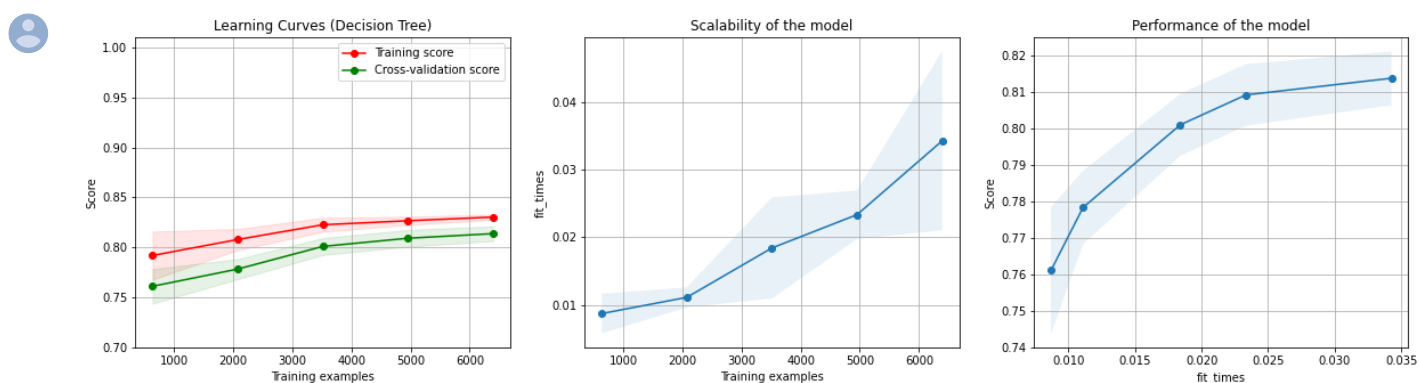
While the scalability graph illustrates the times required by the Decision Tree model to train with various sizes of training dataset.

Based on the third graph, the model achieves a good score (81%) when fit times increase.

```
title = "Learning Curves (Decision Tree)"
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)

plot_learning_curve(decision_tree_model, title, Xtrain, ytrain, ylim=(0.7, 1.01),
                    cv=cv, n_jobs=4)

plt.show()
```



According to the learning curve of Random Forest, the training score and cross-validation score are not very good at beginning. However, the both scores show a gradual enhancement at the end. When comparing the training score with cross-validation score, the cross-validation score is lower than the training score, which means that the model is overfitting.

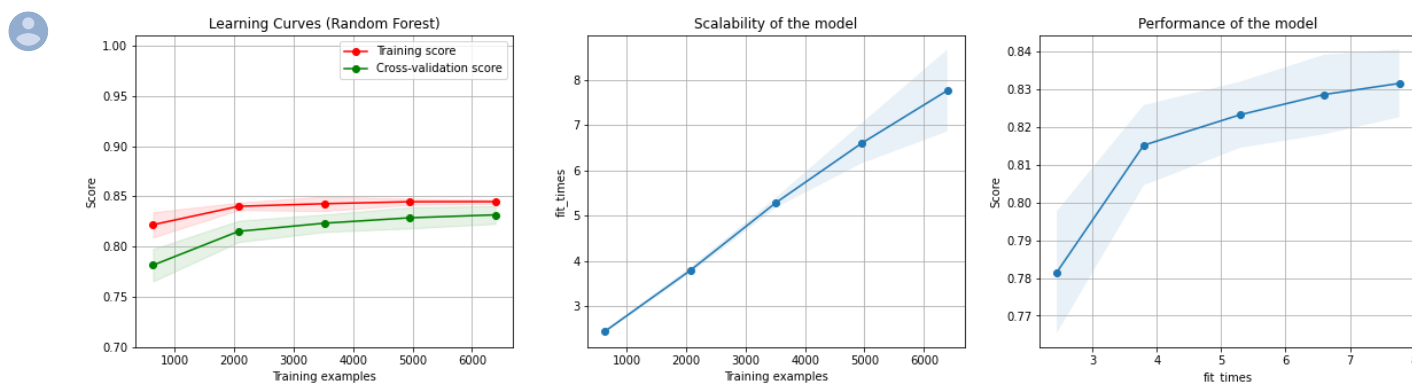
While the scalability graph illustrates the times required by the Random Forest model to train with various sizes of training dataset.

