In [1]:

```
Objective: To perform EDA and to fit a linear regression model on Jamboree Education case s

Insights:

# 32 % values have a university rating 3
# 25 % values have a university rating 2
# 21 % values have a university rating 4
# 56.11 % of students have prior research experience# People with research experience tend
# People with research experience have a greater SOP and LOR score
# Higher the SOP and LOR greater is the university rank students apply to
# Most students score between 310-320 and 320-330.
# Least number of students score between 280-290
# Most students score between 95-105 and 105-125.
# Least number of students score between 85-95
# Most students score between 7-8 and 8-9.
# Least number of students score between 6-7
# Chance of admission and GRE score has spearman corr of 0.82
# Chance of admission and CGPA has spearman corr of 0.89
# Chance of admission and research least spearman corr of 0.56

Linear regression model Insights:

1. Degree of 1 has the bests R2 scores (Test and train)
2. All 7 features in our model are important
3. Simple linear regression and Elasticnet are the best models
4. All 5 assumptions of linear regression are shown in cell 47

Recommendations:

1. CGPA,GRE SCore,LOR,Research are the most important features to predict 'Chance of Admit'
2. Training data has only 400 data points. We need to increase the number of training data
3. In real world , there may be a lot of outliners. We need to build a separate models for
4. With this model we can predict well in advance whether a student will get into a particu
5. If the predicted 'Chance of Admit' value is less :
    1. We can figure out and find which feature that the student needs to improve in order
6. We can have a customized development plans for each individual and guide them to respect
7. Our model will save 3-6 months of students time in case if model predicts lower probabil
8. It is good to add the country to which the student is applying to as a feature
9. It is good to add the work experience of the student as a feature
10.It is good to add the branch in which the student in interested to apply for.
```

In [2]:

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge,Lasso,ElasticNet
import sklearn.metrics as metrics
from sklearn.preprocessing import PolynomialFeatures
from sklearn import decomposition
from  scipy import stats
from sklearn import decomposition
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
```

In [3]:

```
df=pd.read_csv('Jamboree_Admission.csv')
```

In [4]:

```
df.shape
# Shows the shape of data
```

Out[4]:

```
(500, 9)
```

In [5]:

```
df.drop('Serial No.',axis=1,inplace=True)
# Removing the column as it is not needed
```

In [6]:

```
df.drop_duplicates(keep='first', inplace=True, ignore_index=True)
# Removing the duplicates as it is not needed
```

In [7]:

```
df.columns=['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR', 'CGPA',
       'Research', 'Chance of Admit']
# Renaming columns as there are some spaces' ' in its names
```

In [8]:

```
df
```

Out[8]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65
...	...	...	...	...	...	...	...	...
495	332	108	5	4.5	4.0	9.02	1	0.87
496	337	117	5	5.0	5.0	9.87	1	0.96
497	330	120	5	4.5	5.0	9.56	1	0.93
498	312	103	4	4.0	5.0	8.43	0	0.73
499	327	113	4	4.5	4.5	9.04	0	0.84

500 rows × 8 columns

In [9]:

```
df.info()
# All the data types are int or float.
# No need for encoding
```

```
<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
```

In [10]:

```
df.isnull().sum()
# There are no null values
```

Out[10]:

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

In [11]:

```
df.describe()
# Summarize all the features
```

Out[11]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	C of
count	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.
mean	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	0.
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	0
min	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.
25%	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.
50%	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.
75%	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.
max	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.

In [12]:

```
def outliners(x,col):
    Q1 = np.percentile(x[col], 25)
    Q3 = np.percentile(x[col], 75)
    IQR = Q3 - Q1
    upper = Q3 +1.5*IQR
    lower = Q1 - 1.5*IQR
    #print(upper,lower)
    ls=list(x.iloc[((x[col]<lower) | (x[col]>upper)).values].index)
    return ls
```

In [13]:

```
for i in df.columns[:]:
    print('Number of outliners in colums',i, 'is', len(outliners(df,i)))
    df.drop(outliners(df,i),axis=0,inplace=True)
    df.reset_index()
df.reset_index()

#Gives all the outliners in each column
# Reset all the index
```

```
Number of outliners in colums GRE Score is 0
Number of outliners in colums TOEFL Score is 0
Number of outliners in colums University Rating is 0
Number of outliners in colums SOP is 0
Number of outliners in colums LOR is 1
Number of outliners in colums CGPA is 0
Number of outliners in colums Research is 0
Number of outliners in colums Chance of Admit is 2
```

Out[13]:

	index	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	0	337	118	4	4.5	4.5	9.65	1	0.92
1	1	324	107	4	4.0	4.5	8.87	1	0.76
2	2	316	104	3	3.0	3.5	8.00	1	0.72
3	3	322	110	3	3.5	2.5	8.67	1	0.80
4	4	314	103	2	2.0	3.0	8.21	0	0.65
...	...	...	...	...	...	...	...	...	...
492	495	332	108	5	4.5	4.0	9.02	1	0.87
493	496	337	117	5	5.0	5.0	9.87	1	0.96
494	497	330	120	5	4.5	5.0	9.56	1	0.93
495	498	312	103	4	4.0	5.0	8.43	0	0.73
496	499	327	113	4	4.5	4.5	9.04	0	0.84

497 rows × 9 columns

In [14]:

```
df['University Rating'].value_counts(normalize=True)
# 32 % values have university rating 3
# 25 % values have university rating 2
# 21 % values have university rating 4
```

Out[14]:

```
3    0.325956
2    0.249497
4    0.211268
5    0.146881
1    0.066398
Name: University Rating, dtype: float64
```

In [15]:

```
df['SOP'].value_counts(normalize=True)
# Below table shows the percentage students who got specific SOP
```

Out[15]:

```
4.0    0.177062
3.5    0.177062
3.0    0.160966
4.5    0.126761
2.5    0.126761
2.0    0.086519
5.0    0.084507
1.5    0.050302
1.0    0.010060
Name: SOP, dtype: float64
```

In [16]:

```
df['LOR'].value_counts(normalize=True)
# Below table shows the percentage students who got specific LOR
```

Out[16]:

```
3.0    0.197183
4.0    0.189135
3.5    0.173038
4.5    0.126761
2.5    0.100604
5.0    0.100604
2.0    0.090543
1.5    0.022133
Name: LOR, dtype: float64
```

In [17]:

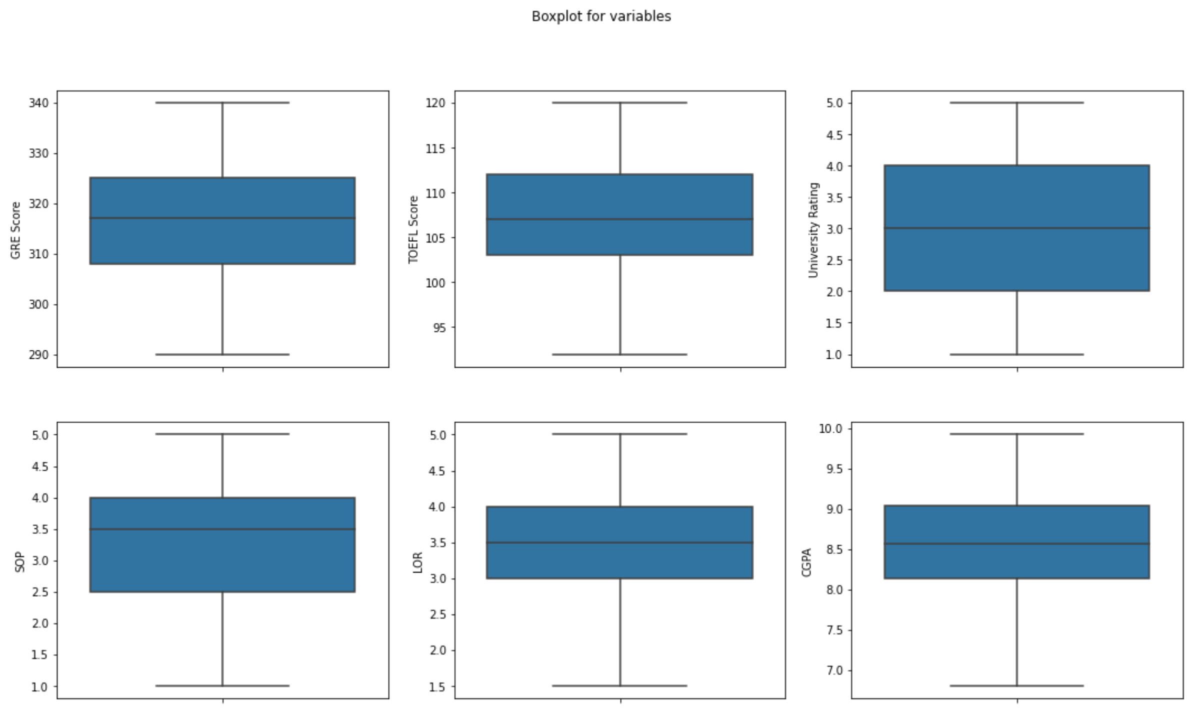
```
df['Research'].value_counts(normalize=True)
# 56.11 % of students have prior research experience
```

Out[17]:

```
1    0.56338
0    0.43662
Name: Research, dtype: float64
```

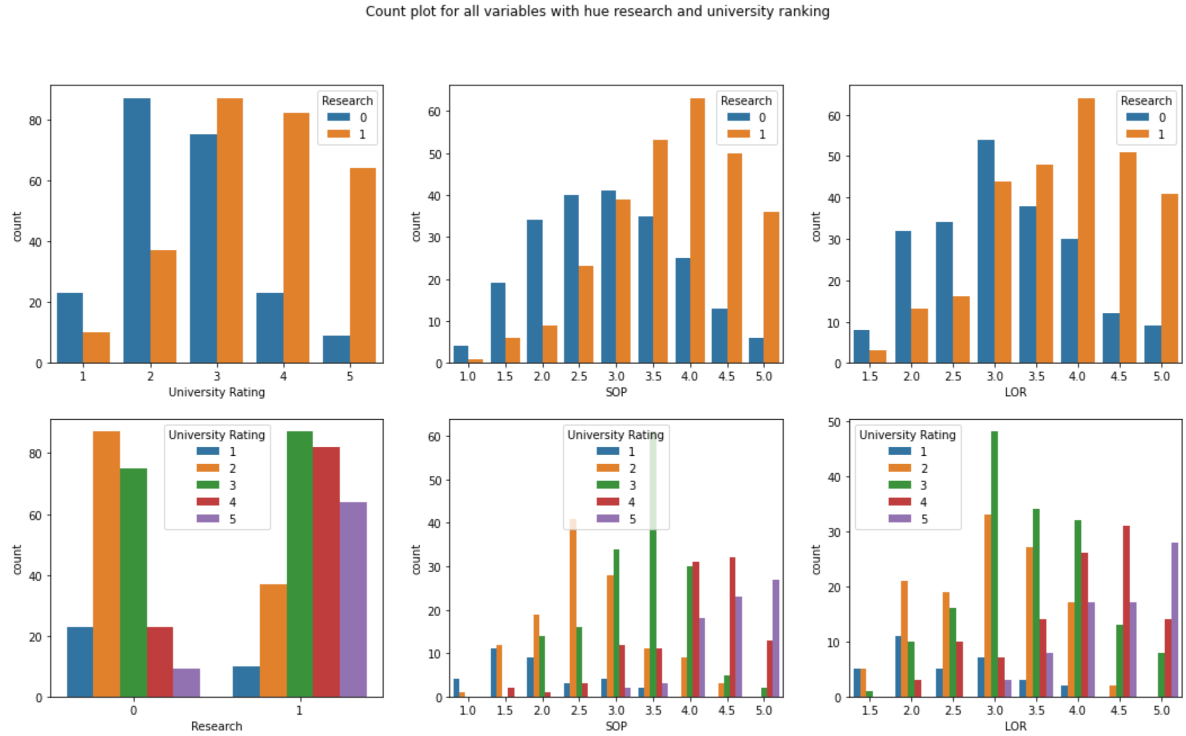
In [18]:

```
fig, axes = plt.subplots(2, 3, figsize=(18, 10))

fig.suptitle('Boxplot for variables')
sns.boxplot(ax=axes[0, 0], data=df, y='GRE Score')
sns.boxplot(ax=axes[0, 1], data=df, y='TOEFL Score')
sns.boxplot(ax=axes[0, 2], data=df, y='University Rating' )
sns.boxplot(ax=axes[1, 0], data=df, y='SOP' )
sns.boxplot(ax=axes[1, 1], data=df, y='LOR')
sns.boxplot(ax=axes[1, 2], data=df, y='CGPA')
plt.show()
# Below table shows the boxplot for all variables.
# There are no outliner as all the outliners were removed
```



In [19]:

```
fig, axes = plt.subplots(2, 3, figsize=(18, 10))

fig.suptitle('Count plot for all variables with hue research and university ranking')
sns.countplot(ax=axes[0, 0], data=df, x='University Rating',hue=df['Research'])
sns.countplot(ax=axes[0, 1], data=df, x='SOP',hue=df['Research'])
sns.countplot(ax=axes[0, 2], data=df, x='LOR',hue=df['Research'] )
sns.countplot(ax=axes[1, 0], data=df, x='Research',hue=df['University Rating'] )
sns.countplot(ax=axes[1, 1], data=df, x='SOP',hue=df['University Rating'])
sns.countplot(ax=axes[1, 2], data=df, x='LOR',hue=df['University Rating'])
plt.show()
# People with research experience tend to apply for higher university ranking colleges
# People with research experience have a greater SOP and LOR score
# Higher the SOP and LOR greater is the university rank students apply to
```

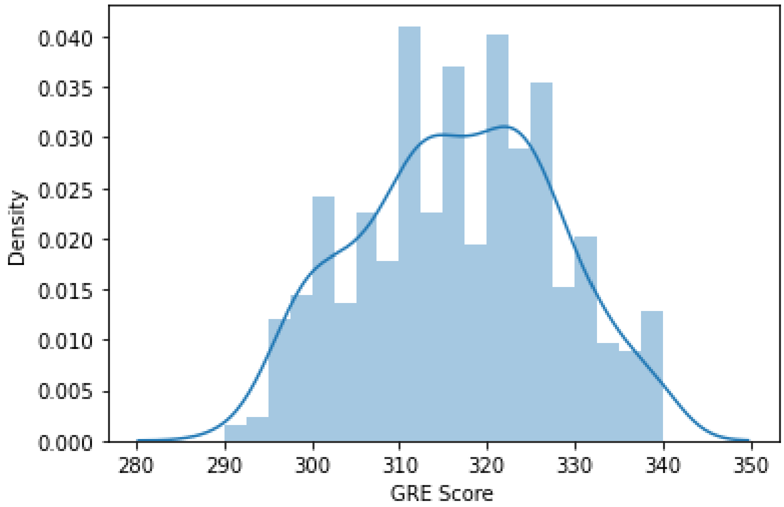


In [20]:

```
sns.distplot(df['GRE Score'],bins=20)
# Below is the distribution plot for GRE score. Looks like a normal distribution
```

Out[20]:

```
<AxesSubplot:xlabel='GRE Score', ylabel='Density'>
```

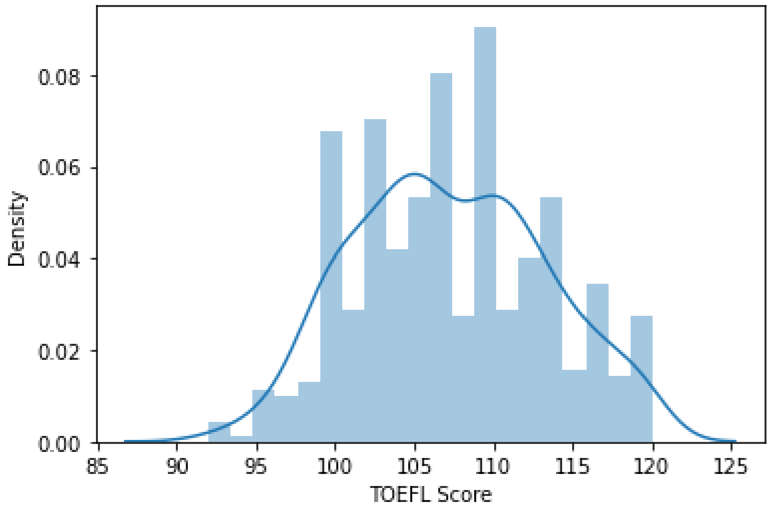


In [21]:

```
sns.distplot(df['TOEFL Score'],bins=20)
# Below is the distribution plot for TOEFL score. Looks like a normal distribution
```

Out[21]:

```
<AxesSubplot:xlabel='TOEFL Score', ylabel='Density'>
```

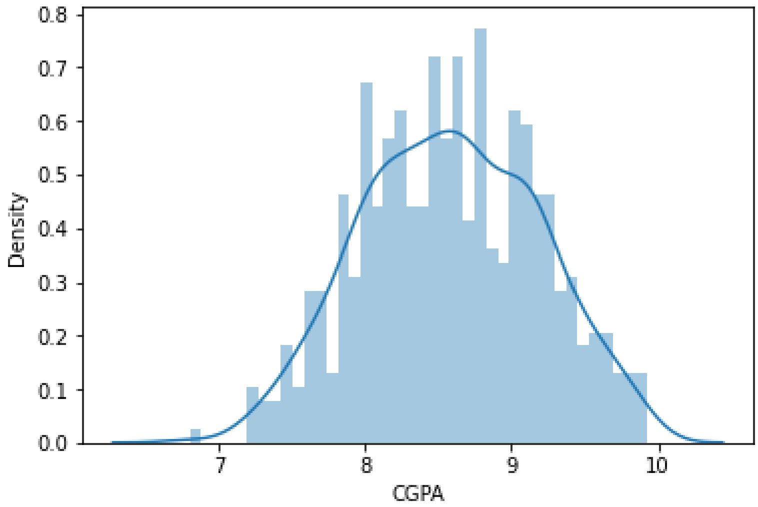


In [22]:

```
sns.distplot(df['CGPA'],bins=40)
# Below is the distribution plot for CGPA. Looks like a normal distribution
```

Out[22]:

<AxesSubplot:xlabel='CGPA', ylabel='Density'>

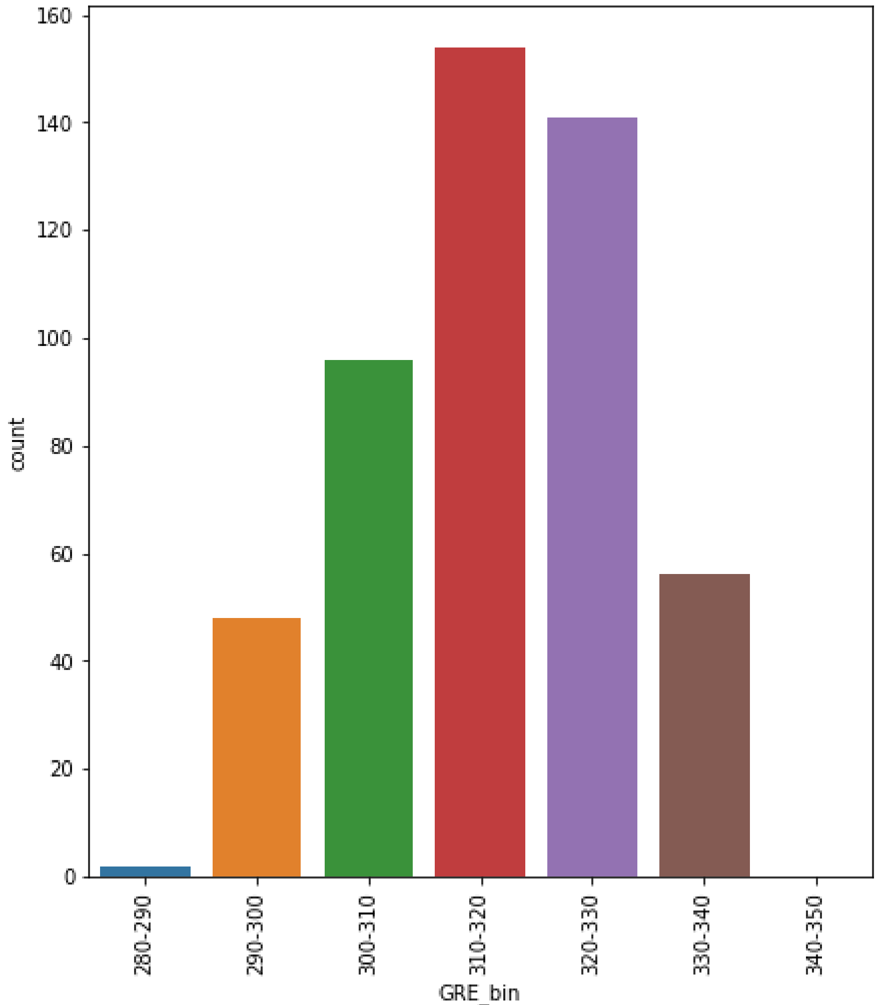


In [23]:

```
bins=[280,290,300,310,320,330,340,350]
labels=['280-290','290-300','300-310','310-320','320-330','330-340','340-350']
df['GRE_bin']=pd.cut(df['GRE Score'], bins=bins, labels=labels)
# Created bins for hospitalization charges
fig,ax=plt.subplots(figsize=(7,8))
plt.xticks(rotation=90)
sns.countplot(df['GRE_bin'])
# Bins were created for GRE score
# Most students score between 310-320 and 320-330.
# Least number of students score between 280-290
```

Out[23]:

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

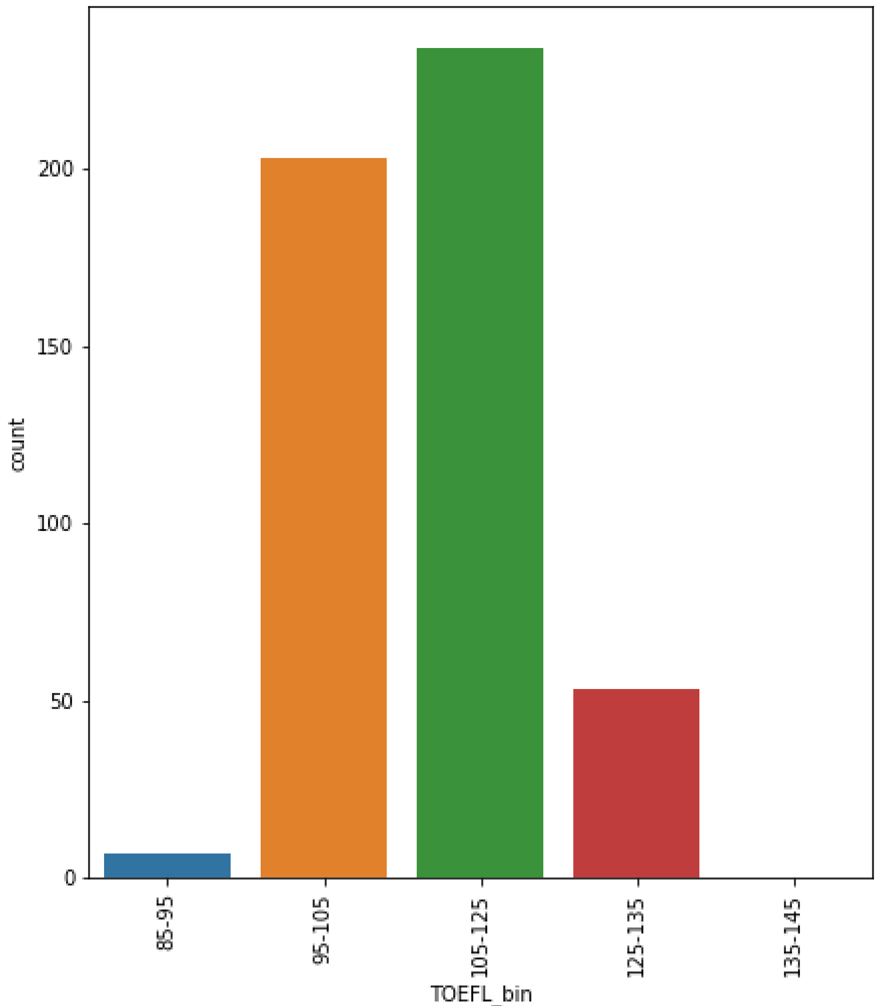


In [24]:

```
bins=[85,95,105,115,125,135]
labels=['85-95','95-105','105-125','125-135','135-145']
df['TOEFL_bin']=pd.cut(df['TOEFL Score'], bins=bins, labels=labels)
# Created bins for hospitalization charges
fig,ax=plt.subplots(figsize=(7,8))
plt.xticks(rotation=90)
sns.countplot(df['TOEFL_bin'])
# Bins were created for TOEFL score
# Most students score between 95-105 and 105-125.
# Least number of students score between 85-95
```

Out[24]:

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

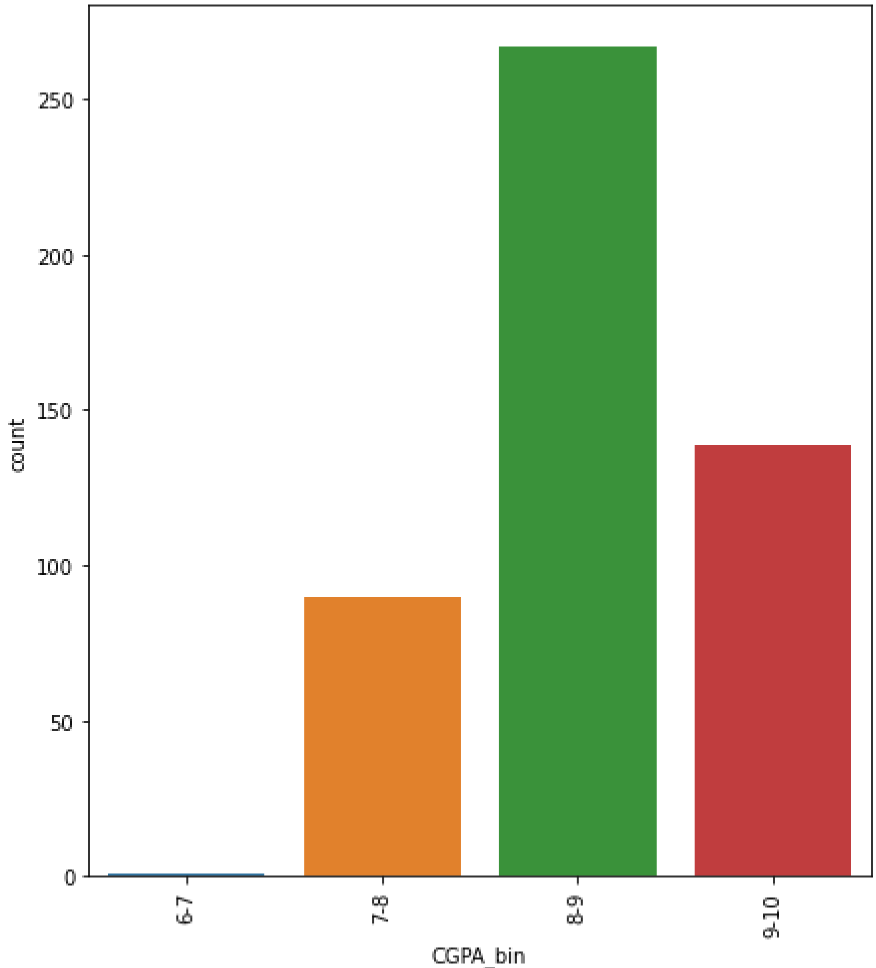


In [25]:

```
bins=[6,7,8,9,10]
labels=['6-7','7-8','8-9','9-10']
df['CGPA_bin']=pd.cut(df['CGPA'], bins=bins, labels=labels)
# Created bins for hospitalization charges
fig,ax=plt.subplots(figsize=(7,8))
plt.xticks(rotation=90)
sns.countplot(df['CGPA_bin'])
# Bins were created for CGPA score
# Most students score between 7-8 and 8-9.
# Least number of students score between 6-7
```

Out[25]:

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

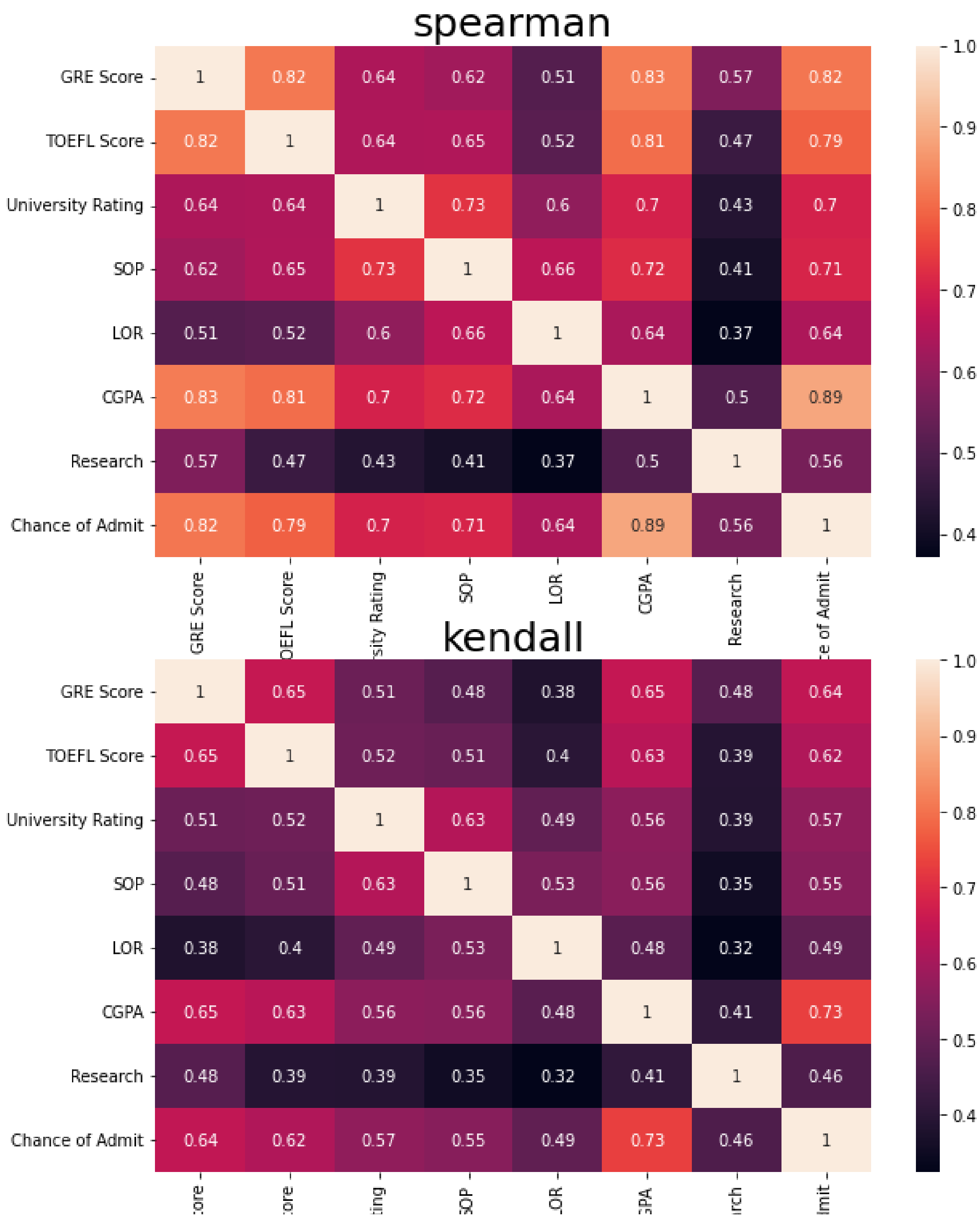


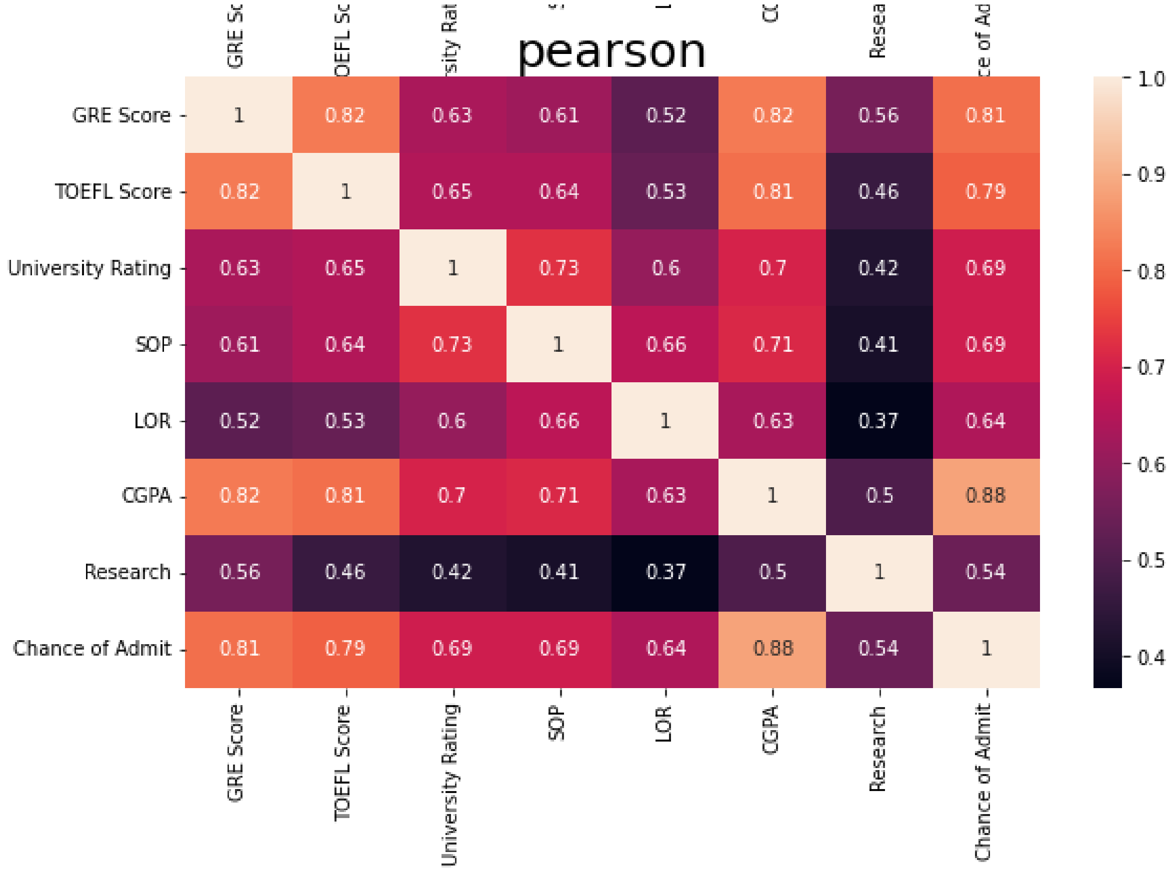
In [26]:

```
df.drop(['GRE_bin','TOEFL_bin','CGPA_bin'],axis=1,inplace=True)
```

In [27]:

```
fig, axes = plt.subplots(3, 1, figsize=(10, 20))


sns.heatmap(df.corr(method ='spearman'),ax=axes[0],annot=True)
axes[0].set_title('spearman',fontsize=25)
sns.heatmap(df.corr(method ='kendall'),ax=axes[1],annot=True)
axes[1].set_title('kendall',fontsize=25)
sns.heatmap(df.corr(method ='pearson'),ax=axes[2],annot=True)
axes[2].set_title('pearson',fontsize=25)

plt.show()
# Below heat map shows spearman, kendall and pearson corr
# Chance of admit and GRE score has spearman corr of 0.82
# Chance of admit and CGPA has spearman corr of 0.89
# Chance of admit and research least spearman corr of 0.56
```





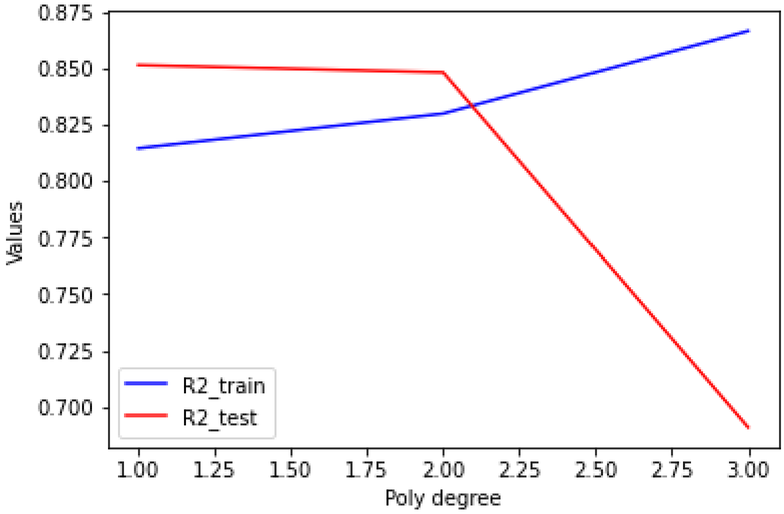
To check if increaing the degrees of the features will in the R2 Score for test and train

In [28]:

```
trainr=[]
testr=[]
degree=10
scores=[]
j=[1,2,3,4,5,6,7,8,9]
for i in range(1,degree):
    X= df.drop('Chance of Admit',axis=1)
    y=df['Chance of Admit']
    X_train, X_test,y_train, y_test = train_test_split(X,y ,
                                   random_state=9,
                                   test_size=0.2,
                                   shuffle=True)

    poly=PolynomialFeatures(i)
    X_train=poly.fit_transform(X_train)
    X_test=poly.fit_transform(X_test)

    scaler = StandardScaler()
    scaler.fit_transform(X_train)
    X_train=scaler.transform(X_train)
    X_test=scaler.transform(X_test)

    pca=decomposition.PCA()
    X_train=pca.fit_transform(X_train)
    X_test=pca.transform(X_test)

    reg = LinearRegression()
    reg.fit(X_train, y_train)
    trainr.append(reg.score(X_train, y_train))
    testr.append(reg.score(X_test, y_test))
```

In [29]:

```
trainr,testr
plt.plot(j[:3],trainr[:3],c='b')
plt.plot(j[:3],testr[:3],c='r')
plt.legend(["R2_train", "R2_test"])
plt.xlabel("Poly degree")
plt.ylabel("Values")
plt.show()
# Below plot shows that for degree 1 we have highest r2 scores for both train and test afte
# R2 test score decreases
# Will use degree 1 in out model
```



In [30]:

```
trainr,testr
# This case for over fitting the data points where R2 train score is near to 1 and R2 test
```

Out[30]:

```
([0.8146382872576212,
  0.8300725473573656,
  0.8667523767423233,
  0.9578318772534942,
  1.0,
  1.0,
  1.0,
  1.0,
  1.0],
 [0.8516084468112952,
  0.8483590233057319,
  0.690873952312723,
  -7.326626152993075,
  -264.2723753477777,
  -179.55540368729532,
  -165.4674492252204,
  -174.033589790237,
  -197.01900196331835])
```

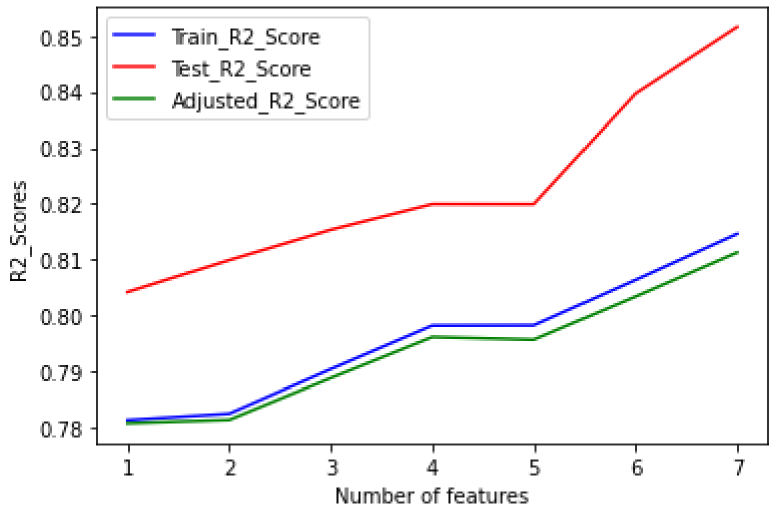
Finding the number of components needed for out model in PCA

In [31]:

```
# Changing PCA values

trainr=[]
testr=[]
adjr=[]
ls=[1,2,3,4,5,6,7]
for i in range(1,8):
    X= df.drop('Chance of Admit',axis=1)
    y=df['Chance of Admit']
    X_train, X_test,y_train, y_test = train_test_split(X,y ,
                                       random_state=9,
                                       test_size=0.2,
                                       shuffle=True)
    scaler = StandardScaler()
    scaler.fit_transform(X_train)
    X_train=scaler.transform(X_train)
    X_test=scaler.transform(X_test)
    pca=decomposition.PCA(n_components=i)
    X_train=pca.fit_transform(X_train)
    X_test=pca.transform(X_test)
    #print(pca.explained_variance_)
    reg = LinearRegression()
    reg.fit(X_train, y_train)
    trainr.append(reg.score(X_train, y_train))
    testr.append(reg.score(X_test, y_test))
    adjr.append(1 - (1-reg.score(X_train, y_train))*(len(y_train)-1)/(len(y_train)-X_train.
```

In [32]:

```
plt.plot(ls,trainr,c='b')
plt.plot(ls,testr,c='r')
plt.plot(ls,adjr,c='g')

plt.legend(["Train_R2_Score", "Test_R2_Score",'Adjusted_R2_Score'])
plt.xlabel("Number of features")
plt.ylabel("R2_Scores")
plt.show()
```



In [33]:

```
# R2 score for both train, test and adjusted R2 score increases with increase in number of
# WE will use all the 7 features in out model
```

Scaling,PCA

In [34]:

```
X= df.drop('Chance of Admit',axis=1)
y=df['Chance of Admit']
X_train, X_test,y_train, y_test = train_test_split(X,y ,
                                   random_state=8,
                                   test_size=0.19,
                                   shuffle=True)

scaler = StandardScaler()
scaler.fit_transform(X_train)
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)

pca=decomposition.PCA()
X_train=pca.fit_transform(X_train)
X_test=pca.transform(X_test)
```

VIF: All features have a VIF factor of <5 after scaling and PCA

In [35]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
X =pd.DataFrame(X_train)
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns[:]
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(len(X.columns[:]))]
print(vif_data)
```

```
   feature  VIF
0        0  1.0
1        1  1.0
2        2  1.0
3        3  1.0
4        4  1.0
5        5  1.0
6        6  1.0
```

Scaling,PCA

In [36]:

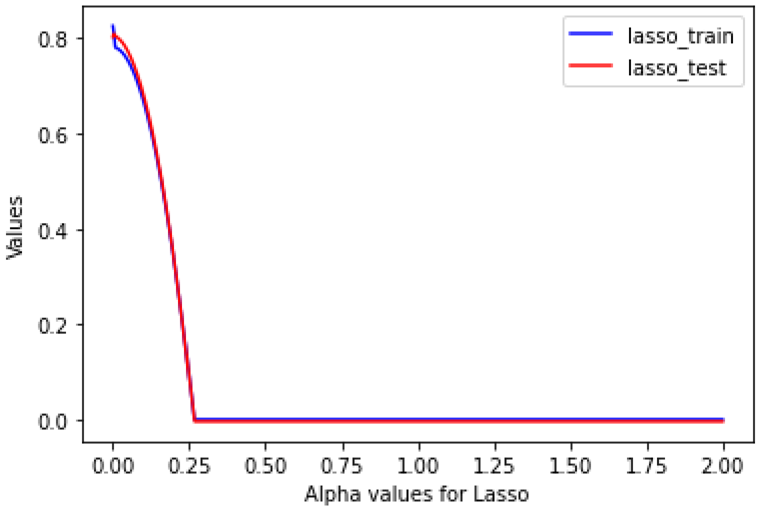
```
X= df.drop('Chance of Admit',axis=1)
y=df['Chance of Admit']
X_train, X_test,y_train, y_test = train_test_split(X,y ,
                                   random_state=8,
                                   test_size=0.19,
                                   shuffle=True)

scaler = StandardScaler()
scaler.fit_transform(X_train)
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
from sklearn import decomposition
pca=decomposition.PCA()
X_train=pca.fit_transform(X_train)
X_test=pca.transform(X_test)
```

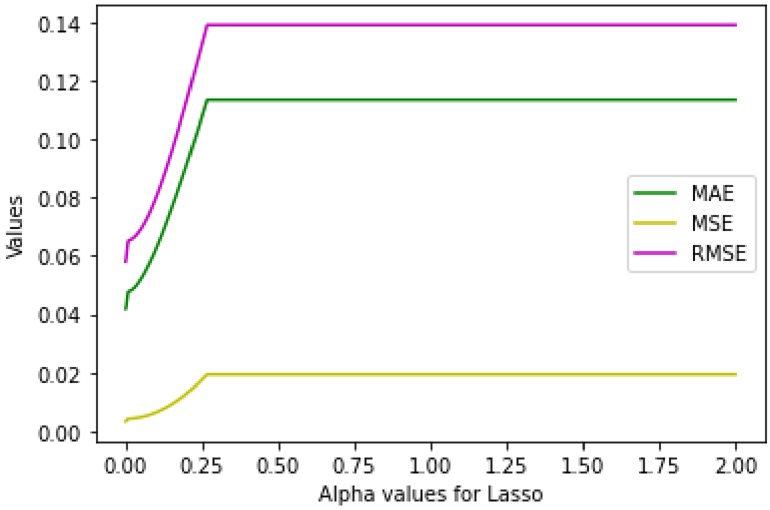
Lasso Reg

1. Alpha value is varied from 0.001 to 2

In [37]:

```
lasso_train=[]
lasso_test=[]
mae=[]
mse=[]
rmse=[]
ls=np.arange(0.001,2,0.001)
for i in ls:
    reg = Lasso(alpha=i)
    reg.fit(X_train, y_train)
    y_hat=reg.predict(X_train)
    mae.append(metrics.mean_absolute_error(y_train, y_hat))
    m=metrics.mean_squared_error(y_train, y_hat)
    mse.append(metrics.mean_squared_error(y_train, y_hat))
    rmse.append(np.sqrt(m))
    lasso_train.append(reg.score(X_train, y_train))
    lasso_test.append(reg.score(X_test, y_test))
plt.plot(ls,lasso_train,c='b')
plt.plot(ls,lasso_test,c='r')
plt.legend(["lasso_train", "lasso_test"])
plt.xlabel("Alpha values for Lasso")
plt.ylabel("Values")
plt.show()
plt.plot(ls,mae,c='g')
plt.plot(ls,mse,c='y')
plt.plot(ls,rmse,c='m')
plt.legend(['MAE','MSE','RMSE'])
plt.xlabel("Alpha values for Lasso")
plt.ylabel("Values")
plt.show()
print('R2 Score for train data is',lasso_train[np.argmax(lasso_test)],'R2 Score for test da
print(ls[np.argmax(lasso_test)])
```






```
R2 Score for train data is 0.8056384249328674 R2 Score for test data is 0.80
94705183668697
0.005
```

R2 score for train and test are plotted for different values for alpha. R2 score for train is 0.8279 and

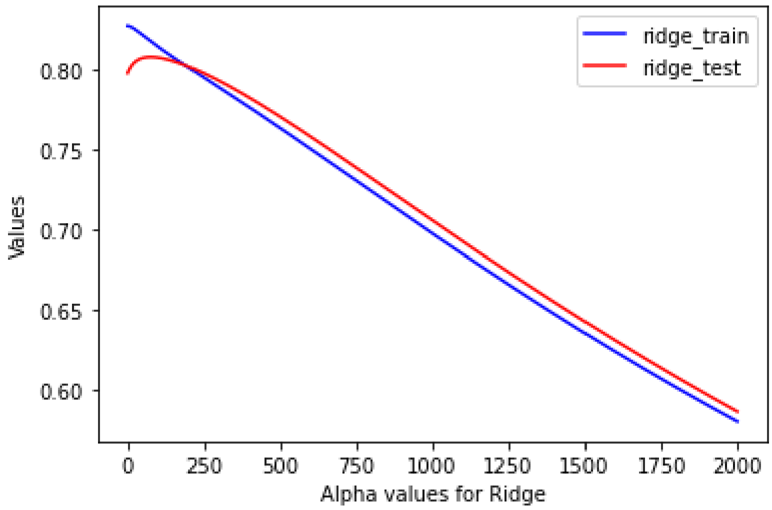
R2 score for test is 0.780 for Lasso. Plotted MAE,MSE and RMSE values for all values for alpha.

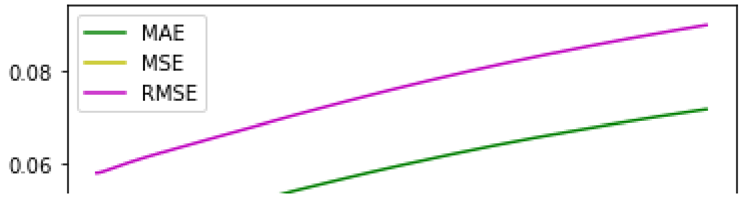
Optimum value of alpha is 0.001

Ridge Reg

In [38]:

```
ridge_train=[]
ridge_test=[]
mae=[]
mse=[]
rmse=[]
ls=np.arange(0.5,2000,0.5)
for i in ls:
    reg = Ridge(alpha=i)
    reg.fit(X_train, y_train)
    y_hat=reg.predict(X_train)
    mae.append(metrics.mean_absolute_error(y_train, y_hat))
    m=metrics.mean_squared_error(y_train, y_hat)
    mse.append(metrics.mean_squared_error(y_train, y_hat))
    rmse.append(np.sqrt(m))
    ridge_train.append(reg.score(X_train, y_train))
    ridge_test.append(reg.score(X_test, y_test))
plt.plot(ls,ridge_train,c='b')
plt.plot(ls,ridge_test,c='r')
plt.legend(["ridge_train", "ridge_test"])
plt.xlabel("Alpha values for Ridge")
plt.ylabel("Values")
plt.show()
plt.plot(ls,mae,c='g')
plt.plot(ls,mse,c='y')
plt.plot(ls,rmse,c='m')
plt.legend(['MAE','MSE','RMSE'])
plt.xlabel("Alpha values for Ridge")
plt.ylabel("Values")
plt.show()
print('R2 Score for train data is',ridge_train[np.argmax(ridge_test)],'R2 Score for test da
print(ls[np.argmax(lasso_test)])
```






```
R2 Score for train data is 0.8173862608805565 R2 Score for test data is 0.80
75875926468006
2.5
```

R2 score for train and test are plotted for different values for alpha. R2 score for train is 0.8289 and

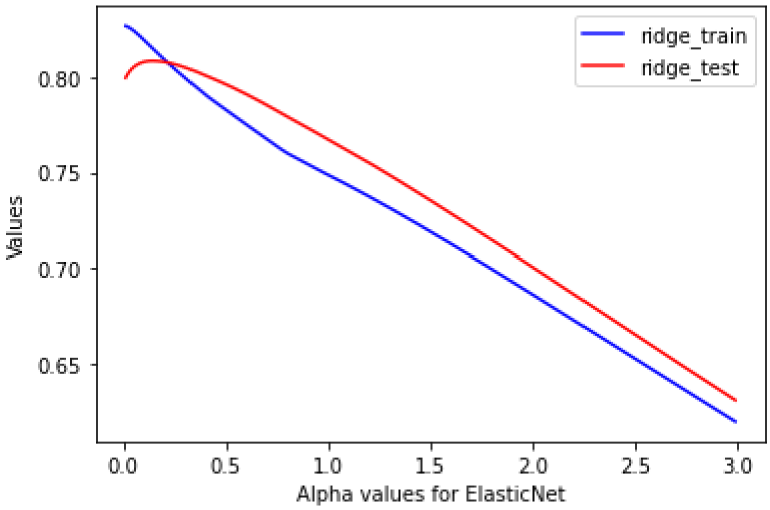
R2 score for test is 0.7798 for Lasso. Plotted MAE,MSE and RMSE values for all values for alpha

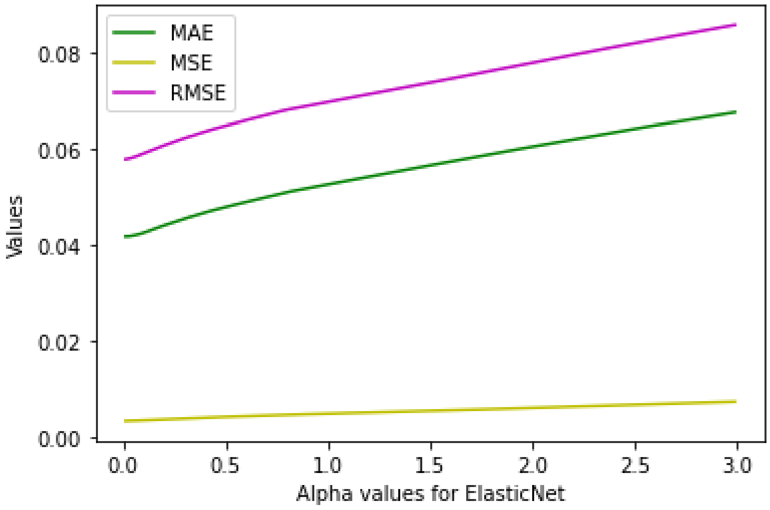
Optimum value of alpha is 0.5

Elastic Net Reg

In [39]:

```
e_train=[]
e_test=[]
mae=[]
mse=[]
rmse=[]
ls=np.arange(0.01,3,0.01)
for i in ls:
    reg = ElasticNet(alpha=i,l1_ratio=0.01)
    reg.fit(X_train, y_train)
    y_hat=reg.predict(X_train)
    mae.append(metrics.mean_absolute_error(y_train, y_hat))
    m=metrics.mean_squared_error(y_train, y_hat)
    mse.append(metrics.mean_squared_error(y_train, y_hat))
    rmse.append(np.sqrt(m))
    e_train.append(reg.score(X_train, y_train))
    e_test.append(reg.score(X_test, y_test))
plt.plot(ls,e_train,c='b')
plt.plot(ls,e_test,c='r')
plt.legend(["ridge_train", "ridge_test"])
plt.xlabel("Alpha values for ElasticNet")
plt.ylabel("Values")
plt.show()
plt.plot(ls,mae,c='g')
plt.plot(ls,mse,c='y')
plt.plot(ls,rmse,c='m')
plt.legend(['MAE','MSE','RMSE'])
plt.xlabel("Alpha values for ElasticNet")
plt.ylabel("Values")
plt.show()
print('R2 Score for train data is',e_train[np.argmax(e_test)],'R2 Score for test data is',e
print(ls[np.argmax(lasso_test)])
```






```
R2 Score for train data is 0.8153431598147003 R2 Score for test data is 0.80
855108305522
0.05
```

R2 score for train and test are plotted for different values for alpha and l1_ratio is set to constant 0.01. R2 score for train is 0.8153 and

R2 score for test is 0.8085 for elastic net. Plotted MAE,MSE and RMSE values for all values for alpha

Optimum value of alpha is 0.01

Simple linear reg, Lasso, Ridge and Elastic Net Coeff, intercept and R2 scores

Simple linear regression and Elasticnet are the best models

In [40]:

```
reg = LinearRegression()
reg.fit(X_train, y_train)
y_hat=reg.predict(X_train)
print(reg.coef_,'Are the coeff')
print(reg.intercept_,'Is the intercept')
print('R2 score train',reg.score(X_train, y_train))
print('R2 score test',reg.score(X_test, y_test))
print('MAE',metrics.mean_absolute_error(y_train, y_hat))
print('MSE',metrics.mean_squared_error(y_train, y_hat))
m=metrics.mean_squared_error(y_train, y_hat)
print('RMSE',np.sqrt(m))
```

```
[-0.05669896 -0.00987978 -0.01760719 -0.01997257  0.00617299 -0.04422795
 -0.02844341] Are the coeff
0.7222139303482586 Is the intercept
R2 score train 0.8270205536350423
R2 score test 0.7973935834274464
MAE 0.04174268656156689
MSE 0.0033462183711267255
RMSE 0.057846506991578374
```

In [41]:

```
reg = Lasso(alpha=0.001)
reg.fit(X_train, y_train)
y_hat=reg.predict(X_train)
print(reg.coef_,'Are the coeff')
print(reg.intercept_,'Is the intercept')
print('R2 score train',reg.score(X_train, y_train))
print('R2 score test',reg.score(X_test, y_test))
print('MAE',metrics.mean_absolute_error(y_train, y_hat))
print('MSE',metrics.mean_squared_error(y_train, y_hat))
m=metrics.mean_squared_error(y_train, y_hat)
print('RMSE',np.sqrt(m))
```

```
[-0.05648628 -0.008577   -0.01575925 -0.01746121  0.00239568 -0.03868441
 -0.02159208] Are the coeff
0.7222139303482586 Is the intercept
R2 score train 0.8258808595805425
R2 score test 0.8028027399177385
MAE 0.041811948325053104
MSE 0.003368265297873069
RMSE 0.058036758161298684
```

In [42]:

```
reg = Ridge(alpha=1)
reg.fit(X_train, y_train)
y_hat=reg.predict(X_train)
print(reg.coef_,'Are the coeff')
print(reg.intercept_,'Is the intercept')
print('R2 score train',reg.score(X_train, y_train))
print('R2 score test',reg.score(X_test, y_test))
print('MAE',metrics.mean_absolute_error(y_train, y_hat))
print('MSE',metrics.mean_squared_error(y_train, y_hat))
m=metrics.mean_squared_error(y_train, y_hat)
print('RMSE',np.sqrt(m))
```

```
[-0.05666898 -0.00984787 -0.01752662 -0.01984858  0.00611553 -0.04362635
 -0.02796677] Are the coeff
0.7222139303482586 Is the intercept
R2 score train 0.8270146623518111
R2 score test 0.797860296546433
MAE 0.0417410191923442
MSE 0.0033463323356502117
RMSE 0.05784749204287262
```

In [43]:

```
reg = ElasticNet(alpha=0.01,l1_ratio=0.01)
reg.fit(X_train, y_train)
y_hat=reg.predict(X_train)
print(reg.coef_,'Are the coeff')
print(reg.intercept_,'Is the intercept')
print('R2 score train',reg.score(X_train, y_train))
print('R2 score test',reg.score(X_test, y_test))
print('MAE',metrics.mean_absolute_error(y_train, y_hat))
print('MSE',metrics.mean_squared_error(y_train, y_hat))
m=metrics.mean_squared_error(y_train, y_hat)
print('RMSE',np.sqrt(m))
```

```
[-0.05655861 -0.00962536 -0.01710939 -0.01924301  0.00558636 -0.04140144
 -0.02599508] Are the coeff
0.7222139303482586 Is the intercept
R2 score train 0.8268708717994689
R2 score test 0.7996892005608697
MAE 0.04174607059890638
MSE 0.0033491139065128365
RMSE 0.05787152932585104
```

Features sorted based on the highest weights

In [44]:

```
X= df.drop('Chance of Admit',axis=1)
y=df['Chance of Admit']
X_train, X_test,y_train, y_test = train_test_split(X,y ,
                                   random_state=0,
                                   test_size=0.19,
                                   shuffle=True)
scaler = StandardScaler()
scaler.fit_transform(X_train)
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
reg = LinearRegression()
reg.fit(X_train, y_train)
new=pd.DataFrame(df.columns[:-1],reg.coef_)
new.reset_index(inplace=True)
new.columns=['Coff','Feature']
new.sort_values(by=['Coff'], ascending=False, inplace=False)
```

Out[44]:

	Coff	Feature
5	0.070183	CGPA
0	0.026712	GRE Score
4	0.017505	LOR
6	0.012377	Research
1	0.010530	TOEFL Score
2	0.006295	University Rating
3	0.004280	SOP

Linear regression using Stats.model

In [45]:

```
X= df.drop('Chance of Admit',axis=1)
y=df['Chance of Admit']
X=sm.add_constant(X)
model=sm.OLS(y,X)
fit=model.fit()
fit.summary()
```

Out[45]:

OLS Regression Results

Dep. Variable:	Chance of Admit	**R-squared:**	0.822
Model:	OLS	**Adj. R-squared:**	0.820
Method:	Least Squares	**F-statistic:**	323.3
Date:	Wed, 26 Oct 2022	**Prob (F-statistic):**	6.00e-179
Time:	15:41:54	**Log-Likelihood:**	706.09
No. Observations:	497	**AIC:**	-1396.
Df Residuals:	489	**BIC:**	-1363.
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.2351	0.103	-12.010	0.000	-1.437	-1.033
GRE Score	0.0018	0.000	3.618	0.000	0.001	0.003
TOEFL Score	0.0026	0.001	3.005	0.003	0.001	0.004
University Rating	0.0054	0.004	1.447	0.149	-0.002	0.013
SOP	0.0039	0.005	0.858	0.391	-0.005	0.013
LOR	0.0160	0.004	3.927	0.000	0.008	0.024
CGPA	0.1185	0.010	12.423	0.000	0.100	0.137
Research	0.0237	0.006	3.659	0.000	0.011	0.037

Omnibus:	111.275	**Durbin-Watson:**	0.819
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	257.638
Skew:	-1.152	**Prob(JB):**	1.13e-56
Kurtosis:	5.670	**Cond. No.**	1.30e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.3e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

Conclusion Results of above summary

In [46]:

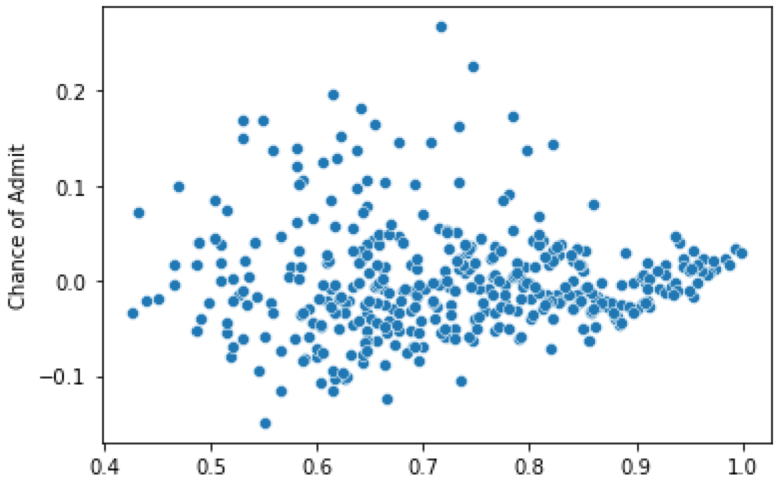
```
# H0 : variable X has no influence on Y
# Ha : X has significant impact on Y
# if p_value>0.05:
#     variable X has no influence on Y
# else:
#     X has significant impact on
# 1. P<|t| shows the p_value. Only GRE,TOEFL,LOR,CGPA and research has a significant impact
# 2. Coeff and STD error are displayed in the table for each feature
# 3. R2 score is 0.82
# 4. R2 and adj R2 are very close, that means there is no there is no feature that we need
# 5. Prob (F-statistic):    6.00e-179 this says there is a good relation between my indepen
# and dependent variables
```

Assumptions for linear regression

1. Linearity of variables (no pattern in the residual plot)

In [47]:

```
sns.scatterplot(reg.predict(X_train),reg.predict(X_train)-y_train)
plt.show()
# There is no pattern in the residual
```



In [48]:

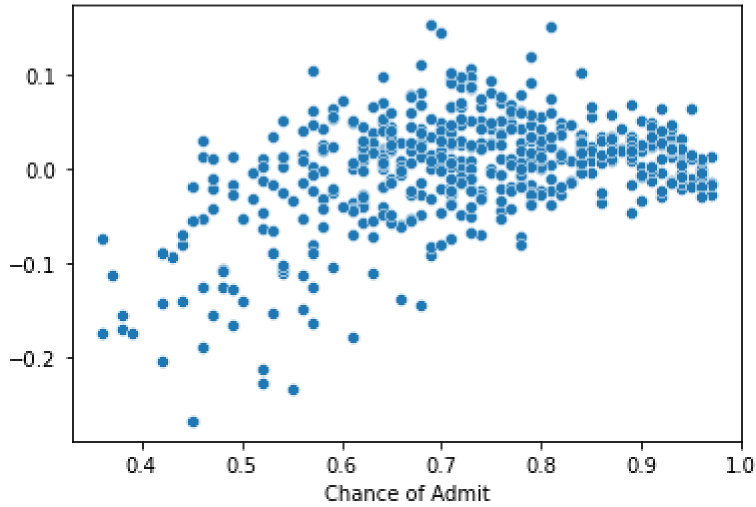
```
X= df.drop('Chance of Admit',axis=1)
y=df['Chance of Admit']
X=sm.add_constant(X)
model=sm.OLS(y,X)
fit=model.fit()
```

In [49]:

```
sns.scatterplot(y,fit.resid)
# scatter plot is centered around the line 0
```

Out[49]:

<AxesSubplot:xlabel='Chance of Admit'>

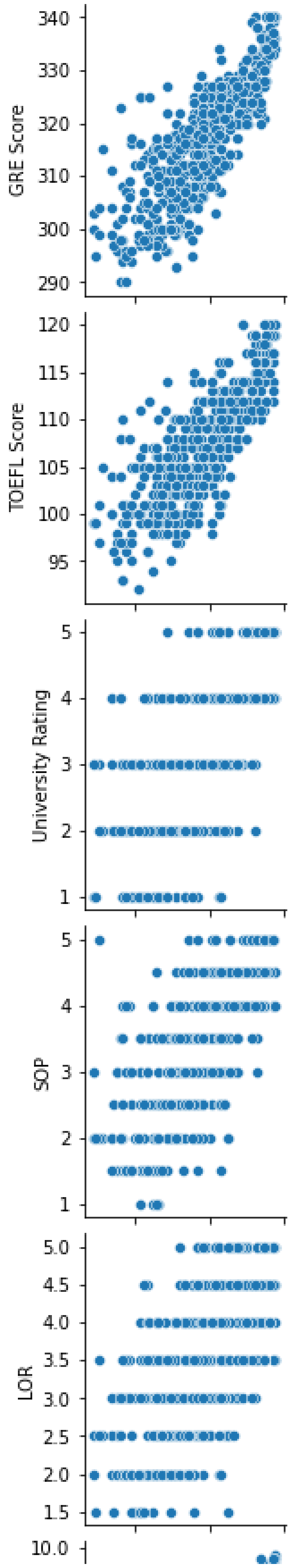


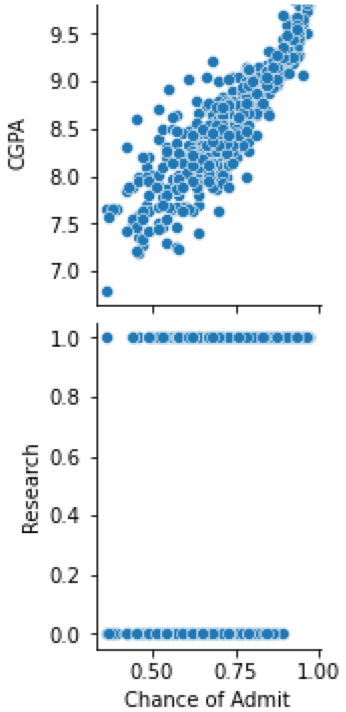
In [50]:

```
sns.pairplot(df,x_vars=['Chance of Admit'], y_vars=['GRE Score', 'TOEFL Score', 'University
       'Research'], kind='scatter')
# All continuous variables have linear relationship with chance of admit
```

Out[50]:

<seaborn.axisgrid.PairGrid at 0x22cc4ac8cd0>





2. Multicollinearity check by VIF score (variables are dropped one-by-one till none has VIF>5)

In [51]:

```
X= df.drop('Chance of Admit',axis=1)
y=df['Chance of Admit']
X_train, X_test,y_train, y_test = train_test_split(X,y ,
                                   random_state=0,
                                   test_size=0.19,
                                   shuffle=True)
scaler = StandardScaler()
scaler.fit_transform(X_train)
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
```

In [52]:

```
X =pd.DataFrame(X_train)
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns[:]
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(len(X.columns[:]))
print(vif_data)
# All the VIF's for 7 features have values less than 5. Hence no feature is removed
```

```
   feature       VIF
0        0  4.663535
1        1  3.879927
2        2  2.499218
3        3  2.704754
4        4  1.896254
5        5  5.012407
6        6  1.468325
```

3. Normality of residuals

In [53]:

```
X= df.drop('Chance of Admit',axis=1)
y=df['Chance of Admit']
X_train, X_test,y_train, y_test = train_test_split(X,y ,
                                   random_state=0,
                                   test_size=0.19,
                                   shuffle=True)
scaler = StandardScaler()
scaler.fit_transform(X_train)
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
reg = LinearRegression()
reg.fit(X_train, y_train)
```

Out[53]:

```
LinearRegression()
```

In [54]:

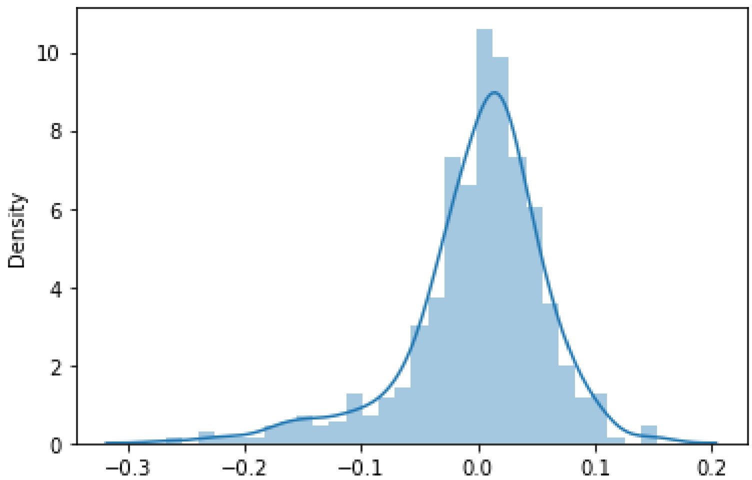
```
X= df.drop('Chance of Admit',axis=1)
y=df['Chance of Admit']
X=sm.add_constant(X)
model=sm.OLS(y,X)
fit=model.fit()
```

In [55]:

```
ans=fit.resid
```

In [56]:

```
sns.distplot(ans)
plt.show()
# Residual plot looks like normal distribution
```



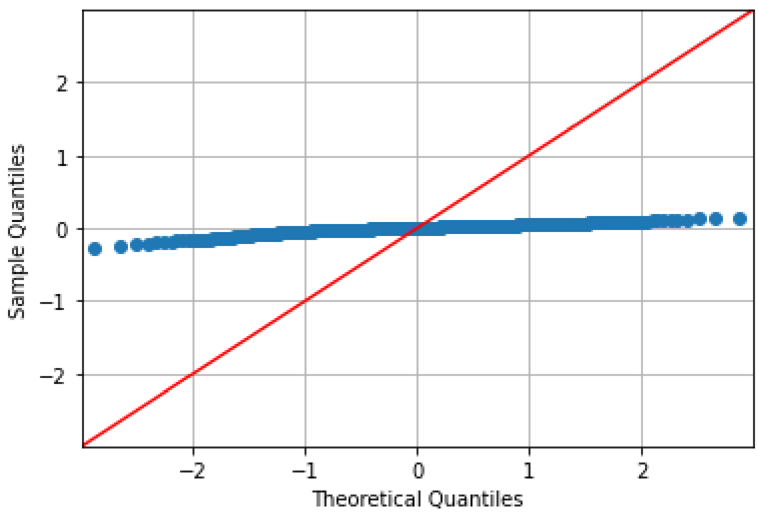
In [57]:

```
# Normality Tests
def qqplot(a):
    print('This test is for visual only')
    fig=sm.qqplot(a,line='45')
    plt.grid()
    plt.show()
def kstest(a):

    stat,p_value=stats.kstest(a,'norm')
    print('Ho: The sample follows normal distribution')
    print('Ha: The sample does not follows normal distribution')
    print('stat=%.3f, p_value=%.3f' % (stat, p_value))
    if p_value>0.05:
        print('The sample follows normal distribution')
    else:
        print('The sample does not follows normal distribution')
def shapiro(a):
    A=pd.DataFrame()
    stat,p_value=stats.shapiro(a)
    print('Ho: The sample follows normal distribution')
    print('Ha: The sample does not follows normal distribution')
    print()
    print('stat=%.3f, p_value=%.3f' % (stat, p_value))
    if p_value>0.05:
        print('The sample follows normal distribution')
    else:
        print('The sample does not follows normal distribution')
```

In [58]:

```
qqplot(ans)
# QQ plot for residuals. Shows it is not normal distribution
```

```
This test is for visual only
```



In [59]:

```
kstest(ans)
# KS test plot for residuals. Shows it is not normal distribution
```

```
Ho: The sample follows normal distribution
Ha: The sample does not follows normal distribution
stat=0.448, p_value=0.000
The sample does not follows normal distribution
```

In [60]:

```
shapiro(ans)
# Shapiro test plot for residuals. Shows it is not normal distribution
```

```
Ho: The sample follows normal distribution
Ha: The sample does not follows normal distribution

stat=0.927, p_value=0.000
The sample does not follows normal distribution
```

4.Test for Homoscedasticity

Goldfeld-Quandt Test

In [61]:

```
X= df.drop('Chance of Admit',axis=1)
y=df['Chance of Admit']
X=sm.add_constant(X)
model=sm.OLS(y,X)
fit=model.fit()
# Goldfeld-Quanndt test for homoscedasticity.
```

In [62]:

```
name = ["F statistic", "p_value"]
test = sms.het_goldfeldquandt(fit.resid, fit.model.exog)
lzip(name, test)
```

Out[62]:

```
[('F statistic', 0.44517108552704354), ('p_value', 0.9999999996886626)]
```

H0=Null Hypothesis: Homoscedasticity is present .

H1=Alternate Hypothesis: Homoscedasticity is not present .

Homoscedasticity is present

In [63]:

```
if test[1]>0.05:
    print('Homoscedasticity is present ')
else:
    print('Homoscedasticity is not present')
```

```
Homoscedasticity is present
```

Breusch-Pagan

In [64]:

```
# H0=Null Hypothesis: Homoscedasticity is present .
# H1=Alternate Hypothesis: Homoscedasticity is not present .
```

In [65]:

```
name = ["Lagrange multiplier statistic", "p-value", "f-value", "f p-value"]
test = sms.het_breuschpagan(fit.resid, fit.model.exog)
lzip(name, test)
```

Out[65]:

```
[('Lagrange multiplier statistic', 28.320557631705583),
 ('p-value', 0.00019233754769815516),
 ('f-value', 4.221207634529353),
 ('f p-value', 0.00015500546087255574)]
```

In [66]:

```
if test[1]>0.05:
    print('Homoscedasticity is present ')
else:
    print('Homoscedasticity is not present')
```

```
Homoscedasticity is not present
```

5. The mean of residuals is nearly zero

In [67]:

```
fit.resid.mean(),fit.resid.median()
```

Out[67]:

```
(-3.9829530239570504e-16, 0.009015175239256656)
```

Mean and median of the Residuals are close to zero