Jamboree Education

Jamboree plays significant role where thousands of students make it to top colleges abroad. Be it GMAT, GRE or SAT, their unique problem-solving methods ensure maximum scores with minimum effort. They recently launched a feature where students/learners can come to their website and check their probability of getting into the IVY league college. This feature estimates the chances of graduate admission from an Indian perspective.

Your analysis will help Jamboree in understanding what factors are important in graduate admissions and how these factors are interrelated among themselves. It will also help predict one's chances of admission given the rest of the variables.

In [138…
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import Lasso, Ridge
```

In [138…
```
dt = pd.read_csv('Jamboree_Admission.csv')
dt.head()
```

Out[1389]:

	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 [139…
```
dt.shape
```

Out[1390]:
```
(500, 9)
```

In [139…
```
dt.size
```

Out[1391]:
```
4500
```

In [139…
```
dt.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         500 non-null    int64  
 1   GRE Score          500 non-null    int64  
 2   TOEFL Score        500 non-null    int64  
 3   University Rating  500 non-null    int64  
 4   SOP                500 non-null    float64
 5   LOR                500 non-null    float64
 6   CGPA               500 non-null    float64
 7   Research           500 non-null    int64  
 8   Chance of Admit    500 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 35.3 KB
```

In [139…
```
dt.describe()
```

Out[1393]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000
mean	250.500000	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000
std	144.481833	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000
25%	125.750000	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000
50%	250.500000	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000
75%	375.250000	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000
max	500.000000	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000

In [139…
```
dt.head()
```

Out[1394]:

	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 [139…
```
dt.isna().sum()
```

Out[1395]:
```
Serial No.           0
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

In [139…
```
dt.columns
```

Out[1396]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

In [139…
```
dt.rename(columns={'LOR ':'LOR'})
```

Out[1397]:

	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
…	…	…	…	…	…	…	…	…	…
495	496	332	108	5	4.5	4.0	9.02	1	0.87
496	497	337	117	5	5.0	5.0	9.87	1	0.96
497	498	330	120	5	4.5	5.0	9.56	1	0.93
498	499	312	103	4	4.0	5.0	8.43	0	0.73
499	500	327	113	4	4.5	4.5	9.04	0	0.84

500 rows × 9 columns

In [139…
```
dt.drop(columns = {'Serial No.'},inplace = True)
dt.head()
```

Out[1398]:

	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 [139…
```
dt.duplicated().sum()
```

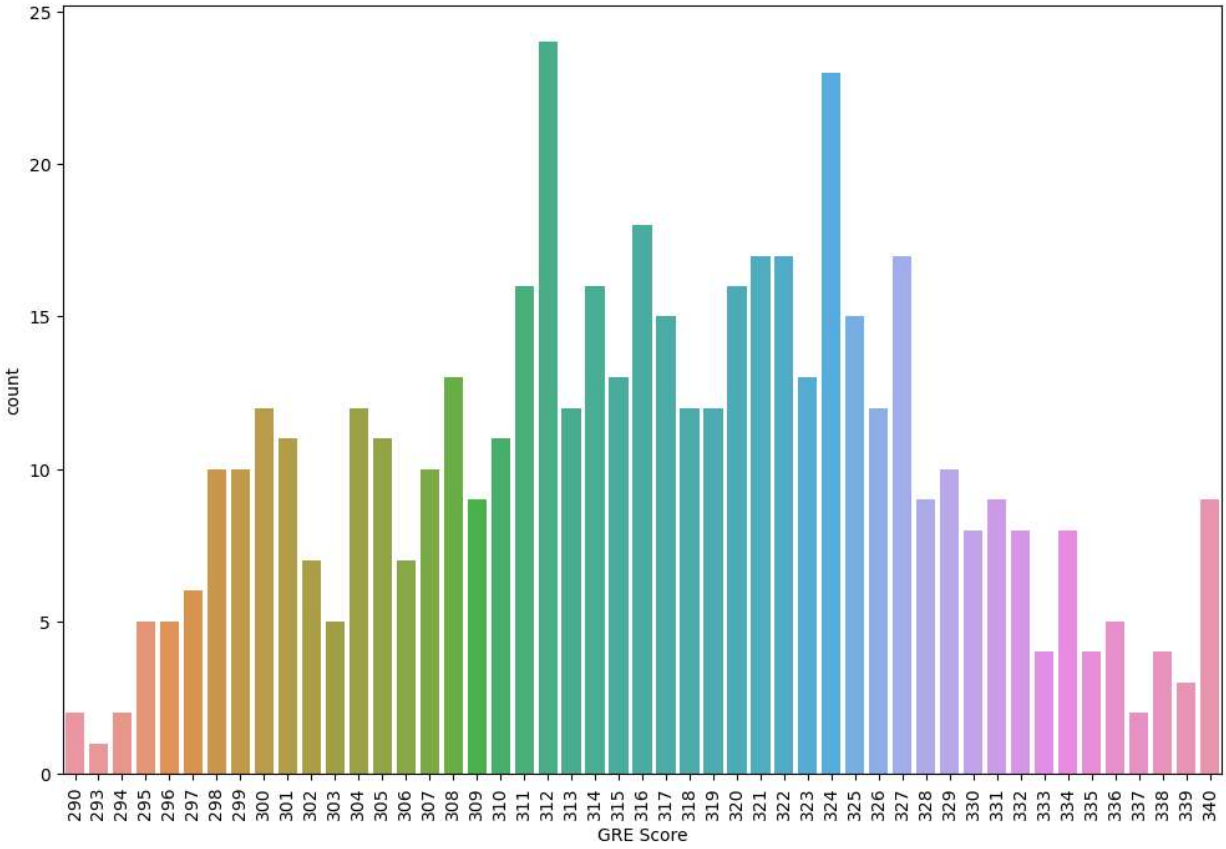
Out[1399]:
```
0
```

In [140…
```
dt.columns
```

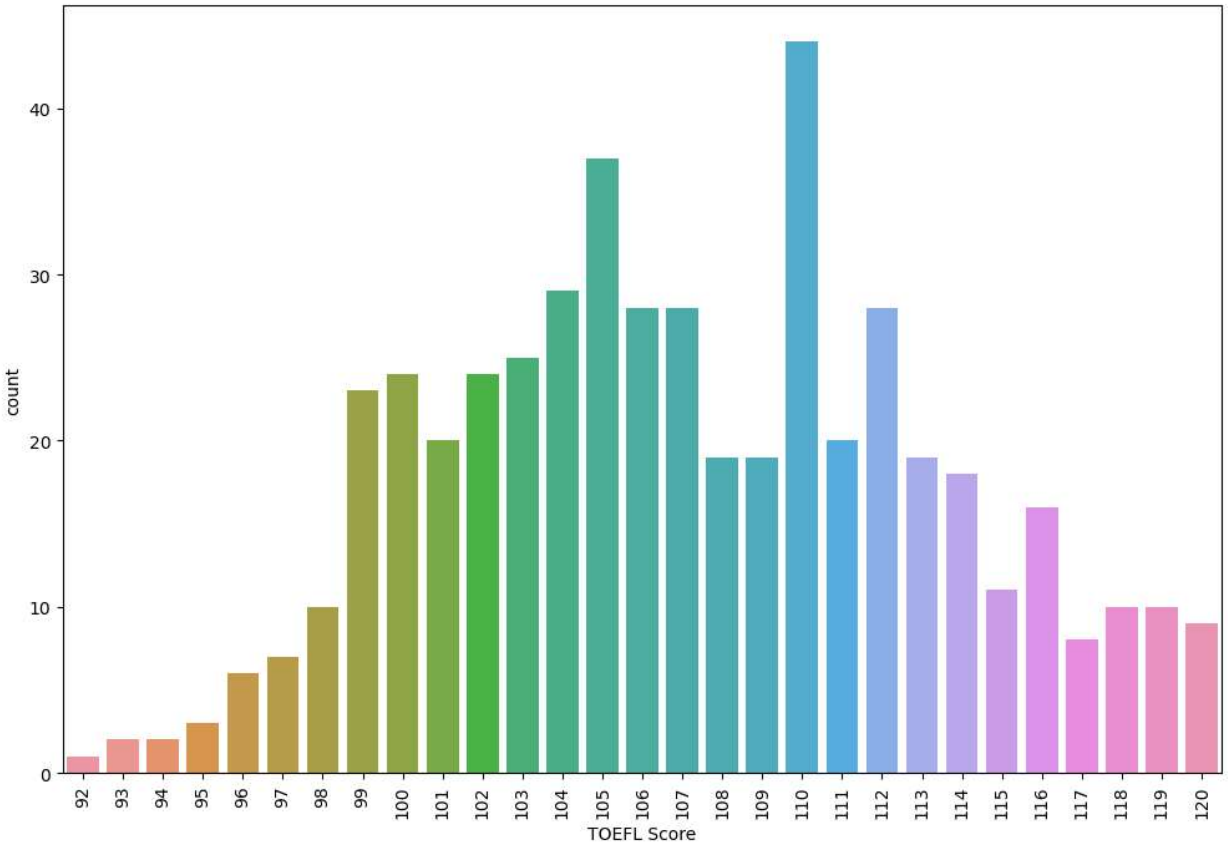
Out[1400]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

Analysis of Univariate and Bivariate

In [140…
```
plt.figure(figsize = (12,8))
sns.countplot(dt, x= dt['GRE Score'])
plt.xticks(rotation = 90)
plt.show()
```



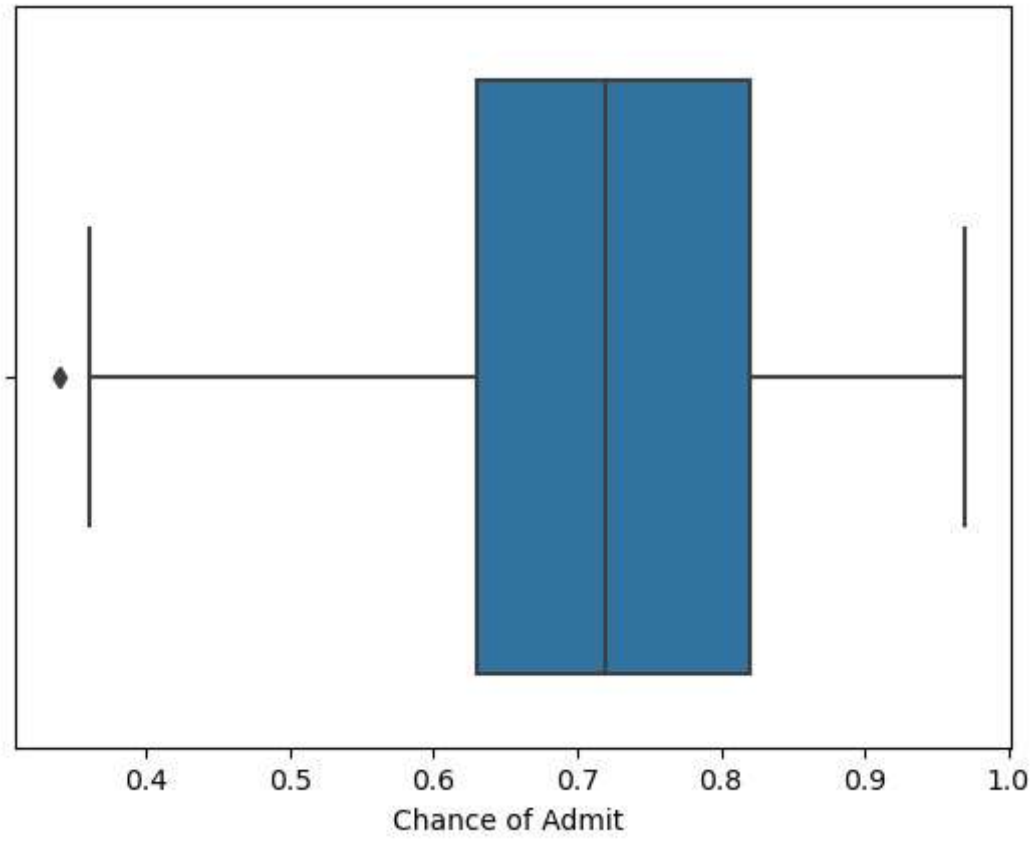
In [140…
```
plt.figure(figsize = (12,8))
sns.countplot(dt, x= dt['TOEFL Score'])
plt.xticks(rotation = 90)
plt.show()
```



In [ ]:

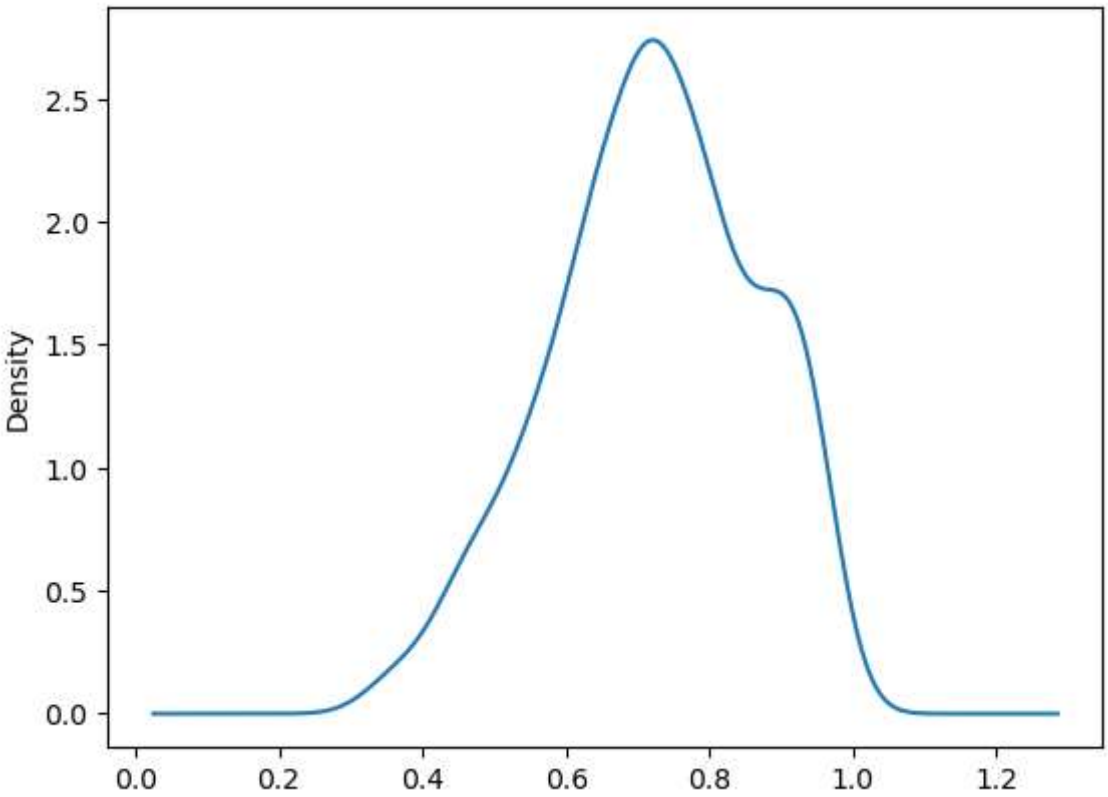
In [140…
```
sns.boxplot(dt, x = dt['Chance of Admit '])
```

Out[1403]: <Axes: xlabel='Chance of Admit '>



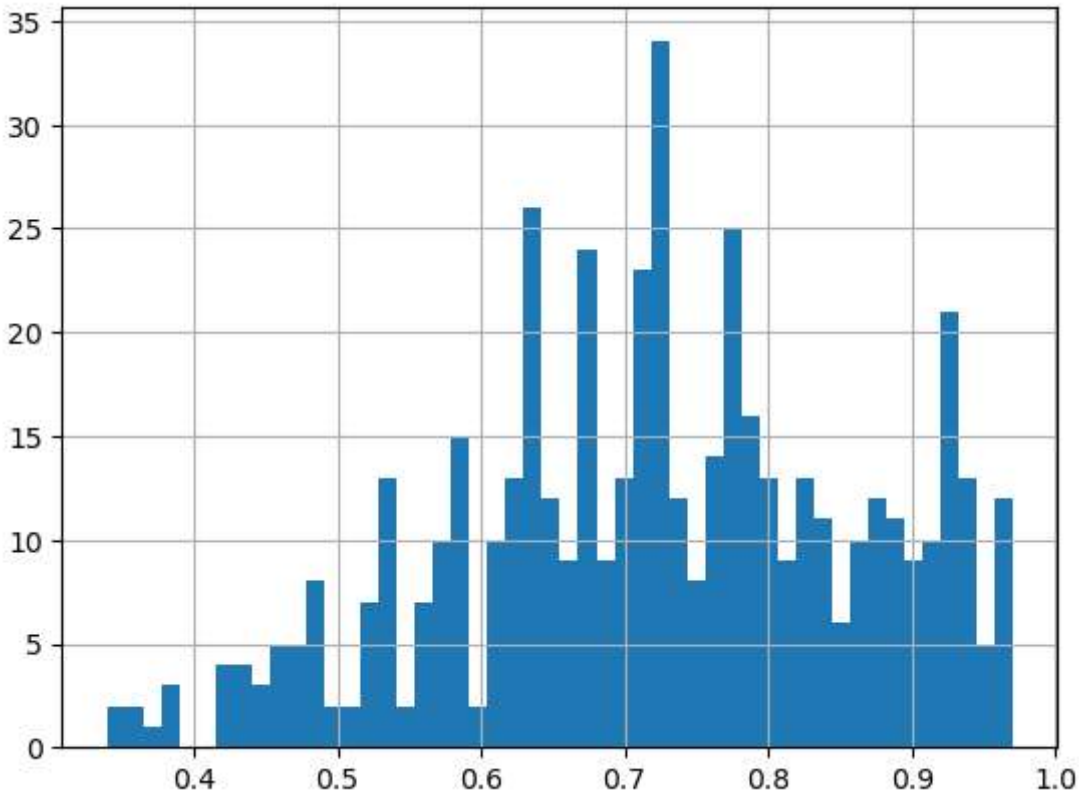
In [140…
```
dt['Chance of Admit '].plot.density()
```

Out[1404]: <Axes: ylabel='Density'>



In [140…
```
dt['Chance of Admit '].hist(bins=50)
```

Out[1405]: <Axes: >



In [140…
```
dt.columns
```

Out[1406]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

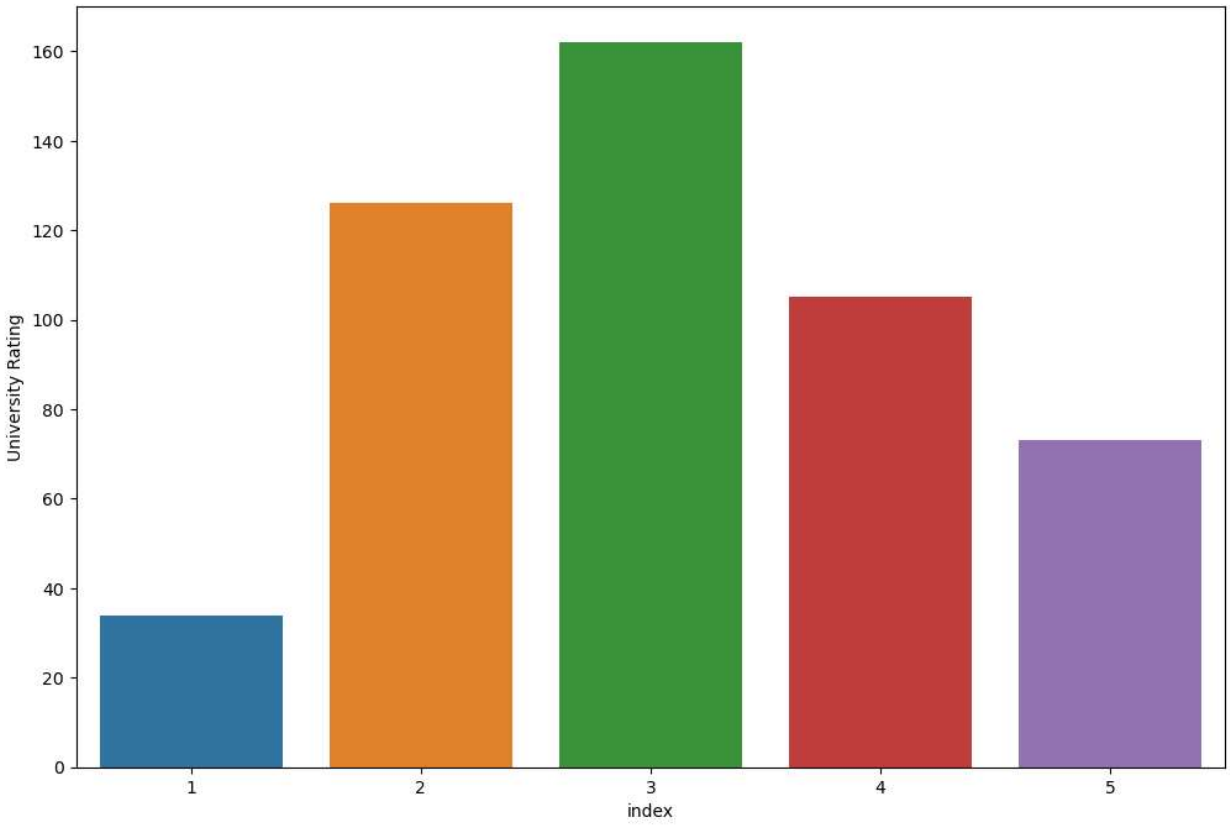
University rating

In [140…
```
University_Rating = pd.DataFrame(dt['University Rating'].value_counts()).reset_index()
University_Rating
```

Out[1407]:

	index	University Rating
0	3	162
1	2	126
2	4	105
3	5	73
4	1	34

In [140…
```
plt.figure(figsize=(12,8))
sns.barplot(data = University_Rating, x = 'index', y = 'University Rating')
plt.show()
```



GRE Score

In [140…

```
GRE_Score = pd.DataFrame(dt['GRE Score'].value_counts()).reset_index()
GRE_Score
```

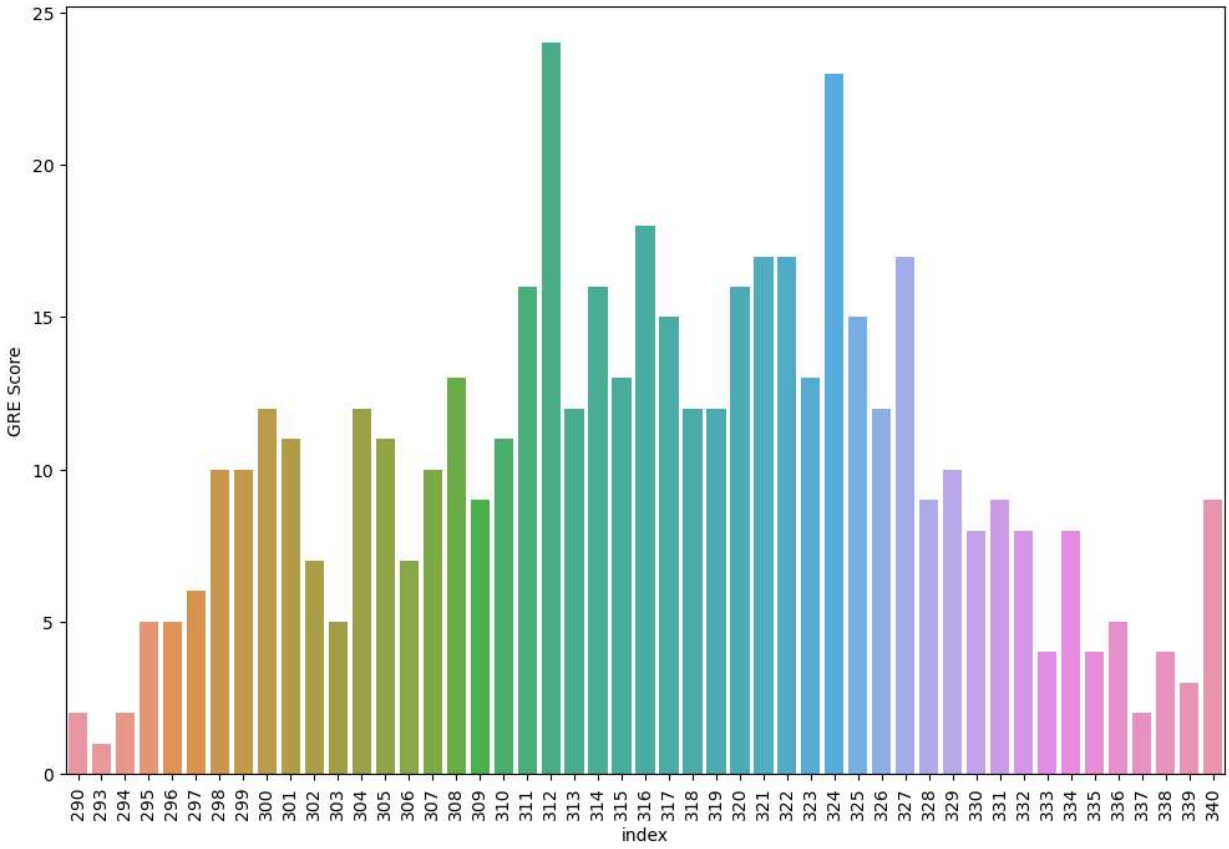
Out[1409]:

	index	GRE Score
0	312	24
1	324	23
2	316	18
3	321	17
4	322	17
5	327	17
6	311	16
7	320	16
8	314	16
9	317	15
10	325	15
11	315	13
12	308	13
13	323	13
14	326	12
15	319	12
16	313	12
17	304	12
18	300	12
19	318	12
20	305	11
21	301	11
22	310	11
23	307	10
24	329	10
25	299	10
26	298	10
27	331	9
28	340	9
29	328	9
30	309	9
31	334	8
32	332	8
33	330	8

	index	GRE Score
34	306	7
35	302	7
36	297	6
37	296	5
38	295	5
39	336	5
40	303	5
41	338	4
42	335	4
43	333	4
44	339	3
45	337	2
46	290	2
47	294	2
48	293	1

In [141…

```
plt.figure(figsize=(12,8))
sns.barplot(data = GRE_Score, x='index', y = 'GRE Score')
plt.xticks(rotation = 90)
plt.show()
```



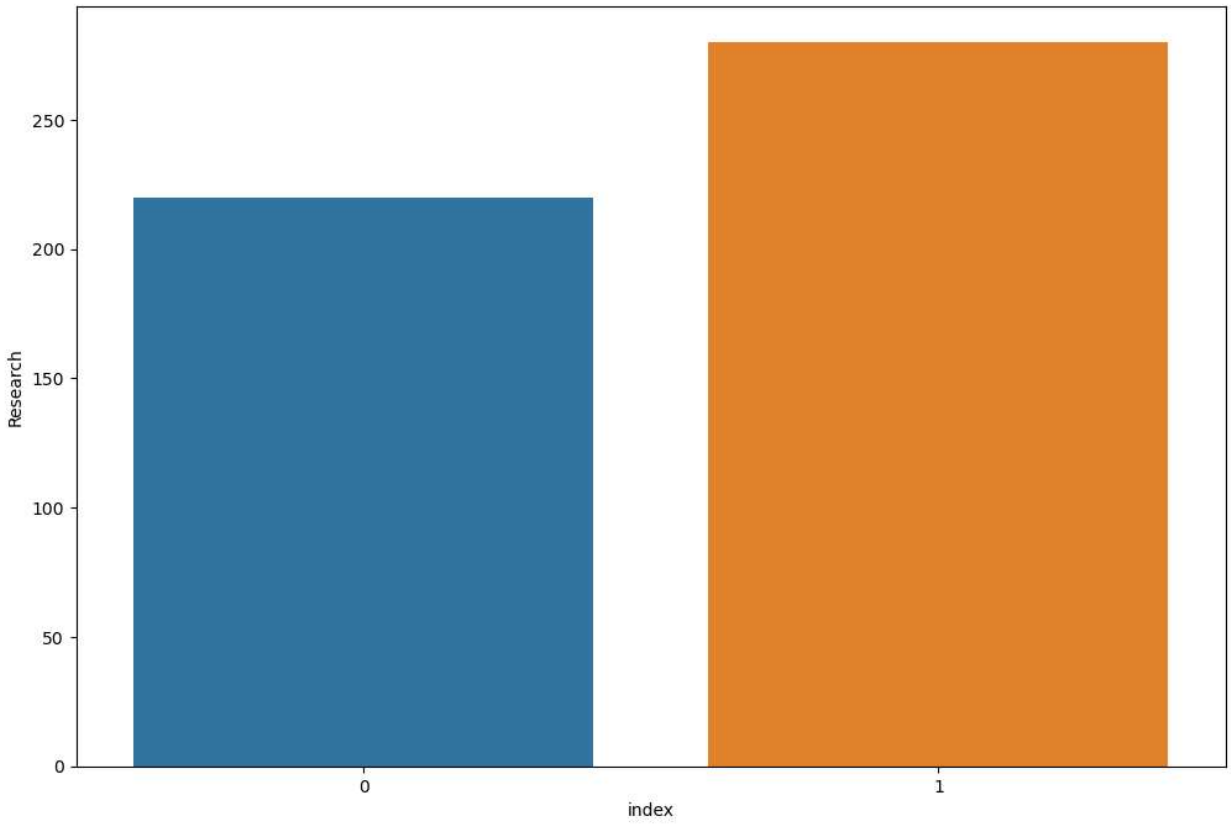
Research

In [141…
```
Research = pd.DataFrame(dt['Research'].value_counts()).reset_index()
Research
```

Out[1411]:

	index	Research
0	1	280
1	0	220

In [141…
```
plt.figure(figsize = (12,8))
sns.barplot(data = Research, x = 'index' , y='Research')
plt.show()
```



The above graph shows the corelation between reserach experiences and students, gshowing the frequency of observations within this relationship.

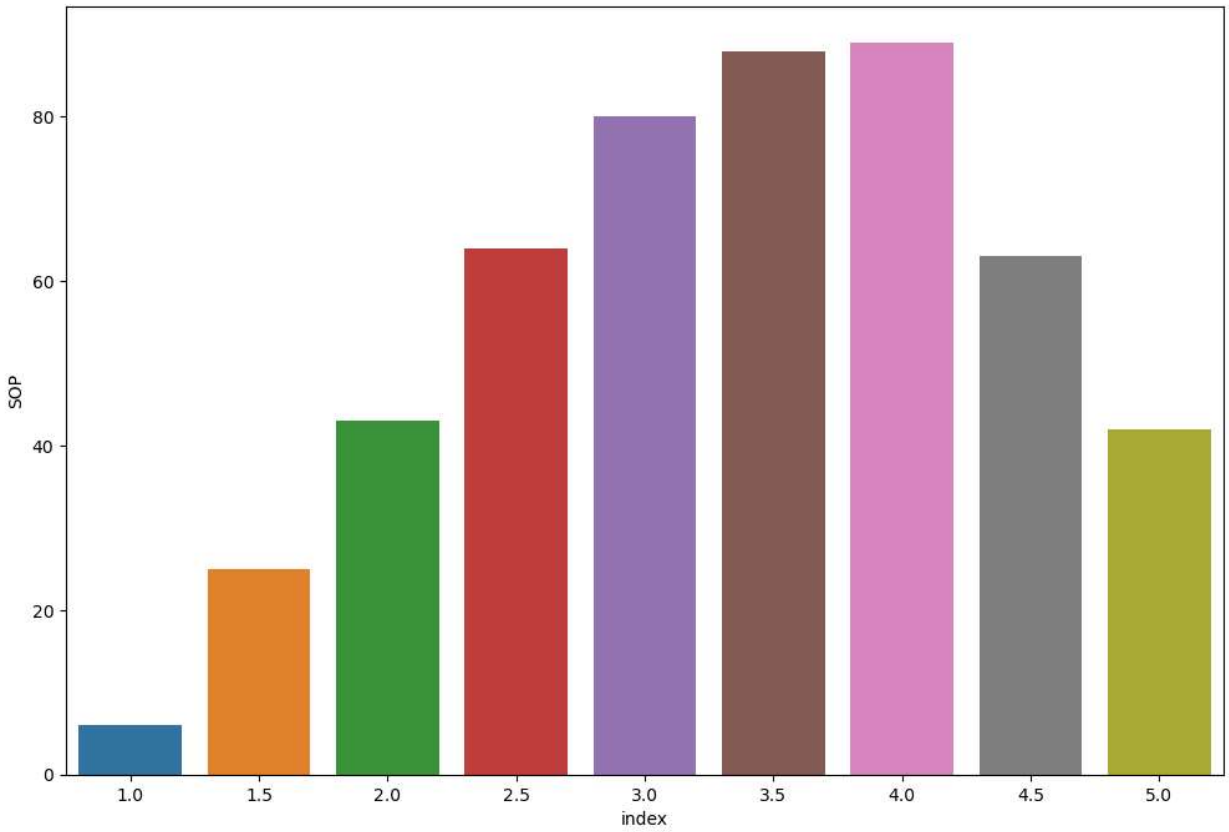
SOP

In [141…
```
SOP = pd.DataFrame(dt['SOP'].value_counts()).reset_index()
SOP
```

Out[1413]:

	index	SOP
0	4.0	89
1	3.5	88
2	3.0	80
3	2.5	64
4	4.5	63
5	2.0	43
6	5.0	42
7	1.5	25
8	1.0	6

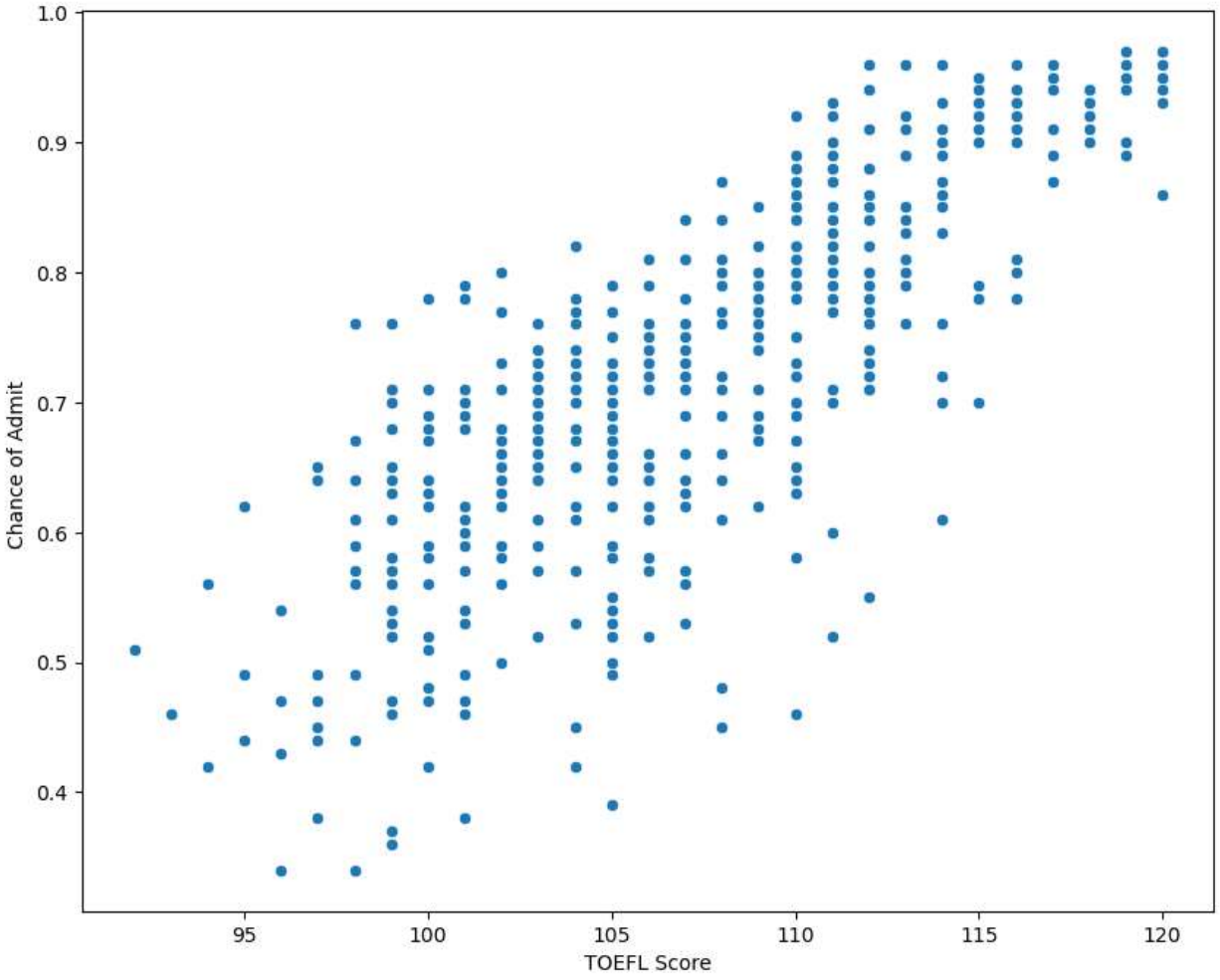
In [141…
```
plt.figure(figsize=(12,8))
sns.barplot(data = SOP, x = 'index', y = 'SOP')
plt.show()
```



Bivariate Analysis

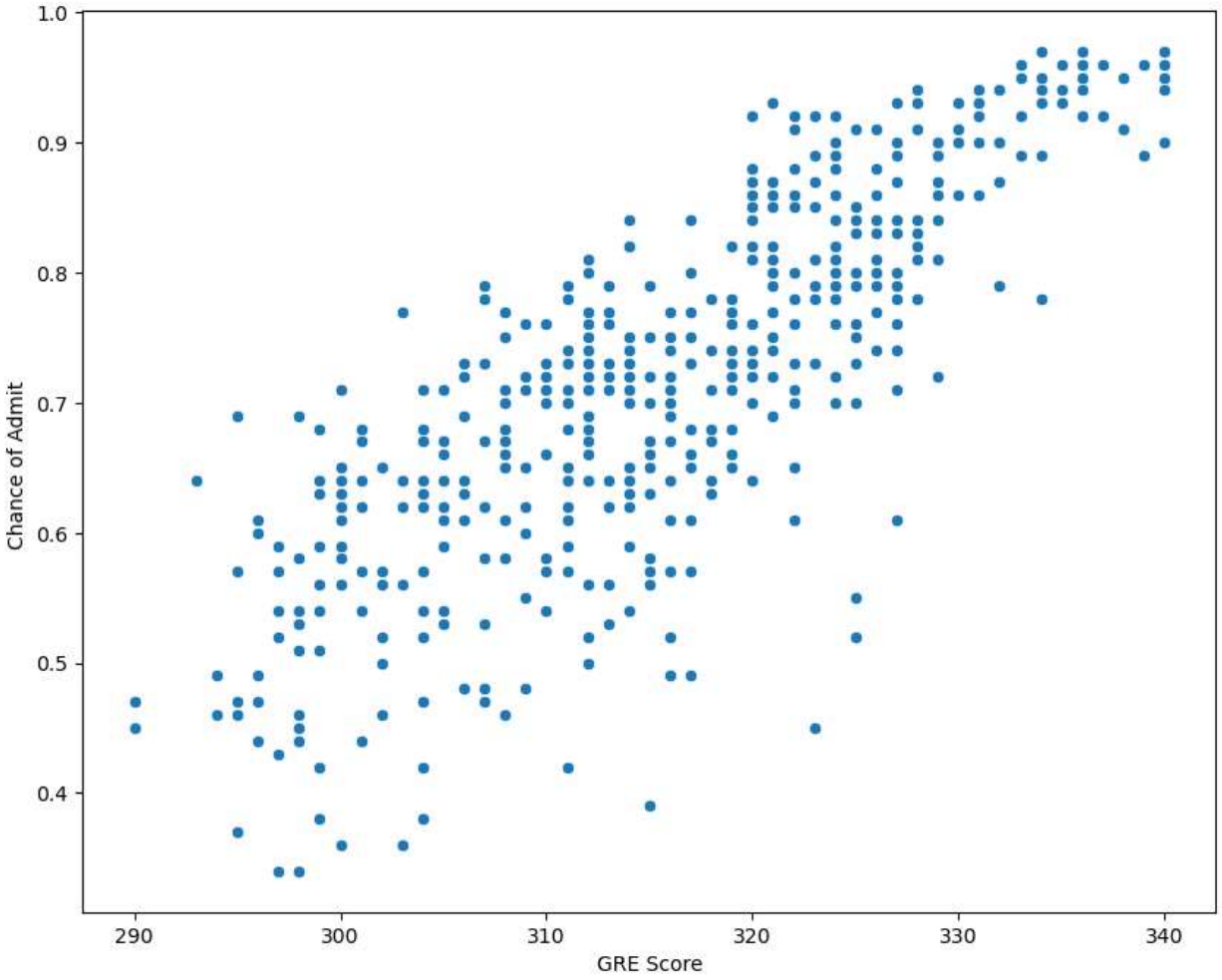
In [141…

```
plt.figure(figsize = (10,8))
sns.scatterplot(data = dt, x='TOEFL Score', y = 'Chance of Admit ')
plt.show()
```



In [141…

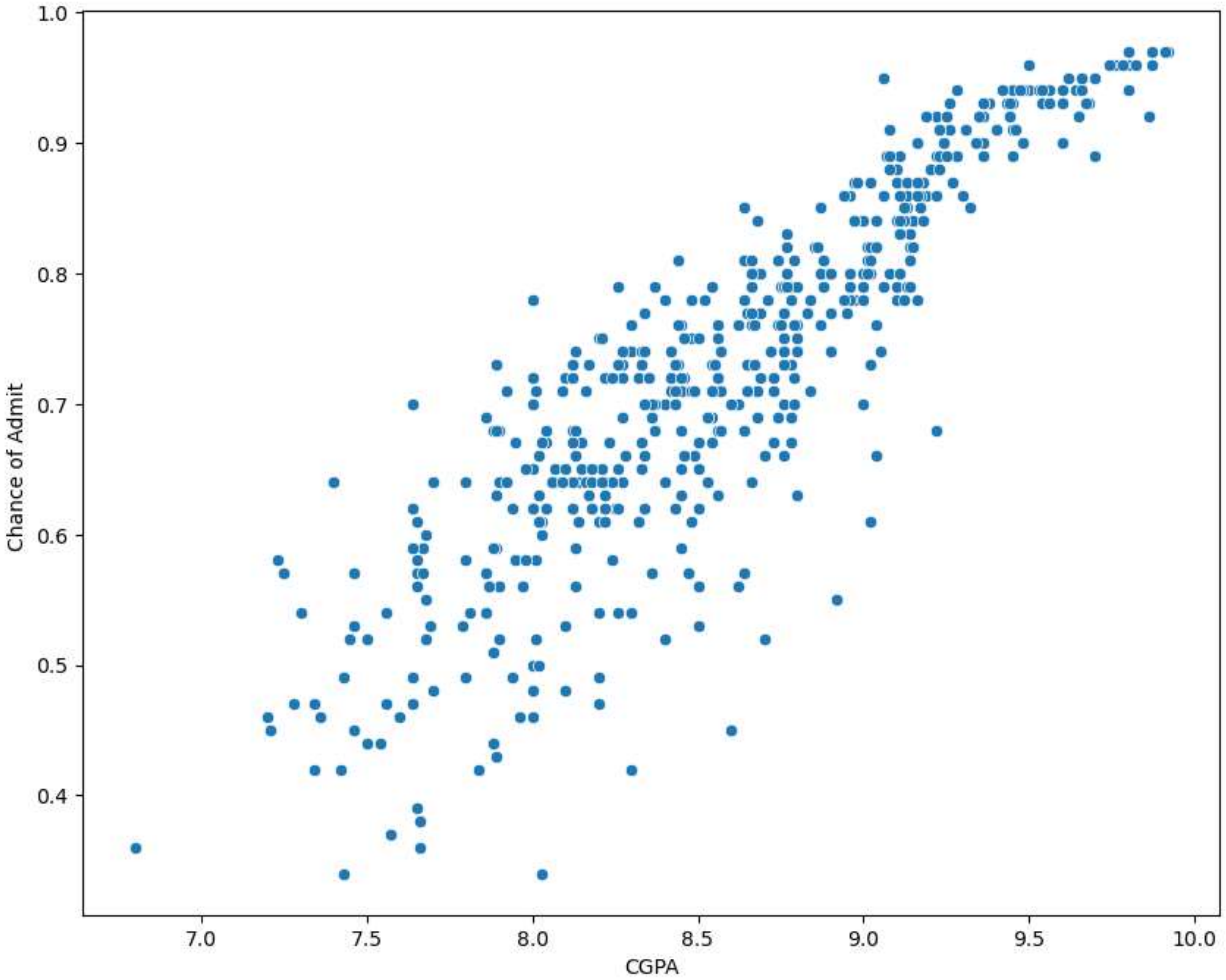
```
plt.figure(figsize = (10,8))
sns.scatterplot(data = dt, x='GRE Score', y = 'Chance of Admit ')
plt.show()
```



This scatter plot shows a nearly linear trend between the variables, indicating a positive coorelation with an upward trajectory

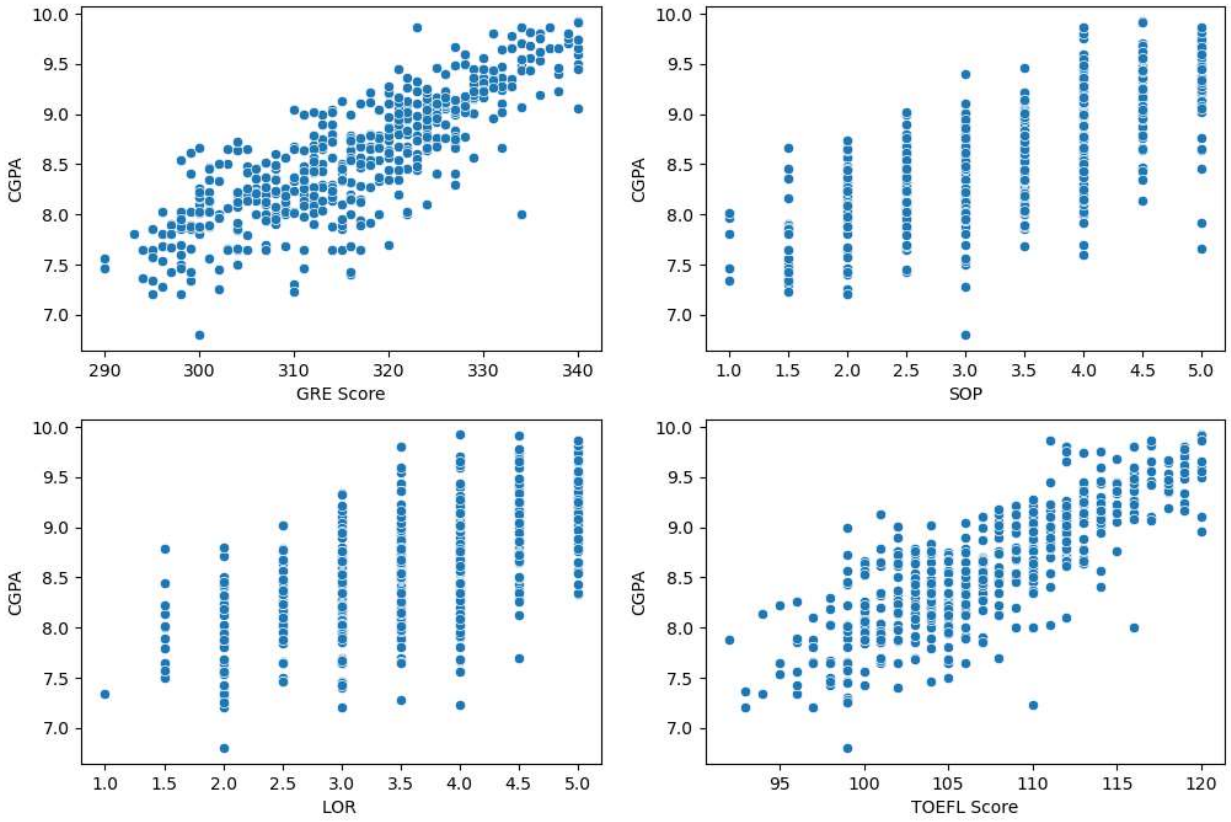
In [141…

```
plt.figure(figsize = (10,8))
sns.scatterplot(data = dt, x='CGPA', y = 'Chance of Admit ')
plt.show()
```



In [141…

```
plt.figure(figsize = (12,8))
plt.subplot(2,2,1)
sns.scatterplot(data = dt, x = 'GRE Score' , y = 'CGPA')
plt.subplot(2,2,2)
sns.scatterplot(data = dt, x = 'SOP' , y = 'CGPA')
plt.subplot(2,2,3)
sns.scatterplot(data = dt, x = 'LOR ' , y = 'CGPA')
plt.subplot(2,2,4)
sns.scatterplot(data = dt, x = 'TOEFL Score' , y = 'CGPA')
plt.show()
```

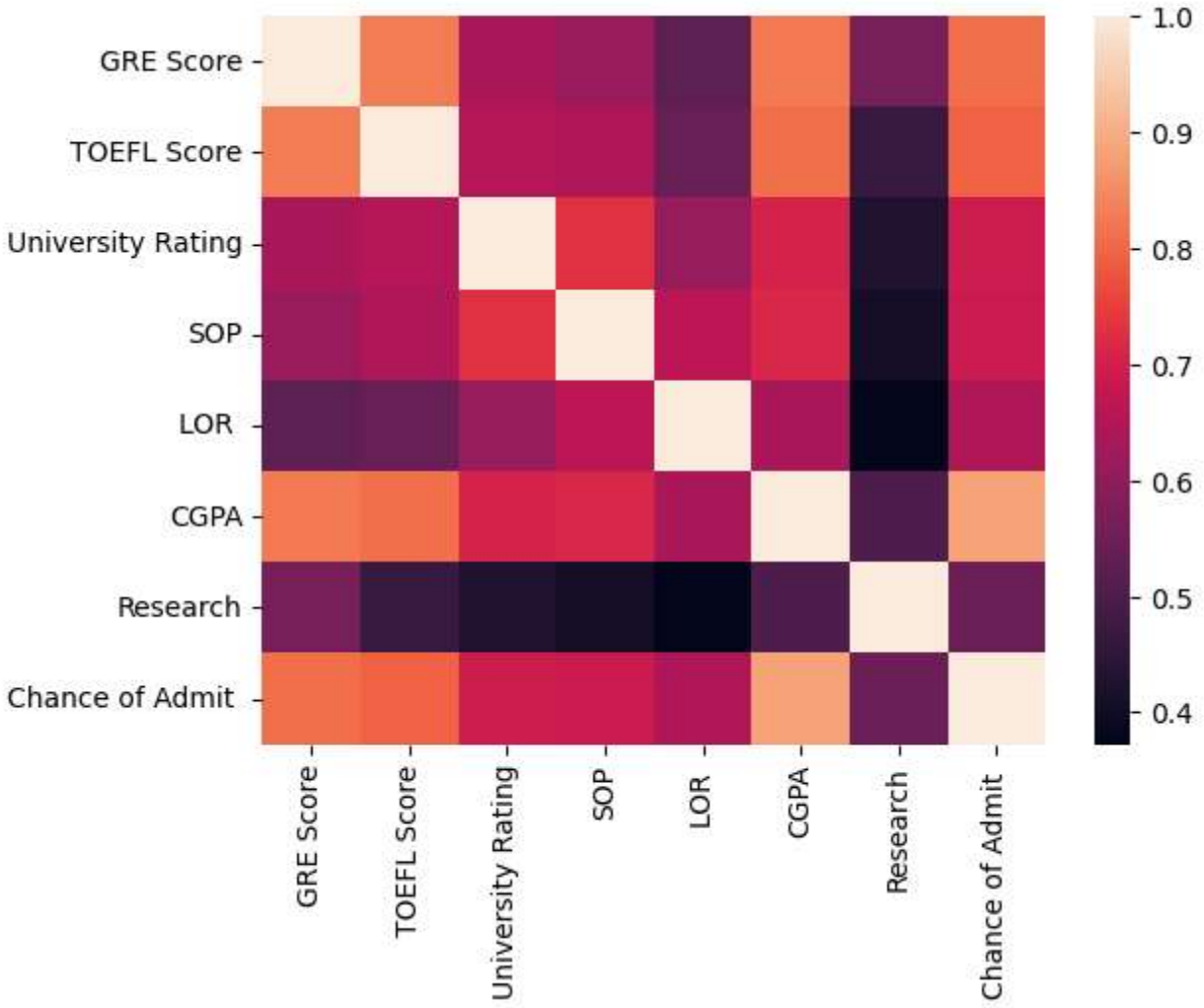


In [141…
```
dt.corr()
```

Out[1419]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
GRE Score	1.000000	0.827200	0.635376	0.613498	0.524679	0.825878	0.563398	0.810351
TOEFL Score	0.827200	1.000000	0.649799	0.644410	0.541563	0.810574	0.467012	0.792228
University Rating	0.635376	0.649799	1.000000	0.728024	0.608651	0.705254	0.427047	0.690132
SOP	0.613498	0.644410	0.728024	1.000000	0.663707	0.712154	0.408116	0.684137
LOR	0.524679	0.541563	0.608651	0.663707	1.000000	0.637469	0.372526	0.645365
CGPA	0.825878	0.810574	0.705254	0.712154	0.637469	1.000000	0.501311	0.882413
Research	0.563398	0.467012	0.427047	0.408116	0.372526	0.501311	1.000000	0.545871
Chance of Admit	0.810351	0.792228	0.690132	0.684137	0.645365	0.882413	0.545871	1.000000

