INTELLIGENT ADMISSION:

THE FUTURE OF UNIVERSITY DECISION MAKING

WITH MACHINE LEARNING

TEAM MEMBERS

- **HEMASRI.S**
- **SIVARANJINI.E**
- **JAYABHARATHI.B**
- **ARCHANA.S**

CLASS :BSC COMPUTER SCIENCE

TABLE OF INDEX

1.INTRODUCTION

1.1 OVERVIEW

- University admission is a process by which students are selected to attend a college or university.the process typically involve several steps,including submitting an application,taking entrance exam and participating in interviews or other evaluations.

- Students are often worried about their chances in admission in university.the university admission process for students can be demanding,but by being well-informed, prepared and organized students can increase their chances of admitted of university to their choice.

1.2 PURPOSE

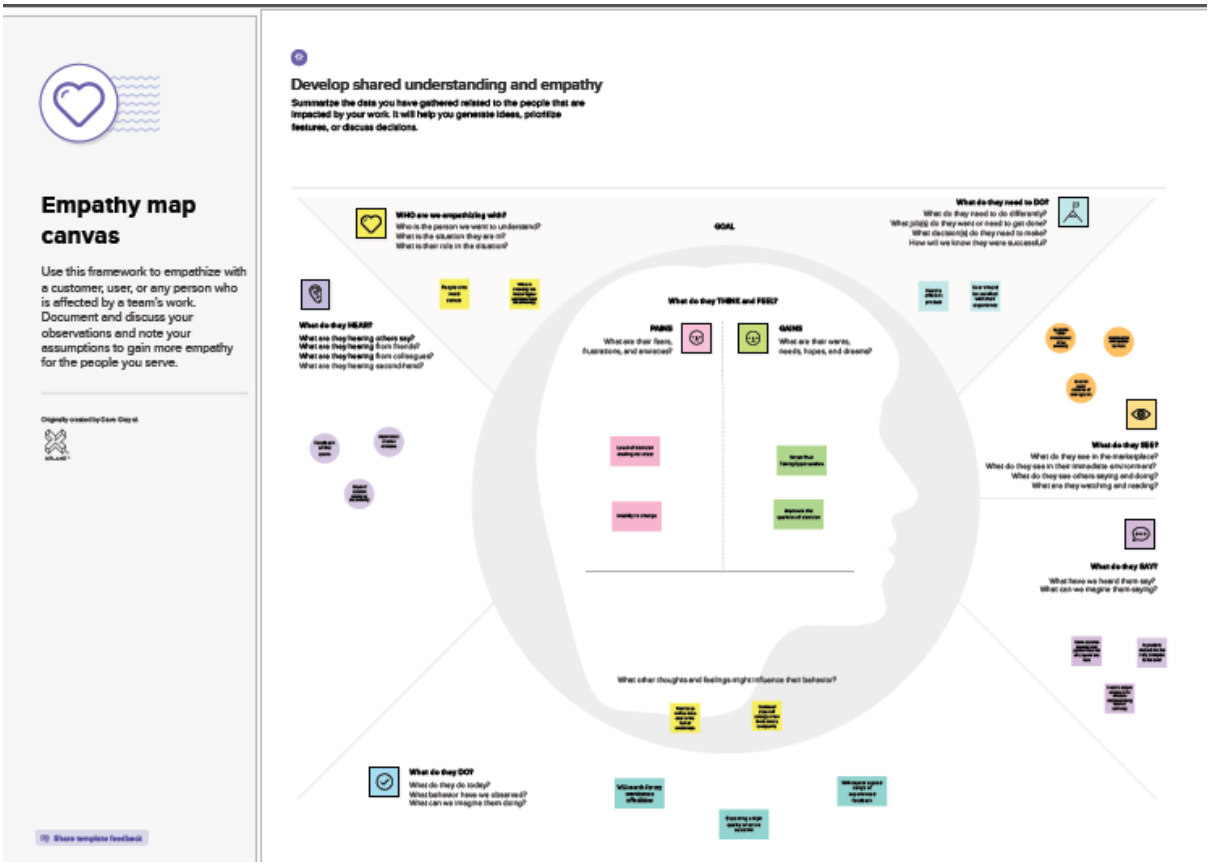
- The aim of this project is to help students in shortlisting the universities with their profiles.machine learning algorithm are then used to train a model on this data,which can be used to predict the chances of future applicants being admitted.with this project students can make more informed decisions about which universities to apply to,and universities can make more use of their resources by focussing on the most promising applicants.

- The predicted output gives them the fair idea about their admission chances in a particular university.this analysis should also help students who currently preparing or will be preparing to get a better idea.

2.PROBLEM DEFINITION & DESIGN THINKING

2.1 EMPATHY MAP

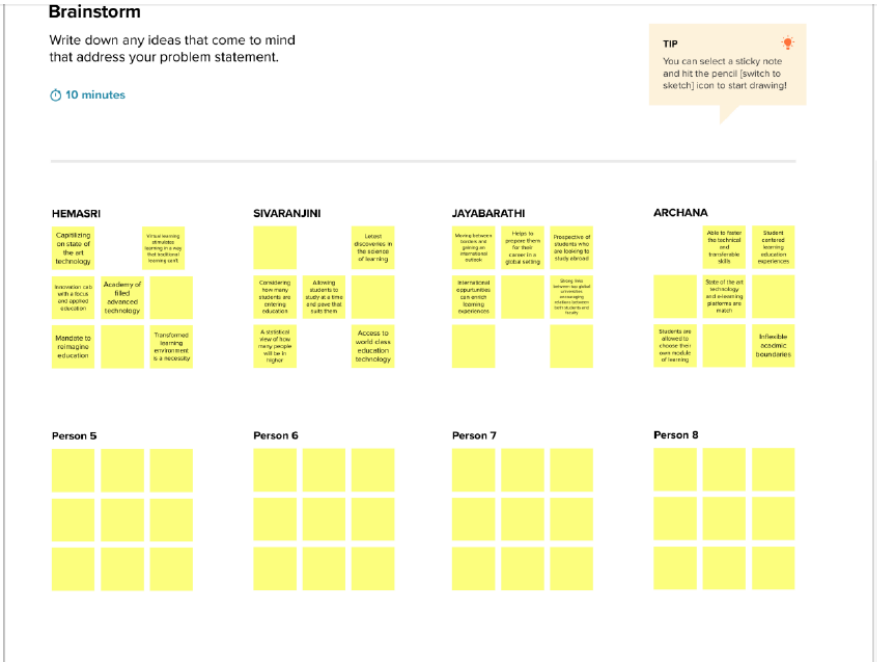
In the ideation phase we have empathized as our client Optimizing spam filtering with machine learning and we have acquired the details which are represented in the Empathy Map given below.



\

2.2IDEATION & BRAINSTORMING MAP

Under this activity our team members have gathered and discussed various idea to solve our project problem. Each member contributed 6 to 10 ideas after gathering all ideas we have assessed the impact and feasibility of each point. Finally, we have assign the priority for each point based on the impact value.



3.RESULT

Descriptive statistical

- Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

Admission.describe()

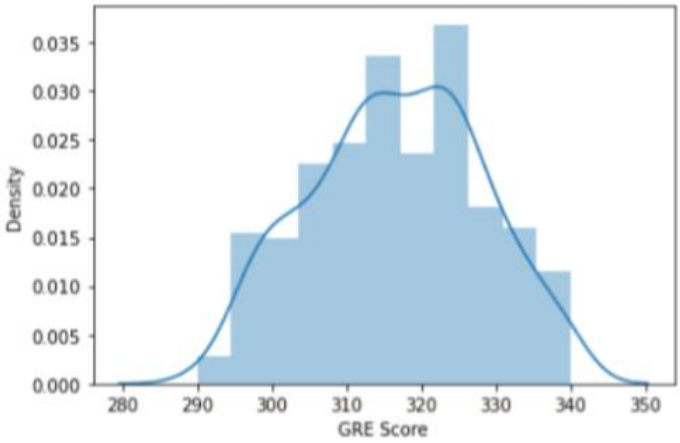
	Serial no.	GRES Score	TOEFL Score	University RATING	SOP	LOR	CGPA	Research	Chance of Admit
Count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.0000000	
Mean	200.500000	316.807500	107.410000	3.087500	3.400000	3.452500	8.598925	0.547500	0.724350
Std	115.614301	11.473646	6.09514	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	0.6400000
50%	200.500000	317.000000	107.000000	3.0000000	3.500000	3.5000000	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	5.0000000	5.000000	5.000000	9.9200000	1.000000	0.970000

.Activity 2.1: Univariate analysis

- In simple words, univariate analysis is understanding the data with a single feature. Here we have displayed two different graphs such as distplot and countplot.
- The Seaborn package provides a wonderful function distplot. With the help of distplot, we can find the distribution of the feature. To make multiple graphs in a single plot, we use subplot.

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

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarn
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7fb383bf3e80>
```



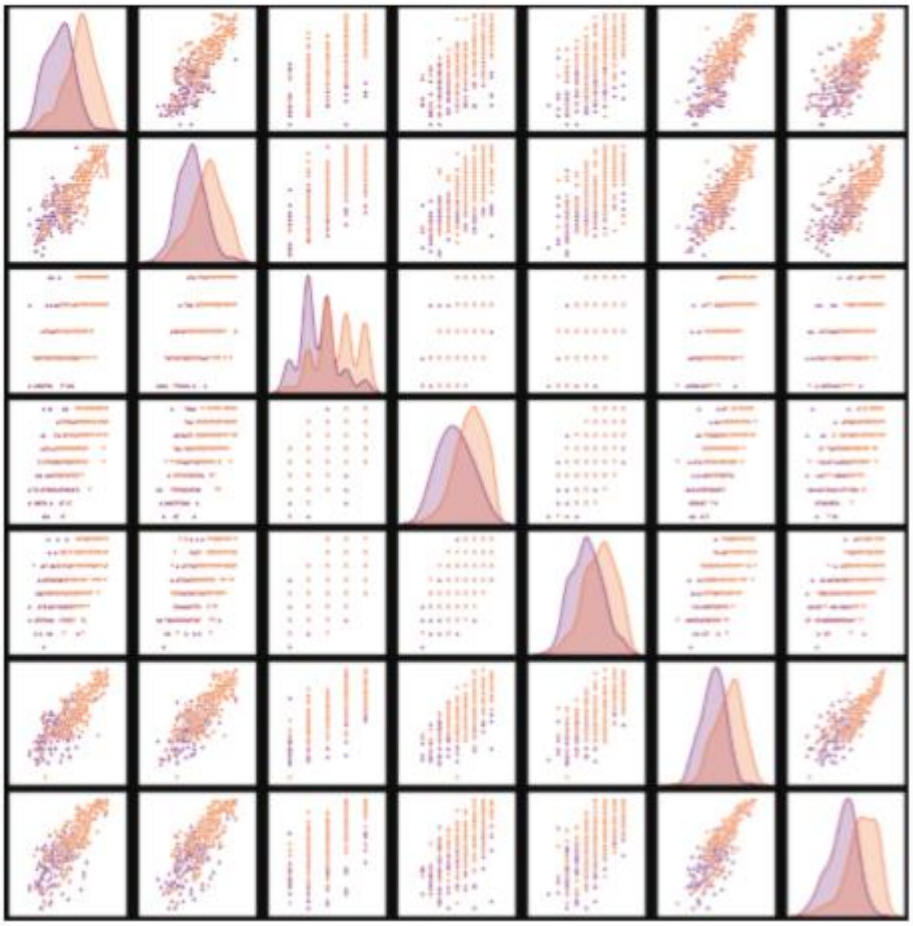
Bivariate analysis

```
plt.figure(figsize=(10, 10))
sns.heatmap(admission.corr(), annot=True, linewidths=0.05, fmt= '.2f',cmap="magma")
plt.show()
```

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

Pair Plot: Plot pairwise relationships in a dataset.

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



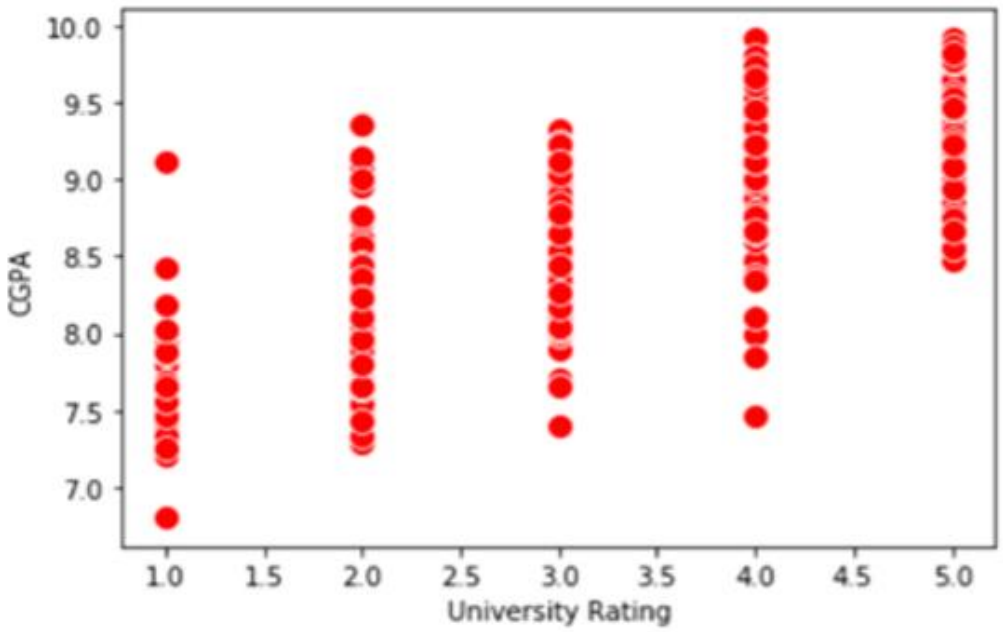
Pair plot usually gives pair wise relationships of the columns in the dataset 1.GRE score TOEFL score and CGPA all are linearly related to each other 2. Students in research score high in TOEFL and GRE compared to non research candidates

Scatter Plot:

- Matplot has a built-in function to createscatterplots called scatter(). A scatter plot is a type of plot that shows the data as a collection of points

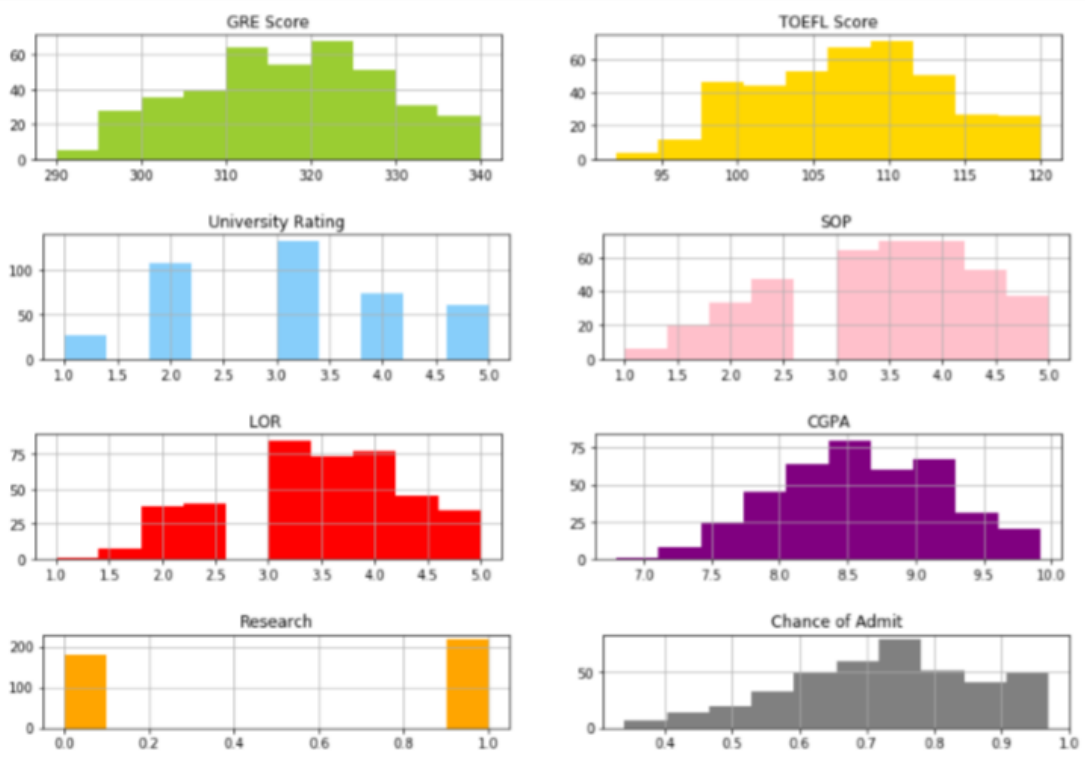
Sns.scatterplot(x='University Rating',y='CGPA',data=data,color='Red', s=100)

```
<matplotlib.axes._subplots.AxesSubplot at 0x2b6e49feec8>
```



Visualizing the Each column in a dataset using subplot( ).

```
category=admission.columns
color=['Red','Pink','Orange','Yellow','Purple','Green','Blue','Brown']
start=True
for i in np.arange(4):
fig=plt.figure(figsize=(14,8))
plt.subplot2grid((4,2),(i,0))
admission[category[2*i]].hist(color=color[2*i],bins=10)
plt.title(category[2*i])
plt.subplot2grid((4,2),(i,1))
admission[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()
```




```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
X_train[X_train.columns] = scaler.fit_transform(X_train[X_train.columns])
X_test[X_test.columns] = scaler.transform(X_test[X_test.columns])
X_train.head()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
189	0.68	0.714286	1.00	1.00	1.000	0.733119	1.0
258	0.72	0.357143	0.75	1.00	1.000	0.630225	1.0
287	0.68	0.785714	1.00	1.00	0.875	0.733119	1.0
61	0.34	0.321429	0.50	0.75	0.500	0.450161	0.0

SCALING THE DATA:

- Scaling is one the important process, we have to perform on the dataset, because of data measures in different ranges can leads to mislead in prediction.
- Models such as KNN, Logistic regression need scaled data, as they follow distance based method and Gradient Descent concept.

```
From sklearn.preprocessing import MinMaxScaler

Sc=MinMaxScaler

x=sc.fit_transform(x)

x

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.        ]])


X=admission.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=admission.iloc[:,7:].values
y
```

```
array([[1. , 0.92],
 [1. , 0.76],
 [1. , 0.72],
[1. , 0.8 ],
[0. , 0.65],
[1. , 0.9 ],
[1. , 0.75],
[0. , 0.68],
[0. , 0.5 ],
[0. , 0.45],
[1. , 0.52],
[1. , 0.84],
[1. , 0.78],
[1. , 0.62],
[1. , 0.61],
[0. , 0.54],
[0. , 0.66],
[1. , 0.65],
[0. , 0.63],
[0. , 0.62],
[1. , 0.64],
[0. , 0.7 ],
```

[1. , 0.94],

[1. , 0.95],

[1. , 0.97],

...

[1. , 0.82],

[1. , 0.84],

#random_state acts as the seed for the random number generator during the split

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=101)

X_train.shape

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using the train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

y_train=(y_train>0.5)

y_train

array([[True, True],

[False, True],

[True, True],

[True, True],

[True, False],

[True, True],

[True, True],

[True, True],

[False, True],

[True, True],

[True, True],

[False, True],

[True, True],

[True, True],

[False, True],

[False, False],

[True, False],

[False, True],

[True, True],

[False, True],

[False, False],

[False, True],

[False, True],

[True, True],

[True, True],

...

[True, True],

[False, True],

[True, True],

[True, True],

[True, True]])

We will perform scaling only on the input values.Once the dataset is scaled, it will be converted into an array and we need to convert it back to a dataframe.

Model Building

Activity 1: Training the model in multiple algorithms Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved based on its performance.

Activity 1.1: logistic Regression Model

- A LogisticRegression algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done

```
From sklearn.linear_model.logistic import LogisticRegression
Cls =LogisticRegression(random_state =0)
xg.score(X_test,y_test)
```

0.801633431009059

```
y_predict=rgr.predict(X_test)
y_predict
```

Array ([0.8358, 0.7768, 0.7005, 0.8156, 0.7636, 0.5561, 0.6297, 0.7655,
0.8587, 0.5996, 0.691 , 0.9493, 0.6728, 0.5955, 0.8783, 0.6703,
0.529 , 0.7117, 0.7959, 0.6556, 0.9094, 0.4648, 0.8739, 0.689 ,
0.9406, 0.5631, 0.4684, 0.9275, 0.54 , 0.8075, 0.5971, 0.6374,
0.839 , 0.6529, 0.9547, 0.9266, 0.9216, 0.6313, 0.7951, 0.8524,
0.73 , 0.7498, 0.546 , 0.5373, 0.8226, 0.6759, 0.7941, 0.6854,
0.725 , 0.9469, 0.8812, 0.577 , 0.9266, 0.8593, 0.6093, 0.6007,
0.5982, 0.6177, 0.5845, 0.7914, 0.4839, 0.6679, 0.937 , 0.7124,
0.6902, 0.6632, 0.7815, 0.6734, 0.721 , 0.6084, 0.7038, 0.7347,
0.9004, 0.7146, 0.7242, 0.6956, 0.7828, 0.8451, 0.9127, 0.8355])

Activity 1.5:

ANN model

Building and training an Artificial Neural Network (ANN) using the Keras library with TensorFlow as the backend. The ANN is initialised as an instance of the Sequential class, which is a linear stack of layers. Then, the input layer and two hidden layers are added to the model using the Dense class, where the number of units and activation function are specified. The output layer is also added using the Dense class with a sigmoid activation function. The model is then compiled with the Adam optimizer, binary cross-entropy loss function, and accuracy metric. Finally, the model is fit to the training data with a batch size of 100, 20% validation split, and 100 epochs.

```
#Libraries to train neural network
import tensorflow as tf
from tensorflow.python import keras
from keras import layers
from keras.layers import Activation,Dense,Dropout
# Initialize the model
classifier=keras.Sequential()
#Add input layer
classifier.add(Dense(7, activation='relu', input_dim=7))
#Add hidden layer
classifier.add(Dense (7, activation='relu'))
#Add output layer
classifier.add(Dense(1, activation='linear'))
classifier.compile(optimizer='adam',loss='binary_crossentropy', metrics=['accuracy'])
classifier.summary()
```

```
Model:                                                    "sequential_4"
_________________________________________________________________
Layer   (type)                 Output   Shape                  Param   #
=================================================================
module_wrapper_12          (ModuleWr  (None,  7)                56
_________________________________________________________________
module_wrapper_13        (ModuleWra    (None,  7)                56
_________________________________________________________________
```

module_wrapper_14 (ModuleWra (None, 1) 8

===

Total params: 120

Trainable params: 120

Non-trainable params: 0

```
model_history=classifier.fit(X_train,y_train,batch_size=20,epochs=100)
```

```
Epoch 1/100
 14/14 [==============================] - 0s 2ms/step - loss: 3.6762 - accuracy: 0.7589
Epoch 2/100
 14/14 [==============================] - 0s 2ms/step - loss: 3.6762 - accuracy: 0.7589
 Epoch 3/100
 14/14 [==============================] - 0s 2ms/step - loss: 3.6762 - accuracy: 0.7589
Epoch 4/100
 14/14 [==============================] - 0s 2ms/step - loss: 3.6762 - accuracy: 0.7589
 Epoch 5/100
 14/14 [==============================] - 0s 2ms/step - loss: 3.6762 - accuracy: 0.7589
 Epoch 6/100
 14/14 [==============================] - 0s 3ms/step - loss: 3.6762 - accuracy: 0.7589
 Epoch 7/100
 14/14 [==============================] - 0s 2ms/step - loss: 3.6762 - accuracy: 0.7589
 Epoch 8/100
 14/14 [==============================] - 0s 2ms/step - loss: 3.6762 - accuracy: 0.7589
 Epoch 9/100
```

14/14 [==============================] - 0s 2ms/step - loss: 3.6762 - accuracy: 0.7589

Epoch 10/100

14/14 [==============================] - 0s 2ms/step - loss: 3.6762 - accuracy: 0.7589

Epoch 11/100

14/14 [==============================] - 0s 2ms/step - loss: 3.6762 - accuracy: 0.7589

Epoch 12/100

14/14 [==============================] - 0s 3ms/step - loss: 3.6762 - accuracy: 0.7589

Epoch 13/100

...

Epoch 99/100

14/14 [==============================] - 0s 2ms/step - loss: 3.6762 - accuracy: 0.7589

Epoch 100/100

14/14 [==============================] - 0s 3ms/step - loss: 3.6762 - accuracy: 0.7589

Activity 2:

Testing the model

In ANN we first have to save the model to the test the inputs

[True], from sklearn.metrics import accuracy_score,classification_report

train_predictions=classifier.predict(X_test)

[[192.80466]

[304.48224]

[270.64758]

```
 [239.98091]
 [245.92453]
 [252.86273]
 [237.20155]
 [215.7854 ]
 [256.528 ]
 [248.01389]
 [266.50876]
 [299.5299 ]
 [255.74731]
 [249.22113]
 [307.9131 ]
 [246.73666]
 [292.78827]
 [240.14366]
 [294.4889 ]
 [187.8512 ]
 [278.0559 ]
 [296.19464]
 [315.9665 ]
 [284.73718]
 [292.48248]
...
[242.09091]
 [275.39032]
 [241.89087]
 [263.56863]]

train_acc=classifier.evaluate(X_train, y_train, verbose=0)[1]
print(train_acc)
```

0.7589285969734192

```
test_acc=classifier.evaluate(X_test,y_test,verbose=0)[1]
print(test_acc)
```

0.6625000238418579

```
pred=classifier.predict(X_test)
pred=(pred>0.5)
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],

...

[True],

[True],

[True],

[True]])

4.ADVANTAGES & DISADVANTAGES OF PROPOSED SOLUTION

ADVANTAGES:

- Gives more information
- Previous people's participation
- Provides more alternatives
- Improves the degree of acceptance and commitment
- Improves the quality of decision

DISADVANTAGES:

- A significant amount of time to complete
- Receive irrelevant opinions and ideas
- Refuses to share peoples perspectives
- The final choice can go against the outcomes
- Reduced amount of accountability

5.APPLICATIONS

ADMISSION PREDICTION APPLICATIONS:

Admission predict has many applications in various fields,including education,business,and healthcare.Here are some examples:

- **Educational Institutions**: Admission prediction is commonly used in universities and colleges to predict the likelihood of students being admitted. By analyzing various factors such as academic performance, standardized test scores, and extracurricular activities, admission officers can predict which students are likely to be admitted to the institution.

- **Scholarship Programs:** Admission prediction can be used to predict the likelihood of students being awarded scholarships. By analyzing a student's academic performance, extracurricular activities, and other factors, scholarship committees can determine which students are most likely to receive the award.

- **Student Recruitment**: Admission prediction can be used to identify potential students who are likely to apply to a particular institution. This can help universities and colleges to target their recruitment efforts more effectively.

- **Business**: Admission prediction can be used by businesses to predict the likelihood of job applicants being hired. By analyzing factors such as work experience, education, and skills, employers can predict which candidates are most likely to be successful in a particular job.
- **Healthcare**: Admission prediction can be used in healthcare to predict the likelihood of patients being admitted to hospitals. By analyzing various factors such as age, medical history, and severity of illness, healthcare providers can predict which patients are likely to require hospitalization.

- **Credit Approval**: Admission prediction can be used by financial institutions to predict the likelihood of loan applicants being approved. By analyzing factors such as credit history, income, and debt, lenders can predict which applicants are most likely to repay the loan.

- **Immigration**: Admission prediction can be used by immigration authorities to predict the likelihood of visa applicants being approved. By analyzing factors such as work experience, education, and language proficiency, immigration authorities can predict which applicants are most likely to meet the requirements for a particular visa.

- **Sports**: Admission prediction can be used in sports to predict the likelihood of athletes being recruited by teams. By analyzing factors such as athletic ability, statistics, and sperformance in competitions, sports teams can predict which athletes are most likely to be successful at the professional level.

- **Crime**: Admission prediction can be used in law enforcement to predict the likelihood of individuals committing crimes. By analyzing factors such as criminal history, socio-economic status, and mental health, law enforcement agencies can predict which individuals are most likely to engage in criminal activity.

- **Insurance**: Admission prediction can be used by insurance companies to predict the likelihood of policyholders filing claims. By analyzing factors such as age, gender, and health status, insurers can predict which policyholders are most likely to require medical treatment or other insurance benefits.

6.CONCLUSION

- **University admission is the process by which students are selected to attend a college or university.The aim of this project is to help students in short listing universities with their profiles.With this project, students can make more informed decisions about which universities to apply to, and universities can make more efficient use of their resources by focusing on the most promising applicants.We have developed machine learning model using python programming language And the reports are shown above.**

7.FUTURE SCOPE

- The proposed changes will reflect in the way we evaluate public,private, and higher education institution.

- Help implement a student–focused framework that will bridge the gap between regional and sub locate institutions.

- These tools will be used to serve general human interest to promote diversity in the social and cultural domains

8.APPENDIX

SOURCE CODE:

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image. (optional) Here we have used visualisation style as fivethirtyeight.
Let us import necessary libraries to get started
Import numpy as np
Import pandas as pd
Import matplotlib.pyplot as plt
Import seaborn as sns
%matplotlib inline

Activity 1.2: Read the Dataset

#read_csv is a pandas function to read csv files
Data = pd.read_csv('Admission_Predict.csv')

Handling missing values

Data.info()
<class 'pandas.core.frame.Dataframe'>
RangeIndex: 400 entries, 0 to 399 Page

Data colimns (total 8 columns):
column Non-Null count Dtype
0 GRE Score 400 non-null int64
1 ROEFL Score 400 non-null int64
2 University Rating 400 non-null int64
3 SOP 400 non-null float64
4 LOR 400non-null float64
5 CGPA 400non-null float64
6 Research 400non-null int 64
7 Chance of Admit 400non-null float64

Dtypes:float64(4),int64(4)

Exploratory Data Analysis

Admission.describe()

Serial no. GRES 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 400.0000000
Mean 200.500000 316.807500 107.410000 3.087500 3.400000 3.452500 8.598925 0.547500 0.724350
Std 115.614301 11.473646 6.09514 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 0.6400000
50% 200.500000 317.000000 107.000000 3.0000000 3.500000 3.5000000 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 5.0000000 5.000000 5.000000 9.9200000 1.000000 0.970000

Activity 2.2: Bivariate analysis

```
plt.figure(figsize=(10, 10))
sns.heatmap(admission.corr(), annot=True, linewidths=0.05, fmt=
'.2f',cmap="magma")
plt.show()
```

Pair Plot: Plot pairwise relationships in a dataset.

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

Pair plot

Scatter Plot:

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

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
X_train[X_train.columns] = scaler.fit_transform(X_train[X_train.columns])
X_test[X_test.columns] = scaler.transform(X_test[X_test.columns])
X_train.head()
```

SCALING THE DATA:

```
From sklearn.preprocessing import MinMaxScaler
Sc=MinMaxScaler
```

```
x=sc.fit_transform(x)
x


X=admission.iloc[:,0:7].values
X



y=admission.iloc[:,7:].values
y


#random_state acts as the seed for the random number generator during the split
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=101)
X_train.shape


y_train=(y_train>0.5)
y_train
```

Model Building

Activity 1.1: logistic Regression Model

```
From sklearn.linear_model.logistic import LogisticRegression

Cls =LogisticRegression(random_state =0)
xg.score(X_test,y_test)


y_predict=rgr.predict(X_test)
y_predict
```