



INTELLIGENT ADMISSIONS: THE FUTURE OF UNIVERSITY DECISION MAKING WITH MACHINE LEARNING

Project Based Experiential Learning Program

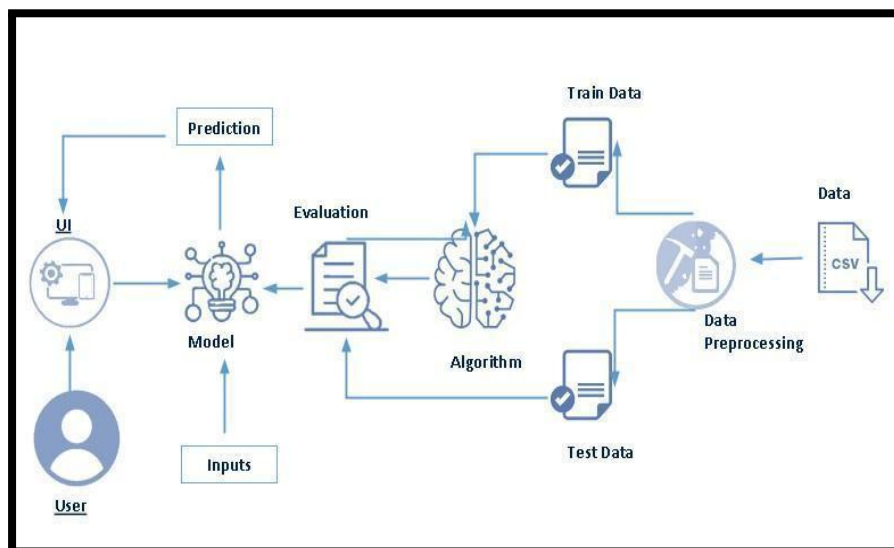
Intelligent Admissions: The Future of University Decision Making with Machine Learning

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

Students are often worried about their chances of 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 being admitted to the university of their choice.

The aim of this project is to help students in short listing universities with their profiles. Machine learning algorithms 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 efficient use of their resources by focusing on the most promising applicants. The predicted output gives them a fair idea about their admission chances in a particular university. This analysis should also help students who are currently preparing or will be preparing to get a better idea.

Technical Architecture:



Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Define Problem / Problem Understanding
 - Specify the business problem
 - Business requirements
 - Literature Survey
 - Social or Business Impact.
- Data Collection & Preparation
 - Collect the dataset
 - Data Preparation
- Exploratory Data Analysis
 - Descriptive statistical
 - Visual Analysis
- Model Building
 - Training the model in multiple algorithms
 - Testing the model
- Performance Testing & Hyperparameter Tuning
 - Testing model with multiple evaluation metrics
 - Comparing model accuracy before & after applying hyperparameter tuning
- Model Deployment
 - Save the best model
 - Integrate with Web Framework
- Project Demonstration & Documentation
 - Record explanation Video for project end to end solution
 - Project Documentation-Step by step project development procedure

Project Structure:

Create the Project folder which contains files as shown below

Name	Date Modified
Dataset	11-11-2022 16:27
└─ Admission_Predict.csv	11-11-2022 16:27
Flask	25-01-2023 12:06
├─ static	11-11-2022 16:27
├─ templates	11-11-2022 16:27
├─ app.py	25-01-2023 11:22
├─ model.h5	25-01-2023 11:38
├─ University Admission Prediction.ipynb	11-11-2022 16:27
├─ university.pkl	11-11-2022 16:27
IBM	11-11-2022 16:27
Training	25-01-2023 10:12
├─ .ipynb_checkpoints	12-11-2022 16:57
├─ model.h5	25-01-2023 10:11
├─ University Admission Prediction.ipynb	25-01-2023 09:58
├─ university.pkl	25-01-2023 09:11

- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.

- model.h5 is our saved model. Further we will use this model for flask integration.
- Training folder contains a model training file.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

Refer Project Description

Activity 2: Business requirements

The business requirements for a machine learning model to predict chances of student admission in the university. A project aims to predict the chances of a student getting admitted to a particular university based on certain factors. The business value of this project is that it will help students make more informed decisions about which universities to apply to, and help university counselors to better advise students on the universities they are most likely to be admitted to the university.

Activity 3: Literature Survey (Student Will Write)

The University Chances of Admission project is a well-researched topic in the field of education and machine learning. Many studies have been conducted to predict university admission using different machine learning techniques. One study by (Hsu and Chen, 2019) used decision tree, random forest, and logistic regression algorithms to predict the chance of university admission based on students' GPA, test scores, and personal information. The study found that the random forest algorithm performed the best with an accuracy of 85.5%. Another study by (Al-Shammari et al., 2018) used the k-nearest neighbor (KNN) algorithm to predict the chance of university admission based on students' GPA, test scores, and family income. The study found that the KNN algorithm performed well with an accuracy of 81.2%. A study by (Najafabadi et al., 2015) used a neural network to predict the chance of university admission based on students' GPA, test scores, and personal information. The study found that the neural network performed well with an accuracy of 94.3%. Overall, these studies suggest that various machine learning algorithms can be used to predict the chance of university admission with high accuracy.

Activity 4: Social or Business Impact.

Social Impact:- The ability to accurately predict the chances of university admission can help students make more informed decisions about which universities to apply to, increasing their chances of being admitted and ultimately gaining access to higher education.

Business Model/Impact:- 1. using machine learning models to predict university admission, the service can help universities more efficiently process and evaluate applications, potentially increasing the number of successful admissions.

2. An increase in the number of successful admissions can lead to an increase in revenue for universities, as well as for the company providing the prediction service.

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

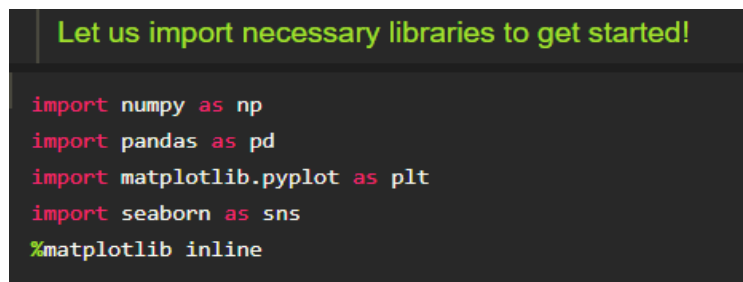
Link: <https://www.kaggle.com/rishal005/admission-predict>

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

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

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

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

Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling Imbalance Data

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 2.1: Handling missing values

- Let's find the shape of our dataset first. To find the shape of our data, the `df.shape()` method is used. To find the data type, `df.info()` function is used.

```
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
```

- For checking the null values, `df.isnull()` function is used. To sum those null values we use `.sum()` function. From the below image we found that there are no null values present in our dataset.

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

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

- Let us rename the column, in python have a inbuilt function rename(). We can easily rename the column names.

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

Milestone 3: Exploratory Data Analysis

Activity 1: 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.

```
data.describe()
```

	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

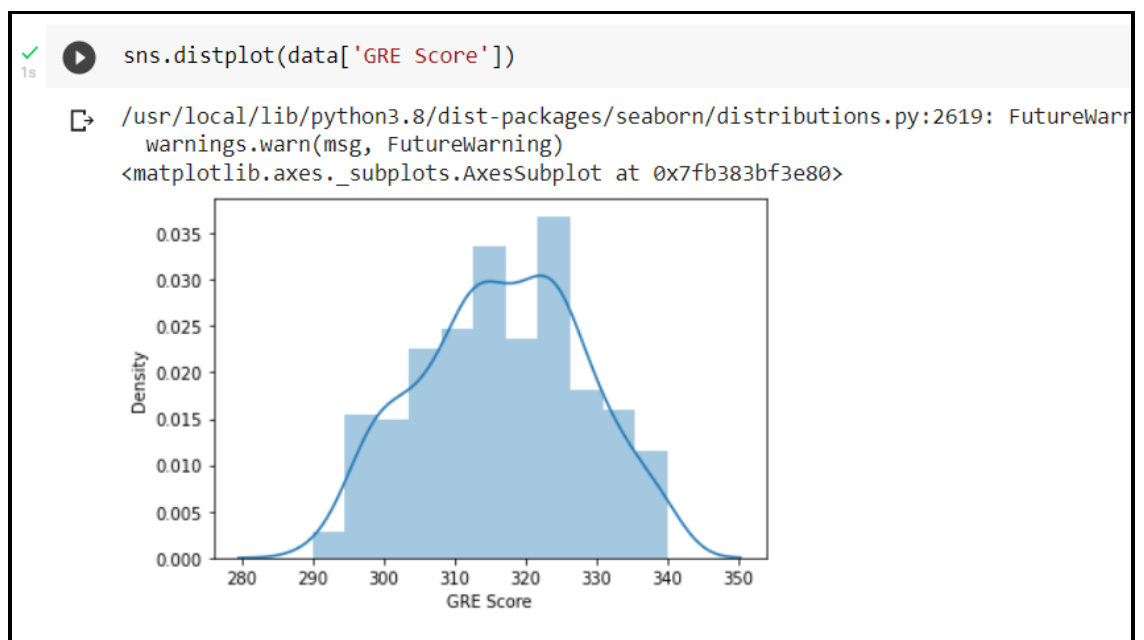
Activity 2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

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.



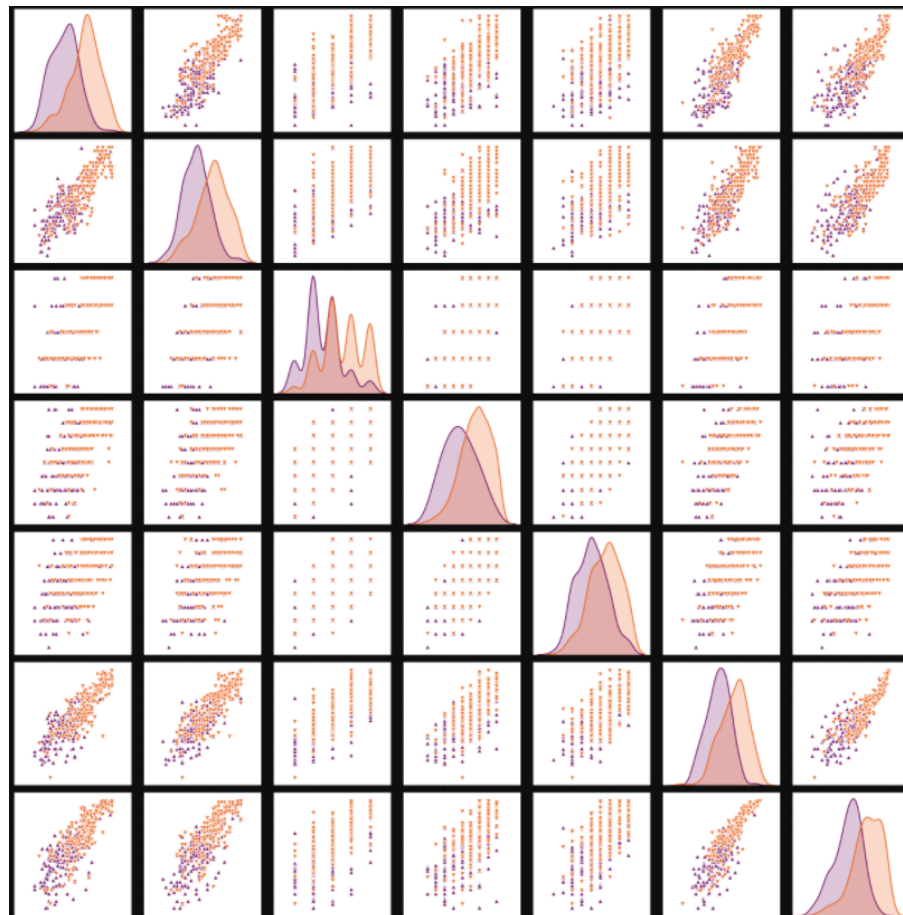
Activity 2.2: Bivariate analysis



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

Pair Plot: Plot pairwise relationships in a dataset

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



Pair plot usually gives pair wise relationships of the columns in the dataset

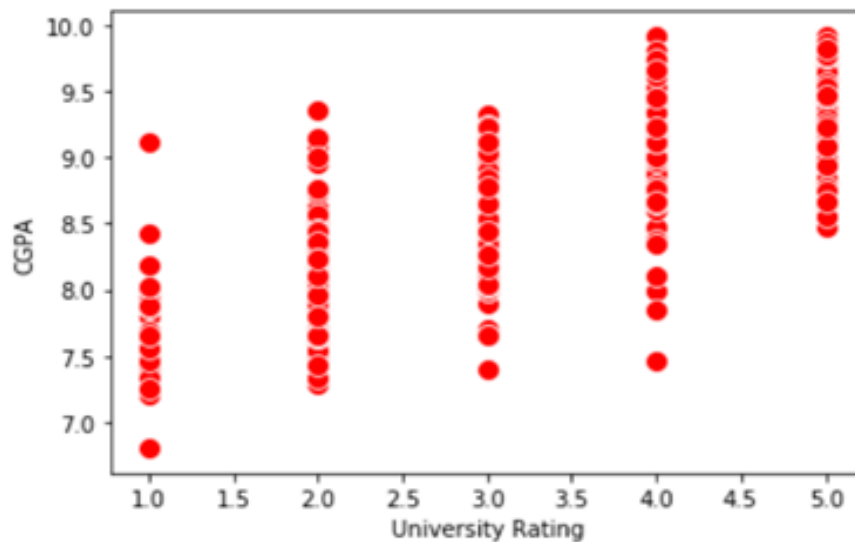
1. GRE score TOEFL score and CGPA all are linearly related to each other
2. Students in research score high in TOEFL and GRE compared to non research candidates

Scatter Plot: Matplot has a built-in function to create scatterplots 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)
```

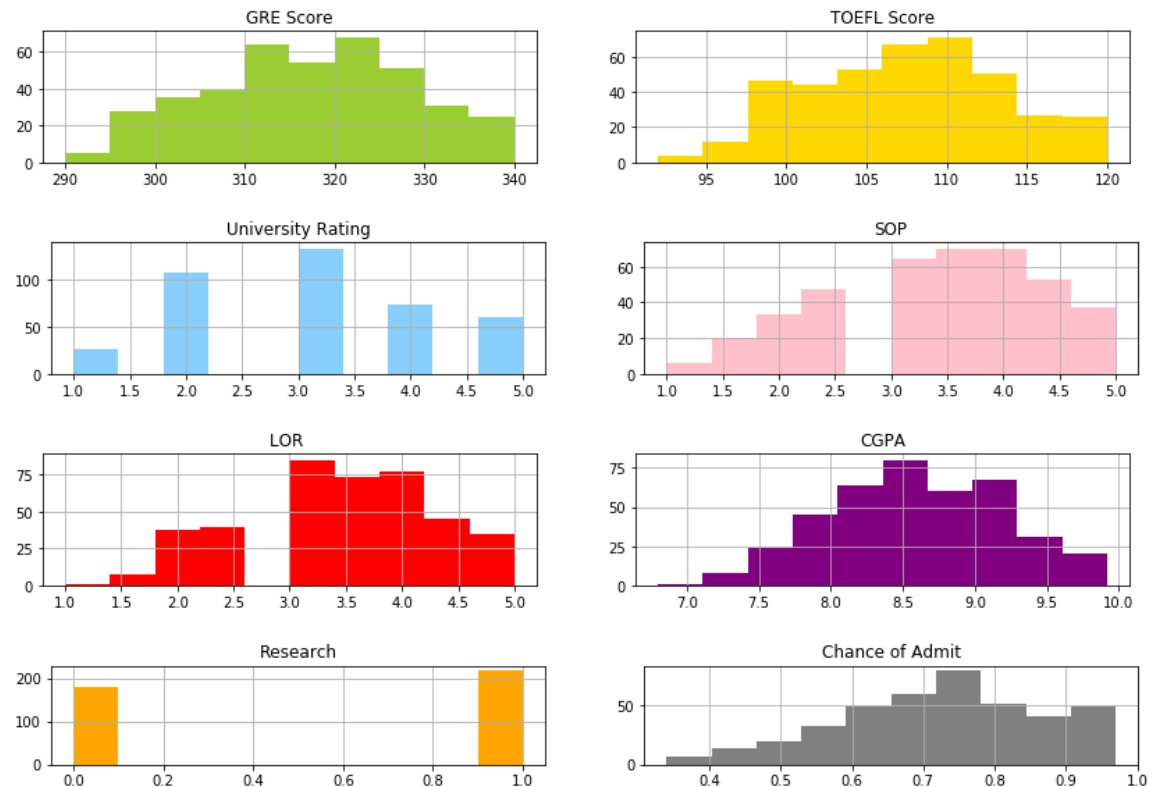
<matplotlib.axes._subplots.AxesSubplot at 0x2b6e49feec8>



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

```
category = ['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()
```



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
```

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.

Splitting data into x and y

Now let's split the Dataset into x and y

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

```
y=data.iloc[:,7:].values  
y
```

Changes: first split the dataset into x and y and then split the data set

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.

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

Let us convert it into classification problem

chance of admit>0.5 as true chance of admit<0.5 as false

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

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

Milestone 4: 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)

lr=cls.fit(x_train, y_train)

C:\Users\Tulasi\anaconda3\lib\site-packages\sklearn\utils\validation.py:760: DataConversionWarn
array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_id(y, warn=True)

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

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.

ANN Model

```
In [29]: #Libraries to train Neural network
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import Adam
```

```
In [30]: # Initialize the model
model=keras.Sequential()

# Add input Layer
model.add(Dense(7,activation = 'relu',input_dim=7))

# Add hidden layers
model.add(Dense(7,activation='relu'))

# Add output layer
model.add(Dense(1,activation='linear'))

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
--------------	--------------	---------

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 7)	56
dense_1 (Dense)	(None, 7)	56
dense_2 (Dense)	(None, 1)	8

Total params: 120
Trainable params: 120
Non-trainable params: 0

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

```
Epoch 1/100
16/16 [=====] - 0s 2ms/step - loss: 1.7298 - accuracy: 0.0781
Epoch 2/100
16/16 [=====] - 0s 1ms/step - loss: 1.3143 - accuracy: 0.0844
Epoch 3/100
16/16 [=====] - 0s 1ms/step - loss: 1.0439 - accuracy: 0.1344
Epoch 4/100
16/16 [=====] - 0s 1ms/step - loss: 0.8401 - accuracy: 0.3219
Epoch 5/100
16/16 [=====] - 0s 1ms/step - loss: 0.6683 - accuracy: 0.5656
Epoch 6/100
16/16 [=====] - 0s 1ms/step - loss: 0.5238 - accuracy: 0.7531
Epoch 7/100
16/16 [=====] - 0s 1ms/step - loss: 0.3918 - accuracy: 0.8844
Epoch 8/100
16/16 [=====] - 0s 1ms/step - loss: 0.2865 - accuracy: 0.9250
Epoch 9/100
16/16 [=====] - 0s 1ms/step - loss: 0.2254 - accuracy: 0.9312
Epoch 10/100
16/16 [=====] - 0s 1ms/step - loss: 0.1820 - accuracy: 0.9321
```

Activity 2: Testing the model

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

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

```
[47]: model.fit(x_train, y_train, batch_size = 20, epochs = 100)
```

```
[46]: from sklearn.metrics import accuracy_score
```

```
# Make predictions on the training data  
train_predictions = model.predict(x_train)  
  
print(train_predictions)
```

```
[36]: # Get the training accuracy  
train_acc = model.evaluate(x_train, y_train, verbose=0)[1]  
  
print(train_acc)
```

```
0.9281250238418579
```

```
[37]: # Get the test accuracy  
test_acc = model.evaluate(x_test, y_test, verbose=0)[1]  
  
print(test_acc)
```

```
0.875
```

```
[45]: print(classification_report(y_test, pred))
```

```
[ ]:   
  
pred=model.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,  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])
```

Milestone 5: Performance Testing & Hyperparameter Tuning

Activity 1: Testing model with multiple evaluation metrics

Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding

of the model's strengths and weaknesses. We are using evaluation metrics for classification tasks including accuracy, precision, recall, support and F1-score.

Activity 1.1: Compare the model

For comparing the above four models, the compareModel function is defined.

Logistics Regression model

```
from sklearn.metrics import accuracy_score, recall_score, roc_auc_score, confusion_matrix
print("\nAccuracy score: %f" %(accuracy_score(y_test, y_pred) * 100))
print("Recall score : %f" %(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: 90.000000
Recall score : 99.074074
ROC score : 53.703704

[[ 1  11]
 [ 1 107]]
```

ANN Model : Training Accuracy

```
from sklearn.metrics import accuracy_score, recall_score, roc_auc_score, confusion_matrix
print(classification_report(y_train, pred))
```

	precision	recall	f1-score	support
False	1.00	0.16	0.28	25
True	0.93	1.00	0.97	295
accuracy			0.93	320
macro avg	0.97	0.58	0.62	320
weighted avg	0.94	0.93	0.91	320

```

from sklearn.metrics import accuracy_score, recall_score, roc_auc_score
print(classification_report(y_test, pred))

```

	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

the results of models are displayed as output. From the both models ANN is performing well. From the below image, We can see the accuracy of the model. ANN is giving the accuracy of 93.% with training data , 88% accuracy for the testing data.

Milestone 6: Model Deployment

Activity 1: Save the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

```

[39] # Save the model in HDF5 format
model.save('model.h5')

```

Activity 2: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server side script

- Run the web application

Activity 2.1: Building Html Pages:

For this project create two HTML files namely

- home.html
- predict.html

and save them in the templates folder.

Activity 2.2: Build Python code:

Import the libraries

```
1 import numpy as np
2 from flask import Flask, request, jsonify, render_template
3 import pickle
4 app = Flask(__name__)
5 # Import necessary libraries
6 from tensorflow.keras.models import load_model
7
8 #model = pickle.load(open('university.pkl', 'rb'))
9
```

Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument.

```
#load model trained model
# Load your trained model
model = load_model('model.h5')
```

Render HTML page:

```
@app.route('/')
def home():
    return render_template('Demo2.html')
```

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with the home.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

```
@app.route('/')
def home():
    return render_template('Demo2.html')

@app.route('/y_predict', methods=['POST'])
def y_predict():
    '''
    For rendering results on HTML GUI
    '''
    #min max scaling
    min1=[290.0, 92.0, 1.0, 1.0, 1.0, 6.8, 0.0]
    max1=[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]-min1[i])/(max1[i]-min1[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 gettin
    else:
        return render_template('chance.html', prediction_text='You have a chance of getting admis
if __name__ == "__main__":
    app.run(debug=False)
```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

```
36         return render_template
37     else:
38         return render_template
39 if __name__ == "__main__":
40     app.run(debug=False)
41
```

Activity 2.3: Run the web application

- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type "python app.py" command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

```
base) D:\TheSmartBridge\Projects\2. DrugClassification\Drug c
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a p
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Now, Go the web browser and write the localhost url (<http://127.0.0.1:5000>) to get the below result

UNIVERSITY ADMISSION PREDICTION SYSTEM

Enter your details and get probability of your admission

Enter GRE Score

Enter TOEFL Score

Select University no

☐ 1

☒ 2

☐ 3

☐ 4

☐ 5

Enter SOP

Enter LOR

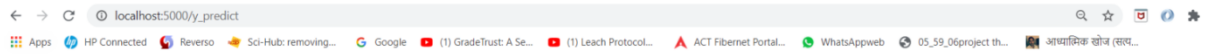
Enter CGPA

Research

☐ Research

☒ NO Research

Now, when you click on click me to predict the button from the banner you will get redirected to the prediction page.



Predicting Chance of Admission

A Machine Learning Web App using Flask.

Prediction : You have a chance



Input 1- Now, the user will give inputs to get the predicted result after clicking onto the predict button

Milestone 7: Project Demonstration & Documentation

Below mentioned deliverables to be submitted along with other deliverables

Activity 1:- Record explanation Video for project end to end solution

Activity 2:- Project Documentation-Step by step project development procedure

Create document as per the template provided