In [142…
```
sns.heatmap(dt.corr())
plt.show()
```



In [142…
```
cols = dt[['GRE Score','TOEFL Score', 'Chance of Admit ']]
cols
```

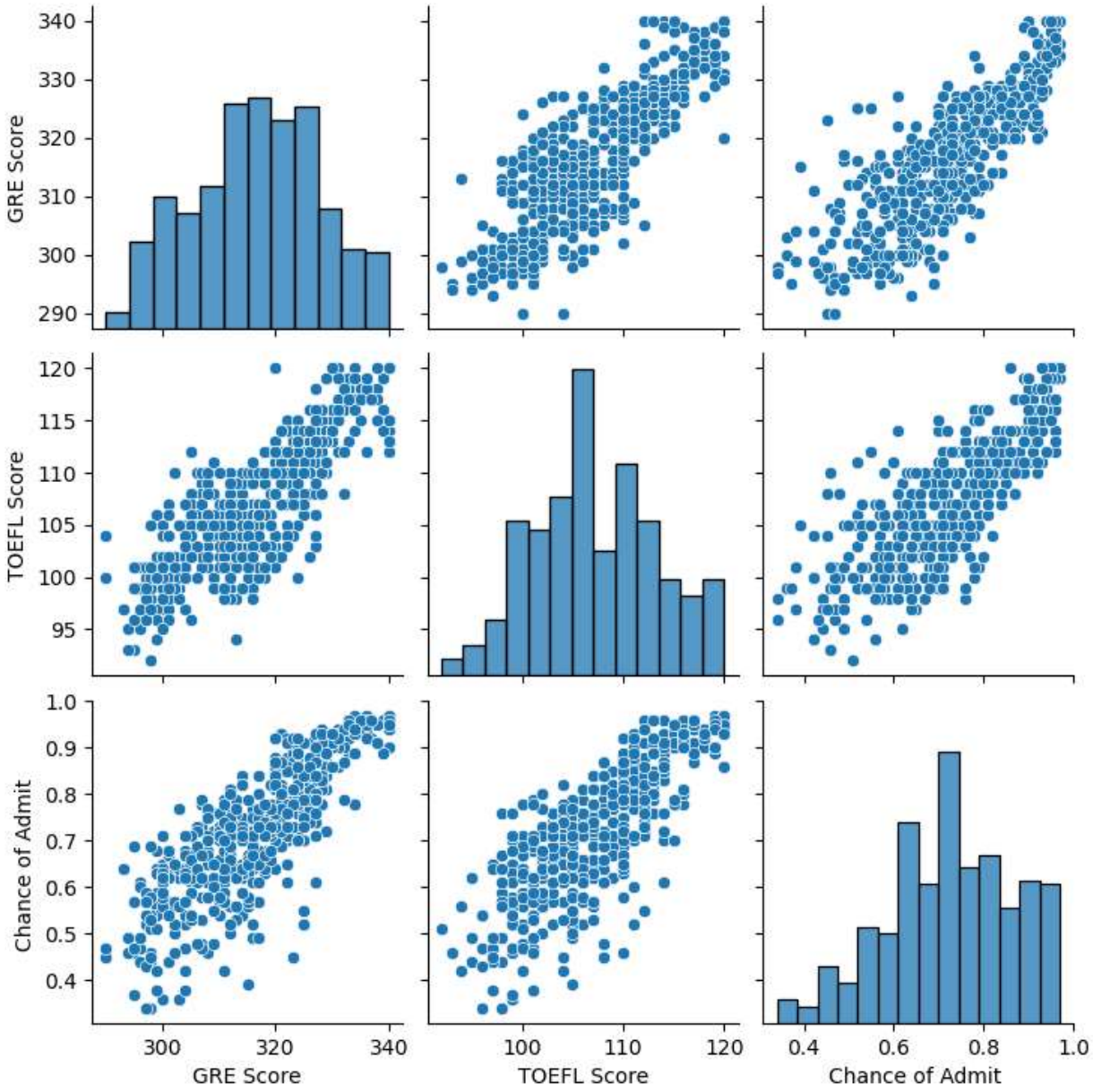
Out[1421]:

	GRE Score	TOEFL Score	Chance of Admit
0	337	118	0.92
1	324	107	0.76
2	316	104	0.72
3	322	110	0.80
4	314	103	0.65
...	...	...	...
495	332	108	0.87
496	337	117	0.96
497	330	120	0.93
498	312	103	0.73
499	327	113	0.84

500 rows × 3 columns

In [142…

```
sns.pairplot(cols)
plt.show()
```



Checking for Outliers

Outliers for GRE Score

In [142…

```
q1 = np.quantile(dt['GRE Score'], 0.25)
q3 = np.quantile(dt['GRE Score'], 0.75)

print("25 percentile is ", q1,". 75 percentile is ",q3)
IQR = q3 - q1

upper_whisker = q3 + (1.5*IQR)
lower_whisker = q1 - (1.5*IQR)

print(dt[(dt['GRE Score'] > upper_whisker) | (dt['GRE Score'] < lower_whisker)])
```

```
25 percentile is  308.0 . 75 percentile is  325.0
Empty DataFrame
Columns: [GRE Score, TOEFL Score, University Rating, SOP, LOR , CGPA, Research, Chanc
e of Admit ]
Index: []
```

Outliers for SOP

In [142…

```
q1 = np.quantile(dt['SOP'], 0.25)
q3 = np.quantile(dt['SOP'], 0.75)

print("25 percentile is ", q1,". 75 percentile is ",q3)
IQR = q3 - q1

upper_whisker = q3 + (1.5*IQR)
lower_whisker = q1 - (1.5*IQR)

print(dt[(dt['SOP'] > upper_whisker) | dt['SOP'] < lower_whisker])
```

```
25 percentile is  2.5 . 75 percentile is  4.0
Empty DataFrame
Columns: [GRE Score, TOEFL Score, University Rating, SOP, LOR , CGPA, Research, Chanc
e of Admit ]
Index: []
```

Outliers for TOEFL Score

In [142…

```
q1 = np.quantile(dt['TOEFL Score'], 0.25)
q3 = np.quantile(dt['TOEFL Score'], 0.75)

print("25 percentile is ", q1,". 75 percentile is ",q3)
IQR = q3 - q1

upper_whisker = q3 + (1.5*IQR)
lower_whisker = q1 - (1.5*IQR)

(dt[(dt['TOEFL Score'] > upper_whisker) | dt['TOEFL Score'] < lower_whisker])
```

```
25 percentile is  103.0 . 75 percentile is  112.0
```

Out[1425]:

	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
...	...	...	...	...	...	...	...	...
495	332	108	5	4.5	4.0	9.02	1	0.87
496	337	117	5	5.0	5.0	9.87	1	0.96
497	330	120	5	4.5	5.0	9.56	1	0.93
498	312	103	4	4.0	5.0	8.43	0	0.73
499	327	113	4	4.5	4.5	9.04	0	0.84

500 rows × 8 columns

Outliers for LOR

In [142...

```
q1 = np.quantile(dt['LOR '], 0.25)
q3 = np.quantile(dt['LOR '], 0.75)

print("25 percentile is ", q1,". 75 percentile is ",q3)
IQR = q3 - q1

upper_whisker = q3 + (1.5*IQR)
lower_whisker = q1 - (1.5*IQR)

print(dt[(dt['LOR '] > upper_whisker) | dt['LOR '] < lower_whisker])
```

```
25 percentile is  3.0 . 75 percentile is  4.0
     GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  Research  \
0          337          118                  4  4.5   4.5  9.65         1
1          324          107                  4  4.0   4.5  8.87         1
2          316          104                  3  3.0   3.5  8.00         1
3          322          110                  3  3.5   2.5  8.67         1
4          314          103                  2  2.0   3.0  8.21         0
..         ...          ...                ...  ...   ...   ...       ...
495        332          108                  5  4.5   4.0  9.02         1
496        337          117                  5  5.0   5.0  9.87         1
497        330          120                  5  4.5   5.0  9.56         1
498        312          103                  4  4.0   5.0  8.43         0
499        327          113                  4  4.5   4.5  9.04         0

     Chance of Admit
0                0.92
1                0.76
2                0.72
3                0.80
4                0.65
..                ...
495              0.87
496              0.96
497              0.93
498              0.73
499              0.84

[500 rows x 8 columns]
```

Outliers for CGPA

In [142…

```
q1 = np.quantile(dt['CGPA'], 0.25)
q3 = np.quantile(dt['CGPA'], 0.75)

print("25 percentile is ", q1,". 75 percentile is ",q3)
IQR = q3 - q1

upper_whisker = q3 + (0.5*IQR)
lower_whisker = q1 - (0.5*IQR)

print(dt[(dt['CGPA'] > upper_whisker) | (dt['CGPA'] < lower_whisker)])
```

```
25 percentile is  8.127500000000001 . 75 percentile is  9.04
     GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  Research  \
0          337          118                  4  4.5   4.5  9.65         1
22         328          116                  5  5.0   5.0  9.50         1
23         334          119                  5  5.0   4.5  9.70         1
24         336          119                  5  4.0   3.5  9.80         1
25         340          120                  5  4.5   4.5  9.60         1
..         ...          ...                ...  ...   ...   ...       ...
455        305          102                  2  1.5   2.5  7.64         0
457        295           99                  1  2.0   1.5  7.57         0
464        298           97                  2  2.0   3.0  7.21         0
496        337          117                  5  5.0   5.0  9.87         1
497        330          120                  5  4.5   5.0  9.56         1

     Chance of Admit
0                0.92
22               0.94
23               0.95
24               0.97
25               0.94
..                ...
455              0.59
457              0.37
464              0.45
496              0.96
497              0.93

[73 rows x 8 columns]
```

Outliers for Research

In [142…

```
q1 = np.quantile(dt['Research'], 0.25)
q3 = np.quantile(dt['Research'], 0.75)

print("25 percentile is ", q1,". 75 percentile is ",q3)
IQR = q3 - q1

upper_whisker = q3 + (0.5*IQR)
lower_whisker = q1 - (0.5*IQR)

print(dt[(dt['Research'] > upper_whisker) | dt['Research'] < lower_whisker])
```

```
25 percentile is  0.0 . 75 percentile is  1.0
Empty DataFrame
Columns: [GRE Score, TOEFL Score, University Rating, SOP, LOR , CGPA, Research, Chanc
e of Admit ]
Index: []
```

Outliers for Chance of Admit

In [142…

```
q1 = np.quantile(dt['Chance of Admit '], 0.25)
q3 = np.quantile(dt['Chance of Admit '], 0.75)

print("25 percentile is ", q1,". 75 percentile is ",q3)
IQR = q3 - q1

upper_whisker = q3 + (0.5*IQR)
lower_whisker = q1 - (0.5*IQR)
```

```
print(dt[(dt['Chance of Admit '] > upper_whisker) | dt['Chance of Admit '] < lower_whi
```

```
25 percentile is  0.63 . 75 percentile is  0.82
Empty DataFrame
Columns: [GRE Score, TOEFL Score, University Rating, SOP, LOR , CGPA, Research, Chanc
e of Admit ]
Index: []
```

Data Preparation for Modelling

In [ ]:

let us calculate taget variable for Linear Regression. Here ' Chance of Admit' serves as the dependent variable.

In [143…
```
dt.head()
```

Out[1430]:

	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 [143…
```
y = dt['Chance of Admit ']
y.head()
```

Out[1431]:
```
0    0.92
1    0.76
2    0.72
3    0.80
4    0.65
Name: Chance of Admit , dtype: float64
```

In [143…
```
x = dt.drop(columns = 'Chance of Admit ')
x.head()
```

Out[1432]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	337	118	4	4.5	4.5	9.65	1
1	324	107	4	4.0	4.5	8.87	1
2	316	104	3	3.0	3.5	8.00	1
3	322	110	3	3.5	2.5	8.67	1
4	314	103	2	2.0	3.0	8.21	0

In [143…
```
x.shape , y.shape
```

Out[1433]: ((500, 7), (500,))

In [143…
```
x_train , x_test , y_train, y_test = train_test_split(x ,y, test_size = 0.2)
```

In [143…
```
x_train.shape
```

Out[1435]: (400, 7)

In [143…
```
x_test.shape
```

Out[1436]: (100, 7)

In [143…
```
y_train.shape
```

Out[1437]: (400,)

In [143…
```
y_test.shape
```

Out[1438]: (100,)

In [143…
```
scaling = MinMaxScaler()
scaling
```

Out[1439]:
▾ MinMaxScaler
MinMaxScaler()

In [144…
```
x_train = pd.DataFrame(scaling.fit_transform(x_train), columns = x_train.columns)
x_train
```

Out[1440]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	0.68	0.750000	0.75	0.875	0.875	0.785256	1.0
1	0.56	0.285714	0.25	0.375	0.625	0.557692	1.0
2	0.38	0.571429	0.50	0.375	0.500	0.423077	0.0
3	0.62	0.678571	0.50	0.375	0.500	0.673077	1.0
4	0.52	0.285714	0.25	0.125	0.500	0.435897	1.0
…	…	…	…	…	…	…	…
395	0.12	0.107143	0.25	0.500	0.250	0.237179	1.0
396	0.22	0.250000	0.50	0.375	0.250	0.528846	1.0
397	0.24	0.357143	0.50	0.625	1.000	0.490385	0.0
398	0.20	0.285714	0.50	0.250	0.500	0.596154	1.0
399	0.50	0.285714	0.00	0.250	0.375	0.368590	0.0

400 rows × 7 columns

In [144…
```
x_test = pd.DataFrame(scaling.transform(x_test), columns = x_test.columns)
x_test
```

Out[1441]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	0.64	0.642857	0.50	0.625	0.500	0.692308	1.0
1	0.38	0.464286	1.00	0.625	0.625	0.564103	0.0
2	0.32	0.642857	0.25	0.625	0.750	0.528846	0.0
3	0.52	0.500000	0.25	0.375	0.750	0.487179	0.0
4	0.62	0.607143	0.50	0.625	0.625	0.641026	1.0
...	...	...	...	...	...	...	...
95	0.60	0.642857	1.00	0.750	0.750	0.791667	1.0
96	0.52	0.642857	0.50	0.625	0.750	0.564103	0.0
97	0.94	0.928571	0.75	0.875	0.875	0.913462	1.0
98	0.28	0.535714	0.50	0.625	0.500	0.339744	0.0
99	0.14	0.142857	0.25	0.375	0.125	0.349359	0.0

100 rows × 7 columns

Preparing the Linear Regression Model

In [144…
```
model = LinearRegression()
```

In [144…
```
model.fit(x_train,y_train)
model
```

Out[1443]:

▾ LinearRegression

LinearRegression()

Feature name with Coefficients

In [144…
```
pd.DataFrame({'Feature' : x_train.columns ,'Coefficients' : model.coef_})
```

Out[1444]:

	Feature	Coefficients
0	GRE Score	0.089281
1	TOEFL Score	0.085700
2	University Rating	0.020013
3	SOP	0.006033
4	LOR	0.070575
5	CGPA	0.370948
6	Research	0.022045

In [144…
```
model.intercept_
```

Out[1445]:
```
0.34650408453218423
```

Traiin and Test Performance Checking

score of the model/Performance

In [144…
```
r2 = model.score(x_train,y_train)
r2
```

Out[1446]:
```
0.813134682707573
```

In [144…
```
r2 = model.score(x_test,y_test)
r2
```

Out[1447]:
```
0.8538497398665569
```

In [144…
```
y_test[:10]
```

Out[1448]:
```
275    0.78
132    0.71
226    0.63
184    0.72
305    0.74
30     0.65
111    0.69
99     0.79
121    0.94
331    0.73
Name: Chance of Admit , dtype: float64
```

