Smart Internz

Project Report Template

Smart Internz

Project Report Template

Introduction
1.1 overview

A brief description about your project

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.

Purpose

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

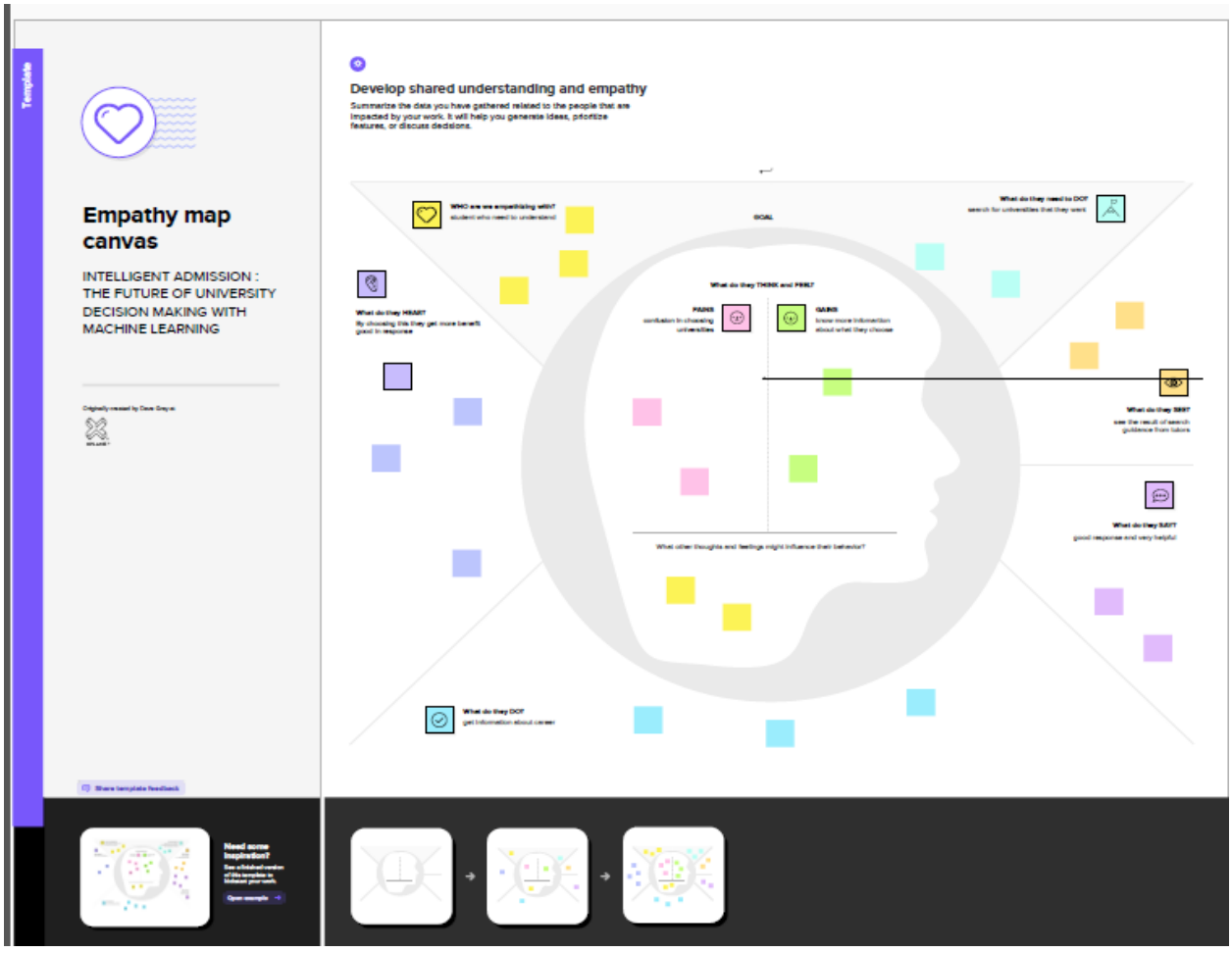
Applications, potentially increasing the number of successful admissions.

2.An increase in the number of successful admissions can lead to an increase in revenue

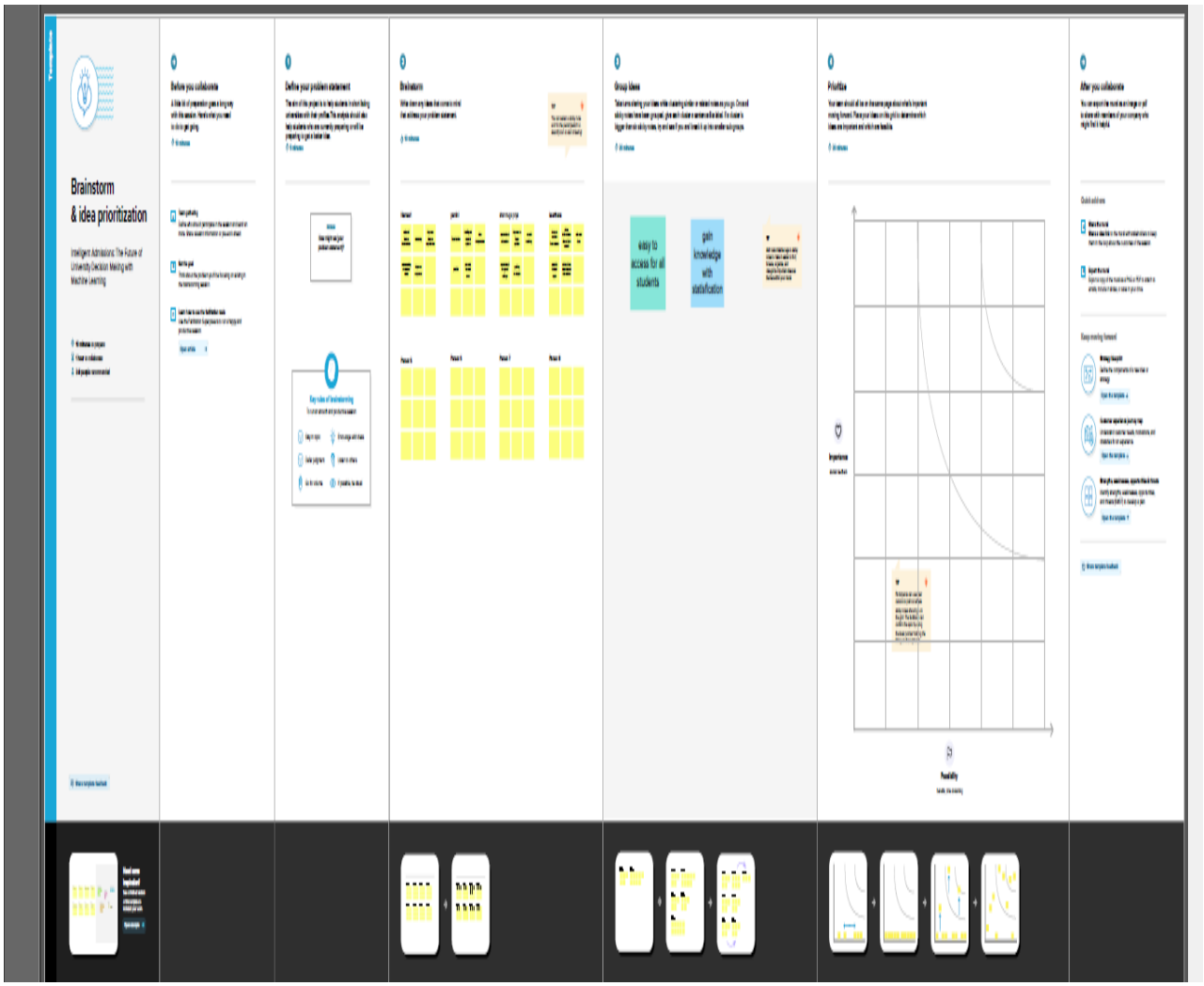
For universities, as well as for the company providing the prediction service.

Problem definition & design thinking

2.1 Empathy map

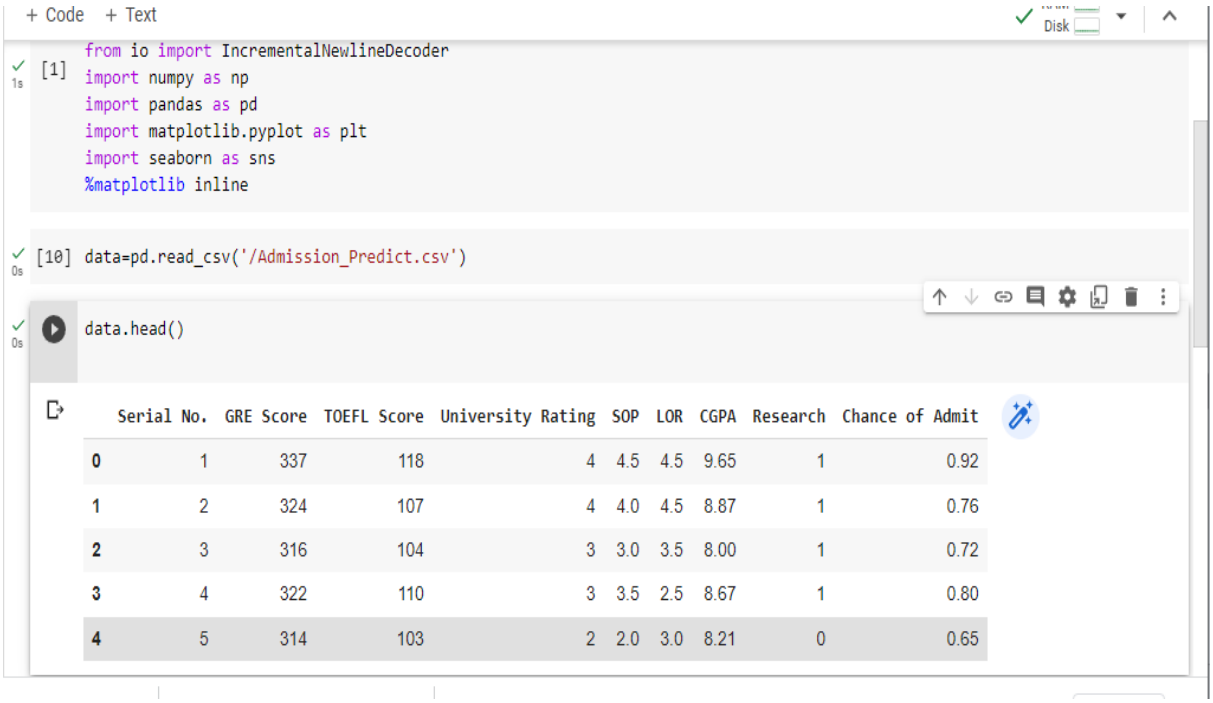


2.2 Ideation & Brainstorming



Result

Read the Dataset



+ Code + Text

```
[1] from io import IncrementalNewlineDecoder
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    %matplotlib inline
```

```
[10] data=pd.read_csv('/Admission_Predict.csv')
```

```
data.head()
```

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

Data preparation

```
data.info()
```

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

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

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

Exploratory data analysis

```
data.describe()
```

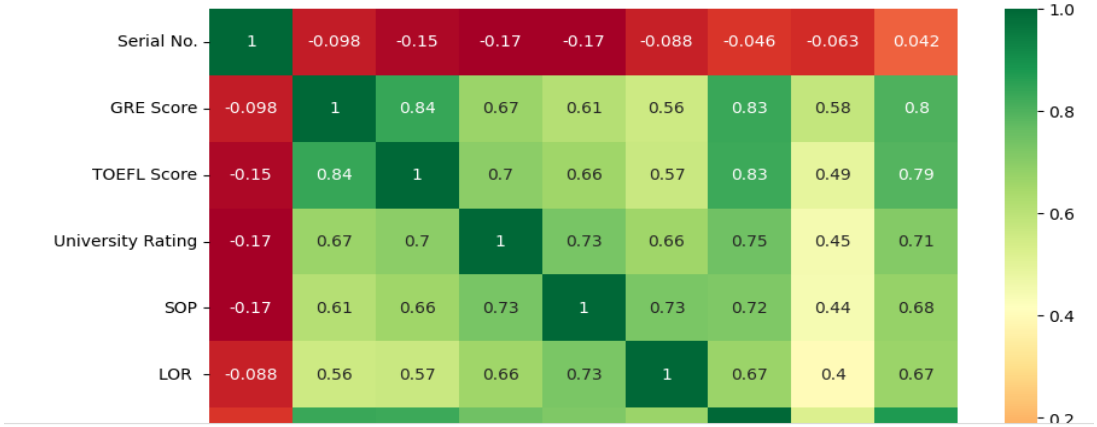
	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of admit
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000
mean	200.500000	316.807500	107.410000	3.087500	3.400000	3.452500	8.598925	0.547500	0.724350
std	115.614301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

```
data.corr()
```

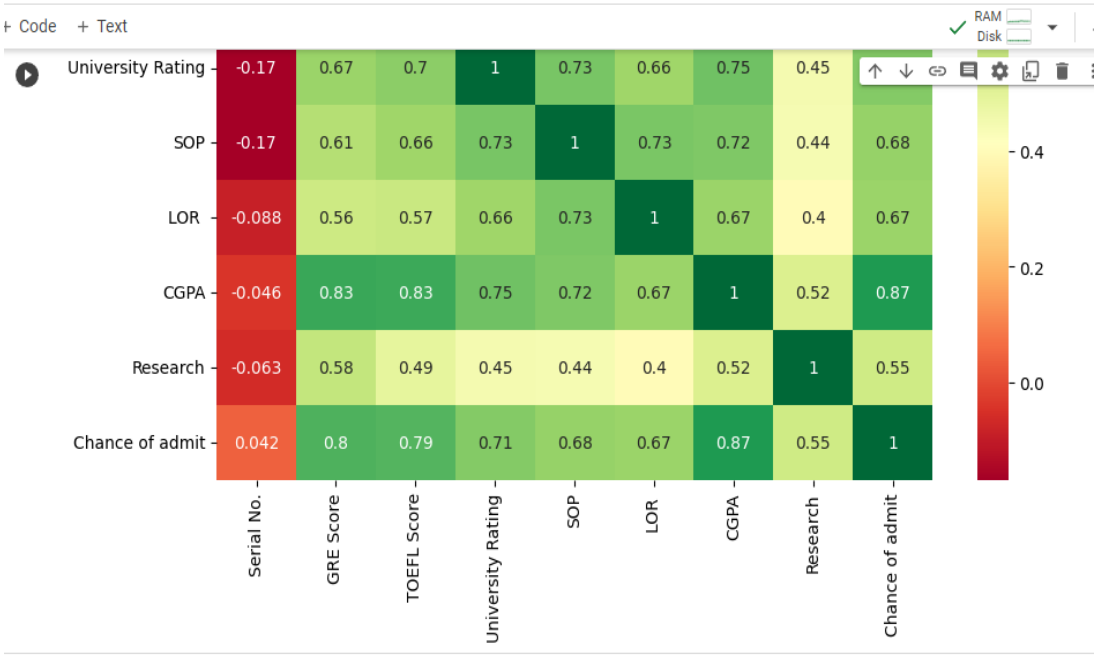
	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of admit
Serial No.	1.000000	-0.097526	-0.147932	-0.169948	-0.166932	-0.088221	-0.045608	-0.063138	0.042336
GRE Score	-0.097526	1.000000	0.835977	0.668976	0.612831	0.557555	0.833060	0.580391	0.802610
TOEFL Score	-0.147932	0.835977	1.000000	0.695590	0.657981	0.567721	0.828417	0.489858	0.791594
University Rating	-0.169948	0.668976	0.695590	1.000000	0.734523	0.660123	0.746479	0.447783	0.711250
SOP	-0.166932	0.612831	0.657981	0.734523	1.000000	0.729593	0.718144	0.444029	0.675732
LOR	-0.088221	0.557555	0.567721	0.660123	0.729593	1.000000	0.670211	0.396859	0.669889
CGPA	-0.045608	0.833060	0.828417	0.746479	0.718144	0.670211	1.000000	0.521654	0.873289
Research	-0.063138	0.580391	0.489858	0.447783	0.444029	0.396859	0.521654	1.000000	0.553202
Chance of admit	0.042336	0.802610	0.791594	0.711250	0.675732	0.669889	0.873289	0.553202	1.000000

```
plt.figure(figsize=(10,7))
sns.heatmap(data.corr(),annot=True,cmap="RdYlGn")
```

```
<Axes: >
```



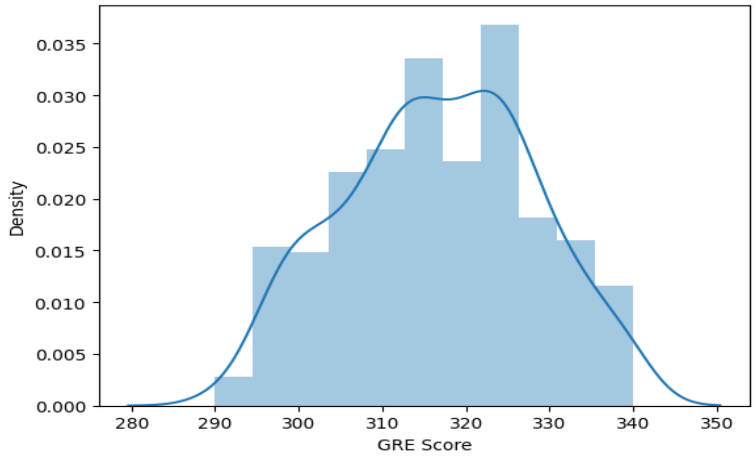
✓ 1s completed at 3:30 PM



Activity 2

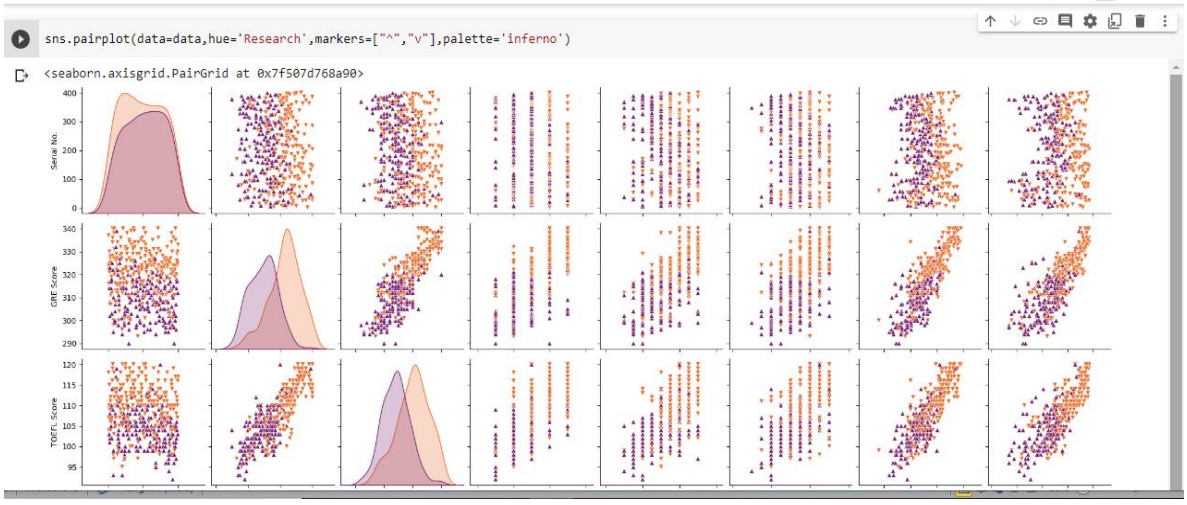
Univariate analysis

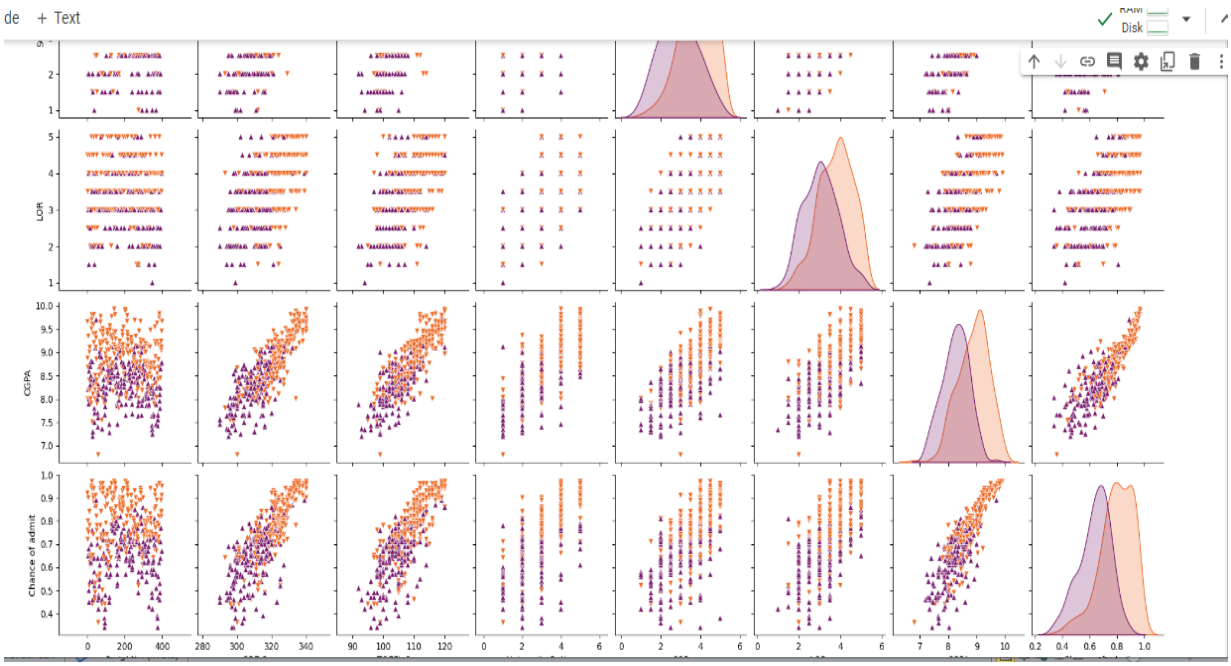
```
sns.distplot(data['GRE Score'])
<Axes: xlabel='GRE Score', ylabel='Density'>
```



Bivariate analysis

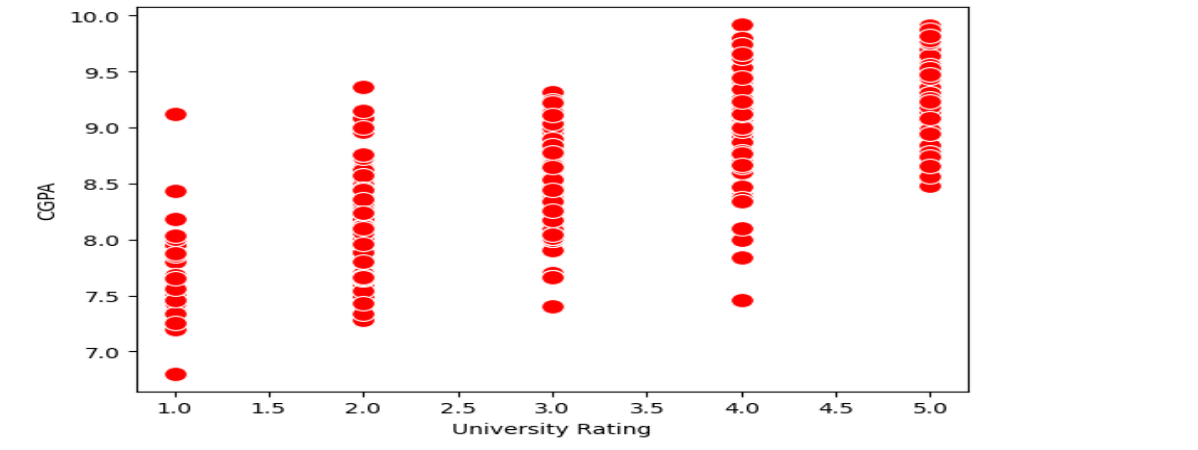
```
sns.pairplot(data=data,hue='Research',markers=["^","v"],palette='inferno')
<seaborn.axisgrid.PairGrid at 0x7f507d768a90>
```





Scatter plot

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

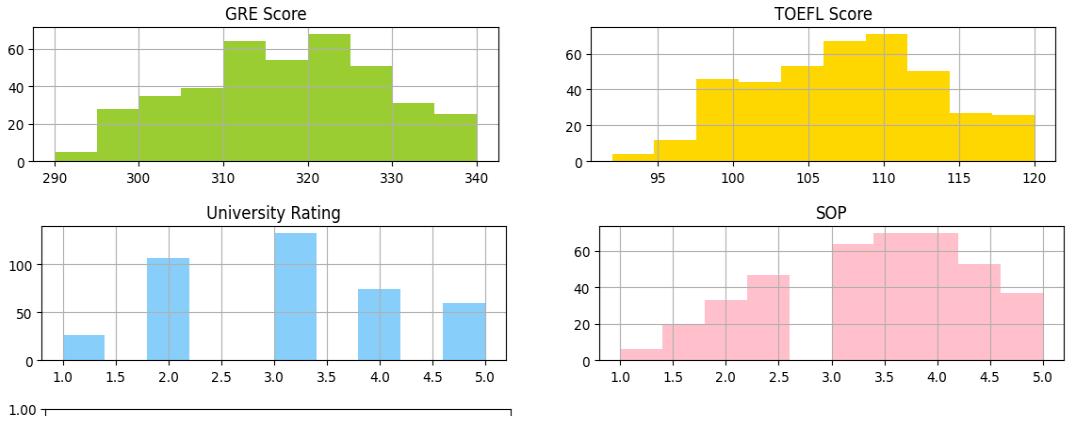


ide + Text

RAM Disk

```
KeyError: 'LOR'
```

SEARCH STACK OVERFLOW




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

```
array([[  1.  , 337.  , 118.  , ...,   4.5 ,   9.65,   1.  ],
       [  2.  , 324.  , 107.  , ...,   4.5 ,   8.87,   1.  ],
       [  3.  , 316.  , 104.  , ...,   3.5 ,   8.  ,   1.  ],
       ...,
       [398.  , 330.  , 116.  , ...,   4.5 ,   9.45,   1.  ],
       [399.  , 312.  , 103.  , ...,   4.  ,   8.78,   0.  ],
       [400.  , 333.  , 117.  , ...,   4.  ,   9.66,   1.  ]])
```

```
[ ]
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler()
x=sc.fit_transform(x)
x
x=data.iloc[:,0:7].values
x
```

0s completed at 4:15 PM

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

Interrupt execution (Ctrl+M I)
cell executed since last change

started at 4:04 PM (0 minutes ago)

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

```
[ ]
```

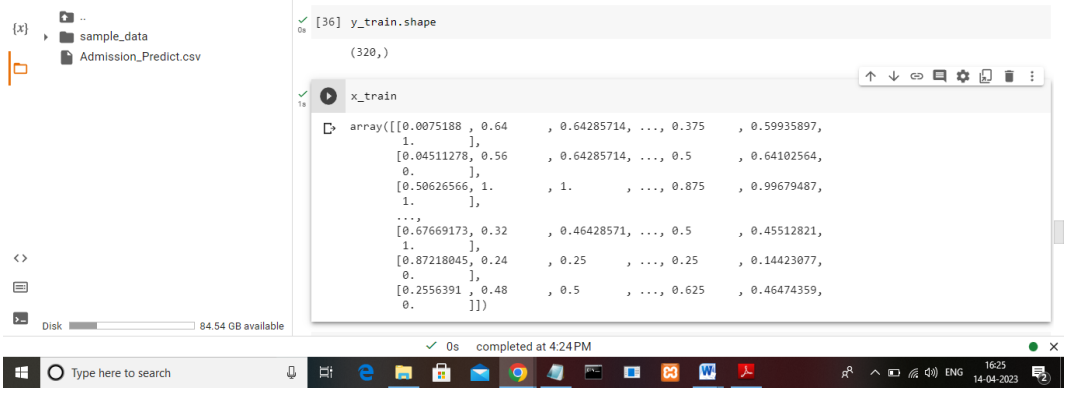
```
y=data['Chance of admit'].values
y
```

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

✓ 0s completed at 4:17 PM

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

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



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

```
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,
       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, False,  True,  True,  True, False,  True,
       False,  True,  True,  True,  True,  True, False,  True,  True,
```

0s completed at 4:26 PM

[38]

```
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True])
```

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

```
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])
```

```
import sklearn.linear_model.logistic
import LogisticRegression
cls=LogisticRegression(random_state=0)
lr=cls.fit(x_train,y_train)
y_pred=lr.predict(x_test)
y_pred
```

```
[56] #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))
```

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

```
0.928125
0.875
***logistic Regression***
Confusion_Matrix
[[ 0 10]
 [ 0 70]]
```

```
0.928125
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

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Pr
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Pr
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Pr
  _warn_prf(average, modifier, msg_start, len(result))
```

```
decisionTree(x_train,x_test,y_train,y_test)
```

```
1.0
0.8875
***Decisioc Tree***
confusion_matrix
[[ 7  3]
 [ 6 64]]
Classification Report
              precision    recall  f1-score   support

       False       0.54      0.70      0.61        10
        True       0.96      0.91      0.93        70

    accuracy                           0.89        80
   macro avg       0.75      0.81      0.77        80
weighted avg       0.90      0.89      0.89        80
```

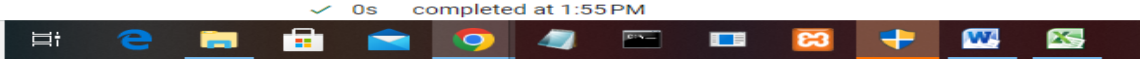
```
[ ]  def RandomForest(x_train,x_test,y_train,y_rest):
       rf=RandomForestClassifier(criterion="entropy",n_estimators=10,random_state
       rf.fit(x_train,y_train)
       y_rf_tr=rf.predict(x_train)
```

```
[42]   print("confusion matrix")
       print(confusion_matrix(y_test,ypred_rf))
       print("classification report")
       print(classification_report(y_test,ypred_rf))
```

```
RandomForest(x_train,x_test,y_train,y_test)
```

```
0.996875
0.925
***Random Forest***
confusion matrix
[[ 6  4]
 [ 2 68]]
classification report
              precision    recall  f1-score   support

       False       0.75      0.60      0.67        10
        True       0.94      0.97      0.96        70

    accuracy                           0.93        80
   macro avg       0.85      0.79      0.81        80
weighted avg       0.92      0.93      0.92        80
```

0s completed at 1:55 PM



Model building

Activity 1

```
[18] import tensorflow as tf
     from tensorflow import keras
     from tensorflow.keras.layers import Dense,Activation,Dropout
     from tensorflow.keras.optimizers import Adam
```

```
model=keras.Sequential()
model.add(Dense(7,activation='relu',input_dim=7))
model.add(Dense(1,activation='linear'))
model.summary()
```

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

 dense_1 (Dense)             (None, 1)                 8

=================================================================
Total params: 64
Trainable params: 64
Non-trainable params: 0
_________________________________________________________________
```

✓ 0s completed at 11:28 PM

```
[46] #importing the keras libraries and packages
     import keras
     from keras.models import Sequential
     from keras.layers import Dense
```

```
[48] #initialising the ANN
     classifier=Sequential()
```

```
[49] classifier.add(Dense(units=7,activation='relu',input_dim=7))
```

```
[50] classifier.add(Dense(units=7,activation='relu'))
```

```
[51] classifier.add(Dense(units=1,activation='linear'))
```

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

Advantage and disadvantage

Personality Builder:

Decision-making skills for students can help the student to provide an insight on their personality as it throws light on their strengths and weaknesses. Thus it can allow the students to focus on their weaknesses so that they can improve them.

Risk Analyses:

Most of the decisions involve some amount of risk.

Career Choices:

As a student, you can feel the importance of developing decision- making skills in every phase of your life.

Confidence Booster:

Good decision-making skills can boost confidence. When you make the right decision, it will give you more confidence to deal with the difficult situations in your life.

Game Changer:

There is no denying fact that a single decision can be a game-changer as it can determine your destiny. For example, you have just passed your board exam and now you have to choose the stream.

Application

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.

Conclusion

This summarised a detailed review on potential applications for the future of University decision making with machine learning .
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.

Future scope

By predicting the admission process students get more benefit and it is more useful for them.

Source code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
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 train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score,classification_repor
t,confusion_matrix,f1_score
import pickle


data=pd.read_csv('/content/Admission_Predict.csv')
data.head()
data.info()
```

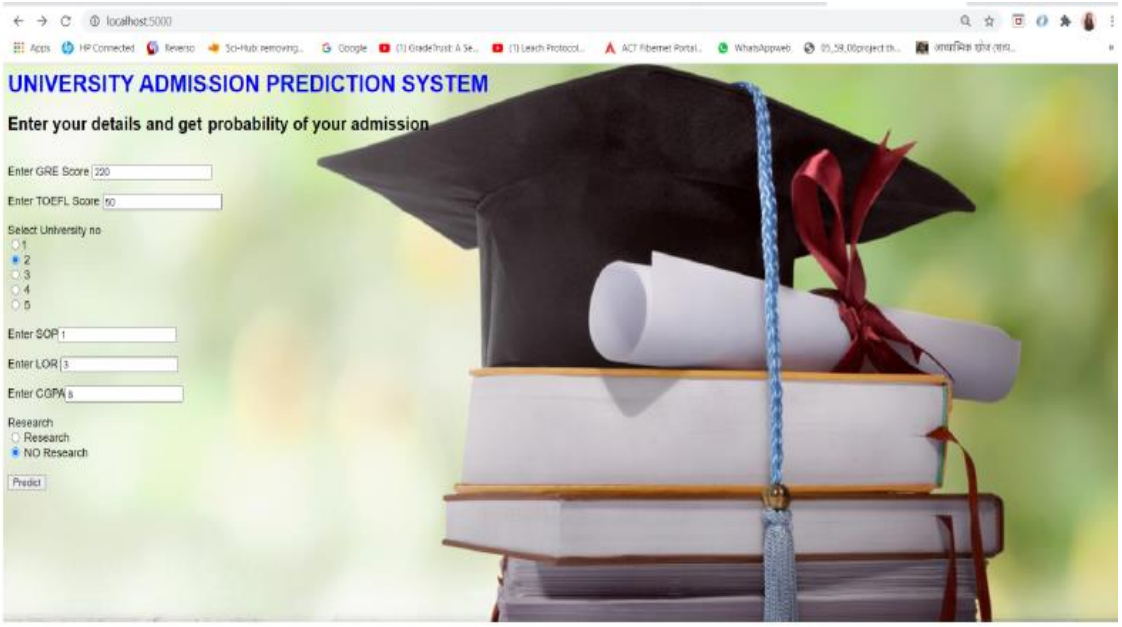
```
data.isnull().any()
data=data.rename(columns ={'Chance of Admit ':'Chance of admit'
})
data.describe()
data.corr()
plt.figure(figsize=(10,7))
sns.heatmap(data.corr(),annot=True,cmap="RdYlGn")
sns.distplot(data['GRE Score'])
sns.pairplot(data=data,hue='Research',markers=["^","v"],palette
='inferno')
sns.scatterplot(x='University Rating',y='CGPA',data=data,color=
'Red',s=100)
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()
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 r
ating']<=500].university rating.mean()))

from sklearn.preprocessing import MinMaxScaler
sc=MinMaxScaler()

x=sc.fit_transform(x)
x



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

```
y=data['Chance of admit'].values
y
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 act as the seed for random number generator durin
g split
y_train.shape
x_train
y_train=(y_train>0.5)
y_train
y_test=(y_test>0.5)
y_test
#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))

logreg(x_train,x_test,y_train,y_test)
#testing on test and 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)
decisionTree(x_train,x_test,y_train,y_test)
def decisionTree(x_train,x_test,y_train,y_rest):
  dtc=DecisionTreeClassifier(criterion="entropy",random_state=0
)
  dtc.fit(x_train,y_train)
  print("Predicting on the test values")
```

```
  dtc_pred=dtc.predict(x_test)
  print("output is :",dtc_pred)
  print("predicting on random values")
  dtc_pred_own=dtc.predict(sc.transfrom([[337,118,4,4.5,4.5,9.6
5,1]]))
  print("output is :",dtc_pred_own)
def RandomForest(x_train,x_test,y_train,y_rest):
  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.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))

RandomForest(x_train,x_test,y_train,y_test)
#inputing the keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense

classifier=sequential()
classifier.add(Dense(units=7,activation='relu',input_dim=7))
classifier.add(Dense(units=7,activation='relu'))
classifier.add(Dense(units=1,activation='linear'))
classifier.compile(optimizer='adam',loss='binary_crossentrophy'
,metrics=['accuracy'])
model=classifier.fit(x_train,y_train,batch_size=10,validation_s
plit=0.33,epochs=20)
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(ann_pred))
print("classfication report")
print(classification_report(y_test,ann_pred))

#testing on test & random input values
```

```
print("predicting on test input")
ann_pred=classifier(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)
pickle.dump(lr,('university.pk1','wb'))
```

```
@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)
```


localhost:5000
UNIVERSITY ADMISSION PREDICTION SYSTEM
Enter your details and get probability of your admission
Enter GRE Score 220
Enter TOEFL Score 50
Select University no
1
2
3
4
5
Enter SOP 1
Enter LOR 3
Enter CGPA 8
Research
Research
NO Research
Predict