Based on the third graph, the model achieves a good score (83%) when fit times increase.

```
title = "Learning Curves (Random Forest)"
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)
```

```
plot_learning_curve(random_forest_model, title, Xtrain, ytrain, ylim=(0.7, 1.01),
                    cv=cv, n_jobs=4)
```

```
plt.show()
```



According to the learning curve of Logistic Regression, the training score and cross-validation score are slightly different at the beginning. However, the both scores are then come to 0.80 at the end of model training.

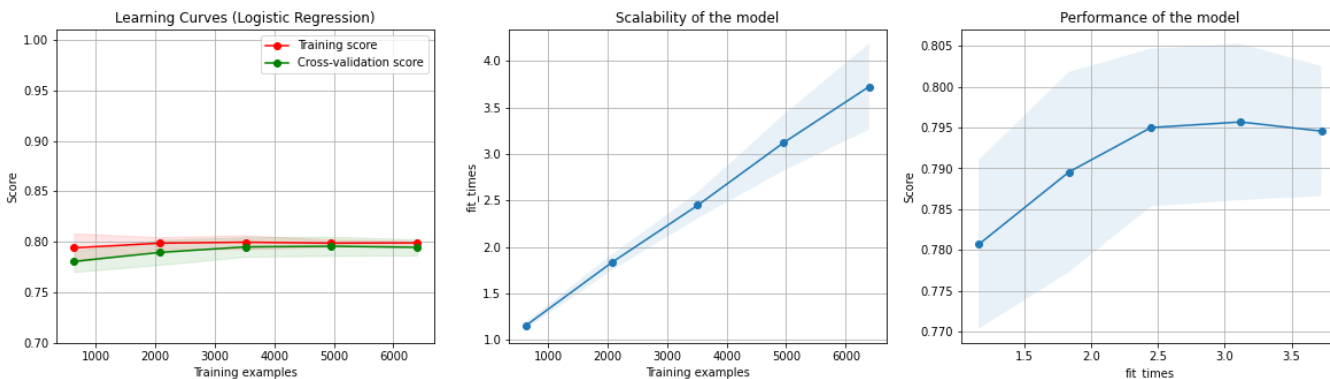
While the next two graphs illustrate the time required for Logistic Regression to train on the different sizes of training dataset and its performance during model training.

```
title = "Learning Curves (Logistic Regression)"
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)

estimator = LogisticRegression(random_state = 30)
plot_learning_curve(logistic_regression_model, title, Xtrain, ytrain, ylim=(0.7, 1.01),
                    cv=cv, n_jobs=4)
```

```
plt.show()
```





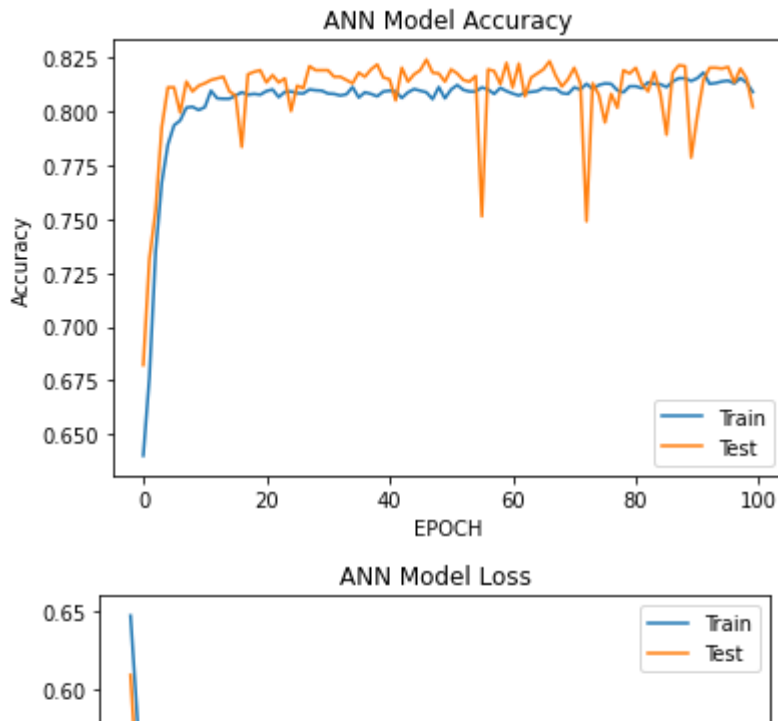
The graphs below are used to visualise the accuracy and loss of ANN model. According to the Model Accuracy graph, the overall accuracy of training and testing datasets are improved at the end.

While for Model Loss graph, the overall loss generated from training and testing datasets are reduced to be less than 0.40 at last.

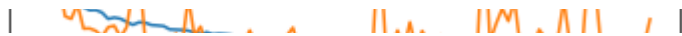
```
#Accuracy and Loss Curves of ANN Model
#Accuracy vs Value Accuracy
ann_model.history.keys()
# summarize history for accuracy
plt.plot(ann_model.history['accuracy'])
plt.plot(ann_model.history['val_accuracy'])
plt.title('ANN Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('EPOCH')
plt.legend(['Train', 'Test'], loc = 'lower right')
plt.show()

#loss vs value loss
plt.plot(ann_model.history['loss'])
plt.plot(ann_model.history['val_loss'])
plt.title('ANN Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()
```





According to the learning curve of Gradient Boosting, the training score and cross-validation score are totally different before 2000 training samples. However, both scores are gradually enhanced at the end. By comparing the training score and cross-validation score, both scores are parallel which means that the model is well fit.



The scalability and performance graphs determine the times required by the Gradient Boosting model to train on the various sizes of training dataset and the model achieves a good score (81.3%) when fit times increase.

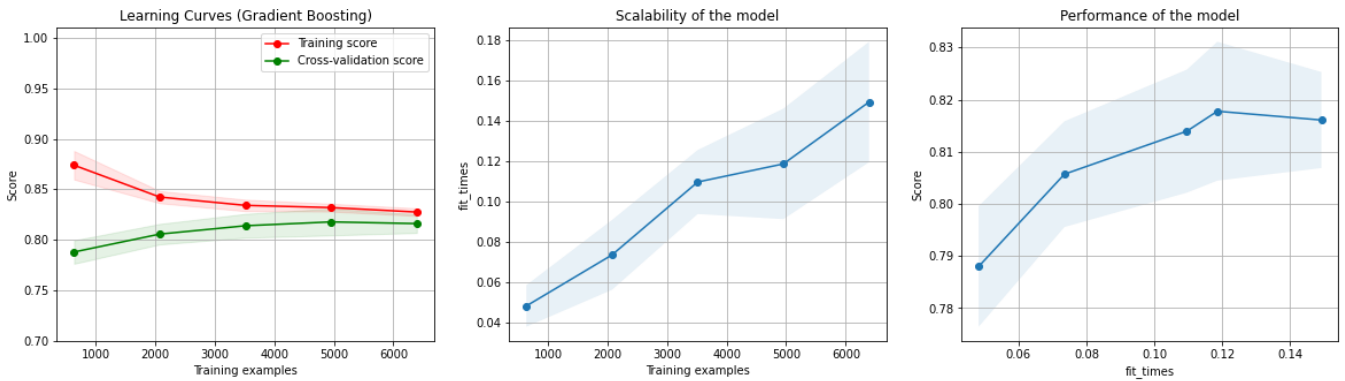
```
title = "Learning Curves (Gradient Boosting)"
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)

estimator = GradientBoostingClassifier()
plot_learning_curve(gradient_boosting_model, title, Xtrain, ytrain, ylim=(0.7, 1.01),
                    cv=cv, n_jobs=4)

plt.show()
```







According to the learning curve of linear SVM, the training score and cross-validation score are almost parallel before 3500 training samples. At the end, both scores are gradually enhanced. By comparing the training score and cross-validation score, both scores are parallel which means that the model is well fit.

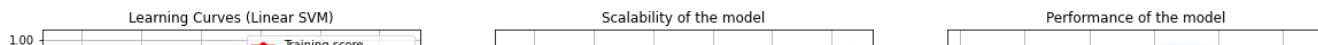
The scalability and performance graphs determine the times required by the linear SVM model to train on the various sizes of training dataset and the model achieves a moderate score (79.6%) when fit times increase.

```
title = "Learning Curves (Linear SVM)"
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)

estimator = SVC()
plot_learning_curve(svm_model, title, Xtrain, ytrain, ylim=(0.7, 1.01),
                    cv=cv, n_jobs=4)

plt.show()
```





## ▼ 5.2 Model Evaluation



```
def model_evaluation(model,name):
```

```
    confusion_matrix = pd.crosstab(ytest, model, rownames=['Actual'], colnames=['Predicted'], m
    sns.heatmap(confusion_matrix, square=True, annot=True, fmt='d', cbar=False)
    plt.xlabel('Prediction label')
    plt.ylabel('True Label');
    plt.title(name)
    plt.yticks([0.5,1.5], [ 'NO', 'YES'],va='center')
    plt.xticks([0.5,1.5], [ 'NO', 'YES'],va='center')
    plt.show()
```

```
    target_names = ['No' , 'Yes']
```

```
    print ('Precision:', precision_score(ytest, model,pos_label=1))
    print ('Accuracy:', accuracy_score(ytest, model))
    print ('F1 score:', f1_score(ytest, model,pos_label=1))
    print ('Recall:', recall_score(ytest, model,pos_label=1))
    print ('\n clasification report:\n', classification_report(ytest,model,target_names=target_
```

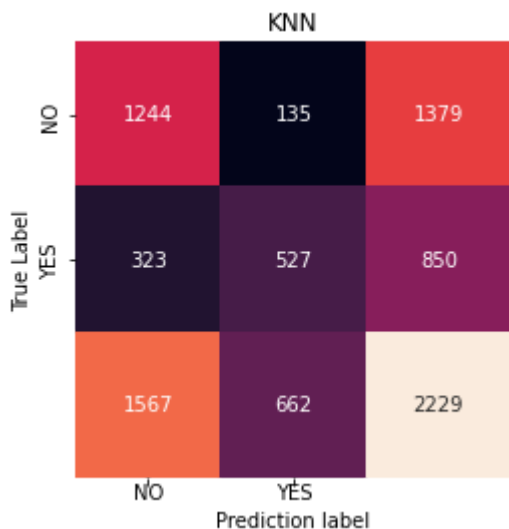
## ▼ KNN

When the KNN predicts a number of 662 (527+135, True Positive + False Positive) subscribed , only 527(True Positive) of which were actual subscribed, while failing to return 323 additional actual subscribed (False Negative), its precision is  $527/662 = 0.7961$  (TP/TP+FP) while its recall is  $527/323+527 = 0.62$  (TP/TP+FN). In other words, when the model predicts the customer term deposit subscription, 135 customers are predicted as they will subscribe the term deposit, but actually they are not subscribing. On the other side, 527 customers are predicted correctly by the model.

While evaluating the model from the accuracy perspective, the result is slightly lower than its precision which is 79.45%. It means that out of 2229 customers, 1244 of customers will not subscribe the term deposit which are predicted correctly by the model. Besides, 527 of customers will subscribe the term deposit which are predicted correctly.

```
model_evaluation(knn_model,"KNN")
```





Precision: 0.7960725075528701

Accuracy: 0.7945266935845671

F1 score: 0.697089947089947

Recall: 0.62

clasification report:

	precision	recall	f1-score	support
No	0.79	0.90	0.84	1379
Yes	0.80	0.62	0.70	850

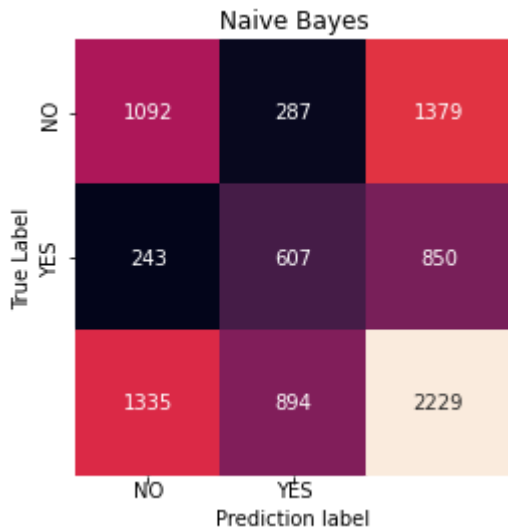
## ▼ Naive Bayes

When the Naive Bayes predicts a number of 894 (607+287, True Positive + False Positive) subscribed, only 607(True Positive) of which were actual subscribed, while failing to return 243 additional actual subscribed (False Negative), its precision is  $607/894 = 0.6789$  (TP/TP+FP) while its recall is  $607/243+607 = 0.7141$  (TP/TP+FN). In other words, when the model predicts the customer term deposit subscription, 287 customers are predicted as they will subscribe the term deposit, but actually they are not subscribing. On the other side, 607 customers are predicted correctly by the model.

While evaluating the model from the accuracy perspective, the result is higher than its precision which is 76.22%. It means that out of 2229 customers, 1092 of customers will not subscribe the term deposit which are predicted correctly by the model. Besides, 607 of customers will subscribe the term deposit which are predicted correctly.

```
model_evaluation(y_naive_model_model,"Naive Bayes")
```





Precision: 0.6789709172259508

Accuracy: 0.7622252131000449

F1 score: 0.6961009174311927

Recall: 0.7141176470588235

clasification report:

	precision	recall	f1-score	support
No	0.82	0.79	0.80	1379
Yes	0.68	0.71	0.70	850

## ▼ Decision Tree

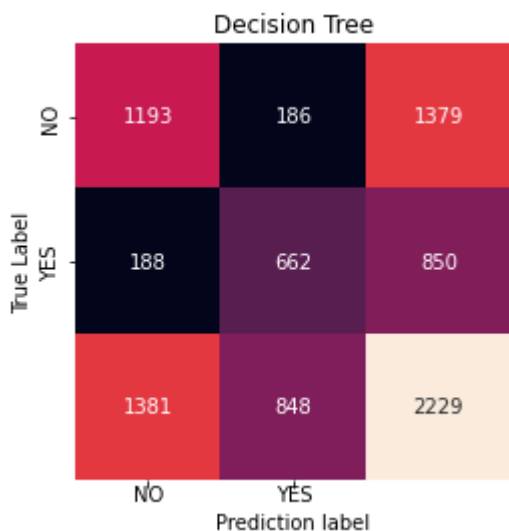
weighted avg	0.76	0.76	0.76	2229
--------------	------	------	------	------

When the Decision tree predicts a number of 848 (662+186, True Positive + False Positive) subscribed , only 662(True Positive) of which were actual subscribed, while failing to return 188 additional actual subscribed (False Negative), its precision is  $662/848 = 0.7807$  (TP/TP+FP) while its recall is  $662/662+188 = 0.7788$  (TP/TP+FN). In other words, when the model predicts the customer term deposit subscription, 186 customers are predicted as they will subscribe the term deposit, but actually they are not subscribing. On the other side, 662 customers are predicted correctly by the model.

While evaluating the model from the accuracy perspective, the result is higher than its precision which is 83.22%. It means that out of 2229 customers, 1193 of customers will not subscribe the term deposit which are predicted correctly by the model. Besides, 662 of customers will subscribe the term deposit which are predicted correctly.

```
model_evaluation(y_pred_decision_tree,"Decision Tree")
```





Precision: 0.7806603773584906

Accuracy: 0.8322117541498429

F1 score: 0.7797408716136631

Recall: 0.7788235294117647

clasification report:

	precision	recall	f1-score	support
No	0.86	0.87	0.86	1379
Yes	0.78	0.78	0.78	850

## ▼ Random Forest

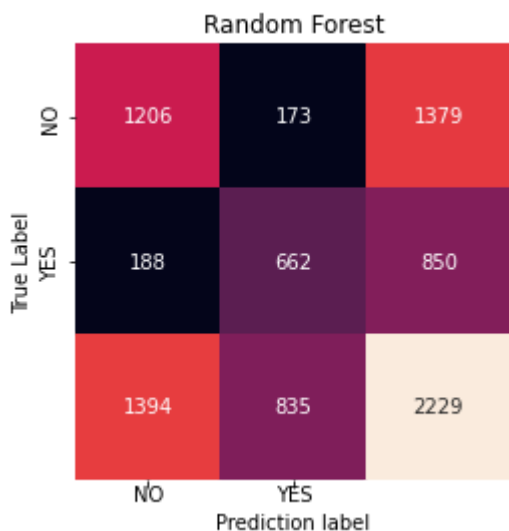
weighted avg	0.83	0.83	0.83	2229
--------------	------	------	------	------

When the Random Forest predicts a number of 835 (662+173, True Positive + False Positive) subscribed, only 662(True Positive) of which were actual subscribed, its precision is  $662/835 = 0.7928$  (TP/TP+FP) while its recall is  $662/662+188 = 0.7788$  (TP/TP+FN). In other words, when the model predicts the customer term deposit subscription, 173 customers are predicted as they will subscribe the term deposit, but actually they are not subscribing. On the other side, 662 customers are predicted correctly by the model.

While evaluating the model from the accuracy perspective, the result is higher than its precision which is 83.80%. It means that out of 2229 customers, 1206 of customers will not subscribe the term deposit which are predicted correctly by the model. Besides, 662 of customers will subscribe the term deposit which are predicted correctly.

```
model_evaluation(y_pred_random_forest,"Random Forest")
```





Precision: 0.792814371257485

Accuracy: 0.8380439659039928

F1 score: 0.7857566765578634

Recall: 0.7788235294117647

clasification report:

	precision	recall	f1-score	support
No	0.87	0.87	0.87	1379
Yes	0.79	0.78	0.79	850

## ▼ Logistic Regression

While Logistic Regression predict a number of 769 subscriber, only 582 which were actual subscribed. Its precision is 0.757 while recall is 0.6847.

While for accuracy perspective, the result is higher than precision result which is 0.7568. It means than out of 2229 customers, 1192 of customers will be predicted correctly in not subscribing the term deposit and 582 of customers will be predicted correctly in subscribing the term deposit.

```
lr_y_pred = logistic_regression_model.predict(Xtest)
model_evaluation(lr_y_pred,"Logistic Regression")
```





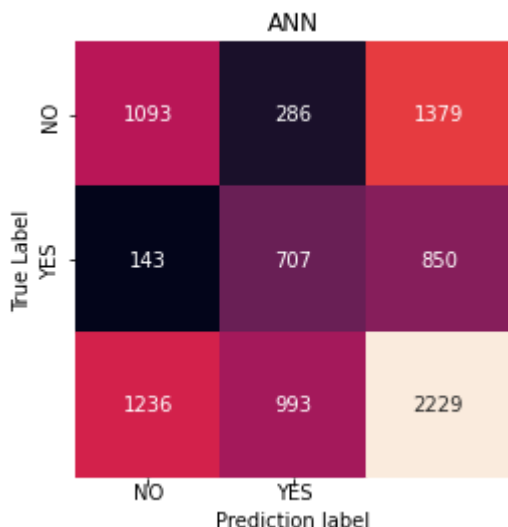
## ▼ ANN

When ANN model predicts a number of 993 (707+286, True Positive + False Positive) subscribed , only 707(True Positive) of which were actual subscribed, while failing to return 143 additional actual subscribed (False Negative), its precision is  $707/993 = 0.7120$  (TP/TP+FP) while its recall is  $707/707+143 = 0.8318$  (TP/TP+FN). In other words, when the model predicts the customer term deposit subscription, 286 customers are predicted as they will subscribe the term deposit, but actually they are not subscribing. On the other side, 707 customers are predicted correctly by the model.

While evaluating the model from the accuracy perspective, the result is higher than its precision which is 80.75%. It means that out of 2229 customers, 1093 of customers will not subscribe the term deposit which are predicted correctly by the model. Besides, 707 of customers will subscribe the term deposit which are predicted correctly.

```
model_evaluation(ann_y_classes,"ANN")
```





## ▼ Gradient Boosting Classification

Recall: 0.831/64/05882353

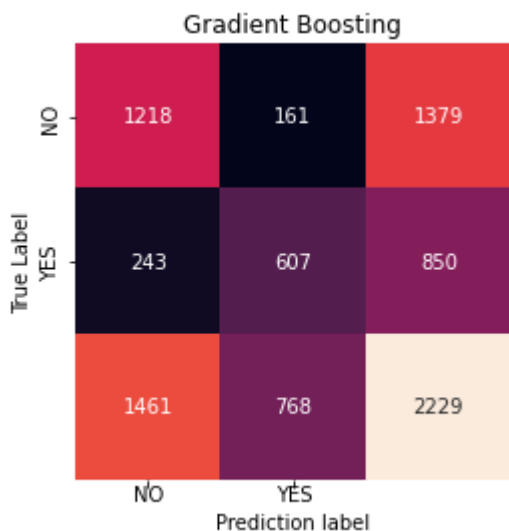
When the Gradient Boosting predicts a number of 768 (607+161, True Positive + False Positive) subscribed, only 607(True Positive) of which were actual subscribed, while failing to return 243 additional actual subscribed (False Negative), its precision is  $607/768 = 0.7904$  (TP/TP+FP) while its recall is  $607/607+243 = 0.7141$  (TP/TP+FN). In other words, when the model predicts the customer term deposit subscription, 161 customers are predicted as they will subscribe the term deposit, but actually they are not subscribing. On the other side, 607 customers are predicted correctly by the model.

While evaluating the model from the accuracy perspective, the result is higher than its precision which is 81.88%. It means that out of 2229 customers, 1218 of customers will not subscribe the term deposit which are predicted correctly by the model. Besides, 607 of customers will subscribe the term deposit which are predicted correctly.

```
model_evaluation(y_pred_gradient_boosting, "Gradient Boosting")
```







Precision: 0.7903645833333334

Accuracy: 0.8187528039479587

F1 score: 0.750309023485785

Recall: 0.7141176470588235

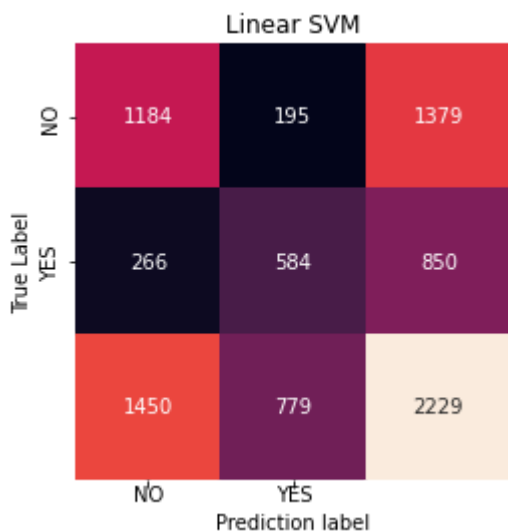
## ▼ Linear SVM

When the linear SVM predicts a number of 779 (584+195, True Positive + False Positive) subscribed, only 584(True Positive) of which were actual subscribed, while failing to return 266 additional actual subscribed (False Negative), its precision is  $584/779 = 0.7497$  (TP/TP+FP) while its recall is  $584/584+266 = 0.6870$  (TP/TP+FN). In other words, when the model predicts the customer term deposit subscription, 195 customers are predicted as they will subscribe the term deposit, but actually they are not subscribing. On the other side, 584 customers are predicted correctly by the model.

While evaluating the model from the accuracy perspective, the result is higher than its precision which is 79.32%. It means that out of 2229 customers, 1184 of customers will not subscribe the term deposit which are predicted correctly by the model. Besides, 584 of customers will subscribe the term deposit which are predicted correctly.

```
model_evaluation(y_pred_svm,"Linear SVM")
```





Precision: 0.7496790757381258

Accuracy: 0.7931807985643786

F1 score: 0.7170042971147943

Recall: 0.6870588235294117

clasification report:

precision recall f1-score support

## ▼ 5.3 Precision-Recall Curve

Precision-Recall Curve (PRC) is a helpful evaluation method of success of prediction when the classes are not balanced. The precision-recall curve demonstrates the compromise between precision and recall for different threshold values. Since the dataset of term deposit prediction is not balanced between subscribers and not subscribers, PRC is chosen to evaluate the models.

A high area under curve (AUC) shows both high recall and precision, where high precision indicates a low false positive rate, and high recall determines a low false negative rate. High scores for both proves that the model has achieved accurate results.

Based on the PR AUC, Random Forest (0.842) outperforms other algorithms which is followed by Decision Tree (0.815) and ANN (0.796). On the other hand, Naive Bayes, Gradient Boosting Classification and Linear SVM are the lowest (0.684). Therefore, it is concluded that Random Forest has the highest precision and recall value overall.

```
from sklearn.metrics import average_precision_score
from sklearn.dummy import DummyClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from matplotlib import pyplot
from sklearn.metrics import precision_recall_curve
# plot no skill and model precision-recall curves
```

```
# no skill model, stratified random class predictions
model = DummyClassifier(strategy='stratified')
model.fit(Xtrain, ytrain)
yhat = model.predict_proba(ytest)
naive_probs = yhat[:, 1]
# calculate the precision-recall auc
precision, recall, _ = precision_recall_curve(ytest, naive_probs)
auc_score = auc(recall, precision)
print('No Skill PR AUC: %.3f' % auc_score)

# decision tree model
dt_yhat = decision_tree_model.predict_proba(Xtest)
dt_model_probs = dt_yhat[:, 1]
# calculate the precision-recall auc
dt_precision, dt_recall, _ = precision_recall_curve(ytest, dt_model_probs)
dt_auc_score = auc(dt_recall, dt_precision)
print('Decision Tree PR AUC: %.3f' % dt_auc_score)

#=====

#random forest model
rf_yhat = random_forest_model.predict_proba(Xtest)
rf_model_probs = rf_yhat[:, 1]
# calculate the precision-recall auc
rf_precision, rf_recall, _ = precision_recall_curve(ytest, rf_model_probs)
rf_auc_score = auc(rf_recall, rf_precision)
print('Random Forest PR AUC: %.3f' % rf_auc_score)

#=====

#knn
knn_yhat = knn.predict_proba(Xtest)
knn_model_probs = knn_yhat[:, 1]
# calculate the precision-recall auc
knn_precision, knn_recall, _ = precision_recall_curve(ytest, knn_model_probs)
knn_auc_score = auc(knn_recall, knn_precision)
print('KNN PR AUC: %.3f' % knn_auc_score)

#=====

#Naive Bayess
n_yhat = naive_model.predict_proba(Xtest)
n_model_probs = n_yhat[:, 1]
# calculate the precision-recall auc
n_precision, n_recall, _ = precision_recall_curve(ytest, n_model_probs)
n_auc_score = auc(n_recall, n_precision)
print('Naive Bayes PR AUC: %.3f' % n_auc_score)

#=====

#Logistic Regression
```

```

lr_yhat = logistic_regression_model.predict_proba(Xtest)
lr_model_probs = lr_yhat[:,1]
# calculate the precision-recall auc
lr_precision, lr_recall, _ = precision_recall_curve(ytest, lr_model_probs)
lr_auc_score = auc(lr_recall, lr_precision)
print('Logistic Regression PR AUC: %.3f' % lr_auc_score)

#=====

#ANN
ann_yhat = ann_classifier.predict_proba(Xtest)
ann_model_probs = ann_yhat[:,0]
# calculate the precision-recall auc
ann_precision, ann_recall, _ = precision_recall_curve(ytest, ann_model_probs)
ann_auc_score = auc(ann_recall, ann_precision)
print('ANN PR AUC: %.3f' % ann_auc_score)

#=====

# Gradient Boosting Classification
gb_yhat = gradient_boosting_model.predict_proba(Xtest)
gb_model_probs = n_yhat[:, 1]
# calculate the precision-recall auc
n_precision, n_recall, _ = precision_recall_curve(ytest, gb_model_probs)
n_auc_score = auc(n_recall, n_precision)
print('Gradient Boosting PR AUC: %.3f' % n_auc_score)

#=====

# Linear SVM
svm_yhat = svm_model.predict_proba(Xtest)
svm_model_probs = n_yhat[:, 1]
# calculate the precision-recall auc
n_precision, n_recall, _ = precision_recall_curve(ytest, svm_model_probs)
n_auc_score = auc(n_recall, n_precision)
print('Linear SVM PR AUC: %.3f' % n_auc_score)

#=====

```



No Skill PR AUC: 0.488  
 Decision Tree PR AUC: 0.815  
 Random Forest PR AUC: 0.842  
 KNN PR AUC: 0.796  
 Naive Bayes PR AUC: 0.684  
 Logistic Regression PR AUC: 0.762  
 ANN PR AUC: 0.796  
 Gradient Boosting PR AUC: 0.684  
 Linear SVM PR AUC: 0.684

```

# calculate the no skill line as the proportion of the positive class
no_skill = len(ytest[ytest==1]) / len(ytest)
# plot the no skill precision-recall curve
pyplot.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')

# plot decision tree model precision-recall curve
dt_precision, dt_recall, _ = precision_recall_curve(ytest, dt_model_probs)
pyplot.plot(dt_recall, dt_precision, marker='x', label='Decision tree')

#plot random forest model precision-recall curve
rf_precision, rf_recall, _ = precision_recall_curve(ytest, rf_model_probs)
pyplot.plot(rf_recall, rf_precision, marker='x', label='Random Forest')

#=====

#plot knn model precision-recall curve
knn_precision, knn_recall, _ = precision_recall_curve(ytest, knn_model_probs)
pyplot.plot(knn_recall, knn_precision, marker='x', label='KNN')

#plot random forest model precision-recall curve
n_precision, n_recall, _ = precision_recall_curve(ytest, n_model_probs)
pyplot.plot(n_recall, n_precision, marker='x', label='Naive Bayes')

#=====

#plot logistic regression model precision-recall curve
lr_precision, lr_recall, _ = precision_recall_curve(ytest, lr_model_probs)
pyplot.plot(lr_recall, lr_precision, marker='x', label='Logistic Regression')

#plot ann model precision-recall curve
ann_precision, ann_recall, _ = precision_recall_curve(ytest, ann_model_probs)
pyplot.plot(ann_recall, ann_precision, marker='x', label='ANN')

#=====

#plot Gradient Boosting model precision-recall curve
gb_precision, gb_recall, _ = precision_recall_curve(ytest, gb_model_probs)
pyplot.plot(gb_recall, gb_precision, marker='x', label='Gradient Boosting')

#plot Linear SVM model precision-recall curve
svm_precision, svm_recall, _ = precision_recall_curve(ytest, svm_model_probs)
pyplot.plot(svm_recall, svm_precision, marker='x', label='Linear SVM')

#=====

# axis labels
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
# show the legend
pyplot.legend()
# show the plot
pyplot.show()

```