In [ ]:
```

```

In [144…
```
y_pred = model.predict(x_test)
y_pred[:10]
```

Out[1449]:
```
array([0.78665694, 0.69736656, 0.6880468 , 0.67669574, 0.77160951,
       0.59957795, 0.77192163, 0.80651112, 0.93543617, 0.62843957])
```

Performance with R2 score

In [145…
```
r2_score(y_test , y_pred)
```

Out[1450]: 0.8538497398665569

Checking for MSE and RMSE

MSE(Mean Square Error)

In [145…
```
np.square(y_train -  model.predict(x_train)).mean()
```

Out[1451]: 0.003804481507644584

MAE (Mean Absolute Error)

In [145…
```
np.abs(y_train - model.predict(x_train)).mean()
```

Out[1452]: 0.04450487347021328

RMSE(Root Mean Square Error)

In [145…
```
n = y_test.shape
n[0]
```

Out[1453]: 100

In [145…
```
np.sqrt(((np.sum(y_pred - y_test))**2)/n[0])
```

Out[1454]: 0.003443893159290745

Actual y_test value and Y prediction Comparision

In [145…
```
test1 = pd.DataFrame({'Actual' : y_test, 'Predicted' : y_pred})
test1
```

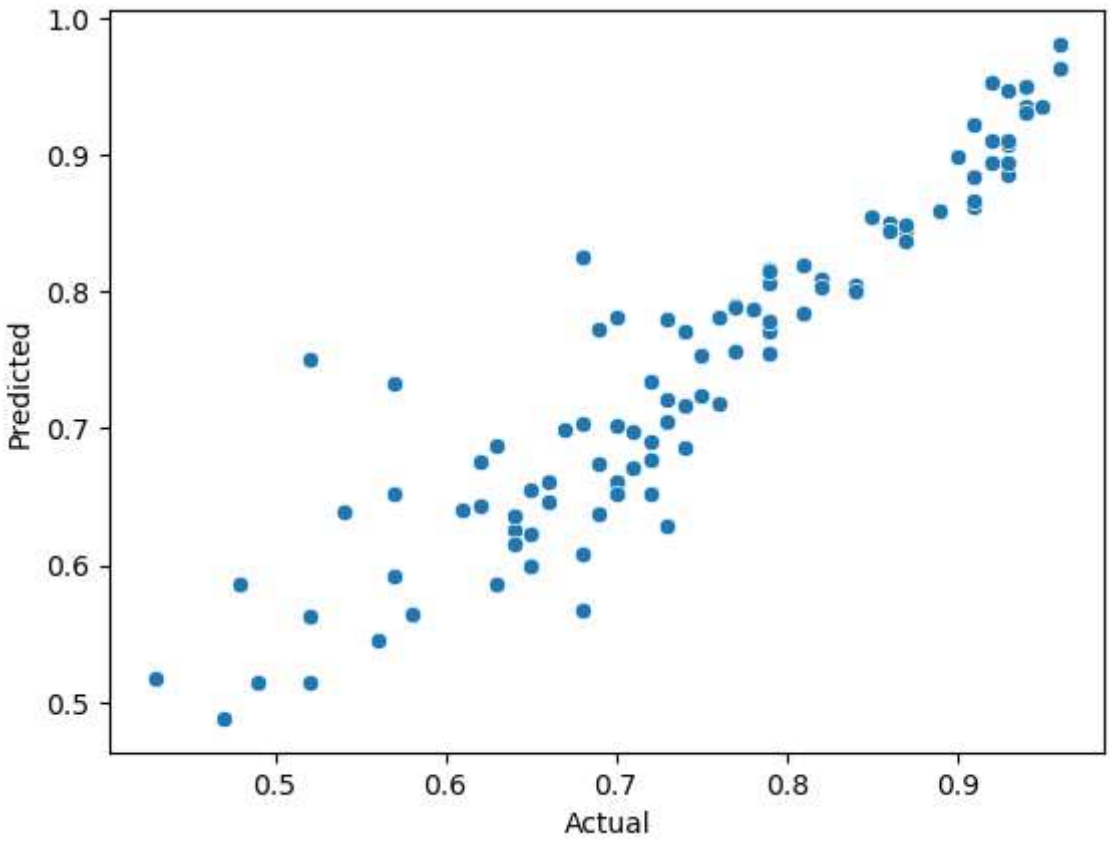
Out[1455]:

	Actual	Predicted
275	0.78	0.786657
132	0.71	0.697367
226	0.63	0.688047
184	0.72	0.676696
305	0.74	0.771610
...	...	...
470	0.87	0.848347
221	0.75	0.723985
0	0.92	0.952941
463	0.57	0.592506
329	0.43	0.516928

100 rows × 2 columns

In [145…

```
sns.scatterplot(test1, x = 'Actual' , y = 'Predicted')
plt.show()
```



This shows the trend between actual and predicted value and this reveals a clear linear pattern. Here the data points alogns themselves in linear trend

Normality of Residuals

Checking for the error

In [145…
```
Error = test1['Actual'] - test1['Predicted']
Error[:10]
```

Out[1457]:
```
275   -0.006657
132    0.012633
226   -0.058047
184    0.043304
305   -0.031610
30     0.050422
111   -0.081922
99    -0.016511
121    0.004564
331    0.101560
dtype: float64
```

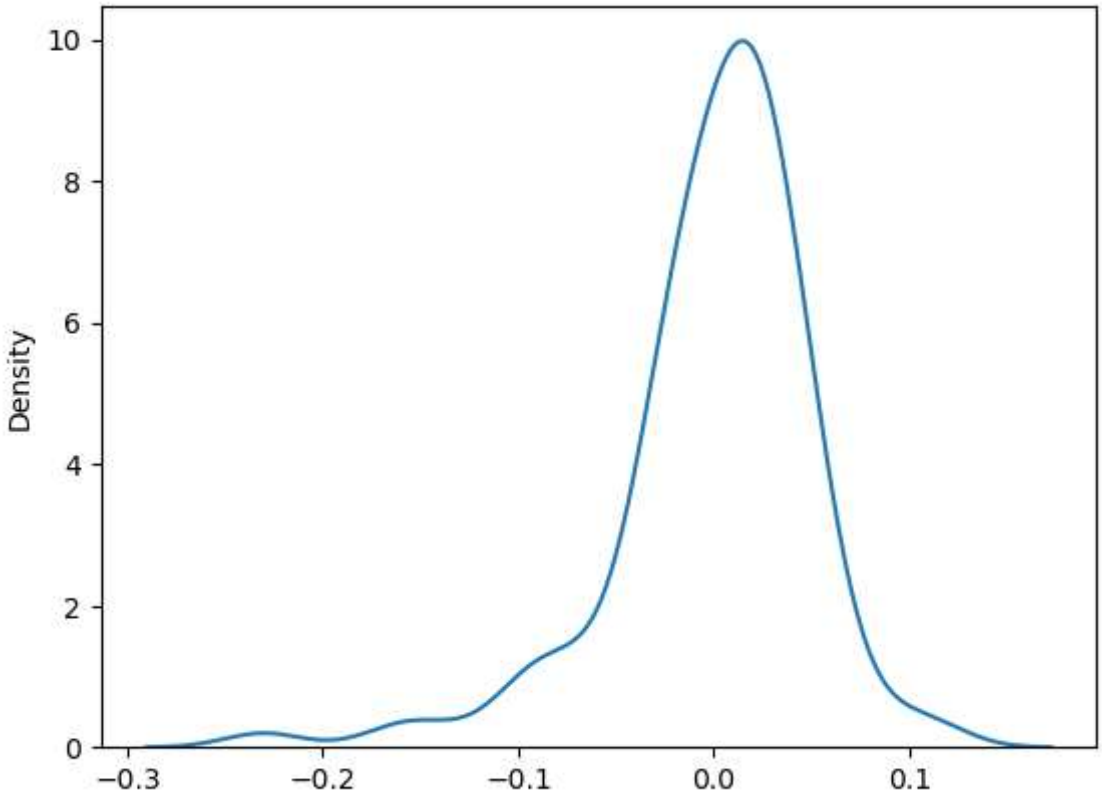
The mean of reesiduals is nearly zero

In [145…
```
Error.mean()
```

Out[1458]:
```
-0.0003443893159290745
```

We can see the difference between actual and predicted valuyes are almost zero which indiactes greaT favourable performances of residuals, contributing to the normality of data.

In [145…
```
sns.kdeplot(Error)
plt.show()
```

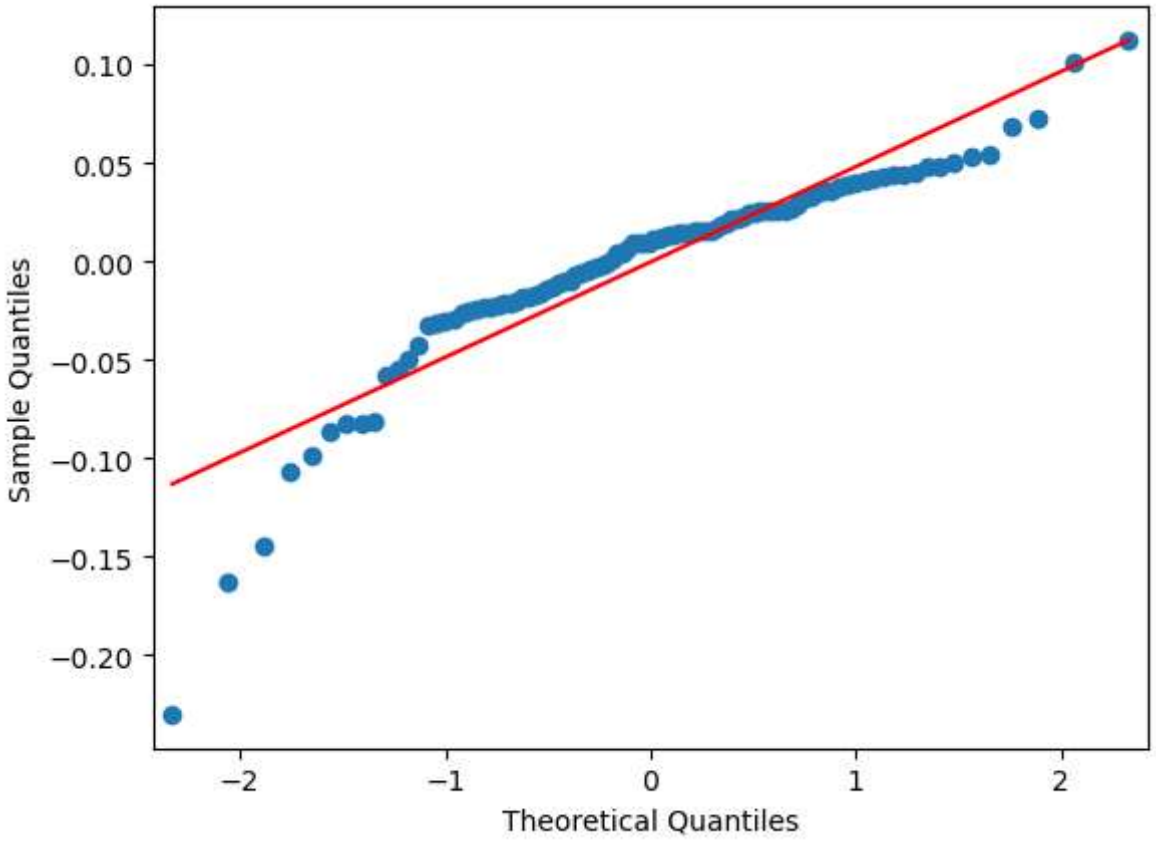


Distribution is somewhat skewed with mean close to zero. but the overall mean is centered around zero

Checking the QQ plot, for the regression of fit of line

In [146…

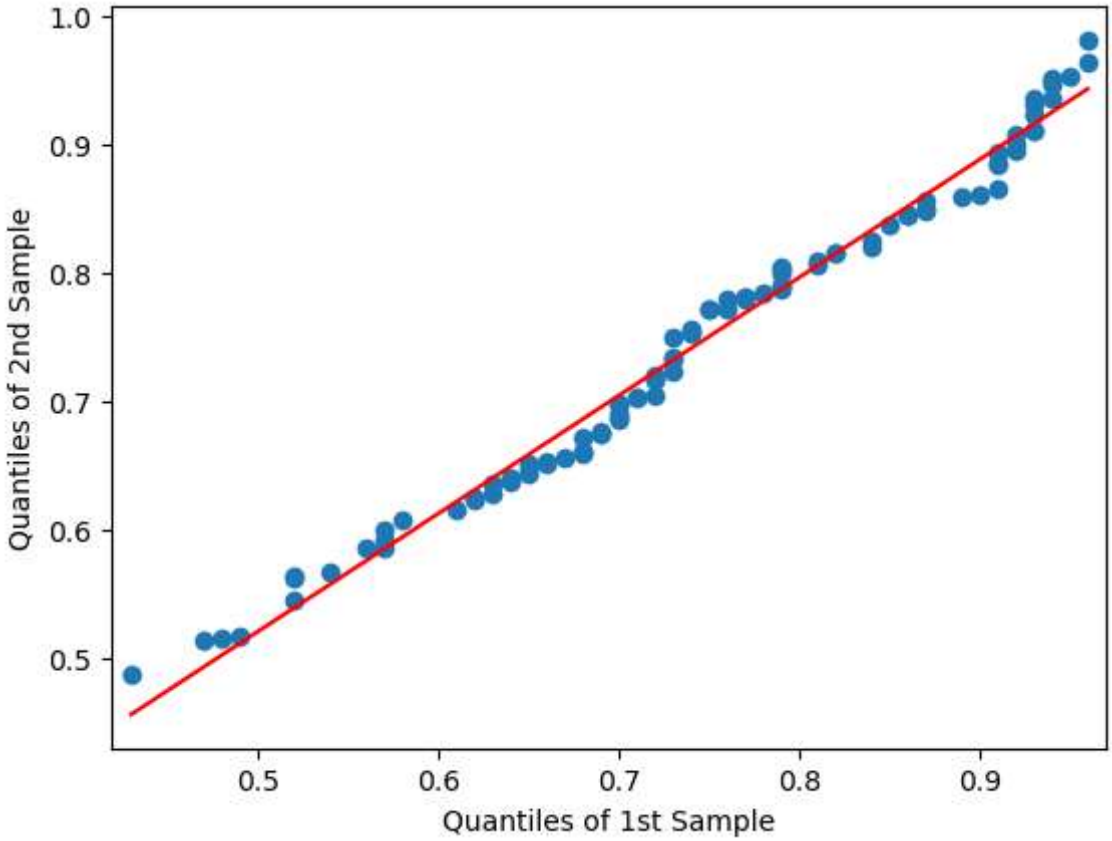
```
sm.qqplot(Error , line = "r")
plt.show()
```



QQ plot with actual and predicted line with best fit of regression

In [146…

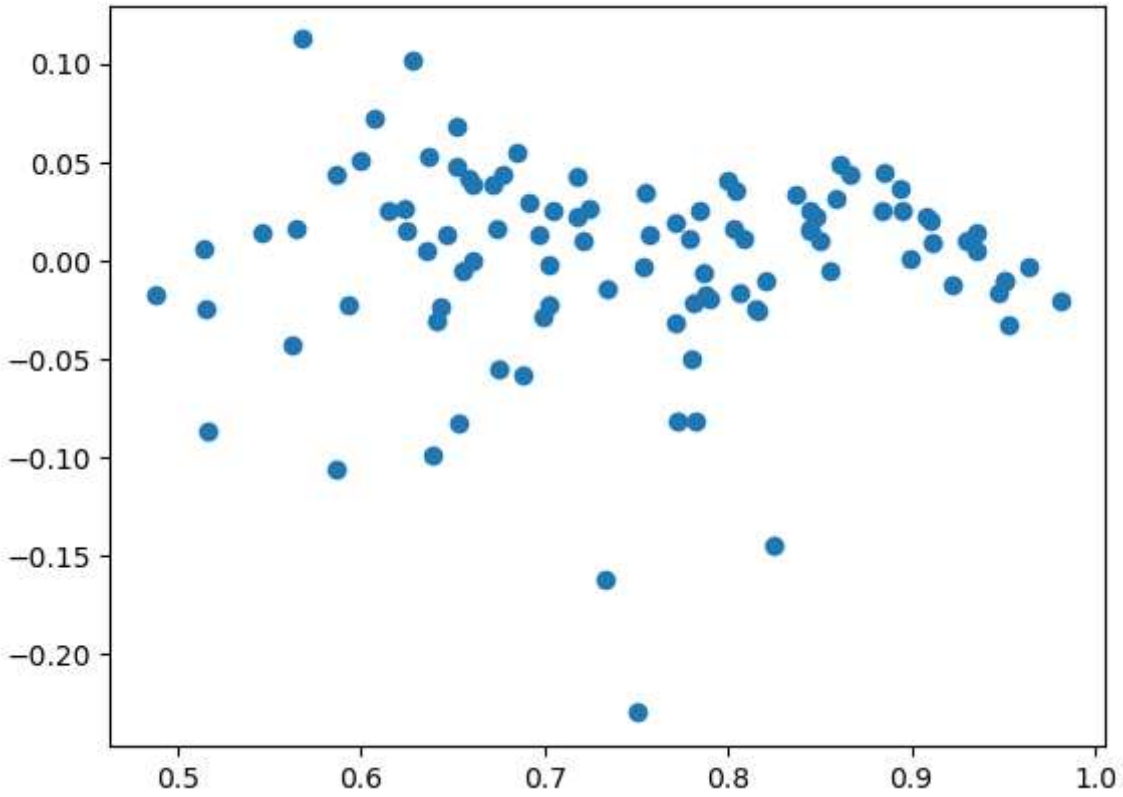
```
sm.qqplot_2samples(y_test,y_pred, line = 'r')
plt.show()
```



Check for Homoscedasticity

In [146…

```
plt.scatter( y_pred, Error)
plt.show()
```



The above graph shows dispersion between actual and predicted values indicating Homoscedasticity.

