In [191]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import sys
import seaborn as sns
import os
import warnings
warnings.filterwarnings("ignore")
from scipy import stats

from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression, Ridge, Lasso,ElasticNet
from sklearn.model_selection import train_test_split,cross_val_score
#Feture scaling
from sklearn.preprocessing import StandardScaler
#metrics
from sklearn.metrics import mean_absolute_error,r2_score,mean_squared_error,mean_absolute_percentage_error

import statsmodels.api as sm
import statsmodels.formula.api as smf
import statsmodels.tools.tools as smt
import statsmodels.stats.diagnostic as smd
```

In [41]:

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

Out[41]:

	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

Problem Statement:

The purpose of helping students in shortlisting universities with their profiles. The predicted output gives them a fair idea about their chances for a particular university.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.

columns profile:

The dataset contains several parameters which are considered important during the application for Masters Programs. The parameters included are:

1. **GRE Scores (out of 340)**
2. **TOEFL Scores (out of 120)**
3. **University Rating (out of 5)**
4. **Statement of Purpose (out of 5) 5.Letter of Recommendation Strength (out of 5)**
5. **Undergraduate GPA (out of 10)**
6. **Research Experience (either 0 or 1)**
7. **Chance of Admit (ranging from 0 to 1)**

In [42]:

```
data.describe()
```

Out[42]:

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

In [43]:

```
data.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
```

Updating column Name

In [44]:

```
#removing the trailing space in the column name
data.rename(columns = {'Chance of Admit ' :'Chance of Admit', 'LOR ':'LOR'}, inplace = True)
```

EDA

In [45]:

```
print("{} columns in the dataset.".format(len(data.columns)),end='\n--------\n')

print(list(data.columns),end='\n--------\n')
print("{} samples in the dataset.".format(data.shape[0]))
```

```
9 columns in the dataset.
--------
['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR', 'CGPA', 'Research', 'Chance of Admit']
--------
500 samples in the dataset.
```

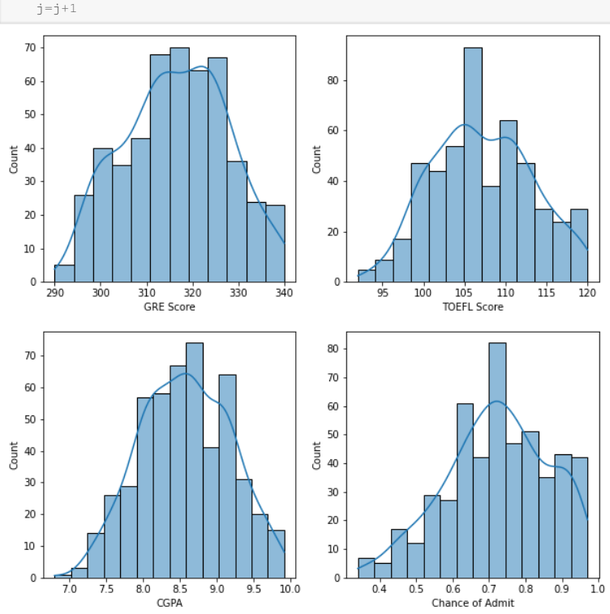
In [46]:

```
print("Not having research : ", len(data[data.Research == 0]))
print("having research : ", len(data[data.Research == 1]))
```

```
Not having research :  220
having research :  280
```

In [47]:

```
fig, axs = plt.subplots(2, 2,figsize=(10, 10))
k=0
j=0
for i in ['GRE Score','TOEFL Score','CGPA','Chance of Admit']:
  sns.histplot(data[i],ax=axs[j,k],kde=True)
  if k!=1:
    k=k+1
  else:
    k=0
    j=j+1
```

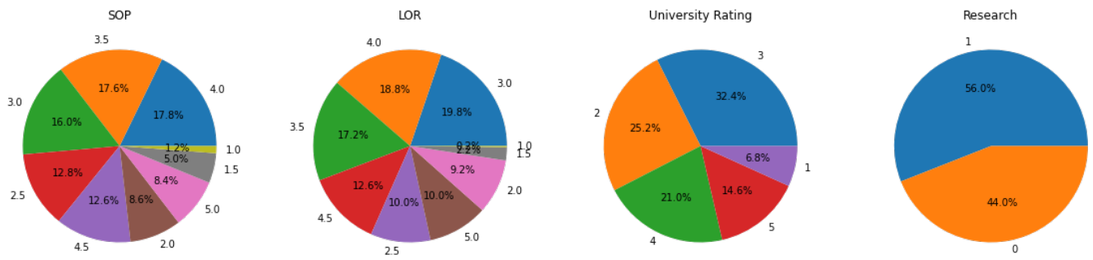


In [48]:

```
cat_col = ['SOP', 'LOR', 'University Rating', 'Research']

plt.figure(figsize=(20,5))

for i in range(len(cat_col)):
    plt.subplot(1,4,i+1)
    val_count=data[cat_col[i]].value_counts()
    plt.pie(val_count, labels = val_count.index, autopct = '%1.1f%%', textprops = {'fontsize':10})
    print("*{}* Column Distribution \n {}".format(cat_col[i],val_count),end='\n----------------------\n')
    plt.title(cat_col[i])
plt.show()
```

```
*SOP* Column Distribution
 4.0    89
3.5    88
3.0    80
2.5    64
4.5    63
2.0    43
5.0    42
1.5    25
```

```
1.0     6
Name: SOP, dtype: int64
----------------------
*LOR* Column Distribution
 3.0    99
4.0     94
3.5     86
4.5     63
2.5     50
5.0     50
2.0     46
1.5     11
1.0      1
Name: LOR, dtype: int64
----------------------
*University Rating* Column Distribution
 3    162
2     126
4     105
5      73
1      34
Name: University Rating, dtype: int64
----------------------
*Research* Column Distribution
 1    280
0    220
Name: Research, dtype: int64
----------------------
```



Missing value treatment.

In [49]:

```
# Function to create a data frame with number and percentage of missing data in a data frame
def missing_to_df(df):
    #Number and percentage of missing data in training data set for each column
    total_missing_df = df.isnull().sum().sort_values(ascending =False)
    percent_missing_df = (df.isnull().sum()/df.isnull().count()*100).sort_values(ascending=False)
    missing_data_df = pd.concat([total_missing_df, percent_missing_df], axis=1, keys=['Total', 'Percent'])
    return missing_data_df
missing_df = missing_to_df(data)
missing_df[missing_df['Total'] > 0]
```

Out[49]:

	Total	Percent

Outlier detection

In [50]:

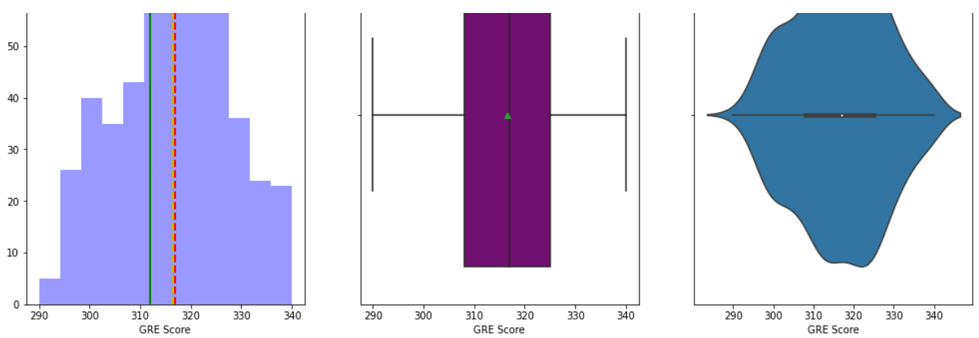
```
def dist_box_violin(data):
 # function plots a combined graph for univariate analysis of continous variable
 #to check spread, central tendency , dispersion and outliers
    Name=data.name.upper()
    fig, axes =plt.subplots(1,3,figsize=(17, 7))
    fig.suptitle("SPREAD OF DATA FOR "+ Name  , fontsize=18, fontweight='bold')
    sns.distplot(data,kde=False,color='Blue',ax=axes[0])
    axes[0].axvline(data.mean(), color='y', linestyle='--',linewidth=2)
    axes[0].axvline(data.median(), color='r', linestyle='dashed', linewidth=2)
    axes[0].axvline(data.mode()[0],color='g',linestyle='solid',linewidth=2)
    axes[0].legend({'Mean':data.mean(),'Median':data.median(),'Mode':data.mode()})
    sns.boxplot(x=data,showmeans=True, orient='h',color="purple",ax=axes[1])
    #just exploring violin plot
    sns.violinplot(data,ax=axes[2],showmeans=True)
    plt.show()
```

In [190]:

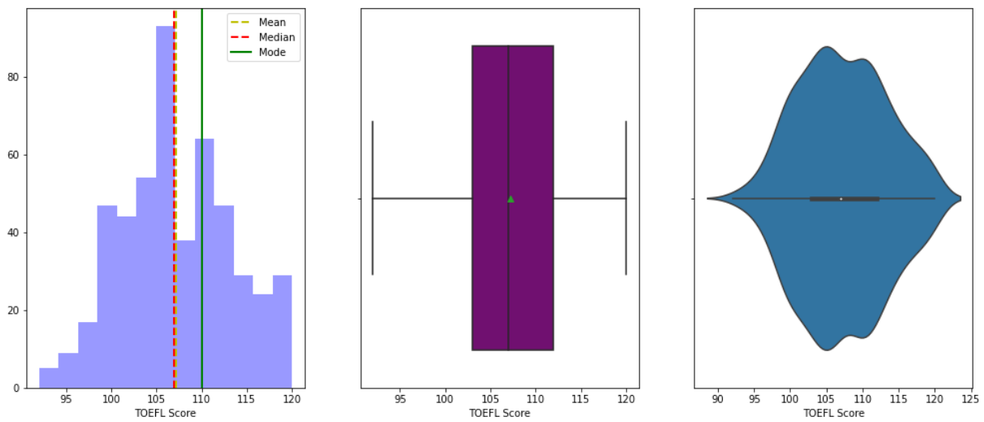
```

```

In [51]:

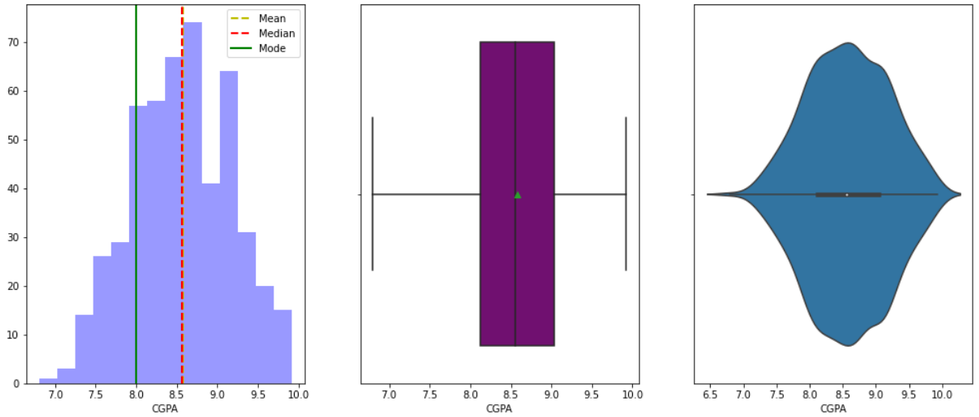
```
#['GRE Score','TOEFL Score','CGPA','Chance of Admit']
for i in ['GRE Score','TOEFL Score','CGPA','Chance of Admit']:
  dist_box_violin(data[i])
```



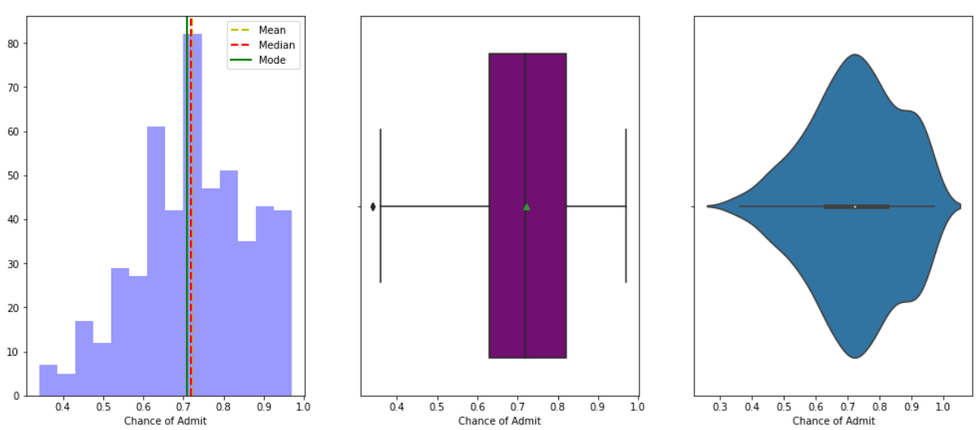
SPREAD OF DATA FOR TOEFL SCORE



SPREAD OF DATA FOR CGPA



SPREAD OF DATA FOR CHANCE OF ADMIT



In [52]:

```
for eachcol in ['GRE Score','TOEFL Score','CGPA','Chance of Admit']:
  #calculating  values at each percntile 0,10,20,30,40,50,60,70,80,90,100
  print(f"**** {eachcol} Column Percentile Dictribution *****")
  for i in range(0,100,10):
      var =data[eachcol].values
      var = np.sort(var,axis = None)
      print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
  print("100 percentile value is ",var[-1])
```

```
**** GRE Score Column Percentile Dictribution *****
0 percentile value is 290
10 percentile value is 300
20 percentile value is 306
30 percentile value is 311
40 percentile value is 313
50 percentile value is 317
60 percentile value is 320
70 percentile value is 324
80 percentile value is 327
90 percentile value is 331
100 percentile value is  340
**** TOEFL Score Column Percentile Dictribution *****
0 percentile value is 92
10 percentile value is 99
20 percentile value is 102
30 percentile value is 104
40 percentile value is 105
50 percentile value is 107
60 percentile value is 109
70 percentile value is 110
80 percentile value is 113
90 percentile value is 116
100 percentile value is  120
**** CGPA Column Percentile Dictribution *****
0 percentile value is 6.8
10 percentile value is 7.81
20 percentile value is 8.02
30 percentile value is 8.22
40 percentile value is 8.42
50 percentile value is 8.56
60 percentile value is 8.75
70 percentile value is 8.96
80 percentile value is 9.13
90 percentile value is 9.38
100 percentile value is  9.92
**** Chance of Admit Column Percentile Dictribution *****
0 percentile value is 0.34
10 percentile value is 0.53
20 percentile value is 0.61
30 percentile value is 0.65
40 percentile value is 0.7
50 percentile value is 0.72
60 percentile value is 0.76
70 percentile value is 0.8
80 percentile value is 0.86
90 percentile value is 0.92
100 percentile value is  0.97
```

In [53]:

```
#sns.lmplot(y="Chance of Admit", x = val_col[i], hue="Research", data=data);
```

In [53]:

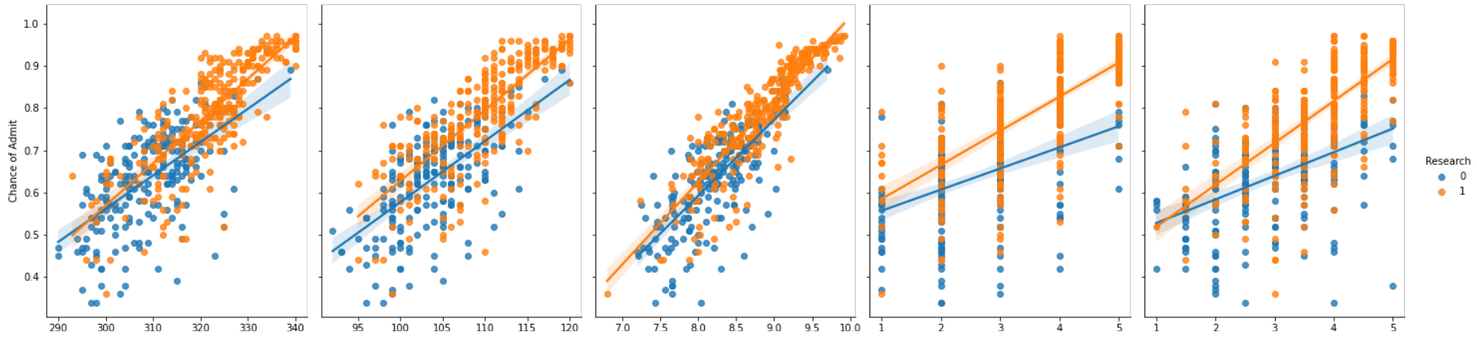
- **As we seen from above distribution of numerical features it is slighly left skewed but we should not call then as outliers.**
- **As we seen from above distribution of target column(Chance of Admit) it is slighly left skewed and it might effect on model performance due to very less data on left skewed side.Lets see further how it will impact**

Bivariate Analysis

In [53]:

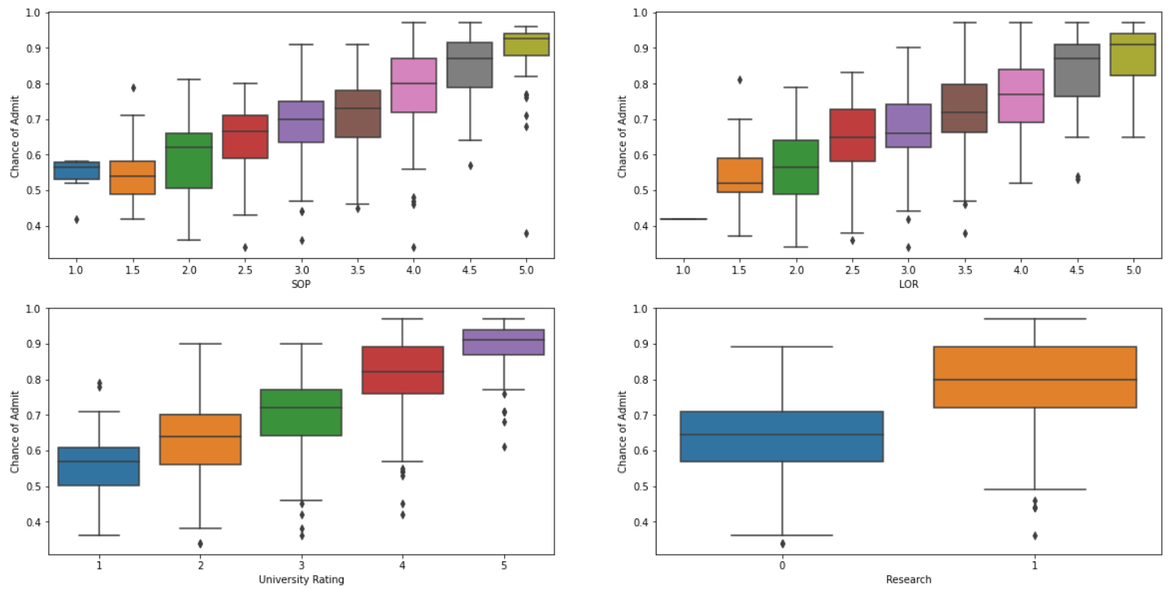
In [54]:

```
val_col = ['GRE Score', 'TOEFL Score', 'CGPA',"University Rating",'SOP']

sns.pairplot(data, x_vars=val_col, y_vars=["Chance of Admit"],
             hue="Research", height=5, aspect=.8, kind="reg");
```



In [55]:

```
plt.figure(figsize=(20,10))
cat_col = ['SOP', 'LOR', 'University Rating', 'Research']

for i in range(len(cat_col)):
    plt.subplot(2,2,i+1)
    sns.boxplot(x=cat_col[i], y='Chance of Admit', data = data)

plt.show()
```

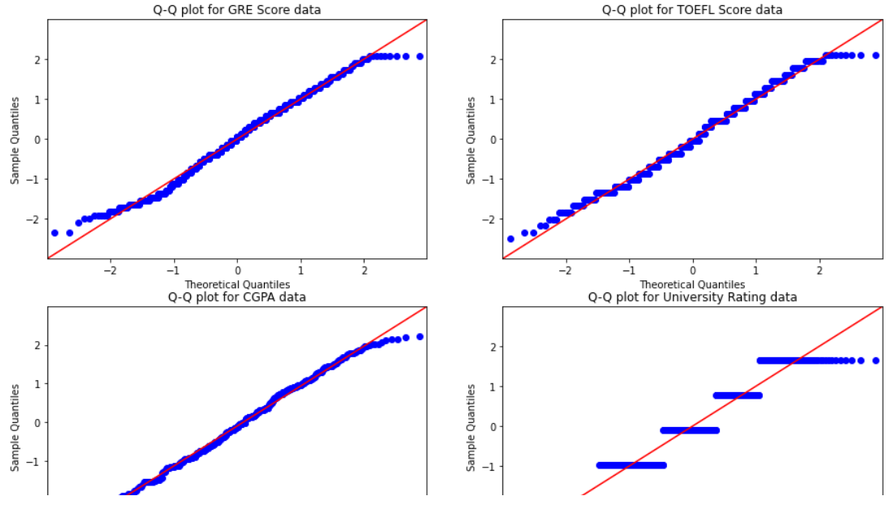


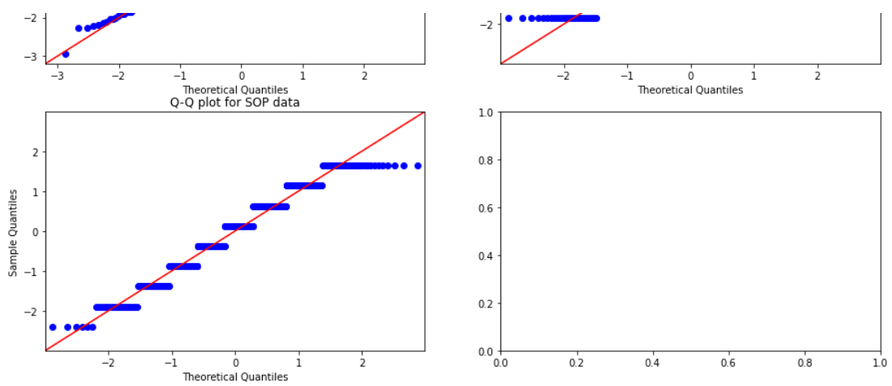
- **It can be infered from the above graphs that there is strong positive relationship exits between Chance of Admit and numberical values - GRE Score, TOEFL score, CGPA.**
- **As the GRE Score or TOEFL score or CGPA increases So there is High Chance of Admit.**
- **Applicants who opts for research and higher GRE Score or TOEFL score or CGPA tends to have higher chance of admit.**
- **As Statement of Purpose and Letter of Recommendation Strength or University Rating increases So there is High Chance of Admit.**
- **Statement of Purpose and Letter of Recommendation Strength or Universities with higher rating have more Research opted applicants tends to have higher chance of admit**

In [55]:

In [56]:

```
import statsmodels.api as sm
fig, axs = plt.subplots(3, 2,figsize=(15, 15))
sm.qqplot(data['GRE Score'],line="45",fit=True,ax=axs[0,0])
axs[0,0].set_title("Q-Q plot for GRE Score data")
sm.qqplot(data['TOEFL Score'],line="45",fit=True,ax=axs[0,1])
axs[0,1].set_title("Q-Q plot for TOEFL Score data")
sm.qqplot(data['CGPA'],line="45",fit=True,ax=axs[1,0])
axs[1,0].set_title("Q-Q plot for CGPA data")
sm.qqplot(data['University Rating'],line="45",fit=True,ax=axs[1,1])
axs[1,1].set_title("Q-Q plot for University Rating data")
sm.qqplot(data['SOP'],line="45",fit=True,ax=axs[2,0])
axs[2,0].set_title("Q-Q plot for SOP data")
plt.show()
```





In [194]:

```
from scipy.stats import kstest
for i in ['GRE Score', 'TOEFL Score', 'CGPA',"University Rating",'SOP']:
  tstat,pvalue=kstest(data[i], 'norm')

  print("##"*50)
  print(tstat,pvalue)
  if pvalue>0.05:
    print("Accept null hypothesis ")
    print(f"Distribution of the given Column {i} data is flollws Gaussian Distribution")
  else:
    print("Reject null Hypothesis",end='--->')
    print(f"Distribution of the given Column {i} data is not Gaussian Distribution")
```

```
####################################################################################################
1.0 0.0
Reject null Hypothesis--->Distribution of the given Column GRE Score data is not Gaussian Distribution
####################################################################################################
1.0 0.0
Reject null Hypothesis--->Distribution of the given Column TOEFL Score data is not Gaussian Distribution
####################################################################################################
0.9999999999947691 0.0
Reject null Hypothesis--->Distribution of the given Column CGPA data is not Gaussian Distribution
####################################################################################################
0.9092498680518208 0.0
Reject null Hypothesis--->Distribution of the given Column University Rating data is not Gaussian Distribution
####################################################################################################
0.9211927987311419 0.0
Reject null Hypothesis--->Distribution of the given Column SOP data is not Gaussian Distribution
```

In [200]:

```
```

In [203]:

```
significance_value=0.05
df_rating_grp = data[['University Rating', 'Chance of Admit']].groupby(['University Rating'])
ftest_statistic, p_value = stats.f_oneway(df_rating_grp.get_group(1)['Chance of Admit'],
                          df_rating_grp.get_group(2)['Chance of Admit'],
                          df_rating_grp.get_group(3)['Chance of Admit'],
                          df_rating_grp.get_group(4)['Chance of Admit'],
                          df_rating_grp.get_group(5)['Chance of Admit'])
null_Hypothesis=f"H0: mean Chance of Admit with University Rating are same"
alternative_Hypothesis="HA: mean Chance of Admit with University Rating are not same or at least one of them are different "

print(f"ftest statistic : {round(ftest_statistic,3)} | p_value : {p_value}")
if p_value>significance_value:
  print(f"Failed to Reject ---> {null_Hypothesis}")
else:
  print(f"Reject ---> {null_Hypothesis}")
  print(f"Accept ---> {alternative_Hypothesis}")
print('-'*100)
############################
df_sop_grp = data[['SOP', 'Chance of Admit']].groupby(['SOP'])
ftest_statistic, p_value = stats.f_oneway(df_sop_grp.get_group(1.0)['Chance of Admit'],
                          df_sop_grp.get_group(1.5)['Chance of Admit'],
                          df_sop_grp.get_group(2.0)['Chance of Admit'],
                          df_sop_grp.get_group(2.5)['Chance of Admit'],
                          df_sop_grp.get_group(3.0)['Chance of Admit'],
                          df_sop_grp.get_group(3.5)['Chance of Admit'],
                          df_sop_grp.get_group(4.0)['Chance of Admit'],
                          df_sop_grp.get_group(4.5)['Chance of Admit'],
                          df_sop_grp.get_group(5.0)['Chance of Admit'])

null_Hypothesis=f"H0: mean Chance of Admit with SOP are same"
alternative_Hypothesis="HA: mean Chance of Admit with SOP are not same or at least one of them are different "

print(f"ftest statistic : {round(ftest_statistic,3)} | p_value : {p_value}")
if p_value>significance_value:
  print(f"Failed to Reject ---> {null_Hypothesis}")
else:
  print(f"Reject ---> {null_Hypothesis}")
  print(f"Accept ---> {alternative_Hypothesis}")
print('-'*100)

########################################
df_lor_grp = data[['LOR', 'Chance of Admit']].groupby('LOR')

ftest_statistic, p_value = stats.f_oneway(df_lor_grp.get_group(1.0)['Chance of Admit'],
                          df_lor_grp.get_group(1.5)['Chance of Admit'],
                          df_lor_grp.get_group(2.0)['Chance of Admit'],
```

```
                        df_lor_grp.get_group(2.5)['Chance of Admit'],
                        df_lor_grp.get_group(3.0)['Chance of Admit'],
                        df_lor_grp.get_group(3.5)['Chance of Admit'],
                        df_lor_grp.get_group(4.0)['Chance of Admit'],
                        df_lor_grp.get_group(4.5)['Chance of Admit'],
                        df_lor_grp.get_group(5.0)['Chance of Admit'])
null_Hypothesis=f"H0: mean Chance of Admit with LOR are same"
alternative_Hypothesis="HA: mean Chance of Admit with LOR are not same or at least one of them are different "

print(f"ftest statistic : {round(ftest_statistic,3)} | p_value : {p_value}")
if p_value>significance_value:
  print(f"Failed to Reject ---> {null_Hypothesis}")
else:
  print(f"Reject ---> {null_Hypothesis}")
  print(f"Accept ---> {alternative_Hypothesis}")
print('-'*100)

###################################
df_res_grp = data[['Research', 'Chance of Admit']].groupby('Research')
ftest_statistic, p_value = stats.f_oneway(df_res_grp.get_group(1)['Chance of Admit'],
                         df_res_grp.get_group(0)['Chance of Admit'])
null_Hypothesis=f"H0: mean Chance of Admit with Research are same"
alternative_Hypothesis="HA: mean Chance of Admit with Research are not same or at least one of them are different "

print(f"ftest statistic : {round(ftest_statistic,3)} | p_value : {p_value}")
if p_value>significance_value:
  print(f"Failed to Reject ---> {null_Hypothesis}")
else:
  print(f"Reject ---> {null_Hypothesis}")
  print(f"Accept ---> {alternative_Hypothesis}")
```

```
ftest statistic : 114.008 | p_value : 7.753395328023128e-69
Reject ---> H0: mean Chance of Admit with University Rating are same
Accept ---> HA: mean Chance of Admit with University Rating are not same or at least one of them are different
----------------------------------------------------------------------------------------------------
ftest statistic : 55.974 | p_value : 2.180473578270692e-64
Reject ---> H0: mean Chance of Admit with SOP are same
Accept ---> HA: mean Chance of Admit with SOP are not same or at least one of them are different
----------------------------------------------------------------------------------------------------
ftest statistic : 44.55 | p_value : 1.2475714296300813e-53
Reject ---> H0: mean Chance of Admit with LOR are same
Accept ---> HA: mean Chance of Admit with LOR are not same or at least one of them are different
----------------------------------------------------------------------------------------------------
ftest statistic : 211.377 | p_value : 3.5954935458406797e-40
Reject ---> H0: mean Chance of Admit with Research are same
Accept ---> HA: mean Chance of Admit with Research are not same or at least one of them are different
```

- **ANOVA shows a very strong association between University Rating,SOP,LOR,Research and Chance of Admit.Mean Chance of Admit with [University Rating,SOP,LOR,Research] are not same or at least one of them are different ,that infers there might be some effect on prediction/target by these features**

In [58]:

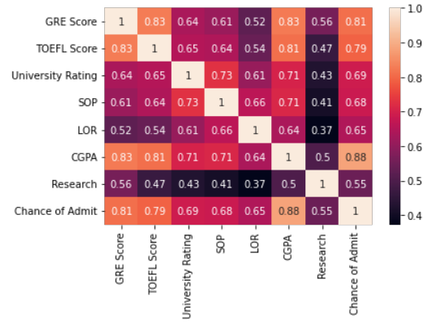
Multivariate Analsyis :

In [59]:

```
sns.heatmap(data.iloc[::,1:].corr(method="pearson"),annot=True)
```

Out[59]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbd7b043ca0>
```

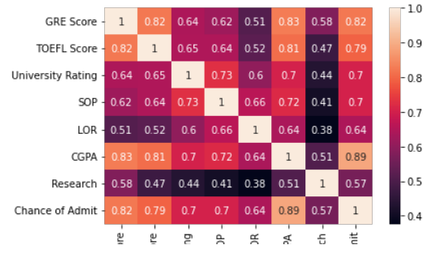


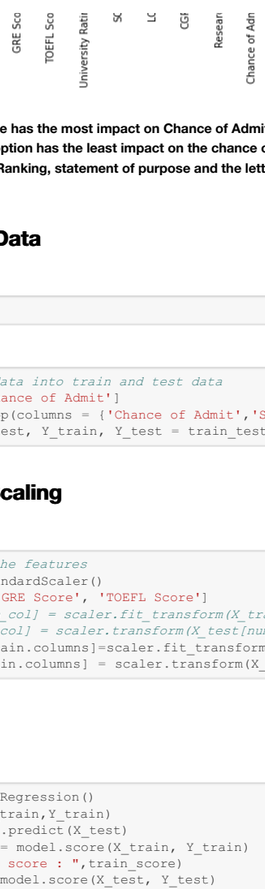
In [60]:

```
sns.heatmap(data.iloc[::,1:].corr(method="spearman"),annot=True)
```

Out[60]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbd7b1d0820>
```





- **CGPA Score has the most impact on Chance of Admit. It is followed by GRE and TOFEL Score.**
- **Research option has the least impact on the chance of admit.**
- **University Ranking, statement of purpose and the letter of recommendation are also having impact on the chance of admit.**

Splitting Data

In [180]:

In [181]:

```
#split our data into train and test data
Y = data['Chance of Admit']
X = data.drop(columns = {'Chance of Admit','Serial No.'})
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,random_state=42, shuffle=True)
```

Feature Scaling

In [182]:

```
#rescaling the features
scaler = StandardScaler()
num_col = ['GRE Score', 'TOEFL Score']
#X_train[num_col] = scaler.fit_transform(X_train[num_col])
#X_test[num_col] = scaler.transform(X_test[num_col])
X_train[X_train.columns]=scaler.fit_transform(X_train)
X_test[X_train.columns] = scaler.transform(X_test)
```

Model

In [147]:

```
model=LinearRegression()
model.fit(X_train,Y_train)
y_pred=model.predict(X_test)
train_score = model.score(X_train, Y_train)
print("train score : ",train_score)
val_score = model.score(X_test, Y_test)
print("Test score : ",val_score)
```

```
train score :  0.8210671369321554
Test score :  0.8188432567829628
```

In [142]:

Assumptions Check

In [91]:

Linearity

This assumes that there is a linear relationship between the predictors (e.g. independent variables or features) and the response variable (e.g. dependent variable or label). This also assumes that the predictors are additive.

`Why it can happen:` **There may not just be a linear relationship among the data. Modeling is about trying to estimate a function that explains a process, and linear regression would not be a fitting estimator (pun intended) if there is no linear relationship.**

`What it will affect:`

The predictions will be extremely inaccurate because our model is underfitting. This is a serious violation that should not be ignored.

`How to detect it:`

If there is only one predictor, this is pretty easy to test with a scatter plot. Most cases aren't so simple, so we'll have to modify this by using a scatter plot to see our predicted values versus the actual values (in other words, view the residuals). Ideally, the points should lie on or around a diagonal line on the scatter plot.

`How to fix it:`

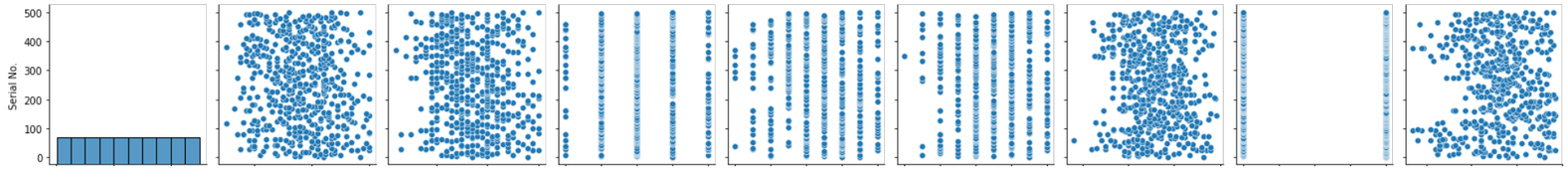
Either adding polynomial terms to some of the predictors or applying nonlinear transformations . If those do not work, try adding additional variables to help capture the relationship between the predictors and the label.

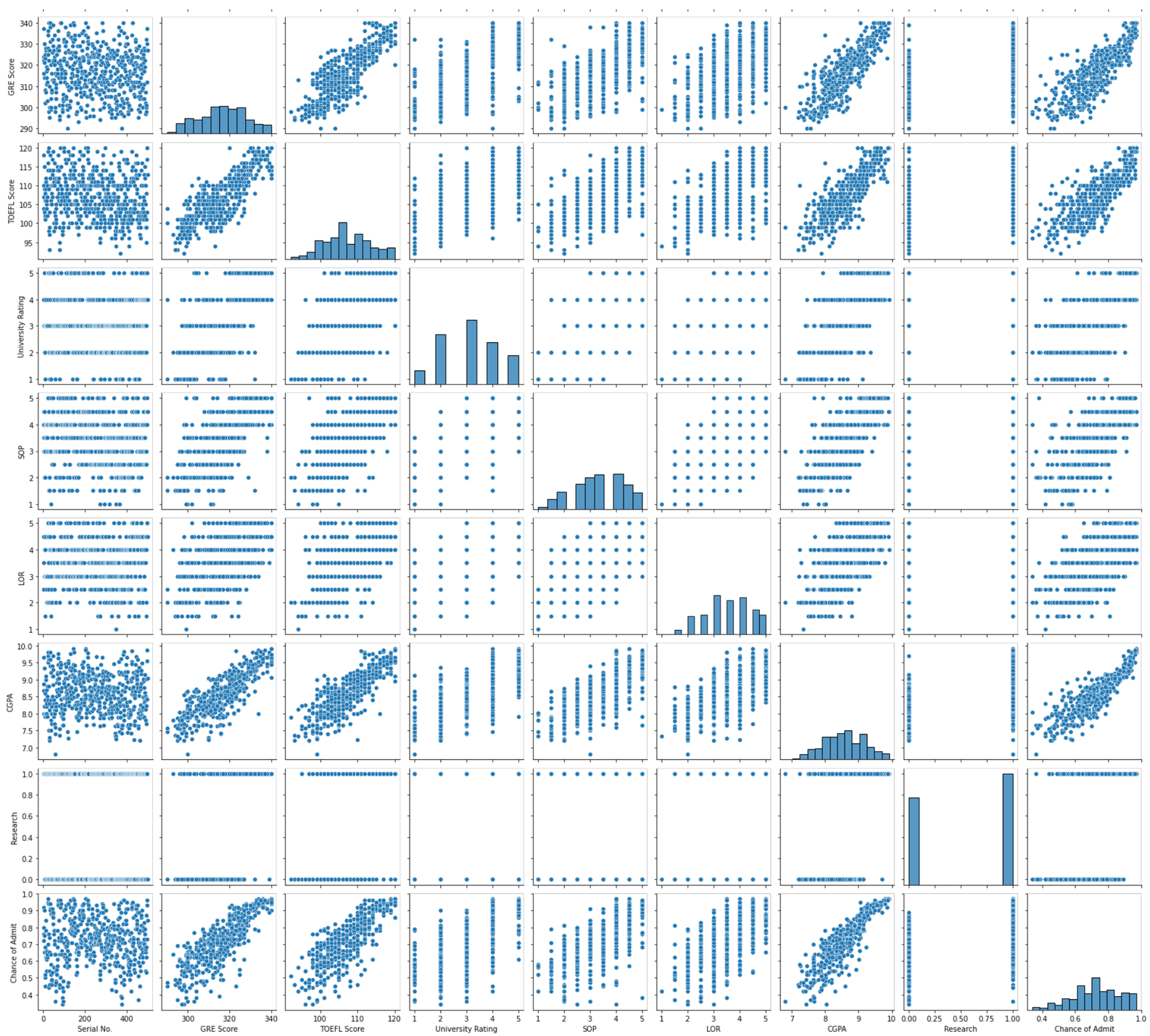
In [117]:

```
sns.pairplot(data)
```

Out[117]:

```
<seaborn.axisgrid.PairGrid at 0x7fbd775a8b80>
```





In [125]:

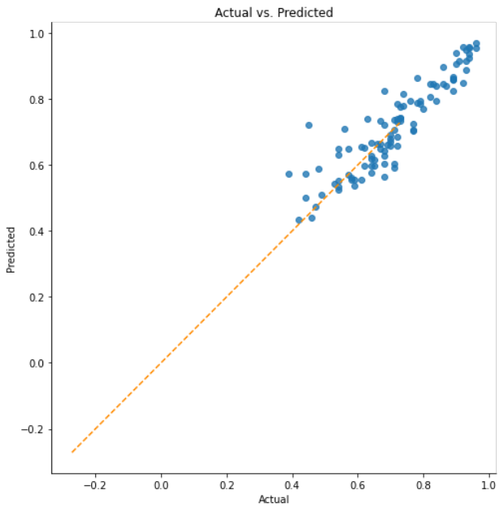
```
def calculate_residuals(model, features, label):
    """
    Creates predictions on the features with the model and calculates residuals
    """
    predictions = model.predict(features)
    df_results = pd.DataFrame({'Actual': label, 'Predicted': predictions})
    df_results['Residuals'] = abs(df_results['Actual']) - abs(df_results['Predicted'])

    return df_results
def linear_assumption(model, features, label):
    """
    Linearity: Assumes that there is a linear relationship between the predictors and
               the response variable. If not, either a quadratic term or another
               algorithm should be used.
    """
    print('Assumption 1: Linear Relationship between the Target and the Feature', '\n')

    print('Checking with a scatter plot of actual vs. predicted.',
          'Predictions should follow the diagonal line.')
```

```
    # Calculating residuals for the plot
    df_results = calculate_residuals(model, features, label)

    # Plotting the actual vs predicted values
    sns.lmplot(x='Actual', y='Predicted', data=df_results, fit_reg=False, size=7)
    # Plotting the diagonal line

    line_coords = np.arange(df_results.min().min(), df_results.max().max())
    print(line_coords)
    plt.plot(line_coords, line_coords,  # X and y points
             color='darkorange', linestyle='--')
    plt.title('Actual vs. Predicted')
    plt.show()
```

In [126]:

```
linear_assumption(model, X_test, Y_test)
```

```
Assumption 1: Linear Relationship between the Target and the Feature

Checking with a scatter plot of actual vs. predicted. Predictions should follow the diagonal line.
[-0.27193204  0.72806796]
```



- **We can see in this case that there is almost a perfect linear relationship. Our predictions are biased towards Higher values compared to lower values and especially at the higher values (above 0.8).when it came to lower values variance slightly increases**

Normality of the Error Terms

In [127]:

```
def normal_errors_assumption(model, features, label, p_value_thresh=0.05):
    """
    Normality: Assumes that the error terms are normally distributed. If they are not,
    nonlinear transformations of variables may solve this.

    This assumption being violated primarily causes issues with the confidence intervals
    """
    from statsmodels.stats.diagnostic import normal_ad
    print('Assumption 2: The error terms are normally distributed', '\n')

    # Calculating residuals for the Anderson-Darling test
    df_results = calculate_residuals(model, features, label)

    print('Using the Anderson-Darling test for normal distribution')

    # Performing the test on the residuals
    p_value = normal_ad(df_results['Residuals'])[1]
    print('p-value from the test - below 0.05 generally means non-normal:', p_value)

    # Reporting the normality of the residuals
    if p_value < p_value_thresh:
        print('Residuals are not normally distributed')
    else:
        print('Residuals are normally distributed')

    # Plotting the residuals distribution
    plt.subplots(figsize=(12, 6))
    plt.title('Distribution of Residuals')
    sns.distplot(df_results['Residuals'])
    plt.show()

    print()
    if p_value > p_value_thresh:
        print('Assumption satisfied')
    else:
        print('Assumption not satisfied')
        print()
        print('Confidence intervals will likely be affected')
        print('Try performing nonlinear transformations on variables')
```

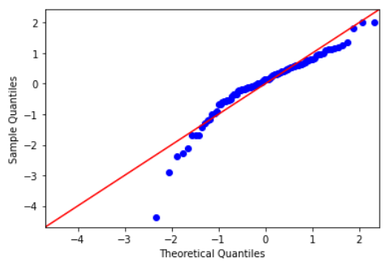
In [128]:

```
y_train_pred=model.predict(X_test)
```

```
Error=Y_test-y_train_pred
```

In [129]:

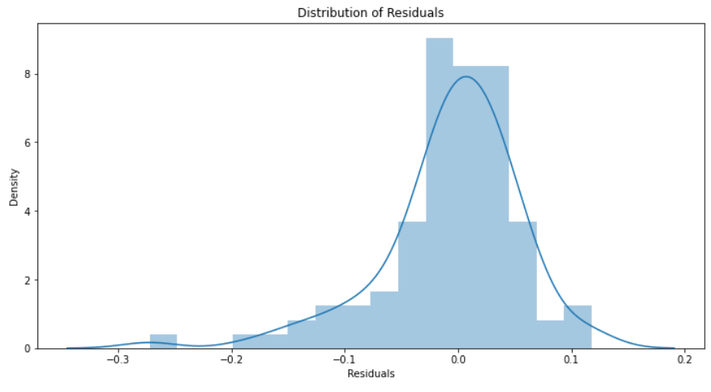
```
import statsmodels.api as sm
data_points=(Error-Error.mean())/Error.std()
sm.qqplot(data_points, line ='45')
plt.show()
```



In [130]:

```
normal_errors_assumption(model, X_test, Y_test)
```

```
Assumption 2: The error terms are normally distributed

Using the Anderson-Darling test for normal distribution
p-value from the test - below 0.05 generally means non-normal: 2.128546023452642e-05
Residuals are not normally distributed
```




```
Assumption not satisfied

Confidence intervals will likely be affected
Try performing nonlinear transformations on variables
```

Multicollinearity Check - VIF Score

In [96]:

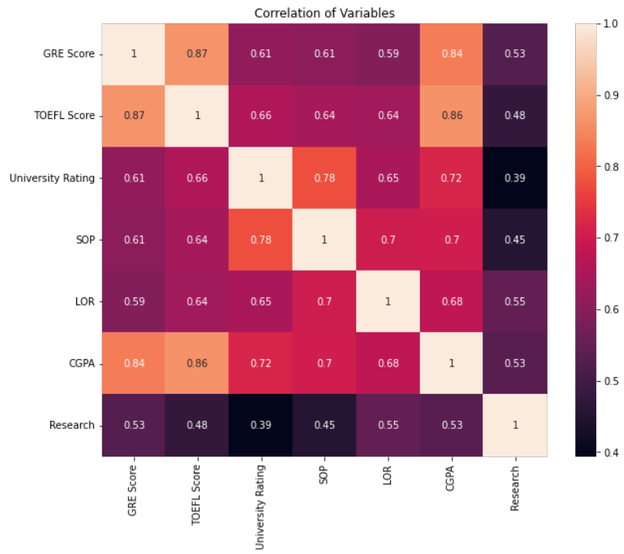
```
def multicollinearity_assumption(model, features, label, feature_names=None):
    """
    Multicollinearity: Assumes that predictors are not correlated with each other. If there is
                       correlation among the predictors, then either remove prepdictors with high
                       Variance Inflation Factor (VIF) values or perform dimensionality reduction

                       This assumption being violated causes issues with interpretability of the
                       coefficients and the standard errors of the coefficients.
    """
    from statsmodels.stats.outliers_influence import variance_inflation_factor
    print('Assumption 3: Little to no multicollinearity among predictors')

    # Plotting the heatmap
    plt.figure(figsize = (10,8))
    sns.heatmap(pd.DataFrame(features, columns=feature_names).corr(), annot=True)
    plt.title('Correlation of Variables')
    plt.show()

    print('Variance Inflation Factors (VIF)')
    print('> 10: An indication that multicollinearity may be present')
    print('> 100: Certain multicollinearity among the variables')
    print('-------------------------------------')

    # Gathering the VIF for each variable

    VIF = [variance_inflation_factor(features.values, i) for i in range(features.shape[1])]
    for idx, vif in enumerate(VIF):
        print('{0}: {1}'.format(feature_names[idx], vif))

    # Gathering and printing total cases of possible or definite multicollinearity
    possible_multicollinearity = sum([1 for vif in VIF if vif > 10])
    definite_multicollinearity = sum([1 for vif in VIF if vif > 100])
    print()
    print('{0} cases of possible multicollinearity'.format(possible_multicollinearity))
    print('{0} cases of definite multicollinearity'.format(definite_multicollinearity))
    print()
```

```
    if definite_multicollinearity == 0:
        if possible_multicollinearity == 0:
            print('Assumption satisfied')
        else:
            print('Assumption possibly satisfied')
            print()
            print('Coefficient interpretability may be problematic')
            print('Consider removing variables with a high Variance Inflation Factor (VIF)')

    else:
        print('Assumption not satisfied')
        print()
        print('Coefficient interpretability will be problematic')
        print('Consider removing variables with a high Variance Inflation Factor (VIF)')
```

In [97]:

```
multicollinearity_assumption(model, X_test, Y_test, X_train.columns)
```

```
Assumption 3: Little to no multicollinearity among predictors
```




```
Variance Inflation Factors (VIF)
> 10: An indication that multicollinearity may be present
> 100: Certain multicollinearity among the variables
--------------------------------------
GRE Score: 4.75425019843904
TOEFL Score: 5.5426431576414315
University Rating: 3.025018162281418
SOP: 3.1491573534726953
LOR: 2.525229116174667
CGPA: 5.630142922365661
Research: 1.6078932764906435

0 cases of possible multicollinearity
0 cases of definite multicollinearity

Assumption satisfied
```

No Autocorrelation of the Error TermsPermalink

In [102]:

```
def autocorrelation_assumption(model, features, label):
    """
    Autocorrelation: Assumes that there is no autocorrelation in the residuals. If there is
                     autocorrelation, then there is a pattern that is not explained due to
                     the current value being dependent on the previous value.
                     This may be resolved by adding a lag variable of either the dependent
                     variable or some of the predictors.
    """
    from statsmodels.stats.stattools import durbin_watson
    print('Assumption 4: No Autocorrelation', '\n')

    # Calculating residuals for the Durbin Watson-tests
    df_results = calculate_residuals(model, features, label)

    print('\nPerforming Durbin-Watson Test')
    print('Values of 1.5 < d < 2.5 generally show that there is no autocorrelation in the data')
    print('0 to 2< is positive autocorrelation')
    print('>2 to 4 is negative autocorrelation')
    print('-------------------------------------')
    durbinWatson = durbin_watson(df_results['Residuals'])
    print('Durbin-Watson:', durbinWatson)
    if durbinWatson < 1.5:
        print('Signs of positive autocorrelation', '\n')
        print('Assumption not satisfied')
    elif durbinWatson > 2.5:
        print('Signs of negative autocorrelation', '\n')
        print('Assumption not satisfied')
    else:
        print('Little to no autocorrelation', '\n')
        print('Assumption satisfied')
```

In [103]:

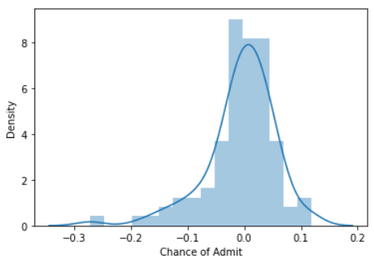
```
autocorrelation_assumption(model, X_test, Y_test)
```

```
Assumption 4: No Autocorrelation


Performing Durbin-Watson Test
Values of 1.5 < d < 2.5 generally show that there is no autocorrelation in the data
0 to 2< is positive autocorrelation
>2 to 4 is negative autocorrelation
-------------------------------------
Durbin-Watson: 2.23879966743785
Little to no autocorrelation

Assumption satisfied
```

Mean of Residuals

In [136]:

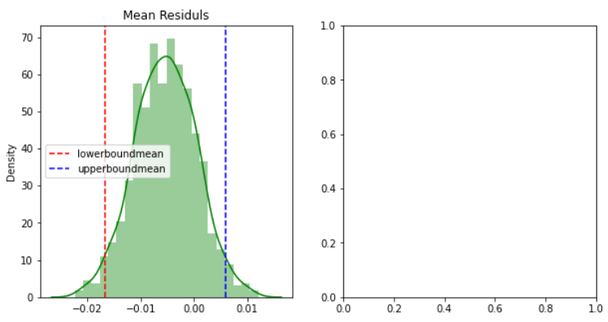
```
residual = Y_test - y_pred
sns.distplot(residual)
plt.show()
```



In [140]:

```
residual_means=[]
for i in range(1,1001):
    residual_means.append(np.random.choice(residual,size=100).mean())
print(f"mean of Residuls 95 CI   :   {np.percentile(residual_means,[2.5,97.5])}")


fig, axs = plt.subplots(1, 2,figsize=(10, 5))
sns.distplot(residual_means,color="g",ax=axs[0])
axs[0].axvline(np.percentile(residual_means,2.5),linestyle="--", color='r', label="lowerboundmean")
axs[0].axvline(np.percentile(residual_means,97.5),linestyle="--", color='b', label="upperboundmean")
axs[0].legend()
axs[0].set_title("Mean Residuls")
plt.show()
```

```
mean of Residuls 95 CI   :   [-0.01671074  0.0058726 ]
```



In [ ]:

```

```

- **The Mean CI value of Ressiduals is [-0.01671074 0.0058726] which is fairly close to Zero.It referes to model is performing well**

Linearity of Variables or NO Heteroskedasticity

In [108]:

```
def homoscedasticity_assumption(model, features, label):
    """
    Homoscedasticity: Assumes that the errors exhibit constant variance
    """
    print('Assumption 5: Homoscedasticity of Error Terms and No Heteroskedasticity', '\n')

    print('Residuals should have relative constant variance')

    # Calculating residuals for the plot
    df_results = calculate_residuals(model, features, label)

    # Plotting the residuals
```

```
    plt.subplots(figsize=(12, 6))
    ax = plt.subplot(111)  # To remove spines
    plt.scatter(x=df_results.index, y=df_results.Residuals, alpha=0.5)
    plt.plot(np.repeat(0, df_results.index.max()), color='darkorange', linestyle='--')
    ax.spines['right'].set_visible(False)  # Removing the right spine
    ax.spines['top'].set_visible(False)  # Removing the top spine
    plt.title('Residuals')
    plt.show()
```

In [109]:

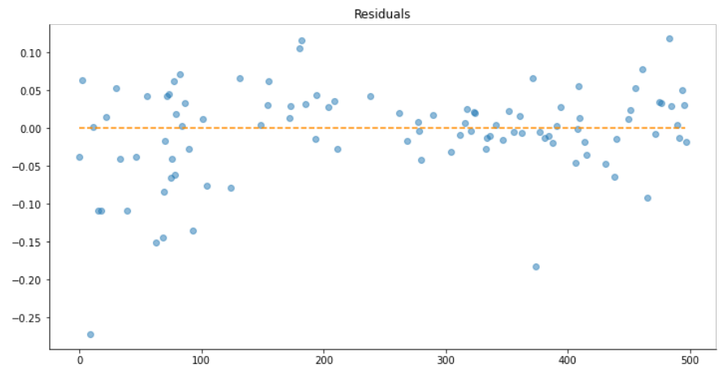
```
```

In [110]:

```
homoscedasticity_assumption(model, X_test, Y_test)
```

Assumption 5: Homoscedasticity of Error Terms and No Heteroskedasticity

Residuals should have relative constant variance



In [190]:

```
```

In [172]:

```
data.shape
```

Out[172]:

```
(500, 9)
```

In [174]:

```
list(data.columns)[1:]
```

Out[174]:

```
['GRE Score',
 'TOEFL Score',
 'University Rating',
 'SOP',
 'LOR',
 'CGPA',
 'Research',
 'Chance of Admit']
```

In [184]:

```
data_copy=data.iloc[:,1:].copy(deep=True)
print(data.columns)
data_copy.columns=["GRE","TOFEL","UR","SOP","LOR","CGPA","CA","R1"]
mlr = smf.ols(formula="CA~GRE+TOFEL+UR+SOP+LOR+CGPA+R1", data=data_copy).fit()
ivar = data_copy.drop("CA",axis=1)
ivarc = smt.add_constant(data=ivar, prepend=True)
print(ivarc.head())
```

```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR', 'CGPA', 'Research', 'Chance of Admit'],
      dtype='object')
   const  GRE  TOFEL  UR  SOP  LOR  CGPA    R1
0    1.0  337    118   4  4.5  4.5  9.65  0.92
1    1.0  324    107   4  4.0  4.5  8.87  0.76
2    1.0  316    104   3  3.0  3.5  8.00  0.72
3    1.0  322    110   3  3.5  2.5  8.67  0.80
4    1.0  314    103   2  2.0  3.0  8.21  0.65
```

Null Hypothesis (H0): Homoscedasticity is present (residuals are equally scattered)

Alternative Hypothesis(HA): Heteroscedasticity is present (residuals are not equally scattered)

In [189]:

```
testres=smd.het_breuschpagan(resid=mlr.resid, exog_het=ivarc)
print("lm:", testres[0], "lm_pvalue:", testres[1])

print("##"*50)
#print(tstat,pvalue)
if  testres[1]<0.05:
  print(f"The data follows Heteroscedasticity and not Homoscedasticity")
else:
  print(f"The data deos not follow Heteroscedasticity")
```

```
lm: 37.48642435269656 lm_pvalue: 3.7932626582322196e-06
###############################################################################################
The data follows Heteroscedasticity and not Homoscedasticity
```

In [190]:

```
wtest = smd.het_white(resid=mlr.resid, exog=ivarc)
print("lm:", wtest[0], "lm_pvalue:", wtest[1])
print("##"*50)
#print(tstat,pvalue)
if  wtest[1]<0.05:
  print(f"The data follows Heteroscedasticity and not Homoscedasticity")
else:
  print(f"Variance is not constant")
```

```
lm: 110.57310308954843 lm_pvalue: 9.108251669811503e-10
###############################################################################################
The data follows Heteroscedasticity and not Homoscedasticity
```

- **By the spread of data in the graph, data have the problem of heteroscedasticity slightly.Why?**
- **Refer: Linear Assumption check : We can see in this case that there is almost a perfect linear relationship. Our predictions are biased towards Higher values compared to lower values and especially at the higher values (above 0.8).when it came to lower values variance slightly increases**
- **Since the p-value is less than 0.05, we fail to reject the null hypothesis.**

This means we do not have sufficient evidence to say that heteroscedasticity is present in the regression model.

Retrain After removing High Variance features

In [148]:

```
#keeping x_train for modelling
x_train_Copy = X_train.copy(deep=True)

#adding constant column
x_train_1 = sm.add_constant(x_train_Copy)

lr = sm.OLS(Y_train, x_train_1).fit()

lr.summary()
```

Out[148]:

OLS Regression Results

Dep. Variable:	Chance of Admit	**R-squared:**	0.821
Model:	OLS	**Adj. R-squared:**	0.818
Method:	Least Squares	**F-statistic:**	257.0
Date:	Thu, 19 Jan 2023	**Prob (F-statistic):**	3.41e-142
Time:	15:51:09	**Log-Likelihood:**	561.91
No. Observations:	400	**AIC:**	-1108.
Df Residuals:	392	**BIC:**	-1076.
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.7242	0.003	241.441	0.000	0.718	0.730
GRE Score	0.0267	0.006	4.196	0.000	0.014	0.039
TOEFL Score	0.0182	0.006	3.174	0.002	0.007	0.030
University Rating	0.0029	0.005	0.611	0.541	-0.007	0.012
SOP	0.0018	0.005	0.357	0.721	-0.008	0.012
LOR	0.0159	0.004	3.761	0.000	0.008	0.024
CGPA	0.0676	0.006	10.444	0.000	0.055	0.080
Research	0.0119	0.004	3.231	0.001	0.005	0.019

Omnibus:	86.232	**Durbin-Watson:**	2.050
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	190.099
Skew:	-1.107	**Prob(JB):**	5.25e-42
Kurtosis:	5.551	**Cond. No.**	5.65

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

In [150]:

```
x_train_Copy.columns
```

Out[150]:

```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR', 'CGPA',
       'Research'],
      dtype='object')
```

`Inferences`

Based on the P-values, the following features were removed

- **University Rating**
- **SOP**

In [151]:

```
#adding constant column
```

```
x_train_1 = sm.add_constant(x_train_Copy[['GRE Score', 'TOEFL Score', 'LOR', 'CGPA',
       'Research']])

lr = sm.OLS(Y_train, x_train_1).fit()

lr.summary()
```

Out[151]:

OLS Regression Results

Dep. Variable:	Chance of Admit	**R-squared:**	0.821
Model:	OLS	**Adj. R-squared:**	0.818
Method:	Least Squares	**F-statistic:**	360.8
Date:	Thu, 19 Jan 2023	**Prob (F-statistic):**	1.36e-144
Time:	15:55:49	**Log-Likelihood:**	561.54
No. Observations:	400	**AIC:**	-1111.
Df Residuals:	394	**BIC:**	-1087.
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.7242	0.003	241.830	0.000	0.718	0.730
GRE Score	0.0269	0.006	4.245	0.000	0.014	0.039
TOEFL Score	0.0191	0.006	3.391	0.001	0.008	0.030
LOR	0.0172	0.004	4.465	0.000	0.010	0.025
CGPA	0.0691	0.006	11.147	0.000	0.057	0.081
Research	0.0122	0.004	3.328	0.001	0.005	0.019

Omnibus:	84.831	**Durbin-Watson:**	2.053
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	185.096
Skew:	-1.094	**Prob(JB):**	6.41e-41
Kurtosis:	5.514	**Cond. No.**	4.76

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

`Inferences`

Based on the P-values, the following features were are not able remove any feature. Even without these features we are able to achive same R2**

In [152]:

```
X_train=X_train[['GRE Score', 'TOEFL Score', 'LOR', 'CGPA',
       'Research']]
X_test=X_test[['GRE Score', 'TOEFL Score', 'LOR', 'CGPA',
       'Research']]
```

In [160]:

```
lasso_reg = Lasso(alpha=0.001)
lasso_reg.fit(X_train, Y_train)
y_pred=lasso_reg.predict(X_test)
train_score = lasso_reg.score(X_train, Y_train)
print("train score : ",train_score)
val_score = lasso_reg.score(X_test, Y_test)
print("Test score : ",val_score)

r2 = r2_score(Y_test,y_pred)
mae = mean_absolute_error(Y_test,y_pred)
mse = mean_squared_error( Y_test, y_pred )
rmse = np.sqrt( mean_squared_error( Y_test, y_pred ))
mape = mean_absolute_percentage_error(Y_test,y_pred)
Adjusted_R2_linear = 1 - (1-r2)*(len(Y_test)-1)/(len(Y_test)-X.shape[1]-1)


print('*'*100)
print('R-Square value:', round(r2,2))
print('Adjusted R-Square Value:', round(Adjusted_R2_linear,2))
print('Mean Absolute Error:', round(mae,2))
print('Mean Square Error:', round(mse,2))
print('Root Mean Square Error:', round(rmse,2))
print('Mean Absolute Percentage Error:', round(mape,2))
lasso_Coefficients = pd.DataFrame(lasso_reg.coef_, columns = ["lasso_reg_Coefficients"], index=X_train.columns)

lasso_Coefficients
```

```
train score :  0.8206504608642765
Test score :  0.8160912923353617
****************************************************************************************************
R-Square value: 0.82
Adjusted R-Square Value: 0.8
Mean Absolute Error: 0.04
Mean Square Error: 0.0
Root Mean Square Error: 0.06
Mean Absolute Percentage Error: 0.07
```

Out[160]:

	lasso_reg_Coefficients
GRE Score	0.026821
TOEFL Score	0.018718
LOR	0.016615
CGPA	0.069092

	lasso_reg_Coefficients
Research	0.011622

In [161]:

```
ridge_reg = Ridge(alpha=0.001)
ridge_reg.fit(X_train, Y_train)
y_pred=ridge_reg.predict(X_test)
train_score = ridge_reg.score(X_train, Y_train)
print("train score : ",train_score)
val_score = ridge_reg.score(X_test, Y_test)
print("Test score : ",val_score)

r2 = r2_score(Y_test,y_pred)
mae = mean_absolute_error(Y_test,y_pred)
mse = mean_squared_error( Y_test, y_pred )
rmse = np.sqrt( mean_squared_error( Y_test, y_pred ))
mape = mean_absolute_percentage_error(Y_test,y_pred)
Adjusted_R2_linear = 1 - (1-r2)*(len(Y_test)-1)/(len(Y_test)-X.shape[1]-1)


print('*'*100)
print('R-Square value:', round(r2,2))
print('Adjusted R-Square Value:', round(Adjusted_R2_linear,2))
print('Mean Absolute Error:', round(mae,2))
print('Mean Square Error:', round(mse,2))
print('Root Mean Square Error:', round(rmse,2))
print('Mean Absolute Percentage Error:', round(mape,2))

Ridge_Coefficients = pd.DataFrame(ridge_reg.coef_, columns = ["ridge_reg_Coefficients"], index=X_train.columns)
Ridge_Coefficients
```

```
train score :  0.8207326947481933
Test score :  0.8155000107921638
****************************************************************************************************
R-Square value: 0.82
Adjusted R-Square Value: 0.8
Mean Absolute Error: 0.04
Mean Square Error: 0.0
Root Mean Square Error: 0.06
Mean Absolute Percentage Error: 0.07
```

Out[161]:

	ridge_reg_Coefficients
GRE Score	0.026879
TOEFL Score	0.019106
LOR	0.017207
CGPA	0.069066
Research	0.012226

In [163]:

```

```

In [162]:

```
model=LinearRegression()
model.fit(X_train,Y_train)
y_pred=model.predict(X_test)
train_score = model.score(X_train, Y_train)
print("train score : ",train_score)
val_score = model.score(X_test, Y_test)
print("Test score : ",val_score)

r2 = r2_score(Y_test,y_pred)
mae = mean_absolute_error(Y_test,y_pred)
mse = mean_squared_error( Y_test, y_pred )
rmse = np.sqrt( mean_squared_error( Y_test, y_pred ))
mape = mean_absolute_percentage_error(Y_test,y_pred)
Adjusted_R2_linear = 1 - (1-r2)*(len(Y_test)-1)/(len(Y_test)-X.shape[1]-1)


print('*'*100)
print('R-Square value:', round(r2,2))
print('Adjusted R-Square Value:', round(Adjusted_R2_linear,2))
print('Mean Absolute Error:', round(mae,2))
print('Mean Square Error:', round(mse,2))
print('Root Mean Square Error:', round(rmse,2))
print('Mean Absolute Percentage Error:', round(mape,2))
linear_reg_Coefficients = pd.DataFrame(model.coef_, columns = ["linear_reg_Coefficients"], index=X_train.columns)
linear_reg_Coefficients
```

```
train score :  0.8207326947514394
Test score :  0.8155002070847488
****************************************************************************************************
R-Square value: 0.82
Adjusted R-Square Value: 0.8
Mean Absolute Error: 0.04
Mean Square Error: 0.0
Root Mean Square Error: 0.06
Mean Absolute Percentage Error: 0.07
```

Out[162]:

	linear_reg_Coefficients
GRE Score	0.026879
TOEFL Score	0.019106
LOR	0.017207
CGPA	0.069066
Research	0.012226

In [163]:

```
elastic_reg = ElasticNet(alpha=0.001)
elastic_reg.fit(X_train, Y_train)

y_pred=elastic_reg.predict(X_test)
train_score = elastic_reg.score(X_train, Y_train)
print("train score : ",train_score)
val_score = elastic_reg.score(X_test, Y_test)
print("Test score : ",val_score)
r2 = r2_score(Y_test,y_pred)

mae = mean_absolute_error(Y_test,y_pred)
mse = mean_squared_error( Y_test, y_pred )
rmse = np.sqrt( mean_squared_error( Y_test, y_pred ))
mape = mean_absolute_percentage_error(Y_test,y_pred)
Adjusted_R2_linear = 1 - (1-r2)*(len(Y_test)-1)/(len(Y_test)-X.shape[1]-1)

print('*'*100)
print('R-Square value:', round(r2,2))
print('Adjusted R-Square Value:', round(Adjusted_R2_linear,2))
print('Mean Absolute Error:', round(mae,2))
print('Mean Square Error:', round(mse,2))
print('Root Mean Square Error:', round(rmse,2))
print('Mean Absolute Percentage Error:', round(mape,2))
Elastic_Coefficients = pd.DataFrame(elastic_reg.coef_, columns = ["elastic_reg_Coefficients"], index=X_train.columns)
Elastic_Coefficients
```

```
train score :  0.8207112948831261
Test score :  0.8157817196133608
****************************************************************************************************
R-Square value: 0.82
Adjusted R-Square Value: 0.8
Mean Absolute Error: 0.04
Mean Square Error: 0.0
Root Mean Square Error: 0.06
Mean Absolute Percentage Error: 0.07
```

Out[163]:

	elastic_reg_Coefficients
GRE Score	0.026876
TOEFL Score	0.018943
LOR	0.016929
CGPA	0.068984
Research	0.011930

In [ ]:

In [156]:

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor

models = {"Linear Regression": LinearRegression(), 'Random Forest':RandomForestRegressor(),
          'KNN':KNeighborsRegressor(),'SVM':SVR(), 'GradientBoost':GradientBoostingRegressor()}
for model_name, model in models.items():
    predictor_model = model
    predictor_model.fit(X_train, Y_train)
    predictions = predictor_model.predict(X_test)
    print('*'*100)
    print(str(model_name) + ": "+ str(mean_squared_error(Y_test, predictions)))
    y_pred=model.predict(X_test)
    train_score = model.score(X_train, Y_train)
    print("train score : ",train_score)
    val_score = model.score(X_test, Y_test)
    print("Test score : ",val_score)
    print()
    r2 = r2_score(Y_test,y_pred)
    mae = mean_absolute_error(Y_test,y_pred)
    mse = mean_squared_error( Y_test, y_pred )
    rmse = np.sqrt( mean_squared_error( Y_test, y_pred ))
    mape = mean_absolute_percentage_error(Y_test,y_pred)
    Adjusted_R2_linear = 1 - (1-r2)*(len(Y_test)-1)/(len(Y_test)-X.shape[1]-1)



    print('R-Square value:', round(r2,2))
    print('Adjusted R-Square Value:', round(Adjusted_R2_linear,2))
    print('Mean Absolute Error:', round(mae,2))
    print('Mean Square Error:', round(mse,2))
    print('Root Mean Square Error:', round(rmse,2))
    print('Mean Absolute Percentage Error:', round(mape,2))
```

```
****************************************************************************************************
Linear Regression: 0.003773020765116889
train score :  0.8207326947514394
Test score :  0.8155002070847488

R-Square value: 0.82
Adjusted R-Square Value: 0.8
Mean Absolute Error: 0.04
Mean Square Error: 0.0
Root Mean Square Error: 0.06
Mean Absolute Percentage Error: 0.07
****************************************************************************************************
Random Forest: 0.004558991700000002
train score :  0.9686260050267471
Test score :  0.7770664205378972
```

```
R-Square value: 0.78
Adjusted R-Square Value: 0.76
Mean Absolute Error: 0.05
Mean Square Error: 0.0
Root Mean Square Error: 0.07
Mean Absolute Percentage Error: 0.07
****************************************************************************************************
KNN: 0.004961799999999999
train score :  0.8500625560682525
Test score :  0.7573691931540343

R-Square value: 0.76
Adjusted R-Square Value: 0.74
Mean Absolute Error: 0.05
Mean Square Error: 0.0
Root Mean Square Error: 0.07
Mean Absolute Percentage Error: 0.08
****************************************************************************************************
SVM: 0.004564393314681 22
train score :  0.7953871401668435
Test score :  0.7768022829006739

R-Square value: 0.78
Adjusted R-Square Value: 0.76
Mean Absolute Error: 0.05
Mean Square Error: 0.0
Root Mean Square Error: 0.07
Mean Absolute Percentage Error: 0.08
****************************************************************************************************
GradientBoost: 0.004585013803193969
train score :  0.9154715700490067
Test score :  0.7757939460540847

R-Square value: 0.78
Adjusted R-Square Value: 0.76
Mean Absolute Error: 0.05
Mean Square Error: 0.0
Root Mean Square Error: 0.07
Mean Absolute Percentage Error: 0.07
```

Conclusions & Recommendations

Inferences based on EDA

- **Based on the analysis we donot have outliers for independent features like 'GRE Score', 'TOEFL Score' & 'CGPA'.**
- **'Chance of Admit' is slightly left screwed. Since 'Chance of Admit' is a slightly left skewed.this means we have less data in that Area.**
  - **Due to skewness might be our models won't work in the perticular skewd area(Error distribution on that are variance slightly increases)**
- **Among students who have done research vs those who did not, 56 % said Yes and 44 % said No**
- **CGPA Score has the most impact on Chance of Admit. It is followed by GRE and TOFEL Score.**
- **Research option has the least impact on the chance of admit.**
- **University Ranking, statement of purpose and the letter of recommendation are also having impact on the chance of admit.**
- **It can be infered from the above graphs that there is strong positive relationship exits between Chance of Admit and numberical values - GRE Score, TOEFL score, CGPA.**
- **As the GRE Score or TOEFL score or CGPA increases So there is High Chance of Admit.**
- **Applicants who opts for research and higher GRE Score or TOEFL score or CGPA tends to have higher chance of admit.**
- **As Statement of Purpose and Letter of Recommendation Strength or University Rating increases So there is High Chance of Admit.**
- **Statement of Purpose and Letter of Recommendation Strength or Universities with higher rating have more Research opted applicants tends to have higher chance of admit**

Possible Model Improvement Areas

We have a couple of options:

- **Add new features GRE_TOEFL_CGPA_Ratio = (GRE & TOEFL Score & CGPA ratio) etc.**
- **Removing outliers or handling outlier by minmax distribution.**
- **By creating some proxy data on skwed area using SMOTE variations**
- **As we can see DT Ensemble models are working good but over fitting so can We finetune those models or we have to design new features(Feature Engineering) by connecting with some domain experts or designing our own features.To reduce variance or model drifting in production.**
- **Try performing nonlinear transformations on variables as we seen in Asumptions tests error not following Normal distribution from that we can infer we need to build some non-linear features**

Graduation Admission - Can use the above model to create new feature where students/learners can come to their website and check their probability of getting into the IVY league college. Key features which influence the chance of Admit are

```
CGPA
GRE Score
TOEFL Score
LOR
Research
```

From EDA we understood

- **A higher University rating will increases the chance of admission**
- **A higher value of LOR and SPO will also increases the chance of admission for the student.**

In [ ]: