INDEX

1. INTRODUCTION

1.1 OVERVIEW

Intelligence admission, or intelligent enrollment, is a rapidly evolving field that is revolutionizing the university admission process. It involves using machine learning algorithms to analyze and evaluate large amounts of data about applicants, such as their academic performance, extracurricular activities, and other relevant factors, to predict which students are most likely to succeed in the university.

The benefits of intelligence admission are numerous. By using machine learning models to analyze and evaluate student data, universities can make more objective and data-driven admission decisions, which can reduce bias and improve diversity in student populations. Additionally, intelligence admission can help universities streamline the admission process, reduce administrative costs, and make admission decisions more quickly and efficiently.

Intelligence admission has the potential to transform the university admission process in numerous ways. For example, universities can use machine learning algorithms to identify high-potential applicants who may not have the traditional academic credentials but have other valuable qualities, such as leadership potential or unique experiences.

Furthermore, intelligence admission can help universities to identify and address retention issues before they become major problems. By analyzing data about student performance and engagement, universities can identify students who may be at risk of dropping out and provide them with targeted support to help them succeed.

While there are many benefits to intelligence admission, it is important to recognize that machine learning models are not perfect and may contain biases or inaccuracies. Therefore, it is important for universities to continuously evaluate and improve their machine learning models to ensure they are making fair and accurate admission decisions.

Overall, intelligence admission is an exciting and rapidly evolving field that has the potential to transform the university admission process, improve student outcomes, and increase diversity in student populations.

1.2 PURPOSE

The purpose of intelligence admission, or intelligent enrollment, is to use machine learning algorithms to assist in the university admission process by analyzing large amounts of data about applicants, such as their academic performance, extracurricular activities, and other relevant factors, to predict which students are most likely to succeed in the university.

The goal of intelligence admission is to make the admission process more objective and data-driven, which can reduce bias and improve diversity in student populations. Additionally, intelligence admission can help universities streamline the admission process, reduce administrative costs, and make admission decisions more quickly and efficiently.

By using machine learning models to analyze and evaluate student data, universities can identify high-potential applicants who may not have the traditional academic credentials but have other valuable qualities, such as leadership potential or unique experiences. This can help universities to create more diverse and inclusive student populations.

Furthermore, intelligence admission can help universities to identify and address retention issues before they become major problems. By analyzing data about student performance and engagement, universities can identify students who may be at risk of dropping out and provide them with targeted support to help them succeed.

2. PROBLEM DEFENITION & DESIGN THINKING

2.1 EMPATHY MAP

Team Id	NM2023TMID32898
Project Name	Intelligent Admissions – The Future Of University
Maximum Marks	5 Marks

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.

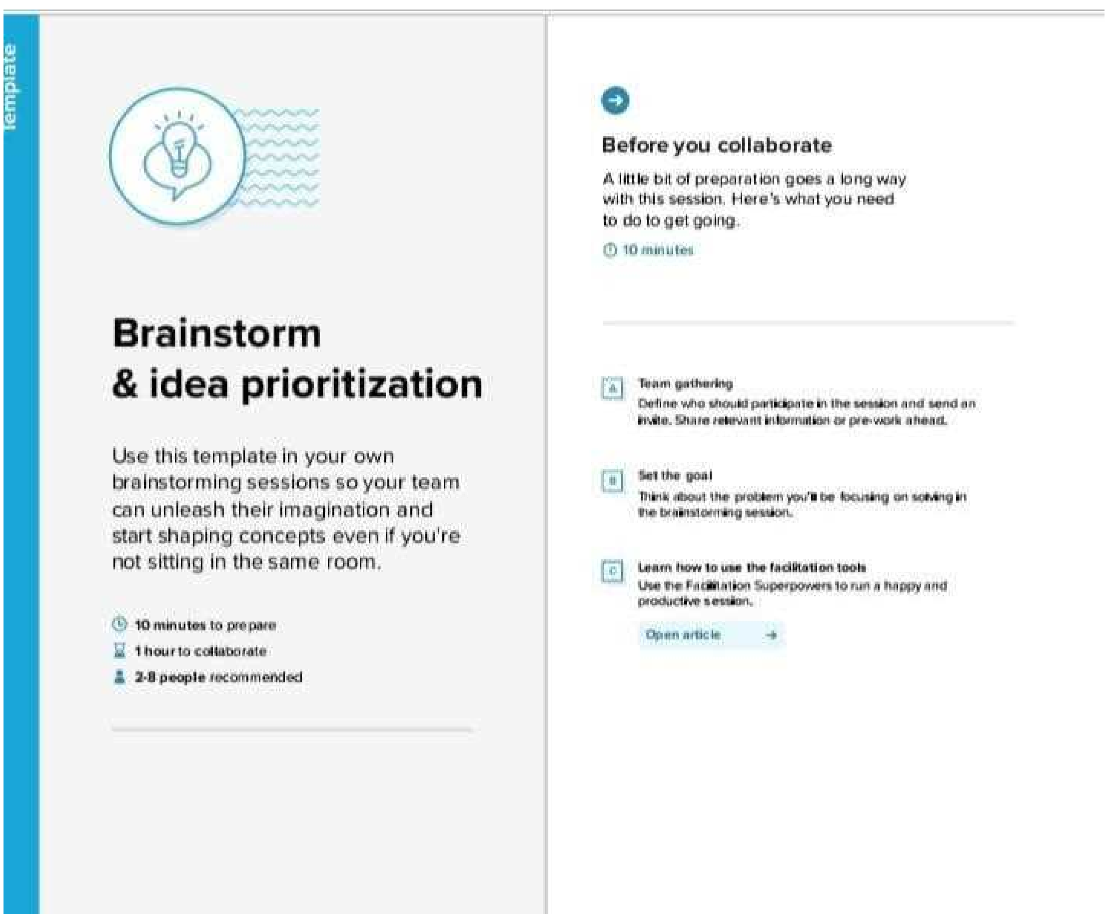
2.2 IDEATION AND BRAINSTORMING MAP

Team Id	NM2023TMID32898
Project Name	Intelligent Admissions – The Future Of University
Maximum Marks	5 Marks

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.



1

Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

5 minutes

PROBLEM

Why do you want to attend this university?

Key rules of brainstorming

To run an smooth and productive session

- Stay in topic.
- Encourage wild ideas.
- Defer judgment.
- Listen to others.
- Go for volume.
- If possible, be visual.

2

Brainstorm

Write down any ideas that come to mind
that address your problem statement.

10 minutes

TIP
You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

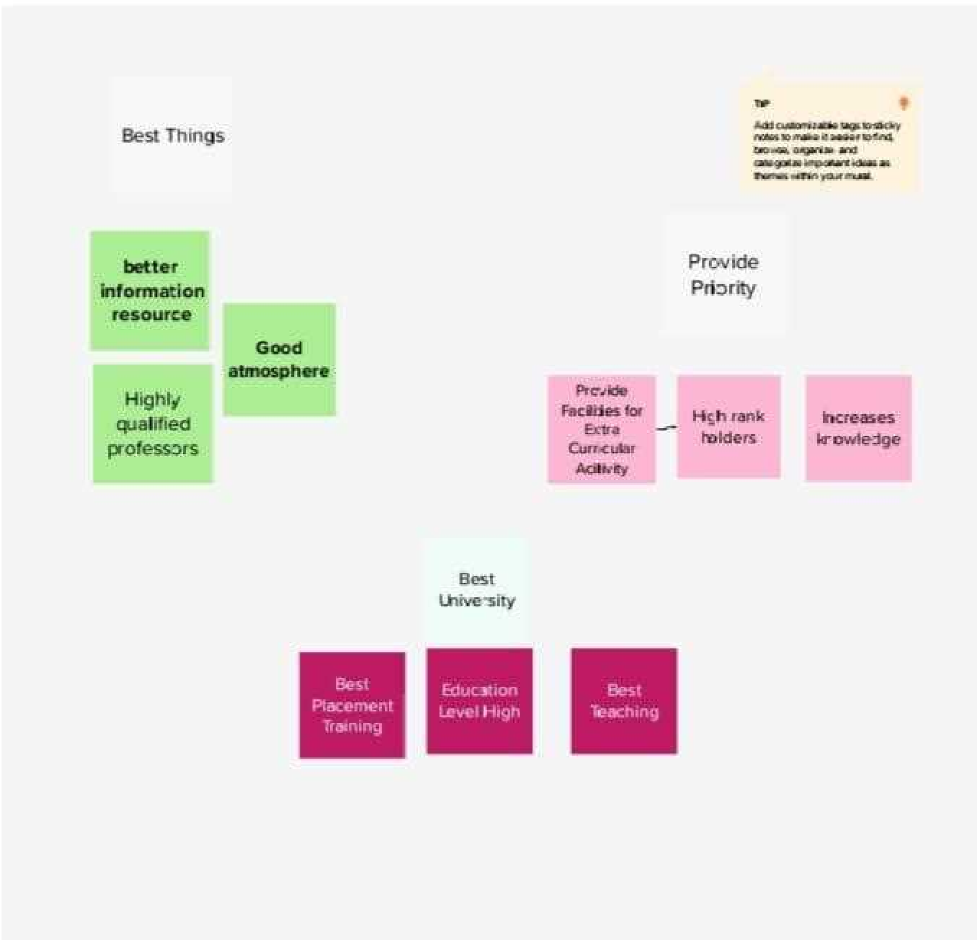
Jeslin Shobia	D.Mahalakshmi	S.Kaviya	M.Mahalakshmi
Best Placement Training	Best Courses Offered	Best Teaching	**better information resource**
Best University	Education Level High	Increases knowledge	**Good atmosphere**
Provide Facilities for Extra Curricular Acitivity	Good Facilities	High rank holders	Highly qualified professors

❸

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

⏱ 20 minutes

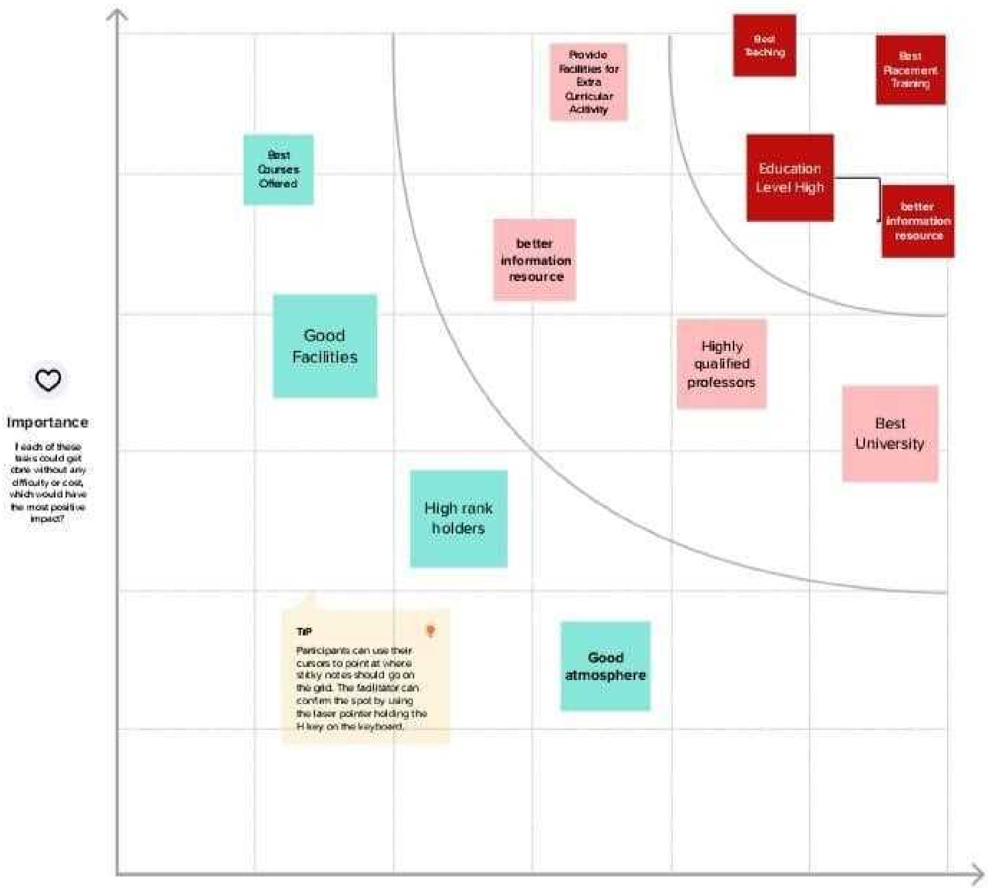


4

Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes



After you collaborate

You can export the mural as an image or pdf to share with members of your company who might find it helpful.

Quick add-ons

A **Share the mural**
Share a view link to the mural with stakeholders to keep them in the loop about the outcomes of the session.

B **Export the mural**
Export a copy of the mural as a PNG or PDF to attach to emails, include in slides, or save in your drive.

Keep moving forward

Strategy blueprint
Define the components of a new idea or strategy.
Open the template →

Customer experience journey map
Understand customer needs, motivations, and obstacles for an experience.
Open the template →

Strengths, weaknesses, opportunities & threats
Identify strengths, weaknesses, opportunities, and threats (SWOT) to develop a plan.
Open the template →

Share template feedback

3. RESULT


UNIVERSITY ADMISSION PREDICTION SYSTEM
Enter your details and get probability of your admission
Enter GRE score
Enter TOEFL score
Select University number
1
2
3
4
5
Enter SOP
Enter LOR
Enter CGPA
Research
No-research
Predict


Predicting Chance of Admission
A Machine learning Web App using Flask
Prediction : You have a chance
ADMITTED
TEST

4. ADVANTAGES & DISADVANTAGES

ADVANTAGES

- Objective and data-driven admission decisions
- Improved efficiency and cost savings
- Identifying high-potential applicants
- Addressing retention issues
- Improved diversity and inclusivity

DISADVANTAGES

- Data bias and inaccuracies
- Overreliance on data
- Lack of transparency
- Privacy concerns
- Cost of implementation

5. APPLICATIONS

- Predictive modeling
- Personalized support
- Streamlining the admission process
- Improving diversity and inclusivity
- Evaluating program effectiveness

6. CONCLUSION

Intelligence admission, or intelligent enrollment, has the potential to transform the university admission process and improve student outcomes by using machine learning algorithms to make more objective and data-driven admission decisions, streamline administrative processes, provide personalized support to students, improve diversity and inclusivity, and evaluate program effectiveness.

While there are potential disadvantages to intelligence admission, such as data bias and inaccuracies, overreliance on data, lack of transparency, privacy concerns, and implementation costs, many universities are already using machine learning algorithms to improve the admission process and create more diverse and inclusive student populations.

As technology continues to evolve and data becomes increasingly accessible, universities that adopt intelligence admission are likely to have a competitive advantage in attracting and retaining high-performing and diverse student populations. However, it is important for universities to carefully evaluate the potential advantages and disadvantages of intelligence admission and continuously monitor and improve their machine learning models to ensure they are making fair and accurate admission decisions.

7.FUTURE SCOPE

The future scope of intelligence admission in universities is vast and promising. As technology continues to advance, machine learning algorithms are likely to become even more sophisticated and capable of analyzing larger and more diverse data sets, making intelligence admission an increasingly valuable tool for universities.

Some of the potential future applications of intelligence admission include:

- Virtual admission interviews: Machine learning algorithms could be used to analyze virtual admission interviews, providing universities with insights into applicants' communication skills, critical thinking abilities, and other qualities that are difficult to measure through traditional admission processes.
- Adapting to changing job markets: As the job market continues to evolve, universities may need to adapt their admission processes to identify and recruit students with skills that are in high demand. Machine learning algorithms could be used to analyze data on job market trends and predict which skills will be most valuable in the future, helping universities tailor their admission processes to meet the needs of employers and students.
- Enhancing student engagement: Machine learning algorithms could be used to analyze data on student engagement, identifying patterns and trends that can be used to improve student retention and success.
- Continuous evaluation and improvement: As machine learning algorithms become more advanced, universities may be able to continuously evaluate and improve their admission processes, making them more accurate, efficient, and inclusive.

8. APPENDIX

A) SOURCE CODE

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

%matplotlib inline
sns.set()
warnings.simplefilter('ignore')
```

In [ ]:
```
data = pd.read_csv('Admission_Predict.csv')
```

In [ ]:
```
df = data.copy()
df.tail(20)
```

Out[ ]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
380	381	322	104	3	3.5	4.0	8.84	1	0.78
381	382	319	105	3	3.0	3.5	8.67	1	0.73
382	383	324	110	4	4.5	4.0	9.15	1	0.82
383	384	300	100	3	3.0	3.5	8.26	0	0.62
384	385	340	113	4	5.0	5.0	9.74	1	0.96
385	386	335	117	5	5.0	5.0	9.82	1	0.96
386	387	302	101	2	2.5	3.5	7.96	0	0.46
387	388	307	105	2	2.0	3.5	8.10	0	0.53
388	389	296	97	2	1.5	2.0	7.80	0	0.49
389	390	320	108	3	3.5	4.0	8.44	1	0.76
390	391	314	102	2	2.0	2.5	8.24	0	0.64
391	392	318	106	3	2.0	3.0	8.65	0	0.71
392	393	326	112	4	4.0	3.5	9.12	1	0.84
393	394	317	104	2	3.0	3.0	8.76	0	0.77
394	395	329	111	4	4.5	4.0	9.23	1	0.89
395	396	324	110	3	3.5	3.5	9.04	1	0.82
396	397	325	107	3	3.0	3.5	9.11	1	0.84
397	398	330	116	4	5.0	4.5	9.45	1	0.91
398	399	312	103	3	3.5	4.0	8.78	0	0.67
399	400	333	117	4	5.0	4.0	9.66	1	0.95

In [ ]:
```
df.drop('Serial No.', axis=1, inplace=True)
```

In [ ]:

```
df.drop('Serial No.', axis=1, inplace=True)
df.head()
```

Out[ ]:

	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 [ ]:

```
df.isnull().sum()
```

Out[ ]:

```
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

In [ ]:

```
df.shape
```

Out[ ]: (400, 8)

In [ ]:

```
df.describe()
```

Out[ ]:

	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

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

In [ ]:
```
df.shape
```

Out[ ]:
```
(400, 8)
```

In [ ]:
```
df.describe()
```

Out[ ]:

	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

In [ ]:
```
df.columns = df.columns.str.strip()
df.columns
```

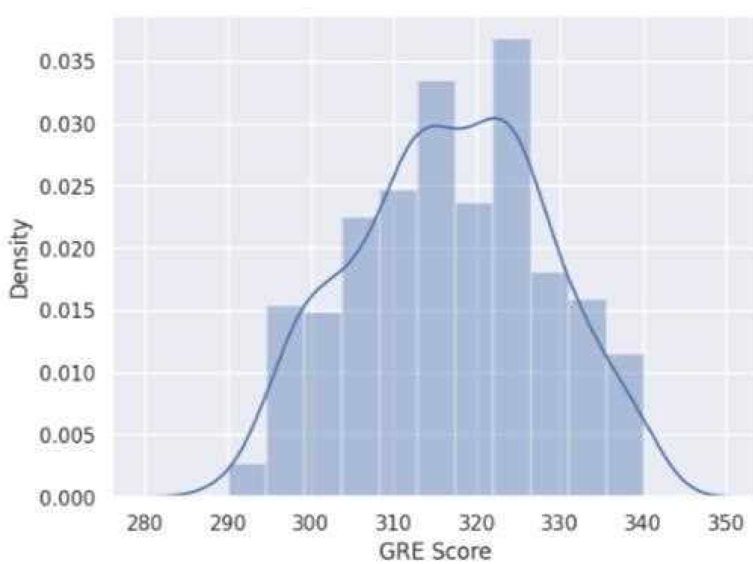
Out[ ]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR', 'CGPA',
       'Research', 'Chance of Admit'],
      dtype='object')
```

In [ ]:
```
df.columns = df.columns.str.strip()
df.columns
```

Out[ ]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR', 'CGPA',
       'Research', 'Chance of Admit'],
      dtype='object')
```

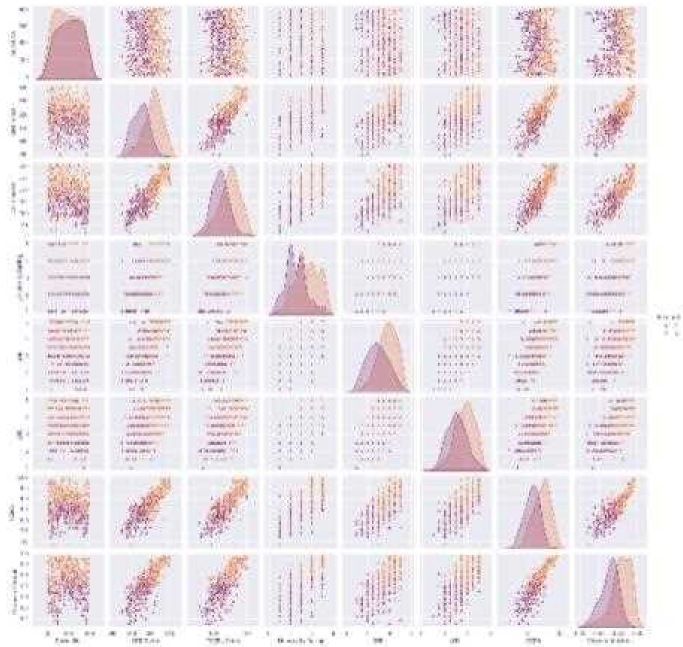

In [ ]:
```
sns.distplot(data['GRE Score'])
```

Out[ ]: <Axes: xlabel='GRE Score', ylabel='Density'>



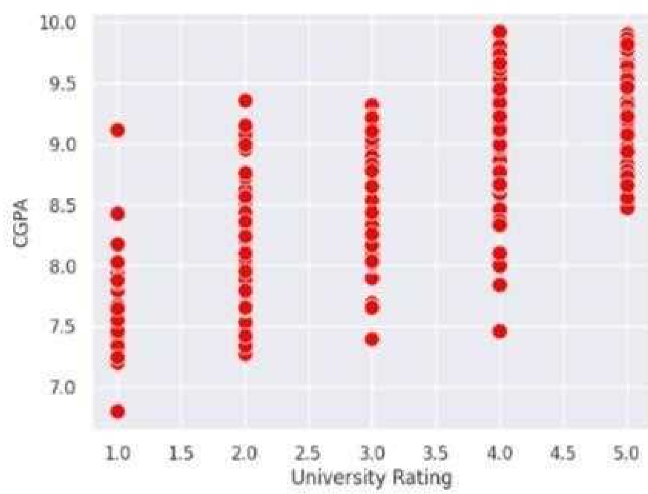
```
In [ ]: sns.pairplot(data=data,hue='Research',markers=["^","v"],palette='inferno')
```

```
Out[ ]: <seaborn.axisgrid.PairGrid at 0x7fe263576df0>
```



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

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



In [ ]:
```
df[['GRE Score', 'Chance of Admit']].corr()
```

Out[ ]:

	GRE Score	Chance of Admit
GRE Score	1.00000	0.80261
Chance of Admit	0.80261	1.00000

In [ ]:
```
category = ['GRE Score','TOEFL Score','University Rating','SOP','LOR','CGPA','Res
earch','Chance of Admit']
color = ['yellowgreen','gold','lightskyblue','pink','red','purple','orange','gra
y']
start = True
for i in np.arange(4):
  fig = plt.figure(figsize=(14,8))
  plt.subplot2grid((4,2),(i,0))
  data[category[i]].hist(color=color[i],bins=10)
  plt.title(category[2*i])
  plt.subplot2grid((4,2),(i,1))
  data[category[i+1]].hist(color=color[i+1],bins=10)
  plt.title(category[i+1])
plt.subplots_adjust(hspace = 0.7,wspace = 0.2)
plt.show()
```

In [ ]:
```
plt.figure(figsize=(10, 6))
sns.regplot(x='GRE Score', y='Chance of Admit', data=df)
```

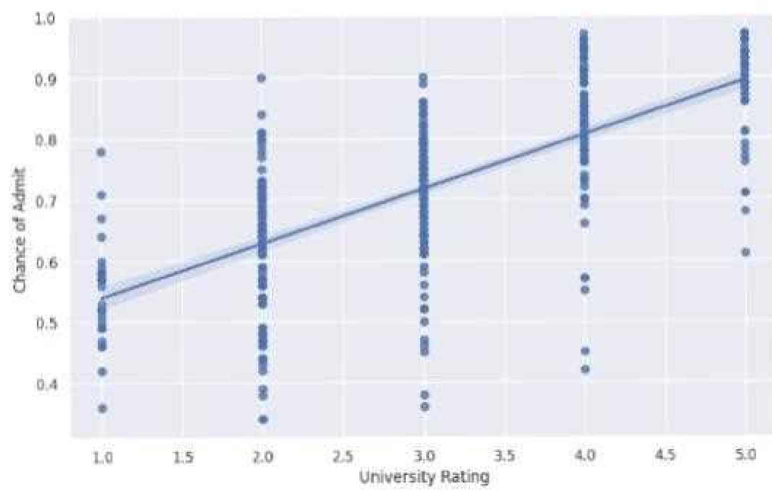
Out[ ]: <Axes: xlabel='GRE Score', ylabel='Chance of Admit'>

```
In [ ]: p_coeff, p_value = stats.pearsonr(df['TOEFL Score'], df['Chance of Admit'])
        print('Pearson Coefficient:', p_coeff)
        print('Pearson Value:      ', p_value)
```

```
Pearson Coefficient: 0.7915939869351045
Pearson Value:       3.6341021759970527e-87
```

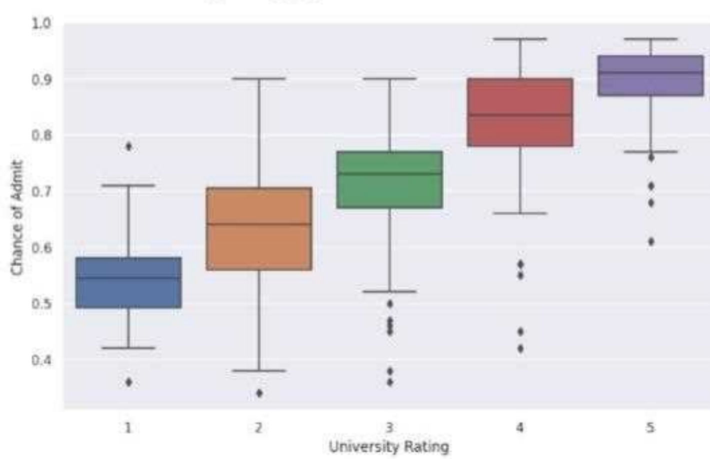
```
In [ ]: plt.figure(figsize=(10, 6))
        sns.regplot(x=df['University Rating'], y=df['Chance of Admit'])
```

Out[ ]: <Axes: xlabel='University Rating', ylabel='Chance of Admit'>




```
In [ ]: plt.figure(figsize=(10, 6))
        sns.boxplot(data=df, x='University Rating', y='Chance of Admit')
```

Out[ ]: <Axes: xlabel='University Rating', ylabel='Chance of Admit'>



In [ ]:
```
df[['University Rating', 'Chance of Admit']].corr()
```

Out[ ]:

	University Rating	Chance of Admit
University Rating	1.00000	0.71125
Chance of Admit	0.71125	1.00000

In [ ]:
```
coef, pvalue = stats.pearsonr(df['University Rating'], df['Chance of Admit'])
coef, pvalue
```

Out[ ]: (0.7112502503917222, 6.635019480888963e-63)

In [ ]:
```
df_rating_grp = df[['University Rating', 'Chance of Admit']].groupby(['University Rating'])
```

In [ ]:
```
f, pvalue = stats.f_oneway(df_rating_grp.get_group(1)['Chance of Admit'],
                           df_rating_grp.get_group(2)['Chance of Admit'],
                           df_rating_grp.get_group(3)['Chance of Admit'],
                           df_rating_grp.get_group(4)['Chance of Admit'],
                           df_rating_grp.get_group(5)['Chance of Admit'])

print('f oneway:', f, '\nP Value:', pvalue)
```

```
f oneway: 102.0800521553914
P Value: 1.313389994668425e-59
```

In [ ]:
```
plt.figure(figsize=(10, 6))
sns.regplot(data=df, x='SOP', y='Chance of Admit')
```

Out[ ]: <Axes: xlabel='SOP', ylabel='Chance of Admit'>

In [ ]:
```
df[['University Rating', 'Chance of Admit']].corr()
```

Out[ ]:

	University Rating	Chance of Admit
University Rating	1.00000	0.71125
Chance of Admit	0.71125	1.00000

In [ ]:
```
coef, pvalue = stats.pearsonr(df['University Rating'], df['Chance of Admit'])
coef, pvalue
```

Out[ ]: (0.7112502503917222, 6.635019480888963e-63)

In [ ]:
```
df_rating_grp = df[['University Rating', 'Chance of Admit']].groupby(['University Rating'])
```

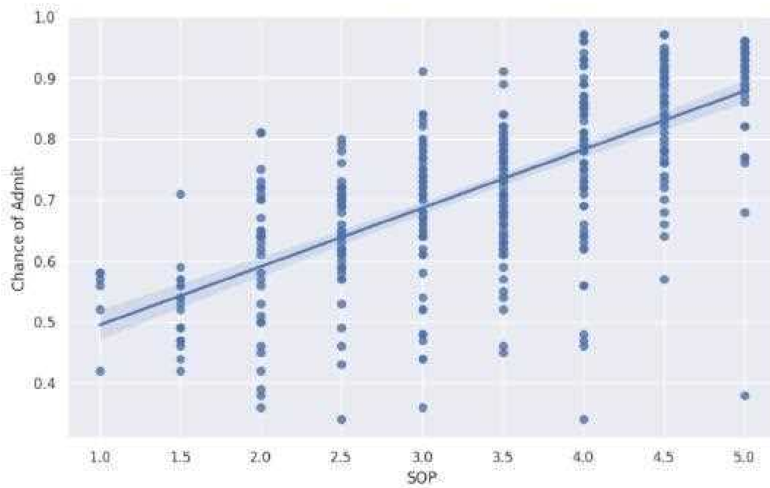
In [ ]:
```
f, pvalue = stats.f_oneway(df_rating_grp.get_group(1)['Chance of Admit'],
                           df_rating_grp.get_group(2)['Chance of Admit'],
                           df_rating_grp.get_group(3)['Chance of Admit'],
                           df_rating_grp.get_group(4)['Chance of Admit'],
                           df_rating_grp.get_group(5)['Chance of Admit'])

print('f oneway:', f, '\nP Value:', pvalue)
```

```
f oneway: 102.0800521553914
P Value: 1.313389994668425e-59
```

In [ ]:
```
plt.figure(figsize=(10, 6))
sns.regplot(data=df, x='SOP', y='Chance of Admit')
```

Out[ ]: <Axes: xlabel='SOP', ylabel='Chance of Admit'>



SOP

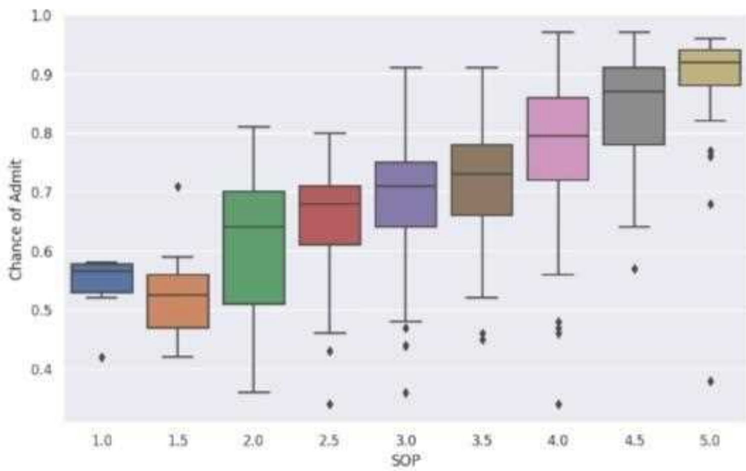
```
In [ ]: df[['SOP', 'Chance of Admit']].corr()
```

Out[ ]:

	SOP	Chance of Admit
SOP	1.000000	0.675732
Chance of Admit	0.675732	1.000000

```
In [ ]: plt.figure(figsize=(10, 6))
        sns.boxplot(data=df, x='SOP', y='Chance of Admit')
```

Out[ ]: <Axes: xlabel='SOP', ylabel='Chance of Admit'>



In [ ]:
```
p_coeff, pvalue = stats.pearsonr(df.SOP, df['Chance of Admit'])

print('Pearson Coefficient: ', p_coeff)
print('P Value:             ', pvalue)
```

```
Pearson Coefficient:  0.675731858388672
P Value:              1.1410946671022982e-54
```

In [ ]:
```
df_sop_grp = df[['SOP', 'Chance of Admit']].groupby(['SOP'])
```

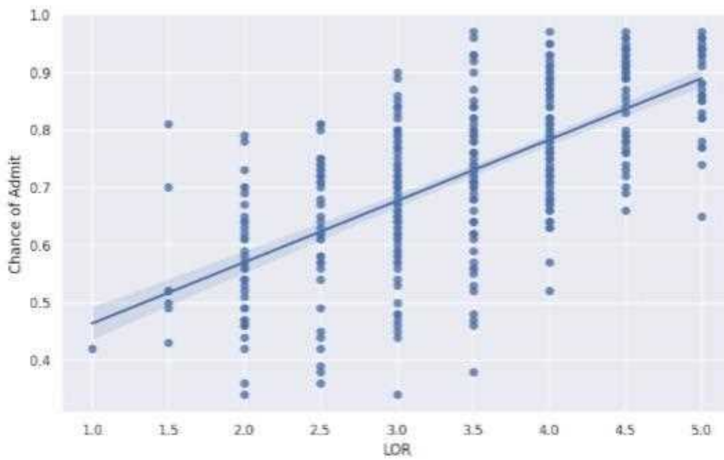
In [ ]:
```
f, pvalue = stats.f_oneway(df_sop_grp.get_group(1.0)['Chance of Admit'],
                           df_sop_grp.get_group(1.5)['Chance of Admit'],
                           df_sop_grp.get_group(2.0)['Chance of Admit'],
                           df_sop_grp.get_group(2.5)['Chance of Admit'],
                           df_sop_grp.get_group(3.0)['Chance of Admit'],
                           df_sop_grp.get_group(3.5)['Chance of Admit'],
                           df_sop_grp.get_group(4.0)['Chance of Admit'],
                           df_sop_grp.get_group(4.5)['Chance of Admit'],
                           df_sop_grp.get_group(5.0)['Chance of Admit'])

f, pvalue
```

Out[ ]:
```
(42.64667458928518, 7.2405682104781e-49)
```

In [ ]:
```
plt.figure(figsize=(10, 6))
sns.regplot(x=df.LOR, y=df['Chance of Admit'])
```

Out[ ]:
```
<Axes: xlabel='LOR', ylabel='Chance of Admit'>
```



In [ ]:
```
df[['LOR', 'Chance of Admit']].corr()
```

Out[ ]:

	LOR	Chance of Admit
LOR	1.000000	0.669889
Chance of Admit	0.669889	1.000000

In [ ]:
```
df[['LOR', 'Chance of Admit']].corr()
```

Out[ ]:

	LOR	Chance of Admit
LOR	1.000000	0.669889
Chance of Admit	0.669889	1.000000

In [ ]:

```python
df[['LOR', 'Chance of Admit']].corr()
```

Out[ ]:

	LOR	Chance of Admit
LOR	1.000000	0.669889
Chance of Admit	0.669889	1.000000

In [ ]:

```python
df[['LOR', 'Chance of Admit']].corr()
```

Out[ ]:

	LOR	Chance of Admit
LOR	1.000000	0.669889
Chance of Admit	0.669889	1.000000

In [ ]:

```python
df_lor_grp = df[['LOR', 'Chance of Admit']].groupby('LOR')
```

In [ ]:

```python
f, pvalue = stats.f_oneway(df_lor_grp.get_group(1.0)['Chance of Admit'],
                           df_lor_grp.get_group(1.5)['Chance of Admit'],
                           df_lor_grp.get_group(2.0)['Chance of Admit'],
                           df_lor_grp.get_group(2.5)['Chance of Admit'],
                           df_lor_grp.get_group(3.0)['Chance of Admit'],
                           df_lor_grp.get_group(3.5)['Chance of Admit'],
                           df_lor_grp.get_group(4.0)['Chance of Admit'],
                           df_lor_grp.get_group(4.5)['Chance of Admit'],
                           df_lor_grp.get_group(5.0)['Chance of Admit'])

f, pvalue
```

Out[ ]:

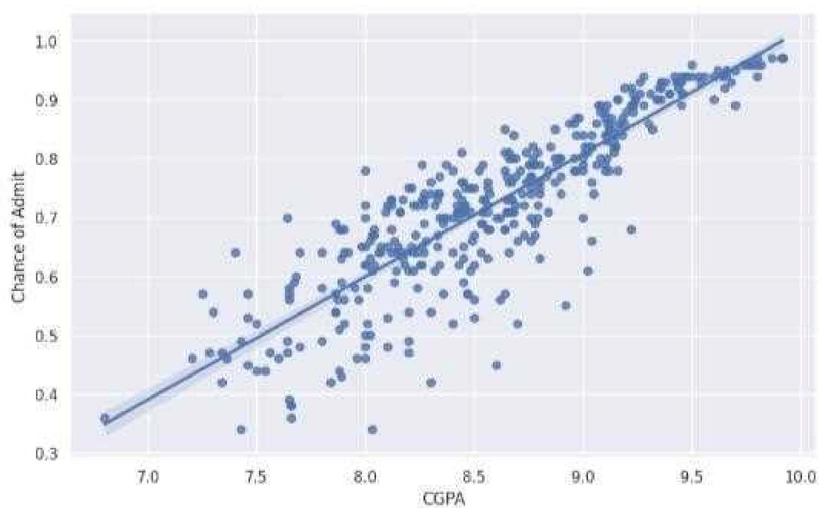
```
(40.94235310849056, 2.675772604311782e-47)
```

In [ ]:

```python
plt.figure(figsize=(10, 6))
sns.regplot(x=df.CGPA, y=df['Chance of Admit'])
```

Out[ ]:

```
<Axes: xlabel='CGPA', ylabel='Chance of Admit'>
```



In [ ]:
```
df[['CGPA', 'Chance of Admit']].corr()
```

Out[ ]:

	CGPA	Chance of Admit
CGPA	1.000000	0.873289
Chance of Admit	0.873289	1.000000

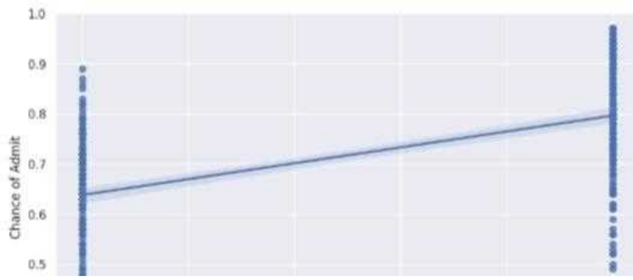
In [ ]:
```
p_coeff, pvalue = stats.pearsonr(df.CGPA, df['Chance of Admit'])
p_coeff, pvalue
```

Out[ ]: (0.8732890993553003, 2.3365140004978882e-126)

In [ ]:

In [ ]:
```
plt.figure(figsize=(10, 6))
sns.regplot(x=df.Research, y=df['Chance of Admit'])
```

Out[ ]: <Axes: xlabel='Research', ylabel='Chance of Admit'>



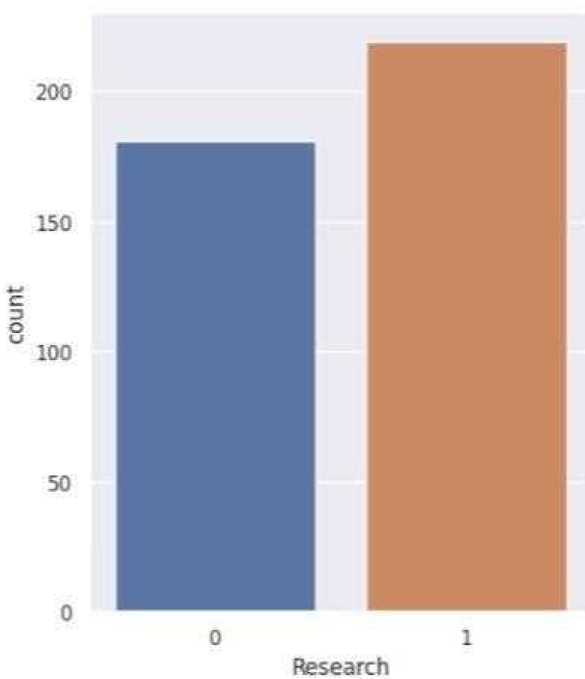
In [ ]:
```
df[['Research', 'Chance of Admit']].corr()
```

Out[ ]:

	Research	Chance of Admit
Research	1.000000	0.553202
Chance of Admit	0.553202	1.000000

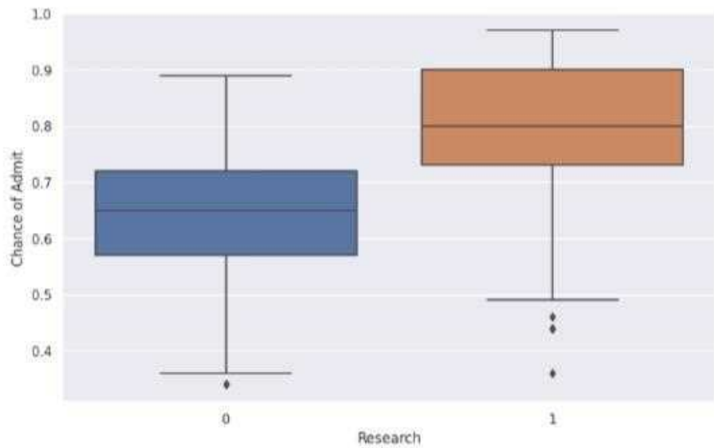
In [ ]:
```
plt.figure(figsize=(5, 6))
sns.countplot(x=df.Research)
```

Out[ ]: <Axes: xlabel='Research', ylabel='count'>



In [ ]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(x=df.Research, y=df['Chance of Admit'])
```

Out[ ]: <Axes: xlabel='Research', ylabel='Chance of Admit'>



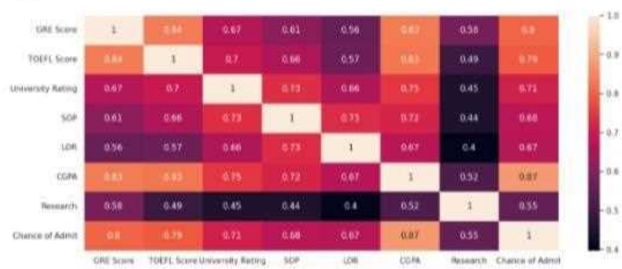
In [ ]:
```
df_res_grp = df[['Research', 'Chance of Admit']].groupby('Research')
```

In [ ]:
```
f, pvalue = stats.f_oneway(df_res_grp.get_group(1)['Chance of Admit'],
                           df_res_grp.get_group(0)['Chance of Admit'])
f, pvalue
```

Out[ ]: (175.51397562026247, 1.9181733806927185e-33)

In [ ]:
```
plt.figure(figsize=(15, 6))
sns.heatmap(data=df.corr(), annot=True)
```

Out[ ]: <Axes: >



In [ ]:
```
X = df.drop('Chance of Admit', axis=1)
Y = df[['Chance of Admit']]

X.shape, Y.shape
```

Out[ ]: ((400, 7), (400, 1))

Out[ ]: ((400, 7), (400, 1))

In [ ]:
```
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(
    X, Y, random_state=42, shuffle=True, test_size=0.30)

print(xtrain.shape, ytrain.shape)
print(xtest.shape, ytest.shape)
```

```
(280, 7) (280, 1)
(120, 7) (120, 1)
```

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

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

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

 dense_1 (Dense)             (None, 7)                 56

 dense_2 (Dense)             (None, 1)                 8

=================================================================
Total params: 120
Trainable params: 120
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
model.compile(loss= 'binary_crossentropy', optimizer ='adam',metrics =['accurac
y'])
```

In [ ]:
```
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(
    X, Y, random_state=42, shuffle=True, test_size=0.30)

print(xtrain.shape, ytrain.shape)
print(xtest.shape, ytest.shape)
```

```
(280, 7) (280, 1)
(120, 7) (120, 1)
```

In [ ]:

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

In [ ]:

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

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

 dense_1 (Dense)             (None, 7)                 56

 dense_2 (Dense)             (None, 1)                 8

=================================================================
Total params: 120
Trainable params: 120
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

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

In [ ]:

```
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(
    X, Y, random_state=42, shuffle=True, test_size=0.30)

print(xtrain.shape, ytrain.shape)
print(xtest.shape, ytest.shape)
```

```
(280, 7) (280, 1)
(120, 7) (120, 1)
```

In [ ]:

```
from sklearn.linear_model import LassoCV

lasso = LassoCV(random_state=42, n_jobs=4)

lasso.fit(xtrain, ytrain)
regularized_model_prediction = lasso.predict(xtest)
```

In [ ]:

```
from sklearn.metrics import r2_score
```

In [ ]:

```
r2_score(ytest, regularized_model_prediction)
```

Out[ ]:

```
0.7904544971285603
```


In [ ]:

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

In [ ]:

```
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(
    X, Y, random_state=42, shuffle=True, test_size=0.30)

print(xtrain.shape, ytrain.shape)
print(xtest.shape, ytest.shape)
```

```
(280, 7) (280, 1)
(120, 7) (120, 1)
```

In [ ]:

```
from sklearn.linear_model import LassoCV

lasso = LassoCV(random_state=42, n_jobs=4)

lasso.fit(xtrain, ytrain)
regularized_model_prediction = lasso.predict(xtest)
```

In [ ]:

```
from sklearn.metrics import r2_score
```

In [ ]:

```
r2_score(ytest, regularized_model_prediction)
```

Out[ ]:

```
0.7904544971285603
```

In [4]:
```
pip install virtualenv
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheel
s/public/simple/
Collecting virtualenv
  Downloading virtualenv-20.21.0-py3-none-any.whl (8.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.7/8.7 MB 18.4 MB/s eta 0:0
0:00
Requirement already satisfied: filelock<4,>=3.4.1 in /usr/local/lib/python3.9/dist
-packages (from virtualenv) (3.10.7)
Collecting distlib<1,>=0.3.6
  Downloading distlib-0.3.6-py2.py3-none-any.whl (468 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 468.5/468.5 kB 7.5 MB/s eta 0:
00:00
Requirement already satisfied: platformdirs<4,>=2.4 in /usr/local/lib/python3.9/di
st-packages (from virtualenv) (3.2.0)
Installing collected packages: distlib, virtualenv
Successfully installed distlib-0.3.6 virtualenv-20.21.0
```

In [10]:
```
pip install flask
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheel
s/public/simple/
Requirement already satisfied: flask in /usr/local/lib/python3.9/dist-packages (2.
2.3)
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.9/dist-
packages (from flask) (2.1.2)
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.9/dist-package
s (from flask) (8.1.3)
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.9/dist-packag
es (from flask) (3.1.2)
Requirement already satisfied: Werkzeug>=2.2.2 in /usr/local/lib/python3.9/dist-pa
ckages (from flask) (2.2.3)
Requirement already satisfied: importlib-metadata>=3.6.0 in /usr/local/lib/python
3.9/dist-packages (from flask) (6.1.0)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.9/dist-packages
(from importlib-metadata>=3.6.0->flask) (3.15.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.9/dist-pa
ckages (from Jinja2>=3.0->flask) (2.1.2)
```

In [13]:
```
pip install virtualenv
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheel
s/public/simple/
Requirement already satisfied: virtualenv in /usr/local/lib/python3.9/dist-package
s (20.21.0)
Requirement already satisfied: distlib<1,>=0.3.6 in /usr/local/lib/python3.9/dist-
packages (from virtualenv) (0.3.6)
Requirement already satisfied: platformdirs<4,>=2.4 in /usr/local/lib/python3.9/di
st-packages (from virtualenv) (3.2.0)
Requirement already satisfied: filelock<4,>=3.4.1 in /usr/local/lib/python3.9/dist
-packages (from virtualenv) (3.10.7)
```

In [14]:
```
!apt-get -qq install -y virtualenv
```

```
Selecting previously unselected package python-pip-whl.
(Reading database ... 122349 files and directories currently installed.)
Preparing to unpack .../0-python-pip-whl_20.0.2-5ubuntu1.8_all.deb ...
Unpacking python-pip-whl (20.0.2-5ubuntu1.8) ...
Selecting previously unselected package python3-appdirs.
Preparing to unpack .../1-python3-appdirs_1.4.3-2.1_all.deb ...
Unpacking python3-appdirs (1.4.3-2.1) ...
Selecting previously unselected package python3-distlib.
Preparing to unpack .../2-python3-distlib_0.3.0-1_all.deb ...
Unpacking python3-distlib (0.3.0-1) ...
Selecting previously unselected package python3-filelock.
Preparing to unpack .../3-python3-filelock_3.0.12-2_all.deb ...
Unpacking python3-filelock (3.0.12-2) ...
Selecting previously unselected package python3-more-itertools.
Preparing to unpack .../4-python3-more-itertools_4.2.0-1build1_all.deb ...
Unpacking python3-more-itertools (4.2.0-1build1) ...
Selecting previously unselected package python3-zipp.
Preparing to unpack .../5-python3-zipp_1.0.0-1_all.deb ...
Unpacking python3-zipp (1.0.0-1) ...
Selecting previously unselected package python3-importlib-metadata.
Preparing to unpack .../6-python3-importlib-metadata_1.5.0-1_all.deb ...
Unpacking python3-importlib-metadata (1.5.0-1) ...
Selecting previously unselected package python3-virtualenv.
Preparing to unpack .../7-python3-virtualenv_20.0.17-1ubuntu0.4_all.deb ...
Unpacking python3-virtualenv (20.0.17-1ubuntu0.4) ...
Selecting previously unselected package virtualenv.
Preparing to unpack .../8-virtualenv_20.0.17-1ubuntu0.4_all.deb ...
Unpacking virtualenv (20.0.17-1ubuntu0.4) ...
Setting up python3-more-itertools (4.2.0-1build1) ...
Setting up python3-filelock (3.0.12-2) ...
Setting up python3-distlib (0.3.0-1) ...
Setting up python3-zipp (1.0.0-1) ...
Setting up python-pip-whl (20.0.2-5ubuntu1.8) ...
Setting up python3-appdirs (1.4.3-2.1) ...
Setting up python3-importlib-metadata (1.5.0-1) ...
Setting up python3-virtualenv (20.0.17-1ubuntu0.4) ...
Setting up virtualenv (20.0.17-1ubuntu0.4) ...
Processing triggers for man-db (2.9.1-1) ...
```

In [ ]:

```
Y=np.array(df[df.columns[-1]])
X=np.array(df.drop(df.columns[-1],axis=1))
```

```
---------------------------------------------------------------------------NameError
----> 1 Y=np.array(df[df.columns[-1]])
      2 X=np.array(df.drop(df.columns[-1],axis=1))
NameError: name 'np' is not defined
```

In [ ]:

```
import numpy as np
```

In [ ]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense ,Dropout,BatchNormalization
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
```

In [ ]:

```
df = pd.read_csv('Admission_Predict.csv')
df.head()
```

Out [5]:

	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

In [ ]:

```
df=df.drop("Serial No.",axis=1)
```

In [ ]:

```
Y=np.array(df[df.columns[-1]])
X=np.array(df.drop(df.columns[-1],axis=1))
```

In [ ]:

```
def baseline_model():
    model = Sequential()
    model.add(Dense(16, input_dim=7, activation='relu'))
    model.add(Dense(16, input_dim=7, activation='relu'))
    model.add(Dense(16, input_dim=7, activation='relu'))
    model.add(Dense(16, input_dim=7, activation='relu'))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.2, ra
```

In [ ]:

```
from sklearn.preprocessing import MinMaxScaler
scaler =  MinMaxScaler()
X_train=scaler.fit_transform(X_train)
X_test=scaler.fit_transform(X_test)
```


In [ ]:

```
def baseline_model():
    model = Sequential()
    model.add(Dense(16, input_dim=7, activation='relu'))
    model.add(Dense(16, input_dim=7, activation='relu'))
    model.add(Dense(16, input_dim=7, activation='relu'))
    model.add(Dense(16, input_dim=7, activation='relu'))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.2, ra
```

In [ ]:

```
from sklearn.preprocessing import MinMaxScaler
scaler =  MinMaxScaler()
X_train=scaler.fit_transform(X_train)
X_test=scaler.fit_transform(X_test)
```

In [ ]:

```
def baseline_model():
    model = Sequential()
    model.add(Dense(16, input_dim=7, activation='relu'))
    model.add(Dense(16, input_dim=7, activation='relu'))
    model.add(Dense(16, input_dim=7, activation='relu'))
    model.add(Dense(16, input_dim=7, activation='relu'))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```

In [ ]:

```
estimator = KerasRegressor(build_fn=baseline_model, epochs=50, batch_size=
estimator.fit(X_train,y_train)
```

```
<ipython-input-18-17af502520a3>:1: DeprecationWarning: KerasRegressor is deprecated, use
Sci-Keras (https://github.com/adriangb/scikeras) instead. See
https://www.adriangb.com/scikeras/stable/migration.html for help migrating.
  estimator = KerasRegressor(build_fn=baseline_model, epochs=50, batch_size=3, verbose=1)
```

```
Epoch 1/50
107/107 [==============================] - 2s 2ms/step - loss: 0.4299
Epoch 2/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0096
Epoch 3/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0058
Epoch 4/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0056
Epoch 5/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0051
Epoch 6/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0049
Epoch 7/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0048
Epoch 8/50
107/107 [==============================] - 0s 3ms/step - loss: 0.0047
Epoch 9/50
107/107 [==============================] - 0s 3ms/step - loss: 0.0047
Epoch 10/50
107/107 [==============================] - 0s 3ms/step - loss: 0.0047
Epoch 11/50
107/107 [==============================] - 0s 3ms/step - loss: 0.0046
Epoch 12/50
107/107 [==============================] - 0s 3ms/step - loss: 0.0046
Epoch 13/50
107/107 [==============================] - 0s 3ms/step - loss: 0.0044
Epoch 14/50
107/107 [==============================] - 0s 3ms/step - loss: 0.0043
Epoch 15/50
107/107 [==============================] - 0s 3ms/step - loss: 0.0043
Epoch 16/50
107/107 [==============================] - 0s 3ms/step - loss: 0.0044
Epoch 17/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0041
Epoch 18/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0043
Epoch 19/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0042
Epoch 20/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0042
Epoch 21/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0042
Epoch 22/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0041
Epoch 23/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0041
Epoch 24/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0040
Epoch 25/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0041
Epoch 26/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0043
Epoch 27/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0043
Epoch 28/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0040
Epoch 29/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0041
Epoch 30/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0038
Epoch 31/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0038
Epoch 32/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0038
Epoch 33/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0039
Epoch 34/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0039
Epoch 35/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0038
Epoch 36/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0038
Epoch 37/50
107/107 [==============================] - 0s 2ms/step - loss: 0.0037
```

In [ ]:

```
prediction = estimator.predict(X_test)
print("ORIGINAL DATA")
print(y_test)
print()
print("PREDICTED DATA")
print(prediction)
```

```
27/27 [==============================] - 0s 2ms/step
ORIGINAL DATA
[0.71 0.7  0.79 0.73 0.72 0.48 0.77 0.71 0.9  0.94 0.58 0.89 0.72 0.57
 0.78 0.42 0.64 0.84 0.63 0.72 0.9  0.83 0.57 0.47 0.85 0.67 0.44 0.54
 0.92 0.62 0.68 0.73 0.73 0.61 0.55 0.74 0.64 0.89 0.73 0.95 0.71 0.72
 0.75 0.76 0.86 0.7  0.39 0.79 0.61 0.64 0.71 0.8  0.61 0.89 0.68 0.79
 0.78 0.52 0.76 0.88 0.74 0.49 0.65 0.59 0.87 0.89 0.81 0.9  0.8  0.76
 0.68 0.87 0.68 0.64 0.91 0.61 0.69 0.62 0.93 0.43]

PREDICTED DATA
[0.6559105  0.67299414 0.75992036 0.6117637  0.7077769  0.5624013
 0.7010281  0.6346817  0.8103199  0.9314947  0.5508568  0.9050411
 0.65451455 0.3717843  0.86447656 0.63095546 0.5532126  0.7783222
 0.5331387  0.731367   0.8869078  0.82971656 0.5965117  0.37757337
 0.7842345  0.5257348  0.4182471  0.61097455 0.9085853  0.5908713
 0.59066164 0.7191422  0.7421001  0.51280713 0.750962   0.7548219
 0.6320852  0.8796706  0.5891839  0.9377698  0.7058518  0.6428809
 0.69463897 0.81177986 0.75831044 0.6005195  0.5537368  0.7040138
 0.6107122  0.5583876  0.6462151  0.77534497 0.5957995  0.9016496
 0.70678914 0.73191154 0.7451943  0.71617424 0.7382009  0.8276694
 0.76691854 0.38362888 0.6127033  0.46261355 0.8665736  0.8076359
 0.70293987 0.88685155 0.7271327  0.7298614  0.5832324  0.81217515
 0.8260118  0.59742844 0.9394785  0.59139097 0.6227827  0.6636932
 0.9141321  0.5074843 ]
```

In [ ]:

```
from sklearn.metrics import accuracy_score

train_error =  np.abs(y_test - prediction)
mean_error = np.mean(train_error)

print("Mean Error: ",mean_error)
```

```
Mean Error:  0.06158885289728642
```

In [ ]:
```
y_train=(y_train>0.5)
y_train
```

Out [23]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True, False,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True, False,  True,  True,  True,  True,  True,
        True,  True,  True,  True, False,  True,  True,  True, False,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True, False,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True, False,  True,  True,  True,
        True, False,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True, False,  True,  True,  True,  True,  True, False,  True,
        True, False,  True,  True,  True,  True,  True,  True,  True,
       False,  True,  True,  True,  True,  True,  True,  True, False,
        True,  True,  True, False,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True, False,  True,  True, False,  True,  True,
        True,  True,  True,  True,  True, False, False,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True, False,  True, False,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True, False,  True, False,  True,
        True,  True,  True, False,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True, False,  True,  True, False,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True, False, False,  True,  True,  True,  True,
        True,  True,  True,  True,  True, False,  True,  True,  True,
        True,  True, False,  True,  True])
```

In [ ]:
```
y_test=(y_test>0.5)
```

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

In [ ]:
```
model=keras.Sequential()
```

In [ ]:
```
model.add(Dense(7,activation='relu',input_dim=7))
```

In [ ]:
```
model.add(Dense(1,activation='linear'))
```

In [ ]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 7)                 56

 dense_6 (Dense)             (None, 1)                 8

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

In [ ]:
```
model.save('model.h5')
```

In [ ]:
```
import numpy as np
from flask import Flask,request,jsonify,render_template
import pickle
app =Flask(__name__)
from tensorflow.keras.models import load_model
```

In [ ]:
```
model =load_model('model.h5')
```

In [ ]:

```
@app.route('/y_predict',methods=['POST'])
def y_predict():

  min1=[290.0,92.0,1.0,1.0,6.8,0.0]
  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(1)
prediction = model.predict([p])
print(prediction)
output=prediction[0]
if(output==false):
  return render_template('nochance.html',prediction_text='you dont have a
else:
  return render_template('chance.html',prediction_text='you hace a chance
if __name__=="__main__" :
  app.run(debug=false)
```