FUTURE OF UNIVERSITY

Introduction

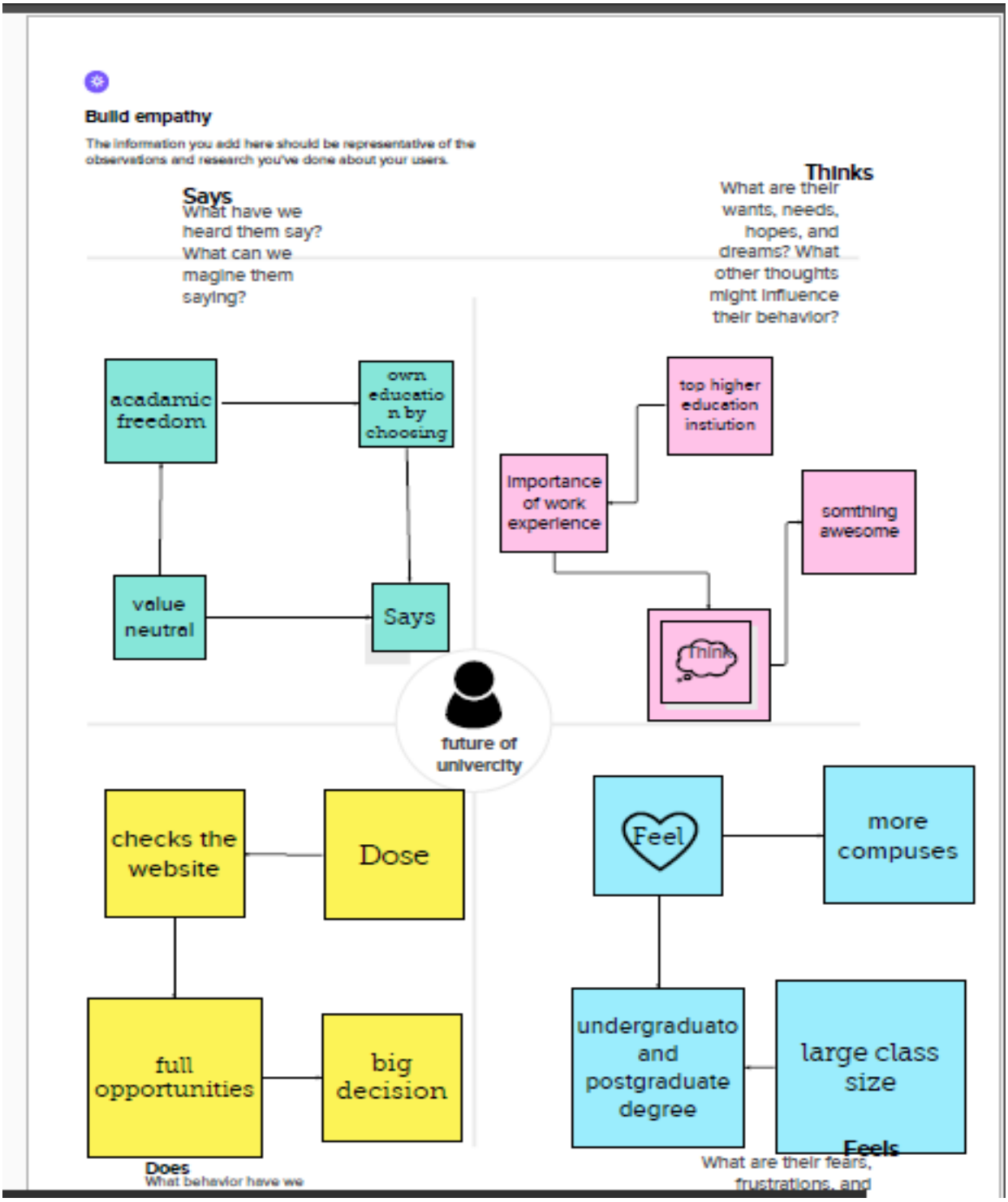
- I am not prophet,not can I look into the future-not even at the end of this productive conference on essential changes in the higher education system.When the work situation of the academic profession ,its diversification and academic freedom are at issue, the university as a whole is called into question,at least the university as we have known and appreciated it for a long time.Will that university have a future?This is not clear at all, especially when we consider the managerial university and the ever increasing marketization of all aspects if university life. In the following, I present a few remarks about the continuously fading theory of the university, centred on the keywords education, university, universality, and quality.

PURPOSES

- Universities, in fact, are institutions that institutionalise ideas, which in turn embody in them universal human values. Universities, thus, perform two normative functions. First, of institutionalising such value-laden ideas, and second, of housing and hosting these ideas.The state should therefore support universities, and not see students as “enemies” of the nations Swatahsiddha Sarkar writes that in colonial India, universities were established to serve the interests of the ruling elite. Today, Sarkar believes, universities are directed to “fine-tune” their academic coursework and research in “consonance with national priorities.” He uses Aristotle’s theory of episteme, techne and phronesis to understand the university and contends that the idea of a university was “premised in the search for episteme.”

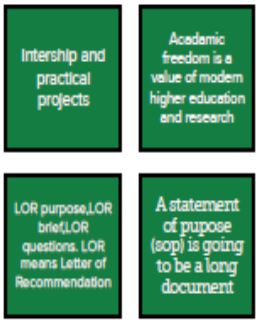
PROBLEM DEFINITION&DESIGN THINKING

EMPATHY MAP:

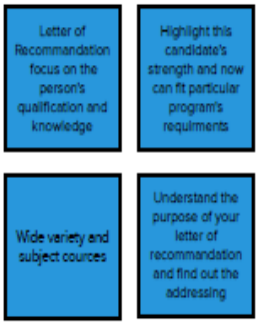


BRAINSTROM:

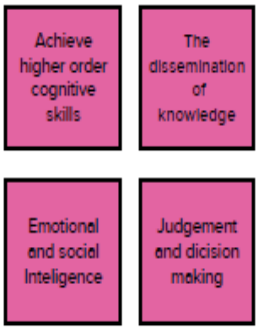
P.Manju



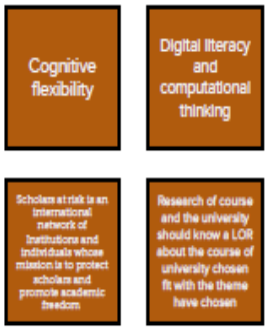
N.Thanam



K.Kiruthika



C.Kumutha



ADVANTAGES

- IT OPENS New avenue of employment,most jobs are that require one to be knowledgeable regarding certain things like medicine, to become a doctor you need to study more than just the basics
- New life experiences , university is a place of self-reflection , it's where you get tested at how you handle situations , the new experiences are exciting and meeting new people is always amazing , you get to blend in with different personalities. Student life is like a booze-filled nirvana..

DISADVANTAGES

- Universities are costly , you spend thousands studying but end up getting no job. Not all of us were born with silver spoons in our mouths so we must think of the money before we decide if whether or not we are going to pursue our studies. The financial aid is not a guarantee most drop out because the financial aid can no longer pay for them
- Lack of practical experience. in University you spend 70% of your time studying big costly books without getting practical experience , if you get that experience it is of limited time.

CONCLUSION

- A strong university system is essential to a country's economic success and the vibrancy and depth of its intellectual and cultural life.The aim of these proposals is to bulid a new national consensus between individuals, government,and employers as to how our higher education system should be supported,adapted and expanded.

APPENDIX

SOURCE CODE

IMPORTING PACKAGES

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

IMPORTING DATASET

```
%matplotlib inline
df=pd.read_csv('/content/Admission_Predict.csv')
df.head()
```

HEAD

```
df.head()
```

TAIL

```
df.head()
```

DATA PREPROCESSING

```
df.info()
df.isnull().any()
```

ANALYSING THE CORRELTION UNIVERSITY ATTRIBUTES WITH OTHER ATTRIBUTES

```
sns.distplot(df['GRE Score'])
sns.pairplot(data=df,hue='Research',markers=["^","v"],palette='inferno')
sns.scatterplot(x='University Rating',y='CGPA',data=df,color='Red',s=100)
```

ERROR OF PREDICTIOIN

```
print(train_predictions)
print(train_acc)
print(test_acc)
print("\nAccuracy score: %f" %(accuracy_score(y_test,y_pred)*100))
print("Recall score : %f n" %(recall_score(y_test,y_pred)*100))
print("ROC score : %f\n" %(roc_auc_score(y_test,y_pred)*100))
print(confusion_matrix(y_test,y_pred))
print(y_train.shape)
print(y_pred.shape)
print(classification_report(y_test,y_pred))
```

CLASSIFICATION REPORT

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

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred=lr.predict(x_test)
y_pred
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
data=pd.read_csv('/content/Admission_Predict.csv')
data.head()
category=['GRE Score','TOEFL Score','University Rating','SOP','LOR','CGPA'
,'Research','Chance of Admit']
color=['yellowgreen','gold','lightskyblue','pink','red','purple','orange',
'gray']
start=True
for i in np.arange(4):
    fig=plt.figure(figsize=(14,8))
    plt.subplot2grid((4,2),(i,0))
    data[category[i]].hist(color=color[i],bins=10)
    plt.title(category[2*i])
    plt.subplot2grid((4,2),(i,1))
    data[category[i]].hist(color=color[i],bins=10)
    plt.title(category[2*i+1])
plt.subplots_adjust(hspace=0.7,wspace=0.2)
plt.show()
```

ERROR OF PREDICTIOIN

```
print(train_predictions)
print(train_acc)
print(test_acc)
print("\nAccuracy score: %f" %(accuracy_score(y_test,y_pred)*100))
print("Recall score : %f n" %(recall_score(y_test,y_pred)*100))
print("ROC score : %f\n" %(roc_auc_score(y_test,y_pred)*100))
print(confusion_matrix(y_test,y_pred))
print(y_train.shape)
print(y_pred.shape)
print(classification_report(y_test,y_pred))
```

CLASSIFICATION REPORT

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

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred=lr.predict(x_test)
y_pred
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
data=pd.read_csv('/content/Admission_Predict.csv')
data.head()
category=['GRE Score','TOEFL Score','University Rating','SOP','LOR','CGPA'
,'Research','Chance of Admit']
color=['yellowgreen','gold','lightskyblue','pink','red','purple','orange',
'gray']
start=True
for i in np.arange(4):
    fig=plt.figure(figsize=(14,8))
    plt.subplot2grid((4,2),(i,0))
    data[category[i]].hist(color=color[i],bins=10)
    plt.title(category[2*i])
    plt.subplot2grid((4,2),(i,1))
    data[category[i]].hist(color=color[i],bins=10)
    plt.title(category[2*i+1])
plt.subplots_adjust(hspace=0.7,wspace=0.2)
```

```
plt.show()
```

SPLITTING RECORDS FOR TRAINING AND TESTING

ERROR OF PREDICTIOIN

```
print(train_predictions)
print(train_acc)
print(test_acc)
print("\nAccuracy score: %f" %(accuracy_score(y_test,y_pred)*100))
print("Recall score : %f n" %(recall_score(y_test,y_pred)*100))
print("ROC score : %f\n" %(roc_auc_score(y_test,y_pred)*100))
print(confusion_matrix(y_test,y_pred))
print(y_train.shape)
print(y_pred.shape)
print(classification_report(y_test,y_pred))
```

CLASSIFICATION REPORT

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

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred=lr.predict(x_test)
y_pred
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
data=pd.read_csv('/content/Admission_Predict.csv')
data.head()
category=['GRE Score','TOEFL Score','University Rating','SOP','LOR','CGPA'
,'Research','Chance of Admit']
color=['yellowgreen','gold','lightskyblue','pink','red','purple','orange',
'gray']
start=True
for i in np.arange(4):
    fig=plt.figure(figsize=(14,8))
    plt.subplot2grid((4,2),(i,0))
    data[category[i]].hist(color=color[i],bins=10)
    plt.title(category[2*i])
```

IMPORTING LOGISTICREGRESSION

```
from sklearn.linear_model._logistic import LogisticRegression
cls=LogisticRegression(random_state=0)

lr=cls.fit(x_train,y_train)
```

TRAINING AND TESTING THE RECORDS OF DATASET FOR PREDICTION

```
model.fit(x_train,y_train,batch_size=20,epochs=100)
```

ERROR OF PREDICTIOIN

```
print(train_predictions)
print(train_acc)
print(test_acc)
print("\nAccuracy score: %f" %(accuracy_score(y_test,y_pred)*100))
print("Recall score : %f n" %(recall_score(y_test,y_pred)*100))
print("ROC score : %f\n" %(roc_auc_score(y_test,y_pred)*100))
print(confusion_matrix(y_test,y_pred))
```

```
print(y_train.shape)
print(y_pred.shape)
print(classification_report(y_test,y_pred))
```

CLASSIFICATION REPORT

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

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred=lr.predict(x_test)
y_pred
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
data=pd.read_csv('/content/Admission_Predict.csv')
data.head()
category=['GRE Score','TOEFL Score','University Rating','SOP','LOR','CGPA'
,'Research','Chance of Admit']
color=['yellowgreen','gold','lightskyblue','pink','red','purple','orange',
'gray']
start=True
for i in np.arange(4):
    fig=plt.figure(figsize=(14,8))
    plt.subplot2grid((4,2),(i,0))
    data[category[i]].hist(color=color[i],bins=10)
    plt.title(category[2*i])
```

ERROR OF PREDICTIOIN

```
print(train_predictions)
print(train_acc)
print(test_acc)
print("\nAccuracy score: %f" %(accuracy_score(y_test,y_pred)*100))
print("Recall score : %f n" %(recall_score(y_test,y_pred)*100))
print("ROC score : %f\n" %(roc_auc_score(y_test,y_pred)*100))
print(confusion_matrix(y_test,y_pred))
print(y_train.shape)
print(y_pred.shape)
print(classification_report(y_test,y_pred))
```

CLASSIFICATION REPORT

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

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred=lr.predict(x_test)
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
data=pd.read_csv('/content/Admission_Predict.csv')
data.head()
category=['GRE Score','TOEFL Score','University Rating','SOP','LOR','CGPA'
,'Research','Chance of Admit']
color=['yellowgreen','gold','lightskyblue','pink','red','purple','orange',
'gray']
start=True
for i in np.arange(4):
    fig=plt.figure(figsize=(14,8))
```

```
    plt.subplot2grid((4,2),(i,0))
    data[category[i]].hist(color=color[i],bins=10)
    plt.title(category[2*i])
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
data=pd.read_csv('/content/Admission_Predict.csv')
data.head()
category=['GRE Score','TOEFL Score','University Rating','SOP','LOR','CGPA'
,'Research','Chance of Admit']
color=['yellowgreen','gold','lightskyblue','pink','red','purple','orange',
'gray']
start=True
for i in np.arange(4):
    fig=plt.figure(figsize=(14,8))
    plt.subplot2grid((4,2),(i,0))
    data[category[i]].hist(color=color[i],bins=10)
    plt.title(category[2*i])
```

IMPORTING DATASET

```
%matplotlib inline
df=pd.read_csv('/content/Admission_Predict.csv')
df.head()
```

HEAD

```
df.head()
```

CLASSIFICATION REPORT

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

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred=lr.predict(x_test)
y_pred
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
data=pd.read_csv('/content/Admission_Predict.csv')
data.head()
category=['GRE Score','TOEFL Score','University Rating','SOP','LOR','CGPA'
,'Research','Chance of Admit']
color=['yellowgreen','gold','lightskyblue','pink','red','purple','orange',
'gray']
start=True
for i in np.arange(4):
    fig=plt.figure(figsize=(14,8))
    plt.subplot2grid((4,2),(i,0))
    data[category[i]].hist(color=color[i],bins=10)
    plt.title(category[2*i]
y
```

TAIL

```
df.head()
```

DATA PREPROCESSING

```
df.info()
df.isnull().any()
```

ANALYSING THE CORRELTION UNIVERSITY ATTRIBUTES WITH OTHER ATTRIBUTES

```
sns.distplot(df['GRE Score'])
sns.pairplot(data=df,hue='Research',markers=["^","v"],palette='inferno')
sns.scatterplot(x='University Rating',y='CGPA',data=df,color='Red',s=100)
```

CLASSIFICATION REPORT

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

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred=lr.predict(x_test)
y_pred
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
data=pd.read_csv('/content/Admission_Predict.csv')
data.head()
category=['GRE Score','TOEFL Score','University Rating','SOP','LOR','CGPA'
,'Research','Chance of Admit']
color=['yellowgreen','gold','lightskyblue','pink','red','purple','orange',
'gray']
start=True
for i in np.arange(4):
    fig=plt.figure(figsize=(14,8))
    plt.subplot2grid((4,2),(i,0))
    data[category[i]].hist(color=color[i],bins=10)
    plt.title(category[2*i])
    plt.subplot2grid((4,2),(i,1))
```

```
    data[category[i]].hist(color=color[i],bins=10)
    plt.title(category[2*i+1])
plt.subplots_adjust(hspace=0.7,wspace=0.2)
plt.show()
```

SPLITTING RECORDS FOR TRAINING AND TESTING

from sklearn.model_selection import train_test_split

TRAINING AND TESTING THE RECORDS OF DATASET FOR PREDICTION

```
model.fit(x_train,y_train,batch_size=20,epochs=100)
```

TRAINING AND TESTING THE RECORDS OF DATASET FOR PREDICTION

```
model.fit(x_train,y_train,batch_size=20,epochs=100)
```

ERROR OF PREDICTIOIN

```
print(train_predictions)
print(train_acc)
print(test_acc)
print("\nAccuracy score: %f" %(accuracy_score(y_test,y_pred)*100))
print("Recall score : %f n" %(recall_score(y_test,y_pred)*100))
print("ROC score : %f\n" %(roc_auc_score(y_test,y_pred)*100))
print(confusion_matrix(y_test,y_pred))
print(y_train.shape)
print(y_pred.shape)
print(classification_report(y_test,y_pred))
```

CLASSIFICATION REPORT

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

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred=lr.predict(x_test)
y_pred
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
data=pd.read_csv('/content/Admission_Predict.csv')
data.head()
category=['GRE Score','TOEFL Score','University Rating','SOP','LOR','CGPA'
,'Research','Chance of Admit']
color=['yellowgreen','gold','lightskyblue','pink','red','purple','orange',
'gray']
start=True
for i in np.arange(4):
    fig=plt.figure(figsize=(14,8))
    plt.subplot2grid((4,2),(i,0))
    data[category[i]].hist(color=color[i],bins=10)
    plt.title(category[2*i])
```

ERROR OF PREDICTIOIN

```
print(train_predictions)
print(train_acc)
```

```
print(test_acc)
print("\nAccuracy score: %f" %(accuracy_score(y_test,y_pred)*100))
print("Recall score : %f n" %(recall_score(y_test,y_pred)*100))
print("ROC score : %f\n" %(roc_auc_score(y_test,y_pred)*100))
print(confusion_matrix(y_test,y_pred))
print(y_train.shape)
print(y_pred.shape)
print(classification_report(y_test,y_pred))
```

RESULT

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
df=pd.read_csv('/content/Admission_Predict.csv')
df.head()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

✓ 1h 35m 31s completed at 10:32 PM

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Serial No.         400 non-null    int64
 1   GRE Score          400 non-null    int64
 2   TOEFL Score        400 non-null    int64
 3   University Rating  400 non-null    int64
 4   SOP                400 non-null    float64
 5   LOR                400 non-null    float64
 6   CGPA               400 non-null    float64
 7   Research           400 non-null    int64
 8   Chance of Admit    400 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 28.2 KB
```

```
df.isnull().any()
```

```
Serial No.           False
GRE Score            False
TOEFL Score          False
University Rating    False
SOP                  False
LOR                  False
CGPA                 False
Research             False
Chance of Admit      False
dtype: bool
```

```
] data=df.rename(columns={'change of Admit':'change of Admit'})
```

```
df.describe()
```

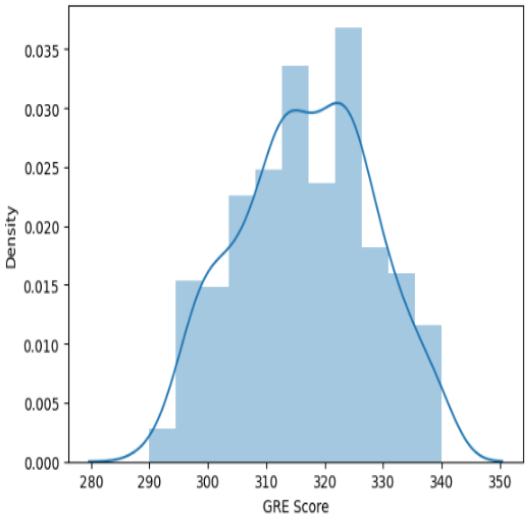
	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000
mean	200.500000	316.807500	107.410000	3.087500	3.400000	3.452500	8.598925	0.547500	0.724350
std	115.614301	11.473646	6.069514	1.143728	1.006869	0.898478	0.596317	0.498362	0.142609
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.000000	6.800000	0.000000	0.340000
25%	100.750000	308.000000	103.000000	2.000000	2.500000	3.000000	8.170000	0.000000	0.640000
50%	200.500000	317.000000	107.000000	3.000000	3.500000	3.500000	8.610000	1.000000	0.730000
75%	300.250000	325.000000	112.000000	4.000000	4.000000	4.000000	9.062500	1.000000	0.830000
max	400.000000	340.000000	120.000000	5.000000	5.000000	5.000000	9.920000	1.000000	0.970000

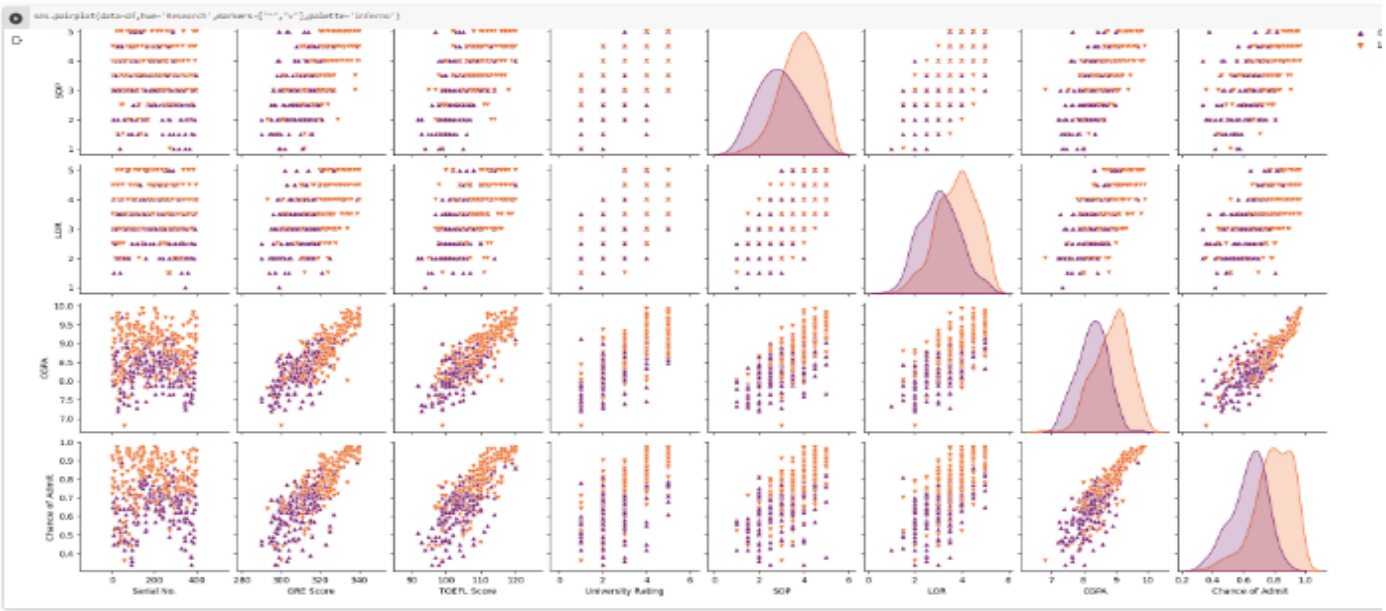
```
sns.distplot(df['GRE Score'])
```

```
<ipython-input-18-3e4f1ef1d79e>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['GRE Score'])
<Axes: xlabel='GRE Score', ylabel='Density'>
```

```
sns.distplot(df['GRE Score'])
```

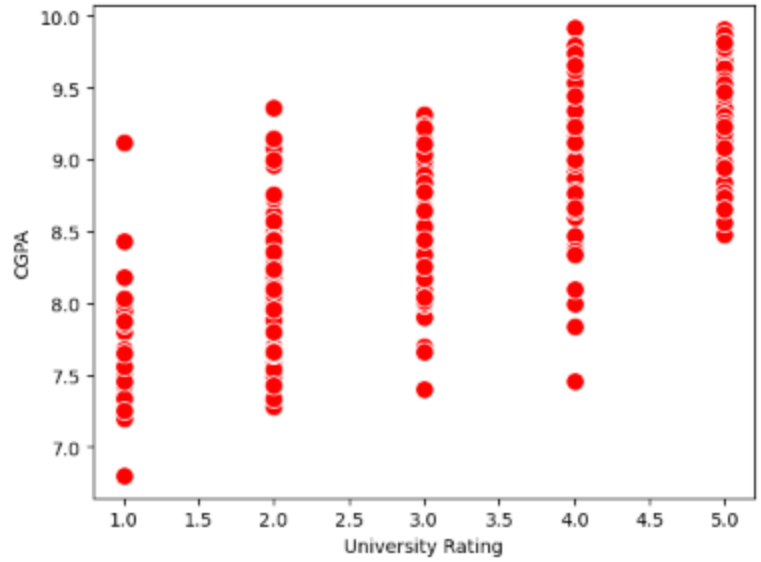
```
<Axes: xlabel='GRE Score', ylabel='Density'>
```



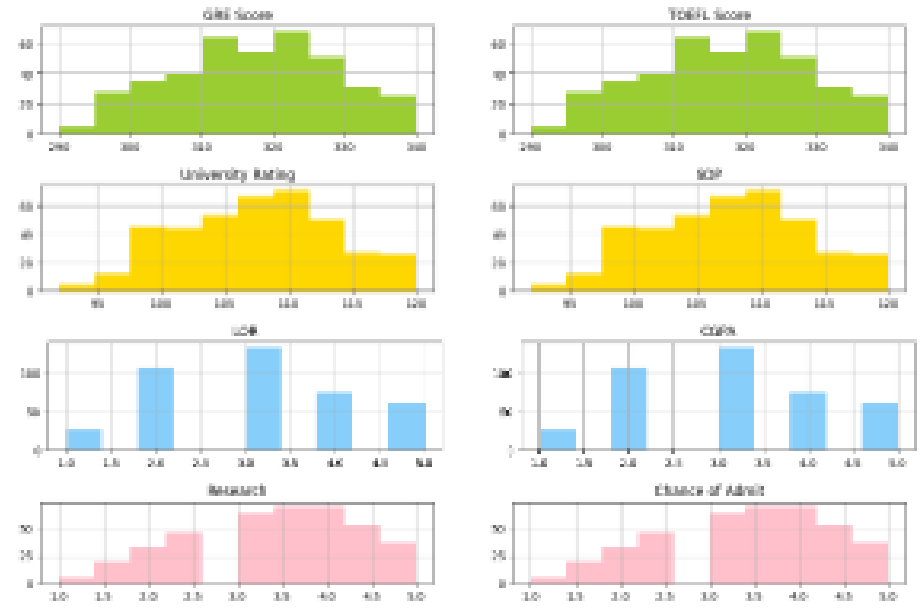


```
sns.scatterplot(x='University Rating',y='CGPA',data=df,color='Red',s=100)
```

```
<Axes: xlabel='University Rating', ylabel='CGPA'>
```




```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
dataset=pd.read_csv('/content/Admission_Predict.csv')
data.head()
categorys=['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR', 'CGPA', 'Research', 'Chance of Admit']
colors=['yellowgreen', 'gold', 'lightskyblue', 'pink', 'red', 'purple', 'orange', 'gray']
startflag=
for i in np.arange(8):
    fig=plt.figure(figsize=(14,8))
    plt.subplot2grid((2,2),(0,0))
    data[category[i]].hist(color=colors[i],bins=10)
    plt.title(category[2*i])
    plt.subplot2grid((2,2),(0,1))
    data[category[i]].hist(color=colors[i],bins=10)
    plt.title(category[2*i+1])
plt.subplots_adjust(hspace=0.7,wspace=0.3)
plt.show()
```



```
from sklearn.preprocessing import MinMaxScaler
sc=MinMaxScaler()
x=sc.fit_transform(x)
x
```

```
array([[0.        , 0.94      , 0.92857143, ..., 0.875     , 0.875     ,
        0.91346154],
       [0.00250627, 0.68      , 0.53571429, ..., 0.75      , 0.875     ,
        0.66346154],
       [0.00501253, 0.52      , 0.42857143, ..., 0.5       , 0.625     ,
        0.38461538],
       ...,
       [0.99498747, 0.8       , 0.85714286, ..., 1.        , 0.875     ,
        0.84935897],
       [0.99749373, 0.44      , 0.39285714, ..., 0.625     , 0.75      ,
        0.63461538],
       [1.        , 0.86      , 0.89285714, ..., 1.        , 0.75      ,
        0.91666667]])
```

```
x=data.iloc[:,0:7].values
x
```

```
array([[  1.  , 337.  , 118.  , ...,   4.5 ,   4.5 ,   9.65],
       [  2.  , 324.  , 107.  , ...,   4.  ,   4.5 ,   8.87],
       [  3.  , 316.  , 104.  , ...,   3.  ,   3.5 ,   8.  ],
       ...,
       [398.  , 330.  , 116.  , ...,   5.  ,   4.5 ,   9.45],
       [399.  , 312.  , 103.  , ...,   3.5 ,   4.  ,   8.78],
       [400.  , 333.  , 117.  , ...,   5.  ,   4.  ,   9.66]])
```

```
y=data.iloc[:,8].values
y
```

```
array([0.92, 0.76, 0.72, 0.8 , 0.65, 0.9 , 0.75, 0.68, 0.5 , 0.45, 0.52,
       0.84, 0.78, 0.62, 0.61, 0.54, 0.66, 0.65, 0.63, 0.62, 0.64, 0.7 ,
       0.94, 0.95, 0.97, 0.94, 0.76, 0.44, 0.46, 0.54, 0.65, 0.74, 0.91,
       0.9 , 0.94, 0.88, 0.64, 0.58, 0.52, 0.48, 0.46, 0.49, 0.53, 0.87,
       0.91, 0.88, 0.86, 0.89, 0.82, 0.78, 0.76, 0.56, 0.78, 0.72, 0.7 ,
       0.64, 0.64, 0.46, 0.36, 0.42, 0.48, 0.47, 0.54, 0.56, 0.52, 0.55,
       0.61, 0.57, 0.68, 0.78, 0.94, 0.96, 0.93, 0.84, 0.74, 0.72, 0.74,
       0.64, 0.44, 0.46, 0.5 , 0.96, 0.92, 0.92, 0.94, 0.76, 0.72, 0.66,
       0.64, 0.74, 0.64, 0.38, 0.34, 0.44, 0.36, 0.42, 0.48, 0.86, 0.9 ,
       0.79, 0.71, 0.64, 0.62, 0.57, 0.74, 0.69, 0.87, 0.91, 0.93, 0.68,
       0.61, 0.69, 0.62, 0.72, 0.59, 0.66, 0.56, 0.45, 0.47, 0.71, 0.94,
       0.94, 0.57, 0.61, 0.57, 0.64, 0.85, 0.78, 0.84, 0.92, 0.96, 0.77,
       0.71, 0.79, 0.89, 0.82, 0.76, 0.71, 0.8 , 0.78, 0.84, 0.9 , 0.92,
       0.97, 0.8 , 0.81, 0.75, 0.83, 0.96, 0.79, 0.93, 0.94, 0.86, 0.79,
       0.8 , 0.77, 0.7 , 0.65, 0.61, 0.52, 0.57, 0.53, 0.67, 0.68, 0.81,
       0.78, 0.65, 0.64, 0.64, 0.65, 0.68, 0.89, 0.86, 0.89, 0.87, 0.85,
       0.9 , 0.82, 0.72, 0.73, 0.71, 0.71, 0.68, 0.75, 0.72, 0.89, 0.84,
       0.93, 0.93, 0.88, 0.9 , 0.87, 0.86, 0.94, 0.77, 0.78, 0.73, 0.73,
       0.7 , 0.72, 0.73, 0.72, 0.97, 0.97, 0.69, 0.57, 0.63, 0.66, 0.64,
       0.68, 0.79, 0.82, 0.95, 0.96, 0.94, 0.93, 0.91, 0.85, 0.84, 0.74,
       0.76, 0.75, 0.76, 0.71, 0.67, 0.61, 0.63, 0.64, 0.71, 0.82, 0.73,
       0.74, 0.69, 0.64, 0.91, 0.88, 0.85, 0.86, 0.7 , 0.59, 0.6 , 0.65,
       0.7 , 0.76, 0.63, 0.81, 0.72, 0.71, 0.8 , 0.77, 0.74, 0.7 , 0.71,
       0.93, 0.85, 0.79, 0.76, 0.78, 0.77, 0.9 , 0.87, 0.71, 0.7 , 0.7 ,
       0.75, 0.71, 0.72, 0.73, 0.83, 0.77, 0.72, 0.54, 0.49, 0.52, 0.58,
       0.78, 0.89, 0.7 , 0.66, 0.67, 0.68, 0.8 , 0.81, 0.8 , 0.94, 0.93,
       0.92, 0.89, 0.82, 0.79, 0.58, 0.56, 0.56, 0.64, 0.61, 0.68, 0.76,
       0.86, 0.9 , 0.71, 0.62, 0.66, 0.65, 0.73, 0.62, 0.74, 0.79, 0.8 ,
       0.69, 0.7 , 0.76, 0.84, 0.78, 0.67, 0.66, 0.65, 0.54, 0.58, 0.79,
       0.8 , 0.75, 0.73, 0.72, 0.62, 0.67, 0.81, 0.63, 0.69, 0.8 , 0.43,
       0.8 , 0.73, 0.75, 0.71, 0.73, 0.83, 0.72, 0.94, 0.81, 0.81, 0.75,
       0.79, 0.58, 0.59, 0.47, 0.49, 0.47, 0.42, 0.57, 0.62, 0.74, 0.73,
       0.64, 0.63, 0.59, 0.73, 0.79, 0.68, 0.7 , 0.81, 0.85, 0.93, 0.91,
       0.69, 0.77, 0.86, 0.74, 0.57, 0.51, 0.67, 0.72, 0.89, 0.95, 0.79,
       0.39, 0.38, 0.34, 0.47, 0.56, 0.71, 0.78, 0.73, 0.82, 0.62, 0.96,
       0.96, 0.46, 0.53, 0.49, 0.76, 0.64, 0.71, 0.84, 0.77, 0.89, 0.82,
       0.84, 0.91, 0.67, 0.95])
```

```python
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=0)
```

```python
y_train=(y_train>0.5)
y_train
```

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

```python
y_test=(y_test>0.5)
y_test
```

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

```python
from sklearn.linear_model._logistic import LogisticRegression
cls=LogisticRegression(random_state=0)

lr=cls.fit(x_train,y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

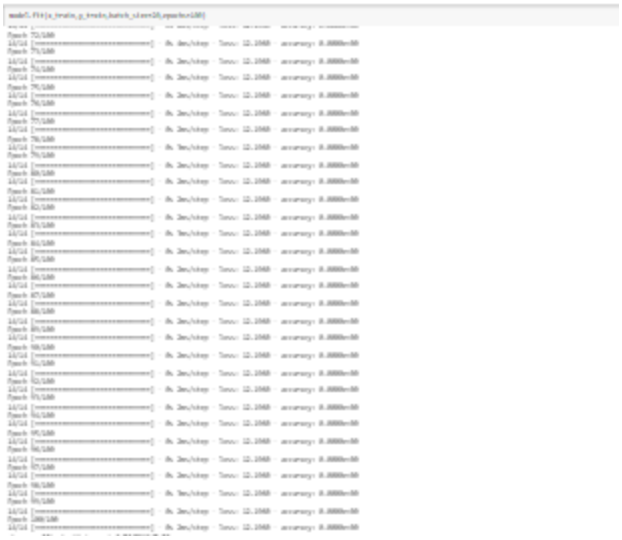
```
y_pred=lr.predict(x_test)
y_pred
```

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

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense,Activation,Dropout
from tensorflow.keras.optimizers import Adam
model=keras.Sequential()
model.add(Dense(7,activation='relu',input_dim=7))
model.add(Dense(7,activation='relu'))
model.add(Dense(7,activation='linear'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 7)                 56

 dense_1 (Dense)             (None, 7)                 56

 dense_2 (Dense)             (None, 7)                 56

=================================================================
Total params: 168
Trainable params: 168
Non-trainable params: 0
_________________________________________________________________
```

```
]  model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```



```
from sklearn.metrics import accuracy_score
train_predictions=model.predict(x_train)
print(train_predictions)
```

```
9/9 [==============================] - 0s 2ms/step
[[ -56.980873 -109.91886   200.0072   ...  -12.871788  -99.03662
  -216.30667 ]
 [ -63.92536  -139.01553   243.37392  ...  -12.543327 -120.13701
  -257.71216 ]
 [ -68.475555 -148.9906    260.70477  ...  -13.355347 -128.7235
  -276.09076 ]
 ...
 [ -57.243675 -112.03797   202.23146  ...  -12.219109 -100.34187
  -218.5444  ]
 [ -64.5374   -113.40946   211.84628  ...  -14.901016 -105.638916
  -233.77078 ]
 [ -64.86071  -133.83914   238.0581   ...  -13.47435  -117.72876
  -254.50139 ]]
```

```
train_acc=model.evaluate(x_train,y_train,verbose=0)[1]
print(train_acc)
```

```
0.0
```

```
test_acc=model.evaluate(x_test,y_test,verbose=0)[1]
print(test_acc)
```

```
0.0
```

```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
pred=model.predict(x_test)
pred=(pred>0.5)
pred
```

```
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
       [False, False,  True, False, False, False, False],
```

```python
from sklearn.metrics import accuracy_score,recall_score,roc_auc_score,confusion_matrix
y_test=(y_test>0.5)
print("\nAccuracy score: %f" %(accuracy_score(y_test,y_pred)*100))
print("Recall score : %f n" %(recall_score(y_test,y_pred)*100))
print("ROC score : %f\n" %(roc_auc_score(y_test,y_pred)*100))
print(confusion_matrix(y_test,y_pred))
```

```
Accuracy score: 89.166667
Recall score : 95.495495 n
ROC score : 53.303303

[[  1   8]
 [  5 106]]
```

```python
from sklearn.metrics import accuracy_score,recall_score,roc_auc_score,confusion_matrix
from sklearn.metrics import classification_report

print(y_train.shape)
print(y_pred.shape)


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

```
(280,)
(120,)
              precision    recall  f1-score   support

       False       0.17      0.11      0.13         9
        True       0.93      0.95      0.94       111

    accuracy                           0.89       120
   macro avg       0.55      0.53      0.54       120
weighted avg       0.87      0.89      0.88       120
```

```
from sklearn.metrics import accuracy_score,recall_score,roc_auc_score
print(y_train.shape)
print(y_pred.shape)


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

```
(280,)
(120,)
              precision    recall  f1-score   support

       False       0.17      0.11      0.13         9
        True       0.93      0.95      0.94       111

    accuracy                           0.89       120
   macro avg       0.55      0.53      0.54       120
weighted avg       0.87      0.89      0.88       120
```

```
model.save('model.h5')
```