Jamboree

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 :

- 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.

Column Profiling:

```
Serial No. (Unique row ID)
GRE Scores (out of 340)
TOEFL Scores (out of 120)
University Rating (out of 5)
Statement of Purpose and Letter of Recommendation Strength (out of 5)
Undergraduate GPA (out of 10)
Research Experience (either 0 or 1)
Chance of Admit (ranging from 0 to 1)
```

Concept Used:

- Exploratory Data Analysis
- Linear Regression

1. Importing and Data Analysis

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import figure
import warnings
warnings.filterwarnings('ignore')
import statsmodels.api as sm
```

In [2]:
```
data = pd.read_csv("https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/839/original/Jamboree_Admission.csv")
```

In [3]:
```
data.sample(5)
```

Out[3]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
423	424	334	119	5	4.5	5.0	9.54	1	0.94
129	130	333	118	5	5.0	5.0	9.35	1	0.92
485	486	311	101	2	2.5	3.5	8.34	1	0.70
253	254	335	115	4	4.5	4.5	9.68	1	0.93
166	167	302	102	3	3.5	5.0	8.33	0	0.65

In [4]:
```
data.shape
```

Out[4]:
```
(500, 9)
```

In [5]:
```
df = data.copy()
```

Dropping first column as it is not required column "Serial No."

In [6]:
```
df.drop(["Serial No."],axis=1,inplace=True)
```

In [7]:
```
# null values check
df.isna().sum()
```

Out[7]:
```
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   GRE Score          500 non-null    int64  
 1   TOEFL Score        500 non-null    int64  
 2   University Rating  500 non-null    int64  
 3   SOP                500 non-null    float64
 4   LOR                500 non-null    float64
 5   CGPA               500 non-null    float64
 6   Research           500 non-null    int64  
 7   Chance of Admit    500 non-null    float64
dtypes: float64(4), int64(4)
memory usage: 31.4 KB
```

No null values detected

In [9]:
```
df.nunique()
```

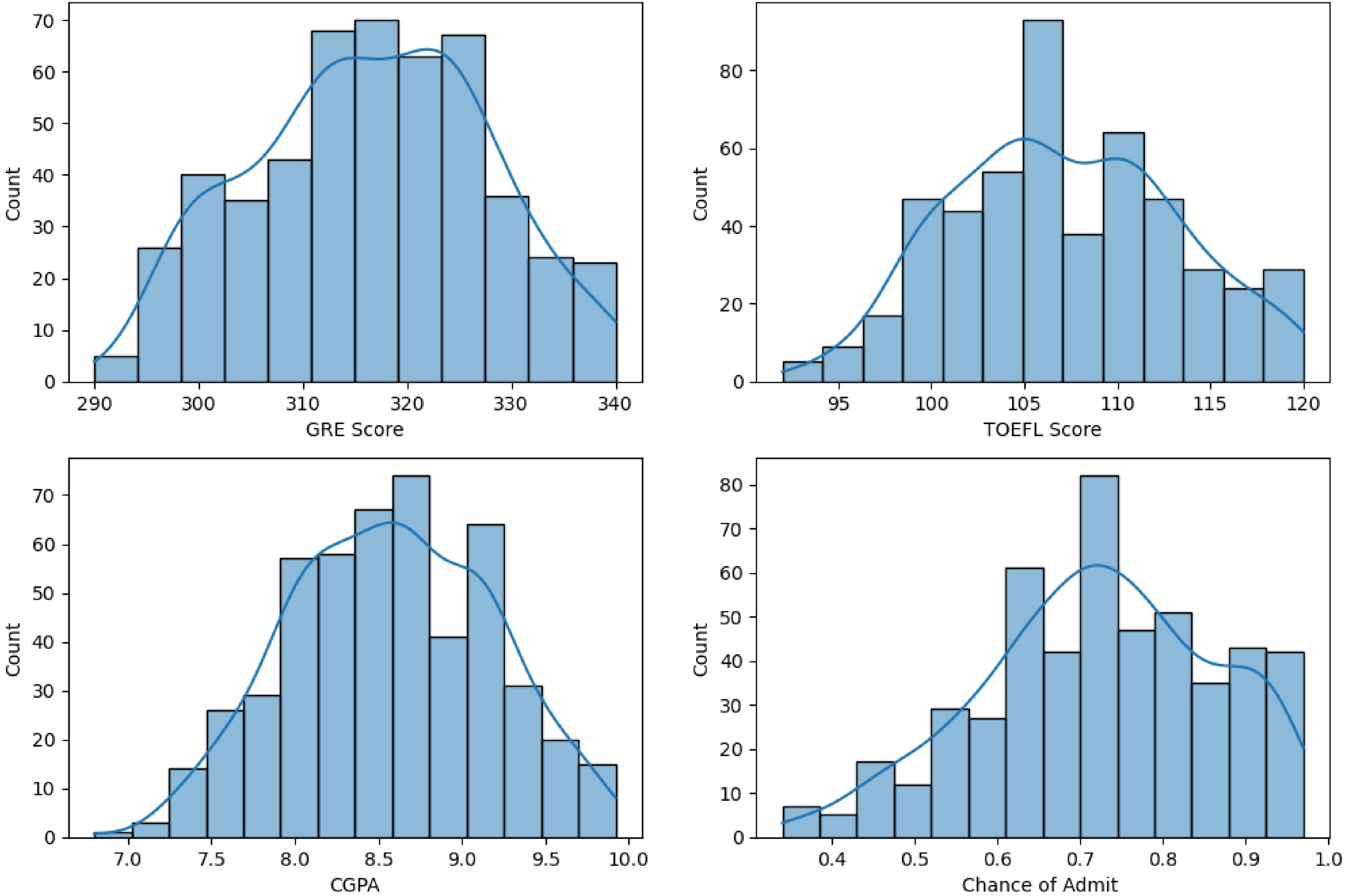
Out[9]:
```
GRE Score             49
TOEFL Score           29
University Rating      5
SOP                    9
LOR                    9
CGPA                 184
Research               2
Chance of Admit       61
dtype: int64
```

University Rating,SOP,LOR,Research are seems to be categorical variables as the number of unique values are very small. rest of the features are numeric , and ordinal . (University Rating,SOP,LOR,Research are discrete) and rest are continuous also if SOP , University rating , LOR and research can be considered as numeric ordinal data.

2. Univariate Analysis

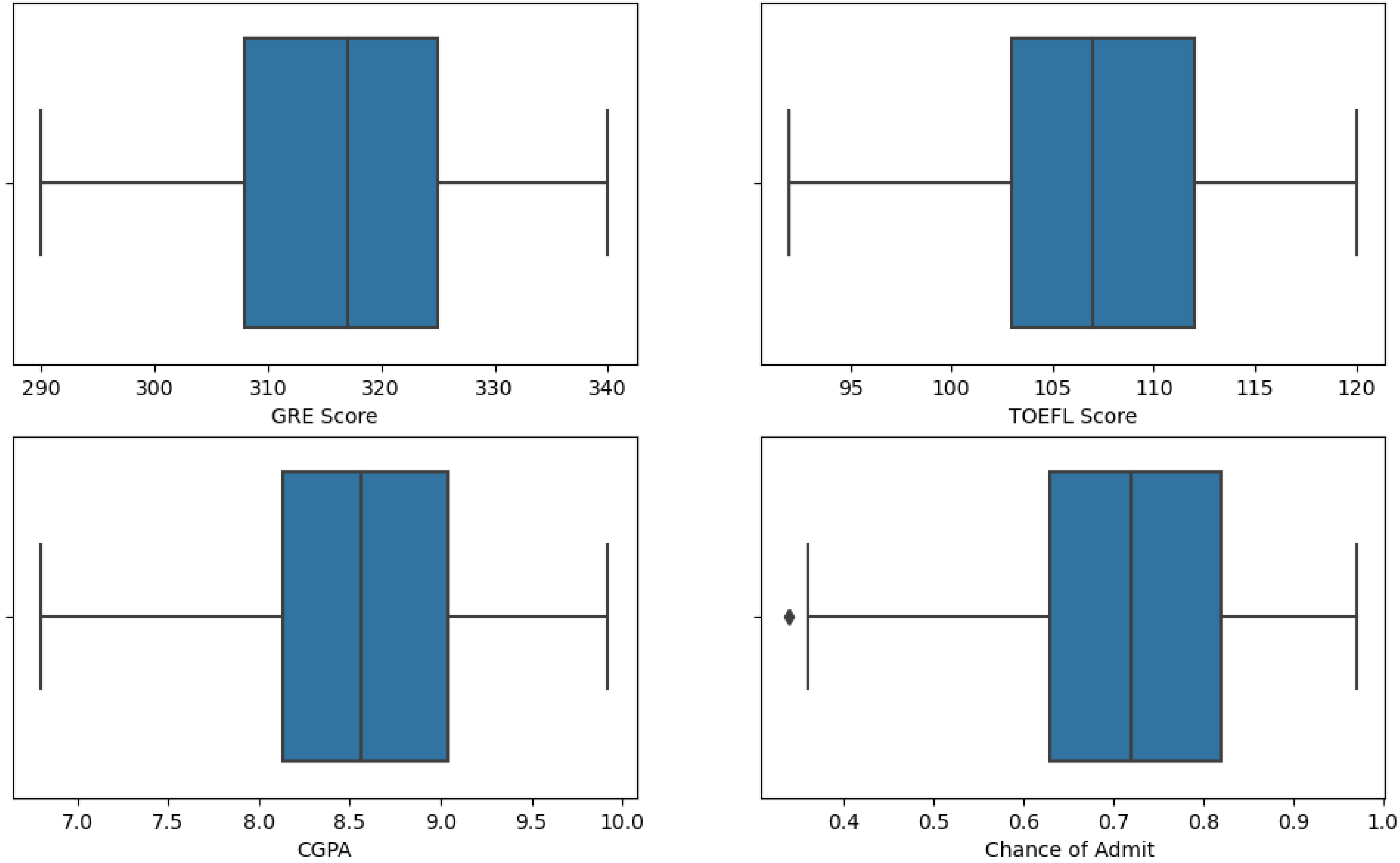
In [10]:
```
cat_cols = ['University Rating', 'SOP', 'LOR ', 'Research']
num_cols = ['GRE Score', 'TOEFL Score', 'CGPA']
target = 'Chance of Admit '
```

In [11]:
```
rows, cols = 2, 2
fig, axs = plt.subplots(rows,cols, figsize=(12, 8))
index = 0
for row in range(rows):
    for col in range(cols):
        sns.histplot(df[num_cols[index]], kde=True, ax=axs[row,col])
        index += 1
    break

sns.histplot(df[num_cols[-1]], kde=True, ax=axs[1,0])
sns.histplot(df[target], kde=True, ax=axs[1,1])
plt.show()
```



Check for outliers using boxplots

In [12]:
```
rows, cols = 2, 2
fig, axs = plt.subplots(rows, cols, figsize=(12, 7))

index = 0
for col in range(cols):
    sns.boxplot(x=num_cols[index], data=df, ax=axs[0,index])
    index += 1

sns.boxplot(x=num_cols[-1], data=df, ax=axs[1,0])
sns.boxplot(x=target, data=df, ax=axs[1,1])
plt.show()
```



There are no outliers present in the dataset.

Check unique values in categorical variables

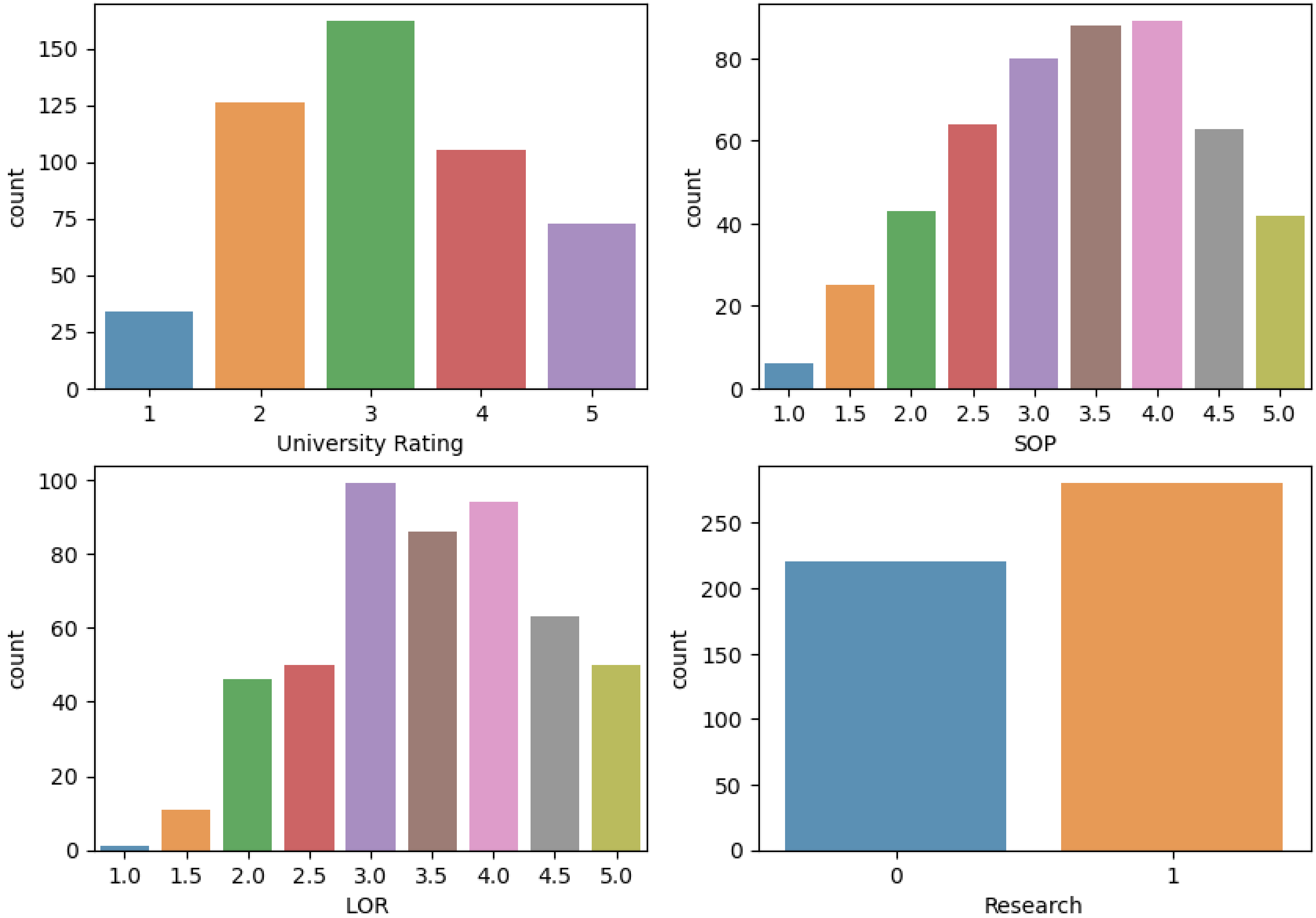
In [13]:
```
for col in cat_cols:
    print("Column:  {:18}   Unique values: {}".format(col, df[col].nunique()))
```

```
Column:  University Rating    Unique values: 5
Column:  SOP                  Unique values: 9
Column:  LOR                  Unique values: 9
Column:  Research             Unique values: 2
```

Countplots for categorical variables

In [14]:

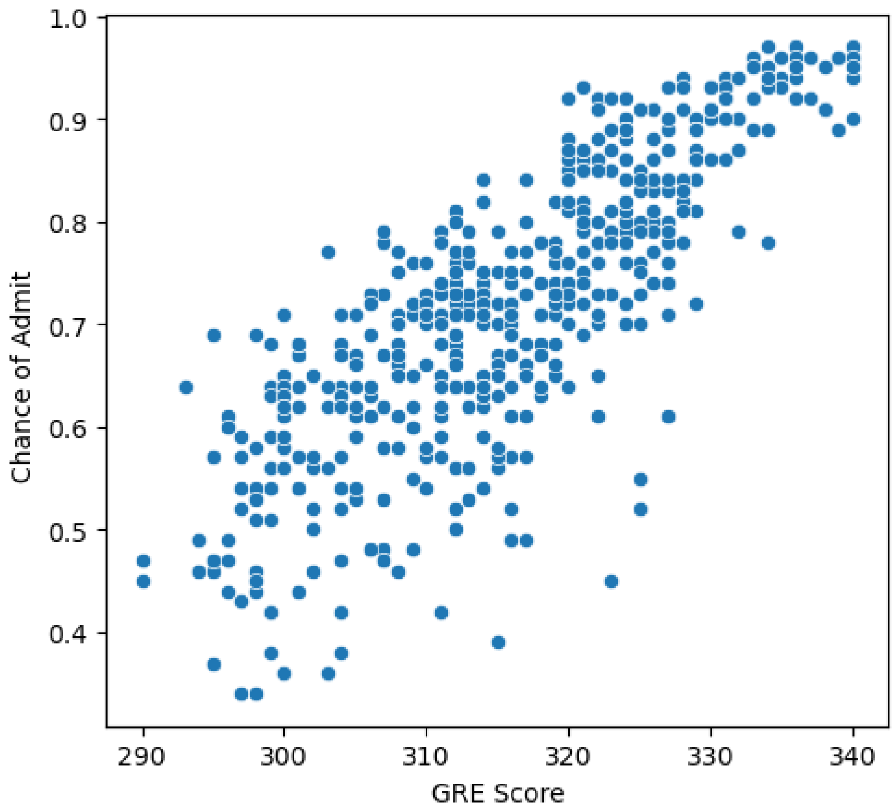
```
cols, rows = 2, 2
fig, axs = plt.subplots(rows, cols, figsize=(10, 7))

index = 0
for row in range(rows):
    for col in range(cols):
        sns.countplot(x=cat_cols[index], data=df, ax=axs[row, col], alpha=0.8)
        index += 1

plt.show()
```

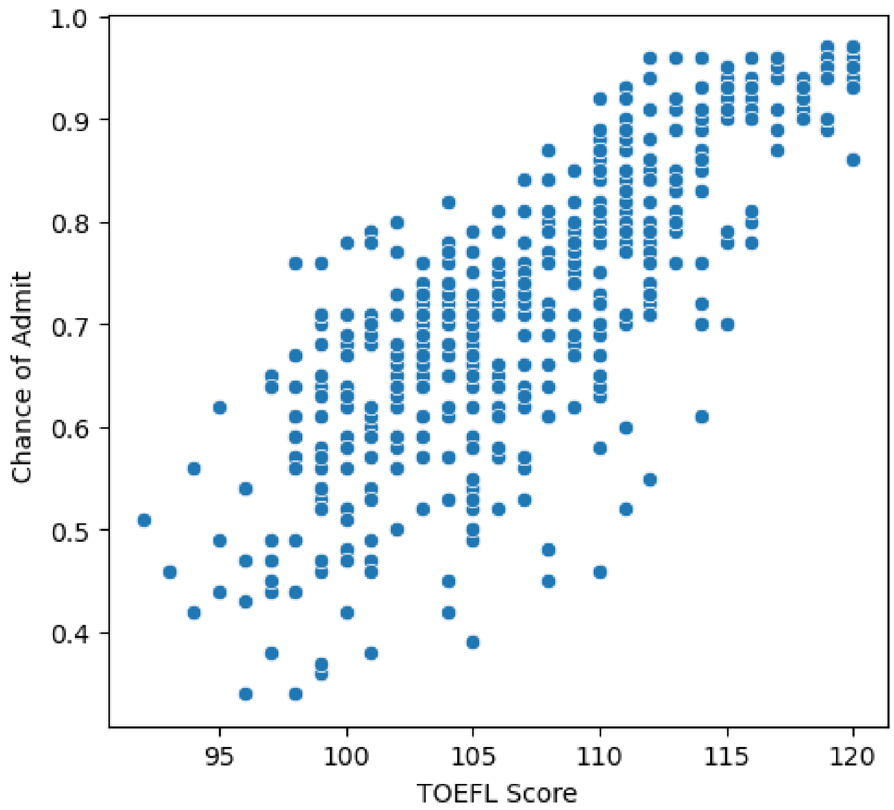


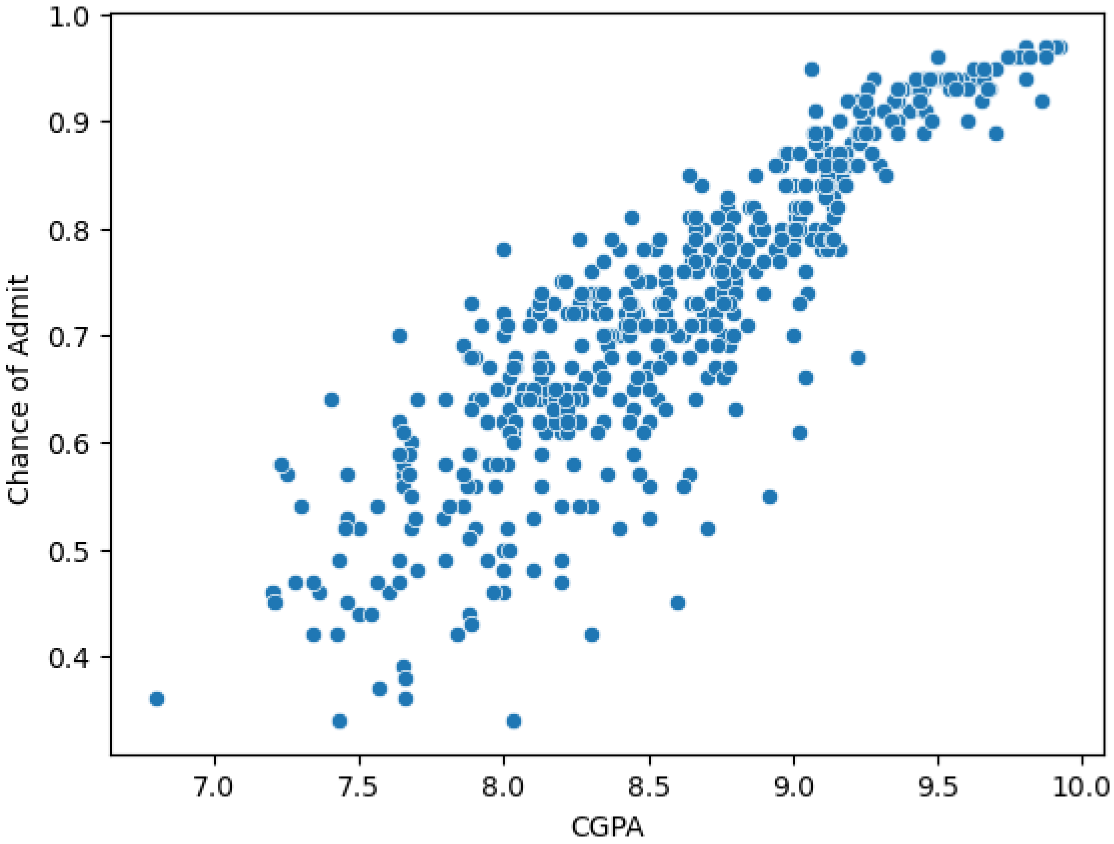
3. Bivariate Analysis

Check relation between continuous variables & target variable

In [15]:
```
fig, axs = plt.subplots(1, 2, figsize=(12,5))