Adjusted R2 Score

Formula : 1- (1-r2)*(n-1)/(n-d-1) r2 - score n - no of Rows d - no of features

In [146…

```
n , d = x_train.shape
print("Number of Rows:", n,"Features", d)
Numerator = (1-r2)*(n-1)
Denominator = (n-d-1)
1-(Numerator/Denominator)
```

```
Number of Rows: 400 Features 7
```

Out[1463]:

```
0.8512399137927453
```

Multicollinearity check using VIF score

In [146…

```
x_multiCheck = pd.DataFrame(x_train , columns = x_train.columns)
vif = pd.DataFrame()
vif['Features'] = x_multiCheck.columns
vif['VIF'] = [variance_inflation_factor(x_multiCheck.values, i) for i in range(x_multi
vif['VIF'] = round(vif['VIF'],2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[1464]:

	Features	VIF
5	CGPA	41.16
0	GRE Score	27.29
1	TOEFL Score	27.29
3	SOP	18.31
4	LOR	14.78
2	University Rating	10.32
6	Research	3.23

CGPA column has higher VIF score compared to rest, which indiactes significant level of multicollinearity. Hence this column is eliminated to enhance predictive and accuracy

In [146…
```
x_multiCheck.drop(columns = {'CGPA'}, inplace = True)
```

In [146…
```
x_multiCheck.head()
```

Out[1466]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	Research
0	0.68	0.750000	0.75	0.875	0.875	1.0
1	0.56	0.285714	0.25	0.375	0.625	1.0
2	0.38	0.571429	0.50	0.375	0.500	0.0
3	0.62	0.678571	0.50	0.375	0.500	1.0
4	0.52	0.285714	0.25	0.125	0.500	1.0

In [146…
```
vif = pd.DataFrame()
vif['Features'] = x_multiCheck.columns
vif['VIF'] = [variance_inflation_factor(x_multiCheck.values, i) for i in range(x_multi
vif['VIF'] = round(vif['VIF'],2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[1467]:

	Features	VIF
1	TOEFL Score	23.90
0	GRE Score	22.39
3	SOP	17.26
4	LOR	13.08
2	University Rating	10.30
5	Research	3.23

Noe GRE Score shows high VIF Score indicating significant level of multicollinearity. Hence this feature will be eliminated

In [146…
```
x_multiCheck.drop(columns ={"GRE Score"}, inplace = True)
```

In [146…
```
vif = pd.DataFrame()
vif['Features'] = x_multiCheck.columns
vif['VIF'] = [variance_inflation_factor(x_multiCheck.values, i) for i in range(x_multi
vif['VIF'] = round(vif['VIF'],2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[1469]:

	Features	VIF
2	SOP	17.16
3	LOR	12.97
0	TOEFL Score	12.76
1	University Rating	10.26
4	Research	2.87

Now SOP feature shows high VIF Score indicating significant level of multicollinearity. Hence this feature will be eliminated

In [147…
```
x_multiCheck.drop(columns ={"SOP"}, inplace = True)
```

In [147…
```
vif = pd.DataFrame()
vif['Features'] = x_multiCheck.columns
vif['VIF'] = [variance_inflation_factor(x_multiCheck.values, i) for i in range(x_multi
vif['VIF'] = round(vif['VIF'],2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[1471]:

	Features	VIF
0	TOEFL Score	11.40
2	LOR	10.23
1	University Rating	8.59
3	Research	2.87

Now TOEFL Score feature shows high VIF Score indicating significant level of multicollinearity. Hence this feature will be eliminated

In [147…
```
x_multiCheck.drop(columns ={"TOEFL Score"}, inplace = True)
```

In [147…
```
vif = pd.DataFrame()
vif['Features'] = x_multiCheck.columns
vif['VIF'] = [variance_inflation_factor(x_multiCheck.values, i) for i in range(x_multi
vif['VIF'] = round(vif['VIF'],2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[1473]:

	Features	VIF
0	University Rating	7.11
1	LOR	7.03
2	Research	2.75

Now University Rating feature shows high VIF Score indicating significant level of multicollinearity. Hence this feature will be eliminated

In [147…
```
x_multiCheck.drop(columns ={"University Rating"}, inplace = True)
```

In [147…
```
vif = pd.DataFrame()
vif['Features'] = x_multiCheck.columns
vif['VIF'] = [variance_inflation_factor(x_multiCheck.values, i) for i in range(x_multi
vif['VIF'] = round(vif['VIF'],2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[1475]:

	Features	VIF
0	LOR	2.56
1	Research	2.56

Checking Adjusted R2 for No Multicollinearity or less than 5 VIF Score

In [147…
```
n,d = x_multiCheck.shape
print("Number of Rows:",n,"Features:",d)
num = (1-r2)*(n-1)
den = n-d-1
1 - (num/den)
```

```
Number of Rows: 400 Features: 2
```

Out[1476]:
```
0.8531134665157587
```

Previously the performance indicated by Adjusted R-squared was 83.39. After removal of featues based on VIF, several features were dropped and there was slight increase in performance at 83.6 in Adjusted R2 score

Analyze Stats summary for Regression Model using OLA

In [147…
```
x_sm = sm.add_constant(x_train)
```

In [147…
```
y_train1 = y_train.values
y_train1 - y_train1.reshape(-1,1)
```

```
y_train1.shape
```

Out[1478]:
```
(400,)
```

In [147…
```
model = sm.OLS(y_train1, x_sm)
result = model.fit()
print(result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.813
Model:                            OLS   Adj. R-squared:                  0.810
Method:                 Least Squares   F-statistic:                     243.7
Date:                Sat, 17 Feb 2024   Prob (F-statistic):          1.64e-138
Time:                        16:40:47   Log-Likelihood:                546.74
No. Observations:                 400   AIC:                            -1077.
Df Residuals:                     392   BIC:                            -1046.
Df Model:                           7
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
const               0.3465      0.010     33.305      0.000       0.326       0.367
GRE Score           0.0893      0.029      3.102      0.002       0.033       0.146
TOEFL Score         0.0857      0.028      3.015      0.003       0.030       0.142
University Rating   0.0200      0.017      1.148      0.252      -0.014       0.054
SOP                 0.0060      0.021      0.285      0.776      -0.036       0.048
LOR                 0.0706      0.019      3.671      0.000       0.033       0.108
CGPA                0.3709      0.036     10.307      0.000       0.300       0.442
Research            0.0220      0.008      2.873      0.004       0.007       0.037
==============================================================================
Omnibus:                       83.042   Durbin-Watson:                   2.044
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              171.969
Skew:                          -1.095   Prob(JB):                     4.54e-38
Kurtosis:                       5.350   Cond. No.                         23.5
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

Lets analyze with Polynomial features for better Performance

In [148…
```
dt.head()
```

Out[1480]:

	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 [148…
```
x_train.shape
```

Out[1481]:
```
(400, 7)
```

In [148…
```
x_test.shape
```

Out[1482]:
```
(100, 7)
```

In [148…
```
poly = PolynomialFeatures(degree = 2, include_bias = False)
poly
```

Out[1483]:
```
PolynomialFeatures
PolynomialFeatures(include_bias=False)
```

In [148…
```
model =LinearRegression()
model
```

Out[1484]:
```
LinearRegression
LinearRegression()
```

In [148…
```
x_train = poly.fit_transform(x_train)
x_train.shape
```

Out[1485]:
```
(400, 35)
```

In [148…
```
x_test = poly.transform(x_test)
x_test.shape
```

Out[1486]:
```
(100, 35)
```

In [148…
```
model.fit(x_train,y_train)
```

Out[1487]:
```
LinearRegression
LinearRegression()
```

In [148…
```
ypoly_pred = model.predict(x_test)
```

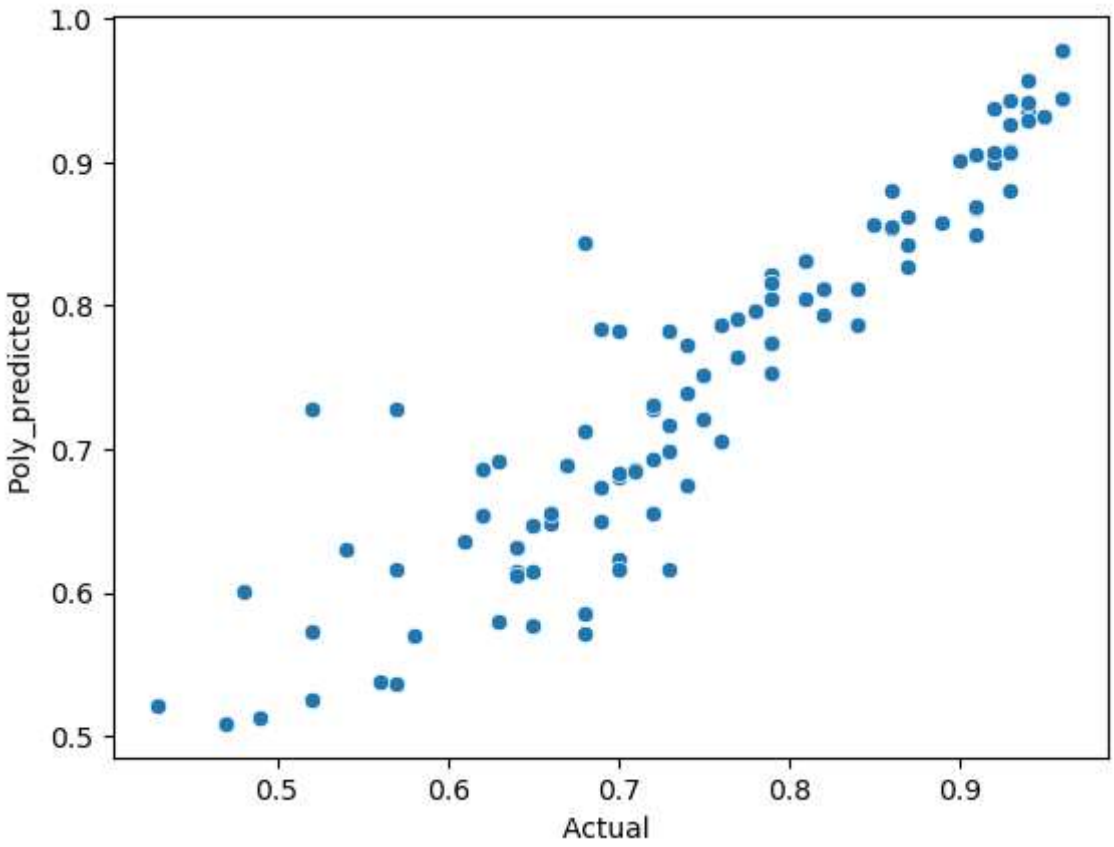
In [148…
```
polypredicted = pd.DataFrame({"Actual ": y_test,
                              "Poly_predicted " : model.predict(x_test)})

polypredicted[:10]
```

Out[1489]:

	Actual	Poly_predicted
275	0.78	0.795872
132	0.71	0.686678
226	0.63	0.691483
184	0.72	0.692443
305	0.74	0.772946
30	0.65	0.577209
111	0.69	0.783747
99	0.79	0.804909
121	0.94	0.934136
331	0.73	0.616857

In [149…

```
sns.scatterplot(data = polypredicted, x ="Actual ", y="Poly_predicted ")
plt.show()
```



Polynomial Prediction Performance

In [149…

```
model.score(x_test, y_test)
```

Out[1491]:

```
0.8429311507638635
```

Earlier Linear Regression score 78.7 was achieved.Upon incorporating Polynomial Regresion, a score of 79.4 was achieved. The difference between linear Regression and polynomial regression is negligible.

R2 score for Polynomial Regression

In [149…
```
r2_score(y_test, ypoly_pred)
```

Out[1492]:
```
0.8429311507638635
```

Adjusted R2 Score

In [149…
```
n,d = x_train.shape
print("Number of Rows:", n ,"Features:", d)
Num= (1-r2)*(n-1)
den = n-d-1
1-(num/den)
```

```
Number of Rows: 400 Features: 35
```

Out[1493]:
```
0.8397968302383412
```

The inclusion of polynomial features has resulted in decrease in Adjusted score. Wegot R2 score 78.3 which is now reduced to 76.6

Residuals

In [149…
```
poly_error = y_test - ypoly_pred
poly_error[:10]
```

Out[1494]:
```
275   -0.015872
132    0.023322
226   -0.061483
184    0.027557
305   -0.032946
30     0.072791
111   -0.093747
99    -0.014909
121    0.005864
331    0.113143
Name: Chance of Admit , dtype: float64
```

Checking for the mean of residuals

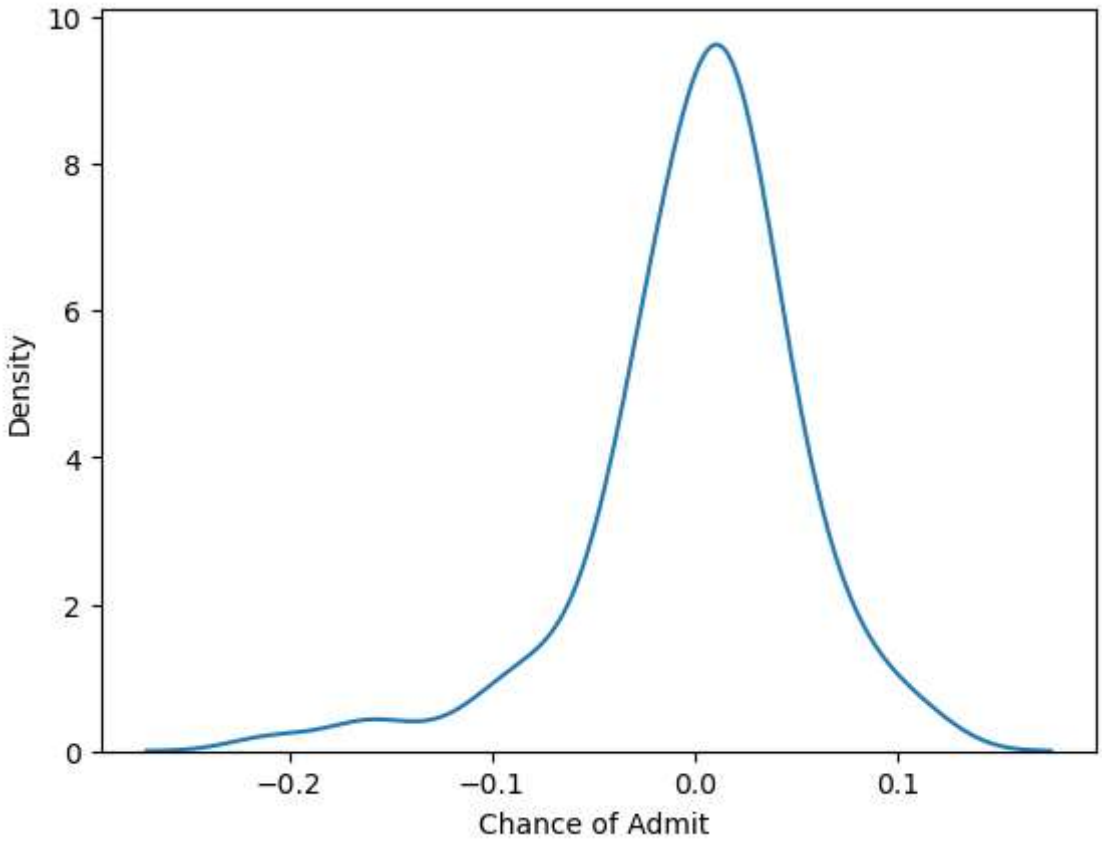
In [149…
```
np.mean(poly_error)
```

Out[1495]:
```
0.000612211934901879
```

Normality of residuals

In [149…

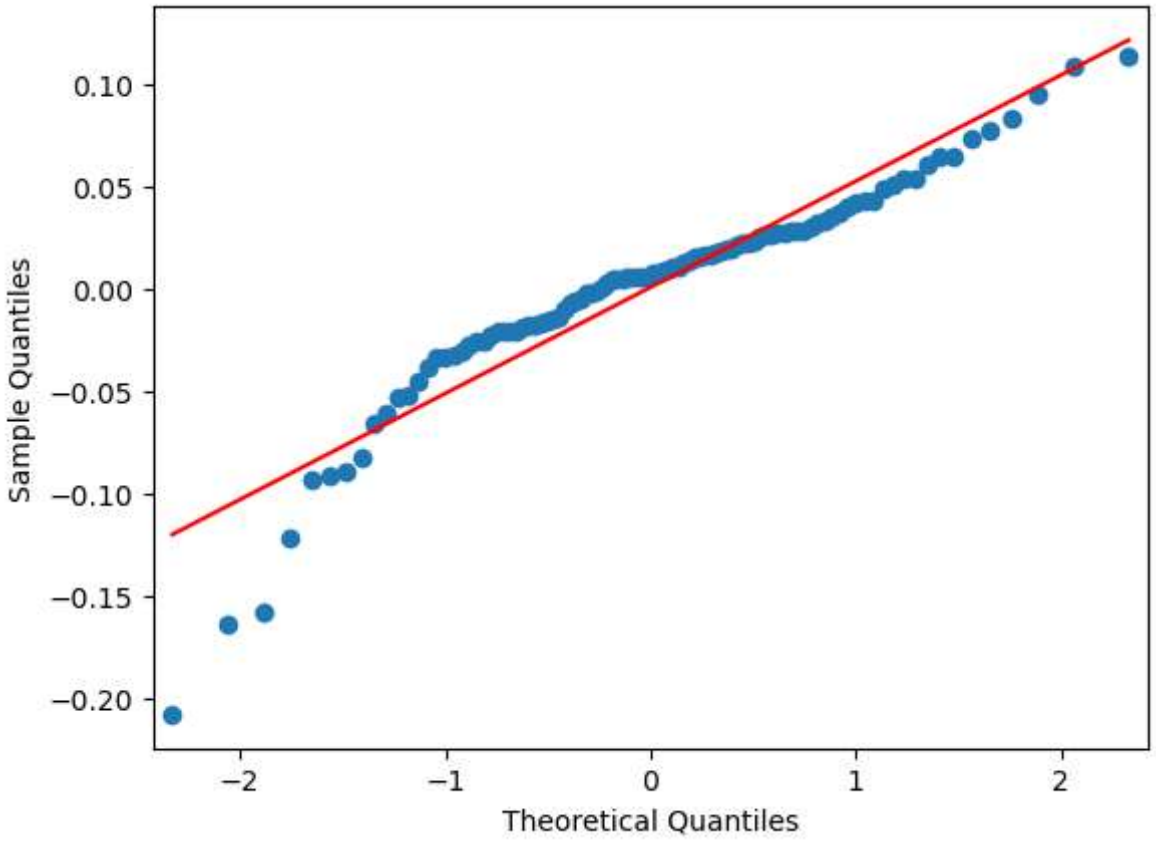
```
sns.kdeplot(poly_error)
plt.show()
```



This is a left skewed distribution, resembling Normal distribution with mean centered around zero. This suggests there are some students having lower scores affecting their chance of admission

In [149…

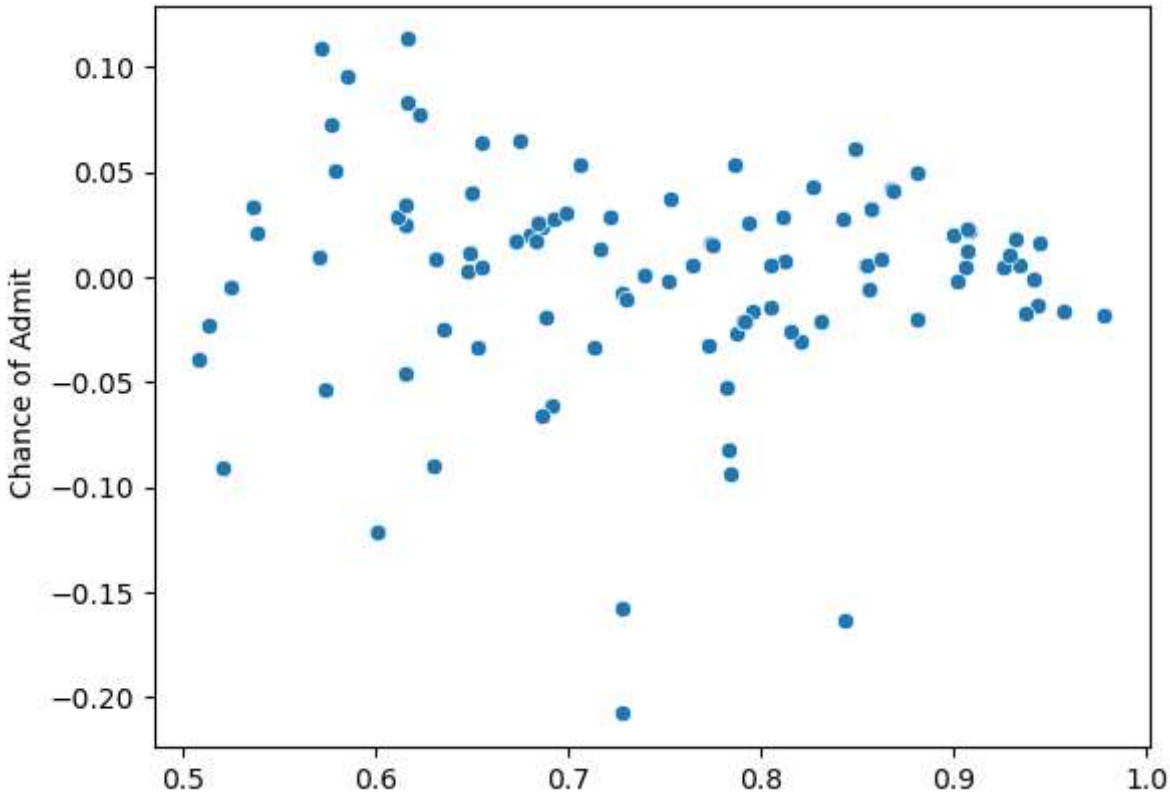
```
sm.qqplot(poly_error, line = 's')
plt.show()
```



Checking for Homoscedasticity

In [149…

```
sns.scatterplot(x=ypoly_pred , y=poly_error)
plt.show()
```



MSE

In [149…
```
mean_squared_error(y_train, model.predict(x_train))
```

Out[1499]:
```
0.003419945397580363
```

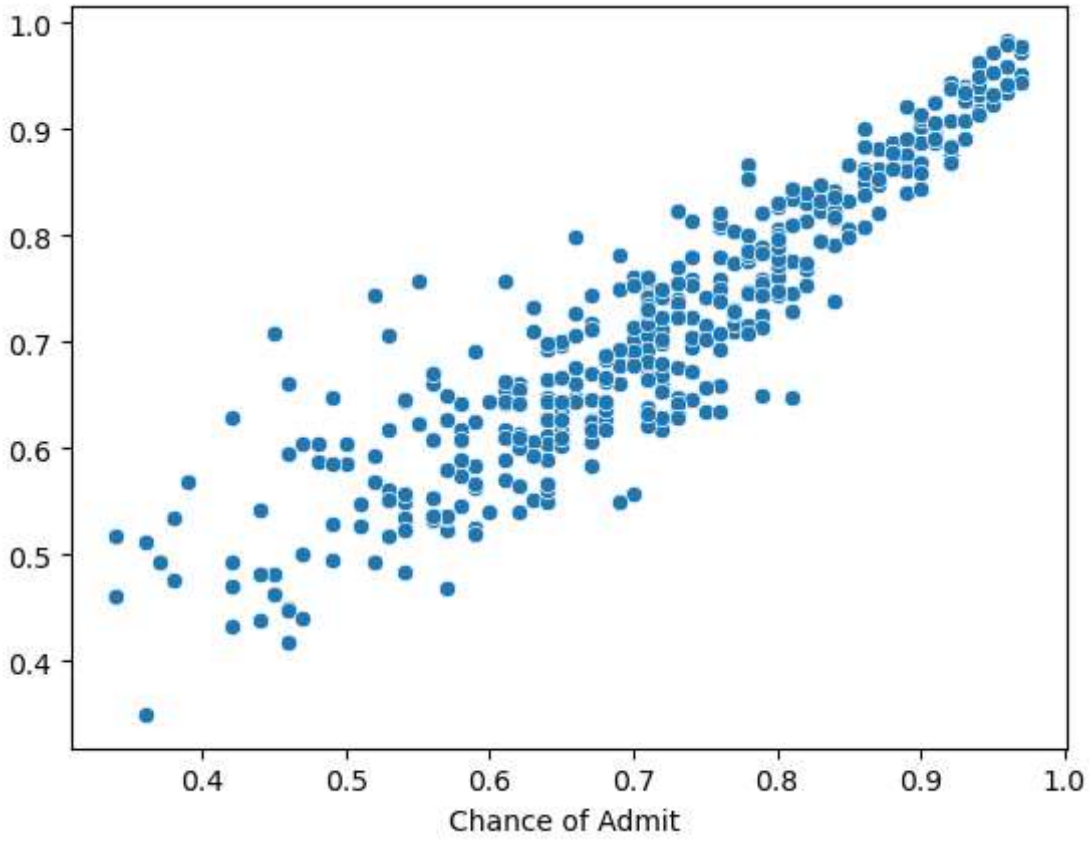
MAE

In [150…
```
mean_absolute_error(y_train ,  model.predict(x_train))
```

Out[1500]:
```
0.041626825862613875
```

RMSE

In [150…
```
sns.scatterplot(x=y_train , y=model.predict(x_train))
plt.show()
```



In [150…
```
n,d = x_test.shape
np.sqrt(((np.sum(model.predict(x_test)-y_test))**2)/n)
```

Out[1502]: 0.006612211934901879

In [150…
```
x_sm = sm.add_constant(x_train)
y_plytrain = y_train.values
y_polytrain = y_train1.reshape(-1,1)
y_polytrain.shape
```

Out[1503]: (400, 1)

In [150…
```
model = sm.OLS(y_polytrain, x_sm)
result = model.fit()
print(result.summary())
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.832
Model:	OLS	Adj. R-squared:	0.816
Method:	Least Squares	F-statistic:	53.17
Date:	Sat, 17 Feb 2024	Prob (F-statistic):	3.25e-120
Time:	16:40:48	Log-Likelihood:	568.05
No. Observations:	400	AIC:	-1066.
Df Residuals:	365	BIC:	-926.4
Df Model:	34		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.3250	0.027	11.944	0.000	0.272	0.379
x1	0.1009	0.092	1.093	0.275	-0.081	0.283
x2	0.1265	0.105	1.204	0.229	-0.080	0.333
x3	-0.0595	0.061	-0.977	0.329	-0.179	0.060
x4	0.0195	0.074	0.263	0.793	-0.126	0.165
x5	0.1147	0.080	1.439	0.151	-0.042	0.271
x6	0.4561	0.122	3.746	0.000	0.217	0.695
x7	-0.0141	0.015	-0.951	0.342	-0.043	0.015
x8	0.1010	0.198	0.511	0.610	-0.288	0.490
x9	-0.1491	0.288	-0.518	0.605	-0.716	0.417
x10	0.0319	0.158	0.202	0.840	-0.278	0.342
x11	-0.0422	0.210	-0.201	0.841	-0.455	0.371
x12	0.3422	0.195	1.759	0.079	-0.040	0.725
x13	-0.4441	0.315	-1.409	0.160	-1.064	0.176
x14	-0.0273	0.075	-0.365	0.715	-0.174	0.120
x15	-0.0285	0.181	-0.158	0.875	-0.384	0.327
x16	-0.0113	0.149	-0.076	0.940	-0.304	0.282
x17	0.3759	0.201	1.873	0.062	-0.019	0.770
x18	-0.2010	0.170	-1.181	0.238	-0.536	0.134
x19	-0.0598	0.335	-0.178	0.858	-0.719	0.599
x20	0.0311	0.074	0.421	0.674	-0.114	0.177
x21	-0.0233	0.066	-0.355	0.723	-0.153	0.106
x22	0.4733	0.119	3.962	0.000	0.238	0.708
x23	-0.1139	0.119	-0.960	0.338	-0.347	0.119
x24	-0.2429	0.198	-1.224	0.222	-0.633	0.147
x25	0.0314	0.044	0.713	0.476	-0.055	0.118
x26	-0.3163	0.112	-2.824	0.005	-0.537	-0.096
x27	0.1094	0.154	0.708	0.479	-0.194	0.413
x28	-0.1628	0.232	-0.702	0.483	-0.619	0.293
x29	-0.0373	0.051	-0.733	0.464	-0.137	0.063
x30	0.0797	0.094	0.844	0.399	-0.106	0.266
x31	-0.3790	0.248	-1.531	0.127	-0.866	0.108
x32	-0.0095	0.045	-0.210	0.834	-0.099	0.080
x33	0.5001	0.267	1.873	0.062	-0.025	1.025
x34	0.1138	0.083	1.368	0.172	-0.050	0.277
x35	-0.0141	0.015	-0.951	0.342	-0.043	0.015

Omnibus:	78.112	Durbin-Watson:	2.069
Prob(Omnibus):	0.000	Jarque-Bera (JB):	171.552
Skew:	-1.008	Prob(JB):	5.60e-38
Kurtosis:	5.496	Cond. No.	2.25e+15

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
[2] The smallest eigenvalue is 6.68e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Checking for the Regularisation as per both Lasso and Ridge Regression

Lasso Regression

1.Performs L1 regularization, i.e adds penalty equivalent to the absolute value of the magnitude of coefficients

2.Minimization objective = LS Obj + alpha * (sum of absolute value of coefficients)

In [150…
```
x_train.shape , x_test.shape
```

Out[1505]:
```
((400, 35), (100, 35))
```

In [150…
```
y_train.shape ,y_test.shape
```

Out[1506]:
```
((400,), (100,))
```

In [150…
```
Lassomodel = Lasso(alpha = 1)
Lassomodel
```

Out[1507]:
```
Lasso
Lasso(alpha=1)
```

In [150…
```
Lassomodel.fit(x_train, y_train)
```

Out[1508]:
```
Lasso
Lasso(alpha=1)
```

In [150…
```
Lassomodel.coef_[:10]
```

Out[1509]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [151…
```
Lassomodel.intercept_
```

Out[1510]:
```
0.715125
```

In [151…
```
y_test[:10]
```

Out[1511]:
```
275    0.78
132    0.71
226    0.63
184    0.72
305    0.74
30     0.65
111    0.69
99     0.79
121    0.94
331    0.73
Name: Chance of Admit , dtype: float64
```

In [151…
```
Lassomodel.predict(x_test)[:10]
```

Out[1512]:
```
array([0.715125, 0.715125, 0.715125, 0.715125, 0.715125, 0.715125,
       0.715125, 0.715125, 0.715125, 0.715125])
```

In [151…
```
Lassomodel.score(x_test, y_test)
```

Out[1513]:
```
-0.0640086002611937
```

In [151…
```
Lassomodel.score(x_train , y_train)
```

Out[1514]:
```
0.0
```

In [151…
```
LassoModelprediction = pd.DataFrame({
    "Actual" : y_test,
    "Lasso_predicted" : Lassomodel.predict(x_test)
})

LassoModelprediction[:10]
```

Out[1515]:

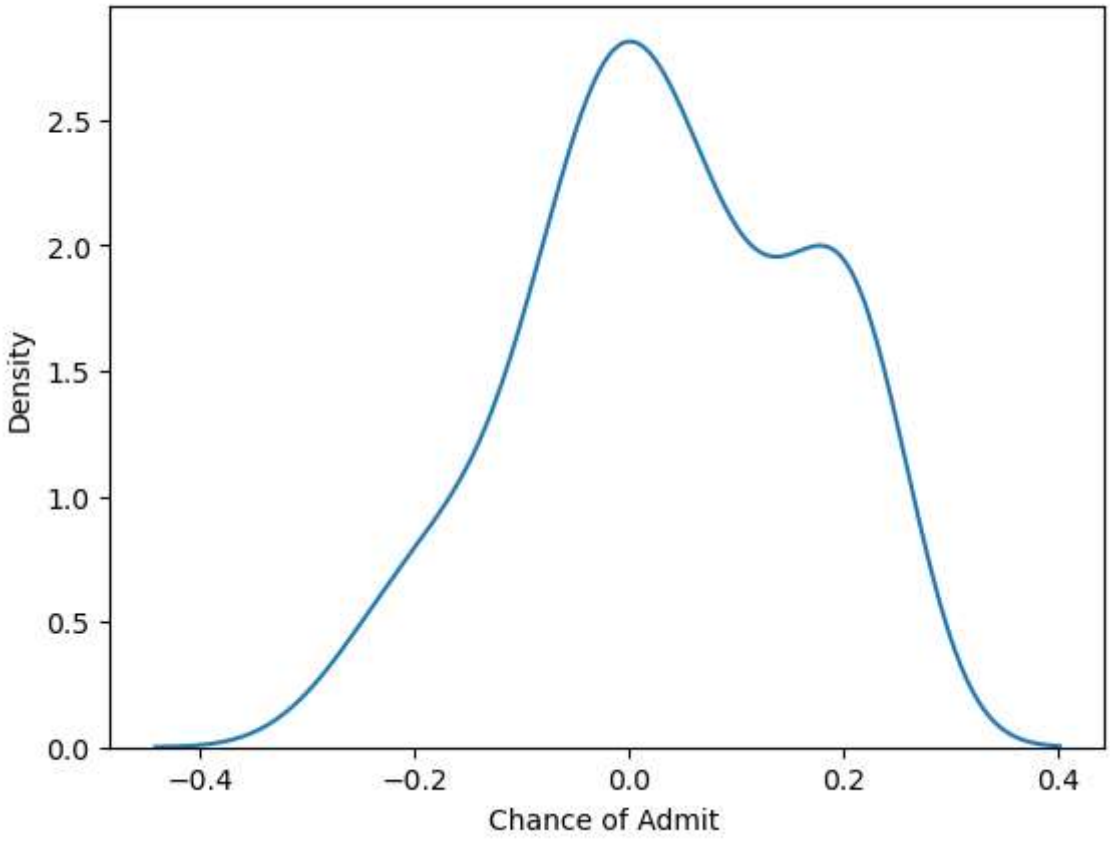
	Actual	Lasso_predicted
275	0.78	0.715125
132	0.71	0.715125
226	0.63	0.715125
184	0.72	0.715125
305	0.74	0.715125
30	0.65	0.715125
111	0.69	0.715125
99	0.79	0.715125
121	0.94	0.715125
331	0.73	0.715125

Residuals

In [151…
```
LassoError = y_test - Lassomodel.predict(x_test)
LassoError
```

Out[1516]:
```
275    0.064875
132   -0.005125
226   -0.085125
184    0.004875
305    0.024875
         ...
470    0.154875
221    0.034875
0      0.204875
463   -0.145125
329   -0.285125
Name: Chance of Admit , Length: 100, dtype: float64
```

In [151…
```
sns.kdeplot(LassoError)
plt.show()
```



From the KDE plot we can say, the normality of residuals deteriotes when using Lasso Regression Model. This indicates an increase in error and instability in the predictions, leading to less reliable outcomes.

Ridge Regression

1. Performs L2 regularization Adds penalty equivalent to the sqaure of the magnitude of coefficients.
2. Minimization objective = LS Obj + alpha*(su of square of coefficients)

In [151…
```
RidgeModel = Ridge(alpha=10)
RidgeModel
```

Out[1518]:
```
Ridge(alpha=10)
```

In [151…
```
RidgeModel.fit(x_train, y_train)
```

Out[1519]:
```
Ridge(alpha=10)
```

In [152…
```
RidgeModel.predict(x_test).round(3)[:10]
```

Out[1520]:
```
array([0.765, 0.681, 0.675, 0.663, 0.757, 0.603, 0.773, 0.801, 0.953,
       0.633])
```

In [152…
```
RidgeModelPrediction = pd.DataFrame({
    "Actual" : y_test,
    "Ridge_predicted" : RidgeModel.predict(x_test)
})

RidgeModelPrediction[:10]
```

Out[1521]:

	Actual	Ridge_predicted
275	0.78	0.765373
132	0.71	0.681277
226	0.63	0.675129
184	0.72	0.663222
305	0.74	0.756828
30	0.65	0.603189
111	0.69	0.772500
99	0.79	0.801137
121	0.94	0.952639
331	0.73	0.633204

In [ ]:
```
```

In [152…
```
RidgeModel.score(x_train , y_train)
```

Out[1522]:
```
0.790957280128402
```

In [152…
```
RidgeModel.score(x_test, y_test)
```

Out[1523]:
```
0.841058970869387
```

Residuals

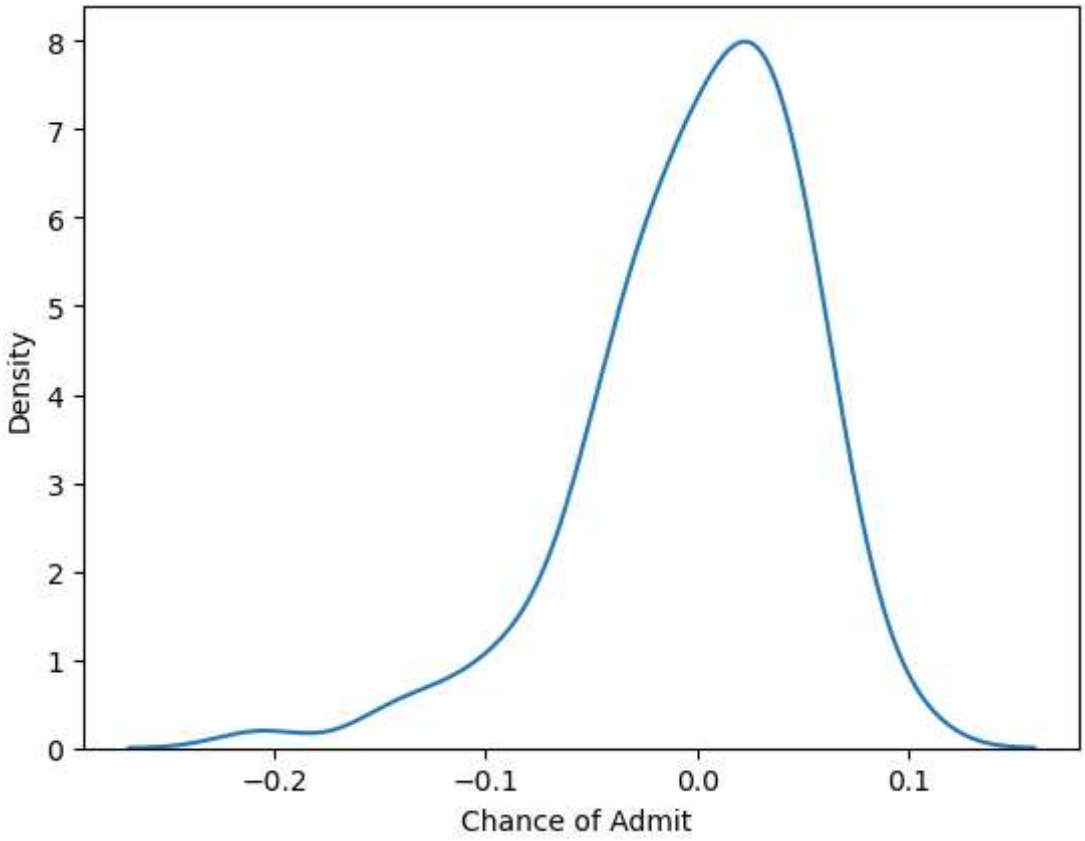
In [152…
```
RidgeError = y_test - RidgeModel.predict(x_test)
RidgeError
```

Out[1524]:
```
275    0.014627
132    0.028723
226   -0.045129
184    0.056778
305   -0.016828
         ...
470    0.036505
221    0.039813
0     -0.043184
463   -0.043371
329   -0.110660
Name: Chance of Admit , Length: 100, dtype: float64
```

Normality of Residuals

In [152…
```
sns.kdeplot(RidgeError)
```

Out[1525]:
```
<Axes: xlabel='Chance of Admit ', ylabel='Density'>
```

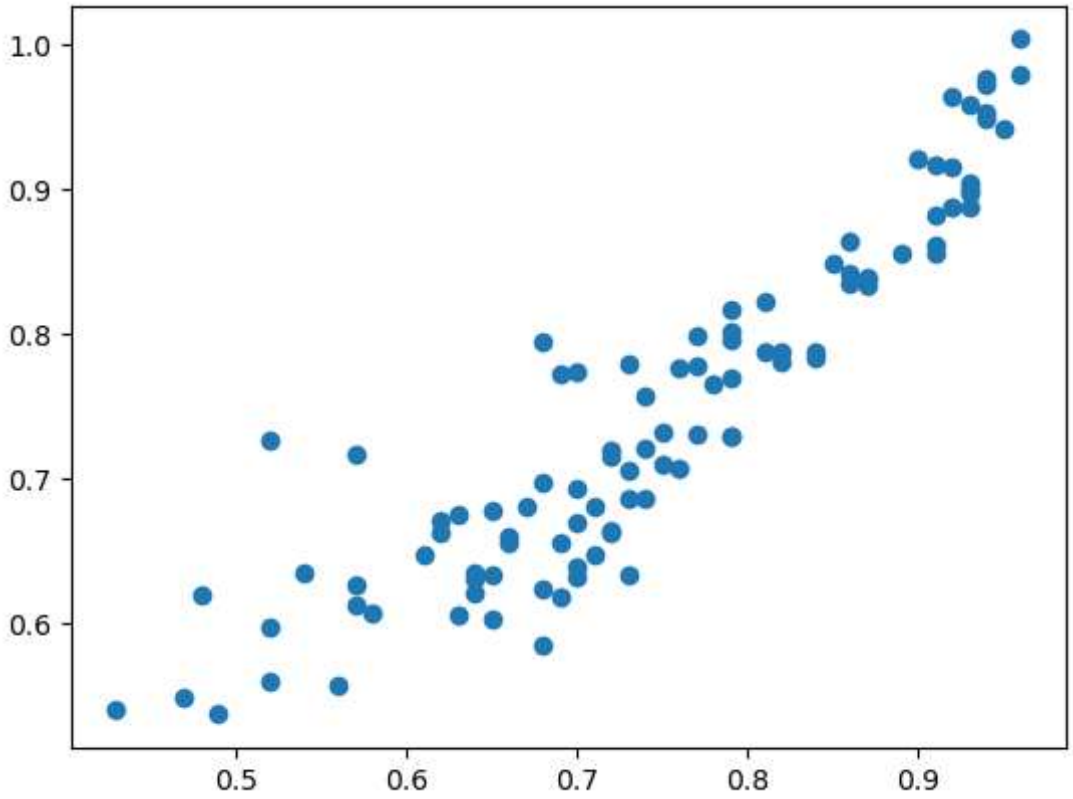


From above analysis its evident that scores obtained for both training and test datasets are slightly lower compared to those achieved using the Linear regression algorithm. KDE plot indicates a slight shift towards the right, suggesting that the mean does not precisely align with

zero. This observation implies that error has a discernible impact, particularly in the context of Ridge regression.

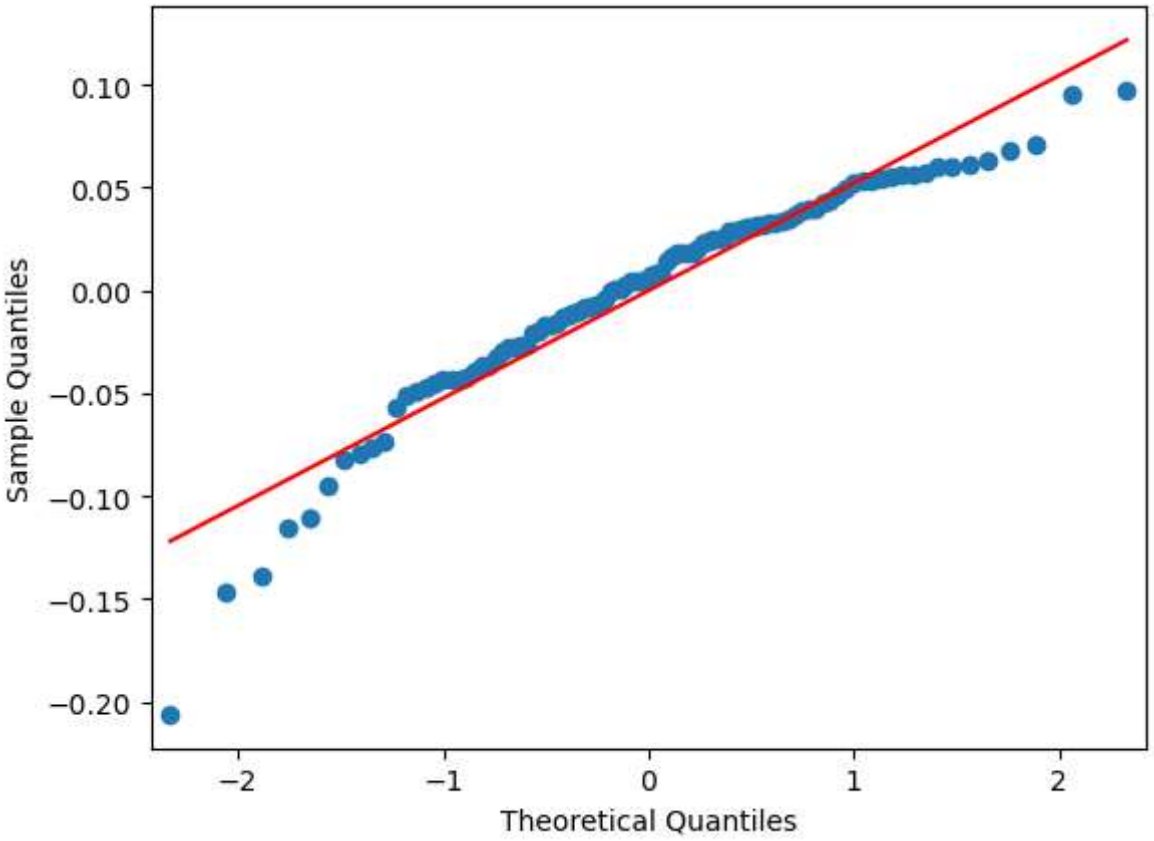
In [152…

```
plt.scatter(x = y_test, y = RidgeModel.predict(x_test))
plt.show()
```



In [152…

```
sm.qqplot(RidgeError, line = "r")
plt.show()
```



we exploered regularization methods such as Lasso and Ridge to potentially enhance the model's predictive accuracy. Despite our efforts, these approaches did not yield much satisfactory results in improving performance. Thus, next step is to go with cross validation method to further enhance model's performance.

Cross Validation

Cross Validation to check performance

In [152…
```
dt1 = dt
dt1.head()
```

Out[1528]:

	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 [152…
```
dt1.shape
```

Out[1529]:
```
(500, 8)
```

In [153…
```
x.shape , y.shape
```

Out[1530]:
```
((500, 7), (500,))
```

In [153…
```
x_train_cv , x_test , y_train_cv , y_test = train_test_split(x,y,test_size = 0.2)
x_train , x_val , y_train , y_val = train_test_split(x_train_cv , y_train_cv, test_siz
```

In [153…
```
print(x_train.shape , y_train.shape)
print(x_val.shape , y_val.shape)
print(x_test.shape , y_test.shape)
```

```
(320, 7) (320,)
(80, 7) (80,)
(100, 7) (100,)
```

In [153…
```
ScalingCV = StandardScaler()
ScalingCV
```

Out[1533]:
```
StandardScaler
StandardScaler()
```

In [153…
```
Xtraincv = ScalingCV.fit_transform(x_train)
Xvalcv = ScalingCV.fit_transform(x_val)
xtestcv = ScalingCV.transform(x_test)
```

In [153…
```
Modelcv = LinearRegression()
Modelcv
```

Out[1535]:
```
LinearRegression
LinearRegression()
```

In [153…
```
Modelcv.fit(Xtraincv , y_train)
```

Out[1536]:
```
LinearRegression
LinearRegression()
```

In [153…
```
Modelcv.coef_
```

Out[1537]:
```
array([0.01133999, 0.01892261, 0.00941541, 0.00490624, 0.01807812,
       0.06744476, 0.01520318])
```

In [153…
```
Modelcv.score(Xtraincv , y_train)
```

Out[1538]:
```
0.8282812324115261
```

In [153…
```
y_predcv = Modelcv.predict(Xvalcv)
y_predcv[:10]
```

Out[1539]:
```
array([0.76723111, 0.86502494, 0.73944216, 0.47599619, 0.7932149 ,
       0.69457135, 0.86790549, 0.56655579, 0.69789627, 0.69124148])
```

In [154…
```
y_val[:10]
```

Out[1540]:
```
299    0.71
99     0.79
226    0.63
79     0.46
263    0.70
465    0.54
140    0.84
205    0.57
54     0.70
181    0.71
Name: Chance of Admit , dtype: float64
```

In [154…
```
pd.DataFrame({
    "Actual" : y_val,
    "predicted" : y_predcv

})[:10]
```

Out[1541]:

	Actual	predicted
299	0.71	0.767231
99	0.79	0.865025
226	0.63	0.739442
79	0.46	0.475996
263	0.70	0.793215
465	0.54	0.694571
140	0.84	0.867905
205	0.57	0.566556
54	0.70	0.697896
181	0.71	0.691241

In [154…
```
Modelcv.score(Xvalcv, y_val)
```

Out[1542]:
```
0.569518118128246
```

In [ ]:
```

```

In [154…
```
Modelcv.score(xtestcv , y_test)
```

Out[1543]:
```
0.6667103779108859
```

Using cross validation method,we partitioned the dataset into traing, testing and validation subsets to enhance predictive accuracy. We say our model achieved 84.3% performance on validation data. However upon evaluating the model on testing data, the performance slightly declined to 75% indicating slight deviation from the linear regression fit. Ridge regression yielded the most favorable performance of 80% beacuse of which we can say it is a preferred method for optimizing the dataset's performance in Jamboree Project.

Insights

1. CGPA, GRE Score, TOEFL Score are the priority features for predicting admission chances with CGPA the most significant
2. In-Depth data exploration of the CGPA column to identify universities where students have a high chances of admission. Followed by FRE score and TOEFL score.
3. University Rating does not significantly impact admission chances. Therefore focus on other influential factors rather than solely relying on one university rankings.

Recommendations

1. While analysis we determined that 'serial no' does not contribute to the predictive power of model, leading us to eliminate it from consideration.
2. We performed thorough outlier treatment, which did not significantly impact the dataset, ensuring data integrity.
3. Standardizing the features resulted in a more uniformly distributed dataset, thereby enhancing the predictive capabilities of our model. These methodical steps underscore the robustness of our approach and demonstrate tyransparency in our methodology.
4. We observed that both GRE Scores and Toefl scores contribute significantly in prediction the chance of admission. Conversely, University rating and Research ccontribute less. Therefore, when considering feature selection for modeling. prioritizing GREScore and TOEFL score would likely yield more accuracy.
5. We employed Linear Reg algorithm for regression Analysis, which yielded best fit model. Analysis also incluided Lasso and ridge to enhance interpretability and optimize feature selection. However linear regreession line emerged as the most suitable fir for our dataset.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: