Jamboree has helped thousands of students like you 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.

Problem Statement :

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

Data Description :

Jamboree collected the Data of different Students who enrolled in Jamboree for preparation in GMAT, GRE or SAT Exams. The Dataset has the following features:

- Serial Number - Represents the Unique Row ID for each Student
- GRE Scores - GRE Scores obtained out of 340
- TOEFL Scores - TOEFL Scores obtained out of 120
- University Ranking - Rankings of Different Universities from 0 to 5
- Statement of Purpose and Letter of Recommendation Strength - SOP and LOR ratings out of 5
- Undergraduate GPA - Marks obtained during the Undergraduation from 0 to 10
- Research Experience - Whether any Student has Prior Research Experience (0 or 1)
- Chances of Admit - Whether any Student will get Admission (0 or 1)

Importing Required Libraries :

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

# Train Data and Test Data Split
from sklearn.model_selection import train_test_split

# Linear Regression using Statsmodel
import statsmodels.api as sm

# Checking for VIF (Variance Inflation Factor) in our Model
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Importing SKLEARN libraries for our Model checking
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error , r2_score , mean_squared_error , mean_absolute_percentage_error
```

In [2]:

```
df = pd.read_csv("Jamboree_Admission.csv", index_col = False)
df
```

Out[2]:

	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

Exploratory Data Analysis :

In [3]:

```
df.shape
```

Out[3]:

```
(500, 9)
```

In [4]:

```
df.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 [5]:

```
df.columns
```

Out[5]:

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

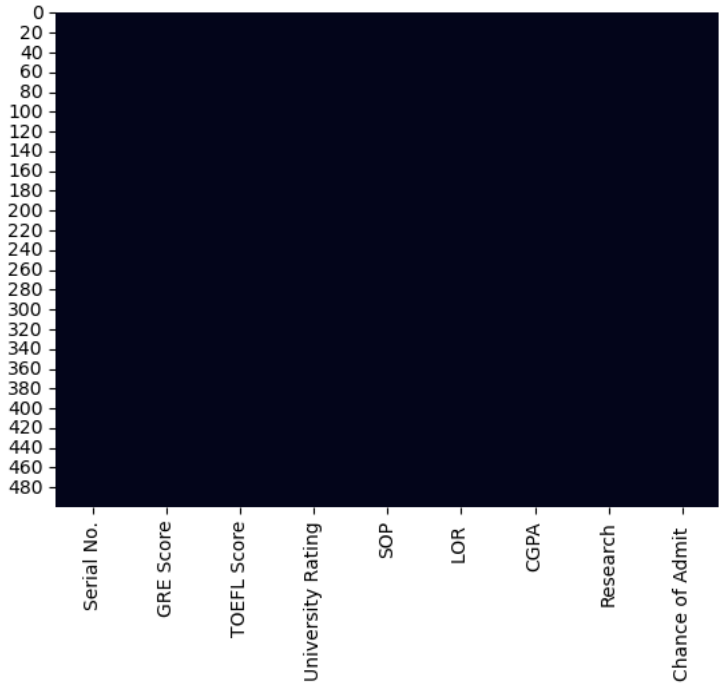
a) Checking for Null and NaN Values in the Dataset :

In [6]:

```
sns.heatmap(df.isnull(), cbar=False)
```

Out[6]:

```
<AxesSubplot:>
```

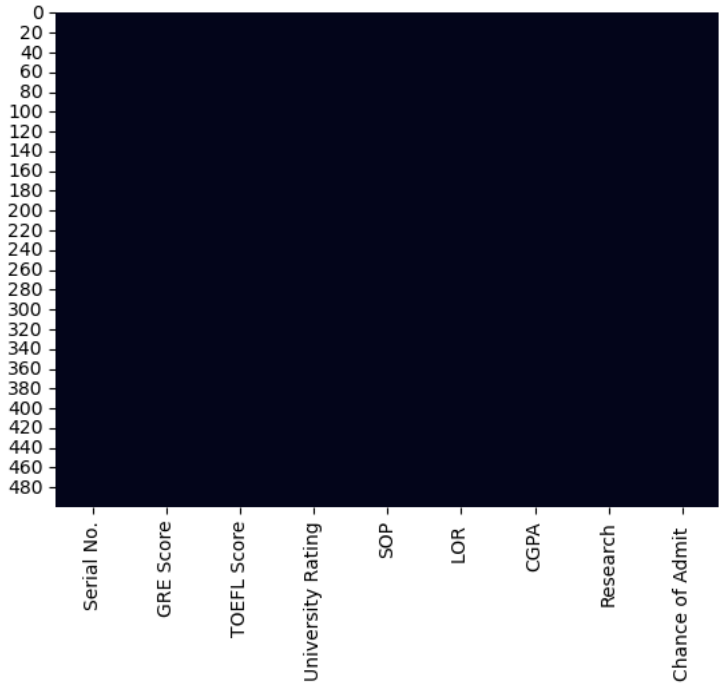


In [7]:

```
sns.heatmap(df.isna(), cbar=False)
```

Out[7]:

<AxesSubplot:>



Thus , Dataset has no Null or NaN Values. So we can proceed further.

b) Other Descriptions and Details on Dataset :

In [8]:

```
df.describe(include = 'int')
```

Out[8]:

	Serial No.	GRE Score	TOEFL Score	University Rating	Research
count	500.000000	500.000000	500.000000	500.000000	500.000000
mean	250.500000	316.472000	107.192000	3.114000	0.560000
std	144.481833	11.295148	6.081868	1.143512	0.496884
min	1.000000	290.000000	92.000000	1.000000	0.000000
25%	125.750000	308.000000	103.000000	2.000000	0.000000
50%	250.500000	317.000000	107.000000	3.000000	1.000000
75%	375.250000	325.000000	112.000000	4.000000	1.000000
max	500.000000	340.000000	120.000000	5.000000	1.000000

In [9]:

```
df.describe(include = 'float')
```

Out[9]:

	SOP	LOR	CGPA	Chance of Admit
count	500.000000	500.00000	500.000000	500.00000
mean	3.374000	3.48400	8.576440	0.72174
std	0.991004	0.92545	0.604813	0.14114
min	1.000000	1.00000	6.800000	0.34000
25%	2.500000	3.00000	8.127500	0.63000
50%	3.500000	3.50000	8.560000	0.72000
75%	4.000000	4.00000	9.040000	0.82000
max	5.000000	5.00000	9.920000	0.97000

In [10]:

```
df.describe(include = 'all')
```

Out[10]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.00000
mean	250.500000	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	0.72174
std	144.481833	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	0.14114
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.34000
25%	125.750000	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.63000
50%	250.500000	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.72000
75%	375.250000	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.82000
max	500.000000	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.97000

Observations :

- GRE Score ranges from 290 to 340, with Mean of 316.47 and Median of 317. Since Mean is almost Equal to the Median, thus 'GRE Score' has to be normally distributed.
- TOEFL Score ranges from 92 to 120, with Mean of 107.19 and Median of 107. Since Mean is almost Equal to the Median, thus 'TOEFL Score' has to be normally distributed.
- Chance of Admit ranges from 0.34 to 0.97, with Mean of 0.72114 and Median of 0.72. Since Mean is almost Equal to the Median, thus 'Chance of Admit' has to be normally distributed.
- CGPA ranges from 6.80 to 9.92, with Mean of 8.57 and Median of 8.56. Since Mean is almost Equal to the Median, thus Undergraduate 'CGPA' has to be normally distributed.
- Dataset has no Duplicated Values. Thus , we find that there are 500 , 49 , 29 , 5 , 9 , 9 , 184 , 2 , 61 unique values for the Columns - 'Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA', 'Research', 'Chance of Admit '.

In [11]:

```
df.duplicated().unique()
```

Out[11]:

```
array([False])
```

In [12]:

```
for column_name in df:
    print(column_name,':',df[column_name].nunique())
```

```
Serial No. : 500
GRE Score : 49
TOEFL Score : 29
University Rating : 5
SOP : 9
LOR  : 9
CGPA : 184
Research : 2
Chance of Admit  : 61
```

In [13]:

```
# Need to remove the Extra Spaces present in the Columns : 'Chance of Admit ' and 'LOR '
df.rename(columns = {'Chance of Admit ' : 'Chance of Admit' , 'LOR ' : 'LOR'} , inplace = True)
```

In [14]:

```
df.columns
```

Out[14]:

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

In [15]:

```
# We can remove the Unwanted Columns like Serial Number
df.drop(columns = 'Serial No.' , inplace = True)
```

In [16]:

```
# df.drop(columns = 'index' , inplace = True)
```

In [17]:

```
df.head()
```

Out[17]:

	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 [18]:

```
# Following are the Categorical Columns present in our Dataset : SOP , LOR , Research , University Rating
# Need to convert these columns into 'Category' Datatype.

categorical_cols = ['SOP' , 'LOR' , 'University Rating' , 'Research']

for i in categorical_cols :
    df[categorical_cols] = df[categorical_cols].astype('category')
```

c) Univariate Analysis and Outlier Treatment:

Univariate Analysis is defined as Analysis carried out on only one (“uni”) variable (“variate”) to Summarize or Describe the Variables.

In [19]:

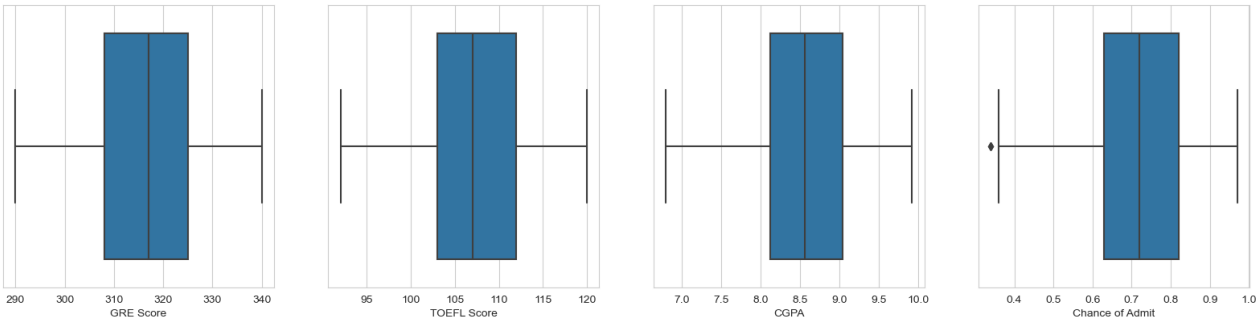
```
sns.set_style('whitegrid')
```

In [20]:

```
numerical_cols = ["GRE Score" , "TOEFL Score" , "CGPA" , "Chance of Admit"]
```

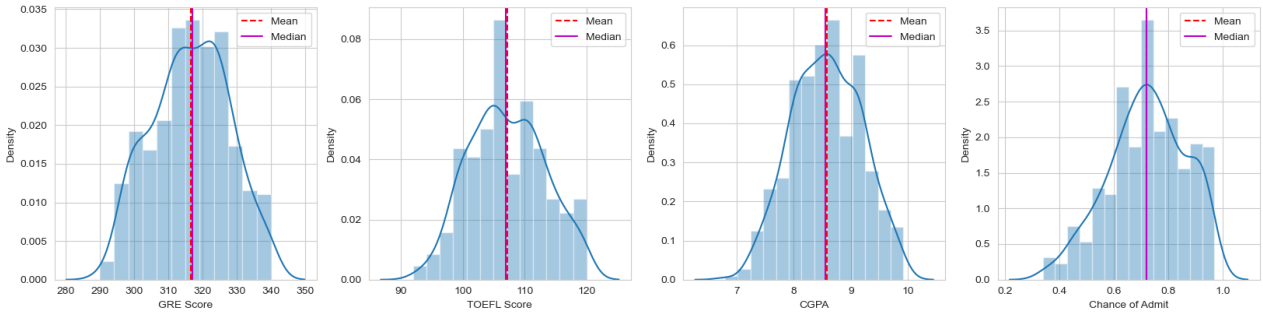
In [21]:

```
plt.figure(figsize = (20,10))

for i in range(len(numerical_cols)):
    plt.subplot(2,4,i + 1)
    sns.boxplot(df[numerical_cols[i]])
```



In [22]:

```
plt.figure(figsize=(20,10))

for i in range(len(numerical_cols)):
    plt.subplot(2,4,i + 1)
    sns.distplot(df[numerical_cols[i]])
    plt.axvline(df[numerical_cols[i]].mean() , color = 'r' , linestyle = '--' , label = 'Mean')
    plt.axvline(df[numerical_cols[i]].median() , color = 'm' , linestyle = '-' , label = 'Median')
    plt.legend()
plt.show()
```



In [23]:

```
# Checking for Outlier Data present in Different Columns

outliers_in_df = df.copy(deep = True)

for i in range(len(numerical_cols)):
    Q1 = df[numerical_cols[i]].quantile(0.25)
    Q3 = df[numerical_cols[i]].quantile(0.75)
    IQR = Q3 - Q1
    lower_limit = round(Q1 - 1.5 * IQR , 2)
    upper_limit = round(Q3 + 1.5 * IQR , 2)
    print(numerical_cols[i])
    print('Lower Limit for Boxplot - ', lower_limit , '\nUpper Limit for Boxplot -', upper_limit)
    outlier_df = df[(df[numerical_cols[i]] > upper_limit) | (df[numerical_cols[i]] < lower_limit)]
    outliers_in_df = outliers_in_df[(outliers_in_df[numerical_cols[i]] <= upper_limit) & (outliers_in_df[numerical_cols[i]] >= lower_limit

    outlier_percentage = round((len(outlier_df) / len(df)) * 100 , 2)
    print('Total Number of Outliers : ', len(outlier_df))
    print('Percentage of Outlier Records :' , outlier_percentage , '%\n')
    print(outlier_df)
    print()

overall_outliers_percentage_df = round(((len(df) - len(outliers_in_df)) / len(df)) * 100 , 2)
print('Total Number of Outliers from all Records :', len(df) - len(outliers_in_df))
print('Percentage of Rows as Outliers :', overall_outliers_percentage_df)
```

```
GRE Score
Lower Limit for Boxplot -  282.5
Upper Limit for Boxplot - 350.5
Total Number of Outliers :  0
Percentage of Outlier Records : 0.0 %

Empty DataFrame
Columns: [GRE Score, TOEFL Score, University Rating, SOP, LOR, CGPA, Research, Chance of Admit]
Index: []

TOEFL Score
Lower Limit for Boxplot -  89.5
Upper Limit for Boxplot - 125.5
Total Number of Outliers :  0
Percentage of Outlier Records : 0.0 %

Empty DataFrame
Columns: [GRE Score, TOEFL Score, University Rating, SOP, LOR, CGPA, Research, Chance of Admit]
Index: []

CGPA
Lower Limit for Boxplot -  6.76
Upper Limit for Boxplot - 10.41
Total Number of Outliers :  0
Percentage of Outlier Records : 0.0 %

Empty DataFrame
Columns: [GRE Score, TOEFL Score, University Rating, SOP, LOR, CGPA, Research, Chance of Admit]
Index: []

Chance of Admit
Lower Limit for Boxplot -  0.35
Upper Limit for Boxplot - 1.1
Total Number of Outliers :  2
Percentage of Outlier Records : 0.4 %

     GRE Score  TOEFL Score University Rating  SOP  LOR  CGPA Research  \
92         298           98                2  4.0  3.0  8.03        0
376        297           96                2  2.5  2.0  7.43        0

     Chance of Admit
92              0.34
376             0.34

Total Number of Outliers from all Records : 2
Percentage of Rows as Outliers : 0.4
```

In [24]:

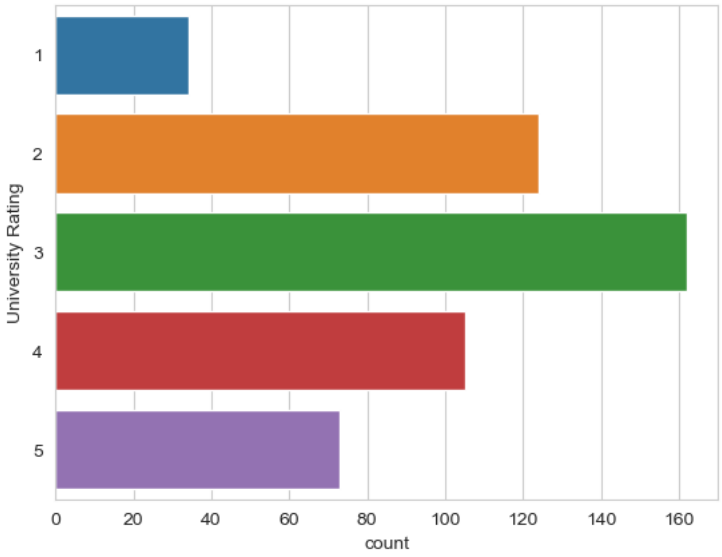
```
df = outliers_in_df.copy(deep = True)
```

In [25]:

```
sns.countplot(data = df , y = "University Rating")
```

Out[25]:

```
<AxesSubplot:xlabel='count', ylabel='University Rating'>
```

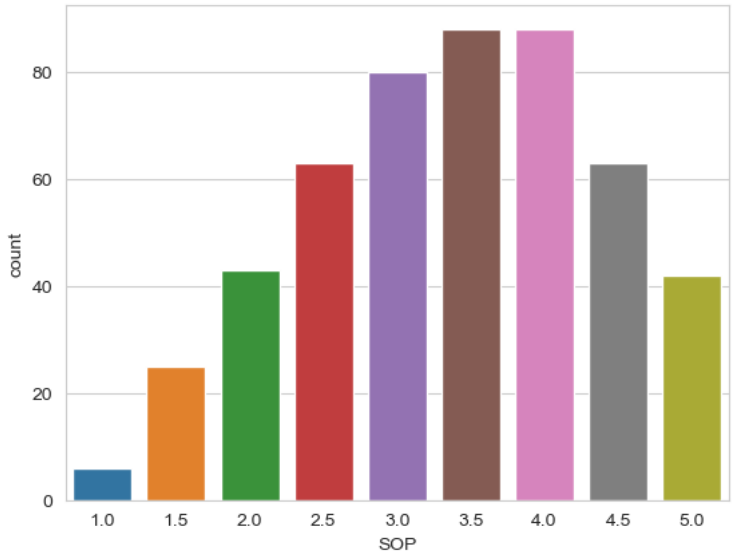


In [26]:

```
sns.countplot(data = df , x = "SOP")
```

Out[26]:

```
<AxesSubplot:xlabel='SOP', ylabel='count'>
```

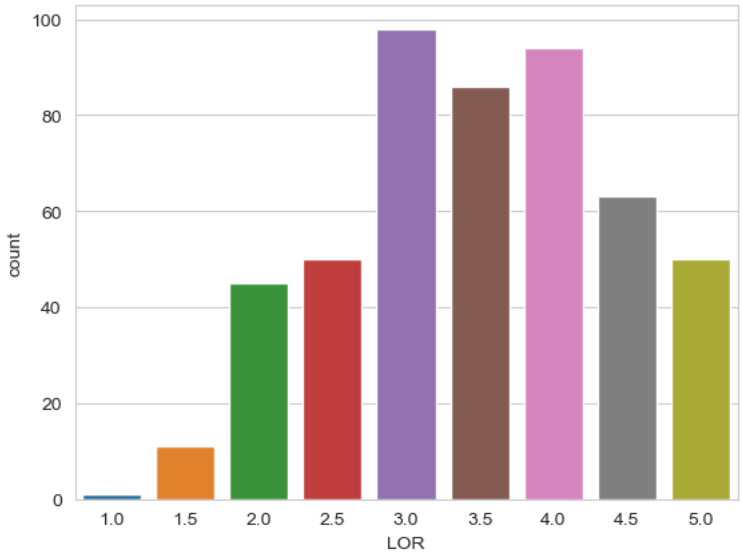


In [27]:

```
sns.countplot(data = df , x = "LOR")
```

Out[27]:

```
<AxesSubplot:xlabel='LOR', ylabel='count'>
```

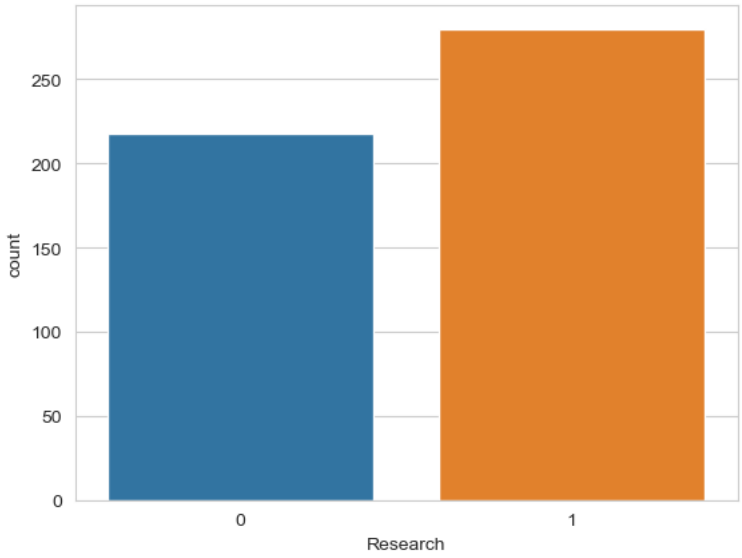


In [28]:

```
sns.countplot(data = df , x = "Research")
```

Out[28]:

```
<AxesSubplot:xlabel='Research', ylabel='count'>
```



Observations :

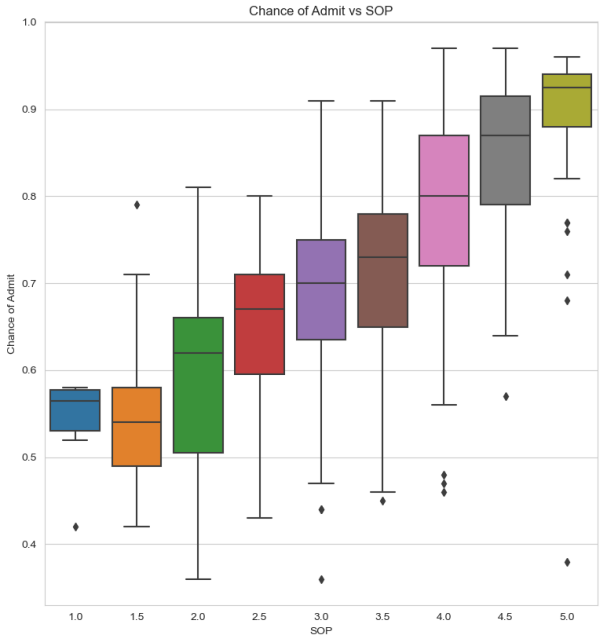
- Considering the University Rating aspect , Maximum Students were enrolled in the Universities having Ratings of 3.
- Considering the SOP aspect , Maximum Students had a SOP Ratings of 3 and 4.
- Considering the LOR aspect , Maximum Students had a LOR Rating of 3.
- Considering the Research aspect , More number of Students had a Prior Research Experience.
- There are 2 Outliers present only in 'Chance of Admit' column. Total Percentage of Data being an Outlier is 0.4%, checked using IQR Method.

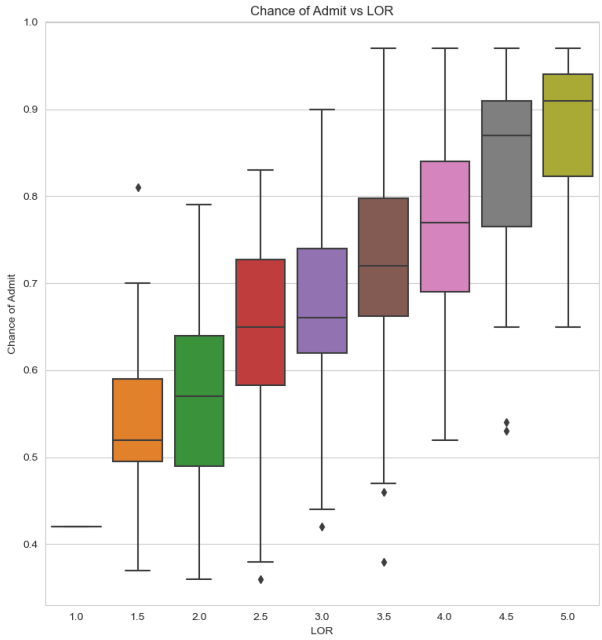
d) Bivariate Analysis :

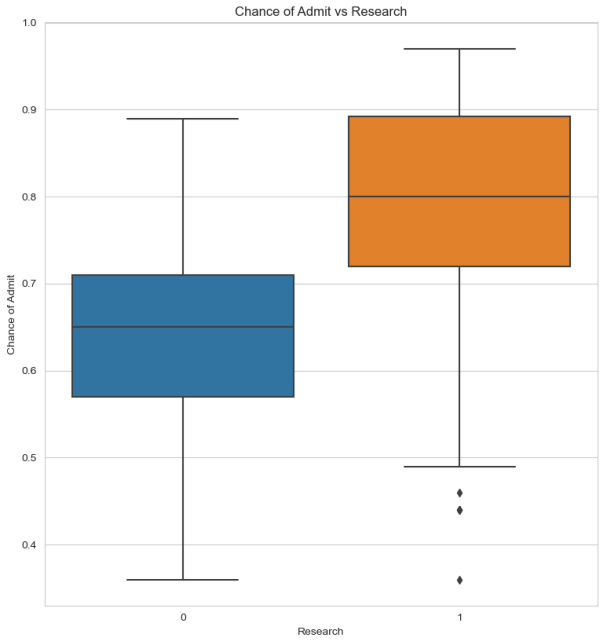
Bivariate Analysis is stated to be an Analysis of any concurrent Relation between two Variables or Attributes.

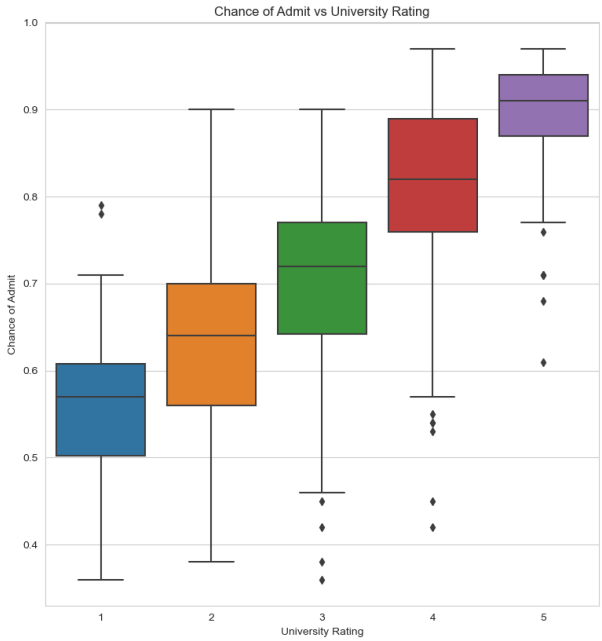
In [29]:

```
categorical_cols = ['SOP' , 'LOR' , 'Research' , 'University Rating' ]
sns.set_style('whitegrid')
fig , axis = plt.subplots(nrows = 2 , ncols = 2 , figsize = (20,15))
fig.subplots_adjust(top = 1.2)
count = 0

for i in range(2):
    for j in range(2):
        sns.boxplot(data = df , y = 'Chance of Admit' , x = categorical_cols[count] , ax = axis[i,j])
        axis[i,j].set_title(f"Chance of Admit vs {categorical_cols[count]}")
        count += 1
```



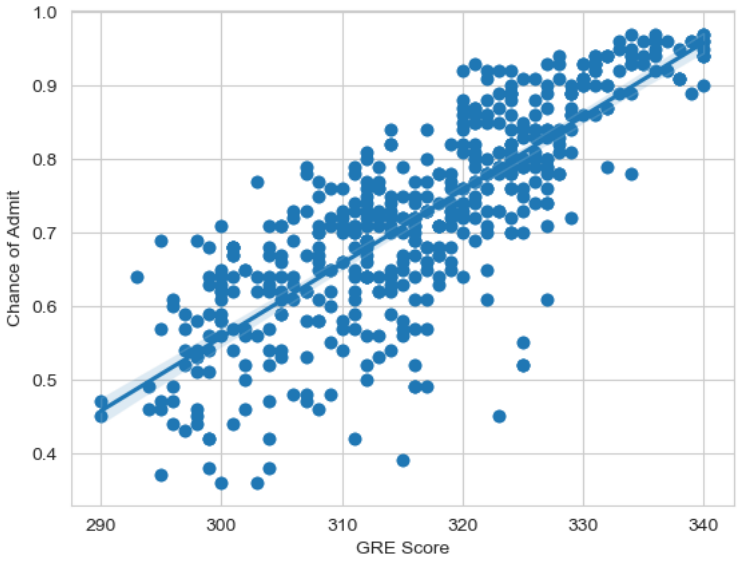


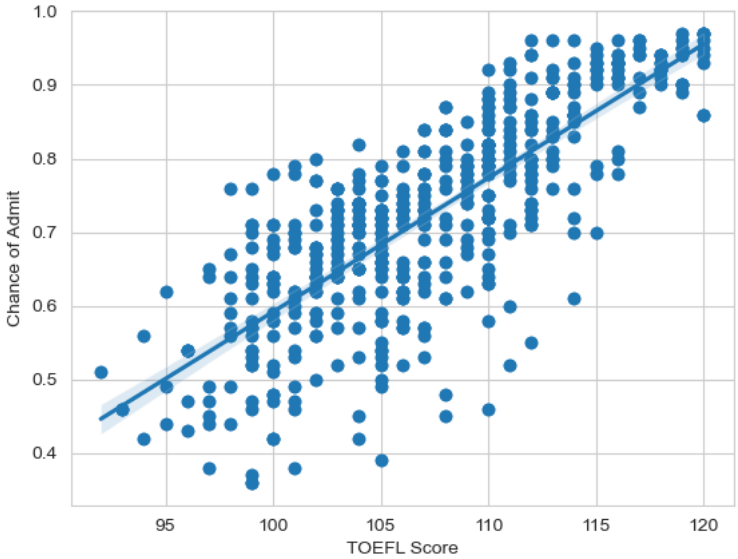


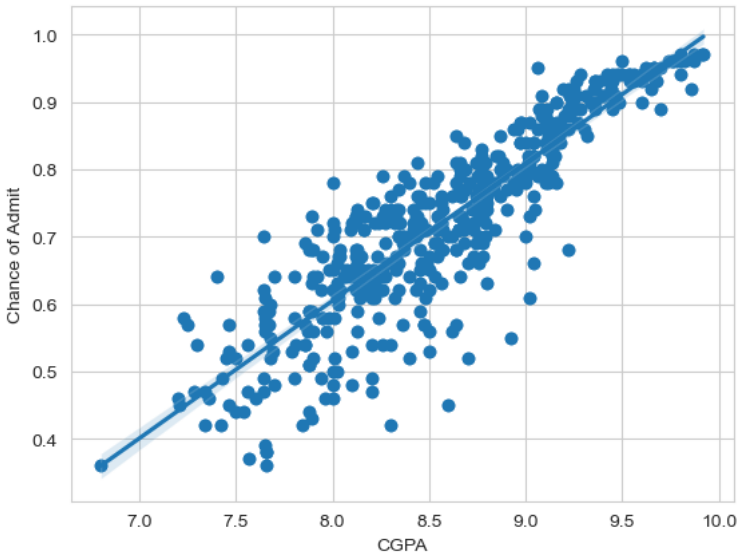


In [30]:

```
num_cols = ["GRE Score" , "TOEFL Score" , "CGPA"]

for i in range(len(num_cols)):
    plt.scatter(df[num_cols[i]] , y = df['Chance of Admit'])
    sns.regplot(x = num_cols[i] , y = 'Chance of Admit' , data = df)
    plt.xlabel(num_cols[i])
    plt.ylabel('Chance of Admit')
    plt.show()
```





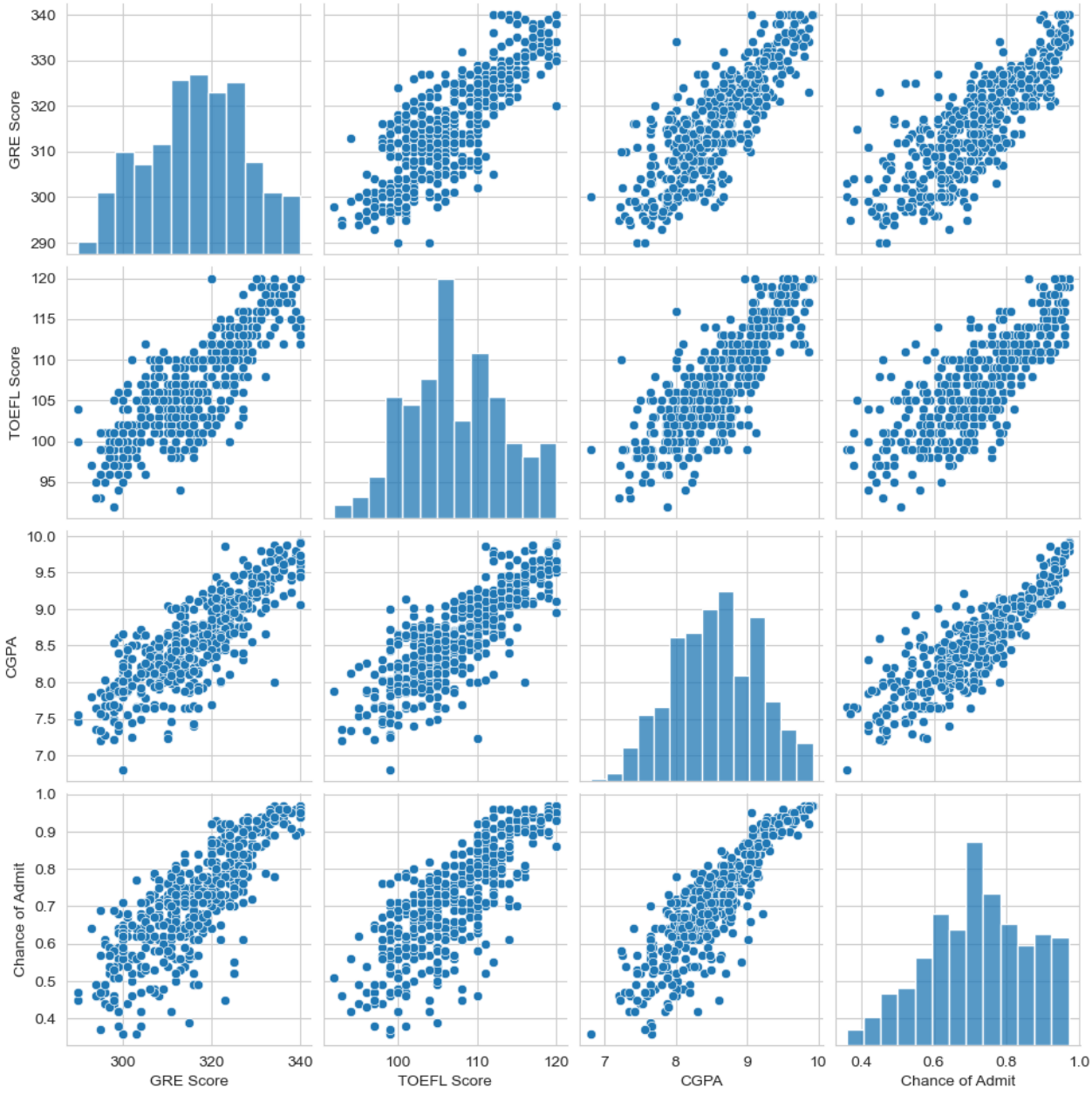


In [31]:

```
# Check for Correlation within Different Columns

sns.pairplot(df)
```

Out[31]:

```
<seaborn.axisgrid.PairGrid at 0x134c7f91a90>
```



In [32]:

```
sns.pairplot(df , hue = 'Research')
```

Out[32]:

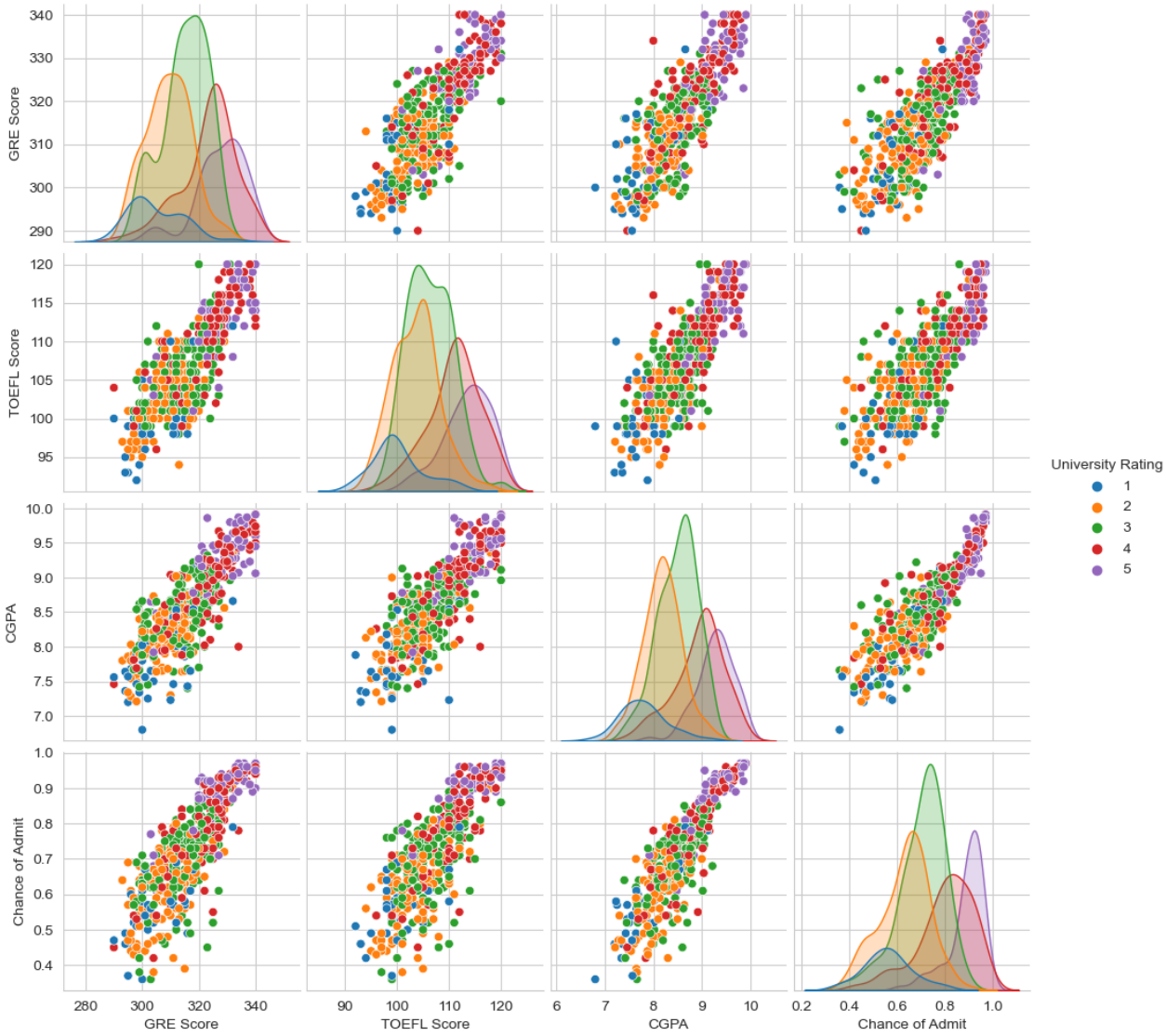
```
<seaborn.axisgrid.PairGrid at 0x134c7e18a00>
```

In [33]:

```
sns.pairplot(df , hue = 'University Rating')
```

Out[33]:

```
<seaborn.axisgrid.PairGrid at 0x134c7dd80d0>
```

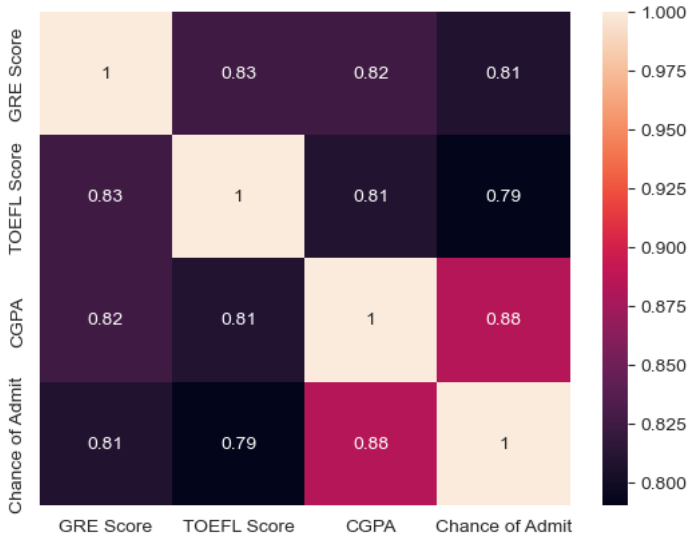


In [34]:

```
# Using Heatmap to check the Correlations as well as for Multicollinearity

sns.heatmap(df.corr() , annot = True)
```

Out[34]:

<AxesSubplot:>



Observations :

- Numerical Columns like GRE Scores , TOEFL Scores and CGPA are following a Linear Relationship with respect to Chances of Admit.
- As per the Pairplot for Research , Students considering themselves towards Research Domain have higher Chances of getting Admission, in comparison to those who don't consider Research into consideration.
- As per the Pairplot for University Ranking , Students require better GRE , TOEFL and CGPA Scores to get admission into better Universities. Considering the Data , Maximum number of Students got admissions in Universities with Ratings 3. And minimum Number of Students got admission in Universities with Ratings 1.
- As per the Heatmap , columns 'GRE Scores' , 'TOEFL Scores' and 'CGPA' have a high correlation of around 0.8. Thus , as per the Assumptions of Linear Regression, we need to deal with Multicollinearity.
- All Numerical column(features) are almost normally distributed whereas 'Chance of Admit' is slightly left skewed. Both mean and median co-incides for all the distribution.
- Column 'Chance of Admit' has a Positive Linear Relation with all of the Numerical and Categorical Features.
- Chance of Admit vs Research :

  Having Research has more Median Value compared to not having Research. Also, Research Students have higher chances than Non-Researchers. Outliers are present only for Research Aspect.

- Chance of Admit vs University Rating :

  As the University Rating increases, the chances of getting admission also Increases, since Rating '5' has the highest Median value followed by 4 , 3 , 2 and 1. Outliers present for most of the Categories.

- Chance of Admit vs SOP :

  As the SOP Rating increases, the chances of getting admission also Increases, since Rating '5' has the highest Median value followed by others. Outliers present for most of the Categories.

- Chance of Admit vs LOR :

  As the LOR Rating increases, the chances of getting admission also Increases, since Rating '5' has the highest Median value followed by others. Outliers present for most of the Categories.

Feature Engineering for Model :

Since the Columns 'GRE Score' , 'TOEFL Score' and 'CGPA' have high correlation , thus we need to drop two of these columns to remove Multicollinearity. With Multicollinearity , our Linear Regression model will show Bad Results.

We can drop 'TOEFL Score' and 'CGPA' Columns

In [35]:

```
df_col = ['GRE Score' , 'University Rating' , 'SOP' , 'LOR' , 'Research' , 'Chance of Admit']
new_df = df[df_col]
new_df.shape
```

Out[35]:

```
(498, 6)
```

Also, for our Linear Regression Model to work, we need all our Inputs in the form of Numerical Data, instead of Categorical Data. Thus converting the Categorical Columns into Numerical Columns.

In [36]:

```
cat_col = ['University Rating' , 'SOP' , 'LOR']
dummyCol = pd.get_dummies(new_df[categorical_cols] , drop_first = True)

dummyCol.rename(columns = {'Research_1 ' : 'University Rating_1'} , inplace = True)
dummyCol.head()
```

Out[36]:

	SOP_1.5	SOP_2.0	SOP_2.5	SOP_3.0	SOP_3.5	SOP_4.0	SOP_4.5	SOP_5.0	LOR_1.5	LOR_2.0	...	LOR_3.0	LOR_3.5	LOR_4.0	LOR_4.5	LOR_5.0
0	0	0	0	0	0	0	1	0	0	0	...	0	0	0	1	0
1	0	0	0	0	0	1	0	0	0	0	...	0	0	0	1	0
2	0	0	0	1	0	0	0	0	0	0	...	0	1	0	0	0
3	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0
4	0	1	0	0	0	0	0	0	0	0	...	1	0	0	0	0

5 rows × 21 columns

In [37]:

```
new_df = pd.concat([new_df , dummyCol] , axis = 1)
new_df.drop(cat_col , axis = 1 , inplace = True)

new_df.shape
```

Out[37]:

```
(498, 24)
```

In [38]:

```
# Rescaling using Standard Scaler

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

num_col = ['GRE Score' , 'Chance of Admit']
new_df[num_col] = scaler.fit_transform(new_df[num_col])
```

In [39]:

```
new_df.head()
```

Out[39]:

	GRE Score	Research	Chance of Admit	SOP_1.5	SOP_2.0	SOP_2.5	SOP_3.0	SOP_3.5	SOP_4.0	SOP_4.5	...	LOR_3.0	LOR_3.5	LOR_4.0	LOR_4.5	LOR_5.0
0	1.819220	1	1.413400	0	0	0	0	0	0	1	...	0	0	0	1	0
1	0.662850	1	0.263867	0	0	0	0	0	1	0	...	0	0	0	1	0
2	-0.048763	1	-0.023516	0	0	0	1	0	0	0	...	0	1	0	0	0
3	0.484947	1	0.551251	0	0	0	0	1	0	0	...	0	0	0	0	0
4	-0.226666	0	-0.526436	0	1	0	0	0	0	0	...	1	0	0	0	0

5 rows × 24 columns

Model Preparation :

In [40]:

```
# Train - Test Split

df_train , df_test = train_test_split(new_df , train_size = 0.7 , random_state = 100)
```

In [41]:

```
df_train.shape
```

Out[41]:

```
(348, 24)
```

In [42]:

```
df_test.shape
```

Out[42]:

```
(150, 24)
```

In [43]:

```
df_train.describe().T
```

Out[43]:

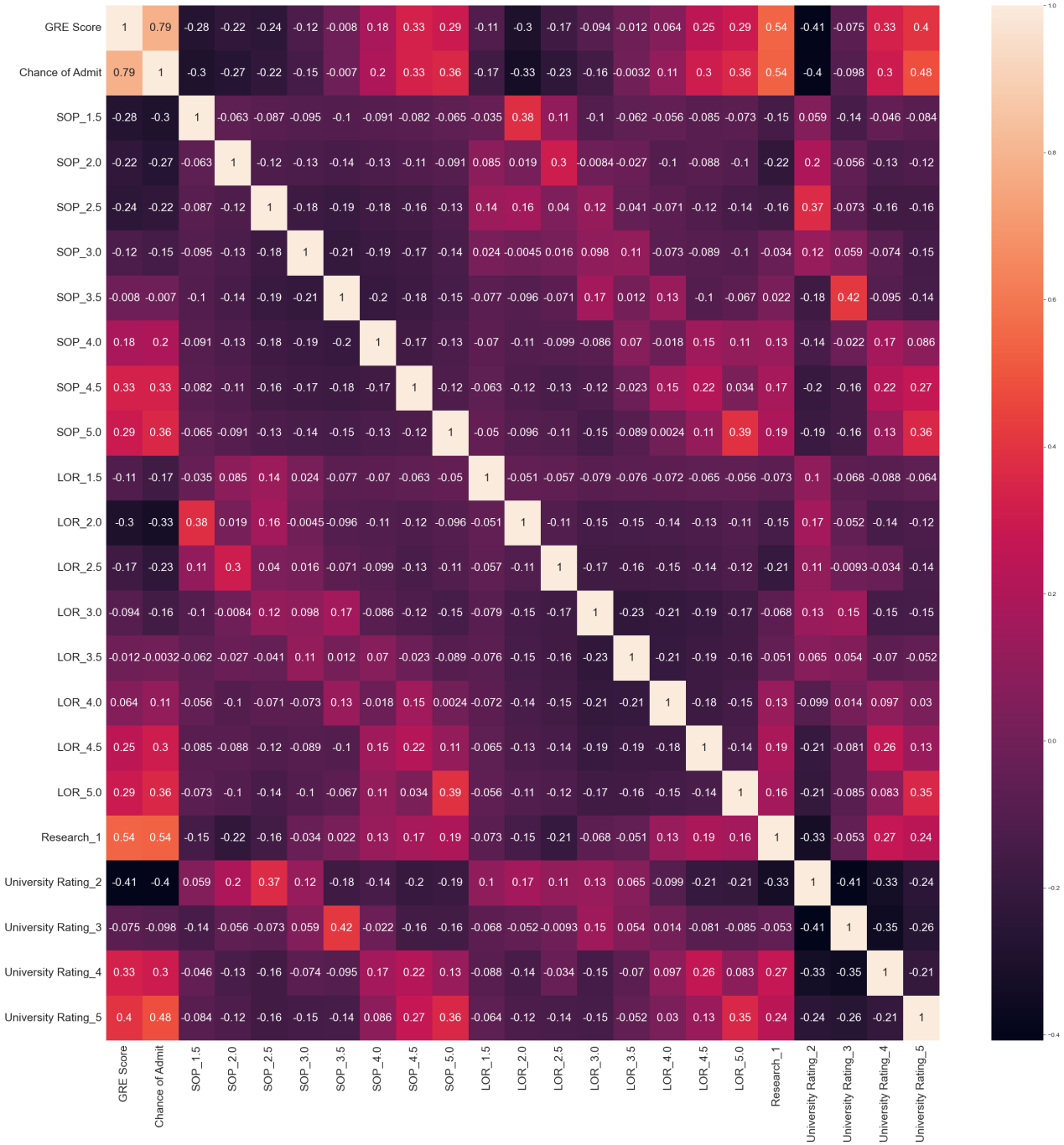
	count	mean	std	min	25%	50%	75%	max
GRE Score	348.0	-0.019623	0.989056	-2.361503	-0.760375	-0.048763	0.662850	2.086075
Chance of Admit	348.0	-0.014225	1.008494	-2.609964	-0.688089	-0.023516	0.712904	1.772629
SOP_1.5	348.0	0.043103	0.203382	0.000000	0.000000	0.000000	0.000000	1.000000
SOP_2.0	348.0	0.080460	0.272395	0.000000	0.000000	0.000000	0.000000	1.000000
SOP_2.5	348.0	0.143678	0.351268	0.000000	0.000000	0.000000	0.000000	1.000000
SOP_3.0	348.0	0.166667	0.373215	0.000000	0.000000	0.000000	0.000000	1.000000
SOP_3.5	348.0	0.183908	0.387967	0.000000	0.000000	0.000000	0.000000	1.000000
SOP_4.0	348.0	0.155172	0.362590	0.000000	0.000000	0.000000	0.000000	1.000000
SOP_4.5	348.0	0.129310	0.336026	0.000000	0.000000	0.000000	0.000000	1.000000
SOP_5.0	348.0	0.086207	0.281073	0.000000	0.000000	0.000000	0.000000	1.000000
LOR_1.5	348.0	0.025862	0.158952	0.000000	0.000000	0.000000	0.000000	1.000000
LOR_2.0	348.0	0.089080	0.285270	0.000000	0.000000	0.000000	0.000000	1.000000
LOR_2.5	348.0	0.109195	0.312333	0.000000	0.000000	0.000000	0.000000	1.000000
LOR_3.0	348.0	0.189655	0.392593	0.000000	0.000000	0.000000	0.000000	1.000000
LOR_3.5	348.0	0.178161	0.383199	0.000000	0.000000	0.000000	0.000000	1.000000
LOR_4.0	348.0	0.163793	0.370621	0.000000	0.000000	0.000000	0.000000	1.000000
LOR_4.5	348.0	0.137931	0.345324	0.000000	0.000000	0.000000	0.000000	1.000000
LOR_5.0	348.0	0.106322	0.308693	0.000000	0.000000	0.000000	0.000000	1.000000
Research_1	348.0	0.554598	0.497726	0.000000	0.000000	1.000000	1.000000	1.000000
University Rating_2	348.0	0.275862	0.447591	0.000000	0.000000	0.000000	1.000000	1.000000
University Rating_3	348.0	0.301724	0.459667	0.000000	0.000000	0.000000	1.000000	1.000000
University Rating_4	348.0	0.224138	0.417614	0.000000	0.000000	0.000000	0.000000	1.000000
University Rating_5	348.0	0.135057	0.342277	0.000000	0.000000	0.000000	0.000000	1.000000

In [44]:

```
plt.figure(figsize= (30,30))
dc = sns.heatmap(df_train.corr() , annot = True , annot_kws={'size': 18})
dc.set_xticklabels(dc.get_xmajorticklabels(), fontsize = 18)
dc.set_yticklabels(dc.get_ymajorticklabels(), fontsize = 18)
dc
```

Out[44]:

<AxesSubplot:>



Thus, using Dummies Feature , No Features are left correlated with each other

Training our Model

Regarding our Model , we are interested in the Following 3 Things :

a) **Coefficient and its p-values** : If the p-value of the coefficient for any Feature is less than 0.05 , it means it is Statistically Significant.

b) **R^2 Values** : As the columns with p-values greater than 0.05 will get removed, our R^2 value will get better.

c) **F Statistics should have a very low p-value** : This means that our entire model is Statistically Significant.

In [45]:

```
x_train = df_train
y_train = df_train.pop('Chance of Admit')

print(x_train.shape)
print(y_train.shape)
```

```
(348, 23)
(348,)
```

In [46]:

```
x_train.head()
```

Out[46]:

	GRE Score	Research	SOP_1.5	SOP_2.0	SOP_2.5	SOP_3.0	SOP_3.5	SOP_4.0	SOP_4.5	SOP_5.0	...	LOR_3.0	LOR_3.5	LOR_4.0	LOR_4.5	LOR_5
84	2.086075	1	0	0	0	0	0	0	1	0	...	0	0	0	1	
309	-0.760375	0	0	0	0	0	1	0	0	0	...	1	0	0	0	
494	-1.383036	1	0	0	1	0	0	0	0	0	...	0	0	0	0	
127	0.218092	1	0	0	1	0	0	0	0	0	...	0	0	0	0	
311	1.018656	1	0	0	0	0	0	0	1	0	...	0	0	1	0	

5 rows × 23 columns

In [47]:

```
# Keep X-Train Data for building the Model
x_train_1 = x_train.copy(deep = True)

# Adding the Constant Column
x_train_1 = sm.add_constant(x_train_1)

print(x_train_1.shape)
LR_1 = sm.OLS(y_train , x_train_1).fit()
LR_1.params
```

```
(348, 24)
```

Out[47]:

```
const                -0.528697
GRE Score             0.486370
Research              0.114278
SOP_1.5              -0.108309
SOP_2.0               0.057947
SOP_2.5               0.291866
SOP_3.0               0.210621
SOP_3.5               0.347001
SOP_4.0               0.373201
SOP_4.5               0.448586
SOP_5.0               0.536352
LOR_1.5              -0.473925
LOR_2.0              -0.317931
LOR_2.5              -0.170782
LOR_3.0              -0.160570
LOR_3.5               0.024940
LOR_4.0               0.056223
LOR_4.5               0.222717
LOR_5.0               0.290631
Research_1            0.114278
University Rating_2   0.037752
University Rating_3   0.033659
University Rating_4   0.129434
University Rating_5   0.437460
dtype: float64
```

In [48]:

```
LR_1.summary()
```

Out[48]:

OLS Regression Results

Dep. Variable:	Chance of Admit	**R-squared:**	0.735
Model:	OLS	**Adj. R-squared:**	0.718
Method:	Least Squares	**F-statistic:**	43.05
Date:	Tue, 03 Jan 2023	**Prob (F-statistic):**	6.71e-81
Time:	23:09:56	**Log-Likelihood:**	-265.17
No. Observations:	348	**AIC:**	574.3
Df Residuals:	326	**BIC:**	659.1
Df Model:	21		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.5287	0.250	-2.115	0.035	-1.020	-0.037
GRE Score	0.4864	0.043	11.219	0.000	0.401	0.572
Research	0.1143	0.036	3.203	0.001	0.044	0.184
SOP_1.5	-0.1083	0.311	-0.349	0.728	-0.719	0.503
SOP_2.0	0.0579	0.299	0.194	0.847	-0.531	0.647
SOP_2.5	0.2919	0.300	0.972	0.332	-0.299	0.883
SOP_3.0	0.2106	0.301	0.699	0.485	-0.382	0.804
SOP_3.5	0.3470	0.310	1.121	0.263	-0.262	0.956
SOP_4.0	0.3732	0.314	1.190	0.235	-0.244	0.990
SOP_4.5	0.4486	0.321	1.396	0.164	-0.184	1.081
SOP_5.0	0.5364	0.330	1.623	0.106	-0.114	1.186
LOR_1.5	-0.4739	0.167	-2.846	0.005	-0.802	-0.146
LOR_2.0	-0.3179	0.106	-3.000	0.003	-0.526	-0.109
LOR_2.5	-0.1708	0.092	-1.847	0.066	-0.353	0.011
LOR_3.0	-0.1606	0.079	-2.032	0.043	-0.316	-0.005
LOR_3.5	0.0249	0.079	0.315	0.753	-0.131	0.181
LOR_4.0	0.0562	0.085	0.663	0.508	-0.111	0.223
LOR_4.5	0.2227	0.095	2.354	0.019	0.037	0.409
LOR_5.0	0.2906	0.109	2.667	0.008	0.076	0.505
Research_1	0.1143	0.036	3.203	0.001	0.044	0.184
University Rating_2	0.0378	0.143	0.264	0.792	-0.243	0.319
University Rating_3	0.0337	0.153	0.220	0.826	-0.267	0.334
University Rating_4	0.1294	0.166	0.778	0.437	-0.198	0.457
University Rating_5	0.4375	0.186	2.351	0.019	0.071	0.804

Omnibus:	58.488	**Durbin-Watson:**	2.045
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	97.297
Skew:	-0.981	**Prob(JB):**	7.45e-22
Kurtosis:	4.692	**Cond. No.**	5.67e+16

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.55e-31. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

From above , we can see that

- R score : 0.735
- Based on the p-values, Features that are Eliminated are - University Rating_3 (0.826) , SOP_2 (0.847) , LOR_3.5 (0.753).

In [49]:

```
x_train_2 = x_train[['GRE Score' , 'Research' , 'University Rating_2' , 'University Rating_4' , 'University Rating_5' , 'SOP_1.5' , 'SOP_

x_train_2 = sm.add_constant(x_train_2)
LR_2 = sm.OLS(y_train , x_train_2).fit()

LR_2.summary()
```

Out[49]:

OLS Regression Results

Dep. Variable:	Chance of Admit	**R-squared:**	0.735
Model:	OLS	**Adj. R-squared:**	0.720
Method:	Least Squares	**F-statistic:**	47.85
Date:	Tue, 03 Jan 2023	**Prob (F-statistic):**	1.48e-82
Time:	23:09:56	**Log-Likelihood:**	-265.23
No. Observations:	348	**AIC:**	570.5
Df Residuals:	328	**BIC:**	647.5
Df Model:	19		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.4308	0.137	-3.140	0.002	-0.701	-0.161
GRE Score	0.4875	0.043	11.322	0.000	0.403	0.572
Research	0.2274	0.071	3.201	0.002	0.088	0.367
University Rating_2	0.0136	0.080	0.170	0.865	-0.144	0.171
University Rating_4	0.0999	0.094	1.061	0.289	-0.085	0.285
University Rating_5	0.4070	0.122	3.346	0.001	0.168	0.646
SOP_1.5	-0.1645	0.178	-0.925	0.356	-0.514	0.185
SOP_2.5	0.2469	0.126	1.960	0.051	-0.001	0.495
SOP_3.0	0.1658	0.126	1.318	0.188	-0.082	0.413
SOP_3.5	0.3060	0.132	2.310	0.022	0.045	0.567
SOP_4.0	0.3302	0.143	2.301	0.022	0.048	0.613
SOP_4.5	0.4051	0.160	2.536	0.012	0.091	0.719
SOP_5.0	0.4928	0.178	2.771	0.006	0.143	0.843
LOR_1.5	-0.5107	0.197	-2.589	0.010	-0.899	-0.123
LOR_2.0	-0.3457	0.131	-2.642	0.009	-0.603	-0.088
LOR_2.5	-0.1952	0.119	-1.636	0.103	-0.430	0.039
LOR_3.0	-0.1854	0.096	-1.925	0.055	-0.375	0.004
LOR_4.0	0.0314	0.101	0.309	0.757	-0.168	0.231
LOR_4.5	0.1990	0.111	1.794	0.074	-0.019	0.417
LOR_5.0	0.2666	0.126	2.121	0.035	0.019	0.514

Omnibus:	57.706	**Durbin-Watson:**	2.045
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	95.660
Skew:	-0.971	**Prob(JB):**	1.69e-21
Kurtosis:	4.682	**Cond. No.**	15.7

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

We can see that

- After removal of 3 features , R Score hasn't changed.
- Also, based on the p-values, we can see that Following Features should be removed : University Rating_2 (0.865) , SOP_1.5 (0.356) , LOR_4 (0.757).

In [50]:

```
x_train_3 = x_train[['GRE Score' , 'Research' , 'University Rating_4' , 'University Rating_5' , 'SOP_2.5' , 'SOP_3.0' , 'SOP_3.5' , 'SOP_

x_train_3 = sm.add_constant(x_train_3)
LR_3 = sm.OLS(y_train , x_train_3).fit()

LR_3.summary()
```

Out[50]:

OLS Regression Results

Dep. Variable:	Chance of Admit	**R-squared:**	0.734
Model:	OLS	**Adj. R-squared:**	0.721
Method:	Least Squares	**F-statistic:**	57.11
Date:	Tue, 03 Jan 2023	**Prob (F-statistic):**	5.82e-85
Time:	23:09:57	**Log-Likelihood:**	-265.76
No. Observations:	348	**AIC:**	565.5
Df Residuals:	331	**BIC:**	631.0
Df Model:	16		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.4558	0.116	-3.945	0.000	-0.683	-0.229
GRE Score	0.4900	0.042	11.545	0.000	0.407	0.574
Research	0.2268	0.070	3.230	0.001	0.089	0.365
University Rating_4	0.0913	0.091	1.007	0.314	-0.087	0.270
University Rating_5	0.3993	0.119	3.342	0.001	0.164	0.634
SOP_2.5	0.3001	0.113	2.660	0.008	0.078	0.522
SOP_3.0	0.2115	0.115	1.846	0.066	-0.014	0.437
SOP_3.5	0.3504	0.119	2.938	0.004	0.116	0.585
SOP_4.0	0.3756	0.133	2.817	0.005	0.113	0.638
SOP_4.5	0.4540	0.150	3.026	0.003	0.159	0.749
SOP_5.0	0.5416	0.169	3.201	0.002	0.209	0.874
LOR_1.5	-0.5099	0.192	-2.659	0.008	-0.887	-0.133
LOR_2.0	-0.3873	0.120	-3.238	0.001	-0.623	-0.152
LOR_2.5	-0.2039	0.111	-1.835	0.067	-0.423	0.015
LOR_3.0	-0.1968	0.085	-2.316	0.021	-0.364	-0.030
LOR_4.5	0.1835	0.097	1.898	0.059	-0.007	0.374
LOR_5.0	0.2493	0.113	2.204	0.028	0.027	0.472

Omnibus:	55.160	**Durbin-Watson:**	2.053
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	90.356
Skew:	-0.938	**Prob(JB):**	2.40e-20
Kurtosis:	4.648	**Cond. No.**	13.7

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

We can see that

- After removal of 3 features , R Score hasn't changed much
- Also, based on the p-values, we can see that Following Features should be removed : University Rating_4 (0.314) , SOP_3.0 (0.066) , LOR_2.5 (0.067).

In [51]:

```
x_train_4 = x_train[['GRE Score' , 'Research' , 'University Rating_5' , 'SOP_2.5' , 'SOP_3.5' , 'SOP_4.0' , 'SOP_4.5' , 'SOP_5.0' , 'LOR_1

x_train_4 = sm.add_constant(x_train_4)
LR_4 = sm.OLS(y_train , x_train_4).fit()

LR_4.summary()
```

Out[51]:

OLS Regression Results

Dep. Variable:	Chance of Admit	**R-squared:**	0.726
Model:	OLS	**Adj. R-squared:**	0.715
Method:	Least Squares	**F-statistic:**	68.03
Date:	Tue, 03 Jan 2023	**Prob (F-statistic):**	1.56e-85
Time:	23:09:57	**Log-Likelihood:**	-271.06
No. Observations:	348	**AIC:**	570.1
Df Residuals:	334	**BIC:**	624.0
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.4037	0.074	-5.487	0.000	-0.548	-0.259
GRE Score	0.5073	0.042	12.153	0.000	0.425	0.589
Research	0.2670	0.070	3.836	0.000	0.130	0.404
University Rating_5	0.3336	0.102	3.277	0.001	0.133	0.534
SOP_2.5	0.1913	0.093	2.052	0.041	0.008	0.375
SOP_3.5	0.2519	0.089	2.821	0.005	0.076	0.428
SOP_4.0	0.3027	0.102	2.975	0.003	0.103	0.503
SOP_4.5	0.3984	0.116	3.433	0.001	0.170	0.627
SOP_5.0	0.4771	0.141	3.392	0.001	0.200	0.754
LOR_1.5	-0.4680	0.189	-2.481	0.014	-0.839	-0.097
LOR_2.0	-0.3548	0.112	-3.174	0.002	-0.575	-0.135
LOR_3.0	-0.1380	0.081	-1.711	0.088	-0.297	0.021
LOR_4.5	0.2311	0.095	2.423	0.016	0.043	0.419
LOR_5.0	0.2901	0.113	2.563	0.011	0.067	0.513

Omnibus:	57.354	**Durbin-Watson:**	2.085
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	95.321
Skew:	-0.964	**Prob(JB):**	2.00e-21
Kurtosis:	4.691	**Cond. No.**	8.82

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

We can see that

- After removal of 3 features , R Score hasn't changed much
- Also, based on the p-values, we can see that Following Features should be removed : LOR_3 (0.088) , SOP_2.5 (0.041) , LOR_1.5 (0.014).

In [52]:

```
x_train_5 = x_train[['GRE Score' , 'Research' , 'University Rating_5' , 'SOP_3.5' , 'SOP_4.0' , 'SOP_4.5' , 'SOP_5.0' , 'LOR_2.0' , 'LOR_

x_train_5 = sm.add_constant(x_train_5)
LR_5 = sm.OLS(y_train , x_train_5).fit()

LR_5.summary()
```

Out[52]:

OLS Regression Results

Dep. Variable:	Chance of Admit	**R-squared:**	0.717
Model:	OLS	**Adj. R-squared:**	0.708
Method:	Least Squares	**F-statistic:**	85.29
Date:	Tue, 03 Jan 2023	**Prob (F-statistic):**	4.57e-86
Time:	23:09:57	**Log-Likelihood:**	-276.73
No. Observations:	348	**AIC:**	575.5
Df Residuals:	337	**BIC:**	617.8
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.4150	0.063	-6.584	0.000	-0.539	-0.291
GRE Score	0.5105	0.042	12.095	0.000	0.427	0.593
Research	0.2651	0.070	3.765	0.000	0.127	0.404
University Rating_5	0.3343	0.103	3.245	0.001	0.132	0.537
SOP_3.5	0.2128	0.085	2.516	0.012	0.046	0.379
SOP_4.0	0.2832	0.098	2.885	0.004	0.090	0.476
SOP_4.5	0.3840	0.113	3.398	0.001	0.162	0.606
SOP_5.0	0.4641	0.139	3.341	0.001	0.191	0.737
LOR_2.0	-0.2749	0.109	-2.518	0.012	-0.490	-0.060
LOR_4.5	0.2653	0.095	2.799	0.005	0.079	0.452
LOR_5.0	0.3170	0.113	2.799	0.005	0.094	0.540

Omnibus:	54.440	**Durbin-Watson:**	2.030
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	83.961
Skew:	-0.961	**Prob(JB):**	5.86e-19
Kurtosis:	4.449	**Cond. No.**	7.78

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

We can see that

- After removal of 3 features , R Score hasn't changed much.
- Also, based on the p-values, we can see that Following Features should be removed : SOP_4 (0.004) , SOP_3.5 (0.012) , LOR_2.0 (0.012).

In [53]:

```
x_train_6 = x_train[['GRE Score' , 'Research' , 'University Rating_5' , 'SOP_4.5' , 'SOP_5.0' , 'LOR_4.5' , 'LOR_5.0']]

x_train_6 = sm.add_constant(x_train_6)
LR_6 = sm.OLS(y_train , x_train_6).fit()

LR_6.summary()
```

Out[53]:

OLS Regression Results

Dep. Variable:	Chance of Admit	**R-squared:**	0.700
Model:	OLS	**Adj. R-squared:**	0.694
Method:	Least Squares	**F-statistic:**	113.2
Date:	Tue, 03 Jan 2023	**Prob (F-statistic):**	7.22e-85
Time:	23:09:57	**Log-Likelihood:**	-286.90
No. Observations:	348	**AIC:**	589.8
Df Residuals:	340	**BIC:**	620.6
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.3403	0.055	-6.212	0.000	-0.448	-0.233
GRE Score	0.5714	0.041	13.956	0.000	0.491	0.652
Research	0.2765	0.072	3.842	0.000	0.135	0.418
University Rating_5	0.3554	0.105	3.394	0.001	0.149	0.561
SOP_4.5	0.2120	0.102	2.080	0.038	0.012	0.412
SOP_5.0	0.2625	0.127	2.059	0.040	0.012	0.513
LOR_4.5	0.3277	0.095	3.466	0.001	0.142	0.514
LOR_5.0	0.3824	0.114	3.350	0.001	0.158	0.607

Omnibus:	48.569	**Durbin-Watson:**	2.031
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	70.338
Skew:	-0.903	**Prob(JB):**	5.32e-16
Kurtosis:	4.260	**Cond. No.**	6.10

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

We can see that

- After removal of 3 features , R Score hasn't changed much.
- Also, based on the p-values, we can see that Following Features should be removed : SOP_4.5 (0.038) , SOP_5.0 (0.040).

In [54]:

```
x_train_7 = x_train[['GRE Score' , 'Research' , 'University Rating_5' , 'LOR_4.5' , 'LOR_5.0']]

x_train_7 = sm.add_constant(x_train_7)
LR_7 = sm.OLS(y_train , x_train_7).fit()

LR_7.summary()
```

Out[54]:

OLS Regression Results

Dep. Variable:	Chance of Admit	**R-squared:**	0.694
Model:	OLS	**Adj. R-squared:**	0.689
Method:	Least Squares	**F-statistic:**	155.0
Date:	Tue, 03 Jan 2023	**Prob (F-statistic):**	1.20e-85
Time:	23:09:57	**Log-Likelihood:**	-290.25
No. Observations:	348	**AIC:**	592.5
Df Residuals:	342	**BIC:**	615.6
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.3133	0.054	-5.821	0.000	-0.419	-0.207
GRE Score	0.5970	0.040	14.999	0.000	0.519	0.675
Research	0.2777	0.072	3.836	0.000	0.135	0.420
University Rating_5	0.4363	0.101	4.340	0.000	0.239	0.634
LOR_4.5	0.3743	0.093	4.007	0.000	0.191	0.558
LOR_5.0	0.4350	0.110	3.968	0.000	0.219	0.651

Omnibus:	47.620	**Durbin-Watson:**	2.061
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	68.401
Skew:	-0.893	**Prob(JB):**	1.40e-15
Kurtosis:	4.237	**Cond. No.**	5.05

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

In [55]:

```
parameters = round(LR_7.params,4)
parameters
```

Out[55]:

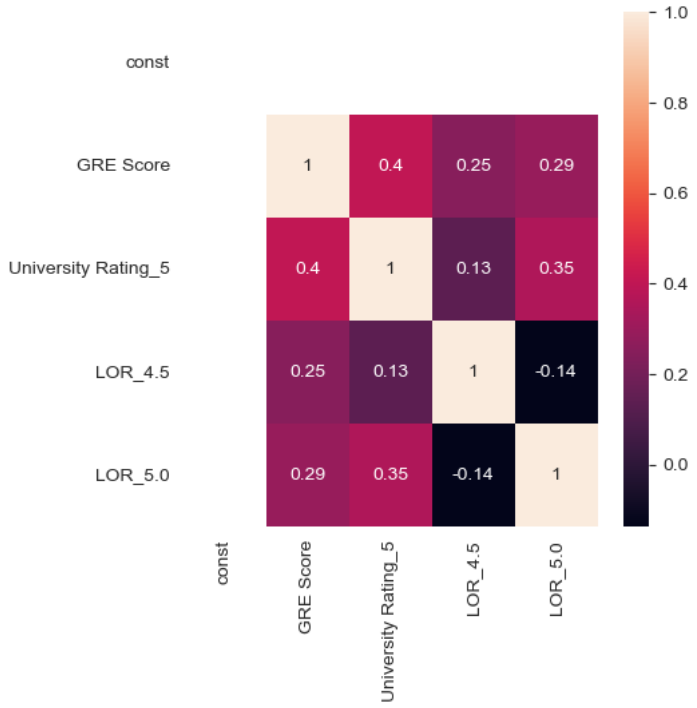
```
const                -0.3133
GRE Score             0.5970
Research              0.2777
University Rating_5   0.4363
LOR_4.5               0.3743
LOR_5.0               0.4350
dtype: float64
```

In [56]:

```
plt.figure(figsize= (5,5))
d_new = sns.heatmap(x_train_7.corr() , annot = True , annot_kws={'size': 10})
d_new
```

Out[56]:

```
<AxesSubplot:>
```



Observations :

- After removal of several Features, R - Score values haven't changed much.
- Now that the p-values for our Columns have reached 0, we can say that our Best-Fit model is attained.
- Thus , the Model coefficients attained are :

  const : -0.3133

  GRE Score : 0.5970

  Research : 0.2777

  University Rating_5 : 0.4363

  LOR_4.5 : 0.3743

  LOR_5.0 : 0.4350

Assumptions of Linear Regression :

- **Linearity**: The relationship between X and the mean of Y is linear.
- **Homoscedasticity**: The variance of residual is the same for any value of X.
- **Independence**: Observations are independent of each other.
- **Normality**: For any fixed value of X, Y is normally distributed.
- **No Autocorrelation** : Error Data shouldn't form any Patterns in its Plots.

1. Multicollinearity Check - Using VIF Score :-

In [57]:

```
vif = pd.DataFrame()
vif["VIF Values"] = [variance_inflation_factor(x_train_7.values , i) for i in range(x_train_7.shape[1])]
vif["Features"] = x_train_7.columns
vif
```

Out[57]:

	VIF Values	Features
0	3.192483	const
1	1.702381	GRE Score
2	1.426649	Research
3	1.300569	University Rating_5
4	1.143123	LOR_4.5
5	1.257986	LOR_5.0

Since all Features are below 5, means No Multicollinearity.

In [58]:

```
df_pred = df_test.copy(deep = True)
df_test.shape
```

Out[58]:

```
(150, 24)
```

In [59]:

```
y_test = df_test.pop('Chance of Admit')
x_test = df_test
x_test = sm.add_constant(x_test)
```

In [60]:

```
x_train_7.columns
```

Out[60]:

```
Index(['const', 'GRE Score', 'Research', 'University Rating_5', 'LOR_4.5',
       'LOR_5.0'],
      dtype='object')
```

In [61]:

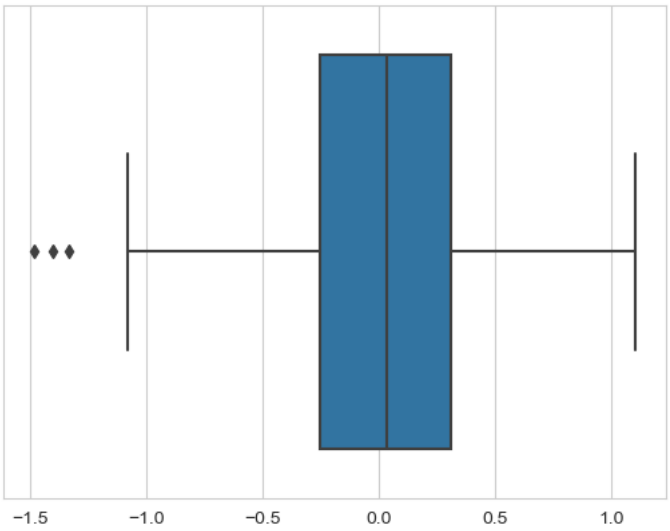
```
x_test_new = x_test[x_train_7.columns]

y_pred = LR_7.predict(x_test_new)
```

In [62]:

```
residual_test = y_test - y_pred
sns.boxplot(x = residual_test)
```

Out[62]:

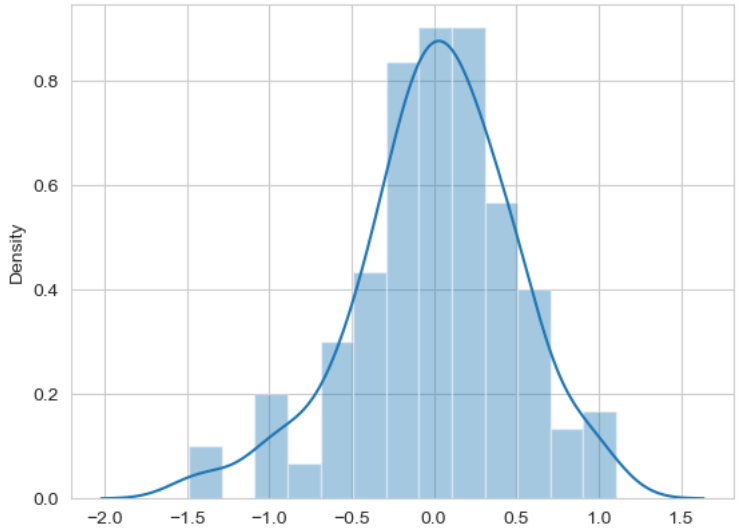
```
<AxesSubplot:>
```



2. Mean of Residuals :-

In [63]:

```
residual = y_test - y_pred
sns.distplot(residual)
plt.show()
```



In [64]:

```
abs(residual.mean())
```

Out[64]:

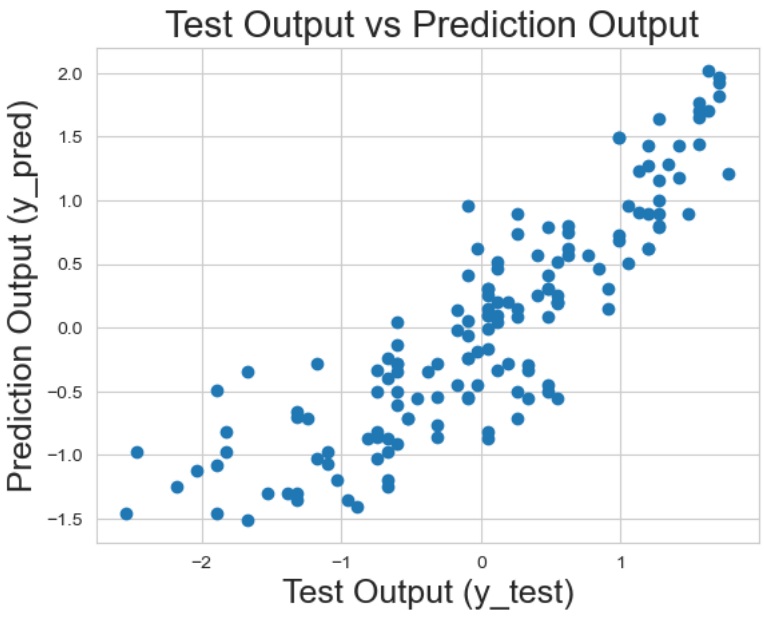
```
0.0073270043134073485
```

From this, It can be seen that our Model's Mean Residual is close to Zero , means it's a Good Model.

3. Linearity of Variables :-

In [65]:

```
plt.scatter(y_test , y_pred)
plt.title(" Test Output vs Prediction Output" , fontsize = 20)
plt.xlabel("Test Output (y_test)", fontsize = 18)
plt.ylabel("Prediction Output (y_pred)", fontsize = 18)
plt.show()
```



4. Homoscedasticity Testing :

Homoscedasticity means that the residuals have equal or almost equal variance across the regression line. By plotting the error terms with predicted terms we can check that there should not be any pattern in the error terms.

There are 2 ways -

1. By checking the Plots Graphically
2. By applying Goldfeld Quandt Test (and aiming for Null Hypothesis)

a) Graphical Approach -

In [66]:

```
fig = plt.figure(figsize=(20,20))

# Using statsmodels.graphics.regressionplots.plot_regress_exog to Plot Regression Results against one Regressor
fig = sm.graphics.plot_regress_exog(LR_7 , "GRE Score" , fig = fig)
fig.tight_layout(pad=1.0)
plt.show()
```

```
eval_env: 1
```

It can be seen that the points are spread Randomly, and Points of Residual are scattered around Zero. Hence, There is no Heteroscedasticity.

b) Goldfeld Quandt Test -

Checking Heteroscedasticity : Using Goldfeld Quandt , we test for heteroscedasticity.

- Null Hypothesis: Error terms are homoscedastic

- Alternative Hypothesis: Error terms are heteroscedastic.

In [67]:

```
import statsmodels.stats.api as sms
from statsmodels.compat import lzip

name = ['F Statistic' , 'p-value']
test = sms.het_goldfeldquandt(y_train , x_train_7)
lzip(name , test)
```

Out[67]:

```
[('F Statistic', 0.8628153111399349), ('p-value', 0.8300617615401447)]
```

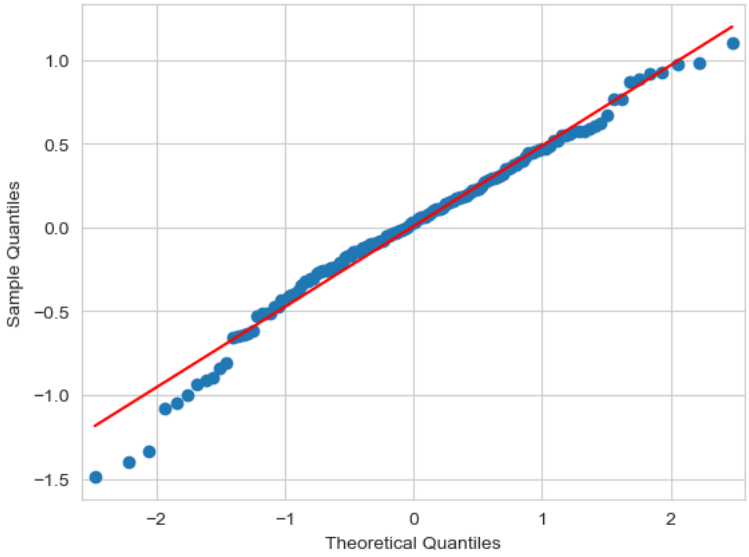
Observations :

Since the p-value from Hypothesis Testing is more than the Significance Value (0.05), this means that we fail to reject the Null Hypothesis. Thus , using both the Graphical Approach and Goldfeld Quandt Test Approach , we see that Heteroscedasticity isn't present in our Model.

5. Normality of Residues :

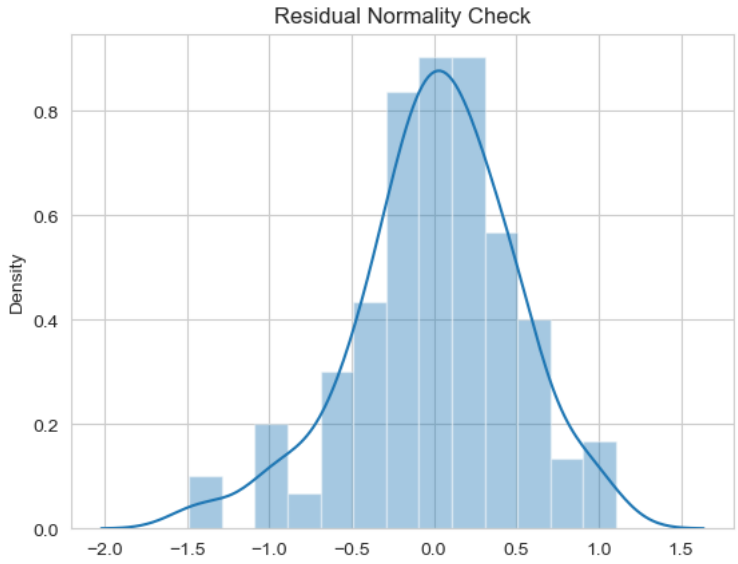
In [68]:

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



In [69]:

```
h = sns.distplot(residual , kde = True)
h = plt.title('Residual Normality Check')
```



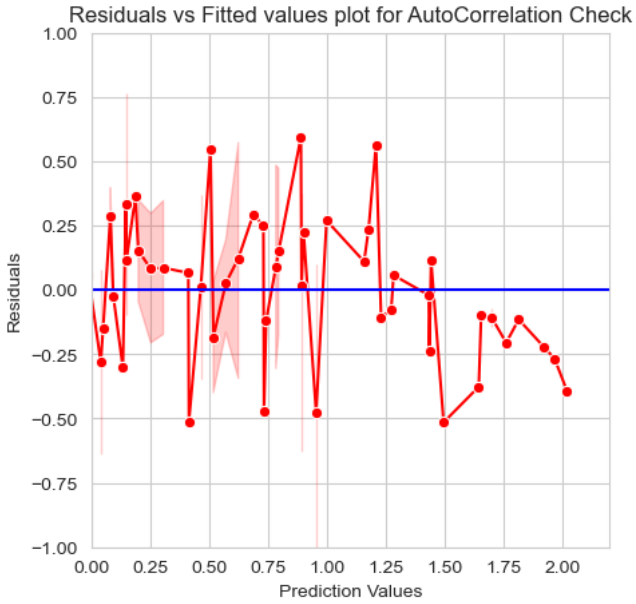
From above, it can se seen that the Data is aligned closely to the Dotted line. Hence, Residuals are Normally Distributed. The Distplot also shows that the Residual Terms are nearly Normally Distributed.

6. No Autocorrelation of Residuals :

When the residuals are autocorrelated, it means that the current value is dependent of the previous (historic) values and that there is a definite unexplained pattern in the Y variable that shows up in the error terms. There should not be autocorrelation in the data so the error terms should not form any pattern.

In [70]:

```
plt.figure(figsize = (5,5))
auto_graph = sns.lineplot(y_pred , residual , marker = 'o' , color = 'red')
plt.xlabel('Prediction Values')
plt.ylabel('Residuals')
plt.ylim(-1,1)
plt.xlim(0,2.2)
p = sns.lineplot([0,2.2],[0,0],color = 'blue')
p = plt.title('Residuals vs Fitted values plot for AutoCorrelation Check')
```



This means that No Auto-correlation exits.

Since all of the conditions are met, this means that Our Model is good to consider for Further Predictions.

Model Performance Evaluation :

In [71]:

```
r2 = r2_score(y_test , y_pred)
adj_r2 = LR_7.rsquared_adj
mae = mean_absolute_error(y_test , y_pred)
mse = mean_squared_error(y_test , y_pred)
rmse = np.sqrt(mean_squared_error(y_test , y_pred))

print('R Square Value :', round(r2,2))
print('Adjusted R Square Value :', round(adj_r2,2))
print('Mean Absolute Error Value :', round(mae,2))
print('Mean Square Error Value :', round(mse,2))
print('Root Mean Square Error Value :', round(rmse,2))
```

```
R Square Value : 0.76
Adjusted R Square Value : 0.69
Mean Absolute Error Value : 0.37
Mean Square Error Value : 0.23
Root Mean Square Error Value : 0.48
```

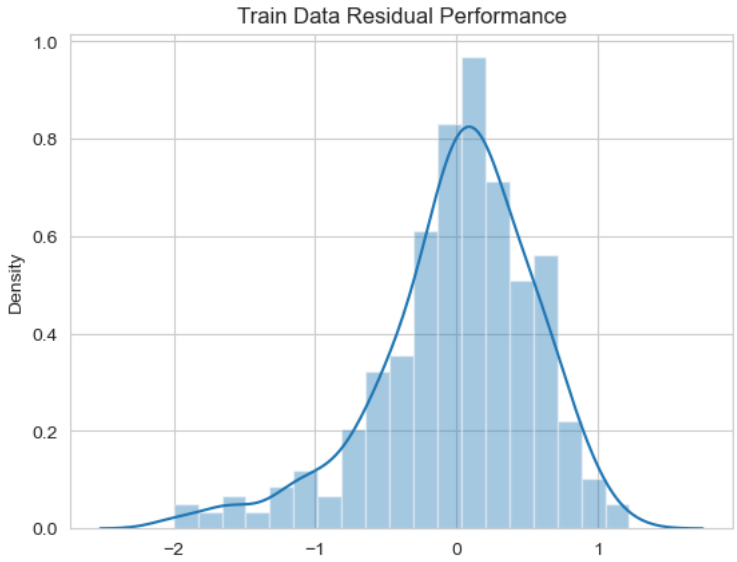
Observations :

- As r2 metric value is 0.76, this means that our model is a Good Model, since r2 value is neither too good nor too low.
- Since MAE error (0.37) is on the lower side , this means that Our Model is Good but not the Perfect Model.
- Since difference between MAE error (0.37) and RMSE error (0.48) is on the Lower Side , this shows Less Outliers.

Test and Train Performances :

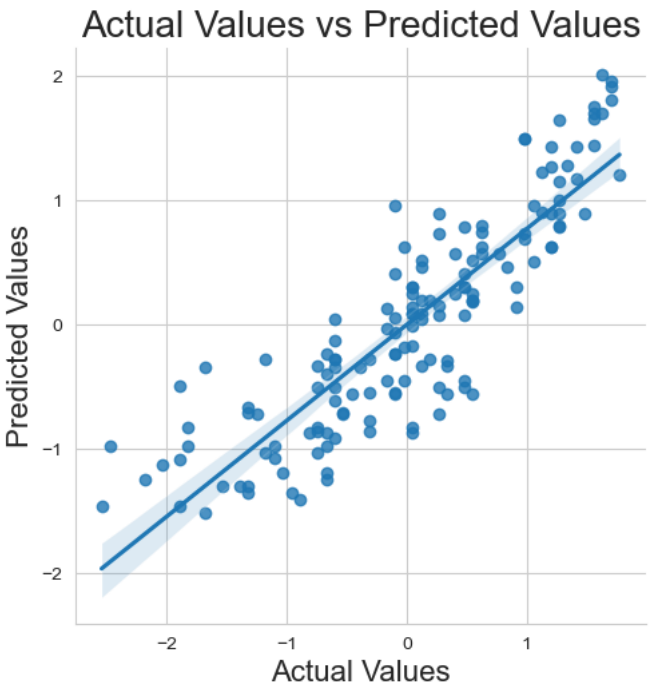
In [72]:

```
# Train Performance

y_train_pred = LR_7.predict(x_train_7)
res = y_train - y_train_pred

sns.distplot(res)
plt.title("Train Data Residual Performance")
plt.show()
```



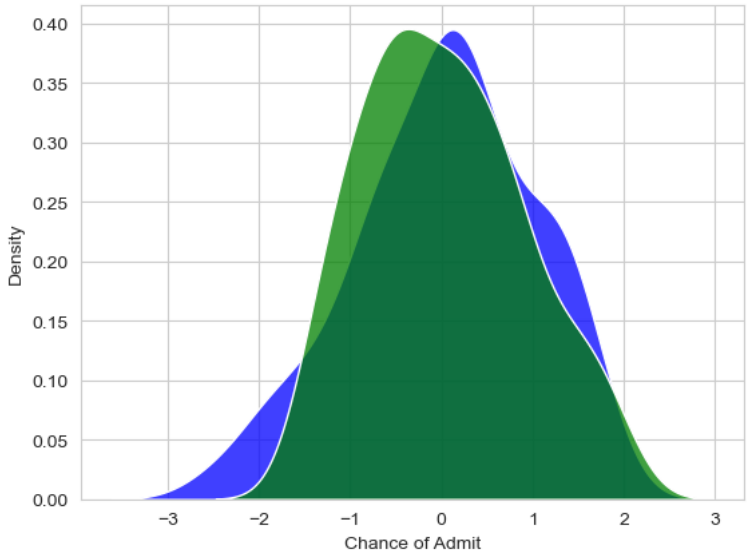
In [73]:

```
# Test Performance

df_pred['predicted'] = y_pred

sns.lmplot(x = 'Chance of Admit' , y = 'predicted' , data = df_pred)
plt.xlabel('Actual Values' , fontsize = 16)
plt.ylabel('Predicted Values' , fontsize = 16)
plt.title('Actual Values vs Predicted Values' , fontsize = 20)
plt.show()
```



In [74]:

```
sns.kdeplot(data = df_pred , x = "Chance of Admit" , color = 'b' , multiple = 'stack' , label = 'Actual Values')
sns.kdeplot(data = df_pred , x = 'predicted' , color = 'g' , multiple = 'stack' , label = 'Predicted Values')
plt.show()
```



Observations :

These above plots show that our Model can be considered for our Purpose.

Lasso Regression or L1 - Regularization :

Lasso regression performs L1 regularization, which adds a penalty equal to the absolute value of the magnitude of coefficients. This type of regularization can result in sparse models with few coefficients; Some coefficients can become zero and eliminated from the model. Larger penalties result in coefficient values closer to zero, which is the ideal for producing simpler models.

In [75]:

```
from sklearn.linear_model import Lasso
```

In [76]:

```
regressor_lasso = Lasso(alpha = 10.0)
regressor_lasso.fit(x_train_7 , y_train)
```

Out[76]:

```
Lasso(alpha=10.0)
```

In [77]:

```
print(regressor_lasso.intercept_)
print(regressor_lasso.coef_)
print(x_train_7.columns)
```

```
-0.014225386554957031
[0. 0. 0. 0. 0. 0.]
Index(['const', 'GRE Score', 'Research', 'University Rating_5', 'LOR_4.5',
       'LOR_5.0'],
      dtype='object')
```

In [78]:

```
# Now we can find out which Features are Important and which aren't Important Features

importance = regressor_lasso.coef_
features = x_train_7.columns

print(np.array(features)[importance != 0])
print(np.array(features)[importance == 0])
```

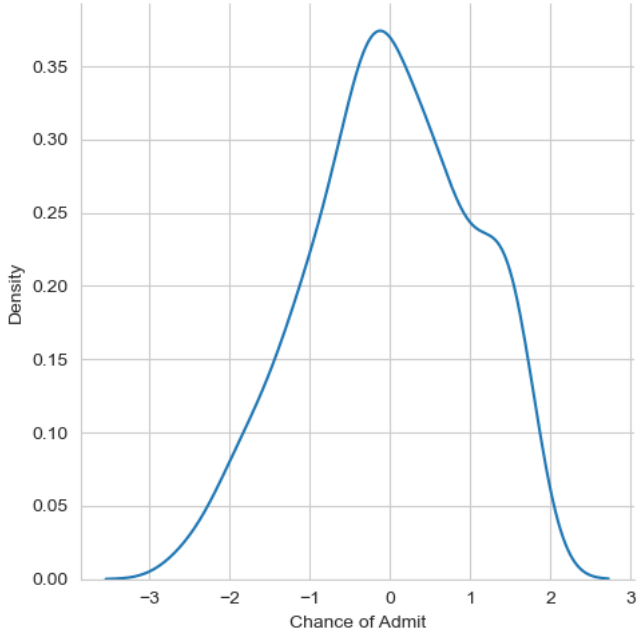
```
[]
['const' 'GRE Score' 'Research' 'University Rating_5' 'LOR_4.5' 'LOR_5.0']
```

In [79]:

```
# Residual Analysis on Training Data for L1 Regularization

y_train_pred_lasso = regressor_lasso.predict(x_train_7)
residual_lasso = y_train - y_train_pred_lasso
sns.displot(residual_lasso, kind='kde')
```

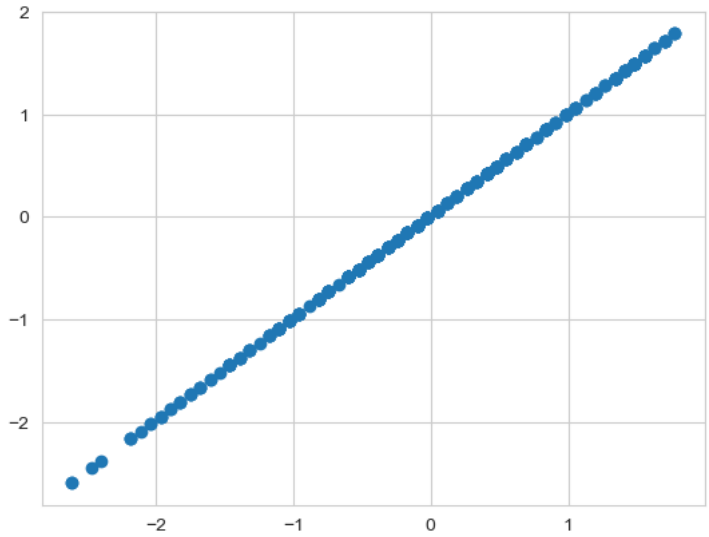
Out[79]:

```
<seaborn.axisgrid.FacetGrid at 0x134ce619be0>
```



In [80]:

```
plt.scatter(y_train, residual_lasso)
plt.show()
```



In [81]:

```
x_test = x_test[['const' , 'GRE Score' , 'Research', 'University Rating_5', 'LOR_4.5', 'LOR_5.0']]
x_test
```

Out[81]:

	const	GRE Score	Research	University Rating_5	LOR_4.5	LOR_5.0
104	1.0	0.840753	1	0	0	0
137	1.0	-0.048763	1	0	0	0
141	1.0	1.374462	1	0	0	0
461	1.0	-1.383036	1	0	0	0
354	1.0	-1.738842	0	0	0	0
...	...	...	...	...	...	...
394	1.0	1.107608	1	0	0	0
457	1.0	-1.916745	0	0	0	0
75	1.0	1.107608	1	0	0	0
471	1.0	-0.493520	0	0	0	0
153	1.0	0.662850	0	0	0	0

150 rows × 6 columns

In [82]:

```
# Prediction for L1

y_test_pred_lasso = regressor_lasso.predict(x_test)
temp_df_L1 = pd.DataFrame({'Actual': y_test, 'Predicted': y_test_pred_lasso})
temp_df_L1.head()
```

Out[82]:

	Actual	Predicted
104	0.120176	-0.014225
137	-0.095362	-0.014225
141	1.269708	-0.014225
461	-0.310899	-0.014225
354	-0.957511	-0.014225

In [83]:

```
from sklearn import metrics
print('Mean Absolute Error: ', metrics.mean_absolute_error(y_test, y_test_pred_lasso))

print('Mean Squared Error: ', metrics.mean_squared_error(y_test, y_test_pred_lasso))

print('Root Mean Squared Error: ', np.sqrt(metrics.mean_squared_error(y_test, y_test_pred_lasso)))
```

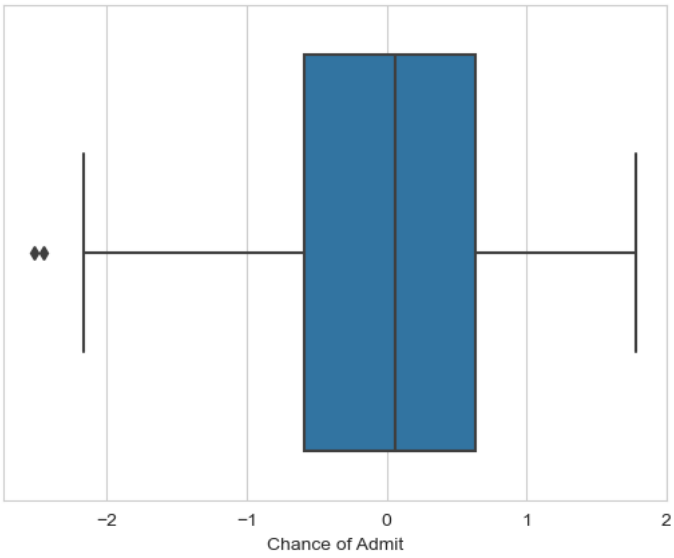
```
Mean Absolute Error:  0.788354754837149
Mean Squared Error:  0.9678748524595752
Root Mean Squared Error:  0.9838063084060679
```

In [84]:

```
residual_test_lasso = y_test - y_test_pred_lasso
sns.boxplot(x = residual_test_lasso)
```

Out[84]:

```
<AxesSubplot:xlabel='Chance of Admit'>
```



Ridge Regression or L2 - Regularization :

L2 regularization adds an L2 penalty, which equals the square of the magnitude of coefficients. All coefficients are shrunk by the same factor (so none are eliminated). Unlike L1 regularization, L2 will not result in sparse models.

In [85]:

```
from sklearn.linear_model import Ridge
```

In [86]:

```
regressor_ridge = Ridge(alpha = 10.0)
regressor_ridge.fit(x_train_7 , y_train)
```

Out[86]:

```
Ridge(alpha=10.0)
```

In [87]:

```
print(regressor_ridge.intercept_)
print(regressor_ridge.coef_)
print(x_train_7.columns)
```

```
-0.2778539126089142
[0.         0.60602009 0.2664116  0.37846057 0.29535884 0.33780735]
Index(['const', 'GRE Score', 'Research', 'University Rating_5', 'LOR_4.5',
       'LOR_5.0'],
      dtype='object')
```

In [88]:

```
# Now we can find out which Features are Important and which aren't Important Features

importance = regressor_ridge.coef_
features = x_train_7.columns

print(np.array(features)[importance != 0])
print(np.array(features)[importance == 0])
```

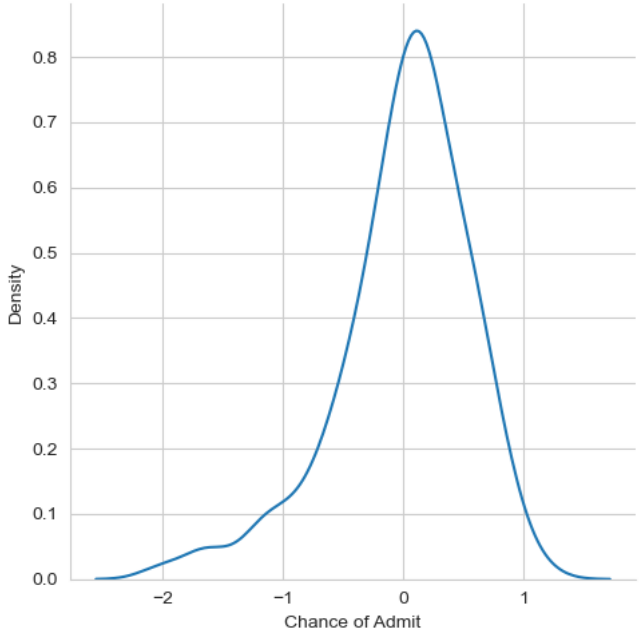
```
['GRE Score' 'Research' 'University Rating_5' 'LOR_4.5' 'LOR_5.0']
['const']
```

In [89]:

```
# Residual Analysis on Training Data for L2 Regularization

y_train_pred_ridge = regressor_ridge.predict(x_train_7)
residual_ridge = y_train - y_train_pred_ridge
sns.displot(residual_ridge, kind='kde')
```

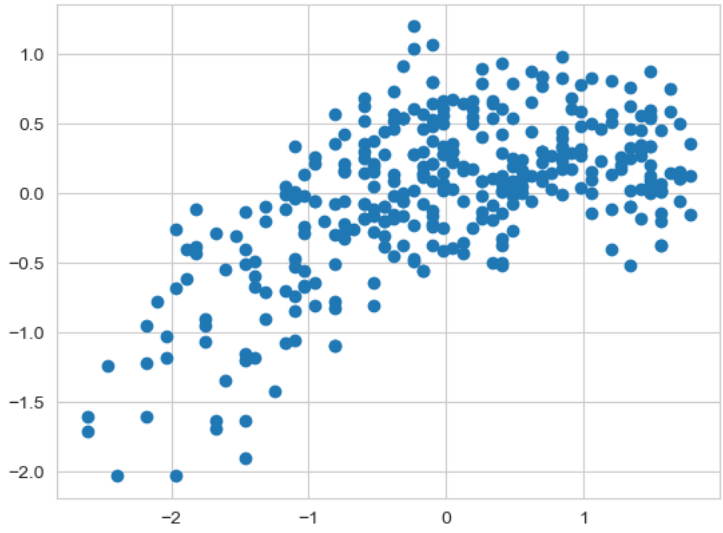
Out[89]:

```
<seaborn.axisgrid.FacetGrid at 0x134ce87dd00>
```



In [90]:

```
plt.scatter(y_train, residual_ridge)
plt.show()
```



In [91]:

```
# Prediction for L2

y_test_pred_ridge = regressor_ridge.predict(x_test)
temp_df_L2 = pd.DataFrame({'Actual': y_test, 'Predicted': y_test_pred_ridge})
temp_df_L2.head()
```

Out[91]:

	Actual	Predicted
104	0.120176	0.498071
137	-0.095362	-0.040993
141	1.269708	0.821509
461	-0.310899	-0.849590
354	-0.957511	-1.331627

In [92]:

```
from sklearn import metrics
print('Mean Absolute Error: ', metrics.mean_absolute_error(y_test, y_test_pred_ridge))

print('Mean Squared Error: ', metrics.mean_squared_error(y_test, y_test_pred_ridge))

print('Root Mean Squared Error: ', np.sqrt(metrics.mean_squared_error(y_test, y_test_pred_ridge)))
```

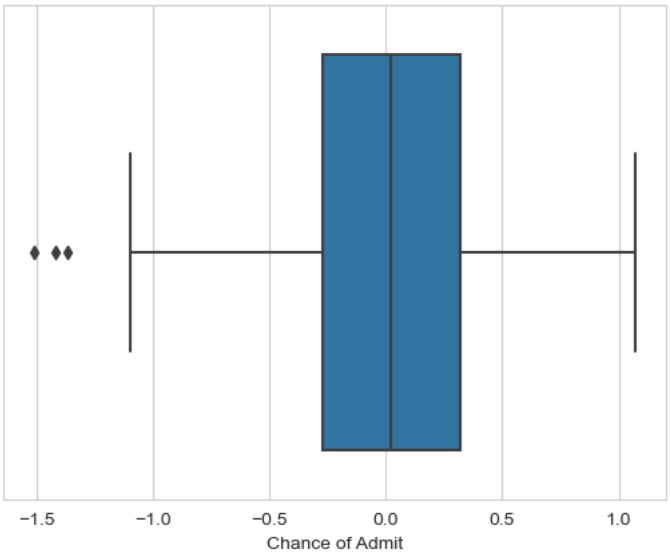
```
Mean Absolute Error:  0.36177107496879285
Mean Squared Error:  0.22909562973605413
Root Mean Squared Error:  0.478639352473294
```

In [93]:

```
residual_test_ridge = y_test - y_test_pred_ridge
sns.boxplot(x = residual_test_ridge)
```

Out[93]:

```
<AxesSubplot:xlabel='Chance of Admit'>
```



Comparison for L1 , L2 Regularizations :

In [94]:

```
R_L1 = r2_score(y_test , y_test_pred_lasso)
R_L2 = r2_score(y_test , y_test_pred_ridge)
print(R_L1)
print(R_L2)
```

```
-0.0023098677774202603
0.7627536145136905
```

In [95]:

```
print('Mean Absolute Error (L1 Regularization) : ', metrics.mean_absolute_error(y_test, y_test_pred_lasso))

print('Mean Squared Error (L1 Regularization) : ', metrics.mean_squared_error(y_test, y_test_pred_lasso))

print('Root Mean Squared Error (L1 Regularization) : ', np.sqrt(metrics.mean_squared_error(y_test, y_test_pred_lasso)))
```

```
Mean Absolute Error (L1 Regularization) :  0.788354754837149
Mean Squared Error (L1 Regularization) :  0.9678748524595752
Root Mean Squared Error (L1 Regularization) :  0.9838063084060679
```

In [96]:

```
print('Mean Absolute Error (L2 Regularization) : ', metrics.mean_absolute_error(y_test, y_test_pred_ridge))

print('Mean Squared Error (L2 Regularization) : ', metrics.mean_squared_error(y_test, y_test_pred_ridge))

print('Root Mean Squared Error (L2 Regularization) : ', np.sqrt(metrics.mean_squared_error(y_test, y_test_pred_ridge)))
```

```
Mean Absolute Error (L2 Regularization) :  0.36177107496879285
Mean Squared Error (L2 Regularization) :  0.22909562973605413
Root Mean Squared Error (L2 Regularization) :  0.478639352473294
```

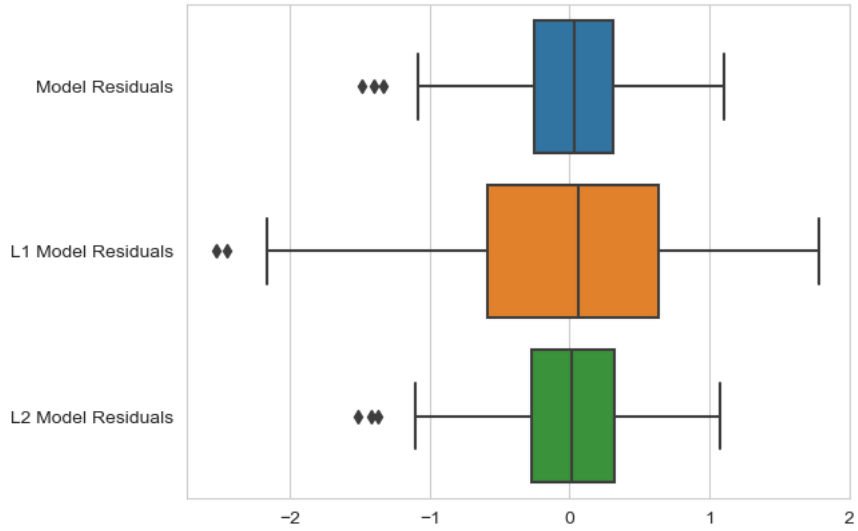
In [97]:

```
res_df = pd.DataFrame({'Model Residuals': residual_test, 'L1 Model Residuals': residual_test_lasso , 'L2 Model Residuals': residual_test_r
print(res_df)
print("Mean Residual without Regularization :", res_df['Model Residuals'].mean())
print("L1 Regularization Mean Residual :", res_df['L1 Model Residuals'].mean())
print("L2 Regularization Mean Residual :", res_df['L2 Model Residuals'].mean())
sns.boxplot(data = res_df, orient='h')
```

```
     Model Residuals  L1 Model Residuals  L2 Model Residuals
104         -0.346142            0.134401           -0.377895
137         -0.030632           -0.081136           -0.054368
141          0.484763            1.283934            0.448199
461          0.550401           -0.296674            0.538691
354          0.393930           -0.943286            0.374116
..                ...                 ...                 ...
394          0.572231            1.212088            0.538072
457         -1.080468           -2.523893           -1.098678
75          -0.649147           -0.009290           -0.683306
471          0.009693           -0.584057           -0.021345
153          0.397019            0.493630            0.355558

[150 rows x 3 columns]
Mean Residual without Regularization : 0.0073270043134073485
L1 Regularization Mean Residual : 0.04722828336245764
L2 Regularization Mean Residual : 0.004336025275475097
```

Out[97]:

```
<AxesSubplot:>
```



Observations :

- Since the Error Values for L2 Regularization seems lesser than those of L1 Regularization, and the r2 metric for L1 Regularization is extremely bad, thus L2 Regularization should be considered here.
- As per L1 Regularization , all the columns seem to be irrelevant , but in L2 Regularization, only the column 'const' seem unimportant.

Insights :

- Considering the University Rating aspect , Maximum Students were enrolled in the Universities having Ratings of 3.
- Considering the SOP aspect , Maximum Students had a SOP Ratings of 3 and 4.
- Considering the LOR aspect , Maximum Students had a LOR Rating of 3.
- Considering the Research aspect , More number of Students had a Prior Research Experience.
- Numerical Columns like GRE Scores , TOEFL Scores and CGPA are following a Linear Relationship with respect to Chances of Admit.
- As per the Pairplot for Research , Students considering themselves towards Research Domain have higher Chances of getting Admission, in comparison to those who don't consider Research into consideration.
- As per the Pairplot for University Ranking , Students require better GRE , TOEFL and CGPA Scores to get admission into better Universities. Considering the Data , Maximum number of Students got admissions in Universities with Ratings 3. And minimum Number of Students got admission in Universities with Ratings 1.
- Column 'Chance of Admit' has a Positive Linear Relation with all of the Numerical and Categorical Features.
- Chance of Admit vs Research :

  Having Research has more Median Value compared to not having Research. Also, Research Students have higher chances than Non-Researchers. Outliers are present only for Research Aspect.

- Chance of Admit vs University Rating :

  As the University Rating increases, the chances of getting admission also Increases, since Rating '5' has the highest Median value followed by 4 , 3 , 2 and 1. Outliers present for most of the Categories.

- Chance of Admit vs SOP :

  As the SOP Rating increases, the chances of getting admission also Increases, since Rating '5' has the highest Median value followed by others. Outliers present for most of the Categories.

- Chance of Admit vs LOR :

  As the LOR Rating increases, the chances of getting admission also Increases, since Rating '5' has the highest Median value followed by others. Outliers present for most of the Categories.
- Thus , the Model coefficients attained are :

  const : -0.3133

  GRE Score : 0.5970

  Research : 0.2777

  University Rating_5 : 0.4363

  LOR_4.5 : 0.3743

  LOR_5.0 : 0.4350

  Means that GRE Score has the Highest Impact on the chances of any Student getting Admission into any University.

- Our Final Model is passing all the Assumptions for Linear Regression correctly, thus the Metric Values for our Model are :

  R Square Value : 0.76 , Adjusted R Square Value : 0.69 , Mean Absolute Error Value : 0.37 , Mean Square Error Value : 0.23 , and Root Mean Square Error Value : 0.48.
- Metric Values for L1 Regularization are :

  R Square Value : -0.0023 , Mean Absolute Error Value : 0.78 , Mean Square Error Value : 0.96 , and Root Mean Square Error Value : 0.98.

- Metric Values for L2 Regularization are :

  R Square Value : 0.76 , Mean Absolute Error Value : 0.36 , Mean Square Error Value : 0.22 , and Root Mean Square Error Value : 0.47.

  Thus , L2 Regularization should be preferred over L1 Regularization for our Model.

Recommendations :

- Main features which influence the chance of Admit are: GRE Score , Research , TOEFL Score , CGPA , LOR greater or equal to than 4.5. For any Applicant , GRE Score should be the Primary Target.
- A Higher University rating will increases the chance of admission
- A Higher Value of LOR and SOP will also increase the chance of admission for the student.