sns.scatterplot(x=num_cols[0], y=target, data=df, ax=axs[0])
sns.scatterplot(x=num_cols[1], y=target, data=df, ax=axs[1])
plt.show()
sns.scatterplot(x=num_cols[2], y=target, data=df)
plt.show()
```

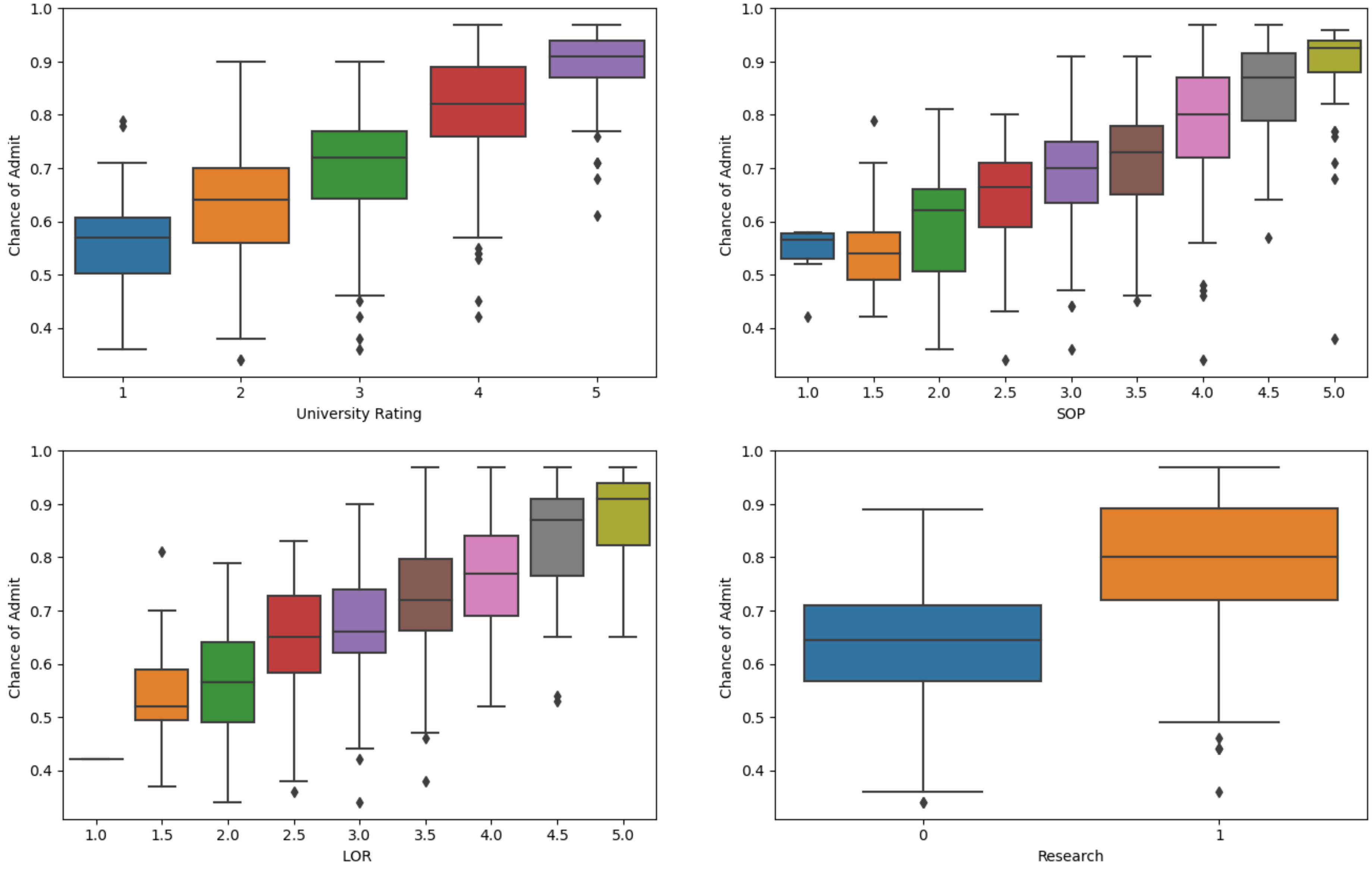






Seems like there is a linear correlation between the continuous variables and the target variable.

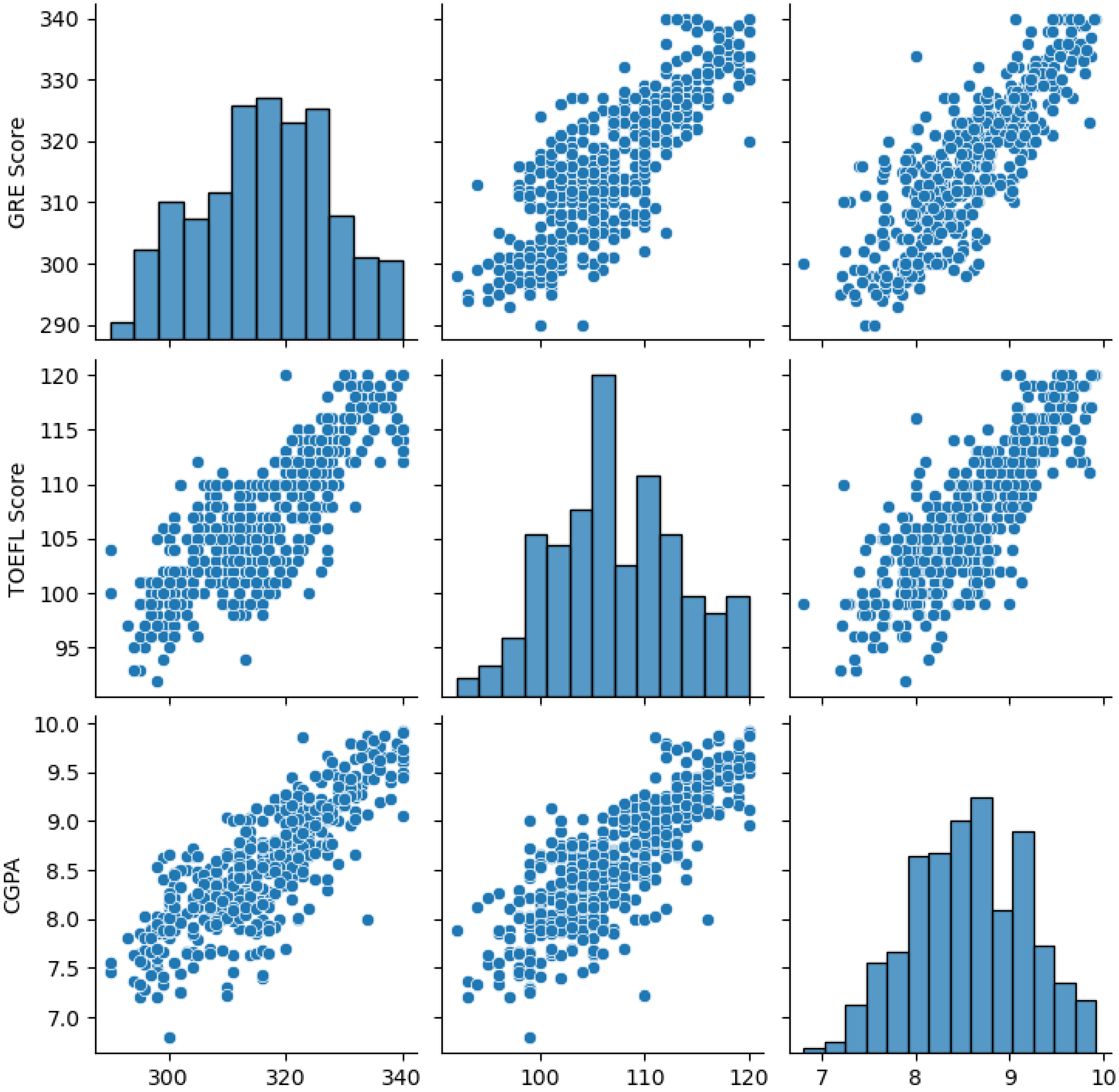
In [16]:
```
rows, cols = 2,2
fig, axs = plt.subplots(rows, cols, figsize=(16,10))

index = 0
for row in range(rows):
    for col in range(cols):
        sns.boxplot(x=cat_cols[index], y=target, data=df, ax=axs[row,col])
        index += 1
```



- As you can see from the graphs, as tge rating increases the Chance of Admit also increases.
- Students who have the research experience have more chances of Admin as compared to other students who don't have the research experience.

4. Multivariate Analysis

In [17]:
```
sns.pairplot(df[num_cols])
plt.show()
```


GRE Score
TOEFL Score
CGPA
340
330
320
310
300
290
120
115
110
105
100
95
10.0
9.5
9.0
8.5
8.0
7.5
7.0
300
320
340
100
110
120
7
8
9
10

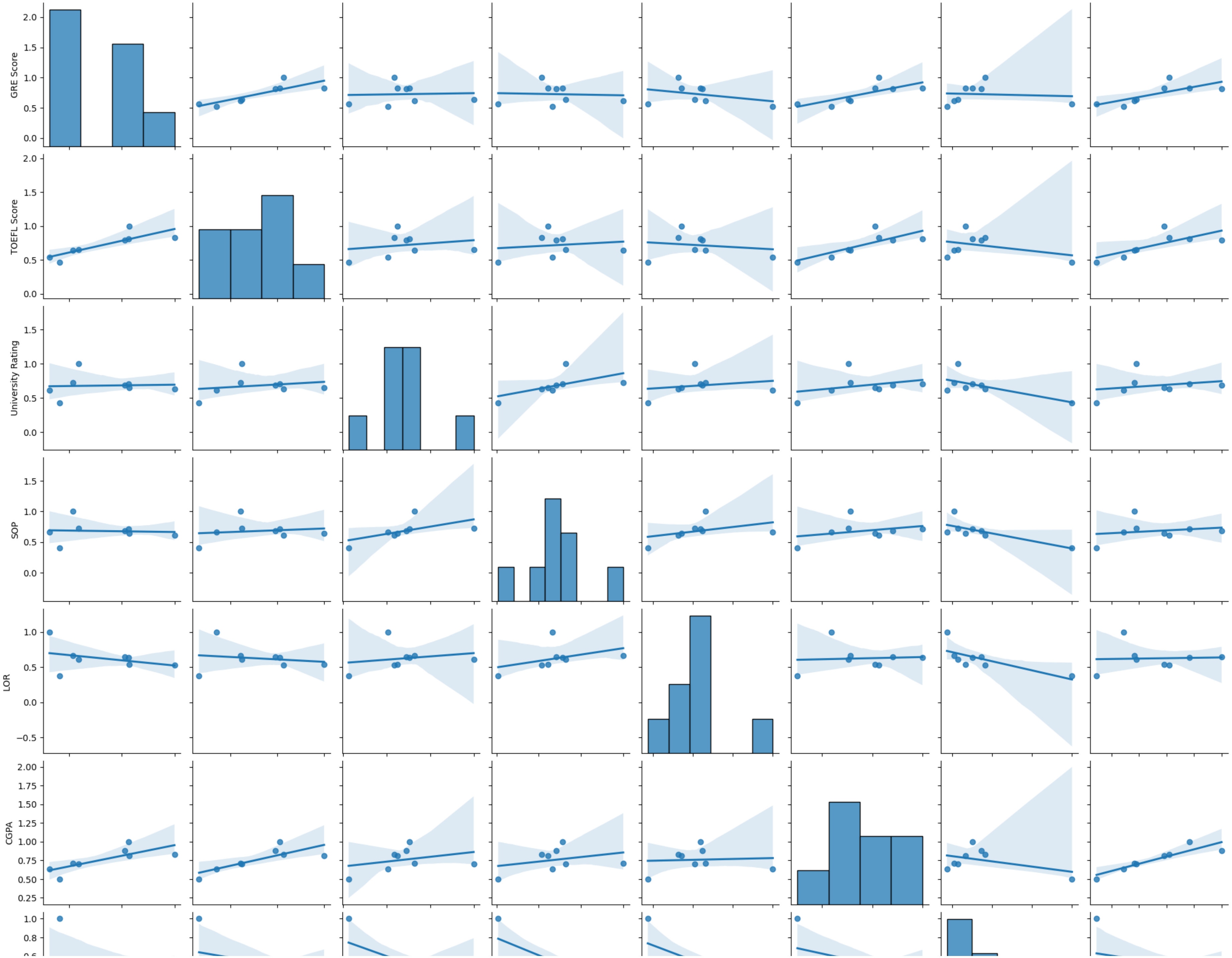
GRE Score TOEFL Score CGPA

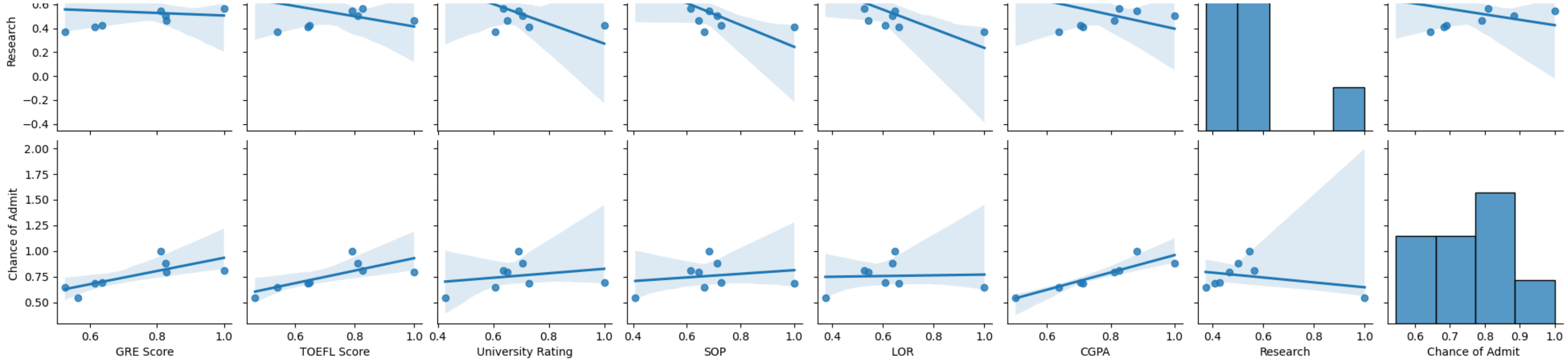
Independent continuous variables are also correlated with each other.

5. Checking the overall linearity and correlation across all features using pairplot :

In [18]:
```
sns.pairplot(df.corr(),kind= 'reg')
```

Out[18]:
```
<seaborn.axisgrid.PairGrid at 0x24f5f52a550>
```

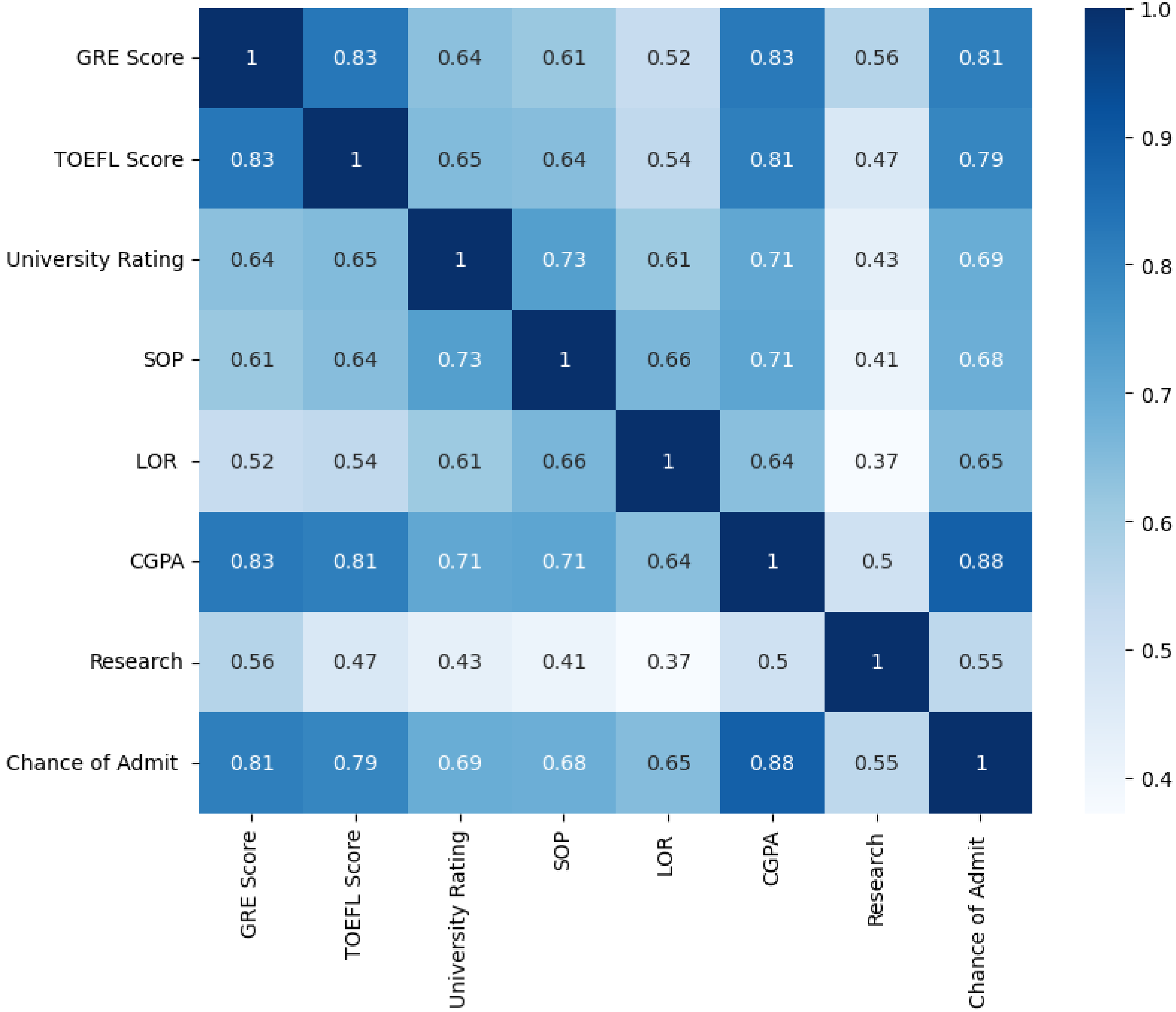

GRE Score
TOEFL Score
University Rating
SOP
LOR
CGPA



6. Correlation :

In [19]:
```
plt.figure(figsize=(9,7))
sns.heatmap(df.corr(),annot=True,cmap = "Blues")
```

Out[19]: <AxesSubplot:>


GRE Score
TOEFL Score
University Rating
SOP
LOR
CGPA
Research
Chance of Admit
1 0.83 0.64 0.61 0.52 0.83 0.56 0.81
0.83 1 0.65 0.64 0.54 0.81 0.47 0.79
0.64 0.65 1 0.73 0.61 0.71 0.43 0.69
0.61 0.64 0.73 1 0.66 0.71 0.41 0.68
0.52 0.54 0.61 0.66 1 0.64 0.37 0.65
0.83 0.81 0.71 0.71 0.64 1 0.5 0.88
0.56 0.47 0.43 0.41 0.37 0.5 1 0.55
0.81 0.79 0.69 0.68 0.65 0.88 0.55 1
1.0
0.9
0.8
0.7
0.6
0.5
0.4

- Independent Variables (Input data): GRE Score, TOEFL Score, University Rating, SOP, LOR, CGPA, Research
- Target/Dependent Variable : Chance of Admit (the value we want to predict)
- from above correlation heatmap , we can observe GRE score TOEFL score and CGPA have very high correlation with Change of admission.

- University rating, SOP ,LOR and Research have comparatively slightly less correlated than other features.

In [20]:
```
df.columns
```

Out[20]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

Changing or removing space between column names.

In [21]:
```
df.columns  = ['GRE_Score', 'TOEFL_Score', 'University_Rating', 'SOP', 'LOR', 'CGPA',
       'Research', 'Chance_of_Admit']
```

In [22]:
```
df.sample(2)
```

Out[22]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Chance_of_Admit
80	312	105	3	2.0	3.0	8.02	1	0.50
212	338	120	4	5.0	5.0	9.66	1	0.95

7. Outliers in the data :

In [23]:
```
def detect_outliers(data):
    length_before = len(data)
    Q1 = np.percentile(data,25)
    Q3 = np.percentile(data,75)
    IQR = Q3-Q1
    upperbound = Q3+1.5*IQR
    lowerbound = Q1-1.5*IQR
    if lowerbound < 0:
        lowerbound = 0
```

```
    length_after = len(data[(data>lowerbound)&(data<upperbound)])
    return f"{np.round((length_before-length_after)/length_before,4)} % Outliers data from input data found"
```

In [24]:
```
for col in df.columns:
    print(col," : ",detect_outliers(df[col]))
```

```
GRE_Score  :  0.0 % Outliers data from input data found
TOEFL_Score  :  0.0 % Outliers data from input data found
University_Rating  :  0.0 % Outliers data from input data found
SOP  :  0.0 % Outliers data from input data found
LOR  :  0.024 % Outliers data from input data found
CGPA  :  0.0 % Outliers data from input data found
Research  :  0.44 % Outliers data from input data found
Chance_of_Admit  :  0.004 % Outliers data from input data found
```

There are no significant amount of outliers found in the data

8. Descriptive analysis of all numerical features :

In [25]:
```
df.describe()
```

Out[25]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Chance_of_Admit
count	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.00000
mean	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	0.72174
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	0.14114
min	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.34000
25%	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.63000
50%	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.72000
75%	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.82000
max	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.97000

- Chances of admit is a probability measure , which is within 0 to 1 which is good (no outliers or missleading data in column).
- Range of GRE score looks like between 290 to 340.

- Range of TOEFL score is between 92 to 120.
- University rating , SOP and LOR are distributed between range of 1 to 5.
- CGPA range is between 6.8 to 9.92.

In [26]:
```
df.columns
```

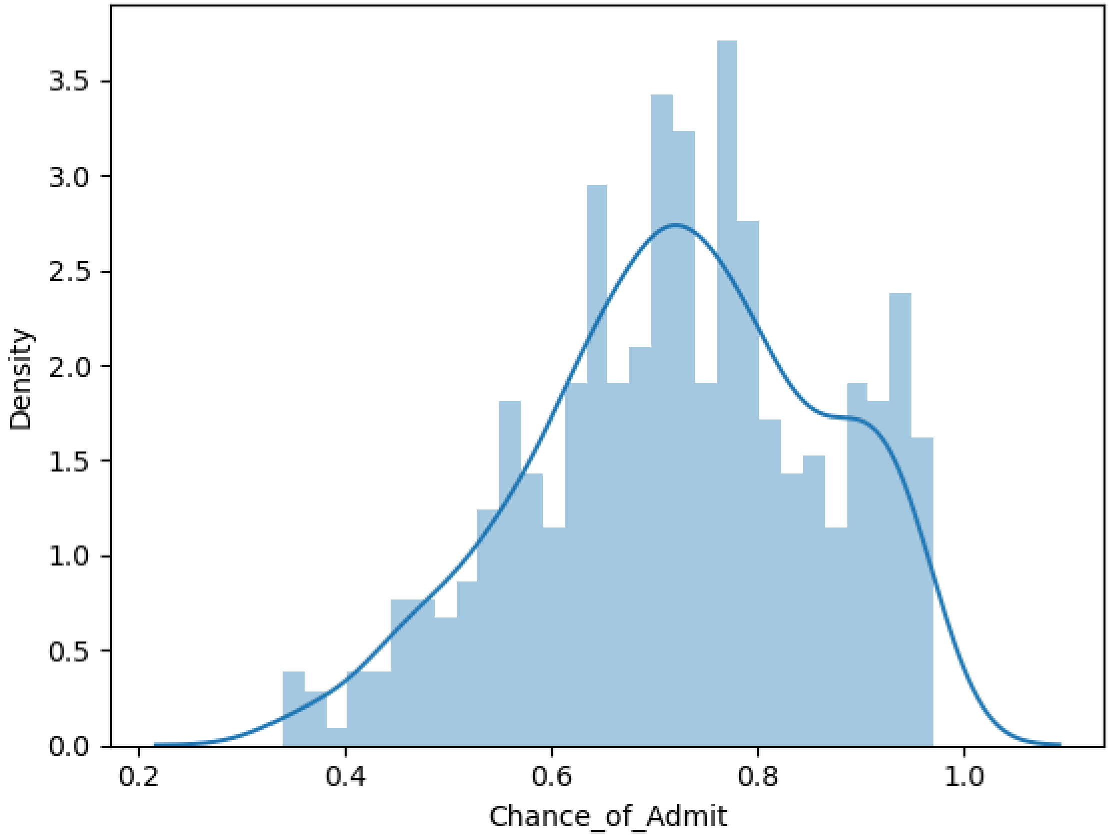
Out[26]:
```
Index(['GRE_Score', 'TOEFL_Score', 'University_Rating', 'SOP', 'LOR', 'CGPA',
       'Research', 'Chance_of_Admit'],
      dtype='object')
```

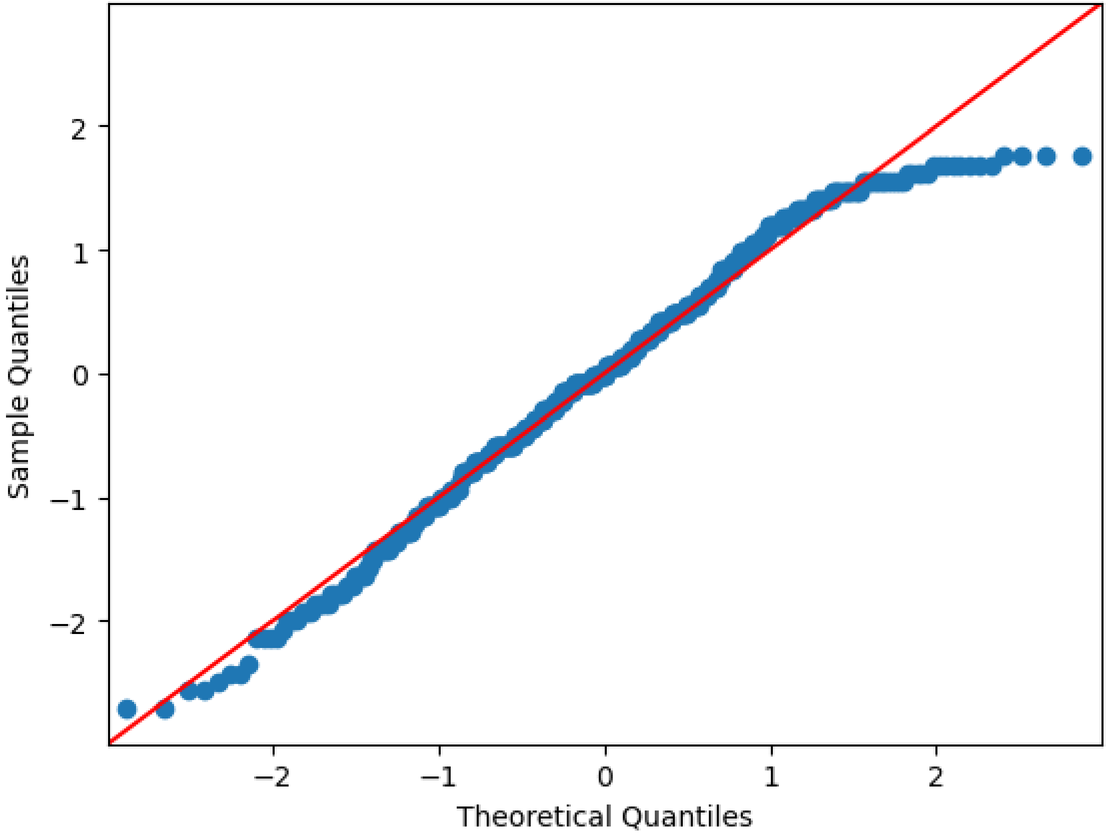
9. Graphical Analysis :

Distributions / Histogram and count plot :

Chance_of_Admit

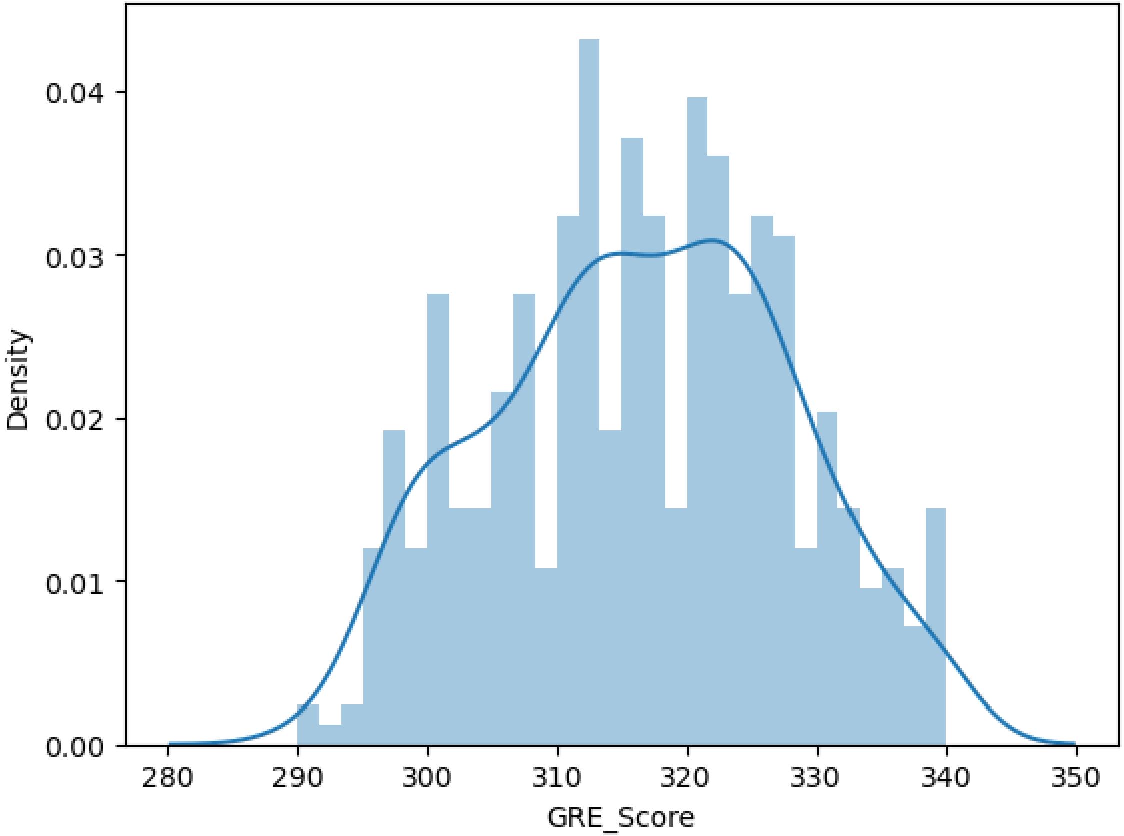
In [27]:
```
sns.distplot(df["Chance_of_Admit"],bins = 30)
sm.qqplot(df["Chance_of_Admit"],fit=True, line="45")
plt.show()
```


3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Density
0.2
0.4
0.6
0.8
1.0
Chance_of_Admit

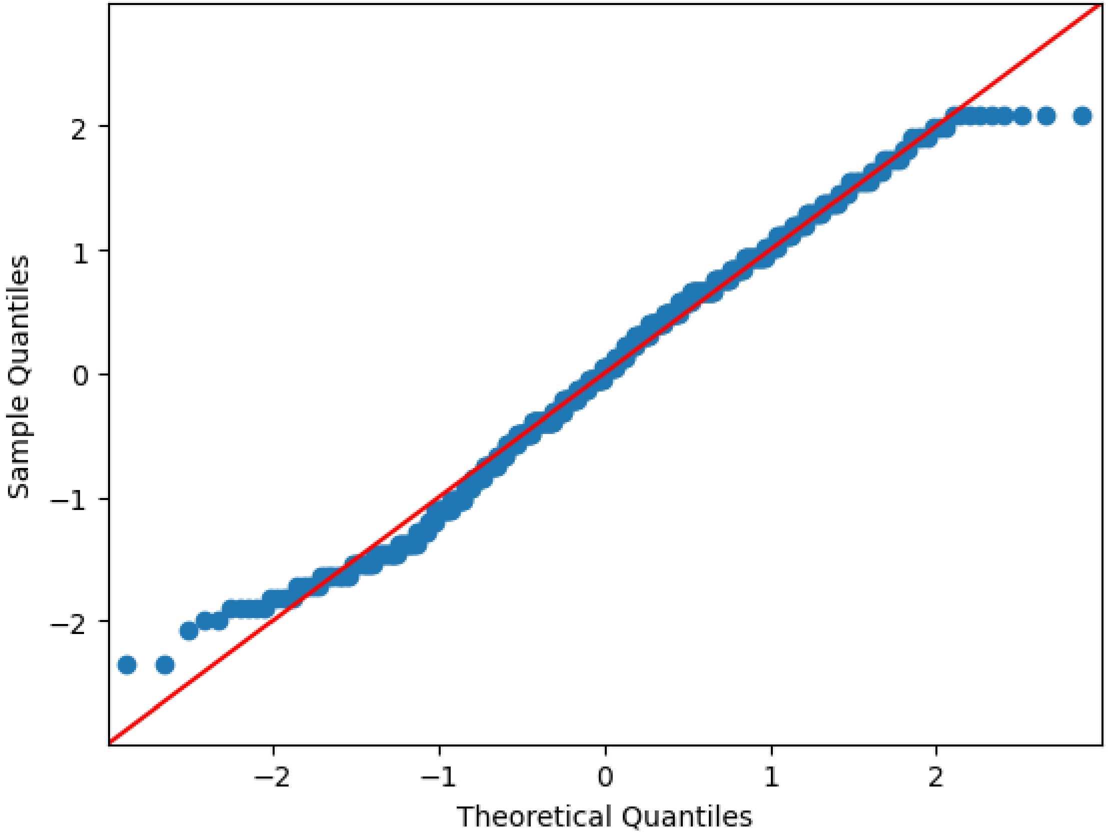


GRE_Score

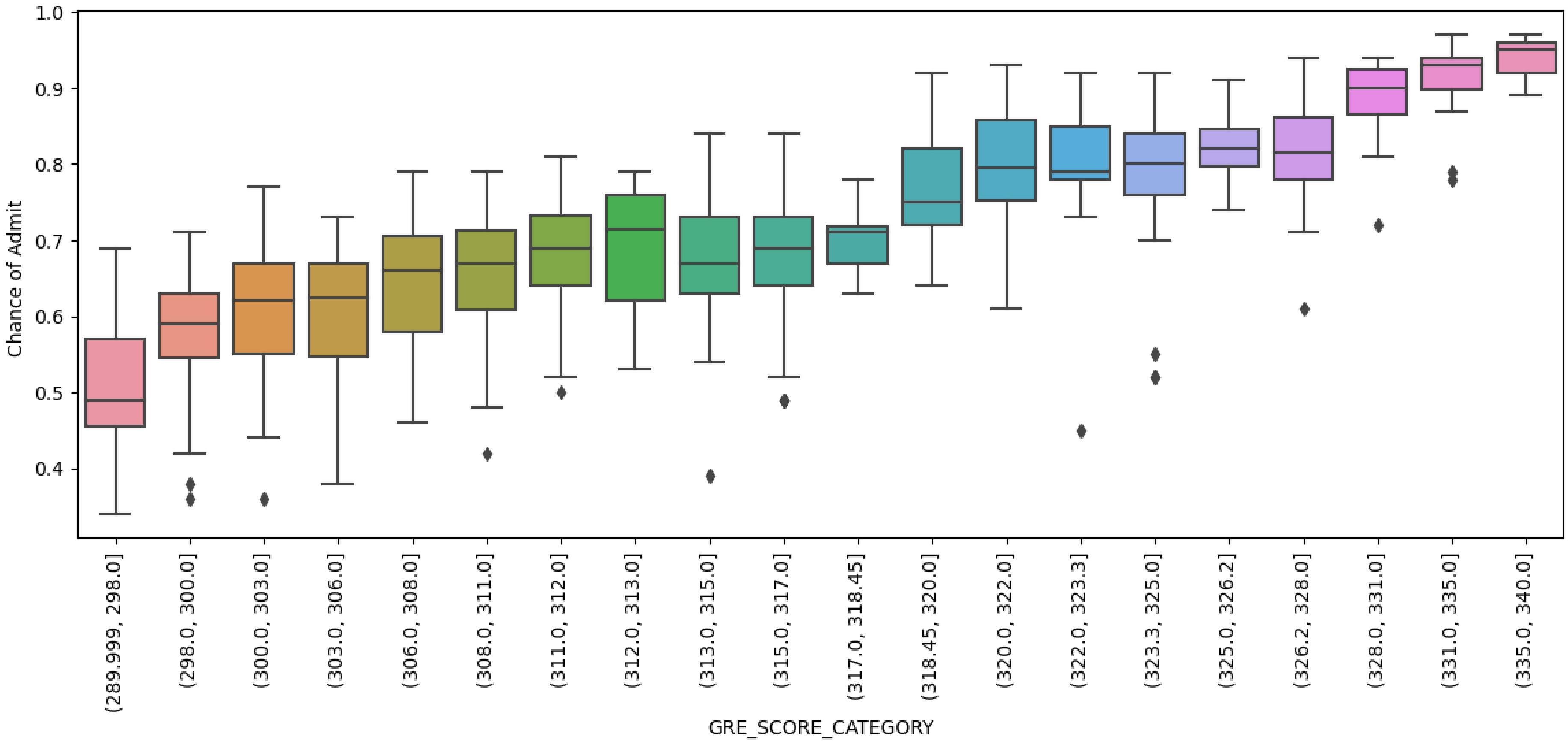
In [28]:
```
sns.distplot(df["GRE_Score"], bins = 30)
sm.qqplot(df["GRE_Score"],fit=True, line="45")
plt.show()
```



Density
0.00
0.01
0.02
0.03
0.04
280
290
300
310
320
330
340
350
GRE_Score



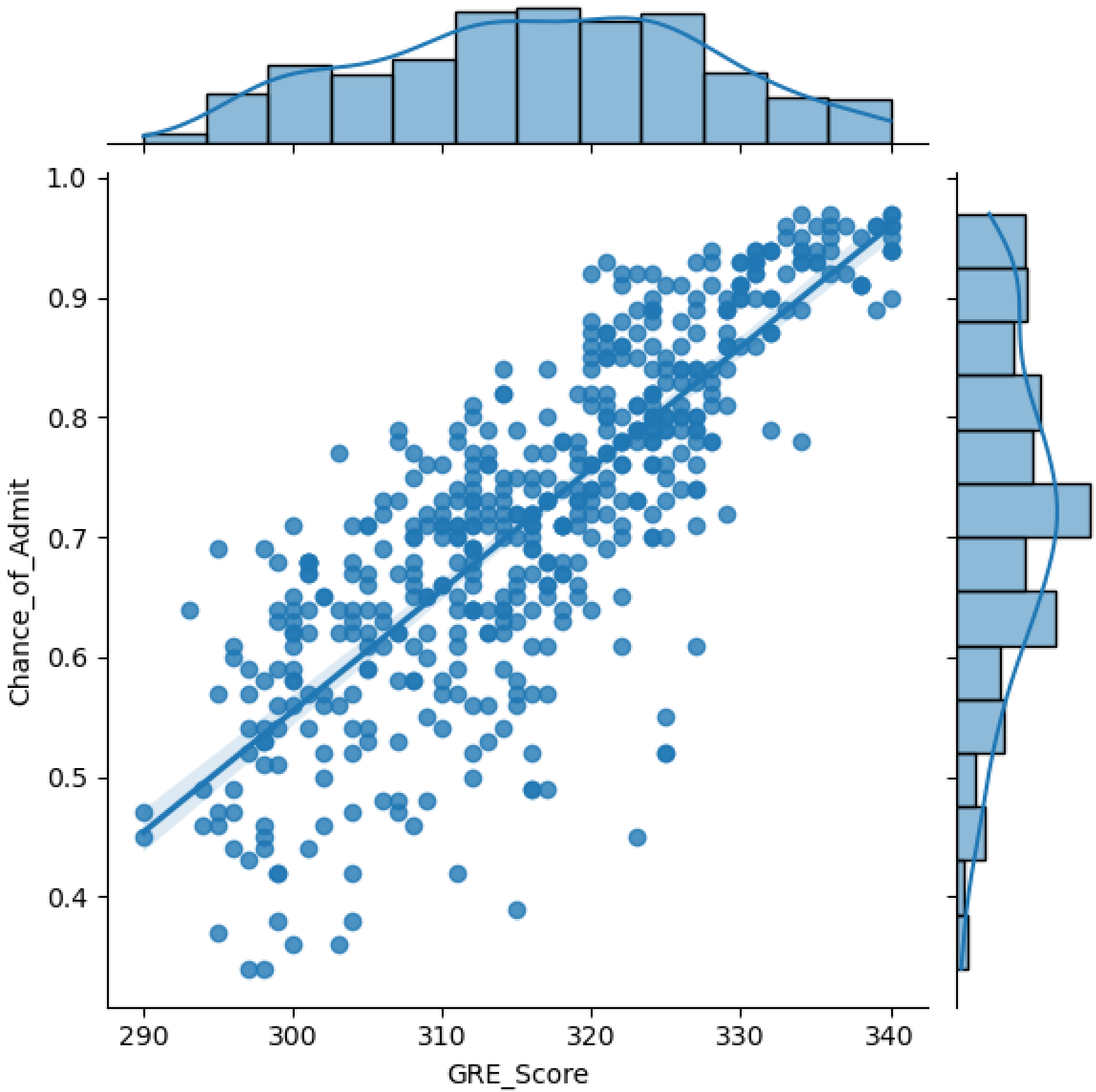
In [29]:
```
data["GRE_SCORE_CATEGORY"]=pd.qcut(data["GRE Score"],20)
plt.figure(figsize=(14,5))
sns.boxplot(y = data["Chance of Admit "], x = data["GRE_SCORE_CATEGORY"])
plt.xticks(rotation = 90)
plt.show()
```



From above boxplot (distribution of chance of admition (probability of getting admition) as per GRE score) : with higher GRE score , there is high probability of getting an admition .

In [30]:
```
sns.jointplot(df["GRE_Score"],df["Chance_of_Admit"], kind = "reg" )
```

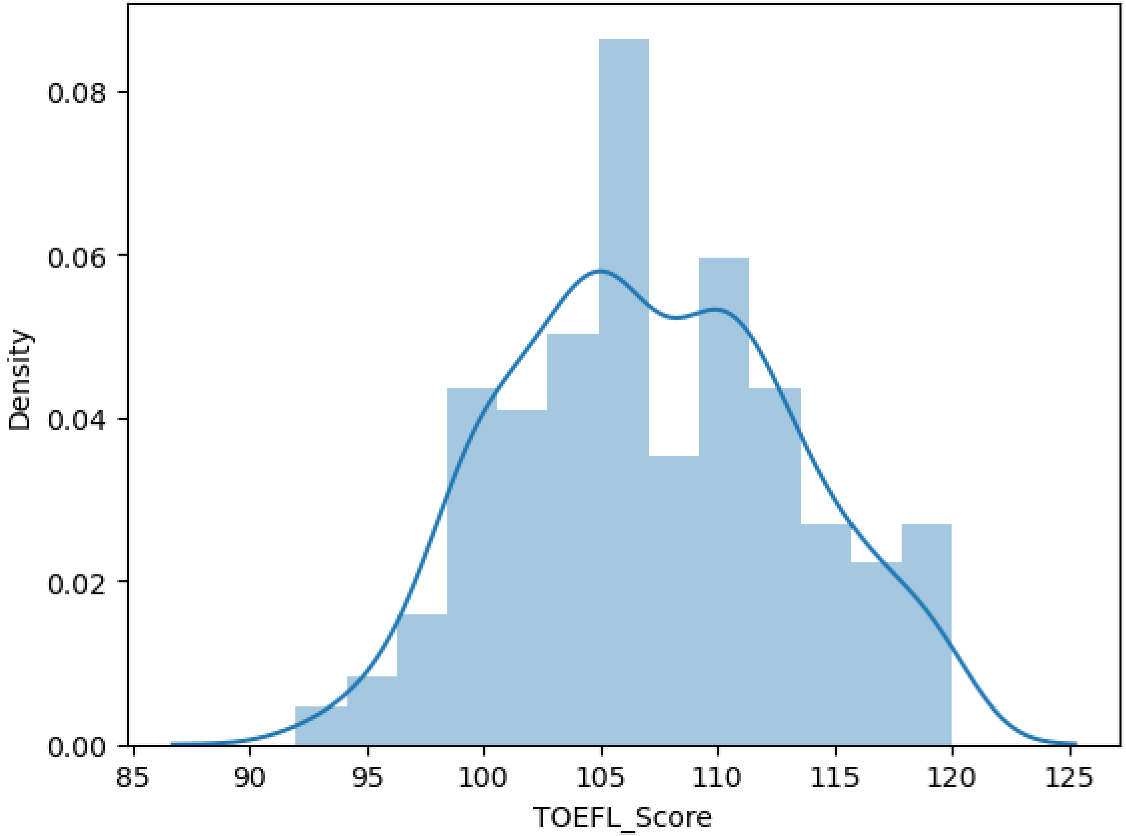
Out[30]: <seaborn.axisgrid.JointGrid at 0x24f63358c10>

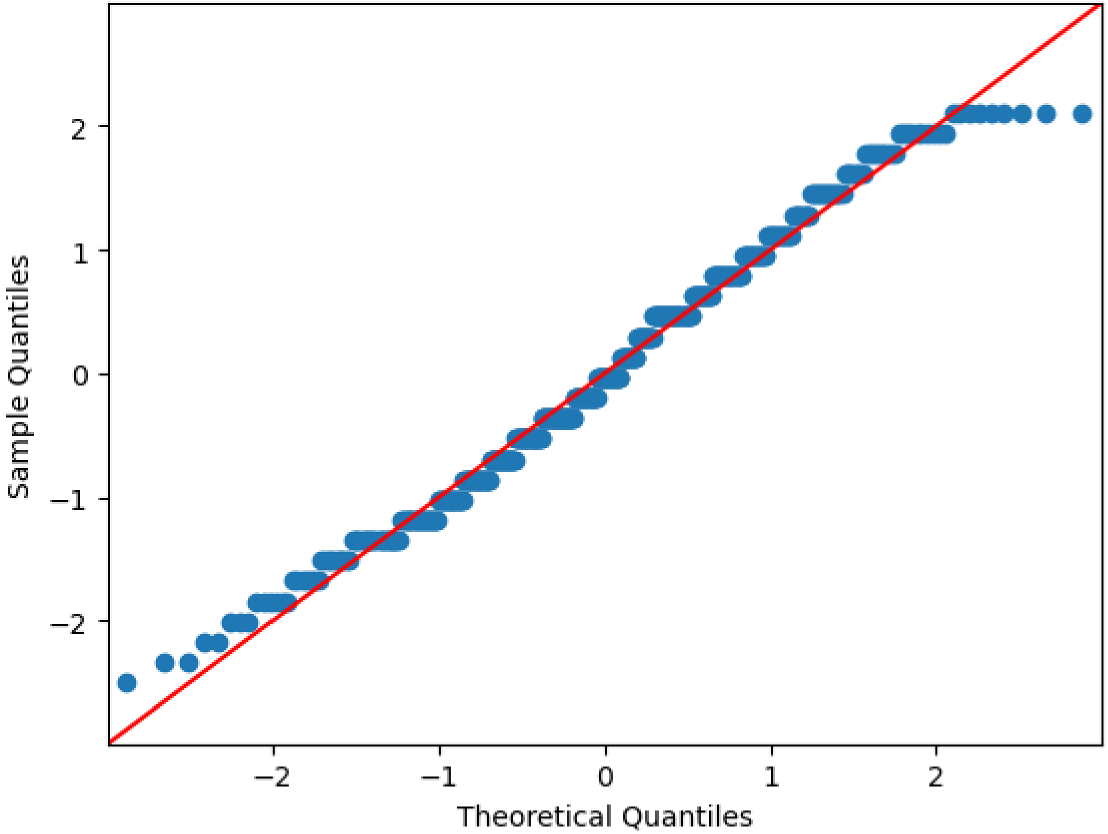


TOEFL Score

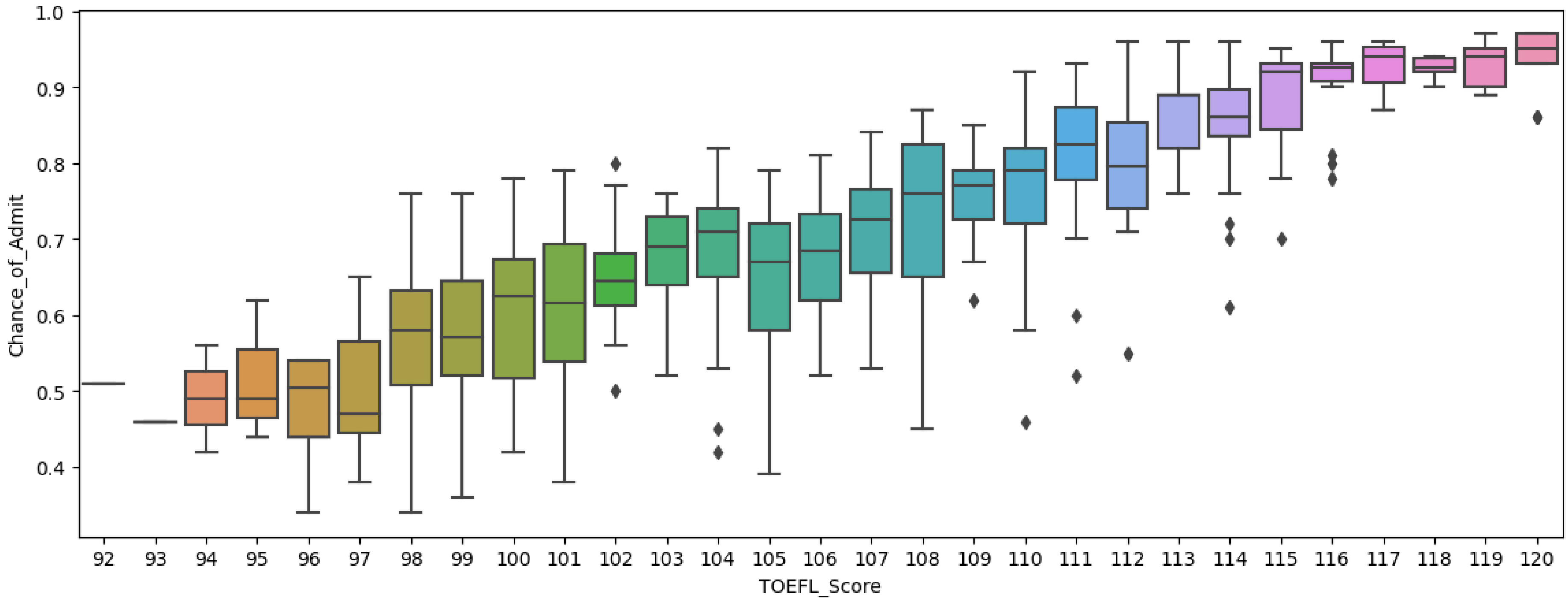
In [31]:
```
sns.distplot(df["TOEFL_Score"])
sm.qqplot(df["TOEFL_Score"],fit=True, line="45")
plt.show()
```

```
plt.figure(figsize=(14,5))
sns.boxplot(y = df["Chance_of_Admit"], x = df["TOEFL_Score"])
```





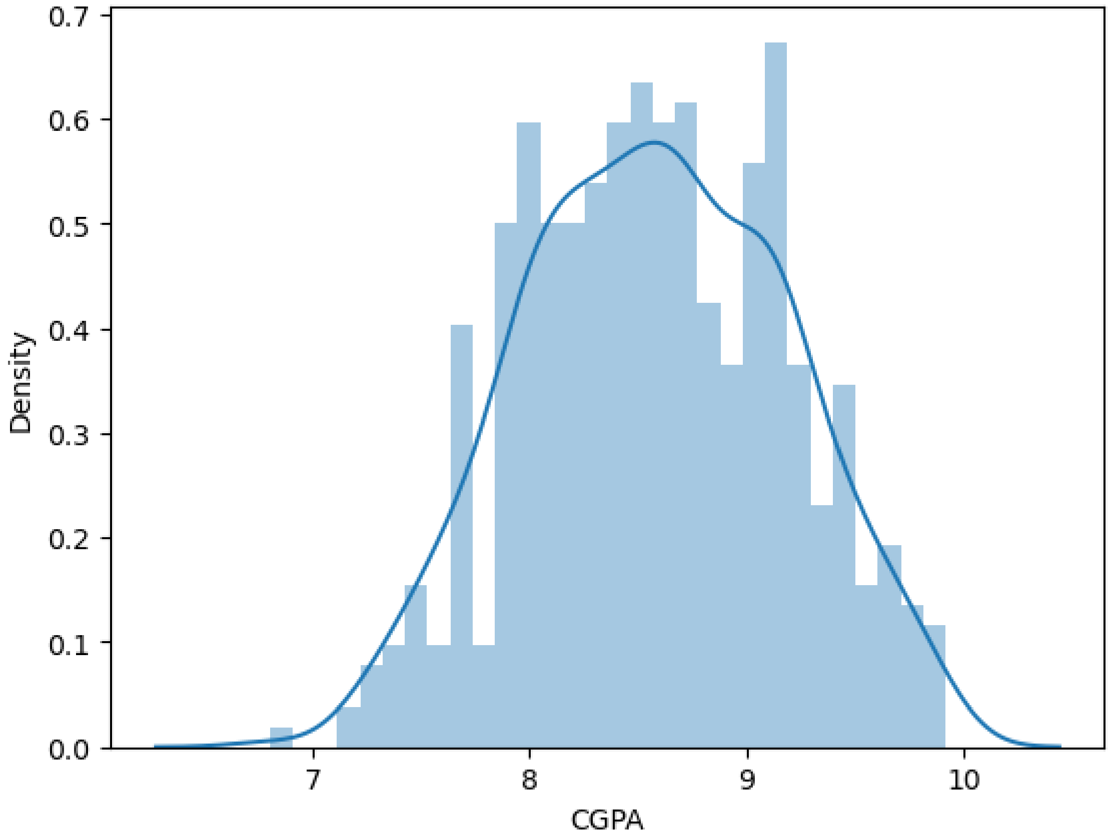
Out[31]: <AxesSubplot:xlabel='TOEFL_Score', ylabel='Chance_of_Admit'>



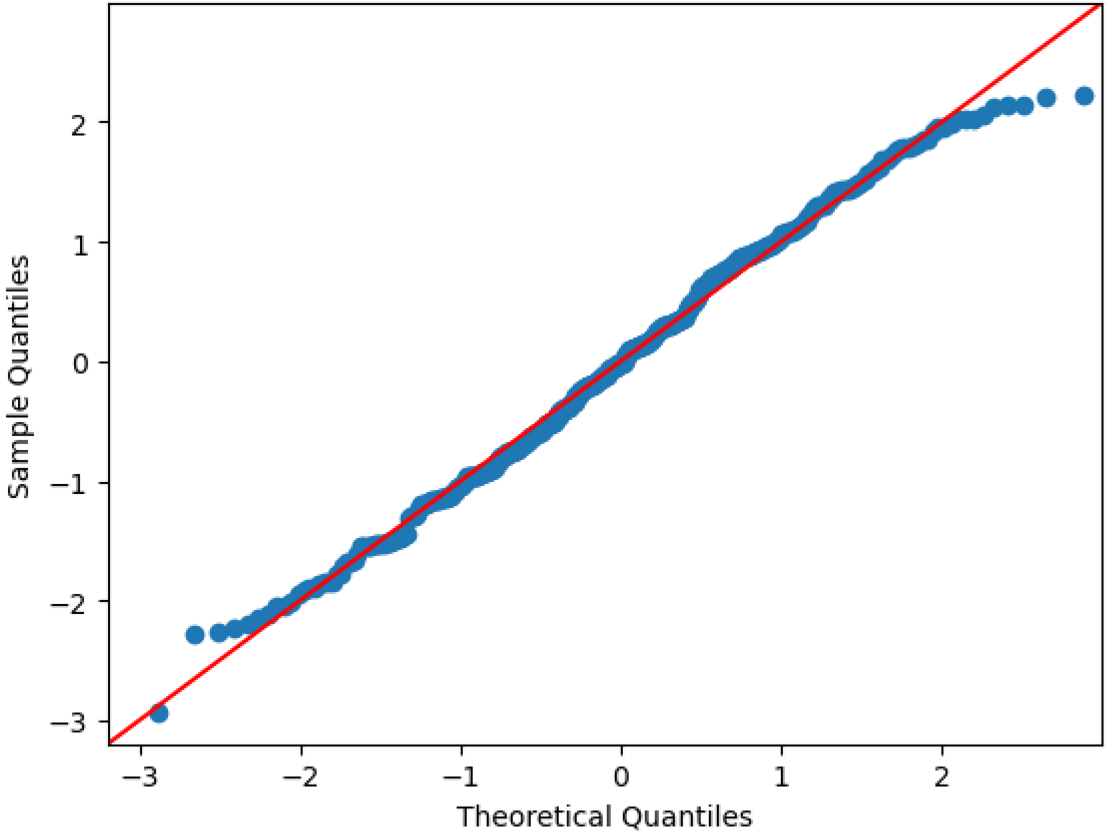
Students having high toefl score , has higher probability of getting admition .

CGPA

In [32]:
```
sns.distplot(df["CGPA"], bins = 30)
sm.qqplot(df["CGPA"],fit=True, line="45")
plt.show()
```



0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Density
7
8
9
10
CGPA



Chance of admit and GRE score are nearly normally distrubted.

GRE score, TOEFL score and CGPA has a strong correlation with chance of addmission .

In [33]:
```
df.columns
```

Out[33]:
```
Index(['GRE_Score', 'TOEFL_Score', 'University_Rating', 'SOP', 'LOR', 'CGPA',
       'Research', 'Chance_of_Admit'],
      dtype='object')
```

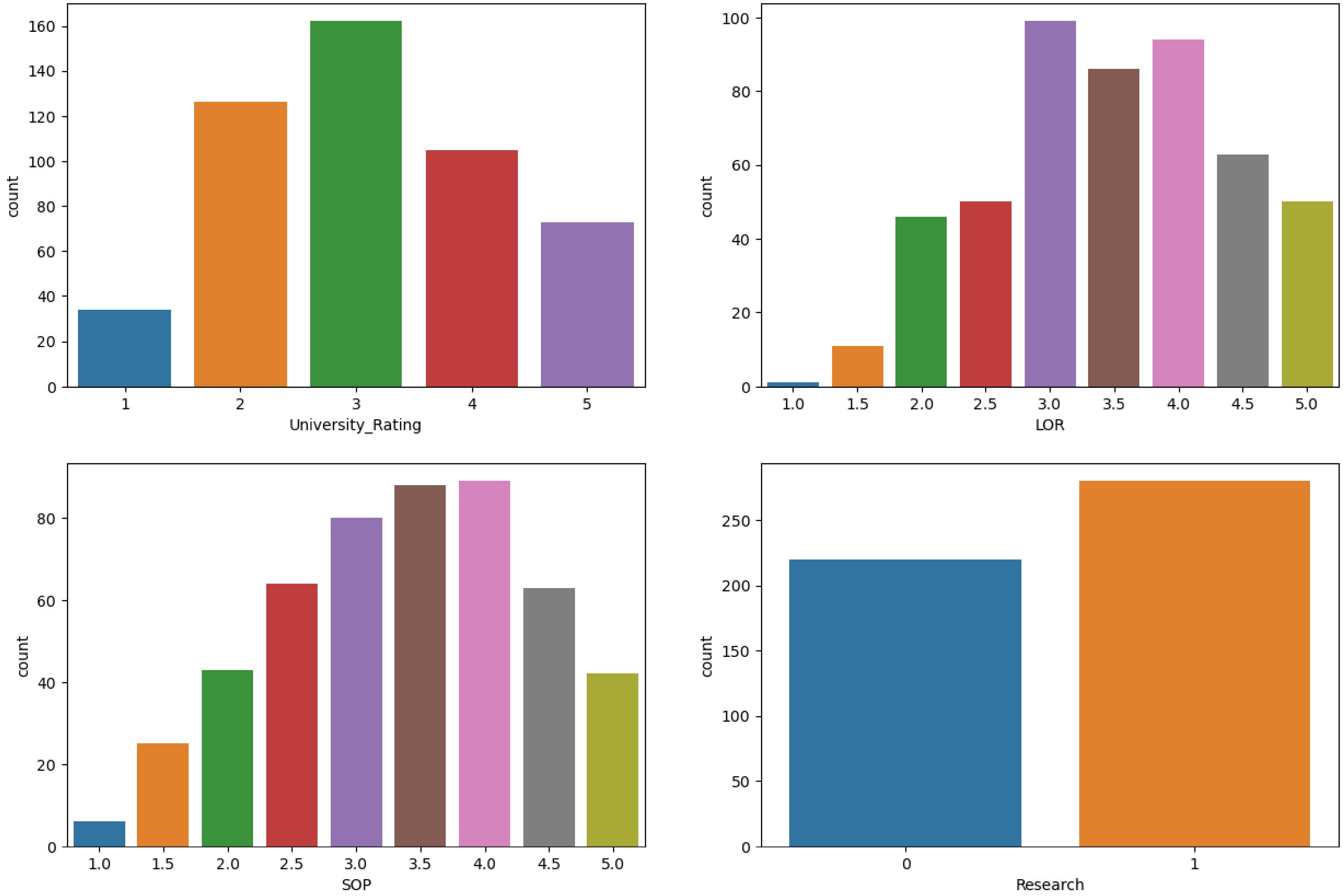
Distribution of all other categorical features :

In [34]:
```
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
```

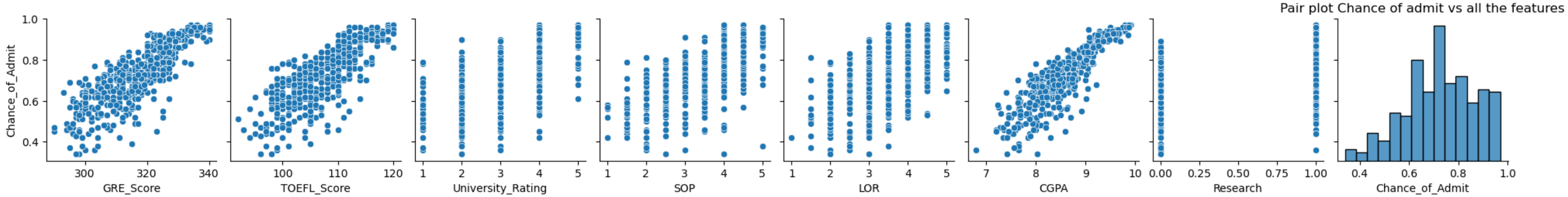
```
sns.countplot(df["University_Rating"])
plt.subplot(2,2,2)
sns.countplot(df["LOR"])
plt.subplot(2,2,3)
sns.countplot(df["SOP"])
plt.subplot(2,2,4)
sns.countplot(df["Research"])
```

Out[34]:

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

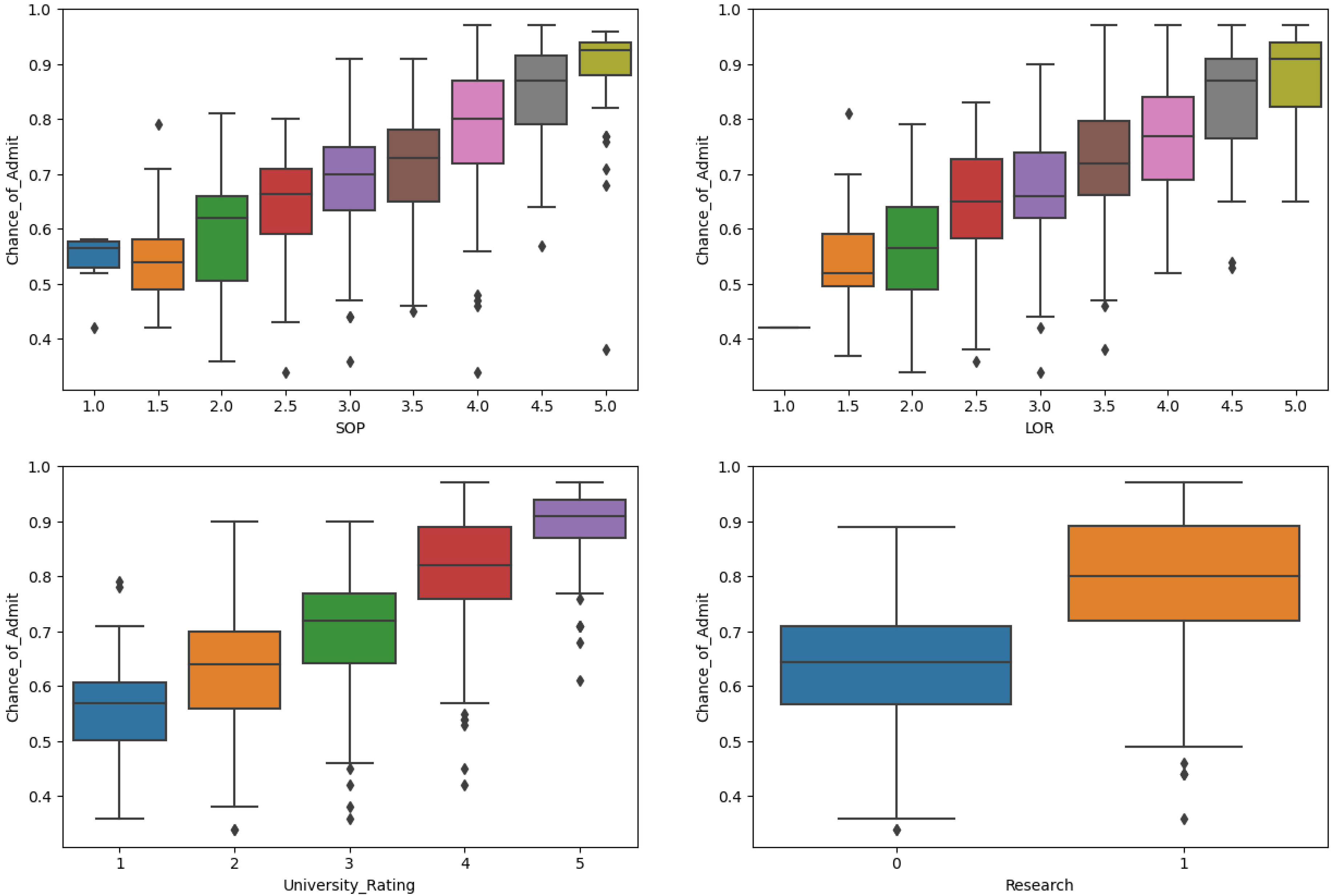


In [35]:
```
sns.pairplot(df,y_vars = ["Chance_of_Admit"])
plt.title("Pair plot Chance of admit vs all the features")
plt.show()
```



Categorical features - vs - chances of admission boxplot:

In [36]:
```
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
sns.boxplot(y = df["Chance_of_Admit"], x = df["SOP"])
plt.subplot(2,2,2)
sns.boxplot(y = df["Chance_of_Admit"], x = df["LOR"])
plt.subplot(2,2,3)
sns.boxplot(y = df["Chance_of_Admit"], x = df["University_Rating"])
plt.subplot(2,2,4)
sns.boxplot(y = df["Chance_of_Admit"], x = df["Research"])
plt.show()
```



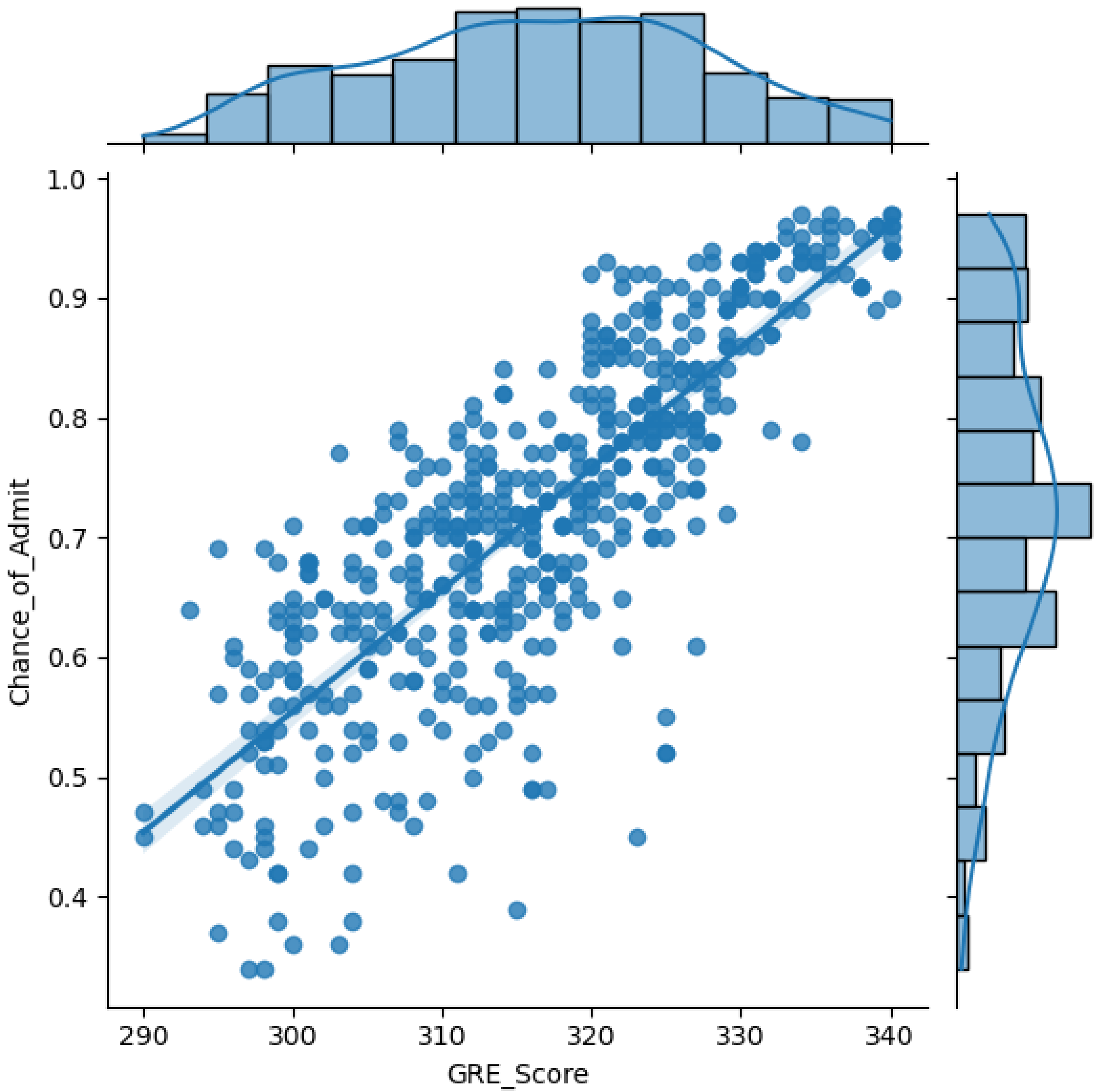
from above plots, we can observe , statement of purpose SOP strength is positively correlated with Chance of Admission . we can also similar pattern in Letter of Recommendation Stength and University rating , have positive correlation with Chaces of Admission . Student having research has higher chances of Admission , but also we can observe some outliers within that caregory.

Linearity : How features are correlated with Target variable - chance of admit:

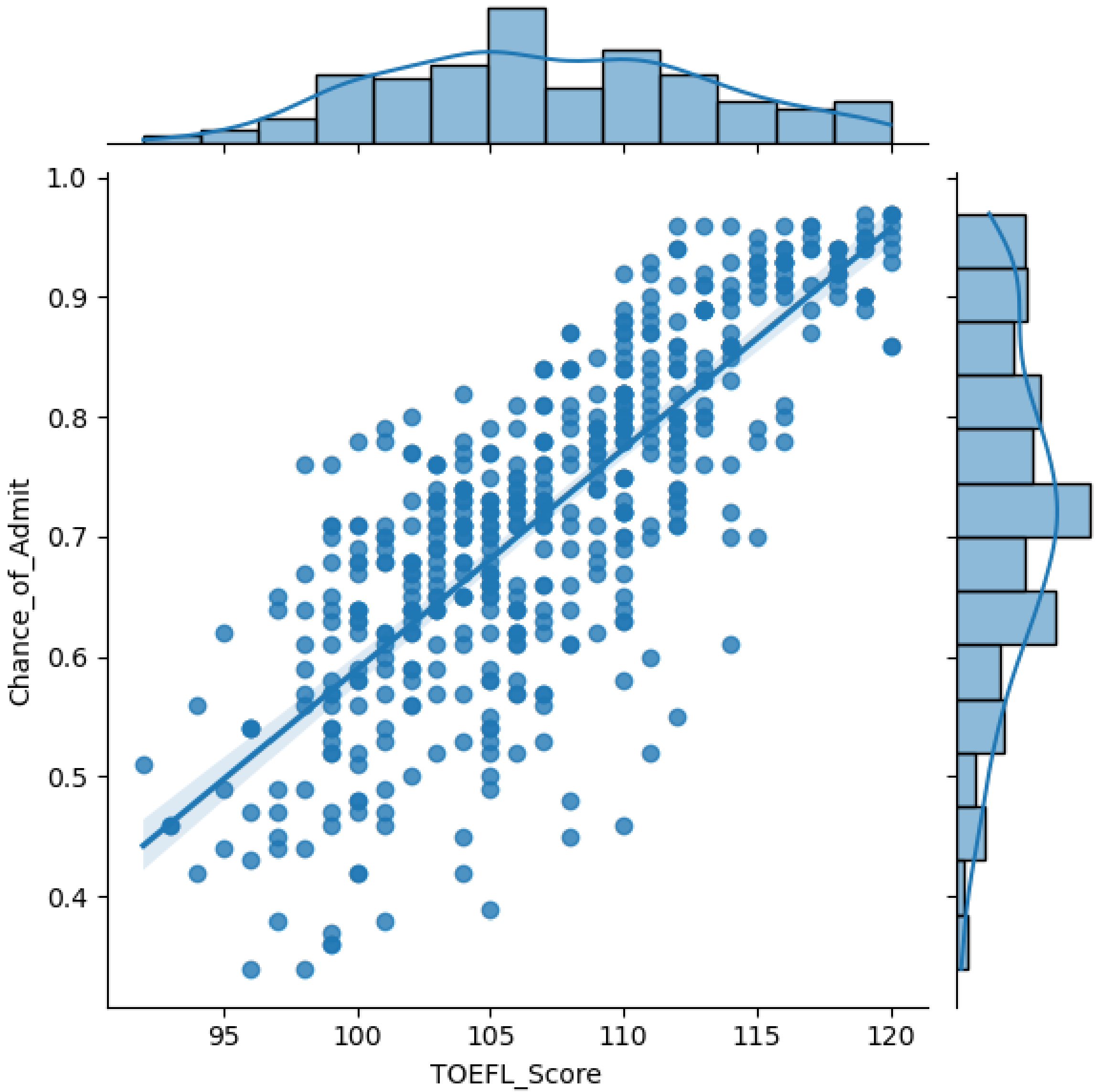
In [37]:

```
for col in df.columns[:-1]:
    print(col)
    plt.figure(figsize=(3,3))
    sns.jointplot(df[col],df["Chance_of_Admit"],kind="reg")
    plt.show()
```

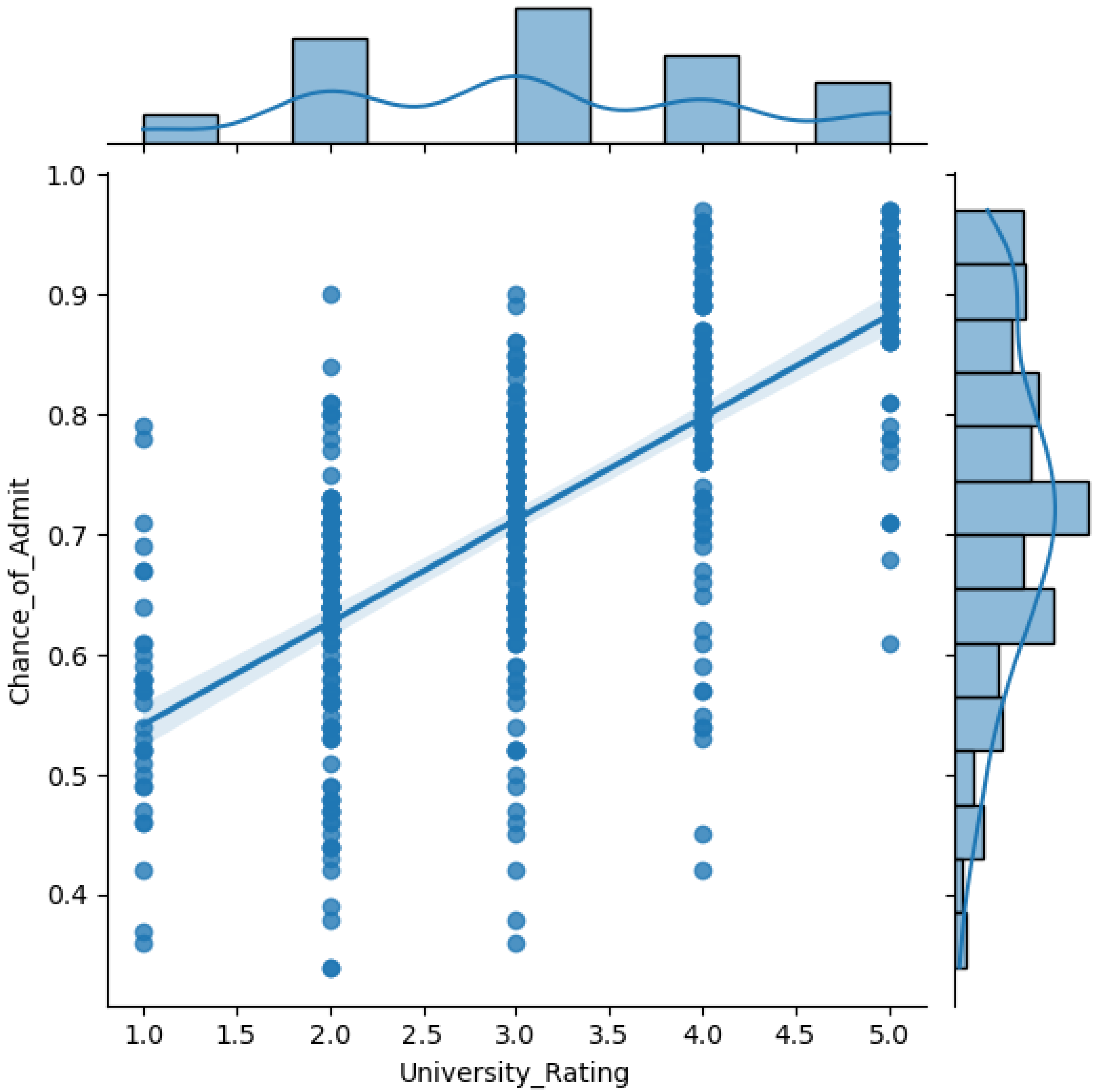
```
GRE_Score
<Figure size 300x300 with 0 Axes>
```



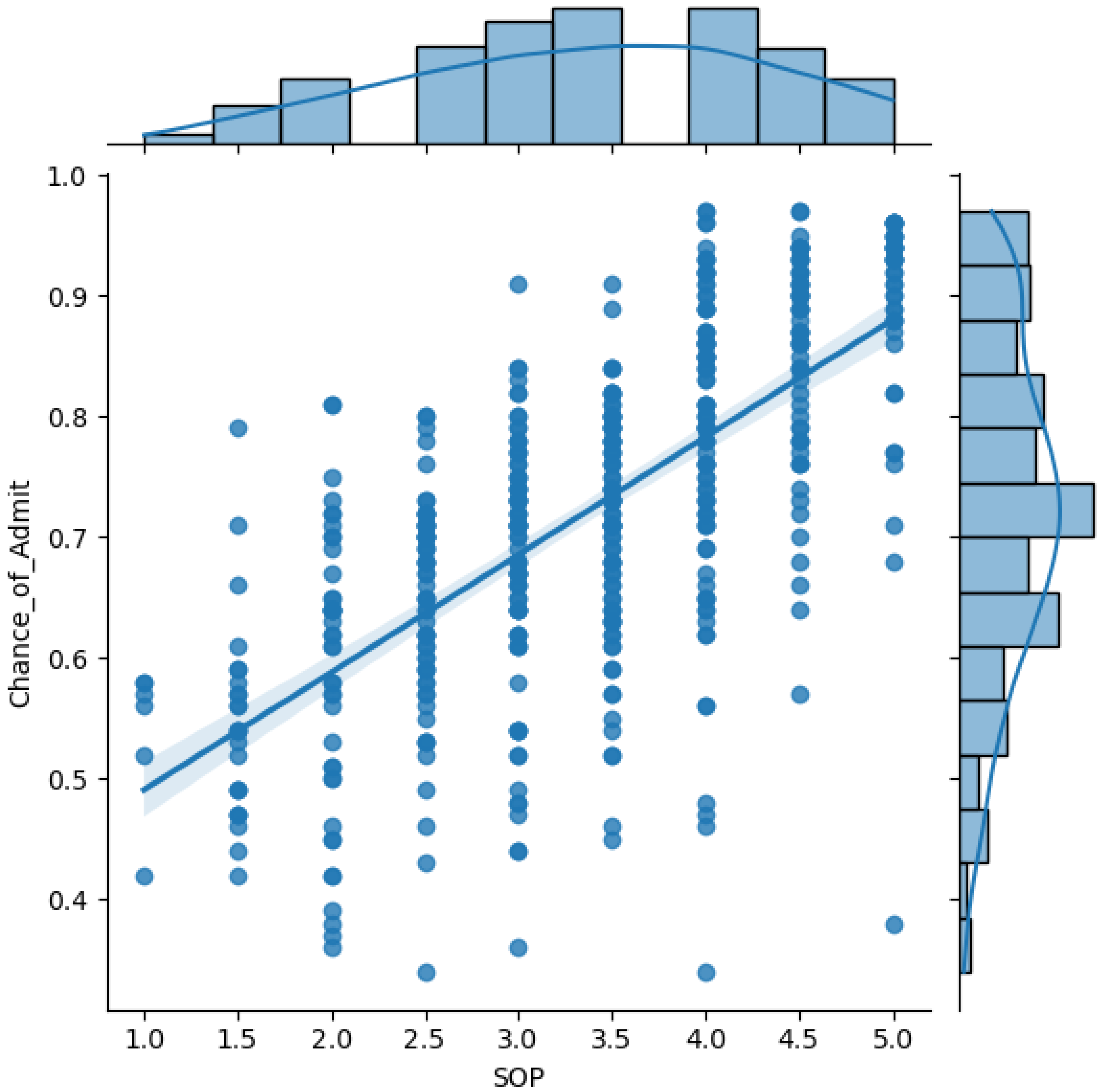

```
TOEFL_Score
<Figure size 300x300 with 0 Axes>
```



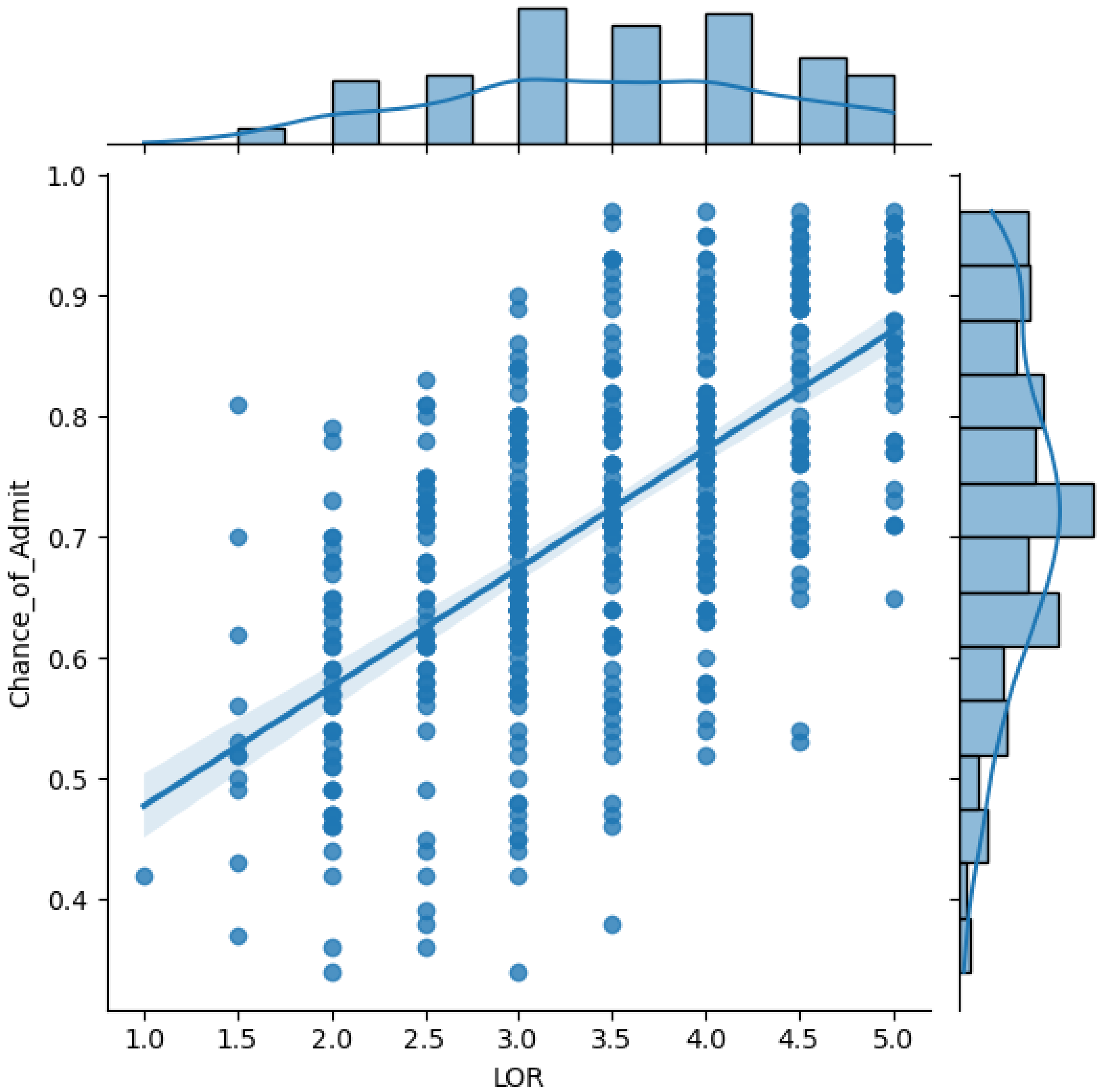

```
University_Rating
<Figure size 300x300 with 0 Axes>
```



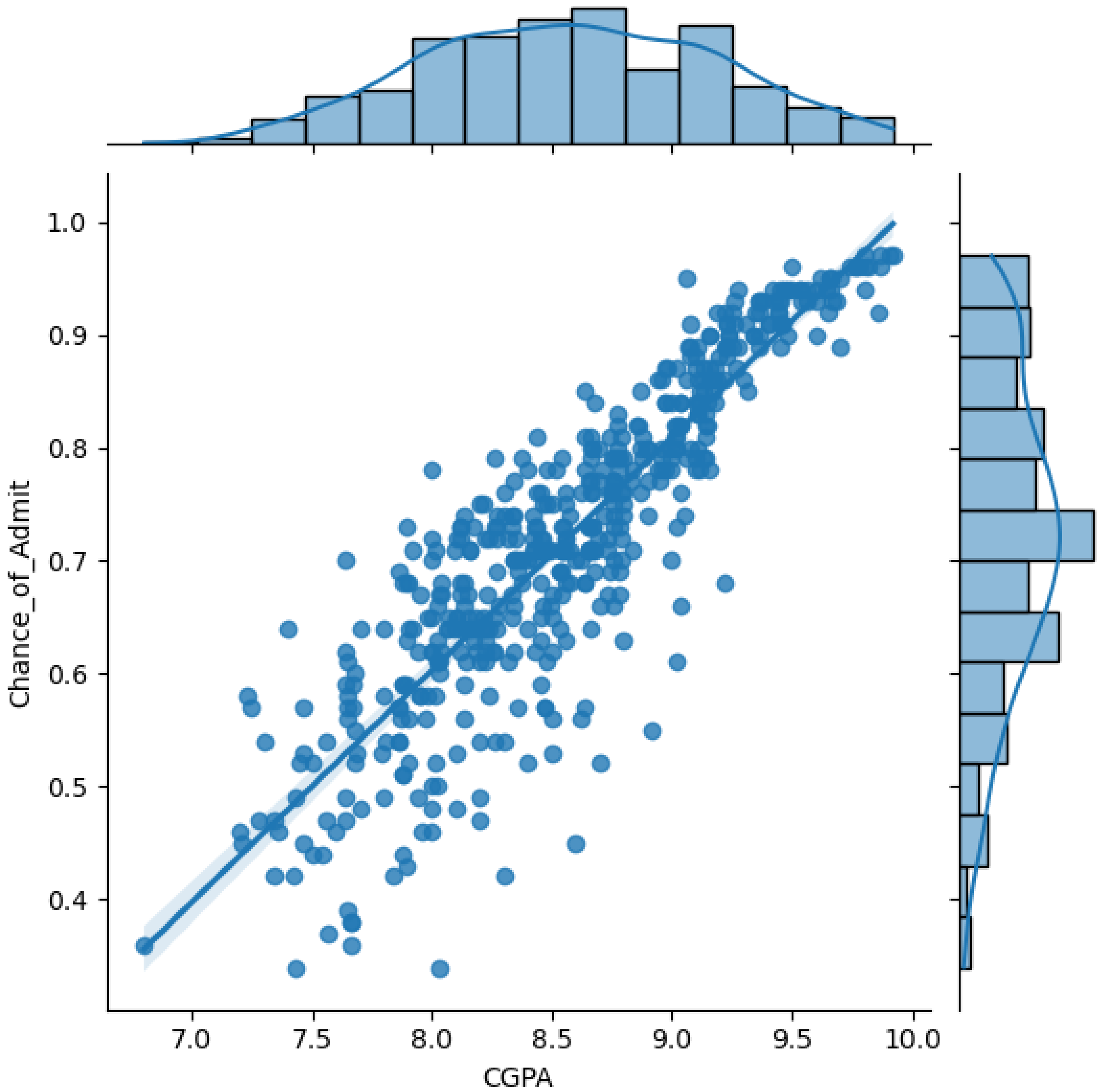

```
SOP
<Figure size 300x300 with 0 Axes>
```



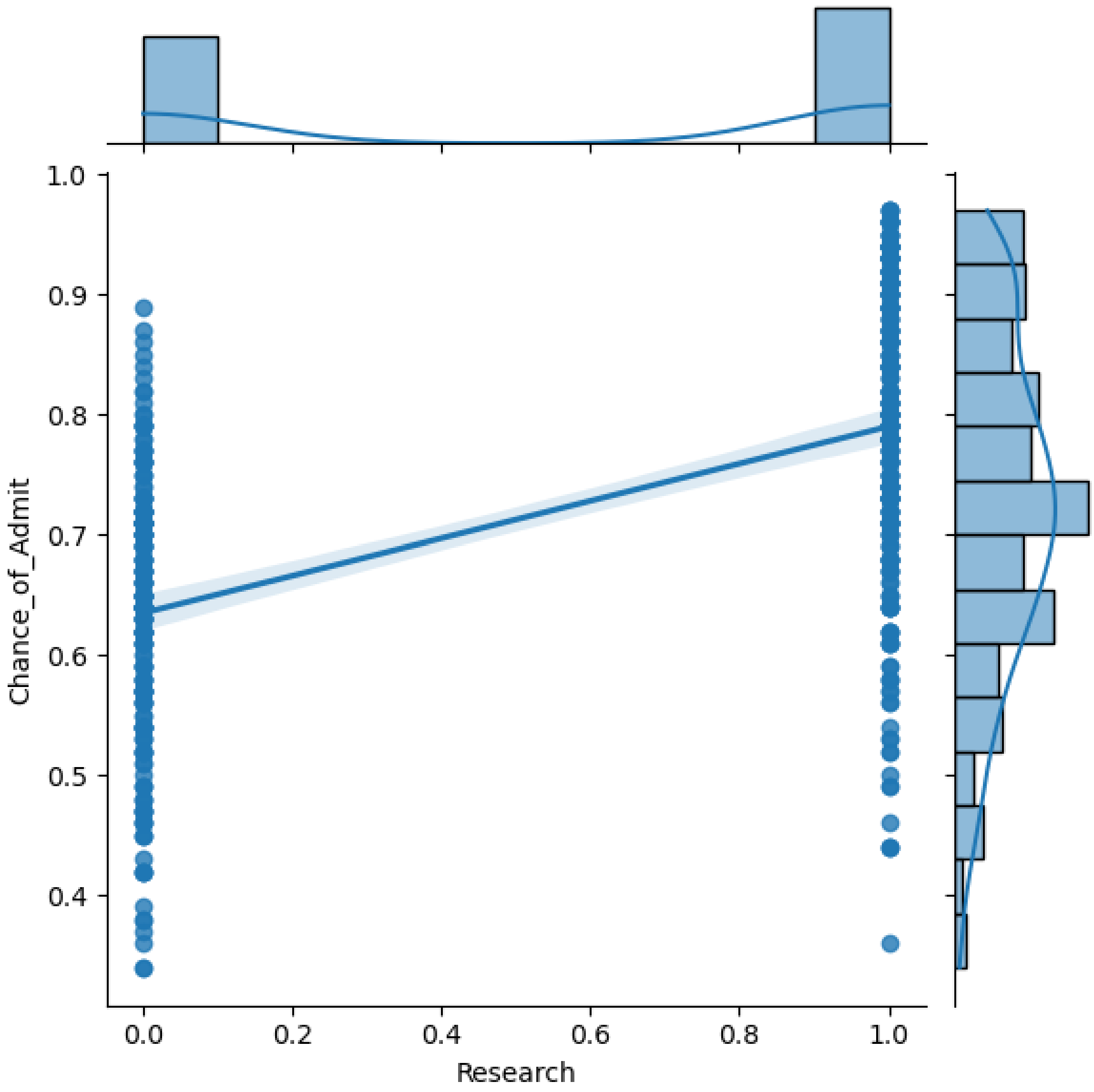

```
LOR
<Figure size 300x300 with 0 Axes>
```




```
CGPA
<Figure size 300x300 with 0 Axes>
```




```
Research
<Figure size 300x300 with 0 Axes>
```



10. Linear Regression :

In [38]:
```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error, adjusted_mutual_info_score
from sklearn.feature_selection import f_regression

from scipy import stats
```

In [39]:
```
X = df.drop(["Chance_of_Admit"],axis = 1)  # independent variables
y = df["Chance_of_Admit"].values.reshape(-1,1) # target / dependent variables
```

Standardising data

In [40]:
```
standardizer = StandardScaler()
standardizer.fit(X)
x = standardizer.transform(X)  # standardising the data
```

Test train spliting :

In [41]:
```
X_train , X_test, y_train , y_test = train_test_split(x,y,
                                                       random_state = 1,
                                                        test_size = 0.2
                                                       )                         # test train split


X_train.shape,X_test.shape  # after spliting, checking for the shape of test and  train data
```

Out[41]:
```
((400, 7), (100, 7))
```

In [42]:
```
y_train.shape, y_test.shape
```

Out[42]:
```
((400, 1), (100, 1))
```

Training the model

In [43]:
```
LinearRegression = LinearRegression()    # training LinearRegression model
LinearRegression.fit(X_train,y_train)
```

Out[43]:
```
LinearRegression()
```

r2 score on train data :

In [44]:
```
r2_score(y_train,LinearRegression.predict(X_train))
```

Out[44]:
```
0.8215099192361265
```

r2 score on test data :

In [45]:
```
r2_score(y_test,LinearRegression.predict(X_test))
```

Out[45]:
```
0.8208741703103732
```

All the feature's coefficients and Intercept :

In [46]:
```
ws = pd.DataFrame(LinearRegression.coef_.reshape(1,-1),columns=df.columns[:-1])
ws["Intercept"] = LinearRegression.intercept_
ws
```

Out[46]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Intercept
0	0.020675	0.019284	0.007001	0.002975	0.013338	0.070514	0.009873	0.722881

In [47]:
```
LinearRegression_Model_coefs = ws
LinearRegression_Model_coefs
```

Out[47]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Intercept
0	0.020675	0.019284	0.007001	0.002975	0.013338	0.070514	0.009873	0.722881

In [48]:
```
def AdjustedR2score(R2,n,d):
    return 1-(((1-R2)*(n-1))/(n-d-1))
```

In [49]:
```
y_pred = LinearRegression.predict(X_test)

print("MSE:",mean_squared_error(y_test,y_pred)) # MSE
print("RMSE:",np.sqrt(mean_squared_error(y_test,y_pred))) #RMSE
```

```
print("MAE :",mean_absolute_error(y_test,y_pred) ) # MAE
print("r2_score:",r2_score(y_test,y_pred)) # r2score
print("Adjusted R2 score :", AdjustedR2score(r2_score(y_test,y_pred),len(X),X.shape[1]))  # adjusted R2 score
```

```
MSE: 0.0034590988971363833
RMSE: 0.058814104576507695
MAE : 0.040200193804157944
r2_score: 0.8208741703103732
Adjusted R2 score : 0.8183256320830818
```

Assumptions of linear regression

```
No multicollinearity
The mean of residual is nearly zero.
Linearity of Variables
Test of homoscedasticity
Normality of residual
```

12. Multicollinearity check :

```
Checking VIF scores :
```

- VIF(Variance Inflation Factor)
  - VIF score of an independent variable represents how well the variable is explained by other independent variables.
  - So, the closer the R^2 value to 1, the higher the value of VIF and the higher the multicollinearity with the particular independent variable.

In [50]:
```
vifs = []

for i in range(X_train.shape[1]):

    vifs.append((variance_inflation_factor(exog = X_train,
                                           exog_idx=i)))

pd.DataFrame({ "coef_name : " : X.columns ,
             "vif : ": np.around(vifs,2)})
```

Out[50]:

	coef_name :	vif :
0	GRE_Score	4.87
1	TOEFL_Score	4.24
2	University_Rating	2.80
3	SOP	2.92
4	LOR	2.08
5	CGPA	4.75
6	Research	1.51

In [ ]:

VIF score are all below 5 , doesnt seem to have very high multicolinearity.

13. Residual analysis :

In [51]:

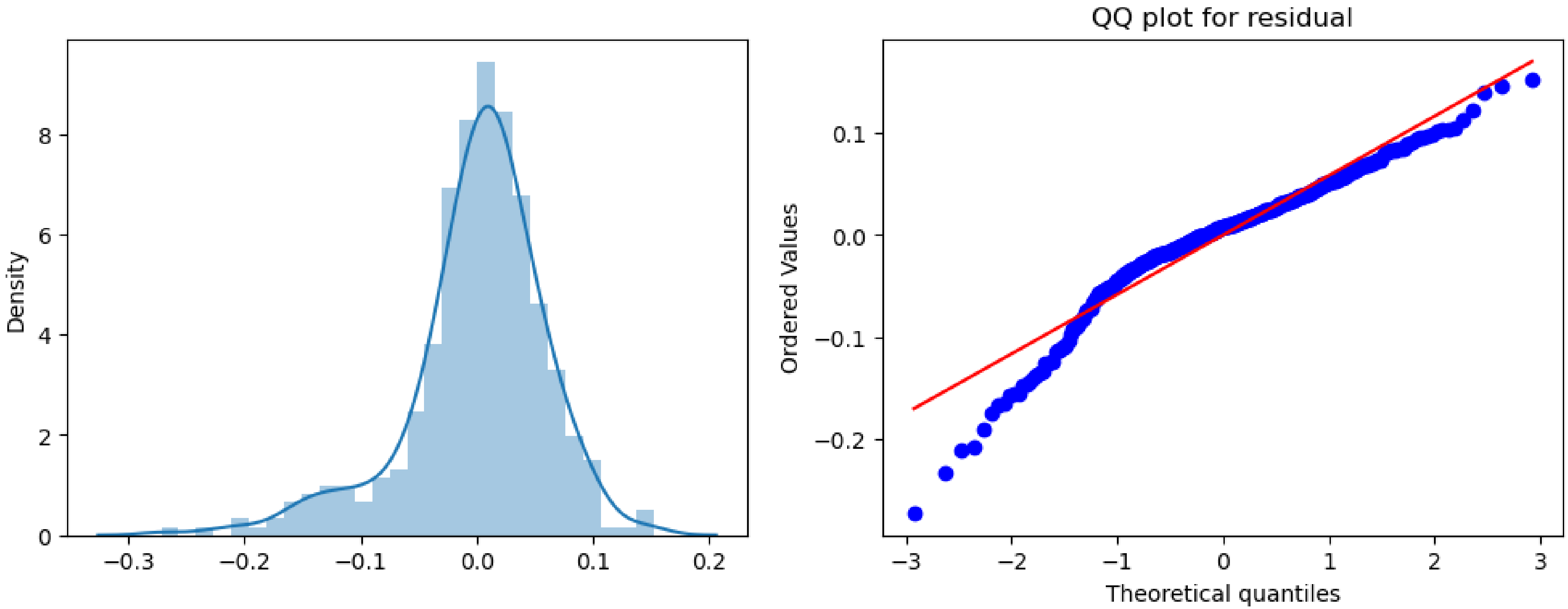
```
y_predicted = LinearRegression.predict(X_train)
y_predicted.shape
```

Out[51]:

```
(400, 1)
```

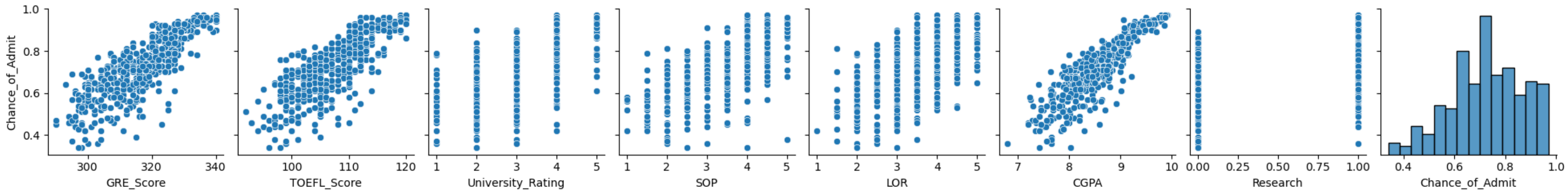
In [52]:

```
residuals = (y_train - y_predicted)
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
sns.distplot(residuals)
plt.subplot(1,2,2)
stats.probplot(residuals.reshape(-1,), plot = plt)
plt.title('QQ plot for residual')
plt.show()
```



Linearity of varibales

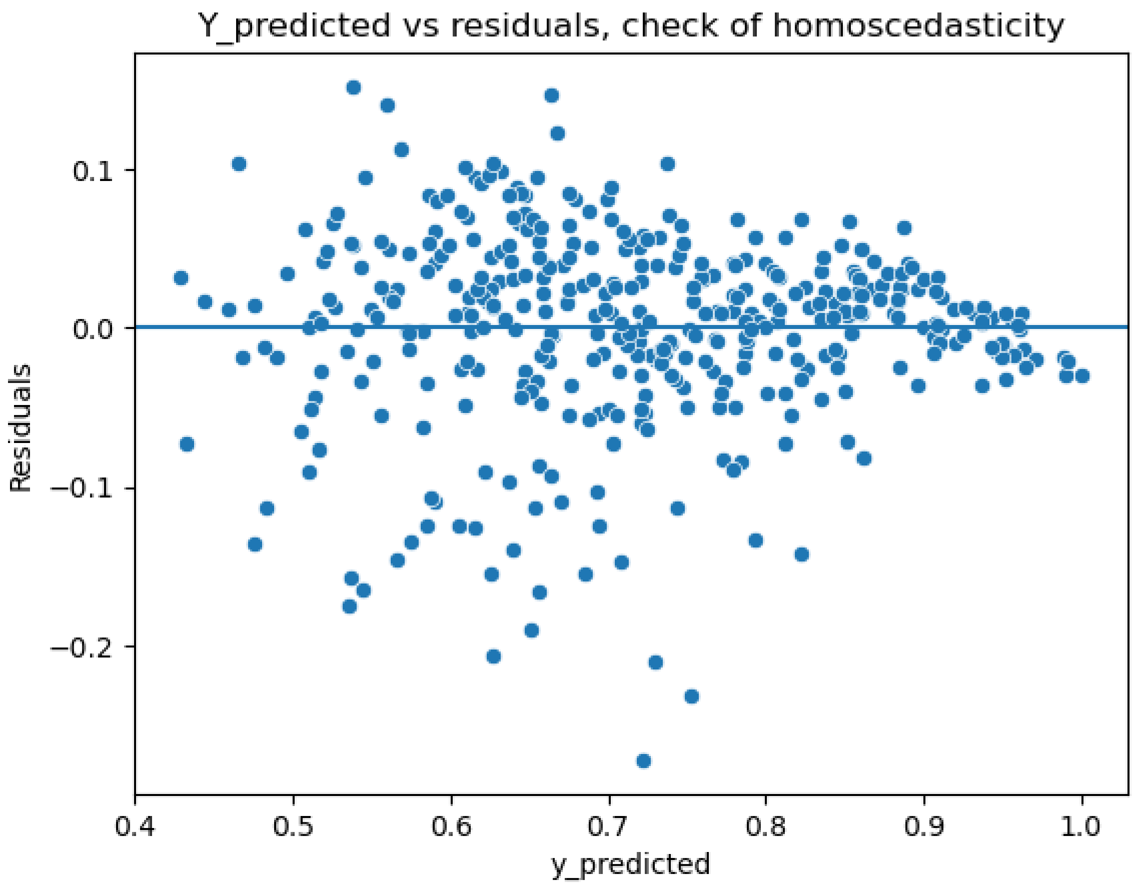
In [53]:
```
sns.pairplot(df,y_vars = ["Chance_of_Admit"])
plt.show()
```



Test of homoscedasticity | plotting y_predicted and residuals

In [54]:
```
sns.scatterplot(y_predicted.reshape(-1,), residuals.reshape(-1,))
plt.xlabel('y_predicted')
plt.ylabel('Residuals')
```

```python
plt.axhline(y=0)
plt.title("Y_predicted vs residuals, check of homoscedasticity")
plt.show()
```



14. Model Regularisation :

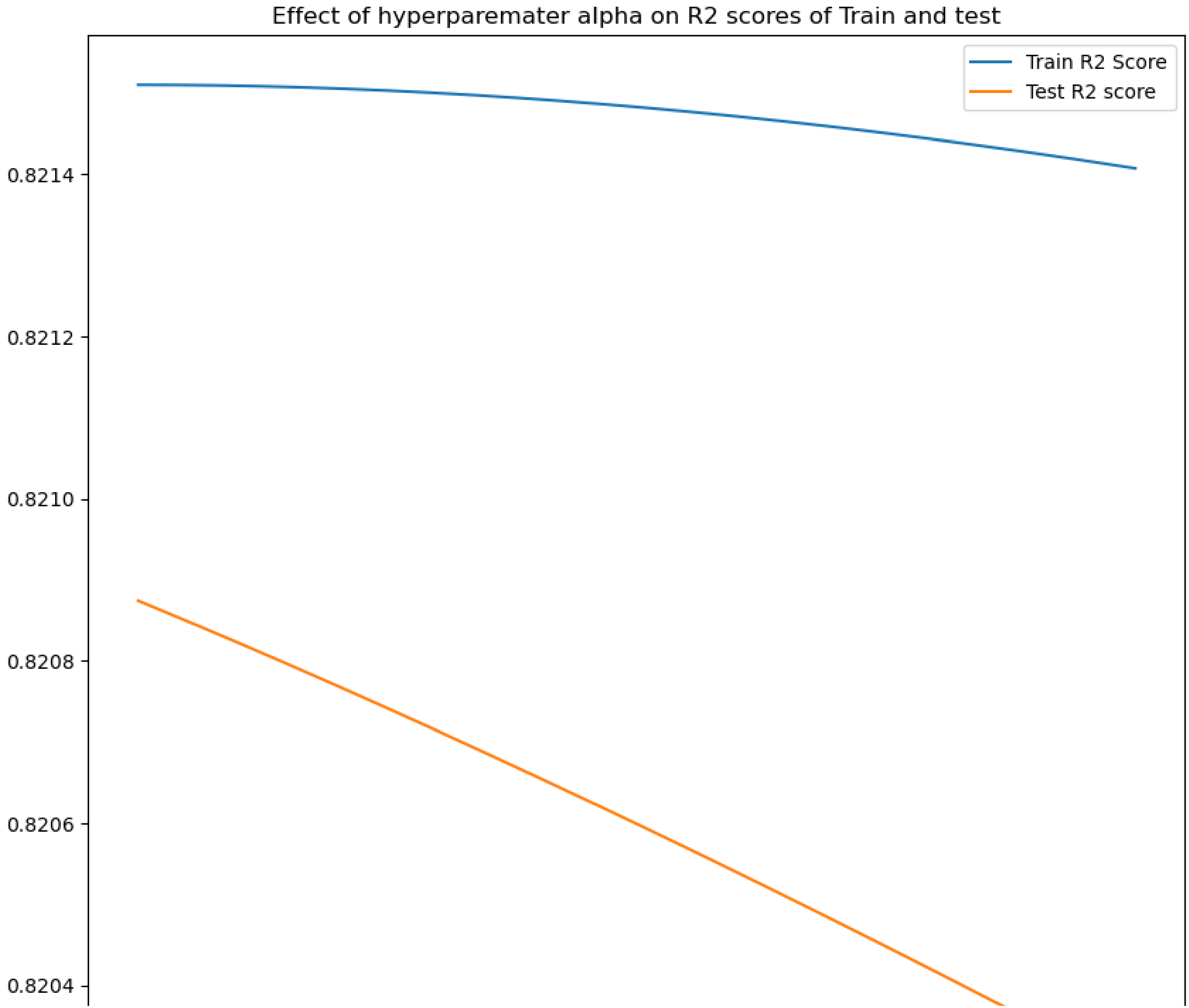
In [55]:
```python
from sklearn.linear_model import Ridge  # L2 regualrization
from sklearn.linear_model import Lasso  # L1 regualrization
from sklearn.linear_model import ElasticNet
```

L2 regularization

Ridge regression :

Hyperparameter Tuning : for appropriate lambda value :

In [56]:
```
train_R2_score = []
test_R2_score = []
lambdas = []
train_test_difference_Of_R2 =  []
lambda_ = 0
while lambda_ <= 5:
    lambdas.append(lambda_)
    RidgeModel = Ridge(lambda_)
    RidgeModel.fit(X_train,y_train)
    trainR2 = RidgeModel.score(X_train,y_train)
    testR2 = RidgeModel.score(X_test,y_test)
    train_R2_score.append(trainR2)
    test_R2_score.append(testR2)

    lambda_ += 0.01

plt.figure(figsize = (10,10))
sns.lineplot(lambdas,train_R2_score,)
sns.lineplot(lambdas, test_R2_score)
plt.legend(['Train R2 Score','Test R2 score'])
plt.title("Effect of hyperparemater alpha on R2 scores of Train and test")

plt.show()
```


Effect of hyperparemater alpha on R2 scores of Train and test
Train R2 Score
Test R2 score
0.8214
0.8212
0.8210
0.8208
0.8206
0.8204

In [57]:
```
RidgeModel = Ridge(alpha = 0.1)
RidgeModel.fit(X_train,y_train)
trainR2 = RidgeModel.score(X_train,y_train)
testR2 = RidgeModel.score(X_test,y_test)

trainR2,testR2
```

Out[57]:
```
(0.8215098726041209, 0.8208639536156423)
```

In [58]:
```
RidgeModel.coef_
```

Out[58]:
```
array([[0.02069489, 0.01929637, 0.00700953, 0.00298992, 0.01334235,
        0.07044884, 0.00987467]])
```

In [59]:
```
RidgeModel_coefs = pd.DataFrame(RidgeModel.coef_.reshape(1,-1),columns=df.columns[:-1])
RidgeModel_coefs["Intercept"] = RidgeModel.intercept_
RidgeModel_coefs
```

Out[59]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Intercept
0	0.020695	0.019296	0.00701	0.00299	0.013342	0.070449	0.009875	0.722882

In [60]:
```
LinearRegression_Model_coefs
```

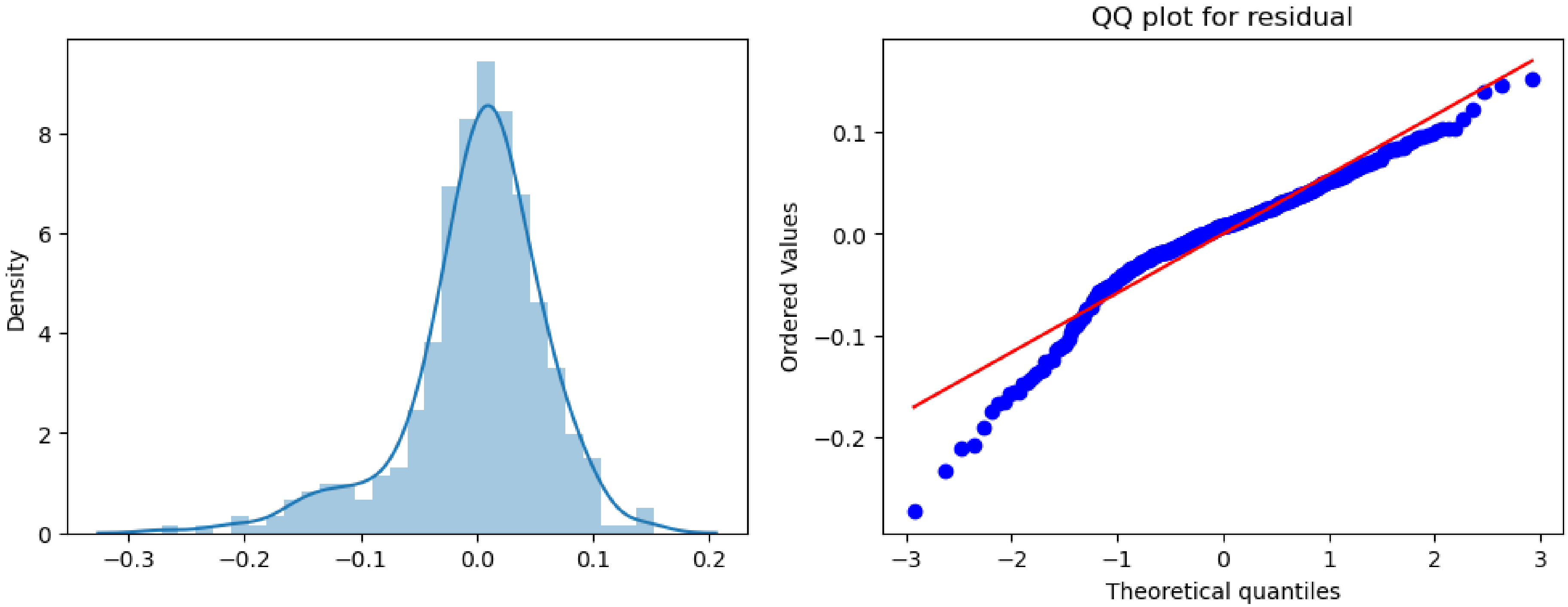
Out[60]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Intercept
0	0.020675	0.019284	0.007001	0.002975	0.013338	0.070514	0.009873	0.722881

In [61]:
```
y_pred = RidgeModel.predict(X_test)

print("MSE:",mean_squared_error(y_test,y_pred)) # MSE
print("RMSE:",np.sqrt(mean_squared_error(y_test,y_pred))) #RMSE
print("MAE :",mean_absolute_error(y_test,y_pred) ) # MAE
```

```
print("r2_score:",r2_score(y_test,y_pred)) # r2score
print("Adjusted R2 score :", AdjustedR2score(r2_score(y_test,y_pred),len(X),X.shape[1]))  # adjusted R2 score

y_predicted = RidgeModel.predict(X_train)

residuals = (y_train - y_predicted)
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
sns.distplot(residuals)
plt.subplot(1,2,2)
stats.probplot(residuals.reshape(-1,), plot = plt)
plt.title('QQ plot for residual')
plt.show()
```

```
MSE: 0.003459296191728331
RMSE: 0.058815781825359854
MAE : 0.040203055117056935
r2_score: 0.8208639536156423
Adjusted R2 score : 0.818315270028873
```

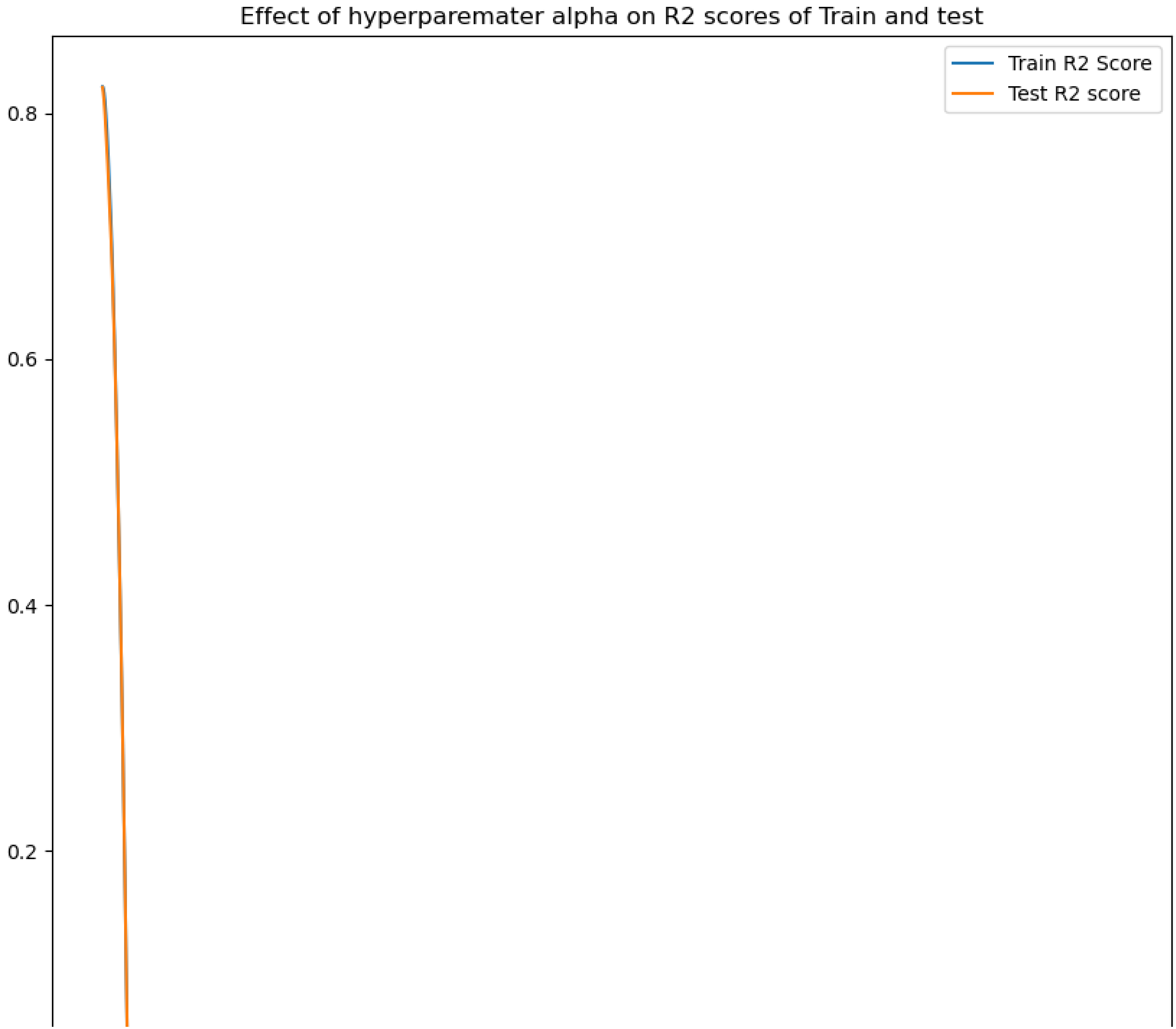


L1 regularization :

Lasso :

Hyperparameter Tuning : for appropriate lambda value :

In [62]:
```
train_R2_score = []
test_R2_score = []
lambdas = []
train_test_difference_Of_R2 =  []
lambda_ = 0
while lambda_ <= 5:
    lambdas.append(lambda_)
    LassoModel = Lasso(alpha=lambda_)
    LassoModel.fit(X_train , y_train)
    trainR2 = LassoModel.score(X_train,y_train)
    testR2 = LassoModel.score(X_test,y_test)
    train_R2_score.append(trainR2)
    test_R2_score.append(testR2)

    lambda_ += 0.001

plt.figure(figsize = (10,10))
sns.lineplot(lambdas,train_R2_score,)
sns.lineplot(lambdas, test_R2_score)
plt.legend(['Train R2 Score','Test R2 score'])
plt.title("Effect of hyperparemater alpha on R2 scores of Train and test")


plt.show()
```


Effect of hyperparemater alpha on R2 scores of Train and test
Train R2 Score
Test R2 score
0.8
0.6
0.4
0.2

In [63]:
```
LassoModel = Lasso(alpha=0.001)
LassoModel.fit(X_train , y_train)
trainR2 = LassoModel.score(X_train,y_train)
testR2 = LassoModel.score(X_test,y_test)

trainR2,testR2
```

Out[63]: (0.82142983289567, 0.8198472607571161)

In [64]:
```
Lasso_Model_coefs = pd.DataFrame(LassoModel.coef_.reshape(1,-1),columns=df.columns[:-1])
Lasso_Model_coefs["Intercept"] = LassoModel.intercept_
Lasso_Model_coefs
```

Out[64]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Intercept
0	0.020616	0.019069	0.006782	0.002808	0.012903	0.070605	0.009278	0.722863

In [65]:
```
RidgeModel_coefs
```

Out[65]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Intercept
0	0.020695	0.019296	0.00701	0.00299	0.013342	0.070449	0.009875	0.722882

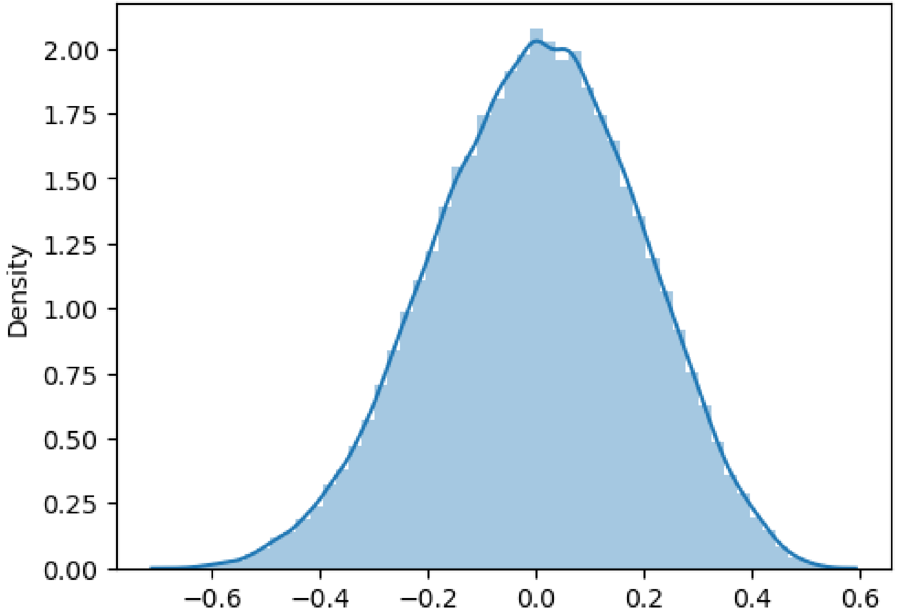
In [66]:
```
LinearRegression_Model_coefs
```

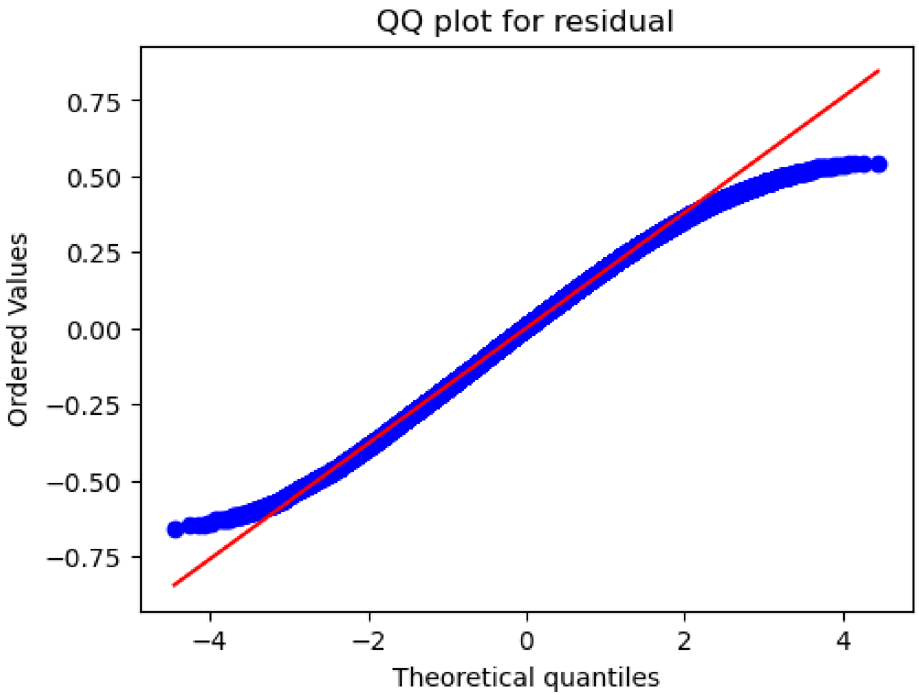
Out[66]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Intercept
0	0.020675	0.019284	0.007001	0.002975	0.013338	0.070514	0.009873	0.722881

In [67]:
```
y_predicted = LassoModel.predict(X_train)
```

```
residuals = (y_train - y_predicted)
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
sns.distplot(residuals)
plt.subplot(1,2,2)
stats.probplot(residuals.reshape(-1,), plot = plt)
plt.title('QQ plot for residual')
plt.show()

y_pred = LassoModel.predict(X_test)

print("MSE:",mean_squared_error(y_test,y_pred)) # MSE
print("RMSE:",np.sqrt(mean_squared_error(y_test,y_pred))) #RMSE
print("MAE :",mean_absolute_error(y_test,y_pred) ) # MAE
print("r2_score:",r2_score(y_test,y_pred)) # r2score
print("Adjusted R2 score :", AdjustedR2score(r2_score(y_test,y_pred),len(X),X.shape[1]))  # adjusted R2 score
```





```
MSE: 0.0034789295475193297
RMSE: 0.058982451182697807
MAE : 0.04022896061335951
r2_score: 0.8198472607571161
Adjusted R2 score : 0.8172841120280507
```

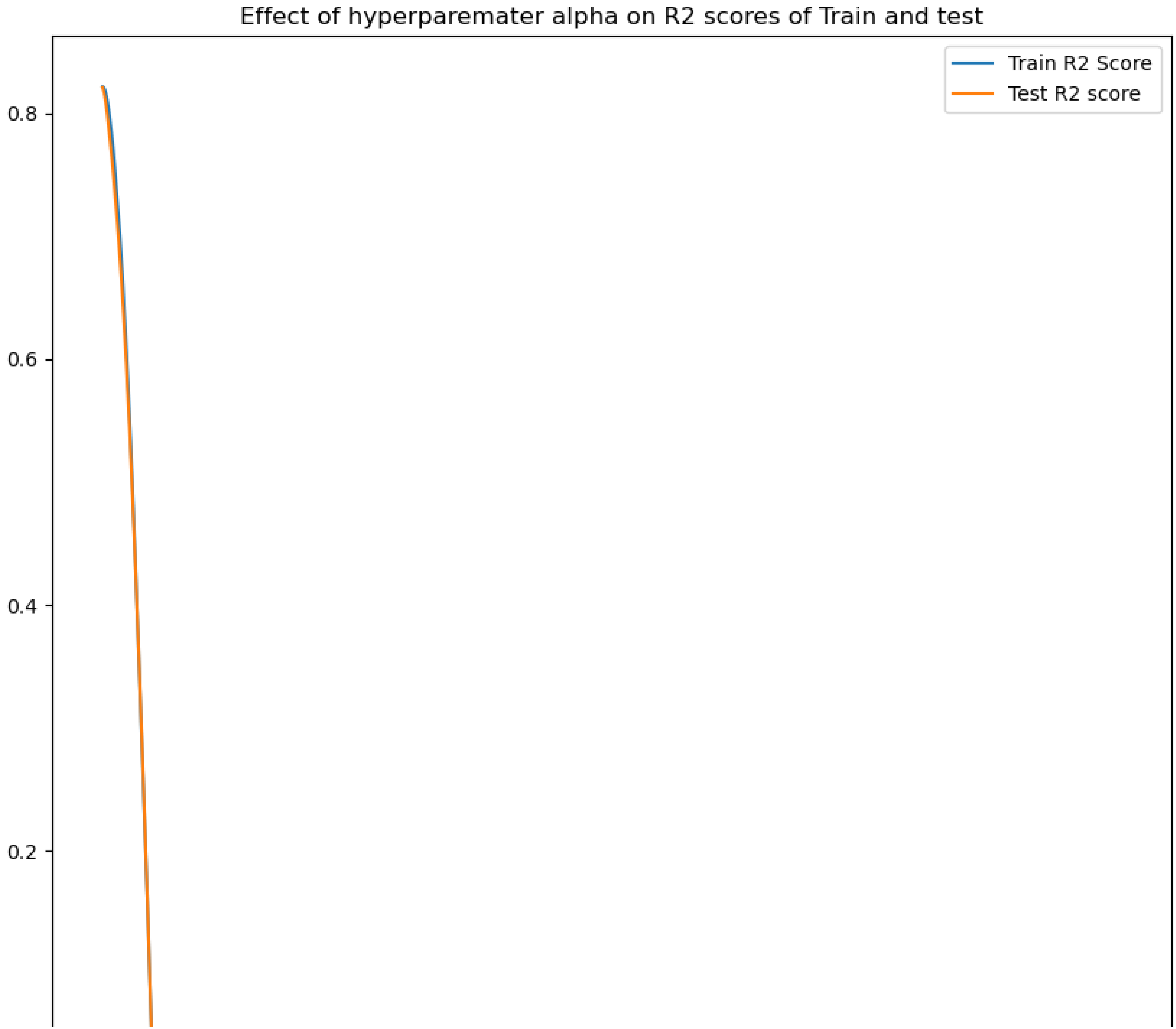
ElasticNet

L1 and L2 regularisation :

> > Elastic net linear regression uses the penalties from both the lasso and ridge techniques to regularize regression models.

Hyperparameter Tuning : for appropriate lambda value :

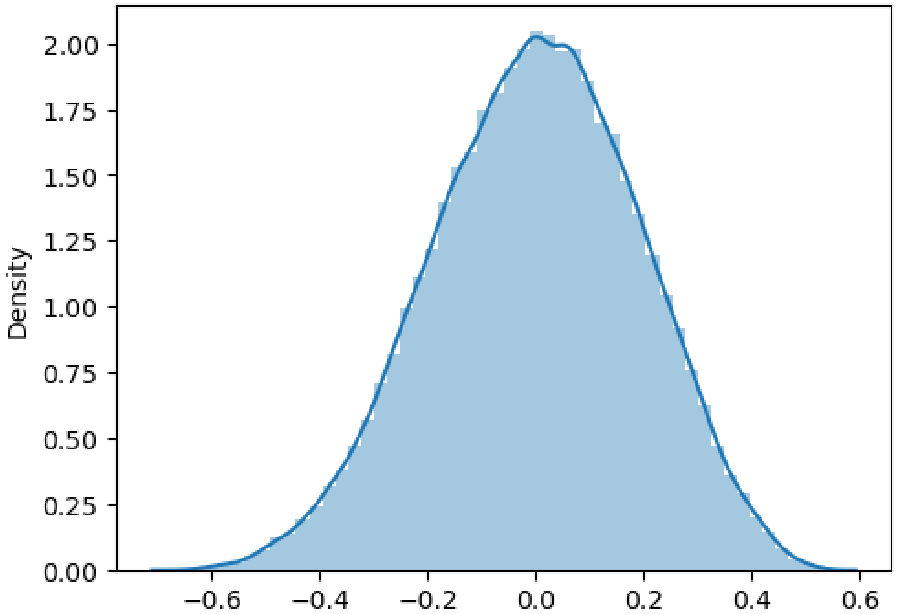
In [68]:
```
train_R2_score = []
test_R2_score = []
lambdas = []
train_test_difference_Of_R2 =  []
lambda_ = 0
while lambda_ <= 5:
    lambdas.append(lambda_)
    ElasticNet_model = ElasticNet(alpha=lambda_)
    ElasticNet_model.fit(X_train , y_train)
    trainR2 = ElasticNet_model.score(X_train,y_train)
    testR2 = ElasticNet_model.score(X_test,y_test)
    train_R2_score.append(trainR2)
    test_R2_score.append(testR2)

    lambda_ += 0.001

plt.figure(figsize = (10,10))
sns.lineplot(lambdas,train_R2_score,)
sns.lineplot(lambdas, test_R2_score)
plt.legend(['Train R2 Score','Test R2 score'])
plt.title("Effect of hyperparemater alpha on R2 scores of Train and test")

plt.show()
```

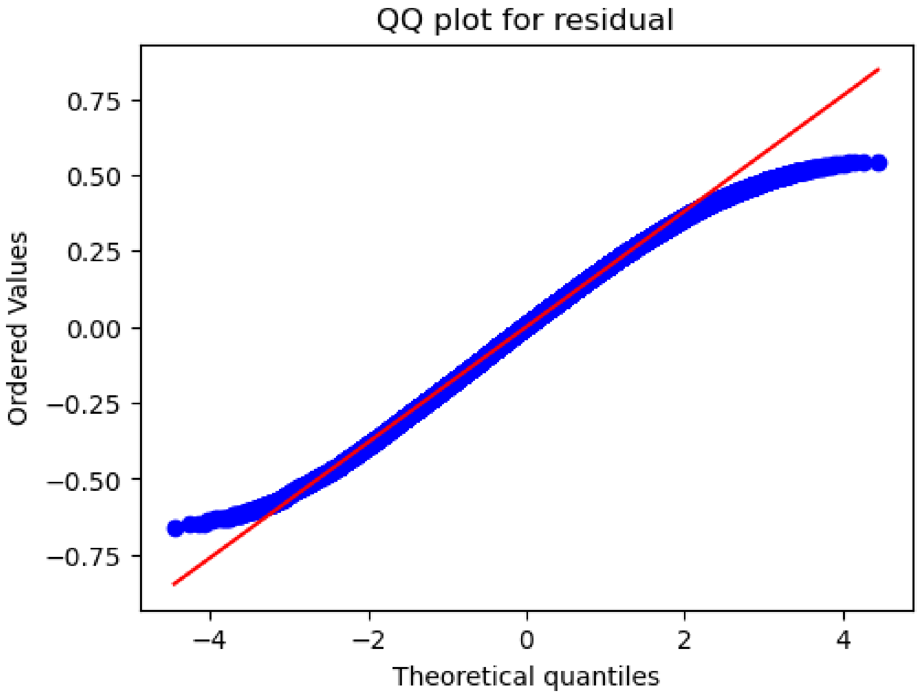

Effect of hyperparemater alpha on R2 scores of Train and test
Train R2 Score
Test R2 score
0.8
0.6
0.4
0.2

In [69]:
```
ElasticNet_model = ElasticNet(alpha=0.001)
ElasticNet_model.fit(X_train , y_train)
trainR2 = ElasticNet_model.score(X_train,y_train)
testR2 = ElasticNet_model.score(X_test,y_test)

trainR2, testR2
```

Out[69]:
```
(0.8214893364453533, 0.8203602261096284)
```

In [70]:
```
y_predicted = ElasticNet_model.predict(X_train)

residuals = (y_train - y_predicted)
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
sns.distplot(residuals)
plt.subplot(1,2,2)
stats.probplot(residuals.reshape(-1,), plot = plt)
plt.title('QQ plot for residual')
plt.show()

y_pred = ElasticNet_model.predict(X_test)

print("MSE:",mean_squared_error(y_test,y_pred)) # MSE
print("RMSE:",np.sqrt(mean_squared_error(y_test,y_pred))) #RMSE
print("MAE :",mean_absolute_error(y_test,y_pred) ) # MAE
print("r2_score:",r2_score(y_test,y_pred)) # r2score
print("Adjusted R2 score :", AdjustedR2score(r2_score(y_test,y_pred),len(X),X.shape[1]))  # adjusted R2 score
```






```
MSE: 0.003469023673596966
RMSE: 0.058898418260569324
MAE : 0.04021407699792928
r2_score: 0.8203602261096284
Adjusted R2 score : 0.8178043756680987
```

In [71]:
```
ElasticNet_model_coefs = pd.DataFrame(ElasticNet_model.coef_.reshape(1,-1),columns=df.columns[:-1])
ElasticNet_model_coefs["Intercept"] = ElasticNet_model.intercept_
ElasticNet_model_coefs
```

Out[71]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Intercept
0	0.020679	0.019199	0.006908	0.00292	0.013128	0.070437	0.009581	0.722873

In [72]:
```
RidgeModel_coefs
```

Out[72]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Intercept
0	0.020695	0.019296	0.00701	0.00299	0.013342	0.070449	0.009875	0.722882

In [73]:

```
Lasso_Model_coefs
```

Out[73]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Intercept
0	0.020616	0.019069	0.006782	0.002808	0.012903	0.070605	0.009278	0.722863

In [74]:

```
LinearRegression_Model_coefs
```

Out[74]:

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Intercept
0	0.020675	0.019284	0.007001	0.002975	0.013338	0.070514	0.009873	0.722881

In [75]:

```
y_pred = ElasticNet_model.predict(X_test)
ElasticNet_model_metrics = []
ElasticNet_model_metrics.append(mean_squared_error(y_test,y_pred)) # MSE
ElasticNet_model_metrics.append(np.sqrt(mean_squared_error(y_test,y_pred))) #RMSE
ElasticNet_model_metrics.append(mean_absolute_error(y_test,y_pred) ) # MAE
ElasticNet_model_metrics.append(r2_score(y_test,y_pred)) # r2score
ElasticNet_model_metrics.append(AdjustedR2score(r2_score(y_test,y_pred),len(X),X.shape[1]))  # adjusted R2 score

y_pred = LinearRegression.predict(X_test)
LinearRegression_model_metrics = []
LinearRegression_model_metrics.append(mean_squared_error(y_test,y_pred)) # MSE
LinearRegression_model_metrics.append(np.sqrt(mean_squared_error(y_test,y_pred))) #RMSE
LinearRegression_model_metrics.append(mean_absolute_error(y_test,y_pred) ) # MAE
LinearRegression_model_metrics.append(r2_score(y_test,y_pred)) # r2score
LinearRegression_model_metrics.append(AdjustedR2score(r2_score(y_test,y_pred),len(X),X.shape[1]))  # adjusted R2 score

y_pred = RidgeModel.predict(X_test)
RidgeModel_model_metrics = []
RidgeModel_model_metrics.append(mean_squared_error(y_test,y_pred)) # MSE
RidgeModel_model_metrics.append(np.sqrt(mean_squared_error(y_test,y_pred))) #RMSE
RidgeModel_model_metrics.append(mean_absolute_error(y_test,y_pred) ) # MAE
RidgeModel_model_metrics.append(r2_score(y_test,y_pred)) # r2score
RidgeModel_model_metrics.append(AdjustedR2score(r2_score(y_test,y_pred),len(X),X.shape[1]))  # adjusted R2 score

y_pred = LassoModel.predict(X_test)
LassoModel_model_metrics = []
LassoModel_model_metrics.append(mean_squared_error(y_test,y_pred)) # MSE
LassoModel_model_metrics.append(np.sqrt(mean_squared_error(y_test,y_pred))) #RMSE
LassoModel_model_metrics.append(mean_absolute_error(y_test,y_pred) ) # MAE
```

```
LassoModel_model_metrics.append(r2_score(y_test,y_pred)) # r2score
LassoModel_model_metrics.append(AdjustedR2score(r2_score(y_test,y_pred),len(X),X.shape[1]))  # adjusted R2 score
```

In [76]:
```
ElasticNet_model_metrics
```

Out[76]:
```
[0.003469023673596966,
 0.058898418260569324,
 0.04021407699792928,
 0.8203602261096284,
 0.8178043756680987]
```

In [77]:
```
A = pd.DataFrame([LinearRegression_model_metrics,LassoModel_model_metrics,RidgeModel_model_metrics,ElasticNet_model_metrics],colu
A
```

Out[77]:

	MSE	RMSE	MAE	R2_SCORE	ADJUSTED_R2
Linear Regression Model	0.003459	0.058814	0.040200	0.820874	0.818326
Lasso Regression Model	0.003479	0.058982	0.040229	0.819847	0.817284
Ridge Regression Model	0.003459	0.058816	0.040203	0.820864	0.818315
ElasticNet Regression Model	0.003469	0.058898	0.040214	0.820360	0.817804

In [78]:
```
B = pd.DataFrame(LinearRegression_Model_coefs.append(Lasso_Model_coefs).append(RidgeModel_coefs).append(ElasticNet_model_coefs))
B.index = ["Linear Regression Model","Lasso Regression Model","Ridge Regression Model","ElasticNet Regression Model"]
```

In [79]:
```
REPORT = B.reset_index().merge(A.reset_index())
```

In [80]:
```
REPORT = REPORT.set_index("index")
REPORT
```

Out[80]:

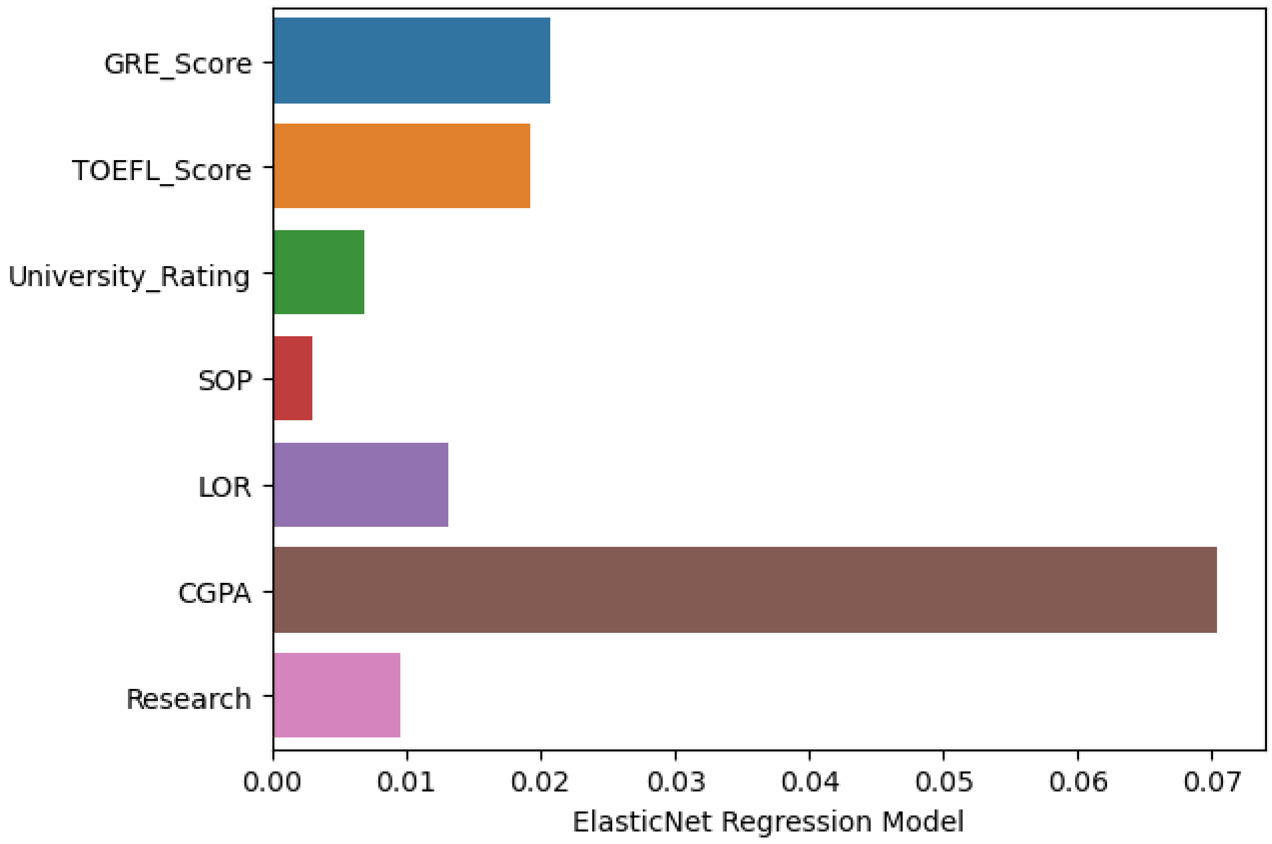
index	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	Intercept	MSE	RMSE	MAE	R2_SCORE	ADJU
Linear Regression Model	0.020675	0.019284	0.007001	0.002975	0.013338	0.070514	0.009873	0.722881	0.003459	0.058814	0.040200	0.820874	
Lasso Regression Model	0.020616	0.019069	0.006782	0.002808	0.012903	0.070605	0.009278	0.722863	0.003479	0.058982	0.040229	0.819847	
Ridge Regression Model	0.020695	0.019296	0.007010	0.002990	0.013342	0.070449	0.009875	0.722882	0.003459	0.058816	0.040203	0.820864	
ElasticNet Regression Model	0.020679	0.019199	0.006908	0.002920	0.013128	0.070437	0.009581	0.722873	0.003469	0.058898	0.040214	0.820360	

In [81]:

```
sns.barplot(y  = REPORT.loc["ElasticNet Regression Model"][0:7].index,
            x = REPORT.loc["ElasticNet Regression Model"][0:7])
```

Out[81]:

```
<AxesSubplot:xlabel='ElasticNet Regression Model'>
```



Insights and Recommendations

Insights , Feature Importance and Interpretations and Recommendations

> > - The first column was observed as a unique row identifier which was dropped as it was not required for model building.
> > - University Rating, SOP and LOR strength and research seems to be discrete random Variables, but also ordinal numeric data remaining features are numeric, ordinal, and continuous.
> > - No null values were present in the dataset.
> > - No Significant amount of outliers were found in the data.
> > - Chance of admission(target variable) and GRE score(an independent feature) are nearly normally distributed.

```
 - Independent Variables (Input data): GRE Score, TOEFL Score, University Rating, SOP,
LOR, CGPA, Research
 - Target/Dependent Variable: Chance of Admit (the value we want to predict)
```

- From the correlation heatmap, GRE score, TOEFL score, and CGPA have a very high correlation with Change of admission as observed.
- University rating, SOP, LOR, and Research have comparatively slightly less correlated than other features.
- Chances of admission is a probability measure, which is within 0 to 1 which is good (no outliers or misleading data in column).
- Range of GRE scores looks between 290 to 340, range of TOEFL scores is between 92 to 120 while university rating, SOP, and LOR are distributed between a range of 1 to 5. and CGPA range is between 6.8 to 9.92.
- From boxplots (distribution of chance of admiration (probability of getting admission) as per GRE score): with a higher GRE score, there is a high probability of getting admission.
- Students having high TOEFL score, has a higher probability of getting admission.
- From the count plots, we can observe, the statement of purpose SOP strength is positively correlated with the Chance of Admission.
- We can also discover a similar pattern in Letter of Recommendation strength and University rating, which have a positive correlation with Chances of Admission.
- Students having research has higher chances of Admission, but also we can observe some outliers within that category.

Actionable Insights and Recommendations

- Educational institutes can not just help the student to improve their CGPA score but also assist them in writing good LOR and SOP thus helping them admit to a better university.
- The educational institute can not just help the student to improve their GRE Score but can also assist them in writing good LOR and SOP thus helping them admit to a better University.
- Awareness of CGPA and Research Capabilities: Seminars can be organized to increase awareness regarding CGPA and Research capabilities to enhance the chance of admission.
- Any student can never change their current state of attributes so awareness and marketing campaigns need to be surveyed hence creating a first impression on students at the undergraduate level, which won't just increase the company's popularity but will also help students get prepared for future plans in advance.

> - A dashboard can be created for students whenever they log in to your website, hence allowing healthy competition also to create a progress report for students.
> - Additional features like the number of hours they put in studying, watching lectures, assignments solved percentage, and marks in mock tests can result in a better report for every student to judge themselves and improve on their own.

Regression Analysis

> - From the regression analysis (above bar chart and REPORT file), we can observe that CGPA is the most important feature for predicting the chances of admission.
> - Another important features are GRE and TOEFL scores.
> - After the first Regression Model, checked for Multicolinearity. Getting all the VIF scores below 5, showing there's no high multicolinearity.
> - All the residuals are not perfectly normally distributed. and so residual plot we can observe some level of heteroscedasticity.
> - Regularised model ridge and lasso both give very similar results to Linear Regression Model.
> - Similarly, ElasticNet (L1+L2) also returns very similar results. along with the rest of the model metrics.