https://drive.google.com/file/d/1wYc6djaXSgazLsHSQn6PtKl_in5mGRCk/view?usp=sharing
(https://drive.google.com/file/d/1wYc6djaXSgazLsHSQn6PtKl_in5mGRCk/view?usp=sharing)

In [452]:
```
!gdown 1wYc6djaXSgazLsHSQn6PtKl_in5mGRCk
```

```
'gdown' is not recognized as an internal or external command,
operable program or batch file.
```

In [128]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

df=pd.read_csv("Jamboree.csv")
df.head()
```

Out[128]:

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

In [413]:
```
df.shape
```

Out[413]: (500, 9)

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         500 non-null    int64  
 1   GRE Score          500 non-null    int64  
 2   TOEFL Score        500 non-null    int64  
 3   University Rating  500 non-null    int64  
 4   SOP                500 non-null    float64
 5   LOR                500 non-null    float64
 6   CGPA               500 non-null    float64
 7   Research           500 non-null    int64  
 8   Chance of Admit    500 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 35.3 KB
```

In [76]:
```
df["SOP"].value_counts()
```

...

In [129]:
```
df["University Rating"]=df["University Rating"].astype("category")
df["SOP"]=df["SOP"].astype("category")
df["LOR "]=df["LOR "].astype("category")
df["Research"]=df["Research"].astype("category")
```

In [236]:
```
df.columns
```

Out[236]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

In [4]:
```
df.isnull().sum().sum()
```

Out[4]:
```
0
```

There are no missing values found.

In [130]:
```
df.drop("Serial No.",inplace=True,axis=1)
```

In [131]:
```
df.describe(include="all")
```

Out[131]:

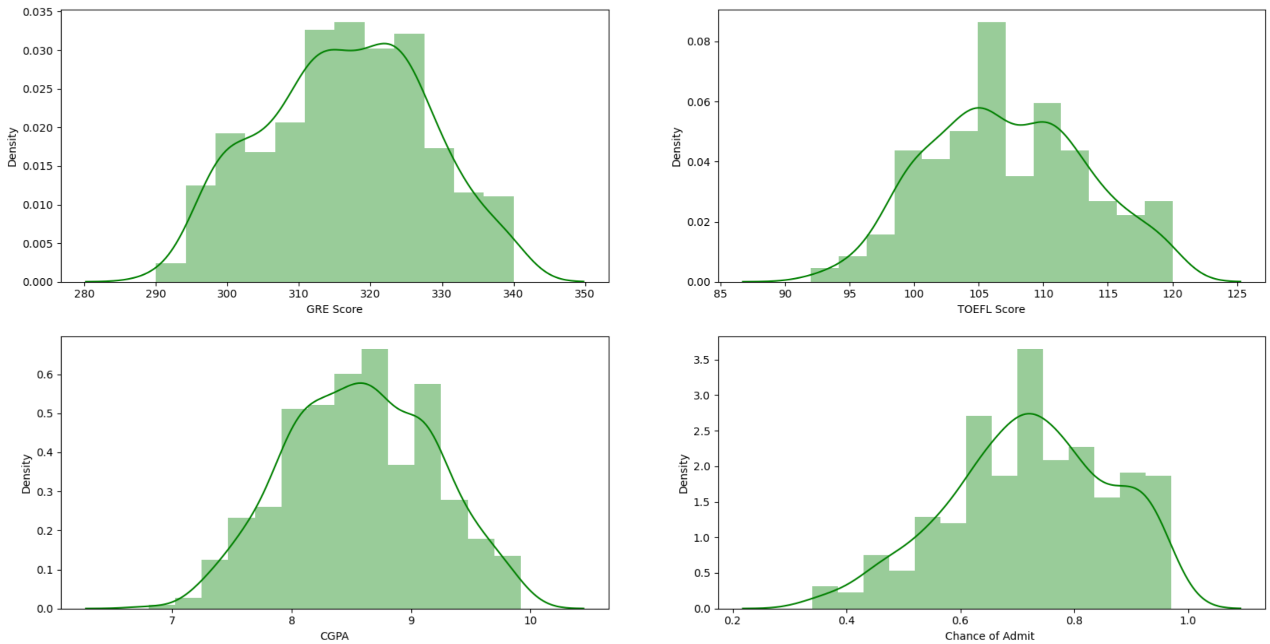
	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	500.000000	500.000000	500.0	500.0	500.0	500.000000	500.0	500.00000
unique	NaN	NaN	5.0	9.0	9.0	NaN	2.0	NaN
top	NaN	NaN	3.0	4.0	3.0	NaN	1.0	NaN
freq	NaN	NaN	162.0	89.0	99.0	NaN	280.0	NaN
mean	316.472000	107.192000	NaN	NaN	NaN	8.576440	NaN	0.72174
std	11.295148	6.081868	NaN	NaN	NaN	0.604813	NaN	0.14114
min	290.000000	92.000000	NaN	NaN	NaN	6.800000	NaN	0.34000
25%	308.000000	103.000000	NaN	NaN	NaN	8.127500	NaN	0.63000
50%	317.000000	107.000000	NaN	NaN	NaN	8.560000	NaN	0.72000
75%	325.000000	112.000000	NaN	NaN	NaN	9.040000	NaN	0.82000
max	340.000000	120.000000	NaN	NaN	NaN	9.920000	NaN	0.97000

In [424]:
```
df["LOR "].unique()
```

Out[424]:
```
[4.5, 3.5, 2.5, 3.0, 4.0, 1.5, 2.0, 5.0, 1.0]
Categories (9, float64): [1.0, 1.5, 2.0, 2.5, ..., 3.5, 4.0, 4.5, 5.0]
```

UNIVARIATE ANALYSIS

In [396]:
```
fig,axs=plt.subplots(nrows=2,ncols=2,figsize=(20,10))

cols=["GRE Score","TOEFL Score","CGPA","Chance of Admit "]
count=0

for i in range(2):
    for j in range(2):
            sns.distplot(df[cols[count]],ax=axs[i,j],color="g")
            count +=1

plt.show()
```

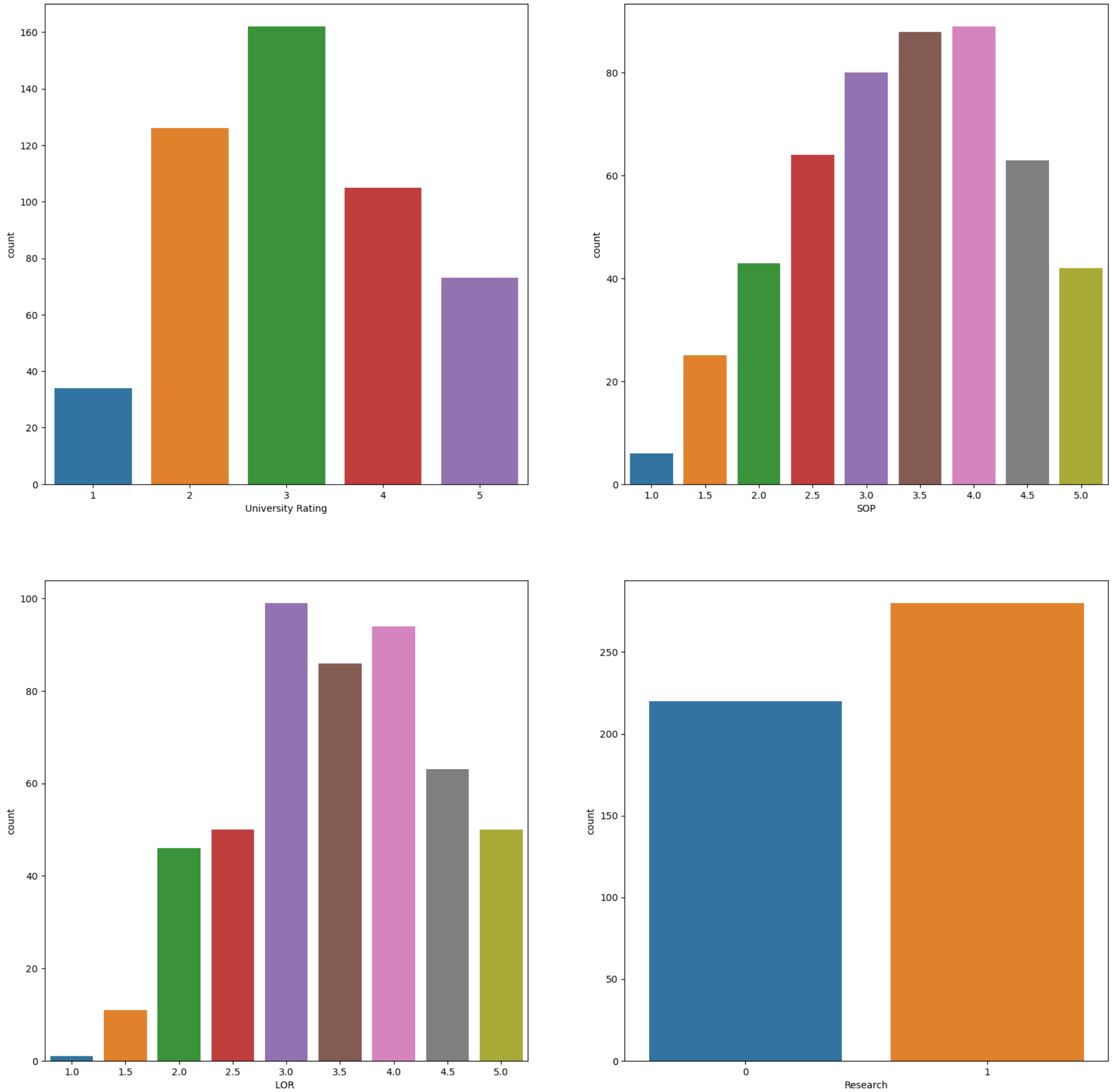


In [ ]:
```
Above distribution plots shows that they are normally distributed
```

In [244]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype   
---  ------             --------------  -----   
 0   GRE Score          500 non-null    int64   
 1   TOEFL Score        500 non-null    int64   
 2   University Rating  500 non-null    category
 3   SOP                500 non-null    category
 4   LOR                500 non-null    category
 5   CGPA               500 non-null    float64 
 6   Research           500 non-null    category
 7   Chance of Admit    500 non-null    float64 
dtypes: category(4), float64(2), int64(2)
memory usage: 18.8 KB
```

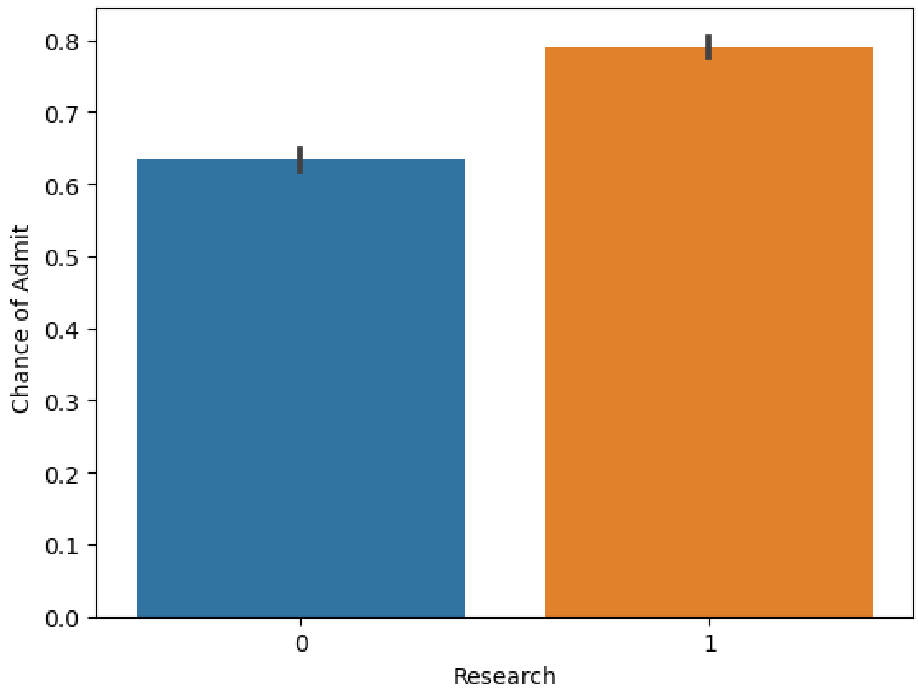
In [417]:
```
fig,axs=plt.subplots(nrows=2,ncols=2,figsize=(20,20))

cols=["University Rating","SOP","LOR ","Research"]
count=0

for i in range(2):
    for j in range(2):
            sns.countplot(data=df,x=cols[count],ax=axs[i,j])
            count +=1

plt.show()
```



Bivariate Analysis

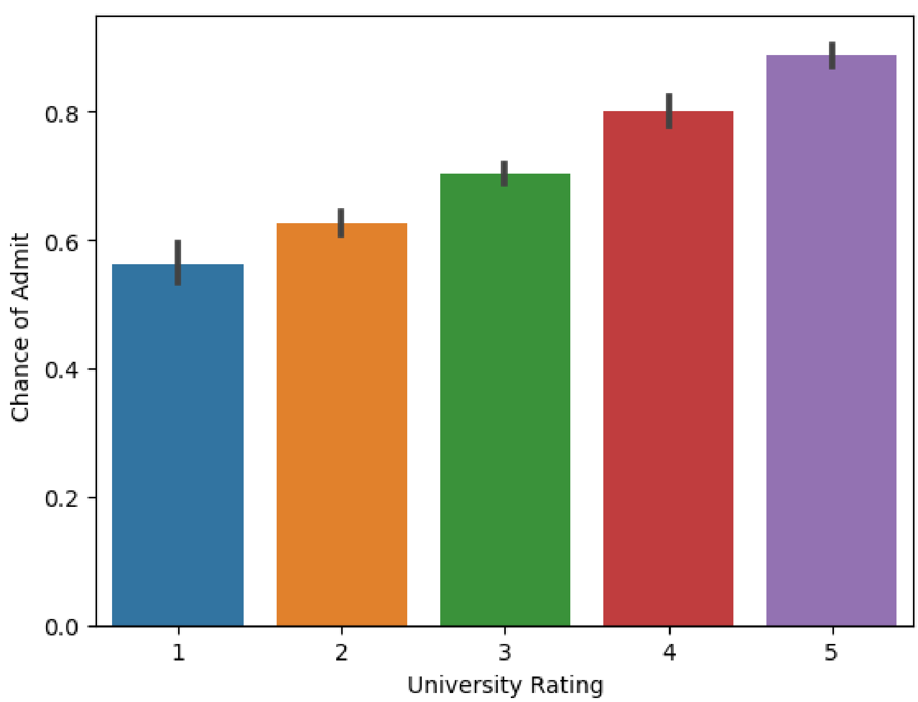
In [6]:
```
sns.barplot(data=df,x="Research",y="Chance of Admit ")
```

Out[6]: <Axes: xlabel='Research', ylabel='Chance of Admit '>



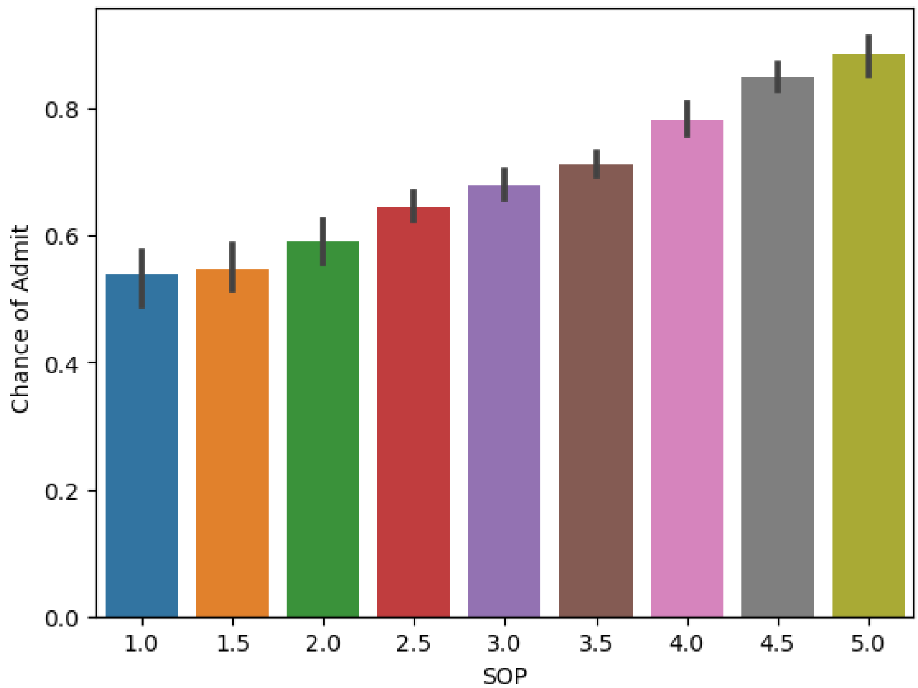
In [64]:
```
sns.barplot(data=df,x="University Rating",y="Chance of Admit ")
```

Out[64]: <Axes: xlabel='University Rating', ylabel='Chance of Admit '>



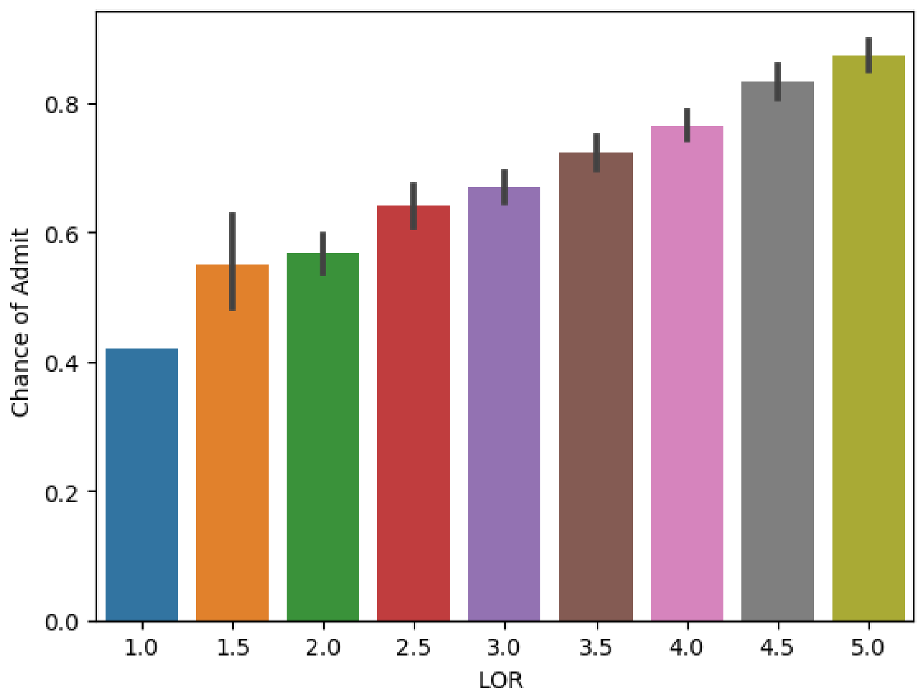
In [71]:
```
sns.barplot(data=df,x="SOP",y="Chance of Admit ")
```

Out[71]: <Axes: xlabel='SOP', ylabel='Chance of Admit '>



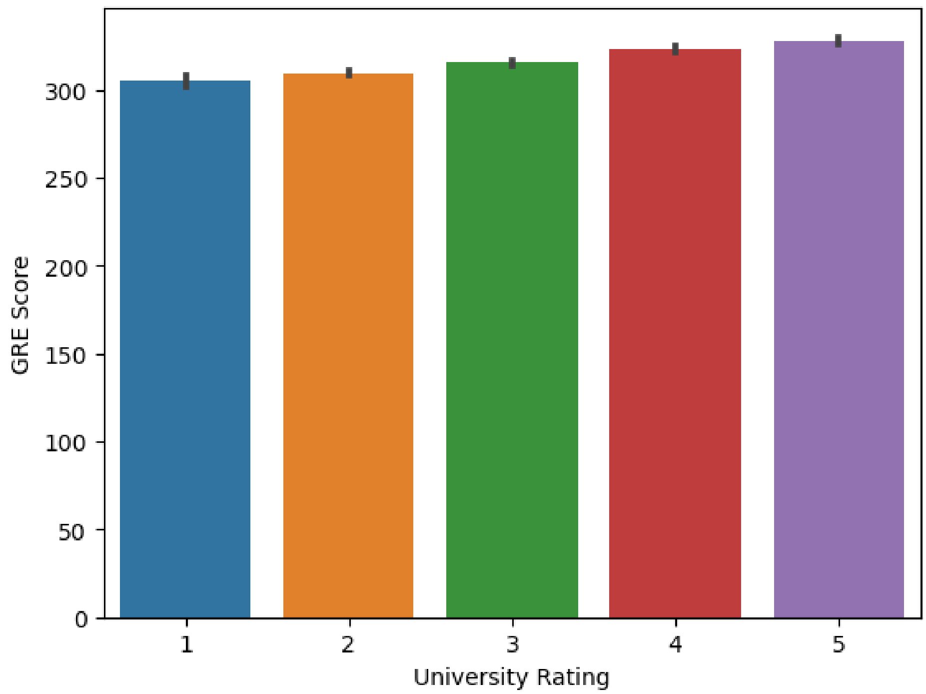
In [73]:
```
sns.barplot(data=df,x="LOR ",y="Chance of Admit ")
```

Out[73]: <Axes: xlabel='LOR ', ylabel='Chance of Admit '>



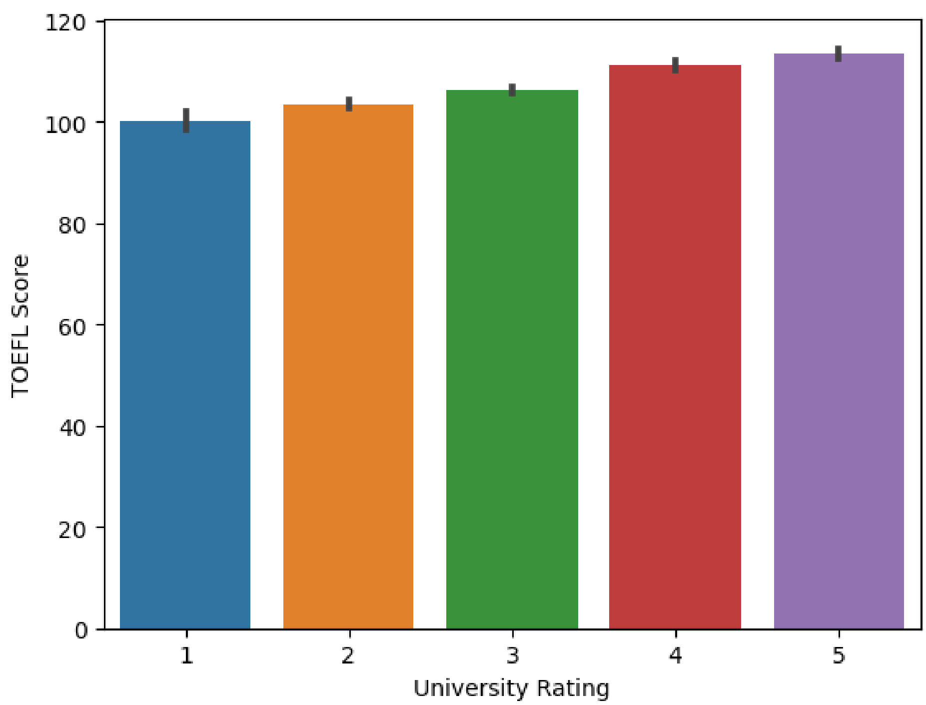
In [65]:
```
sns.barplot(data=df,x="University Rating",y="GRE Score")
```

Out[65]: <Axes: xlabel='University Rating', ylabel='GRE Score'>



In [74]:
```
sns.barplot(data=df,x="University Rating",y="TOEFL Score")
```

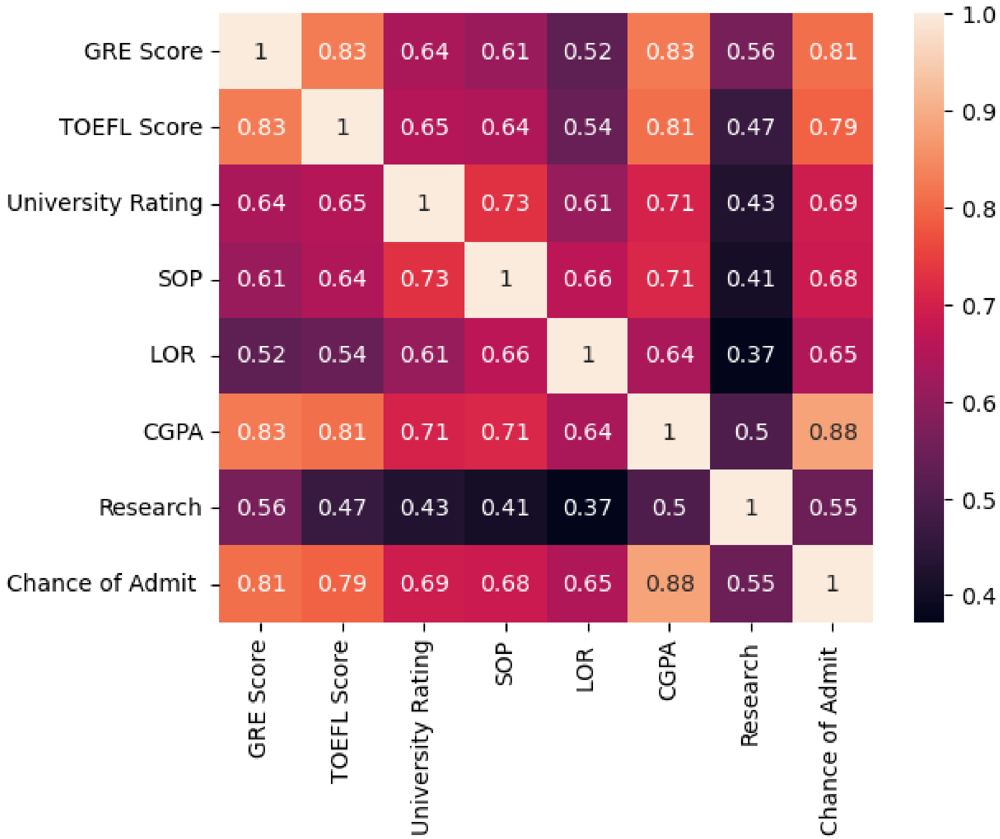
Out[74]: <Axes: xlabel='University Rating', ylabel='TOEFL Score'>



In [ ]:

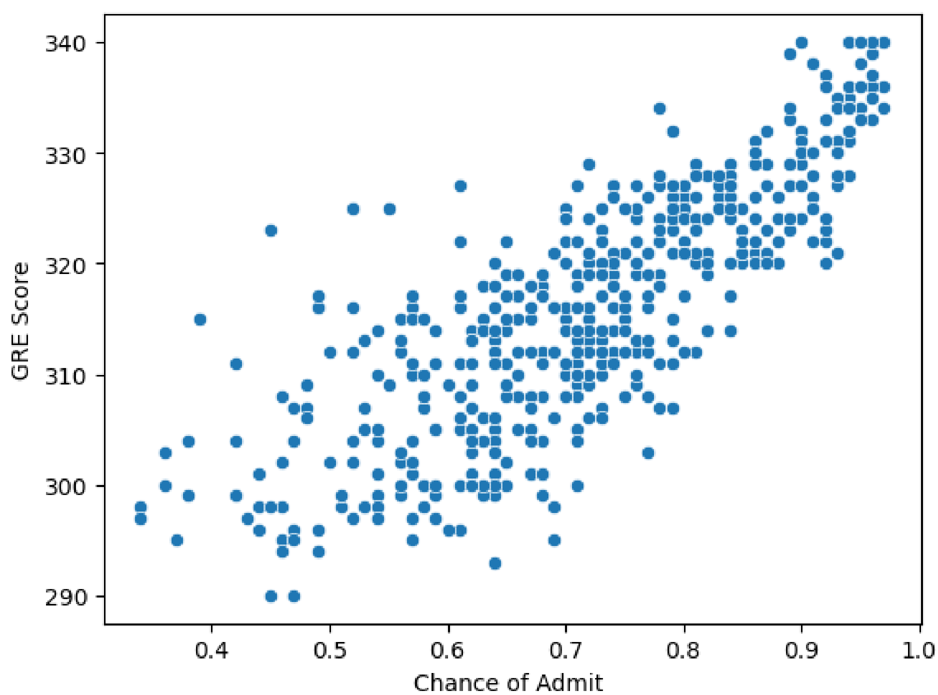
In [418]:
```
sns.heatmap(df.corr(),annot=True)
```

Out[418]: <Axes: >



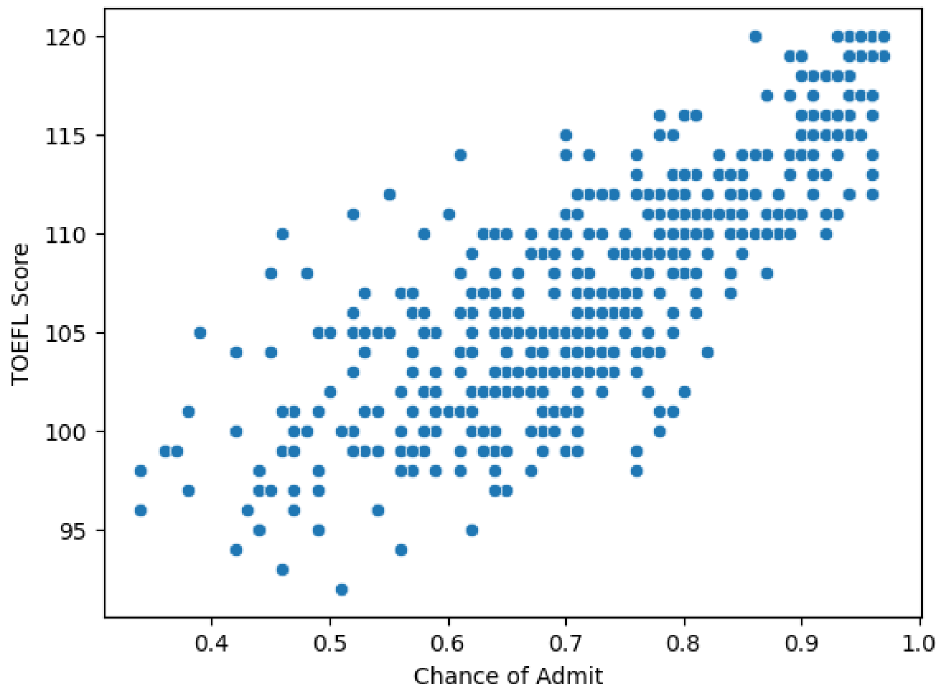
In [132]:
```
sns.scatterplot(data=df,x="Chance of Admit ",y="GRE Score")
```

Out[132]: <Axes: xlabel='Chance of Admit ', ylabel='GRE Score'>



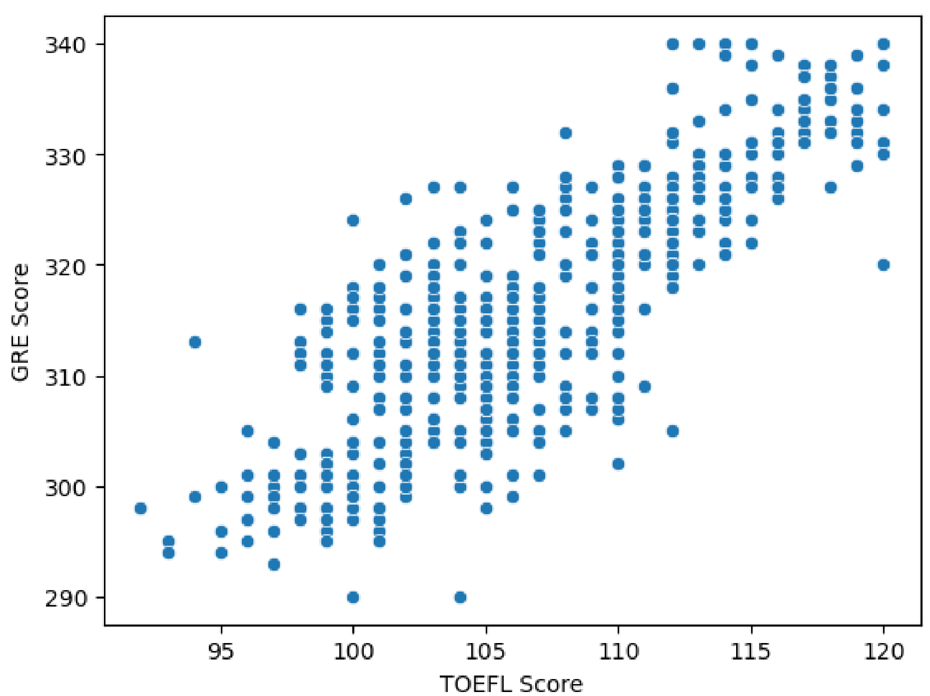
In [82]:
```
sns.scatterplot(data=df,x="Chance of Admit ",y="TOEFL Score")
```

Out[82]: <Axes: xlabel='Chance of Admit ', ylabel='TOEFL Score'>

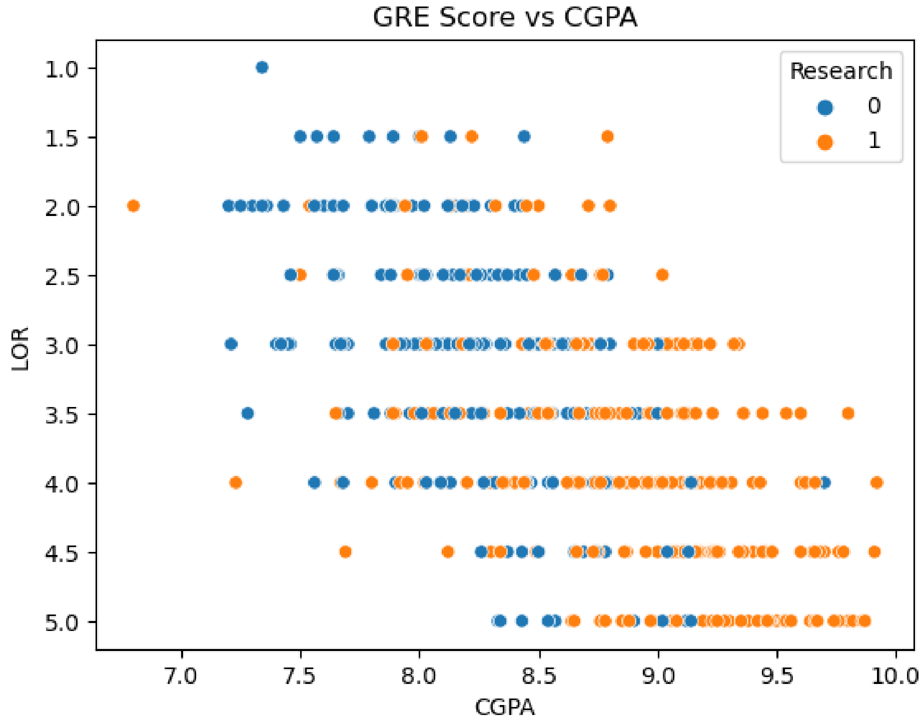


In [83]:
```
sns.scatterplot(data=df,x="TOEFL Score",y="GRE Score")
```

Out[83]: <Axes: xlabel='TOEFL Score', ylabel='GRE Score'>

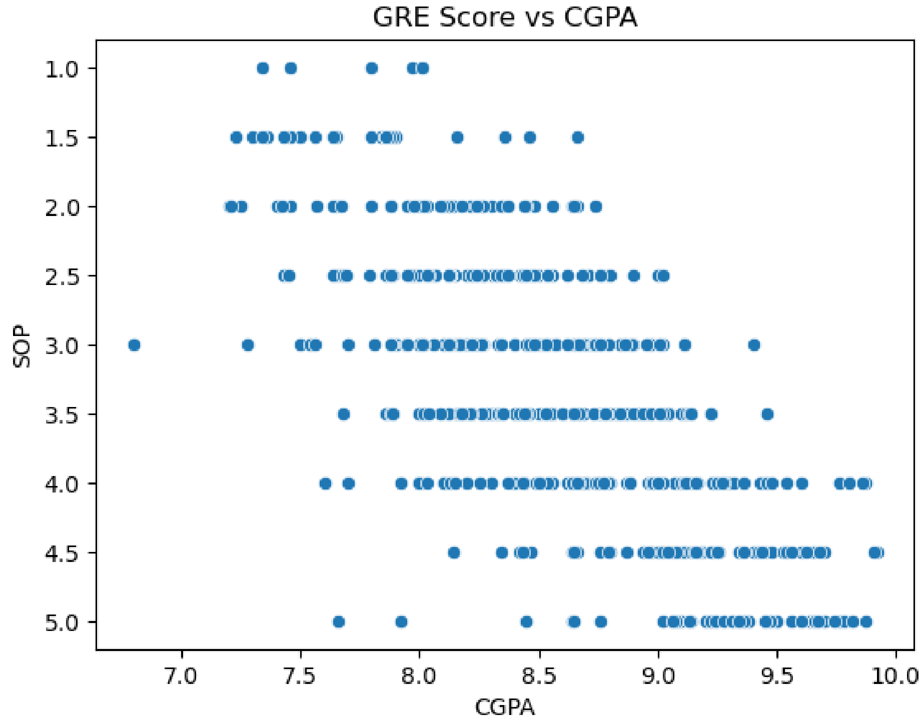


In [84]:
```
fig = sns.scatterplot(x="CGPA", y="LOR ", data=df, hue="Research")
plt.title("GRE Score vs CGPA")
plt.show()
```

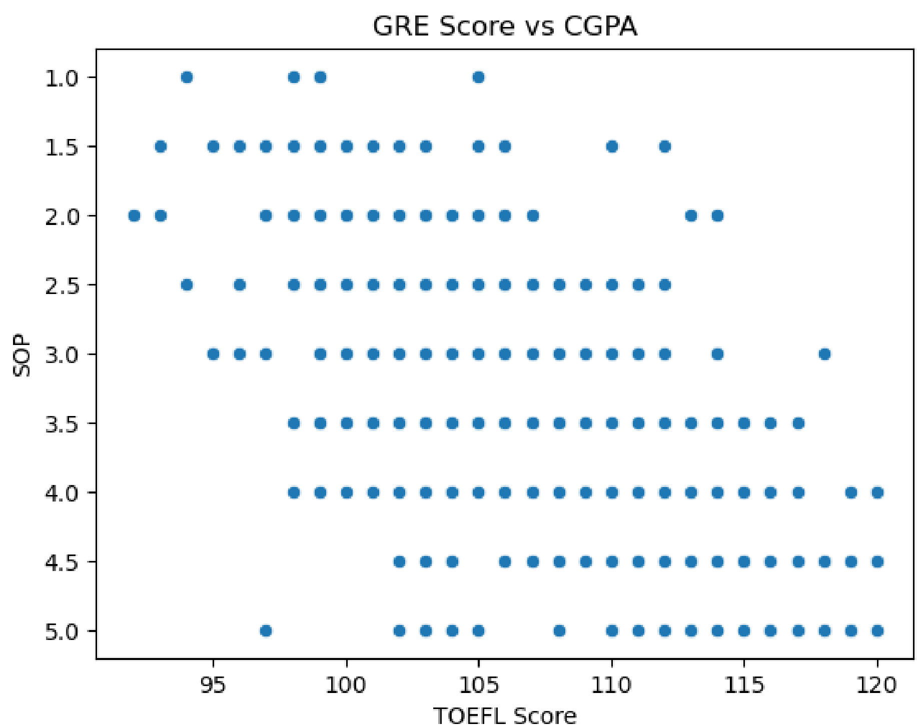


In [246]:
```
fig = sns.scatterplot(x="TOEFL Score", y="LOR ", data=df, hue="Research")
plt.title("GRE Score vs CGPA")
plt.show()
```

In [86]:
```
fig = sns.scatterplot(x="CGPA", y="SOP", data=df)
plt.title("GRE Score vs CGPA")
plt.show()
```



In [459]:
```
fig = sns.scatterplot(x="TOEFL Score", y="SOP", data=df)
plt.title("GRE Score vs CGPA")
plt.show()
```

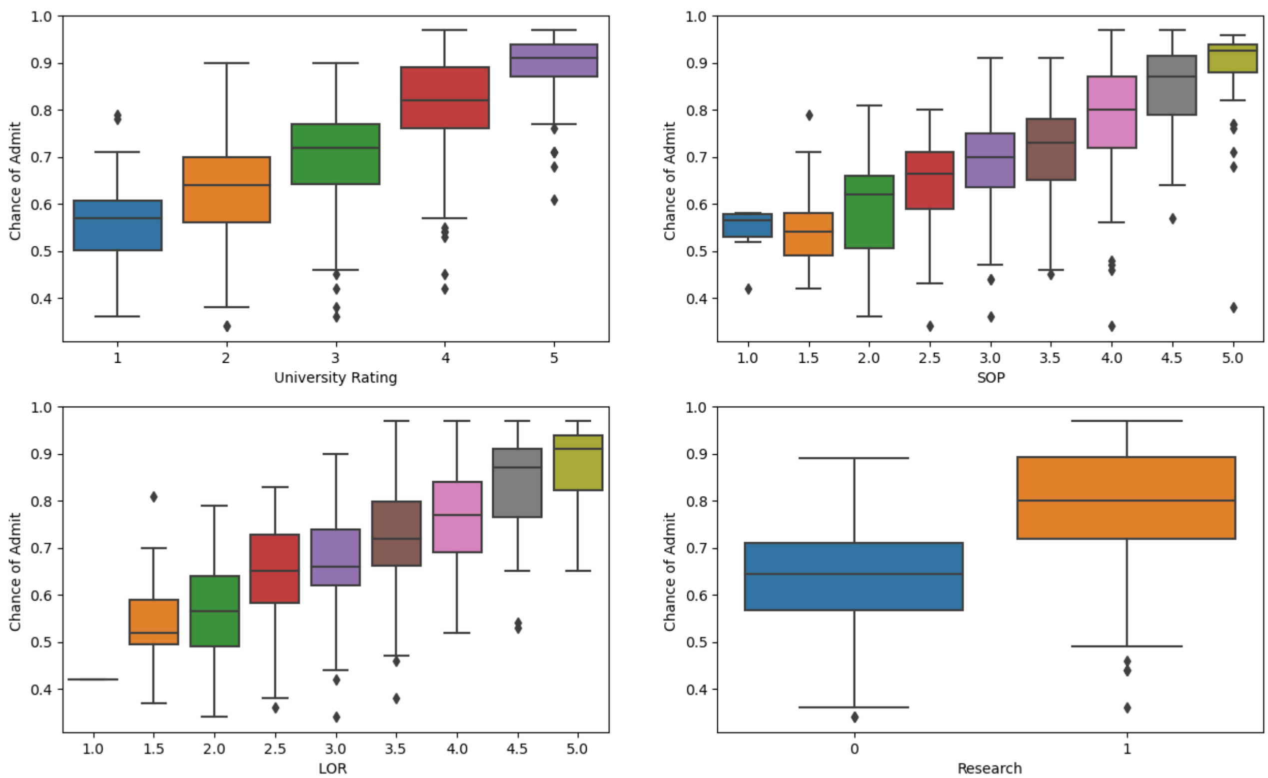


In [460]:
```
df.dtypes
```

Out[460]:
```
GRE Score               int64
TOEFL Score             int64
University Rating    category
SOP                  category
LOR                  category
CGPA                  float64
Research             category
Chance of Admit       float64
dtype: object
```

In [461]:
```
fig,axs=plt.subplots(nrows=2,ncols=2,figsize=(15,9))
sns.boxplot(x="University Rating",y="Chance of Admit ",data=df,ax=axs[0,0])
sns.boxplot(x="SOP",y="Chance of Admit ",data=df,ax=axs[0,1])
sns.boxplot(x="LOR ",y="Chance of Admit ",data=df,ax=axs[1,0])
sns.boxplot(x="Research",y="Chance of Admit ",data=df,ax=axs[1,1])
```

Out[461]:
```
<Axes: xlabel='Research', ylabel='Chance of Admit '>
```



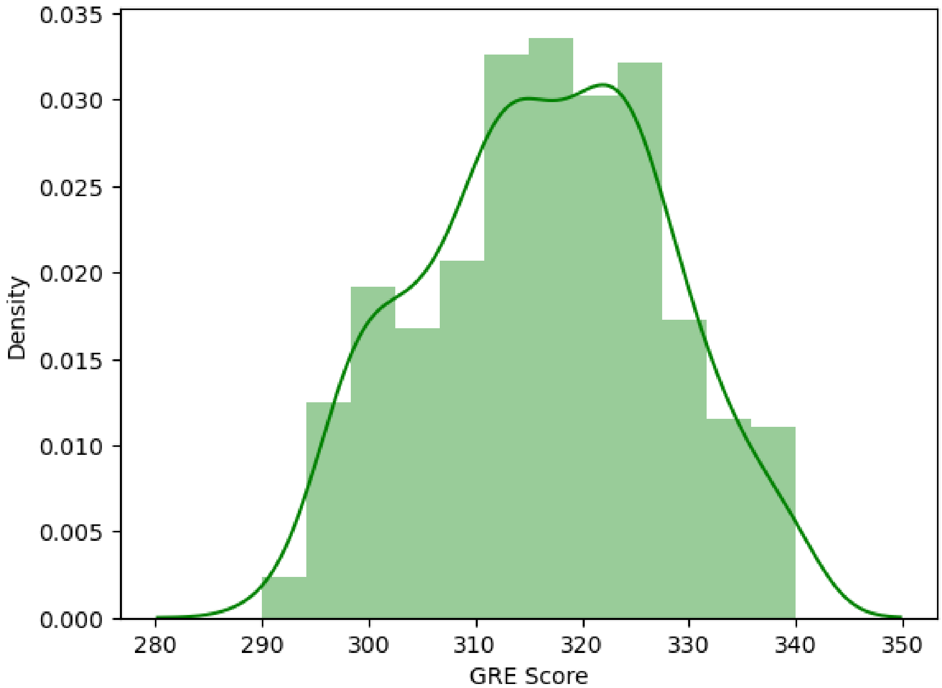
In [462]:
```
df.duplicated().sum()
```

Out[462]:
```
0
```

In [ ]:
```
Treating outliers
```

In [463]:
```
sns.distplot(df["GRE Score"],color="g")
```

Out[463]: <Axes: xlabel='GRE Score', ylabel='Density'>



In [133]:
```
x=df.drop("Chance of Admit ",axis=1)
y=df["Chance of Admit "]
```

In [134]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)
```

In [135]:
```
x_train
```

Out[135]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
84	340	115	5	4.5	4.5	9.45	1
416	315	104	3	4.0	2.5	8.10	0
133	323	112	5	4.0	4.5	8.78	0
60	309	100	2	3.0	3.0	8.10	0
322	314	107	2	2.5	4.0	8.27	0
...	...	...	...	...	...	...	...
143	340	120	4	4.5	4.0	9.92	1
261	312	104	3	3.5	4.0	8.09	0
200	317	103	3	2.5	3.0	8.54	1
421	321	112	3	3.0	4.5	8.95	1
82	320	110	5	5.0	4.5	9.22	1

400 rows × 7 columns

In [137]:
```
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[137]: ((400, 7), (400,), (100, 7), (100,))

In [138]:
```
from sklearn.preprocessing import StandardScaler
X_train_columns=x_train.columns
std=StandardScaler()
X_train_std=std.fit_transform(x_train)
```

In [139]:
```
X_train_std
```

Out[139]:
```
array([[ 2.08194   ,  1.31212211,  1.62731455, ...,  1.07999323,
         1.49551735,  0.89543386],
       [-0.1139676 , -0.49997371, -0.09927914, ..., -1.05596372,
        -0.75727045, -1.11677706],
       [ 0.58872283,  0.81791416,  1.62731455, ...,  1.07999323,
         0.37746711, -1.11677706],
       ...,
       [ 0.061705  , -0.66470969, -0.09927914, ..., -0.52197448,
        -0.0230285 ,  0.89543386],
       [ 0.41305022,  0.81791416, -0.09927914, ...,  1.07999323,
         0.6611515 ,  0.89543386],
       [ 0.32521392,  0.48844219,  1.62731455, ...,  1.07999323,
         1.11170906,  0.89543386]])
```

In [140]:
```
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Lasso,Ridge,LinearRegression
from sklearn.metrics import accuracy_score
```

In [141]:
```
X_train=pd.DataFrame(X_train_std, columns=X_train_columns)
```

In [142]:
```
X_train
```

...

In [143]:
```
lin = LinearRegression()
lin.fit(X_train,y_train.values)
y_pred3=lin.predict(std.transform(x_test))
lin.coef_
```

Out[143]:
```
array([0.02417929, 0.01875231, 0.00598916, 0.00269915, 0.01675084,
       0.06675081, 0.01017487])
```

In [144]:
```
lin.intercept_
```

Out[144]:
```
0.7179750000000001
```

In [145]:
```
mse = mean_squared_error(y_test,y_pred3)
print('Mean SquAred Error = ',mse )
```

```
Mean SquAred Error =  0.0033975693207486113
```

In [146]:
```
reg = Lasso(alpha = 0.1)
reg.fit(X_train,y_train)
y_pred1=reg.predict(std.transform(x_test))
reg.coef_
```

Out[146]:
```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.02328122, 0.        ])
```

In [147]:
```
reg.intercept_
```

Out[147]:
```
0.717975
```

In [148]:
```
mse = mean_squared_error(y_test,y_pred1)
print('\nMean SquAred Error = ',mse )
```

```
Mean SquAred Error =  0.014711318628660916
```

In [149]:
```
reg1 = Ridge(alpha = 1.0)
reg1.fit(X_train,y_train)
y_pred2=reg1.predict(std.transform(x_test))
reg1.coef_
```

Out[149]:
```
array([0.02429928, 0.01888516, 0.00607765, 0.00284522, 0.01676942,
       0.06619082, 0.0102024 ])
```

In [97]:
```
reg1.intercept_
```

Out[97]:
```
0.722475
```

In [150]:
```
mse = mean_squared_error(y_test,y_pred2)
print('\nMean SquAred Error = ',mse )
```

```
Mean SquAred Error =  0.003401805424686462
```

In [151]:
```
import statsmodels.api as sm
X_train = sm.add_constant(X_train)
model = sm.OLS(y_train.values, X_train).fit()
print(model.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                      y   R-squared:                       0.819
Model:                            OLS   Adj. R-squared:                  0.815
Method:                 Least Squares   F-statistic:                     252.7
Date:                Sun, 07 Jan 2024   Prob (F-statistic):          5.01e-141
Time:                        16:33:05   Log-Likelihood:                 558.55
No. Observations:                 400   AIC:                            -1101.
Df Residuals:                     392   BIC:                            -1069.
Df Model:                           7                                         
Covariance Type:            nonrobust                                         
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const                 0.7180      0.003    237.371      0.000       0.712       0.724
GRE Score             0.0242      0.006      3.889      0.000       0.012       0.036
TOEFL Score           0.0188      0.006      3.228      0.001       0.007       0.030
University Rating     0.0060      0.005      1.231      0.219      -0.004       0.016
SOP                   0.0027      0.005      0.532      0.595      -0.007       0.013
LOR                   0.0168      0.004      3.864      0.000       0.008       0.025
CGPA                  0.0668      0.006     10.310      0.000       0.054       0.079
Research              0.0102      0.004      2.758      0.006       0.003       0.017
==============================================================================
Omnibus:                       87.250   Durbin-Watson:                   2.129
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              199.415
Skew:                          -1.104   Prob(JB):                     4.98e-44
Kurtosis:                       5.663   Cond. No.                         5.45
==============================================================================

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

R-squared & Adj. R-squared both are almost same 82 % which is a good value because values grester than 0.5-1 are considered good fit model.Though it is noticed that SOP has low coeff are weight so it can be neglected.

In [152]:
```
X_train_new = X_train.drop(columns="SOP")
```

In [153]:
```
model1 = sm.OLS(y_train.values, X_train_new).fit()
print(model1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.818
Model:                            OLS   Adj. R-squared:                  0.816
Method:                 Least Squares   F-statistic:                     295.3
Date:                Sun, 07 Jan 2024   Prob (F-statistic):          3.22e-142
Time:                        16:33:13   Log-Likelihood:                 558.41
No. Observations:                 400   AIC:                            -1103.
Df Residuals:                     393   BIC:                            -1075.
Df Model:                           6
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const                 0.7180      0.003    237.588      0.000       0.712       0.724
GRE Score             0.0242      0.006      3.890      0.000       0.012       0.036
TOEFL Score           0.0189      0.006      3.260      0.001       0.008       0.030
University Rating     0.0070      0.004      1.551      0.122      -0.002       0.016
LOR                   0.0175      0.004      4.263      0.000       0.009       0.026
CGPA                  0.0674      0.006     10.611      0.000       0.055       0.080
Research              0.0101      0.004      2.754      0.006       0.003       0.017
==============================================================================
Omnibus:                       85.381   Durbin-Watson:                   2.129
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              192.913
Skew:                          -1.086   Prob(JB):                     1.29e-42
Kurtosis:                       5.619   Cond. No.                         5.05
==============================================================================

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

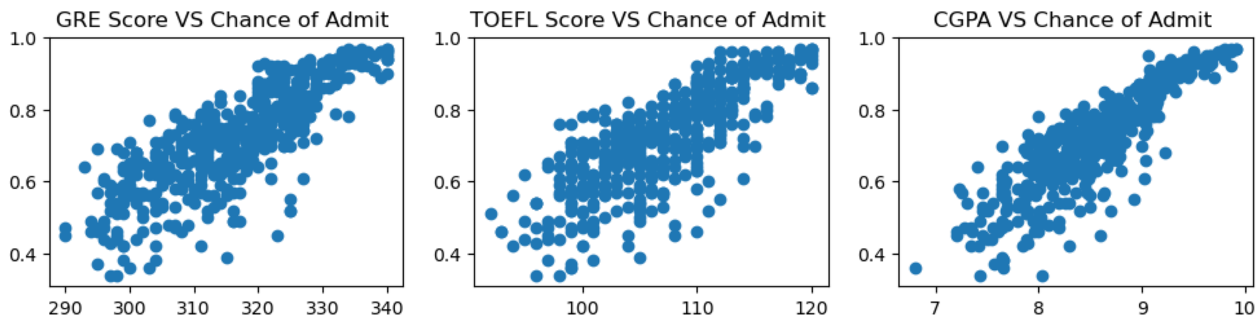
After removing it found that both R-squared & Adj. R-squared values are same with 82 % there is not much difference and still model remains fit.

Testing the assumptions of LR model

1)Linearity :

```
All the nuerical features like GRE TOEFL,CGPA  are linear with Chance of Admit
```

In [176]:
```
fig,(ax1,ax2,ax3) = plt.subplots(ncols=3,figsize=(12,2.5))
ax1.scatter(df["GRE Score"],df["Chance of Admit "])
ax1.set_title("GRE Score VS Chance of Admit ")
ax2.scatter(df["TOEFL Score"],df["Chance of Admit "])
ax2.set_title("TOEFL Score VS Chance of Admit ")
ax3.scatter(df["CGPA"],df["Chance of Admit "])
ax3.set_title("CGPA VS Chance of Admit ")
plt.show()
```



2)Multicollinearity:

In [154]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
VIF=[]

for i in range(X_train_new.shape[1]):
    VIF.append(variance_inflation_factor(X_train_new,i))

pd.DataFrame({"VIF" : VIF},index=X_train_new.columns).T
```

Out[154]:

	const	GRE Score	TOEFL Score	University Rating	LOR	CGPA	Research
VIF	1.0	4.225817	3.681038	2.213631	1.842974	4.417958	1.486966

Since VIF < 5 , there is low or no multicollinearity.Thus

Model performance evaluation"

```
Setting test dataset to match with dimension of train dataset.
```

In [156]:
```
x_test_std=std.fit_transform(x_test)
```

In [158]:
```
X_test=pd.DataFrame(x_test_std, columns=X_train_columns)
```

In [159]:
```
X_test = sm.add_constant(X_test)
```

In [161]:
```
X_test_del=list(set(X_test.columns).difference(set(X_train_new.columns)))
X_test_del
```

Out[161]: ['SOP']

In [162]:
```
X_test_new=X_test.drop(columns=X_test_del)
```

In [191]:
```
y_pred = model1.predict(X_test_new)

from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error

print('Mean Absolute Error ', mean_absolute_error(y_test.values,y_pred) )
print('Root Mean Square Error ', np.sqrt(mean_squared_error(y_test.values,y_pred) ))
print('R2 Score', r2_score(y_test.values,y_pred))

n=y_test.shape[0]
k = 1
Adj_r2score = 1-((1 - r2_score(y_test.values,y_pred)) * (n-1) / (n-k-1))
print('Adj_r2 Score', Adj_r2score)
```

```
Mean Absolute Error  0.04698336629208801
Root Mean Square Error  0.0613138027935518
R2 Score 0.8124412578774362
Adj_r2 Score 0.8105273931618998
```

```
MAE & RMSE values tends to zero and is very low which is good sign that model is performing well.
R2 score  and Adj_r2score = 0.8 which tells that model capable of explaining 80% of variance of
data.
```

In [197]:
```
y_pred_t = model1.predict(X_train_new)

from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error

print('Mean Absolute Error ', mean_absolute_error(y_train.values,y_pred_t) )
print('Root Mean Square Error ', np.sqrt(mean_squared_error(y_train.values,y_pred_t) ))
print('R2 Score', r2_score(y_train.values,y_pred_t))

n=y_train.shape[0]
k = 1
Adj_r2score = 1-((1 - r2_score(y_train.values,y_pred_t)) * (n-1) / (n-k-1))
print('Adj_r2 Score', Adj_r2score)
```

```
Mean Absolute Error  0.04319820422265458
Root Mean Square Error  0.05990745344713923
R2 Score 0.8184594131773991
Adj_r2 Score 0.8180032810497042
```

```
MAE & RMSE values tends to zero and is very low which is good sign that model is performing well.
R2 score  and Adj_r2score = 0.8 which tells that model capable of explaining 80% of variance of
data.
```

3)Normality of Residuals:

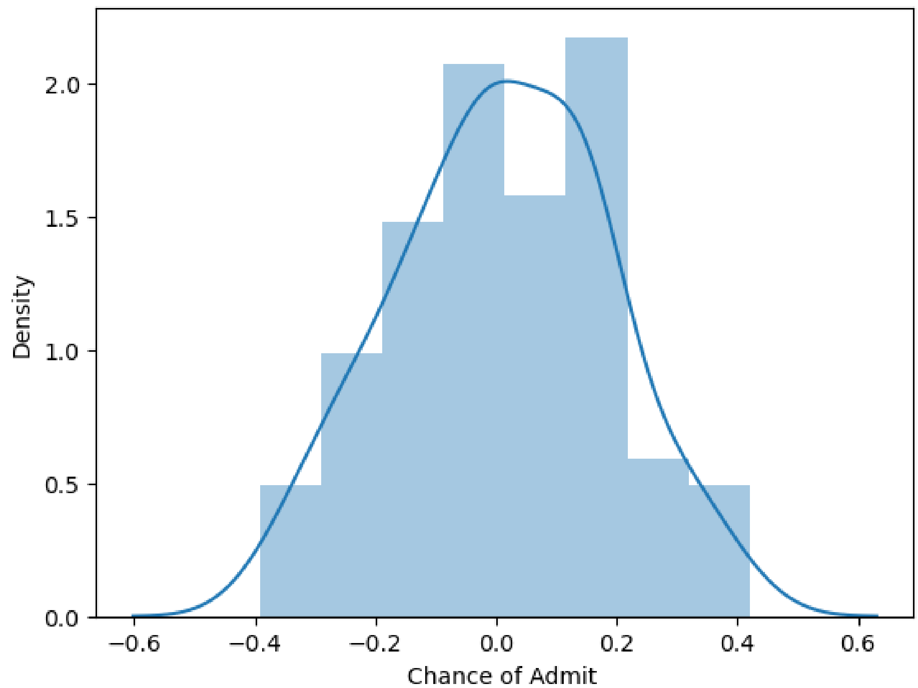
In [ ]:
```
residuals =  y_test - y_pred
```

In [165]:
```
np.mean(residuals)
```

Out[165]: 0.010404491966023461

In [166]:
```
sns.distplot(residuals,kde=True)
```

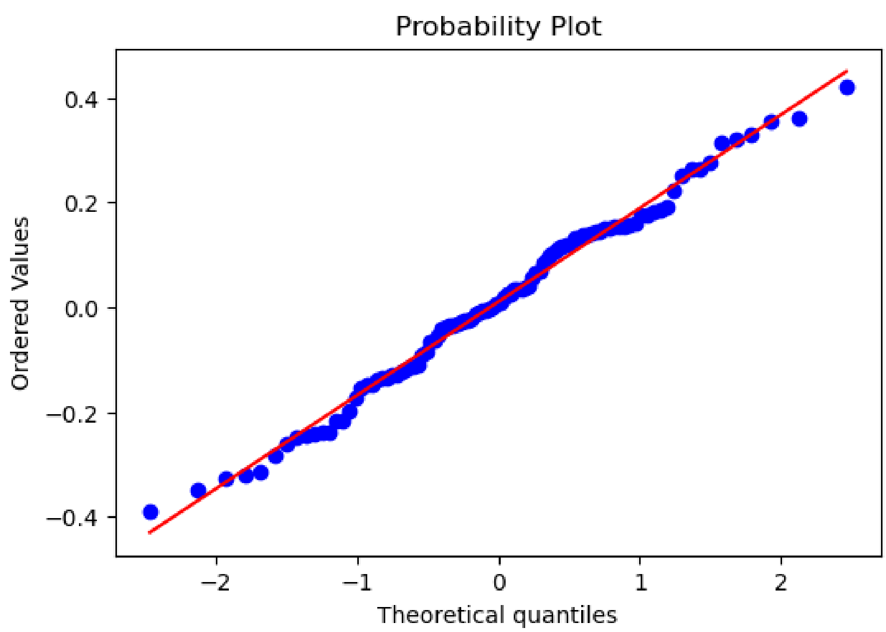
Out[166]: <Axes: xlabel='Chance of Admit ', ylabel='Density'>



In [295]:
```
x_test_new.shape
```

Out[295]: (100, 6)

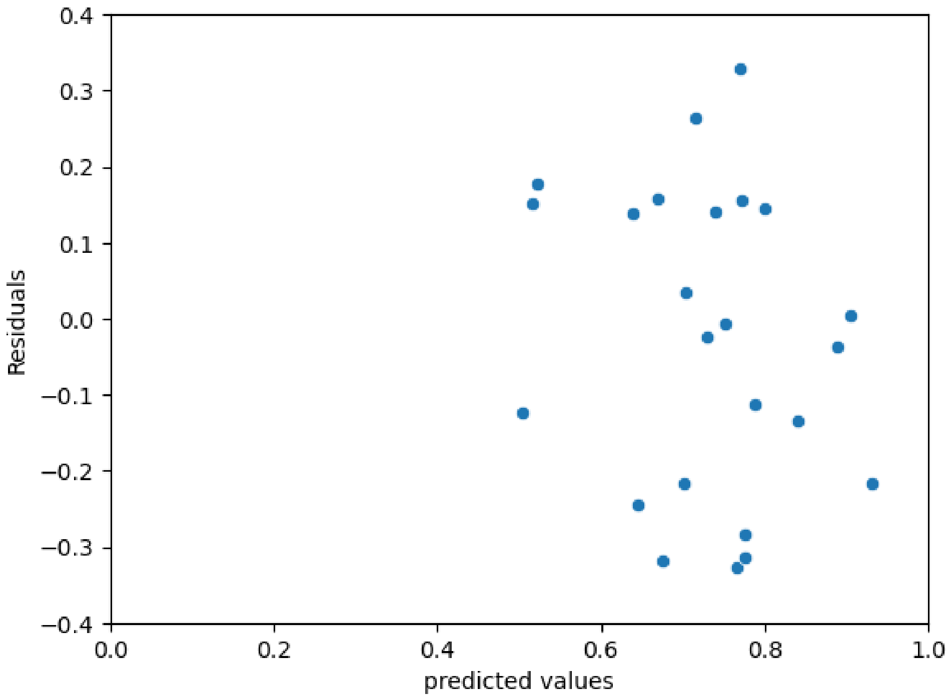
In [167]:
```
import scipy as sp
fig,ax=plt.subplots(figsize=(6,4))
sp.stats.probplot(residuals,plot=ax,fit=True)
plt.show()
```



Test for Homoscedasticity

In [181]:
```
p = sns.scatterplot(x=pred,y=residuals)
plt.xlabel('predicted values')
plt.ylabel('Residuals')
plt.ylim(-0.4,0.4)
plt.xlim(0,1)
```

Out[181]: (0.0, 1.0)

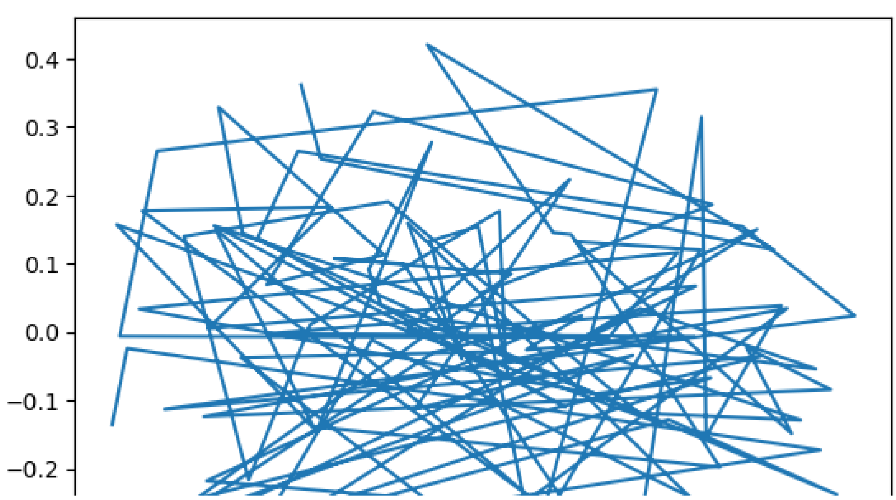


when you imagine a straightline passing at 0 of residual y-axis , you can see the plot is uniform or same on both sides of the plot.

Autocorrelation check

In [182]:
```
plt.plot(residuals)
```

Out[182]: [<matplotlib.lines.Line2D at 0x206fc4ec2d0>]



No autocorrelation , that is no pattern found in residual data.

In [ ]: