Artificial Neural Network (ANN)

```
pip install tensorflow
# pip install tensorflow-gpu
pip install keras
```

import tensorflow

```
import tensorflow as tf
```

```
tf.__version__
```

```
'2.13.0'
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
# from google.colab import files
# uploaded = files.upload()
# # D16data1.csv

import os
os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
os.getcwd()
```

```
'C:\\Users\\surya\\Downloads\\PG-DBDA-Mar23\\Datasets'
```

```
dataset = pd.read_csv('D16data1.csv')
dataset.shape
```

```
(10000, 14)
```

```
dataset.describe()
```

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	Nu
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	

```
dataset.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance
0	1	15634602	Hargrave	619	France	Female	42	2	0.00
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86
2	3	15619304	Onio	502	France	Female	42	8	159660.80
3	4	15701354	Boni	699	France	Female	39	1	0.00
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

null check

```
dataset.isnull().sum()
```

```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

EDA

```
dataset['Gender'].value_counts()
```

```
Gender
Male      5457
Female    4543
Name: count, dtype: int64
```

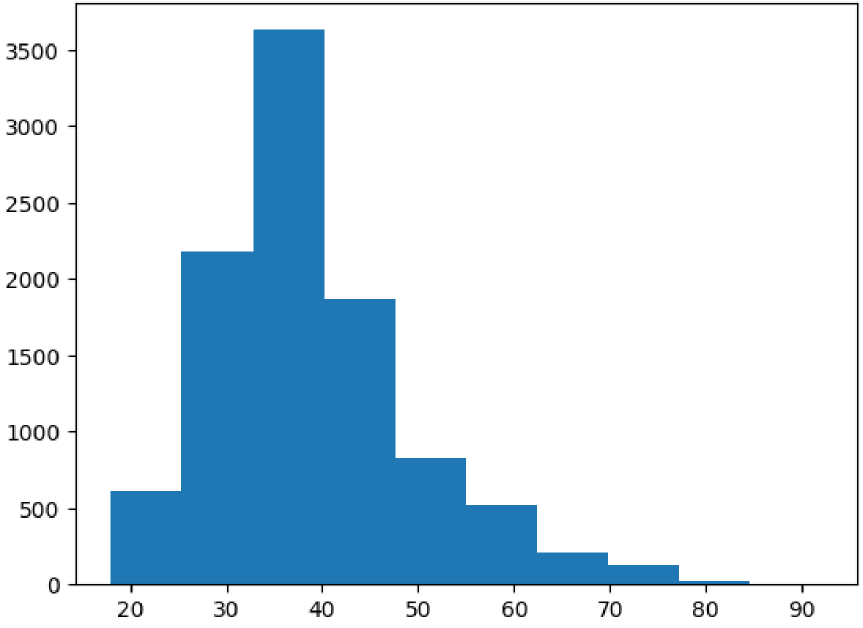
hist: Gender

```
plt.hist(x=dataset['Gender'])
plt.show()
```

5000
4000
3000
2000

hist: Age

```
plt.hist(x=dataset['Age'])
plt.show()
```

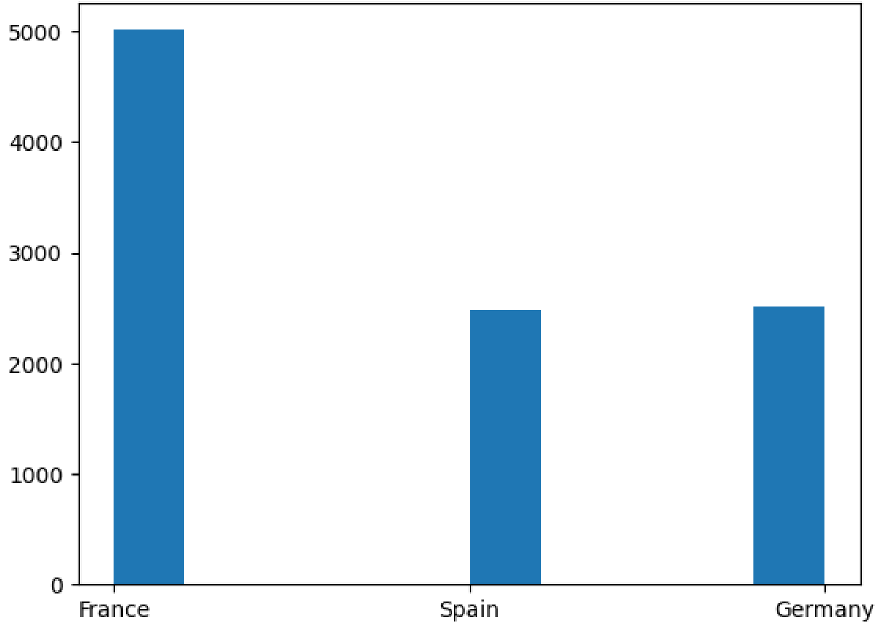


hist: Geography

```
dataset['Geography'].value_counts()
```

```
Geography
France     5014
Germany    2509
Spain      2477
Name: count, dtype: int64
```

```
plt.hist(dataset['Geography'])
plt.show()
```



classification criteria: identify X & Y

```
x = dataset.iloc[ : , 3:-1].values
x[:5]
```

```
array([[619, 'France', 'Female', 42, 2, 0.0, 1, 1, 1, 101348.88],
       [608, 'Spain', 'Female', 41, 1, 83807.86, 1, 0, 1, 112542.58],
       [502, 'France', 'Female', 42, 8, 159660.8, 3, 1, 0, 113931.57],
       [699, 'France', 'Female', 39, 1, 0.0, 2, 0, 0, 93826.63],
       [850, 'Spain', 'Female', 43, 2, 125510.82, 1, 1, 1, 79084.1]],
      dtype=object)
```

```
y = dataset.iloc[ : , -1].values
y[:5]
```

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

Preprocessing

Label Encoding

```
from sklearn.preprocessing import LabelEncoder
```

```
la = LabelEncoder()
```

```
x[ : , 1] = la.fit_transform(x[ : , 1])
x[:5]
```

```
array([[619, 0, 'Female', 42, 2, 0.0, 1, 1, 1, 101348.88],
       [608, 2, 'Female', 41, 1, 83807.86, 1, 0, 1, 112542.58],
       [502, 0, 'Female', 42, 8, 159660.8, 3, 1, 0, 113931.57],
       [699, 0, 'Female', 39, 1, 0.0, 2, 0, 0, 93826.63],
       [850, 2, 'Female', 43, 2, 125510.82, 1, 1, 1, 79084.1]],
      dtype=object)
```

```
x[ : , 2] = la.fit_transform(x[ : , 2])
x[:5]
```

```
array([[619, 0, 0, 42, 2, 0.0, 1, 1, 1, 101348.88],
       [608, 2, 0, 41, 1, 83807.86, 1, 0, 1, 112542.58],
       [502, 0, 0, 42, 8, 159660.8, 3, 1, 0, 113931.57],
       [699, 0, 0, 39, 1, 0.0, 2, 0, 0, 93826.63],
       [850, 2, 0, 43, 2, 125510.82, 1, 1, 1, 79084.1]], dtype=object)
```

Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
x = sc.fit_transform(x)
x[:2]
```

```
array([[-0.32622142, -0.90188624, -1.09598752,  0.29351742, -1.04175968,
        -1.22584767, -0.91158349,  0.64609167,  0.97024255,  0.02188649],
       [-0.44003595,  1.51506738, -1.09598752,  0.19816383, -1.38753759,
         0.11735002, -0.91158349, -1.54776799,  0.97024255,  0.21653375]])
```

splitting

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

Modeling : ANN

```
# initialize ANN
ann = tf.keras.models.Sequential()
```

adding input layer

```
# adding input layer & the first hidden layer
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

adding another input layer

```
# adding the second hidden layer
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

adding output layer

```
# adding output layer
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

compiling the model

```
# compiling the ANN model
ann.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Training the ANN model

```
ann.fit(x_train, y_train, batch_size=32, epochs=100)
```

```
Epoch 1/100
250/250 [==============================] - 1s 2ms/step - loss: 0.5687 - accuracy: 0.7610
Epoch 2/100
250/250 [==============================] - 0s 2ms/step - loss: 0.4768 - accuracy: 0.7960
Epoch 3/100
250/250 [==============================] - 0s 2ms/step - loss: 0.4544 - accuracy: 0.7960
Epoch 4/100
250/250 [==============================] - 0s 2ms/step - loss: 0.4420 - accuracy: 0.7960
Epoch 5/100
250/250 [==============================] - 0s 1ms/step - loss: 0.4341 - accuracy: 0.7983
Epoch 6/100
250/250 [==============================] - 0s 2ms/step - loss: 0.4287 - accuracy: 0.8049
Epoch 7/100
250/250 [==============================] - 0s 1ms/step - loss: 0.4240 - accuracy: 0.8076
Epoch 8/100
250/250 [==============================] - 0s 1ms/step - loss: 0.4191 - accuracy: 0.8110
Epoch 9/100
250/250 [==============================] - 0s 1ms/step - loss: 0.4137 - accuracy: 0.8129
Epoch 10/100
250/250 [==============================] - 0s 2ms/step - loss: 0.4086 - accuracy: 0.8144
Epoch 11/100
250/250 [==============================] - 0s 2ms/step - loss: 0.4037 - accuracy: 0.8150
Epoch 12/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3994 - accuracy: 0.8171
Epoch 13/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3953 - accuracy: 0.8186
Epoch 14/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3922 - accuracy: 0.8201
Epoch 15/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3890 - accuracy: 0.8199
Epoch 16/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3866 - accuracy: 0.8191
Epoch 17/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3843 - accuracy: 0.8300
Epoch 18/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3819 - accuracy: 0.8346
Epoch 19/100
```

```
250/250 [==============================] - 0s 2ms/step - loss: 0.3799 - accuracy: 0.8363
Epoch 20/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3783 - accuracy: 0.8364
Epoch 21/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3766 - accuracy: 0.8391
Epoch 22/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3754 - accuracy: 0.8390
Epoch 23/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3735 - accuracy: 0.8420
Epoch 24/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3722 - accuracy: 0.8419
Epoch 25/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3711 - accuracy: 0.8443
Epoch 26/100
250/250 [==============================] - 1s 2ms/step - loss: 0.3695 - accuracy: 0.8443
Epoch 27/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3687 - accuracy: 0.8465
Epoch 28/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3676 - accuracy: 0.8504
Epoch 29/100
250/250 [==============================] - 0s 2ms/step - loss: 0.3670 - accuracy: 0.8511
```

Prediction

```python
y_pred = ann.predict(x_test)
y_pred[:5]
```

```
63/63 [==============================] - 0s 2ms/step
array([[0.18556872],
       [0.29198548],
       [0.14576535],
       [0.06885878],
       [0.05560081]], dtype=float32)
```

transforming prediction

- based on threshold

```python
y_pred = (y_pred > 0.5)
y_pred[:5]
```

```
array([[False],
       [False],
       [False],
       [False],
       [False]])
```

```
np.concatenate((y_pred.reshape(len(y_pred), 1), y_test.reshape(len(y_test), 1)), 1)
```

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

Evaluation

confusion_matrix

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test, y_pred)
```

accuracy_score

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(y_test, y_pred)
```

Natural Language Processing

```
pip install nltk
```

```
Requirement already satisfied: nltk in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (3.8.1)Note: you may need to restart the kernel to us

Requirement already satisfied: click in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (from nltk) (8.1.5)
Requirement already satisfied: joblib in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (from nltk) (1.3.1)
Requirement already satisfied: regex>=2021.8.3 in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (from nltk) (2023.6.3)
Requirement already satisfied: tqdm in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (from nltk) (4.65.0)
Requirement already satisfied: colorama in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (from click->nltk) (0.4.6)
```

```
import nltk
nltk.download('all')
```

```
[nltk_data] Downloading collection 'all'
[nltk_data]    |
[nltk_data]    | Downloading package abc to
[nltk_data]    |     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]    |   Package abc is already up-to-date!
[nltk_data]    | Downloading package alpino to
[nltk_data]    |     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]    |   Package alpino is already up-to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]    |   Package averaged_perceptron_tagger is already up-
[nltk_data]    |       to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger_ru to
[nltk_data]    |     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]    |   Package averaged_perceptron_tagger_ru is already
[nltk_data]    |       up-to-date!
[nltk_data]    | Downloading package basque_grammars to
[nltk_data]    |     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]    |   Package basque_grammars is already up-to-date!
[nltk_data]    | Downloading package bcp47 to
[nltk_data]    |     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]    |   Package bcp47 is already up-to-date!
[nltk_data]    | Downloading package biocreative_ppi to
[nltk_data]    |     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]    |   Package biocreative_ppi is already up-to-date!
[nltk_data]    | Downloading package bllip_wsj_no_aux to
[nltk_data]    |     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]    |   Package bllip_wsj_no_aux is already up-to-date!
[nltk_data]    | Downloading package book_grammars to
[nltk_data]    |     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]    |   Package book_grammars is already up-to-date!
[nltk_data]    | Downloading package brown to
[nltk_data]    |     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]    |   Package brown is already up-to-date!
[nltk_data]    | Downloading package brown_tei to
[nltk_data]    |     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]    |   Package brown_tei is already up-to-date!
[nltk_data]    | Downloading package cess_cat to
[nltk_data]    |     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]    |   Package cess_cat is already up-to-date!
[nltk_data]    | Downloading package cess_esp to
[nltk_data]    |     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]    |   Package cess_esp is already up-to-date!
[nltk_data]    | Downloading package chat80 to
[nltk_data]    |     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]    |   Package chat80 is already up-to-date!
[nltk_data]    | Downloading package city_database to
[nltk_data]    |     C:\Users\surya\AppData\Roaming\nltk_data...
```

```
[nltk_data]    |   Package city_database is already up-to-date!
[nltk_data]    | Downloading package cmudict to
[nltk_data]    |     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]    |   Package cmudict is already up-to-date!
[nltk_data]    | Downloading package comparative_sentences to
[nltk_data]    |     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]    |   Package comparative_sentences is already up-to-
[nltk_data]    |       date!
[nltk_data]    | Downloading package comtrans to
[nltk_data]    |     C:\Users\surya\AppData\Roaming\nltk_data...
```

1. Tokenization

- Is the process of dividing the whole text into tokens
- It is maily of two types
  - Word Tokenization (separated by words)
  - Sentence Tokenizer (separated by sentence)

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
STOPWORDS = set(stopwords.words('english'))
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
from nltk.tokenize import sent_tokenize, word_tokenize
text = 'Hello there, how are you doing today? The weather is great today. The sky is blue. Python is awesome.'
```

```
print(sent_tokenize(text))
```

```
['Hello there, how are you doing today?', 'The weather is great today.', 'The sky is blue.', 'Python is awesome.']
```

```
print(word_tokenize(text))
```

```
['Hello', 'there', ',', 'how', 'are', 'you', 'doing', 'today', '?', 'The', 'weather', 'is', 'great', 'today', '.', 'The', 'sky', 'is', 'blue', '.', 'Python', 'is',
```

2. Stopword

```
from nltk.corpus import stopwords
```

```
print(stopwords.words('english'))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
```

```
from nltk.corpus import stopwords
```

```
text = 'he is a good boy, he is very good in coding'
text = word_tokenize(text)
text
```

```
['he',
 'is',
 'a',
 'good',
 'boy',
 ',',
 'he',
 'is',
 'very',
 'good',
 'in',
 'coding']
```

```
text_with_no_stopword = [word for word in text if word not in stopwords.words('english')]
text_with_no_stopword
```

```
['good', 'boy', ',', 'good', 'coding']
```

3. Stemming

- convert the derived words into root word

```
from nltk.stem import PorterStemmer
```

```
ps = PorterStemmer()
```

```
example = ['earn', 'earning', 'earned', 'earns']
for w in example:
    print(ps.stem(w))
```

```
earn
earn
earn
earn
```

4. Lemmatizing

```
from nltk.stem import WordNetLemmatizer
```

```
lemmatizer = WordNetLemmatizer()
```

```
example = ['history', 'formality', 'changes', 'histori']
for w in example:
    print(lemmatizer.lemmatize(w))
```

```
history
formality
change
histori
```

5. Wordnet

```
from nltk.corpus import wordnet
```

```
synonyms = []
antonyms = []
for syn in wordnet.synsets('happy'):
    for i in syn.lemmas():
        synonyms.append(i.name())
        if i.antonyms():
            antonyms.append(i.antonyms()[0].name())
```

```
print(set(synonyms))
print(set(antonyms))
```

```
{'happy', 'well-chosen', 'glad', 'felicitous'}
{'unhappy'}
```

6. Part of Speech Tagging

```
from nltk.tokenize import word_tokenize
```

```
text = 'he is a good boy, he is very good in coding'
text = word_tokenize(text)
text
```

```
['he',
 'is',
 'a',
 'good',
 'boy',
 ',',
 'he',
 'is',
 'very',
 'good',
 'in',
 'coding']
```

```
nltk.pos_tag(text)
```

```
[('he', 'PRP'),
 ('is', 'VBZ'),
 ('a', 'DT'),
 ('good', 'JJ'),
 ('boy', 'NN'),
 (',', ','),
 ('he', 'PRP'),
 ('is', 'VBZ'),
 ('very', 'RB'),
 ('good', 'JJ'),
 ('in', 'IN'),
 ('coding', 'VBG')]
```

7. Bag of Words

- what we want to use for our application as input data
- tab separated file

```

```

NLP Application

import libs

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

import dataset

```
# from google.colab import files
# uploaded = files.upload()
# D14data1.csv

import os
os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
os.getcwd()
```

```
'C:\\Users\\surya\\Downloads\\PG-DBDA-Mar23\\Datasets'
```

```
dataset = pd.read_csv('D16data2.tsv', sep='\t')
dataset.shape
```

```
(1000, 2)
```

```
dataset.head()
```

	Review	Liked
0	Wow... Loved this place.	1
1	Crust is not good.	0

```
dataset.describe()
```

	Liked
count	1000.00000
mean	0.50000
std	0.50025
min	0.00000
25%	0.00000
50%	0.50000
75%	1.00000
max	1.00000

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Review  1000 non-null   object
 1   Liked   1000 non-null   int64 
dtypes: int64(1), object(1)
memory usage: 15.8+ KB
```

download stopwords

```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\surya\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
```

```
import re
corpus = []
ps = PorterStemmer()
for i in range(0, 1000):
    review = re.sub('[^a-zA-Z]', ' ',dataset['Review'][i])
    review = review.lower()
    review = review.split()
    review = [ps.stem(word) for word in review
              if not word in set(stopwords.words('english'))]
    review = ' '.join(review)
    corpus.append(review)
```

identify X & Y

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
cv = CountVectorizer(max_features=1500)
```

```
x = cv.fit_transform(corpus).toarray()
x[:2]
```

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

```
y = dataset.iloc[ : , 1].values
y[:2]
```

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

splitting

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.20, random_state=0)
```

Naive-Bayes Model

Modeling

```
from sklearn.naive_bayes import GaussianNB
```

```
classifier = GaussianNB()
```

Training

```
classifier.fit(x_train, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

predict

```
y_pred = classifier.predict(x_test)
y_pred[:5]
```

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

Evaluation

```
from sklearn.metrics import accuracy_score, confusion_matrix
```

```
accuracy_score(y_test, y_pred)
```

```
0.73
```

```
confusion_matrix(y_test, y_pred)
```

```
array([[55, 42],
       [12, 91]], dtype=int64)
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
```

Modeling

```
dt_classifier = DecisionTreeClassifier(random_state=0, criterion='entropy')
```

Training

```
dt_classifier.fit(x_train, y_train)
```

```
▾                  DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

Prediction

```
y_pred = dt_classifier.predict(x_test)
y_pred[:5]
```

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

Evaluation

```
from sklearn.metrics import accuracy_score, confusion_matrix
```

```
accuracy_score(y_test, y_pred)
```

```
0.71
```

```
confusion_matrix(y_test, y_pred)
```

```
array([[74, 23],
       [35, 68]], dtype=int64)
```

Random Forest

Modeling

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf_classifier = RandomForestClassifier(n_estimators=500, criterion='gini', random_state=0)
```

Training

```
rf_classifier.fit(x_train, y_train)
```

```
▾                  RandomForestClassifier
RandomForestClassifier(n_estimators=500, random_state=0)
```

Prediction

```
rf_classifier.predict(x_test)
rf_classifier[:5]
```

```
[DecisionTreeClassifier(max_features='sqrt', random_state=209652396),
 DecisionTreeClassifier(max_features='sqrt', random_state=398764591),
 DecisionTreeClassifier(max_features='sqrt', random_state=924231285),
 DecisionTreeClassifier(max_features='sqrt', random_state=1478610112),
 DecisionTreeClassifier(max_features='sqrt', random_state=441365315)]
```

Evaluation

```
from sklearn.metrics import accuracy_score, confusion_matrix
```

```
accuracy_score(y_test, y_pred)
```

```
0.71
```

```
confusion_matrix(y_test, y_pred)
```

```
array([[74, 23],
       [35, 68]], dtype=int64)
```

```
Soch ko badloge toh sitare badal jayenge
Disha ko badloge toh manzil badal jayegi
```

- Note: do not try to brain wash disha and manzil