In [57]:
```
import pandas as pd
import numpy as np
from numpy import nan, NaN,NAN
from matplotlib import pyplot as plt
import seaborn as sns
import warnings
import scipy
warnings.filterwarnings("ignore")
import statsmodels.api as sm
import math
```

In [2]:
```
orig_df=pd.read_csv("Jamboree_Admission.csv")
```

In [3]:
```
df=orig_df.copy()
df.head()
```

Out[3]:

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

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

Out[4]: (500, 9)

In [5]:
```
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 [6]:
```
#drop Serial No> column
df.drop("Serial No.",axis=1,inplace=True)
```

In [7]:
```
#Chance of Admit  column renamed as an extra space found at its end
df.rename(columns={"Chance of Admit ":"Chance of Admit"},inplace=True)
```

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

Out[8]:

	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

Comparing the mean and median of the independent Variables suggest not much of outliers could be there.The same is rechecked in the boxplot of these variables below

In [9]:
```
#Missing Value Detection
df.isna().sum()
```

Out[9]:
```
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

No missing values are present in any of the columns

In [10]:
```
##Checking Duplicates
df.loc[df.sort_values(["GRE Score"]).duplicated()==True]
```

Out[10]:

GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit

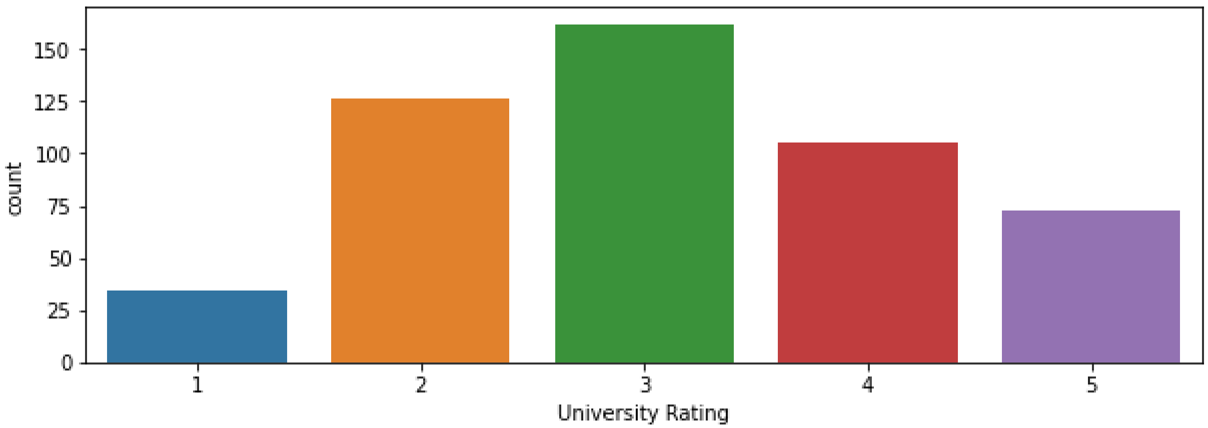
No duplicates are present

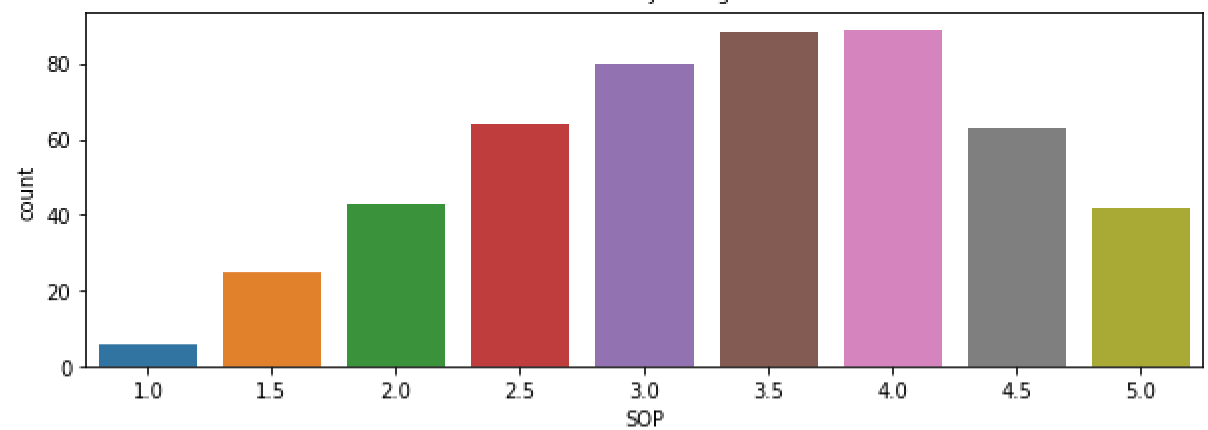
In [11]:
```python
#Categorical columns and % of each values there
for col in df.columns:
    if col in("GRE Score","TOEFL Score","CGPA","Chance of Admit"):
        continue
    else:
        print("% of each of the unique values in column",col)
        print(df[col].value_counts(normalize=True)*100)
        print("*"*50)
```

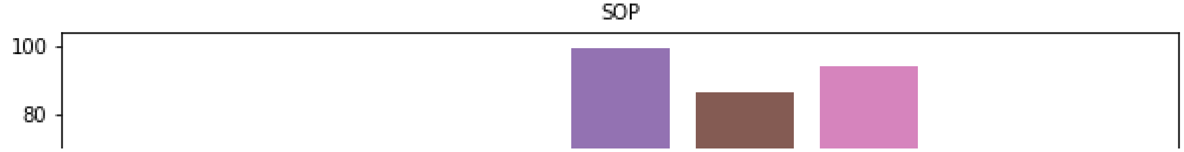
```
% of each of the unique values in column University Rating
3    32.4
2    25.2
4    21.0
5    14.6
1     6.8
Name: University Rating, dtype: float64
**************************************************
% of each of the unique values in column SOP
4.0    17.8
3.5    17.6
3.0    16.0
2.5    12.8
4.5    12.6
2.0     8.6
5.0     8.4
1.5     5.0
1.0     1.2
Name: SOP, dtype: float64
**************************************************
% of each of the unique values in column LOR
3.0    19.8
4.0    18.8
3.5    17.2
4.5    12.6
2.5    10.0
5.0    10.0
2.0     9.2
1.5     2.2
1.0     0.2
Name: LOR , dtype: float64
**************************************************
% of each of the unique values in column Research
```

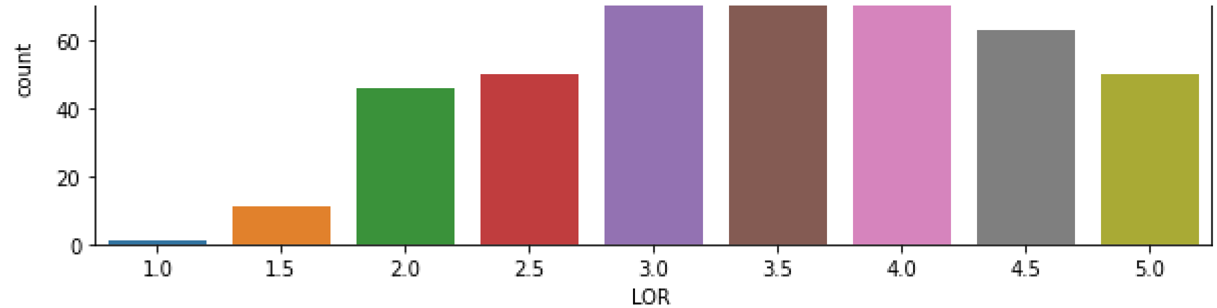
```
1    56.0
0    44.0
Name: Research, dtype: float64
**************************************************
```

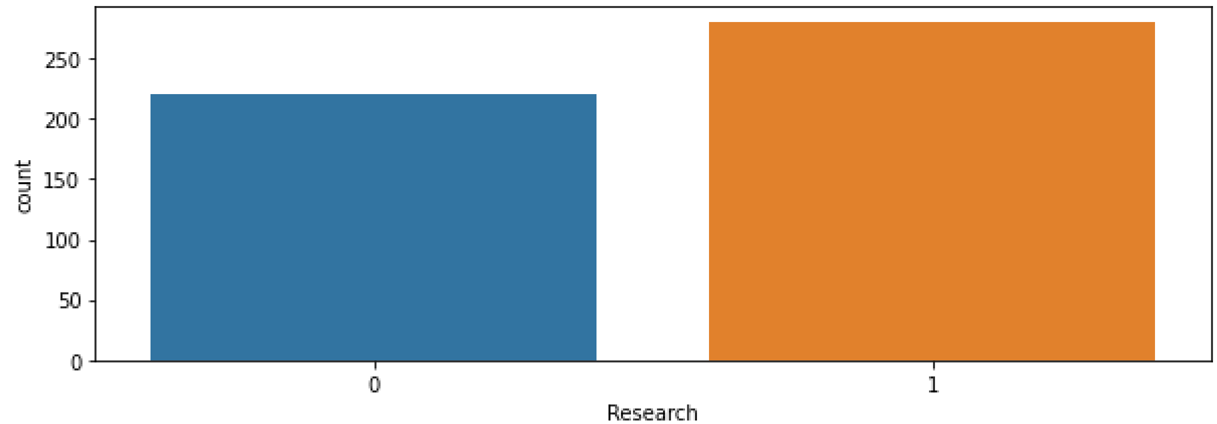
In [115]:
```
#Univariate analysis of categorical columns
i=411
plt.rcParams["figure.figsize"] = (10,15)
for col in df.columns:

    if col in("GRE Score","TOEFL Score","CGPA","Chance of Admit"):
        continue
    else:
        plt.subplot(i)

        sns.countplot(df[col])
        i+=1
```











Observations from Univariate and Non Graphical representations of categorical Variables

1)The Categorical columns identified are University Rating,SOP,LOR and Research

2)University Rating -3 topped the list with 33% applicants and least is from rating 1

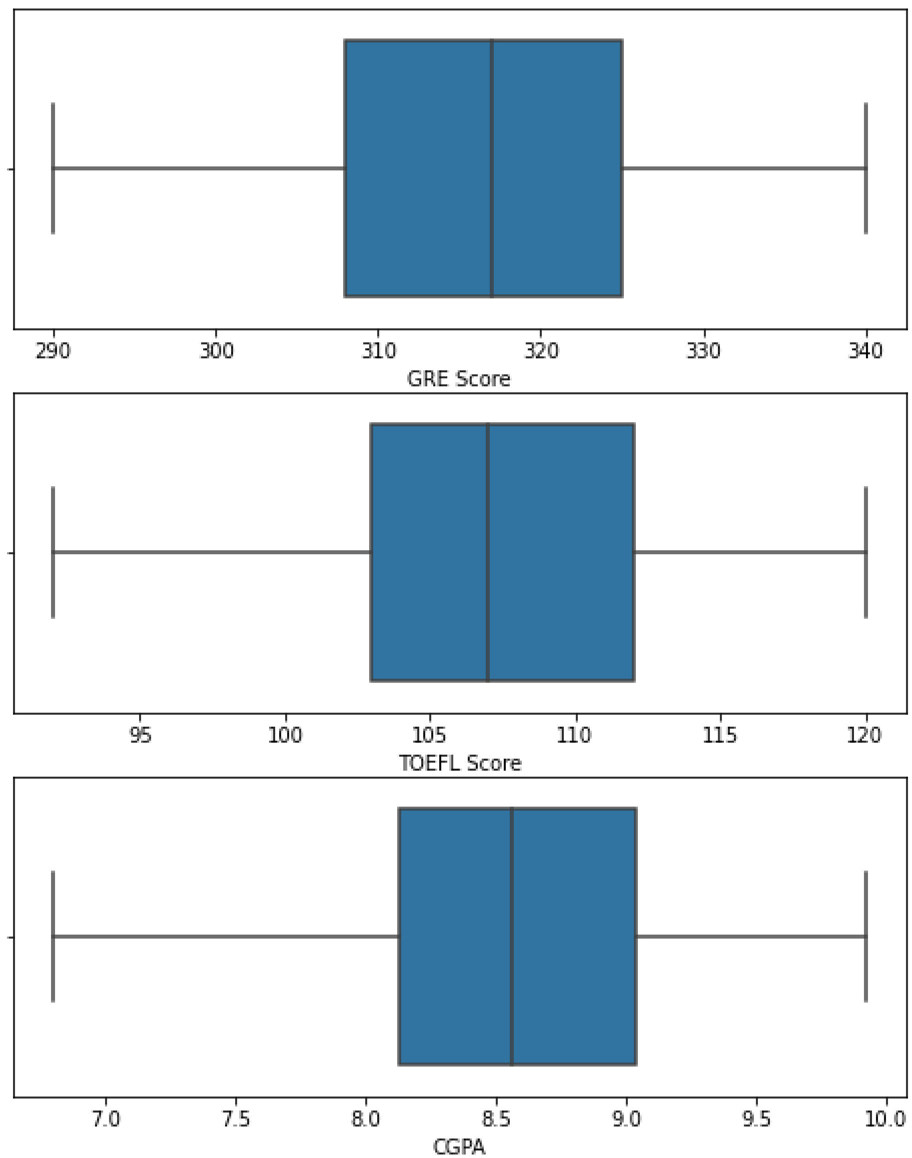
3)Applicants with SOP rating 4 and 3.5 has applied more to the graduate programs

4)Applicants with LOR rating 3 and 4 has applied more for the graduate programs

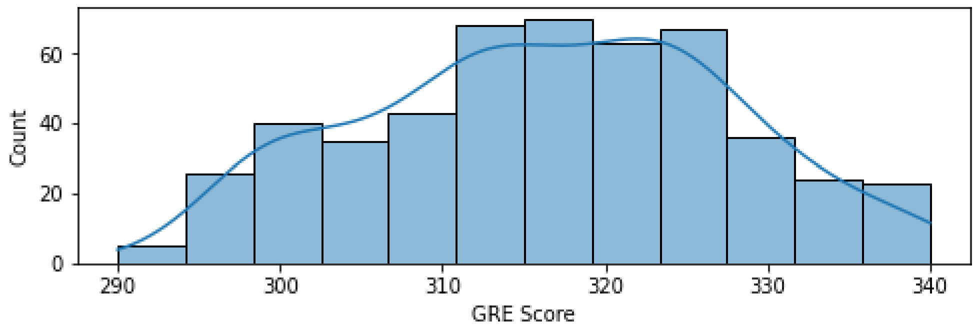
5)Applicants who has done some research work has applied more ,However applicants not done any research work has also applied with a ratio of 14:11

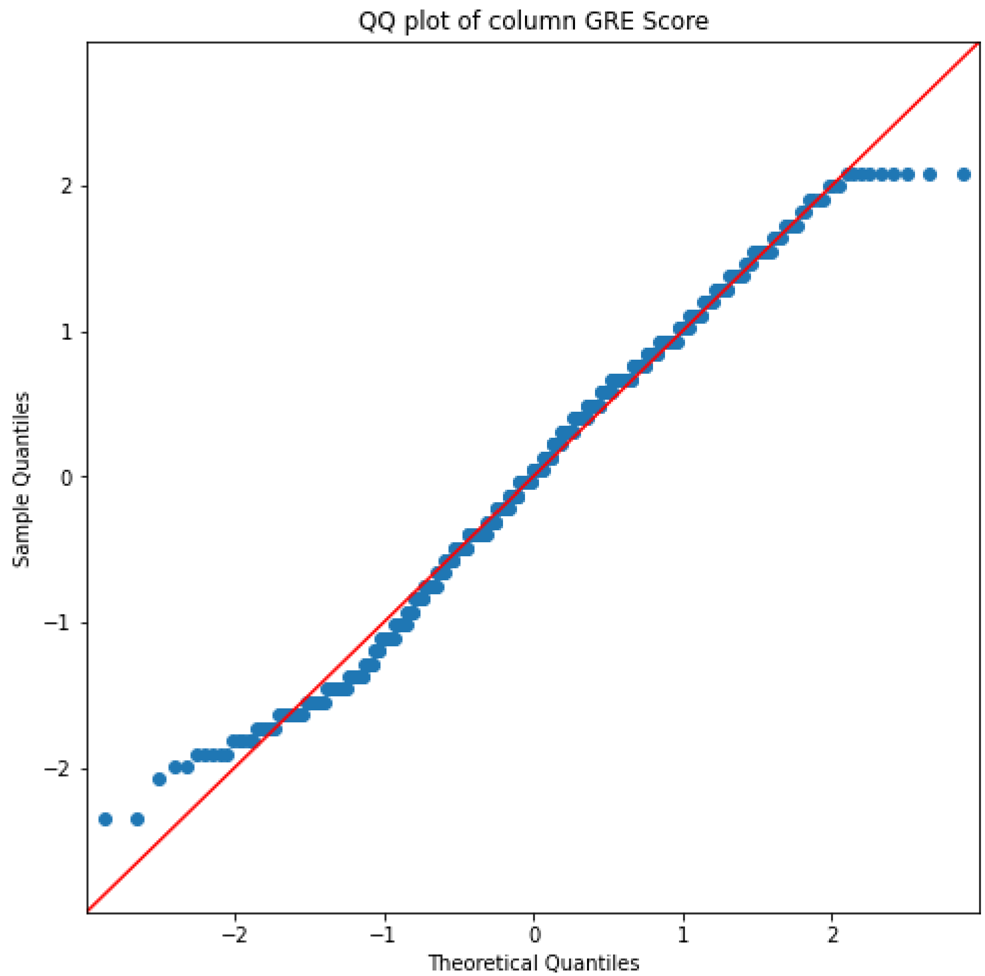
In [13]:

```
i=311

for col in ("GRE Score","TOEFL Score","CGPA"):
    plt.rcParams["figure.figsize"] = (8,10)
    plt.subplot(i)
    sns.boxplot(df[col])
    i+=1
```

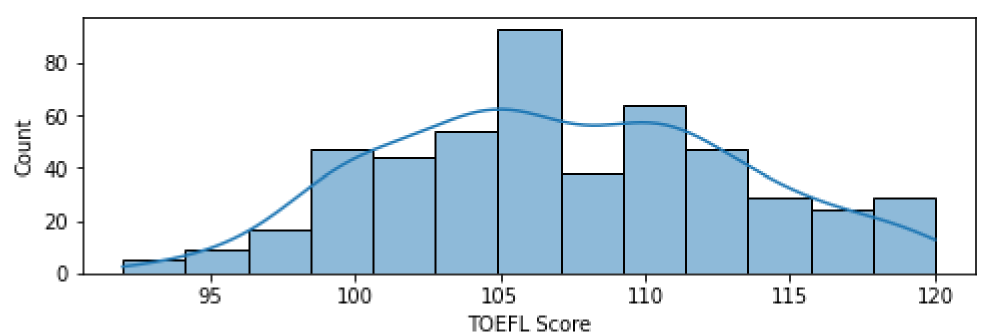


In [14]:
```
#Univariate of continous features
i=311

for col in ("GRE Score","TOEFL Score","CGPA"):
    plt.rcParams["figure.figsize"] = (8,8)
    plt.subplot(i)
    sns.histplot(df[col],kde=True)

    fig=sm.qqplot(df[col],line='45',fit=True)
    ttle="QQ plot of column "+col
    plt.title(ttle)

    plt.show()
    i+=1
```

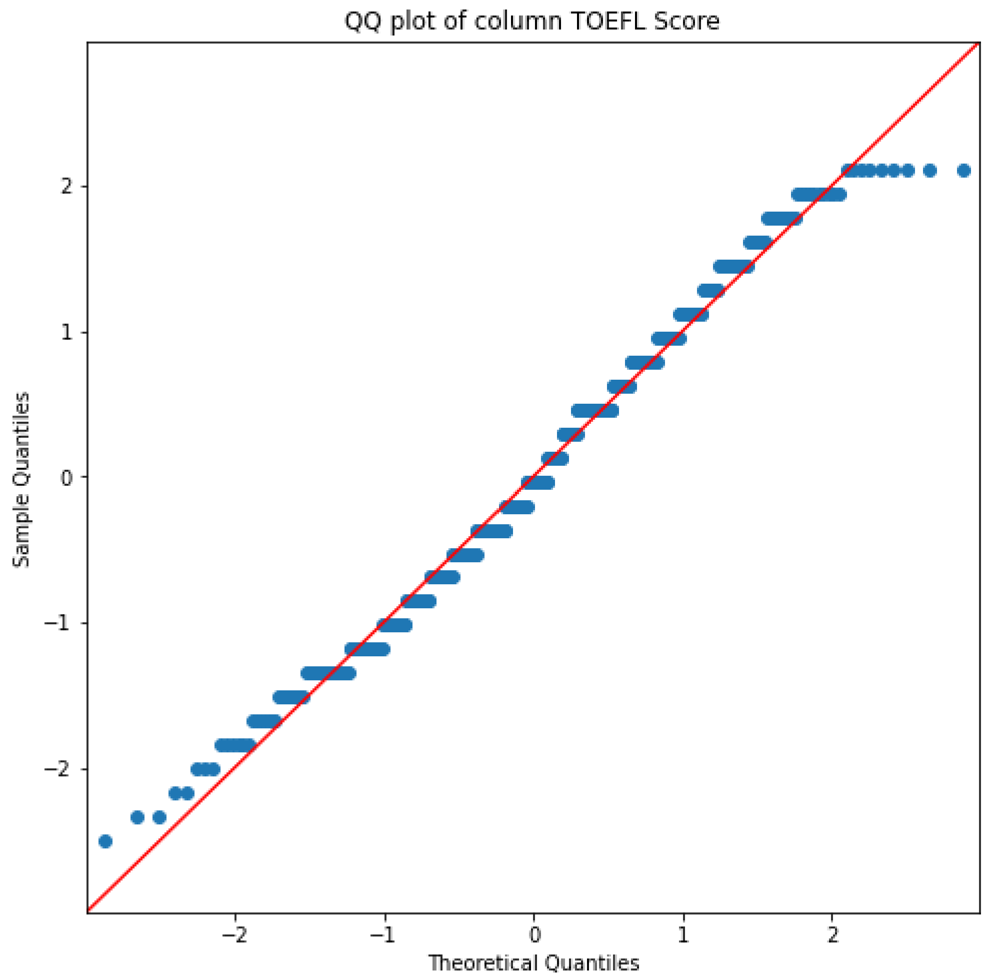




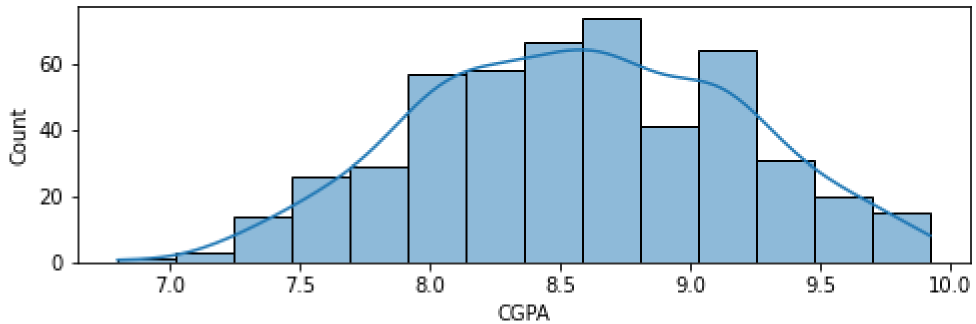
QQ plot of column GRE Score
Sample Quantiles
Theoretical Quantiles
−2
−1
0
1
2



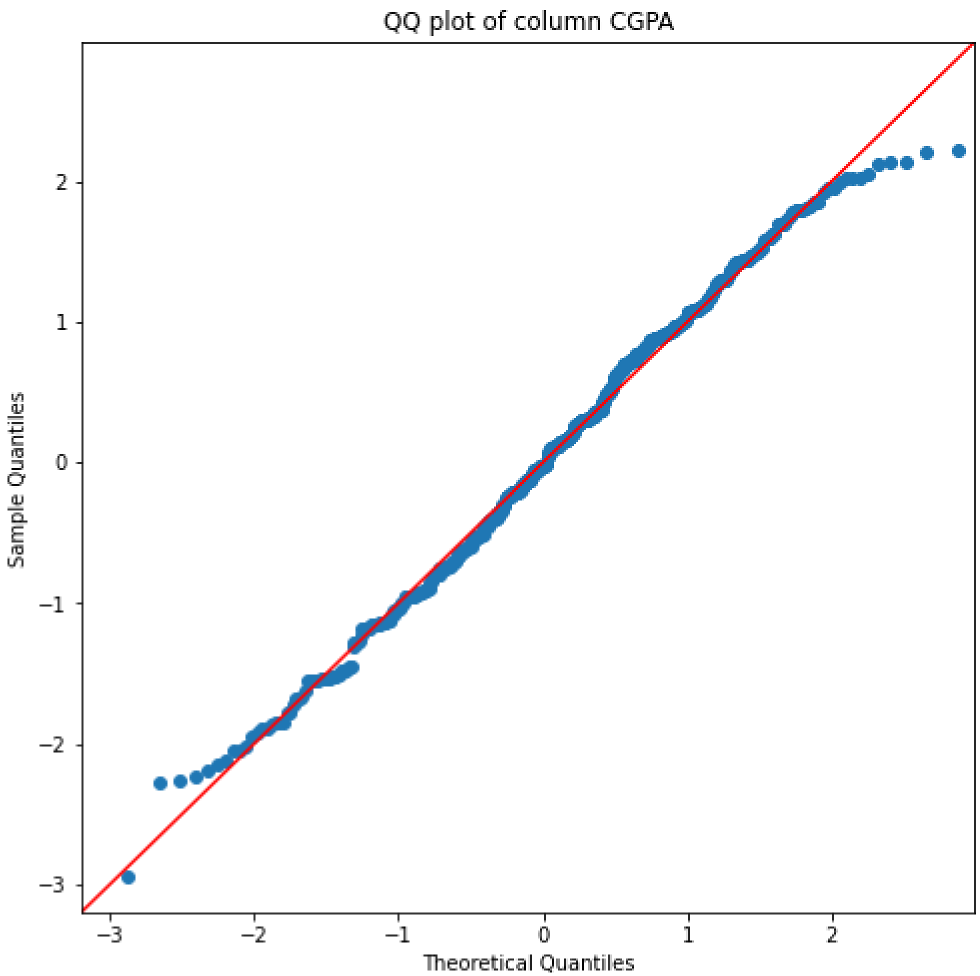
Count
TOEFL Score
0
20
40
60
80
95
100
105
110
115
120



QQ plot of column TOEFL Score
Sample Quantiles
Theoretical Quantiles
2
1
0
−1
−2
−2
−1
0
1
2



Count
0
20
40
60
7.0
7.5
8.0
8.5
9.0
9.5
10.0
CGPA



QQ plot of column CGPA
Sample Quantiles
Theoretical Quantiles
−3
−2
−1
0
1
2

Observations from Univariate Analysis of continous Features

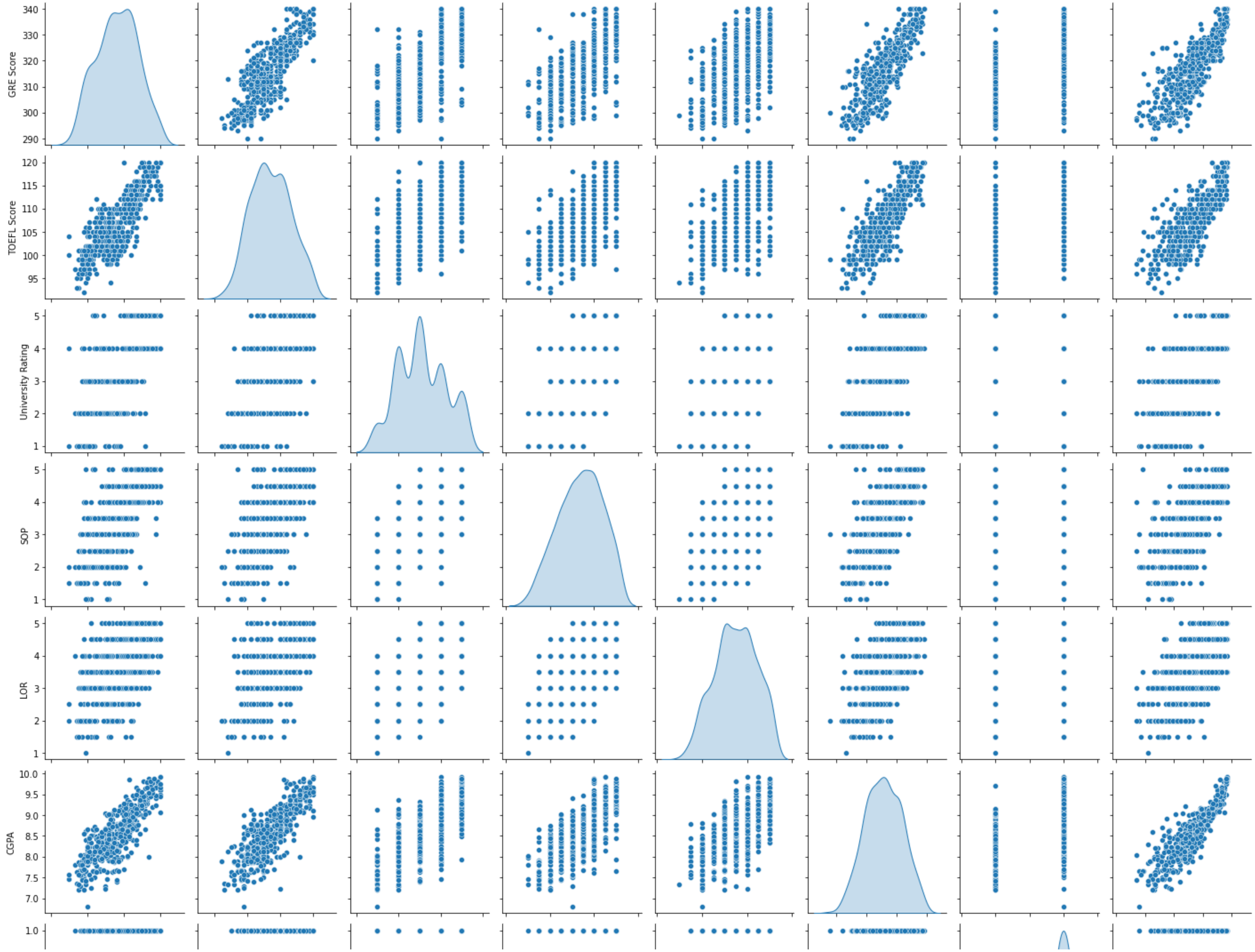
1)GRE Score,TOEFL Score and CGPA rating are observed as continous features

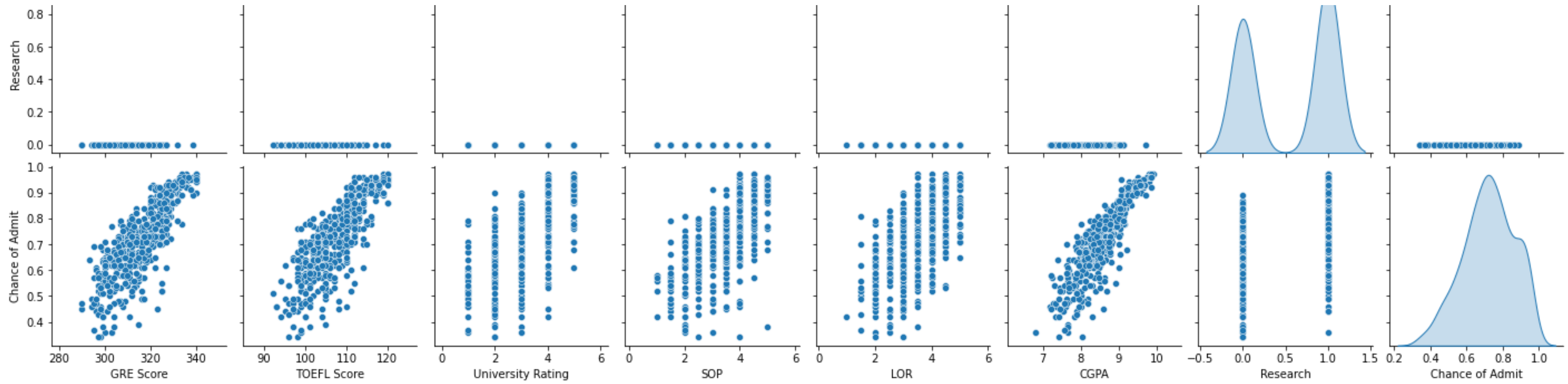
2)The boxplot of these columns suggests there ae no outliers in data.

3)The Density plots suggests that these features doesn't follow a perfect gaussian.The QQplots also confirms the same

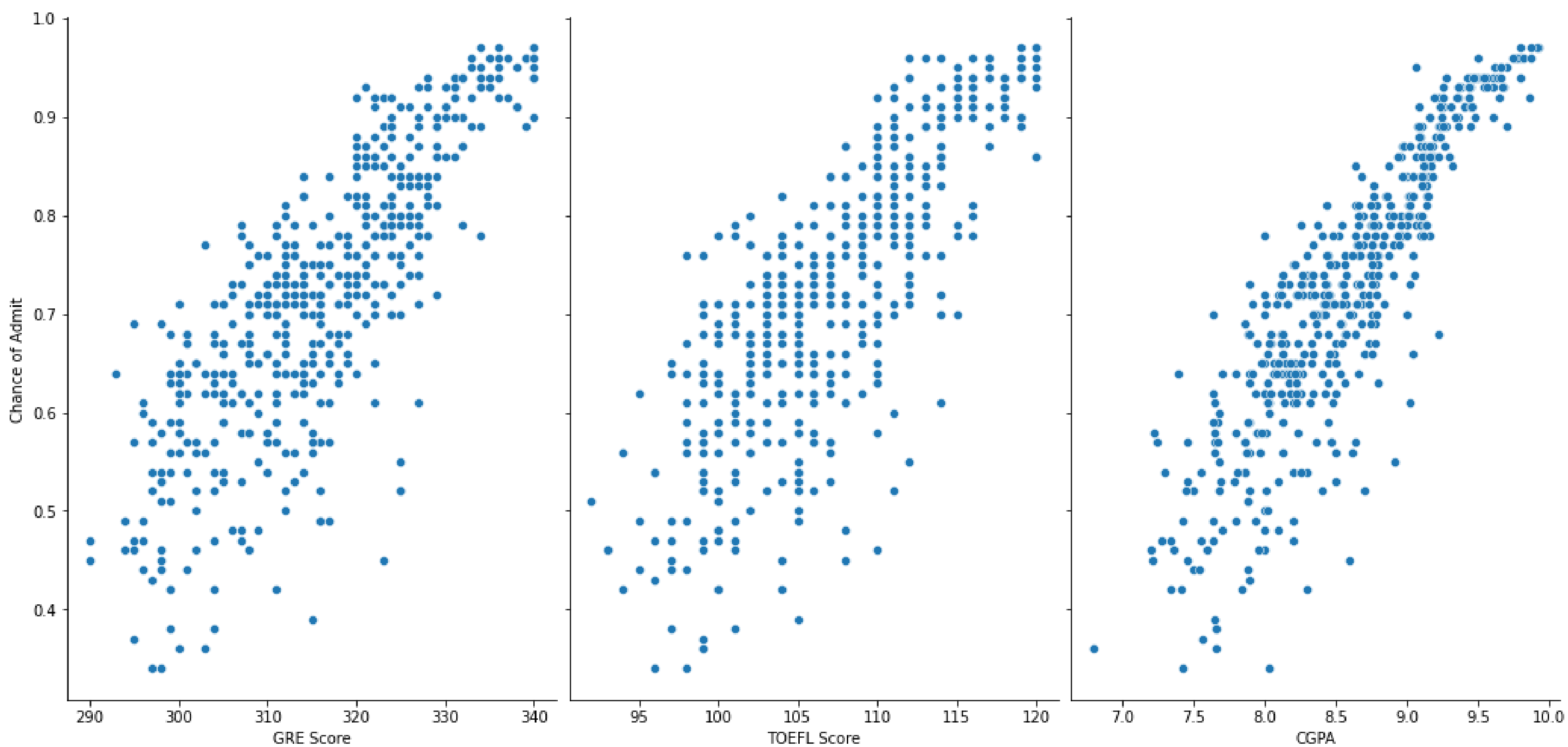
In [15]:
```
#Bivariate Analysis
sns.pairplot(df,diag_kind='kde')
```

Out[15]: <seaborn.axisgrid.PairGrid at 0x250c898beb0>



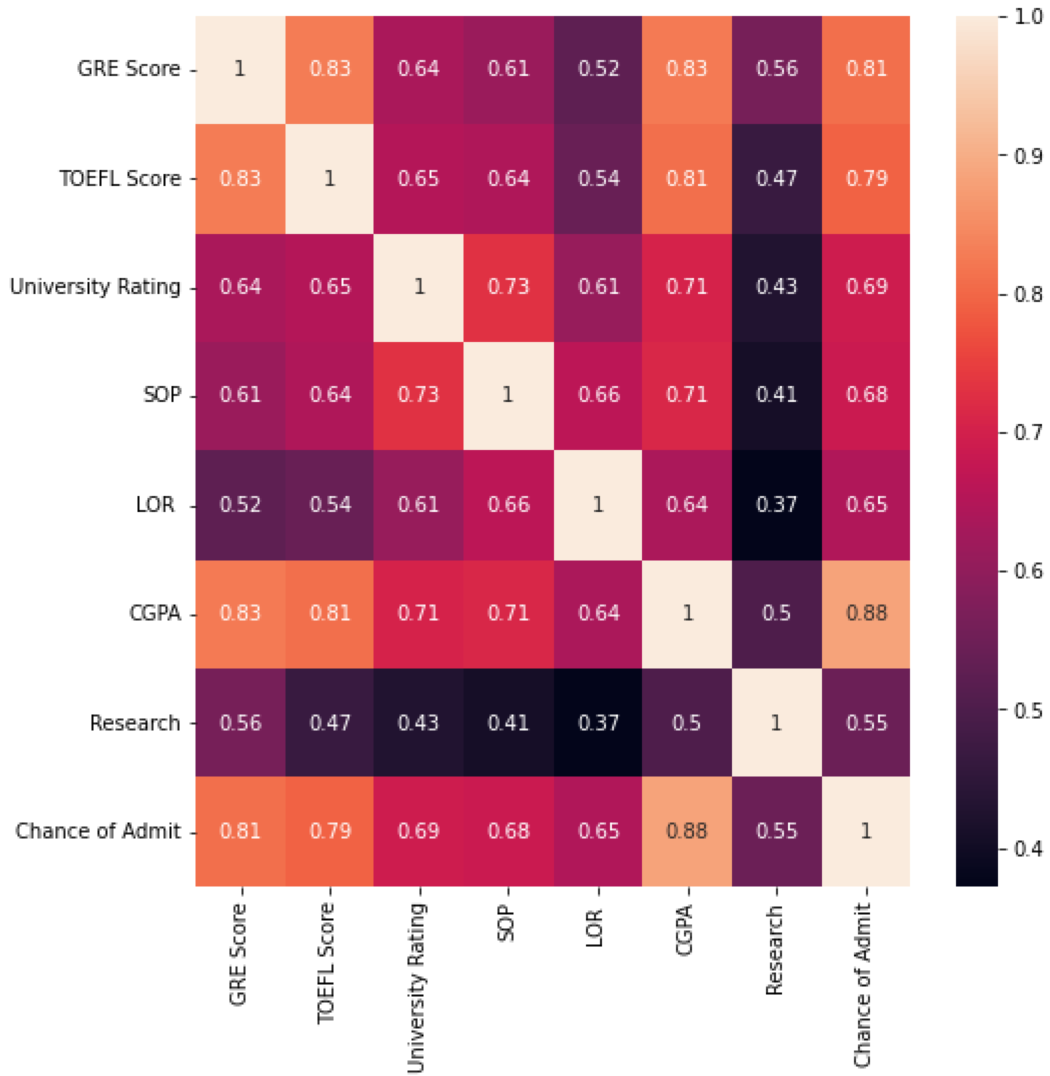


In [16]:
```
# Visualise the relationship between the features and the response using scatterplots
sns.pairplot(df, x_vars=["GRE Score","TOEFL Score","CGPA"], y_vars='Chance of Admit',size=7, aspect=0.7, kind='s
plt.show()
```



In [17]:

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



The bivariate analysis of continuous numeric features shows almost a linear relatonships between them.

The scatter plot between the Chance of Admit(target var) shows a linear relationship with the other continous numeric fetaures(GRE,TOEFLand CGPA scores.The assumption of Linear Regression is true here.Rest of the assumptions to be checked after doing Regression Analysis

The heatmap shows the most of the independent numeric variables are positively correlated.The assumption that there should not be any correlation within independent features (multicollinearity) is been violated here

The target variable (Chance of Admit)is also positively correlated with the independent features with the highest with CGPA ,TOEFL and GRE score.

In [18]:
```
#Linear Regression
#Independent features in x
x=df.iloc[:,0:7]

#target var in y
y=df.iloc[:,7:8]
```

In [19]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.7 , random_state=1)
```

In [20]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
lr
```

Out[20]:
```
▾ LinearRegression

LinearRegression()
```

In [21]:
```
# print the intercept
print(lr.intercept_)
```

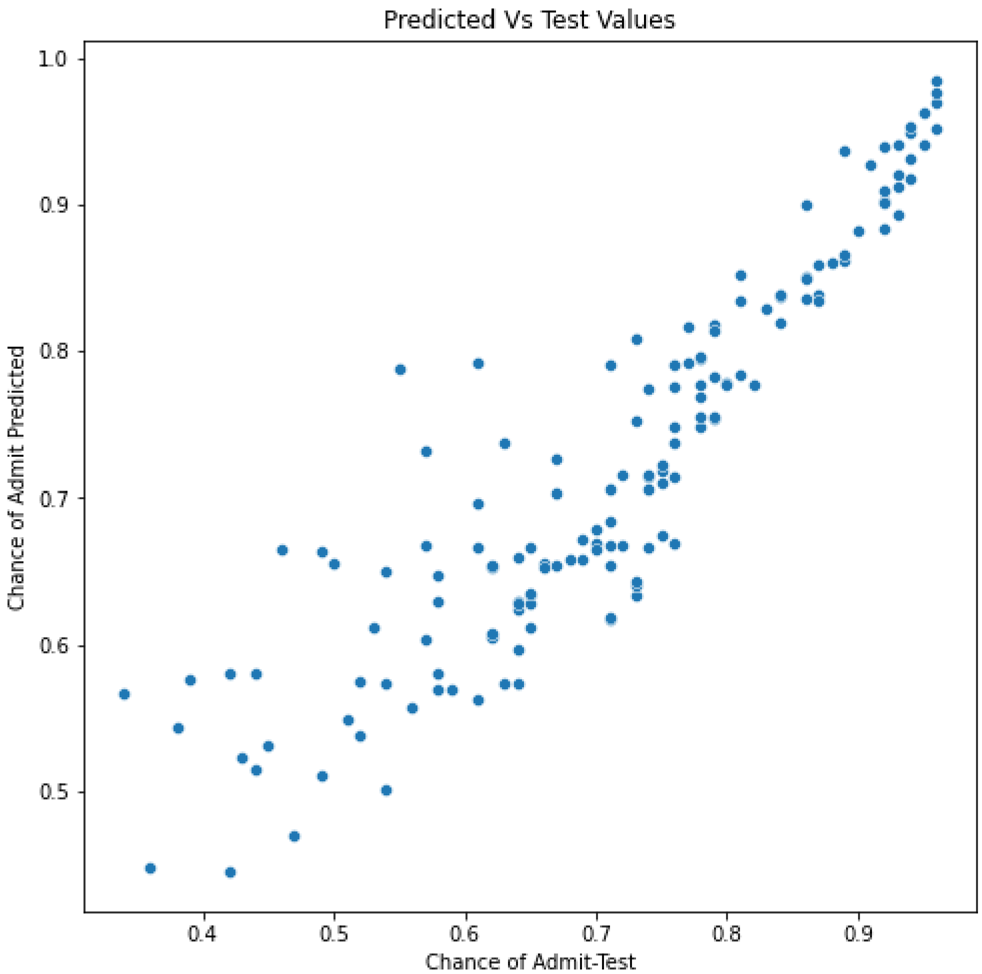
```
[-1.21611312]
```

In [22]:
```
for idx,col in enumerate (x_train.columns):
    print("The coefficient for column ",col,"is",lr.coef_[0][idx])
```

```
The coefficient for column  GRE Score is 0.001653419072663 65
The coefficient for column  TOEFL Score is 0.0038145301713297178
The coefficient for column  University Rating is 0.010123492386499228
The coefficient for column  SOP is -0.0010095236789056 91
The coefficient for column  LOR  is 0.013517323023096998
The coefficient for column  CGPA is 0.10703418872845212
The coefficient for column  Research is 0.028139654722785963
```

In [23]:
```
# Making predictions using the model
y_pred = lr.predict(x_test)
```

In [88]:
```
y_pred1=y_pred.reshape(len(y_pred,))
y_test1=y_test.values.reshape(len(y_test),)
plt.title("Predicted Vs Test Values")
plt.xlabel("Chance of Admit-Test")
plt.ylabel("Chance of Admit Predicted")
sns.scatterplot(y_test1,y_pred1)
plt.show()
```



In [104]:
```
scipy.stats.pearsonr(y_test1,y_pred1)
```

Out[104]:
```
(0.907497162180639, 1.280604344600469e-57)
```

The scatter plot between the predicted and test data shows a strong Positive correlation with a Pearson Correlation Coefficient of 0.91 .Since P-val is very much less compared to alpha(.05) it can be concluded that there is a statistically significant correlation between the two variables

In [60]:
```
#Error term calculations
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(y_test, y_pred)
rmse=math.sqrt(mse)
r_squared = r2_score(y_test, y_pred)
```

In [61]:
```
print('Mean_Squared_Error :' ,mse)
print("Root Mean Square Error :",rmse)
print('r_square_value :',r_squared)
```

```
Mean_Squared_Error : 0.0041259342367078004
Root Mean Square Error : 0.06423343550447695
r_square_value : 0.8157672116057979
```

In [26]:
```
lr.score(x_train, y_train)
```

Out[26]:
```
0.8209843725364347
```

In [27]:
```
lr.score(x_test, y_test)
```

Out[27]:
```
0.8157672116057979
```

Since the model is performing well on test data,its said it can be a fairly good model.But need to check the rest of the assumptions of Linear Regression

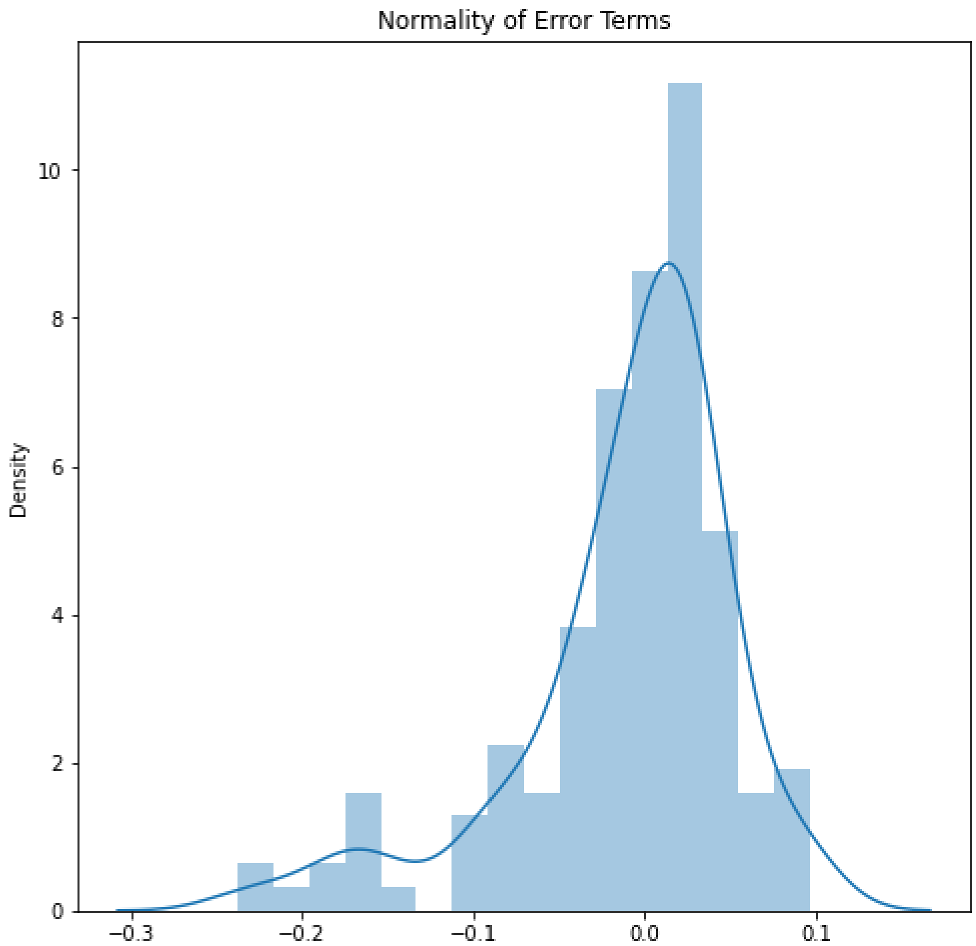
Check Assumptions of Regression

In [28]:
```
#Mean of Residuals must be zero
residual=y_test.values-y_pred
mean_residual=np.mean(residual)
print("Mean of Residual Errro ",mean_residual)
```

```
Mean of Residual Errro  -0.010793738256654774
```

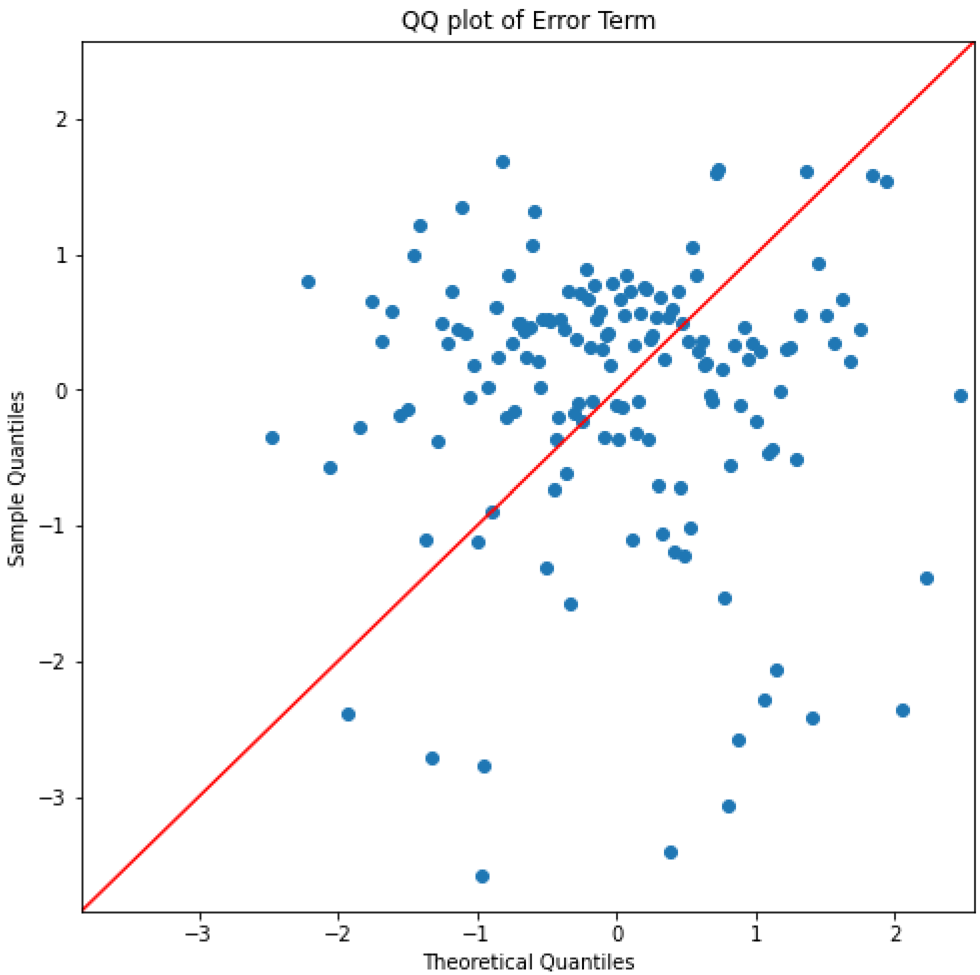
The mean of Residual error is close to zero(.01).The Assumption that mean of Residual should be zero is met

In [29]:
```
#Checking Normality of Residuals
sns.distplot(residual)
plt.title("Normality of Error Terms")
plt.show()
```



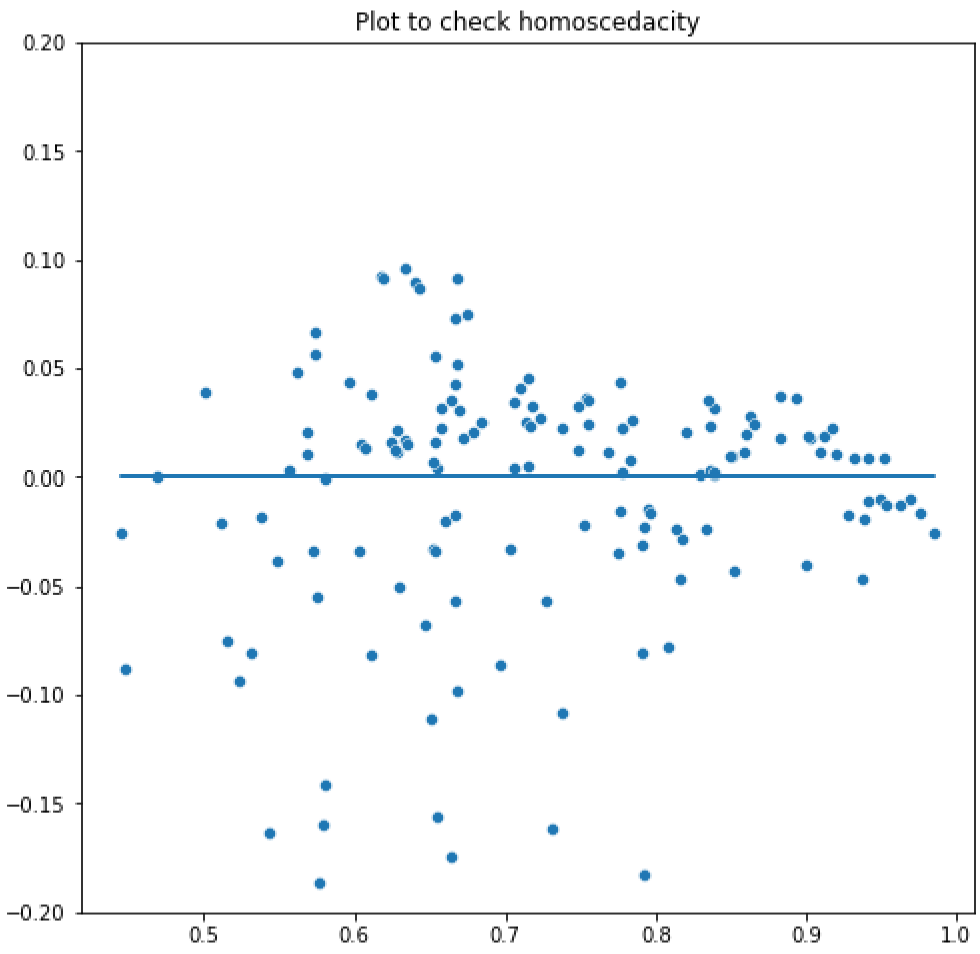
In [30]:
```
fig=sm.qqplot(residual,line='45',fit=True)
plt.title("QQ plot of Error Term")
```

Out[30]: `Text(0.5, 1.0, 'QQ plot of Error Term')`



QQplot suggests the distribution of Error term is not Gaussian.The Assumption that error term must be normally distributed is violated here

In [31]:
```
#Check for Homoscedacity
y_pred1=y_pred.reshape(len(y_pred,))
residual1=residual.reshape(len(residual,))
sns.scatterplot(y_pred1,residual1)
plt.plot(y_pred1,[0]*len(y_pred1))
plt.ylim(-.20,.20)
plt.title("Plot to check homoscedacity")
plt.show()
```



The plot shows the error terms are not having a constant error,meaning a constant deviation of the points from the zero-line.The Assumption that error

points must be Homoscedastic(constant variance) is violated here

In [32]:

```
#Multicollinearity check by VIF Score
dict={"Independent_variable":x.columns.to_list()}
vif_data=pd.DataFrame(dict)

vif_data
```

Out[32]:

	Independent_variable
0	GRE Score
1	TOEFL Score
2	University Rating
3	SOP
4	LOR
5	CGPA
6	Research

In [33]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(x.values, i)
                          for i in range(len(x.columns))]

print(vif_data)
```

```
  Independent_variable          VIF
0            GRE Score  1308.061089
1          TOEFL Score  1215.951898
2    University Rating    20.933361
3                  SOP    35.265006
4                 LOR     30.911476
5                 CGPA   950.817985
6             Research     2.869493
```

Except Research all the other features are highy correlated.Lets remove each features with high correlation and compute the VIF again

In [48]:
```python
to_remove=["GRE Score","TOEFL Score","CGPA","SOP","LOR ","University Rating"]
for i in range(1,7):
    col=to_remove[0]
    print("Column removed:",to_remove[0])
    temp=col
    to_remove.remove(col)
    dict={"Independent_variable":to_remove}
    vif_data=pd.DataFrame(dict)
    z=df[to_remove]
    vif_data["VIF"] = [variance_inflation_factor(z.values, i)
                          for i in range(len(z.columns))]
    print(vif_data)
    print("*"*50)
    to_remove.append(temp)
```

```
Column removed: GRE Score
  Independent_variable         VIF
0          TOEFL Score  635.033433
1                 CGPA  725.262710
2                  SOP   33.491456
3                 LOR    30.494371
4    University Rating   19.254913
**************************************************
Column removed: TOEFL Score
  Independent_variable         VIF
0                 CGPA  864.847352
1                  SOP   34.716350
2                 LOR    30.791506
3    University Rating   20.179880
4            GRE Score  682.284974
**************************************************
Column removed: CGPA
  Independent_variable          VIF
0                  SOP    33.557483
1                 LOR     29.335641
2    University Rating    19.596446
3            GRE Score  1002.544753
4          TOEFL Score  1112.701474
**************************************************
Column removed: SOP
  Independent_variable          VIF
```

```
0                  LOR     27.899042
1    University Rating    16.650986
2            GRE Score  1246.253994
3          TOEFL Score  1202.367893
4                 CGPA   903.344683
**************************************************
Column removed: LOR
  Independent_variable          VIF
0    University Rating    19.862737
1            GRE Score  1292.612862
2          TOEFL Score  1214.816029
3                 CGPA   899.572068
4                  SOP    31.780853
**************************************************
Column removed: University Rating
  Independent_variable          VIF
0            GRE Score  1233.273872
1          TOEFL Score  1203.004675
2                 CGPA   908.002108
3                  SOP    28.660606
4                  LOR    30.012920
**************************************************
```

The above output shows that even after removing each columns with high VIF and recomputing the same ,VIF still remains high

The following assumptions are violated in the model built:

1) No Multicollinearity

2)Error terms to be normally distributed

3)Homoscedacity of error terms

Also from the coefficients got from this model,its hard to imploy which of the predictor variables are influential is predicting the chance of admit of the student

Hence it can be said that the Linear model built is not an apt one

Lets standardize the data to Std. Normal distribution using sklearn preprocessing.

In [63]:
```
from sklearn.preprocessing import StandardScaler
s_scaler = StandardScaler()
```

In [68]:
```
df_scaled=s_scaler.fit_transform(df)
df_scaled=pd.DataFrame(df_scaled,columns=df.columns)
df_scaled.head()
```

Out[68]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1.819238	1.778865	0.775582	1.137360	1.098944	1.776806	0.886405	1.406107
1	0.667148	-0.031601	0.775582	0.632315	1.098944	0.485859	0.886405	0.271349
2	-0.041830	-0.525364	-0.099793	-0.377773	0.017306	-0.954043	0.886405	-0.012340
3	0.489904	0.462163	-0.099793	0.127271	-1.064332	0.154847	0.886405	0.555039
4	-0.219074	-0.689952	-0.975168	-1.387862	-0.523513	-0.606480	-1.128152	-0.508797

In [70]:
```
#Perform Linear Regresion on scaled data
x_scaled=df_scaled.iloc[:,0:7]
y_scaled=df_scaled.iloc[:,7:8]
```

In [71]:
```
x_strain, x_stest, y_strain, y_stest = train_test_split(x_scaled, y_scaled, train_size=0.7 , random_state=1)
```

In [73]:
```
lr_scaled=LinearRegression()
lr_scaled.fit(x_strain,y_strain)
```

Out[73]:
```
▾ LinearRegression

LinearRegression()
```

In [74]:
```
for idx,col in enumerate (x_strain.columns):
    print("The coefficient for column ",col,"is",lr_scaled.coef_[0][idx])
```

```
The coefficient for column  GRE Score is 0.13231940128209482
The coefficient for column  TOEFL Score is 0.1643715551048881
The coefficient for column  University Rating is 0.08201997928913549
The coefficient for column  SOP is -0.007088272337575197
The coefficient for column  LOR  is 0.08863231563634404
The coefficient for column  CGPA is 0.45866134433795697
The coefficient for column  Research is 0.09906551219898176
```

In [75]:
```
print("Intercept of scaled model",lr_scaled.intercept_)
```

```
Intercept of scaled model [0.02296553]
```

In [97]:
```
y_spredict=lr_scaled.predict(x_stest)
residual_scaled=y_stest.values-y_spredict
```

After standardizing data ,it can be said that CGPA has more influence on Chance of Admit than Other variables ,followed by GRE and TOEFL scores

Model evaluation by OLS model

In [76]:
```
x_strain_sm = x_strain

x_strain_sm = sm.add_constant(x_strain_sm)

lr_ols = sm.OLS(y_strain,x_strain_sm).fit()
```

In [77]:
```
lr_ols.summary()
```

Out[77]:

OLS Regression Results

Dep. Variable:	Chance of Admit	**R-squared:**	0.821
Model:	OLS	**Adj. R-squared:**	0.817
Method:	Least Squares	**F-statistic:**	224.1
Date:	Tue, 04 Oct 2022	**Prob (F-statistic):**	1.27e-123
Time:	12:13:41	**Log-Likelihood:**	-185.45
No. Observations:	350	**AIC:**	386.9
Df Residuals:	342	**BIC:**	417.8
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0230	0.022	1.031	0.303	-0.021	0.067
GRE Score	0.1323	0.047	2.798	0.005	0.039	0.225
TOEFL Score	0.1644	0.045	3.653	0.000	0.076	0.253
University Rating	0.0820	0.039	2.116	0.035	0.006	0.158
SOP	-0.0071	0.037	-0.189	0.850	-0.081	0.067
LOR	0.0886	0.031	2.849	0.005	0.027	0.150
CGPA	0.4587	0.049	9.356	0.000	0.362	0.555
Research	0.0991	0.027	3.613	0.000	0.045	0.153

Omnibus:	77.752	**Durbin-Watson:**	1.981
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	179.766
Skew:	-1.100	**Prob(JB):**	9.21e-40
Kurtosis:	5.736	**Cond. No.**	5.69

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

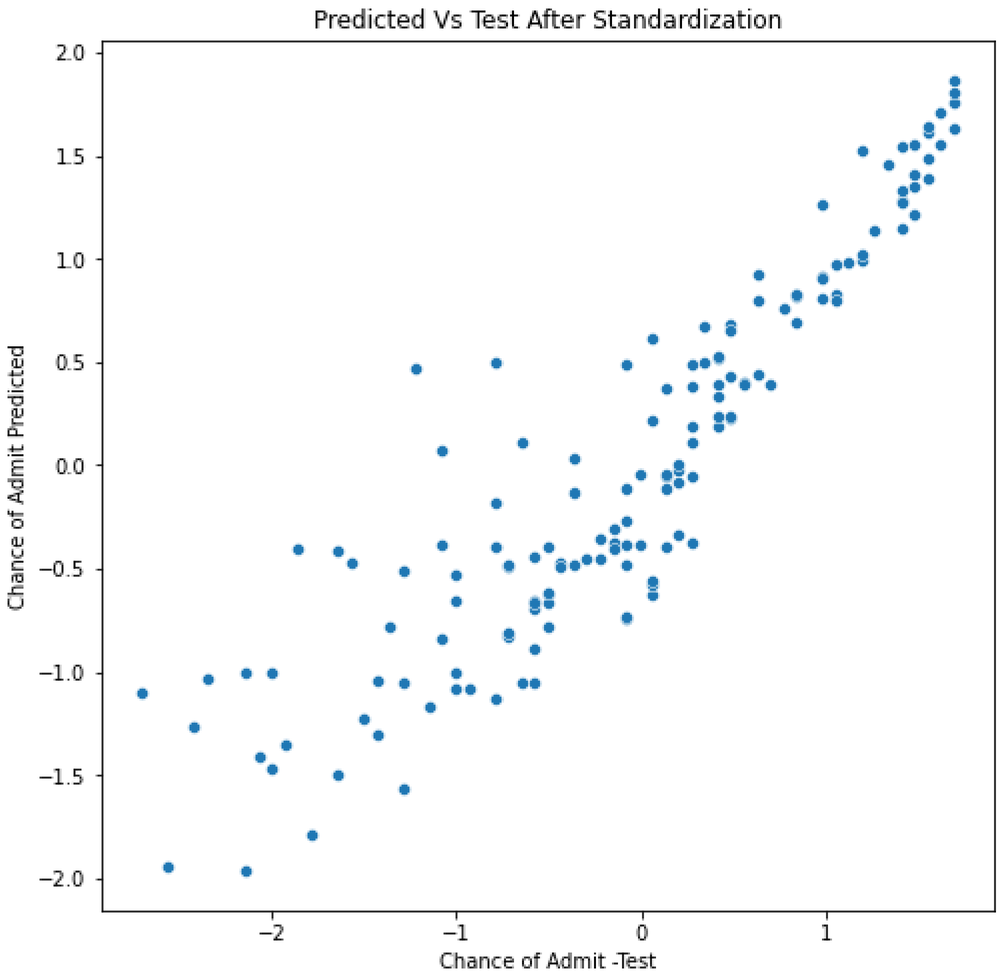
After standardizing the coefficients are similar comparing the OLS and the LR models

The R-squared value after standardizing has become 0.821.Prior to which it was 0.815.Not much increase is observed

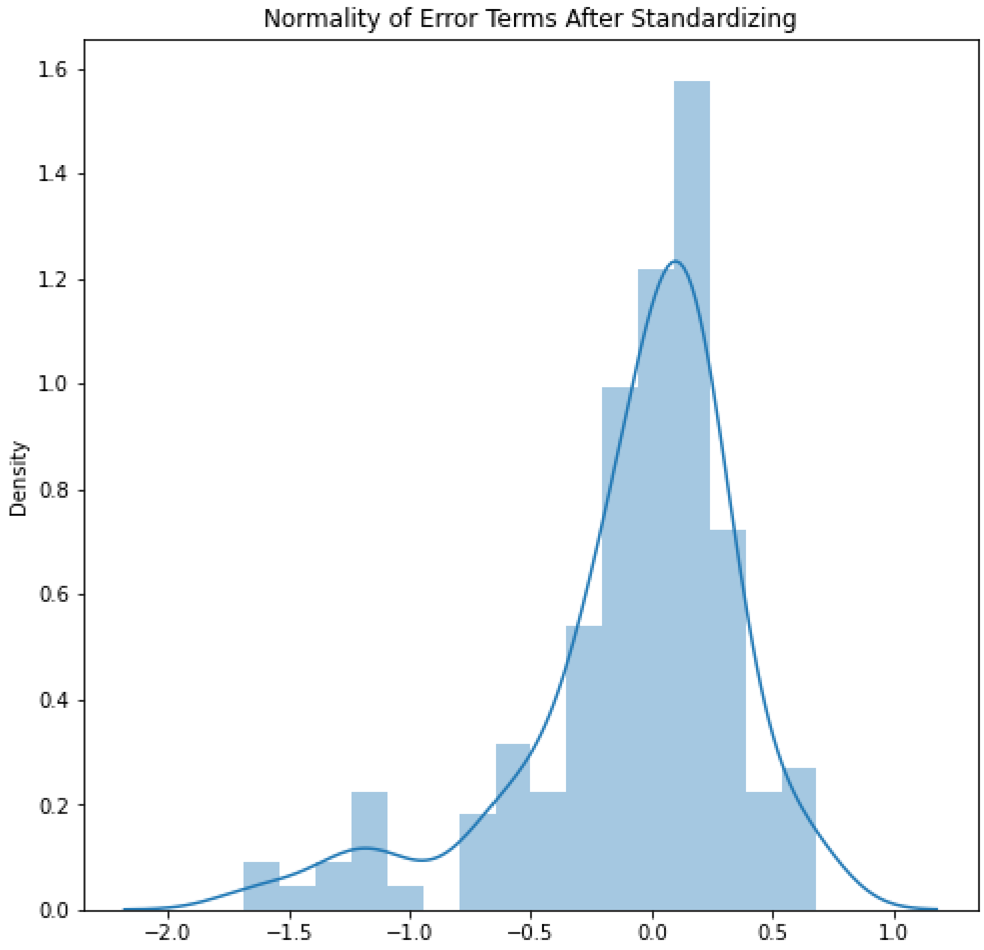
Adjusted R^2 is 0.817

Model Evaluation using charts

In [91]:
```
x1=y_stest.values.reshape(len(y_stest),)
y1=y_spredict.reshape(len(y_spredict),)
plt.xlabel("Chance of Admit -Test")
plt.ylabel("Chance of Admit Predicted")
plt.title("Predicted Vs Test After Standardization")
sns.scatterplot(x1,y1)
plt.show()
```

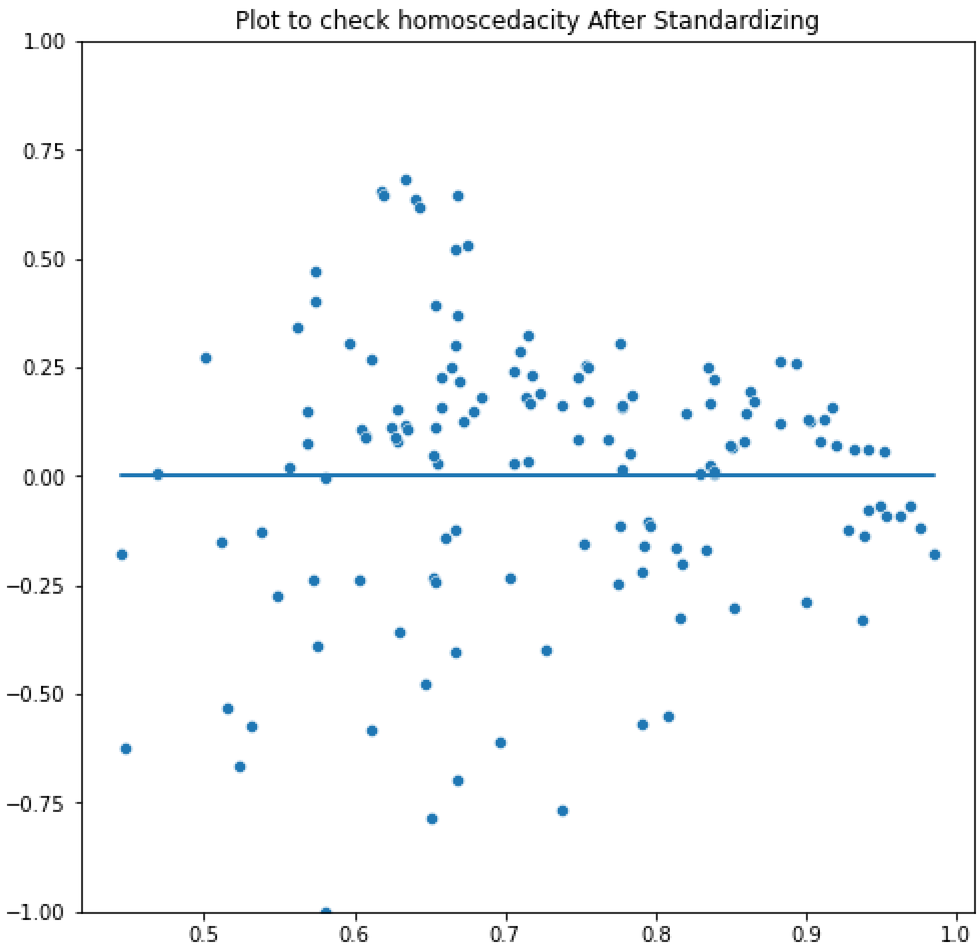


In [98]:
```
#Checking Normality of Residuals
sns.distplot(residual_scaled)
plt.title("Normality of Error Terms After Standardizing")
plt.show()
```



In [102]:

```
#Check for Homoscedacity
y_spred1=y_pred.reshape(len(y_spredict,))
residual1_scaled=residual_scaled.reshape(len(residual_scaled,))
sns.scatterplot(y_spred1,residual1_scaled)
plt.plot(y_spred1,[0]*len(y_spred1))
plt.ylim(-1,1)
plt.title("Plot to check homoscedacity After Standardizing")
plt.show()
```



Even after Standardizing the assumptions of Linear Regression like Homoscedacity of error terms,normality of error terms are violated

Lets Try out Ridge and Lasso Regression

In [50]:
```
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
```

In [51]:
```
#Ridge
params = {'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1,
 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0,
 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50, 100, 500, 1000 ]}


ridge = Ridge()

# cross validation
folds = 5
model_cv = GridSearchCV(estimator = ridge,
                        param_grid = params,
                        scoring= 'neg_mean_absolute_error',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)
model_cv.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
```

Out[51]:

```
▸ GridSearchCV
▸ estimator: Ridge
   ▸ Ridge
```

In [52]:
```
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results = cv_results[cv_results['param_alpha']<=200]
cv_results.head()
```

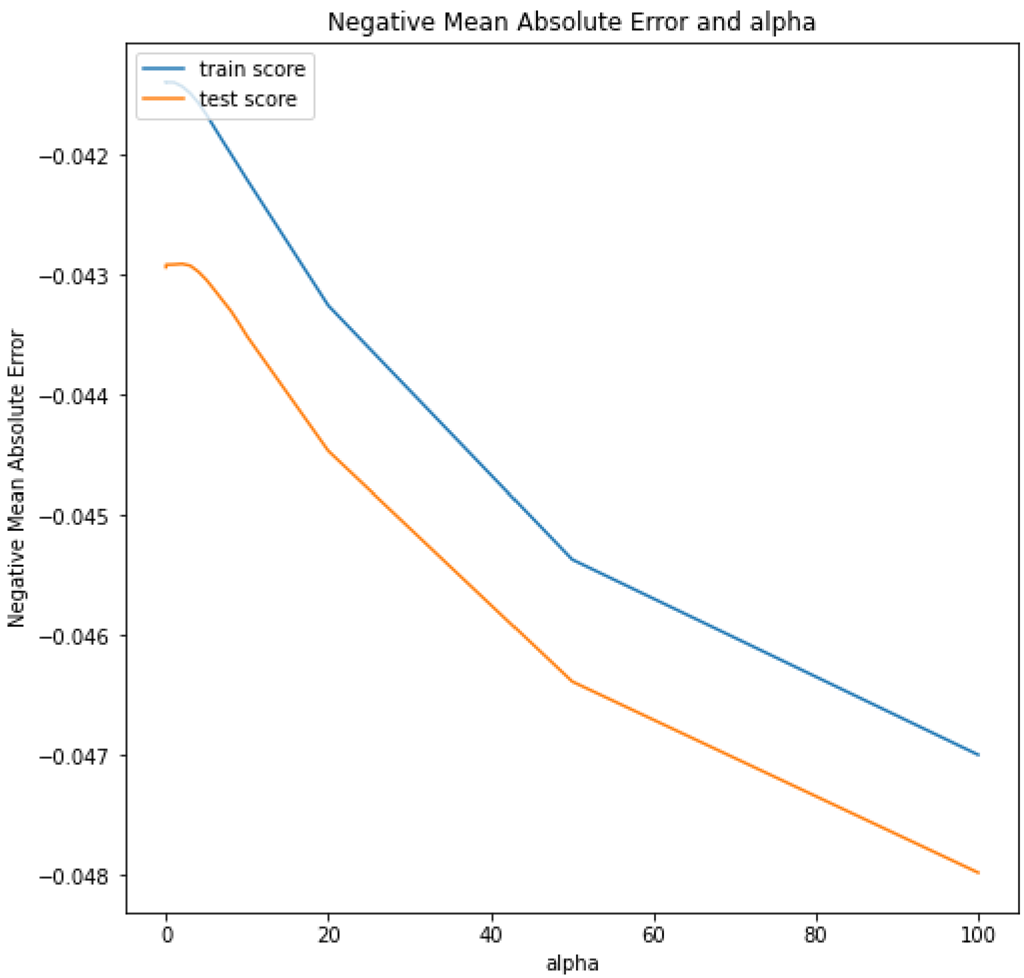
Out[52]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_score
	0.005470	0.003296	0.001565	0.001983	0.0001	{'alpha': 0.0001}	-0.054859	-0.029594	-0.043994
	0.001730	0.002128	0.003626	0.003636	0.001	{'alpha': 0.001}	-0.054860	-0.029593	-0.043994
	0.000203	0.000407	0.003385	0.006769	0.01	{'alpha': 0.01}	-0.054862	-0.029590	-0.043992
	0.000203	0.000407	0.003363	0.006727	0.05	{'alpha': 0.05}	-0.054874	-0.029577	-0.043987
	0.000000	0.000000	0.000000	0.000000	0.1	{'alpha': 0.1}	-0.054889	-0.029560	-0.043979

ows × 21 columns

In [53]:

```
# plotting mean test and train scoes with alpha
cv_results['param_alpha'] = cv_results['param_alpha'].astype('int32')

# plotting
plt.plot(cv_results['param_alpha'], cv_results['mean_train_score'])
plt.plot(cv_results['param_alpha'], cv_results['mean_test_score'])
plt.xlabel('alpha')
plt.ylabel('Negative Mean Absolute Error')
plt.title("Negative Mean Absolute Error and alpha")
plt.legend(['train score', 'test score'], loc='upper left')
plt.show()
```



In [54]:
```
#Lasso
lasso = Lasso()

# cross validation
model_cv = GridSearchCV(estimator = lasso,
                        param_grid = params,
                        scoring= 'neg_mean_absolute_error',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)

model_cv.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
```

Out[54]:

```
▸ GridSearchCV
▸ estimator: Lasso
    ▸ Lasso
```

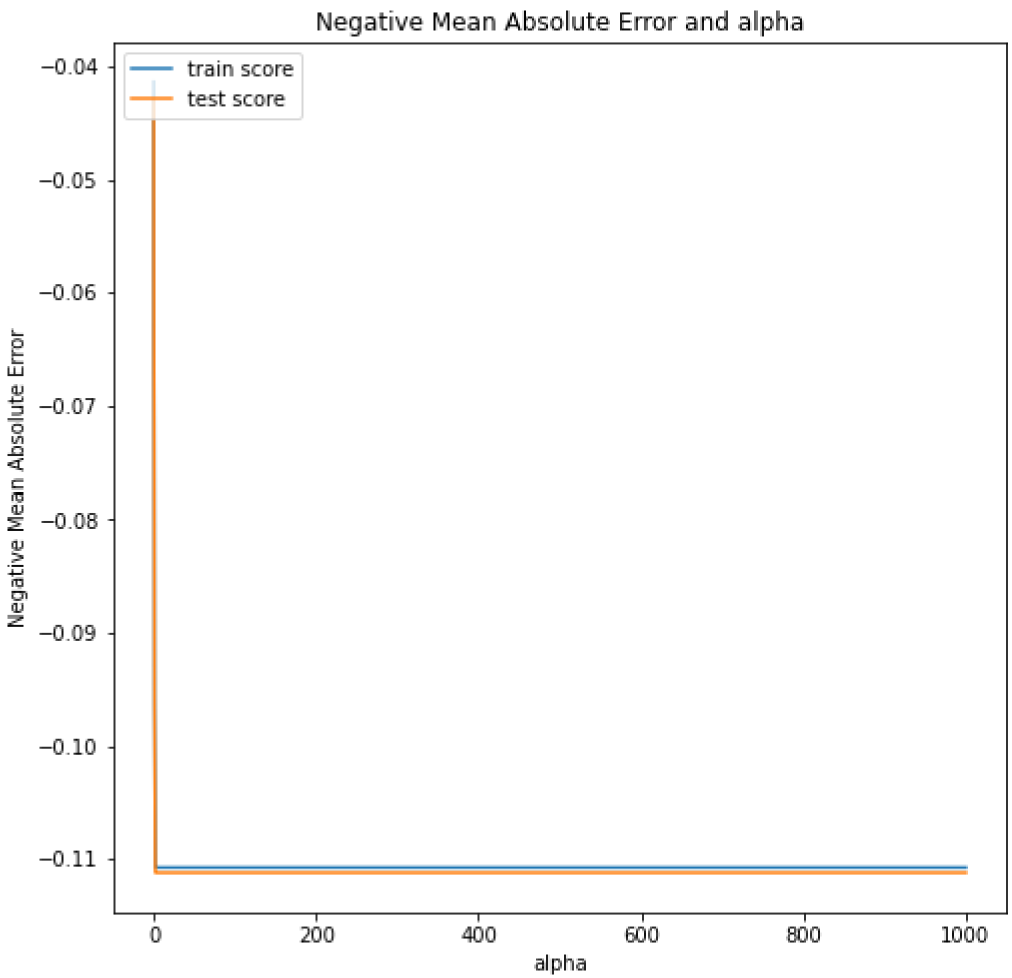
In [55]:
```
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results.head()
```

Out[55]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_sc
0	0.002758	0.003383	0.004824	0.004323	0.0001	{'alpha': 0.0001}	-0.054933	-0.029532	-0.0439
1	0.001805	0.003134	0.000000	0.000000	0.001	{'alpha': 0.001}	-0.055603	-0.029212	-0.0437
2	0.001634	0.003268	0.000203	0.000406	0.01	{'alpha': 0.01}	-0.065026	-0.037292	-0.0471
3	0.006250	0.007655	0.000218	0.000435	0.05	{'alpha': 0.05}	-0.069097	-0.047426	-0.0529
4	0.003405	0.006316	0.000403	0.000806	0.1	{'alpha': 0.1}	-0.069858	-0.050023	-0.0558

5 rows × 21 columns

In [56]:
```
# plotting mean test and train scoes with alpha
cv_results['param_alpha'] = cv_results['param_alpha'].astype('float32')

# plotting
plt.plot(cv_results['param_alpha'], cv_results['mean_train_score'])
plt.plot(cv_results['param_alpha'], cv_results['mean_test_score'])
plt.xlabel('alpha')
plt.ylabel('Negative Mean Absolute Error')

plt.title("Negative Mean Absolute Error and alpha")
plt.legend(['train score', 'test score'], loc='upper left')
plt.show()
```



Recommendation and Insights

The heat map showed a very high positive correlation between the Chance of Admit(target Variable) with CGPA ,TOEFL and GRE score

On building the Linear Regression model its seen that apart from CGPA(with coefficient 0.45) all other features were having negligible effect on the response variable.Their coefficients were lying in range 0.1 to -0.01.Most of the assumption of Linear Regression is also violated

The data was standardized ,Linear Regression performed again ,still the coefficients were almost the same as before .The assumptions too were violated .OLS model was built again but not much of difference in coefficients was found

In both the models built the independent variable SOP had a negative coefficient which suggests that as the independent variable increases, the dependent variable tends to decrease

The R-squared value after standardizing has become 0.821.Prior to which it was 0.815.Not much increase is observed

So it can be inferred that more features to be added in dataset to build a more robust model