# ARIGNAR ANNA GOVERNMENT ARTS COLLEGE VILLUPURAM

# INTELLIGENT ADMISSION THE FUTURE OF UNIVERSITY DECISION MAKING

Team ID : NM2023TMID16922

Team Leader : LOGESH S

Team Member : KARTHIKEYAN K

Team Member : ARAVINTH R

Team Member : SIVAGNANAM K

# ABSTRACT

The "Intelligent Admission: The Future of University Decision Making" project is a machine learning initiative aimed at enhancing the university admission process. The project proposes the use of artificial intelligence to streamline the admission process by analyzing various factors and making decisions based on data-driven insights. This abstract outlines the objectives and potential impact of the project, highlighting the benefits of adopting a more intelligent approach to university admissions. Through this project, universities can improve their admission process, increase efficiency, reduce bias, and ultimately make more informed decisions about their future students.

# INTRODUCTION

## Overview

Intelligent admission using machine learning is a promising area of research that aims to revolutionize the university admission process. This project involves the use of machine learning algorithms to analyze vast amounts of data and predict the likelihood of a student's admission based on a variety of factors.

The project involves collecting and processing data from various sources, such as standardized test scores, academic records, extracurricular activities, essays, and recommendations. This data is then used to train machine learning models that can accurately predict a student's likelihood of admission based on historical admission data and other relevant factors.

By leveraging the power of machine learning, universities can reduce bias in the admission process and make more informed decisions. This technology can also help universities improve student retention rates and ensure that students are matched with programs that are a good fit for their interests and abilities.

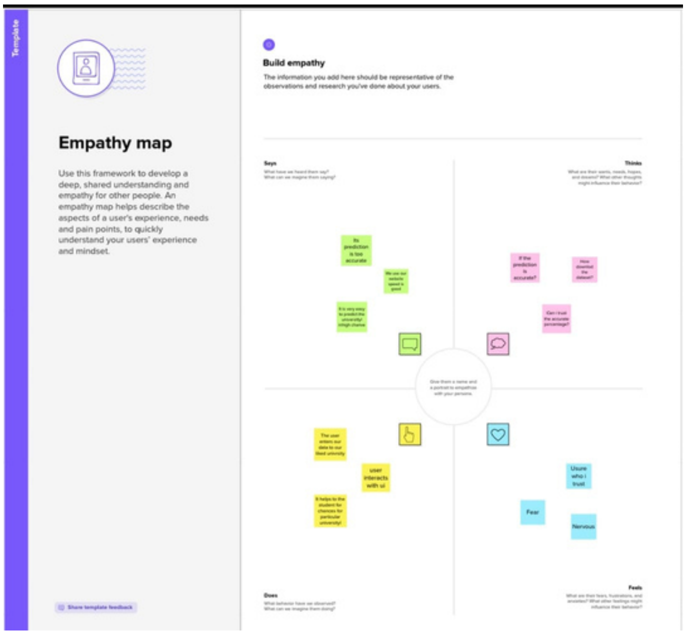
# Purpose

The purpose of an intelligent admission machine learning (ML) project is to improve the accuracy and efficiency of university admission decisions using artificial intelligence (AI) algorithms. By analyzing large amounts of student data, including academic achievements, extracurricular activities, and personal characteristics, an intelligent admission system can help universities make more informed and unbiased decisions about which applicants to admit.

The future of university decision-making is increasingly reliant on ML and AI technologies. These technologies have the potential to streamline the admission process, reduce bias, and provide universities with valuable insights into student performance and behavior.

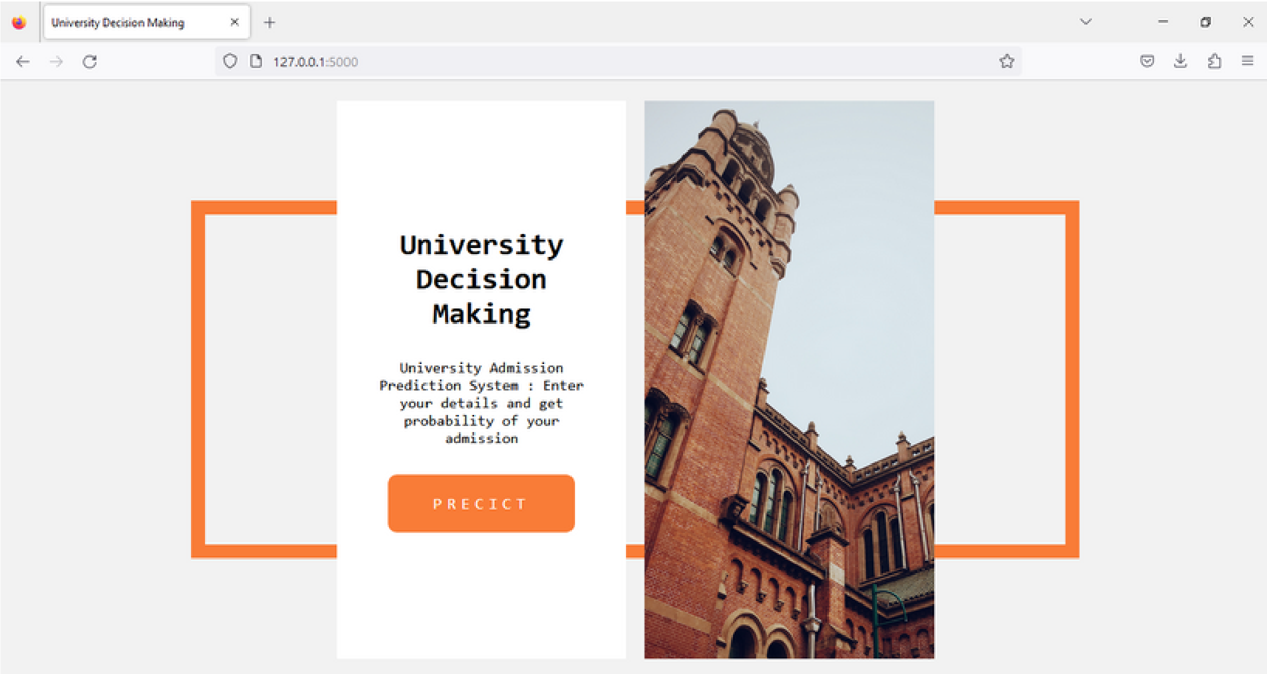
# PROBLEM DEFINITION & DESIGN THINKING

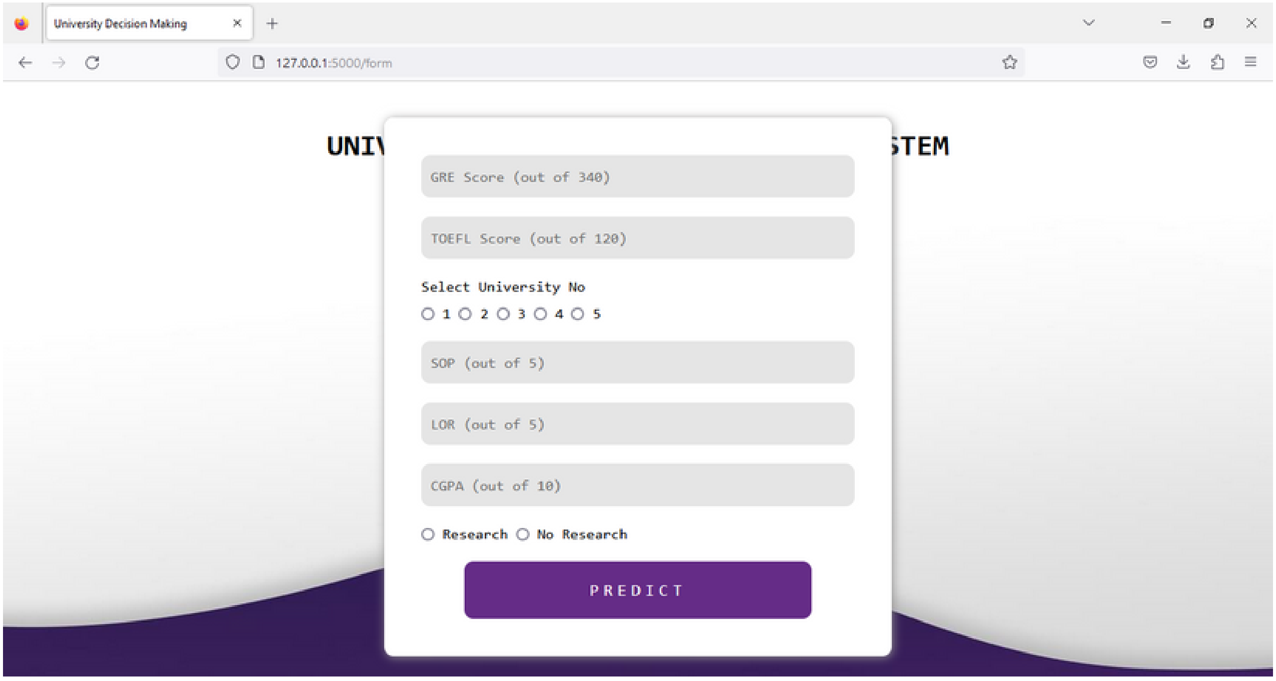
## Empathy Map

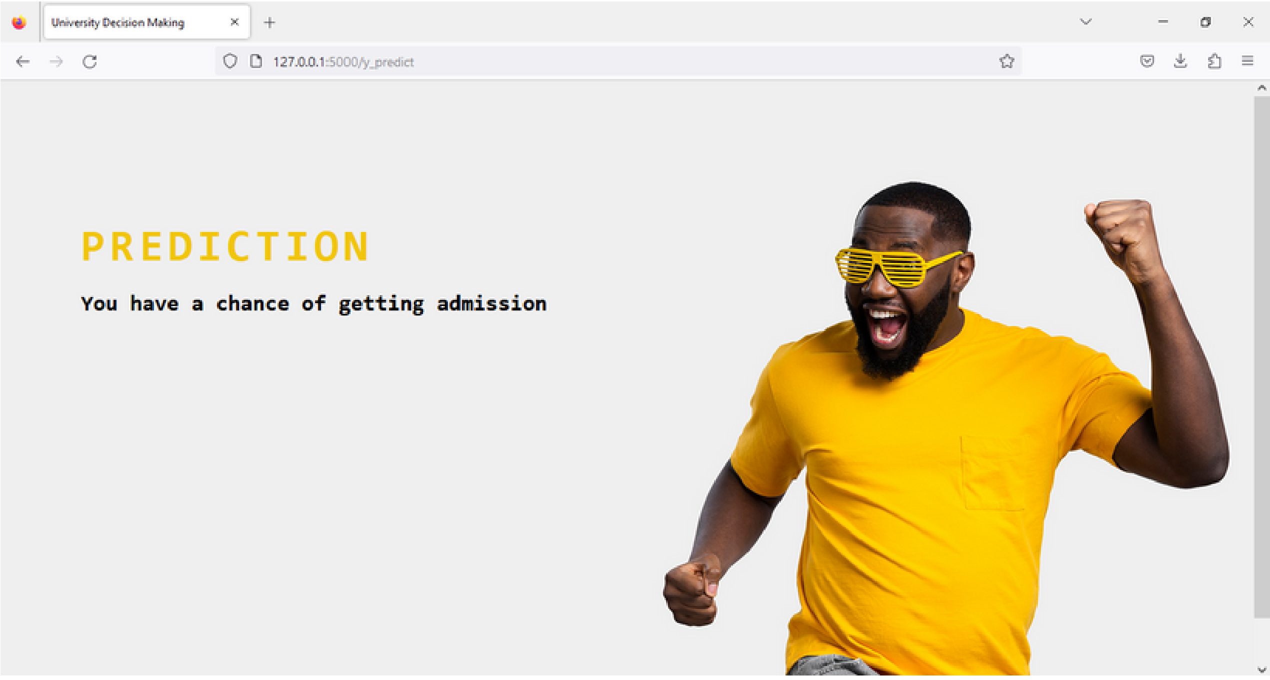


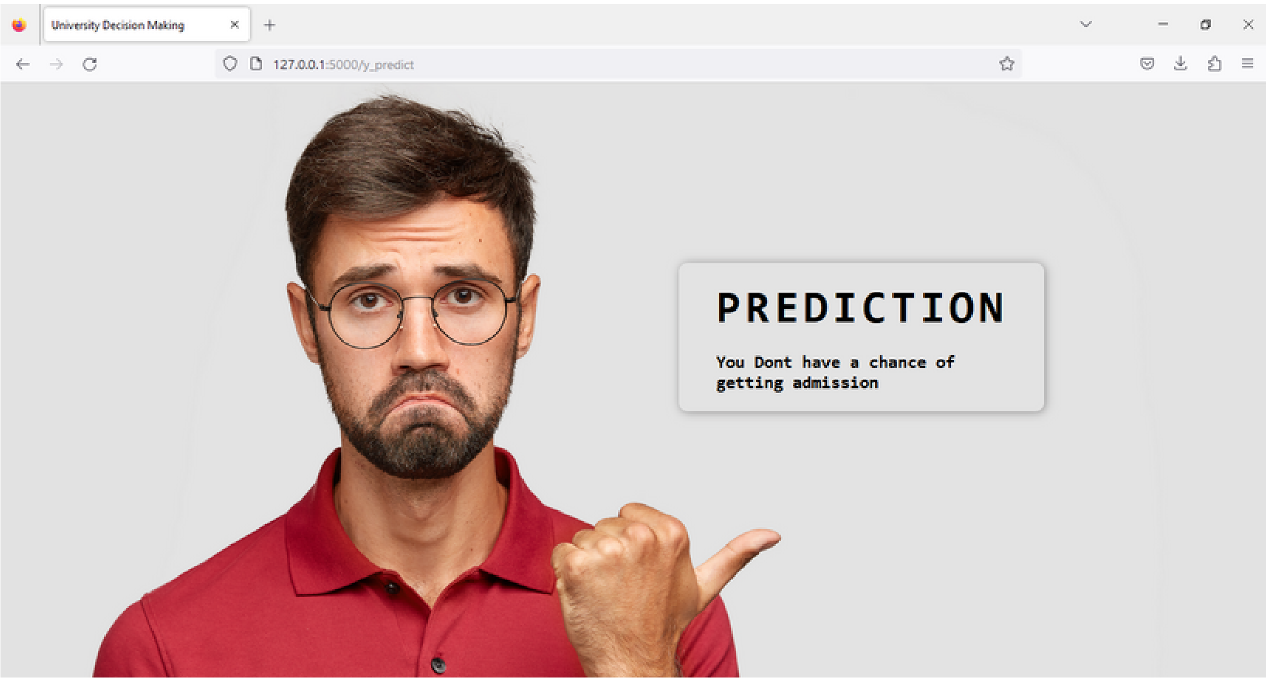

## Ideation & Brainstorm

# RESULT


University Decision Making
127.0.0.1:5000
University Decision Making
University Admission Prediction System : Enter your details and get probability of your admission
PRECICT


University Decision Making
127.0.0.1:5000/form
UNIV
STEM
GRE Score (out of 340)
TOEFL Score (out of 120)
Select University No
1 2 3 4 5
SOP (out of 5)
LOR (out of 5)
CGPA (out of 10)
Research No Research
PREDICT


University Decision Making
127.0.0.1:5000/y_predict
PREDICTION
You have a chance of getting admission


University Decision Making
127.0.0.1:5000/y_predict
PREDICTION
You Dont have a chance of getting admission

# Advantages

- Enhanced student experience
- Better retention rates
- Improved diversity
- Improved accuracy
- Increased efficiency

# Disadvantages

- Unforeseen errors
- Privacy concerns
- Cost
- Bias
- Lack of personal touch

# Applications

- Predictive Analytics
- Streamlined Application Process
- Personalized Recommendations
- Improved Documentation

# Future Scope

- Personalized Recommendations
- Enhanced Diversity and Inclusion
- Real-time Decision Making

# APPENDIX

## SOURCE CODE

app.py - D:\admission\Flask\app.py (3.10.0)

File Edit Format Run Options Window Help

```python
from flask import Flask, request, jsonify, render_template
import sklearn
import pickle
app = Flask(__name__)
model = pickle.load(open('university.pkl', 'rb'))


@app.route('/')
@app.route('/home')
def homePage():
    return render_template('index.html')

@app.route('/form')
def form():
    return render_template('form.html')

@app.route('/y_predict', methods=['POST'])
def y_predict():
    '''
    For rendering results on HTML GUI
    '''
    # min max scaling
    minl = [290.0, 92.0, 1.0, 1.0, 1.0, 6.8, 0.0]
    maxl = [340.0, 120.0, 5.0, 5.0, 5.0, 9.92, 1.0]
    k = [float(x) for x in request.form.values()]
    p = []
    for i in range(7):
        l = (k[i]-minl[i])/(maxl[i]-minl[i])
        p.append(l)
    prediction = model.predict([p])
    print(prediction)
    output = prediction[0]
    if (output == False):
        return render_template('noChance.html', prediction_text='You Dont have a chance of getting admission')
    else:
        return render_template('chance.html', prediction_text='You have a chance of getting admission')


if __name__ == "__main__":
    app.run(debug=True)
```

Ln: 16 Col: 0

# Source code

## Import Necessary Libraries

**Let us import necessary libraries to get started!**

In [1]:

```
#import necessary libraries
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import sklearn
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import  RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_s
core
```

## The Data

**Let's start by reading in the Admission_Predict.csv file into a pandas dataframe.**

In [2]:

```
#read_csv is a pandas function to read csv files
data = pd.read_csv(r"Admission_Predict.csv")
```

In [3]:

```
#head() method is used to return top n (5 by default) rows of a DataFrame or series.
data.head(8)
```

Out[3]:

| | 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 |
| **5** | 6 | 330 | 115 | 5 | 4.5 | 3.0 | 9.34 | 1 | 0.90 |
| **6** | 7 | 321 | 109 | 3 | 3.0 | 4.0 | 8.20 | 1 | 0.75 |
| **7** | 8 | 308 | 101 | 2 | 3.0 | 4.0 | 7.90 | 0 | 0.68 |

In [4]:

```
#let us drop Serial No. Column as it is not required for prediction
data.drop(["Serial No."],axis=1,inplace=True)
data.head()
```

Out[4]:

Out[4]:

| | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research | Chance of Admit |
|---|---|---|---|---|---|---|---|---|
| **0** | 337 | 118 | 4 | 4.5 | 4.5 | 9.65 | 1 | 0.92 |
| **1** | 324 | 107 | 4 | 4.0 | 4.5 | 8.87 | 1 | 0.76 |
| **2** | 316 | 104 | 3 | 3.0 | 3.5 | 8.00 | 1 | 0.72 |
| **3** | 322 | 110 | 3 | 3.5 | 2.5 | 8.67 | 1 | 0.80 |
| **4** | 314 | 103 | 2 | 2.0 | 3.0 | 8.21 | 0 | 0.65 |

# describe() method computes a summary of statistics like count, mean, standard deviation, min, max and quartile values.

In [5]:

```
data.describe()
```

Out[5]:

| | 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 |
| **mean** | 316.807500 | 107.410000 | 3.087500 | 3.400000 | 3.452500 | 8.598925 | 0.547500 | 0.724350 |
| **std** | 11.473646 | 6.069514 | 1.143728 | 1.006869 | 0.898478 | 0.596317 | 0.498362 | 0.142609 |
| **min** | 290.000000 | 92.000000 | 1.000000 | 1.000000 | 1.000000 | 6.800000 | 0.000000 | 0.340000 |
| **25%** | 308.000000 | 103.000000 | 2.000000 | 2.500000 | 3.000000 | 8.170000 | 0.000000 | 0.640000 |
| **50%** | 317.000000 | 107.000000 | 3.000000 | 3.500000 | 3.500000 | 8.610000 | 1.000000 | 0.730000 |
| **75%** | 325.000000 | 112.000000 | 4.000000 | 4.000000 | 4.000000 | 9.062500 | 1.000000 | 0.830000 |
| **max** | 340.000000 | 120.000000 | 5.000000 | 5.000000 | 5.000000 | 9.920000 | 1.000000 | 0.970000 |

**From the data we infer that there are only decimal values and no categorical values**

In [6]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   GRE Score          400 non-null    int64  
 1   TOEFL Score        400 non-null    int64  
 2   University Rating  400 non-null    int64  
 3   SOP                400 non-null    float64
 4   LOR                400 non-null    float64
 5   CGPA               400 non-null    float64
 6   Research           400 non-null    int64  
 7   Chance of Admit    400 non-null    float64
dtypes: float64(4), int64(4)
memory usage: 25.1 KB
```

In [7]:

```
#Let us rename the column Chance of Admit because it has trainling space
data=data.rename(columns = {'Chance of Admit ':'Chance of Admit'})
```

# Exploratory Data Analysis

## Missing Data

**We can use seaborn to create a simple heatmap to see where we have missing data!**

In [8]:

```
data.isnull().any()
```

Out[8]:

```
GRE Score            False
TOEFL Score          False
University Rating    False
SOP                  False
LOR                  False
CGPA                 False
Research             False
Chance of Admit      False
dtype: bool
```

**From the heatmap,we see that there are no missing values in the dataset**

In [9]:

```
data.corr()
```

Out[9]:

| | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research | Chance of Admit |
|---|---|---|---|---|---|---|---|---|
| **GRE Score** | 1.000000 | 0.835977 | 0.668976 | 0.612831 | 0.557555 | 0.833060 | 0.580391 | 0.802610 |
| **TOEFL Score** | 0.835977 | 1.000000 | 0.695590 | 0.657981 | 0.567721 | 0.828417 | 0.489858 | 0.791594 |
| **University Rating** | 0.668976 | 0.695590 | 1.000000 | 0.734523 | 0.660123 | 0.746479 | 0.447783 | 0.711250 |
| **SOP** | 0.612831 | 0.657981 | 0.734523 | 1.000000 | 0.729593 | 0.718144 | 0.444029 | 0.675732 |
| **LOR** | 0.557555 | 0.567721 | 0.660123 | 0.729593 | 1.000000 | 0.670211 | 0.396859 | 0.669889 |
| **CGPA** | 0.833060 | 0.828417 | 0.746479 | 0.718144 | 0.670211 | 1.000000 | 0.521654 | 0.873289 |
| **Research** | 0.580391 | 0.489858 | 0.447783 | 0.444029 | 0.396859 | 0.521654 | 1.000000 | 0.553202 |
| **Chance of Admit** | 0.802610 | 0.791594 | 0.711250 | 0.675732 | 0.669889 | 0.873289 | 0.553202 | 1.000000 |

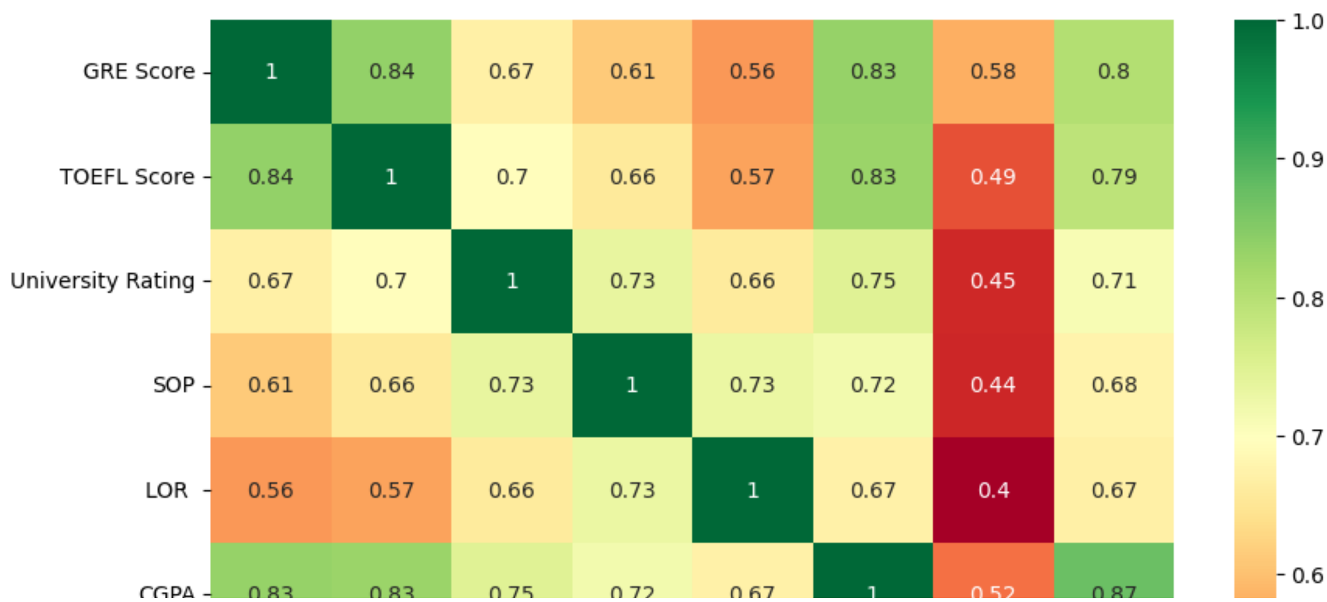
In [10]:

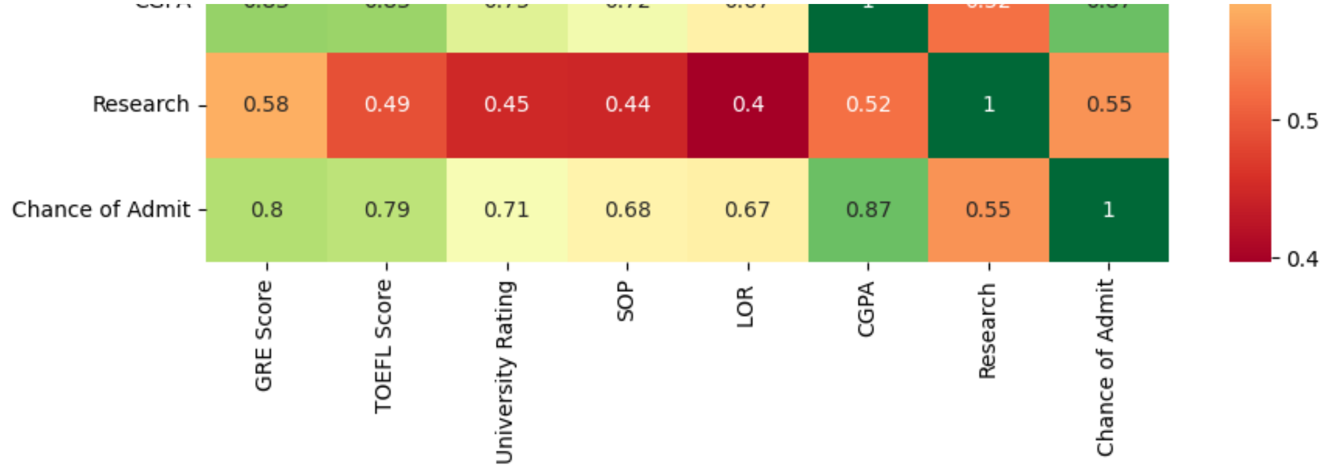
```
plt.figure(figsize=(10,7))

sns.heatmap(data.corr(),annot=True,cmap="RdYlGn")
```

Out[10]:

<AxesSubplot: >






**We see that the output variable "Chance of Admit" depends on CGPA,GRE,TOEFEL.The columns SOP,LOR and Reserach have less impact on university admission.**

In [11]:

```
sns.pairplot(data=data,hue='Research',markers=["^", "v"],palette='inferno')
```

Out[11]:

```
<seaborn.axisgrid.PairGrid at 0x267f509b3d0>
```

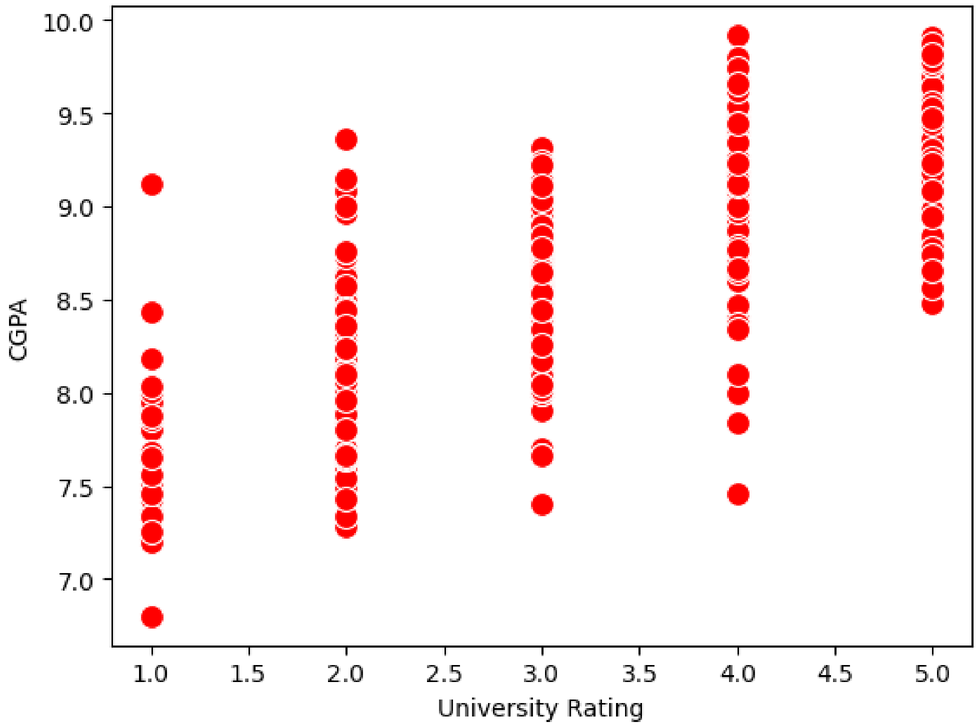
**Pair plot usually gives pair wise relationships of the columns in the dataset From the above pairplot we infer that 1.GRE score TOEFL score and CGPA all are linearly related to each other 2. Students in research score high in TOEFL and GRE compared to non research candidates.**

In [12]:

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

Out[12]:

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

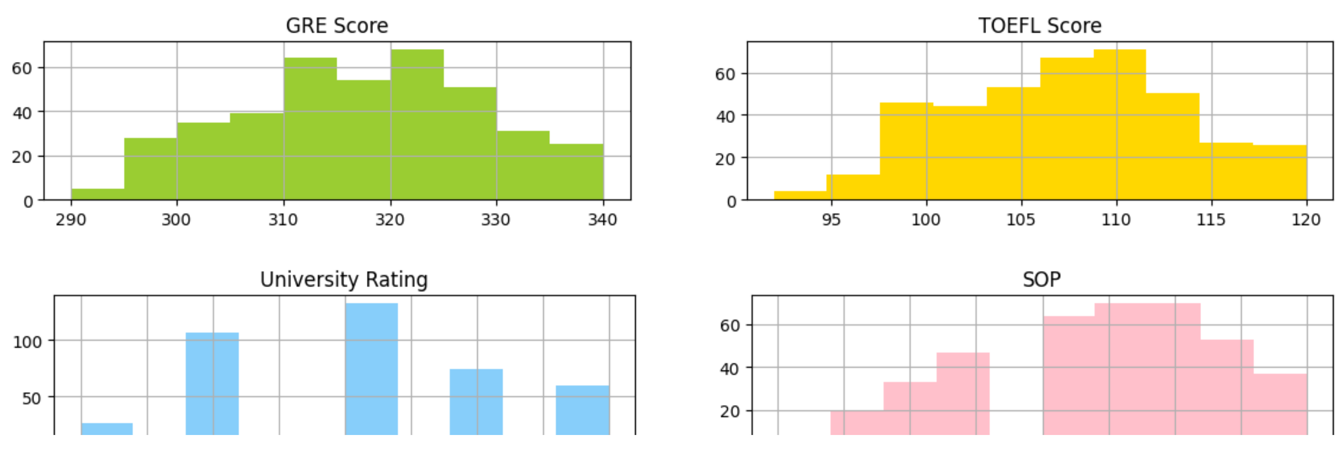


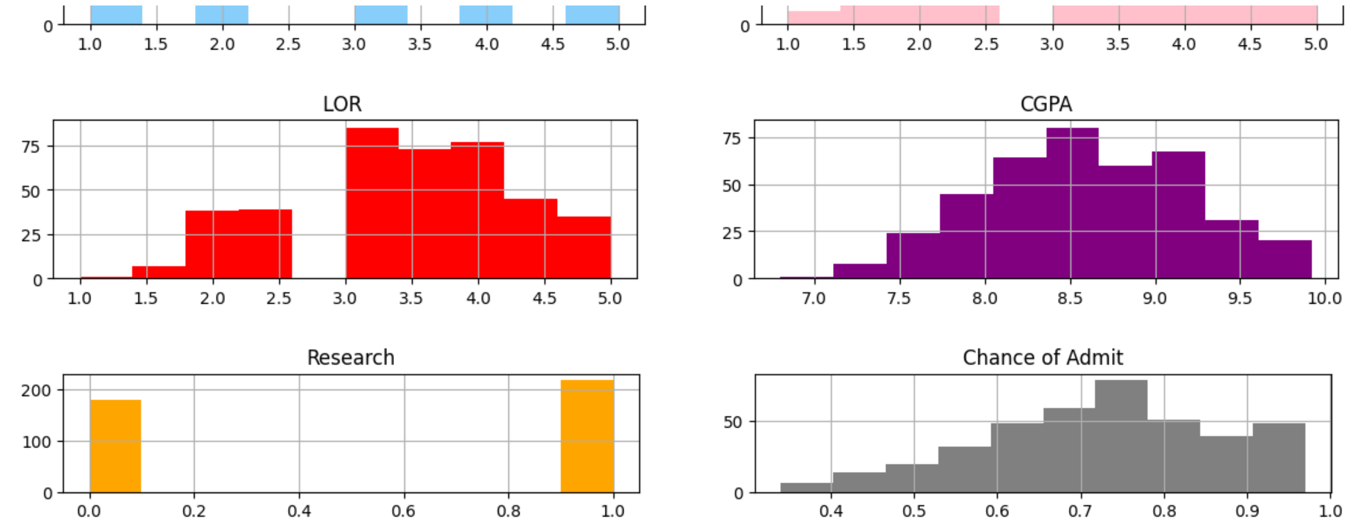

**From the above scatter plot we infer that as the CGPA increases the university ratings increases**

In [13]:

```
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[2*i]].hist(color=color[2*i],bins=10)
    plt.title(category[2*i])
    plt.subplot2grid((4,2),(i,1))
    data[category[2*i+1]].hist(color=color[2*i+1],bins=10)
    plt.title(category[2*i+1])

plt.subplots_adjust(hspace = 0.7, wspace = 0.2)
plt.show()
```






In [ ]:

In [14]:

```
print('Mean CGPA Score is :',int(data['CGPA'].mean()))
print('Mean GRE Score is :',int(data['GRE Score'].mean()))
print('Mean TOEFL Score is :',int(data['TOEFL Score'].mean()))
#print('Mean University rating is :',int(data[data['University Rating']<=500].University
Rating.mean()))
```

```
Mean CGPA Score is : 8
Mean GRE Score is : 316
Mean TOEFL Score is : 107
```

**The chance of admission is high if the aspirant score more than the above mean values**

# Machine Learning

### 1.Let's start by splitting the data into dependent and independent variable

In [15]:

```
data.head()
```

Out[15]:

| | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research | Chance of Admit |
|---|---|---|---|---|---|---|---|---|
| **0** | 337 | 118 | 4 | 4.5 | 4.5 | 9.65 | 1 | 0.92 |
| **1** | 324 | 107 | 4 | 4.0 | 4.5 | 8.87 | 1 | 0.76 |
| **2** | 316 | 104 | 3 | 3.0 | 3.5 | 8.00 | 1 | 0.72 |
| **3** | 322 | 110 | 3 | 3.5 | 2.5 | 8.67 | 1 | 0.80 |
| **4** | 314 | 103 | 2 | 2.0 | 3.0 | 8.21 | 0 | 0.65 |

In [16]:

```
x=data.iloc[:,0:-1].values
x
```

Out[16]:

```
array([[337.  , 118.  ,   4.  , ...,   4.5 ,   9.65,   1.  ],
       [324.  , 107.  ,   4.  , ...,   4.5 ,   8.87,   1.  ],
       [316.  , 104.  ,   3.  , ...,   3.5 ,   8.  ,   1.  ],
       ...,
       [330.  , 116.  ,   4.  , ...,   4.5 ,   9.45,   1.  ],
```

```
       [312.  , 103.  ,   3.  , ...,   4.  ,   8.78,   0.  ],
       [333.  , 117.  ,   4.  , ...,   4.  ,   9.66,   1.  ]])
```

In [17]:

```
y=data['Chance of Admit'].values
y
```

Out[17]:

```
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])
```

## 2.Data Normalisation

**There is huge disparity between the x values so we let us use feature scaling. Feature scaling is a method used to normalize the range of independent variables or features of data.**

In [18]:

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

Out[18]:

```
array([[0.94      , 0.92857143, 0.75      , ..., 0.875     , 0.91346154,
        1.        ],
       [0.68      , 0.53571429, 0.75      , ..., 0.875     , 0.66346154,
        1.        ],
       [0.52      , 0.42857143, 0.5       , ..., 0.625     , 0.38461538,
        1.        ],
       ...,
       [0.8       , 0.85714286, 0.75      , ..., 0.875     , 0.84935897,
        1.        ],
```

```
        [0.44      , 0.39285714, 0.5       , ..., 0.75      , 0.63461538,
         0.        ],
        [0.86      , 0.89285714, 0.75      , ..., 0.75      , 0.91666667,
         1.        ]])
```

## 3.Splitting our data into a training set and test set .

In [19]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.20,random_state=42)
#random_state acts as the seed for the random number generator during the split
```

In [20]:

```
y_train.shape
```

Out[20]:

```
(320,)
```

In [21]:

```
x_train
```

Out[21]:

```
array([[0.64      , 0.64285714, 0.5       , ..., 0.375     , 0.59935897,
         1.        ],
       [0.56      , 0.64285714, 0.5       , ..., 0.5       , 0.64102564,
         0.        ],
       [1.        , 1.        , 1.        , ..., 0.875     , 0.99679487,
         1.        ],
       ...,
       [0.32      , 0.46428571, 0.25      , ..., 0.5       , 0.45512821,
         1.        ],
       [0.24      , 0.25      , 0.        , ..., 0.25      , 0.14423077,
         0.        ],
       [0.48      , 0.5       , 0.25      , ..., 0.625     , 0.46474359,
         0.        ]])
```

**### Let us convert it into classification problem chance of admit>0.5 as true chance of admit<0.5 as false**

In [22]:

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

Out[22]:

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

In [23]:

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

In [24]:

```
y_test
```

Out[24]:

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

In [25]:

```
#Model building - Logistic Regression
def logreg(x_train,x_test,y_train,y_test):
    lr = LogisticRegression(random_state=0)
    lr.fit(x_train,y_train)
    y_lr_tr = lr.predict(x_train)
    print(accuracy_score(y_lr_tr,y_train))
    yPred_lr = lr.predict(x_test)
    print(accuracy_score(yPred_lr,y_test))
    print("***Logistic Regression***")
    print("Confusion_Matrix")
    print(confusion_matrix(y_test,yPred_lr))
    print("Classification Report")
    print(classification_report(y_test,yPred_lr))
```

In [26]:

```
#printing the train accuracy and test accuracy respectively
logreg(x_train,x_test,y_train,y_test)
```

```
0.934375
0.875
***Logistic Regression***
Confusion_Matrix
[[ 0 10]
 [ 0 70]]
Classification Report
              precision    recall  f1-score   support

       False       0.00      0.00      0.00        10
        True       0.88      1.00      0.93        70

    accuracy                           0.88        80
   macro avg       0.44      0.50      0.47        80
weighted avg       0.77      0.88      0.82        80
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\HP\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\HP\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [27]:

```
#testing on test & random input values
lr = LogisticRegression(random_state=0)
lr.fit(x_train,y_train)
print("Predicting on test values")
lr_pred =lr.predict(x_test)
print("output is: ",lr_pred)
print("Predicting on random input")
lr_pred_own = lr.predict(sc.transform([[337,118,4,4.5,4.5,9.65,1]]))
print("output is: ",lr_pred_own)
```

```
Predicting on test values
output is:  [ 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  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]
Predicting on random input
output is:  [ True]
```

In [28]:

```
#Model building - Decision Tree Classifier
def decisionTree(x_train,x_test,y_train,y_test):
    dtc = DecisionTreeClassifier(criterion="entropy",random_state=0)
    dtc.fit(x_train,y_train)
    y_dt_tr = dtc.predict(x_train)
    print(accuracy_score(y_dt_tr,y_train))
    yPred_dt = dtc.predict(x_test)
    print(accuracy_score(yPred_dt,y_test))
    print("***Decision Tree***")
    print("Confusion_Matrix")
    print(confusion_matrix(y_test,yPred_dt))
    print("Classification Report")
    print(classification_report(y_test,yPred_dt))
```

In [29]:

```
#printing the train accuracy and test accuracy respectively
decisionTree(x_train,x_test,y_train,y_test)
```

```
1.0
0.8875
***Decision Tree***
Confusion_Matrix
[[ 5  5]
 [ 4 66]]
Classification Report
              precision    recall  f1-score   support

       False       0.56      0.50      0.53        10
        True       0.93      0.94      0.94        70
```

```
    accuracy                           0.89        80
   macro avg       0.74      0.72      0.73        80
weighted avg       0.88      0.89      0.88        80
```

In [30]:

```
#testing on test & random input values
dtc = DecisionTreeClassifier(criterion="entropy",random_state=0)
dtc.fit(x_train,y_train)
print("Predicting on test values")
dtc_pred =dtc.predict(x_test)
print("output is: ",dtc_pred)
print("Predicting on random input")
dtc_pred_own = dtc.predict(sc.transform([[337,118,4,4.5,4.5,9.65,1]]))
print("output is: ",dtc_pred_own)
```

```
Predicting on test values
output is:  [ True  True  True  True  True  True  True False  True  True  True  True
 False  True  True  True  True False  True  True False  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 False  True  True  True  True  True
  True False  True False False  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True]
Predicting on random input
output is:  [ True]
```

In [31]:

```
#Model building - Random Forest Classifier
def RandomForest(x_tarin,x_test,y_train,y_test):
    rf = RandomForestClassifier(criterion="entropy",n_estimators=10,random_state=0)
    rf.fit(x_train,y_train)
    y_rf_tr = rf.predict(x_train)
    print(accuracy_score(y_rf_tr,y_train))
    yPred_rf = rf.predict(x_test)
    print(accuracy_score(yPred_rf,y_test))
    print("***Random Forest***")
    print("Confusion_Matrix")
    print(confusion_matrix(y_test,yPred_rf))
    print("Classification Report")
    print(classification_report(y_test,yPred_rf))
```

In [32]:

```
#printing the train accuracy and test accuracy respectively
RandomForest(x_train,x_test,y_train,y_test)
```

```
0.996875
0.9
***Random Forest***
Confusion_Matrix
[[ 2  8]
 [ 0 70]]
Classification Report
              precision    recall  f1-score   support

       False       1.00      0.20      0.33        10
        True       0.90      1.00      0.95        70

    accuracy                           0.90        80
   macro avg       0.95      0.60      0.64        80
weighted avg       0.91      0.90      0.87        80
```

In [33]:

```
#testing on test & random input values
rf = RandomForestClassifier(criterion="entropy",n_estimators=10,random_state=0)
```

```
rf.fit(x_train,y_train)
print("Predicting on test values")
rf_pred =rf.predict(x_test)
print("output is: ",rf_pred)
print("Predicting on random input")
rf_pred_own = rf.predict(sc.transform([[337,118,4,4.5,4.5,9.65,1]]))
print("output is: ",rf_pred_own)
```

```
Predicting on test values
output is:  [ 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  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 False  True  True  True  True  True]
Predicting on random input
output is:  [ True]
```

## ANN Model

In [34]:

```
# Importing the Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense
```

In [35]:

```
# Initialising the ANN
classifier = Sequential()
```

In [36]:

```
# Adding the input layer and the first hidden layer
classifier.add(Dense(units=7, activation='relu', input_dim=7))
```

In [37]:

```
# Adding the second hidden layer
classifier.add(Dense(units=7, activation='relu'))
```

In [38]:

```
# Adding the output layer
classifier.add(Dense(units=1, activation='linear'))
```

In [39]:

```
# Compiling the ANN
classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [40]:

```
# Fitting the ANN to the Training set
model = classifier.fit(x_train, y_train, batch_size=10, validation_split=0.33, epochs=20
)
```

```
Epoch 1/20
22/22 [==============================] - 1s 10ms/step - loss: 7.1056 - accuracy: 0.0748 -
val_loss: 4.1124 - val_accuracy: 0.0849
Epoch 2/20
22/22 [==============================] - 0s 3ms/step - loss: 3.4779 - accuracy: 0.0748 -
val_loss: 3.4566 - val_accuracy: 0.0849
Epoch 3/20
22/22 [==============================] - 0s 3ms/step - loss: 3.0676 - accuracy: 0.0748 -
val_loss: 3.2765 - val_accuracy: 0.0849
Epoch 4/20
```

```
22/22 [==============================] - 0s 3ms/step - loss: 2.9815 - accuracy: 0.0748 - val_loss: 3.1219 - val_accuracy: 0.0849
Epoch 5/20
22/22 [==============================] - 0s 3ms/step - loss: 2.8782 - accuracy: 0.0748 - val_loss: 3.0871 - val_accuracy: 0.0849
Epoch 6/20
22/22 [==============================] - 0s 4ms/step - loss: 2.8392 - accuracy: 0.0748 - val_loss: 2.8734 - val_accuracy: 0.0849
Epoch 7/20
22/22 [==============================] - 0s 3ms/step - loss: 2.8043 - accuracy: 0.0748 - val_loss: 2.7491 - val_accuracy: 0.0849
Epoch 8/20
22/22 [==============================] - 0s 3ms/step - loss: 2.7718 - accuracy: 0.0748 - val_loss: 2.7241 - val_accuracy: 0.0849
Epoch 9/20
22/22 [==============================] - 0s 3ms/step - loss: 2.6964 - accuracy: 0.0748 - val_loss: 2.6600 - val_accuracy: 0.0849
Epoch 10/20
22/22 [==============================] - 0s 3ms/step - loss: 2.6591 - accuracy: 0.0748 - val_loss: 2.6206 - val_accuracy: 0.0849
Epoch 11/20
22/22 [==============================] - 0s 3ms/step - loss: 2.5769 - accuracy: 0.0748 - val_loss: 2.3962 - val_accuracy: 0.0849
Epoch 12/20
22/22 [==============================] - 0s 3ms/step - loss: 2.5431 - accuracy: 0.0748 - val_loss: 2.3447 - val_accuracy: 0.0849
Epoch 13/20
22/22 [==============================] - 0s 3ms/step - loss: 2.4715 - accuracy: 0.0748 - val_loss: 2.2969 - val_accuracy: 0.0849
Epoch 14/20
22/22 [==============================] - 0s 3ms/step - loss: 2.3207 - accuracy: 0.0748 - val_loss: 2.2504 - val_accuracy: 0.0849
Epoch 15/20
22/22 [==============================] - 0s 3ms/step - loss: 2.2775 - accuracy: 0.0748 - val_loss: 2.2181 - val_accuracy: 0.0849
Epoch 16/20
22/22 [==============================] - 0s 2ms/step - loss: 2.1400 - accuracy: 0.0748 - val_loss: 2.1615 - val_accuracy: 0.0849
Epoch 17/20
22/22 [==============================] - 0s 3ms/step - loss: 2.0751 - accuracy: 0.0748 - val_loss: 1.9292 - val_accuracy: 0.0849
Epoch 18/20
22/22 [==============================] - 0s 2ms/step - loss: 1.9991 - accuracy: 0.0748 - val_loss: 1.8918 - val_accuracy: 0.0849
Epoch 19/20
22/22 [==============================] - 0s 2ms/step - loss: 1.9652 - accuracy: 0.0748 - val_loss: 1.8571 - val_accuracy: 0.0849
Epoch 20/20
22/22 [==============================] - 0s 3ms/step - loss: 1.9359 - accuracy: 0.0748 - val_loss: 1.8307 - val_accuracy: 0.0849
```

In [41]:

```python
ann_pred = classifier.predict(x_test)
ann_pred = (ann_pred>0.5)
print(accuracy_score(ann_pred,y_test))
print("***ANN Model***")
print("Confusion_Matrix")
print(confusion_matrix(y_test,ann_pred))
print("Classification Report")
print(classification_report(y_test,ann_pred))
```

```
3/3 [==============================] - 0s 0s/step
0.125
***ANN Model***
Confusion_Matrix
[[10  0]
 [70  0]]
Classification Report
              precision    recall  f1-score   support

       False       0.12      1.00      0.22        10
```

```
        True       0.00      0.00      0.00        70

    accuracy                           0.12        80
   macro avg       0.06      0.50      0.11        80
weighted avg       0.02      0.12      0.03        80
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\HP\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\HP\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [42]:

```python
#testing on test & random input values
print("Predicting on test input")
ann_pred = classifier.predict(x_test)
ann_pred = (ann_pred>0.5)
print("output is: ",ann_pred)
print("Predicting on random input")
ann_pred_own = classifier.predict(sc.transform([[337,118,4,4.5,4.5,9.65,1]]))
ann_pred_own = (ann_pred_own>0.5)
print("output is: ",ann_pred_own)
```

```
Predicting on test input
3/3 [==============================] - 0s 0s/step
output is:  [[False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
```

```
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]
 [False]]
Predicting on random input
1/1 [==============================] - 0s 15ms/step
output is:  [[False]]
```

In [43]:

```python
ann_pred_train = classifier.predict(x_train)
ann_pred_train = (ann_pred_train>0.5)
print(accuracy_score(ann_pred_train,y_train))
print("***ANN Model***")
print("Confusion_Matrix")
print(confusion_matrix(ann_pred_train,y_train))
print("Classification Report")
print(classification_report(ann_pred_train,y_train))
```

```
10/10 [==============================] - 0s 889us/step
0.078125
***ANN Model***
Confusion_Matrix
[[ 25 295]
 [  0   0]]
Classification Report
              precision    recall  f1-score   support

       False       1.00      0.08      0.14       320
        True       0.00      0.00      0.00         0

    accuracy                           0.08       320
```

```
    accuracy                           0.08       320
   macro avg       0.50      0.04      0.07       320
weighted avg       1.00      0.08      0.14       320
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_cl
assification.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defined and bein
g set to 0.0 in labels with no true samples. Use `zero_division` parameter to control thi
s behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\HP\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_cl
assification.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defined and bein
g set to 0.0 in labels with no true samples. Use `zero_division` parameter to control thi
s behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\HP\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_cl
assification.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defined and bein
g set to 0.0 in labels with no true samples. Use `zero_division` parameter to control thi
s behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [44]:

```
pickle.dump(lr,open('university.pkl','wb'))
```

In [ ]:

```

```