Context

Jamboree has helped thousands of students like you make it to top colleges abroad. Be it GMAT, GRE or SAT, their unique problem-solving methods ensure maximum scores with minimum effort. They recently launched a feature where students/learners can come to their website and check their probability of getting into the IVY league college. This feature estimates the chances of graduate admission from an Indian perspective.

How can you help here?

Your analysis will help Jamboree in understanding what factors are important in graduate admissions and how these factors are interrelated among themselves. It will also help predict one's chances of admission given the rest of the variables.

Dataset:

Dataset Link: jamboree_admission.csv

Column Profiling:

Serial No. (Unique row ID)

GRE Scores (out of 340)

TOEFL Scores (out of 120)

University Rating (out of 5)

Statement of Purpose and Letter of Recommendation Strength (out of 5)

Undergraduate GPA (out of 10)

Research Experience (either 0 or 1)

Chance of Admit (ranging from 0 to 1)

Concept Used:

Exploratory Data Analysis

Linear Regression

In [68]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import stats
import statsmodels.api as sm
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
JDF = pd.read_csv(r"H:\Scaler\MACHINE LEARNING INTRO\PROJECT\Jamboree_Admission.cs\
```

In [3]:
```
JDF.head()
```

Out[3]:

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

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

Out[4]: (500, 9)

In [5]:
```
JDF.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%	max
Serial No.	500.0	250.50000	144.481833	1.00	125.7500	250.50	375.25	500.00
GRE Score	500.0	316.47200	11.295148	290.00	308.0000	317.00	325.00	340.00
TOEFL Score	500.0	107.19200	6.081868	92.00	103.0000	107.00	112.00	120.00
University Rating	500.0	3.11400	1.143512	1.00	2.0000	3.00	4.00	5.00
SOP	500.0	3.37400	0.991004	1.00	2.5000	3.50	4.00	5.00
LOR	500.0	3.48400	0.925450	1.00	3.0000	3.50	4.00	5.00
CGPA	500.0	8.57644	0.604813	6.80	8.1275	8.56	9.04	9.92
Research	500.0	0.56000	0.496884	0.00	0.0000	1.00	1.00	1.00
Chance of Admit	500.0	0.72174	0.141140	0.34	0.6300	0.72	0.82	0.97

In [6]:
```
JDF.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 [7]:
```
# Missing value detection ..
def missingValue(df):
    total_null = df.isnull().sum().sort_values(ascending = False)
    percent = ((df.isnull().sum()/df.isnull().count())*100).sort_values(ascending 
    print("Total number of records = ", df.shape[0])
    md = pd.concat([total_null, percent.round(2)],axis=1,keys=['Total Missing valu
    return md
```

In [8]:
```
for i in JDF.columns:
    print(i, ':',JDF[i].nunique())
```

```
Serial No. : 500
GRE Score : 49
TOEFL Score : 29
University Rating : 5
SOP : 9
LOR  : 9
CGPA : 184
Research : 2
Chance of Admit  : 61
```

Observations:

From data, we can see that there are 9 columns and 500 rows.

Out of all columns , 5 numerical descrete and 4 numerical continuous variables.

There are no missing values in the data.

GRE Score has range from 290 to 340 with average score as 316.

TOEFL Score has range from 92 to 120 with average score as 107.

University ratings ranges from 1 to 5.

CGPA has range from 6.8 to 9.92 with average score as 8.57.

Research has only 2 possible values 0 and 1.

Chance of Admit ranges from .34 to .97 with average as .72.

In [9]:
```
JDF[JDF.duplicated(keep=False)]
```

Out[9]:

Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit

In [10]:
```
JDF_data =JDF.copy()
```

In [11]:
```
def get_analysis_of_univariate_cat(df, colnames, nrows=2, mcols=2, width=20, heigh
    fig , ax = plt.subplots(nrows,mcols,figsize=(width,height))
    fig.set_facecolor(color = 'white')
    string = "Frequency of "
    rows = 0
    for colname in colnames:
        count = (df[colname].value_counts(normalize=True)*100)
        string += colname + ' in (%)'
        if sortbyindex:
                count = count.sort_index()
        # Plot count Plot for categorical features
        count.plot.bar(color=sns.color_palette("Set2"),ax=ax[rows][0])
        ax[rows][0].set_ylabel(string, fontsize=14,family = "Comic Sans MS")
        ax[rows][0].set_xlabel(colname, fontsize=14,family = "Comic Sans MS")
        # Plot Pie chart for categorical features
        count.plot.pie(colors = sns.color_palette("Set2"),autopct='%0.0f%%',
                       textprops={'fontsize': 14,'family':"Comic Sans MS"},ax=ax[r
        string = "Frequency of "
        rows += 1
```

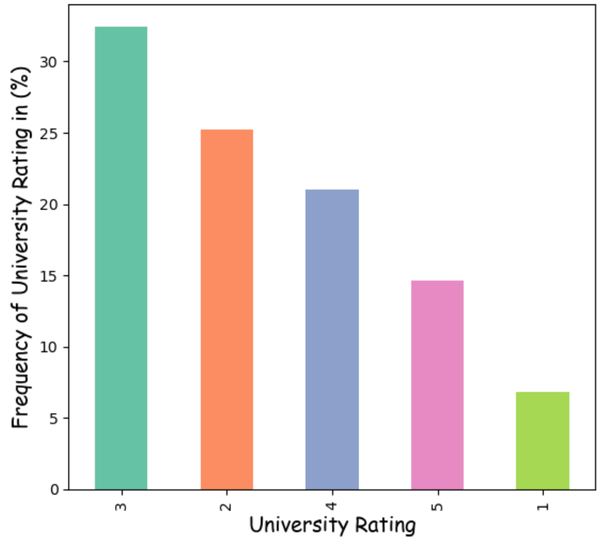
In [12]:
```
def get_analysis_of_bivariate_cat(df, colname, depend_var, nrows=2, mcols=2, width
    fig , ax = plt.subplots(nrows, mcols, figsize=(width,height))
    sns.set(style='white')
    rows = 0
    string = " based Distribution"
    for var in colname:
        string = var + string

        # countplot with x and hue
        sns.countplot(data=df, x=depend_var, hue = var, palette="Set2", ax = ax[ro

        # countplot with x and hue
        sns.countplot(data=df, x=var, hue = depend_var, palette="Set2", ax = ax[ro

        ax[rows][0].set_title(string, fontweight="bold", fontsize=14, family = "Co
        ax[rows][1].set_title(string, fontweight="bold", fontsize=14, family = "Co

        ax[rows][0].set_ylabel('count', fontweight="bold", fontsize=14, family = "
        ax[rows][0].set_xlabel(depend_var, fontweight="bold", fontsize=14, family

        ax[rows][1].set_ylabel('count', fontweight="bold",fontsize=14, family = "C
        ax[rows][1].set_xlabel(var, fontweight="bold", fontsize=14, family = "Comi

        rows += 1
        string = " based Distribution"
    plt.show()
```

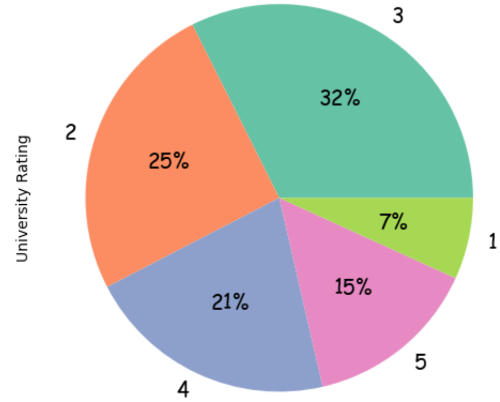
In [13]:
```
def get_outlier(df, colname, nrows=2, mcols=2, width=14, height=20):
    fig , ax = plt.subplots(nrows, mcols, figsize=(width,height))
    fig.set_facecolor("lightgrey")
    rows = 0
    for var in colname:
        ax[rows][0].set_title("Boxplot for Outlier Detection ", fontweight="bold")
        plt.ylabel(var, fontsize=12, family = "Comic Sans MS")

        # Plot Boxplot to get outliers for continuous numerical features
        sns.boxplot(y = df[var], color='m', ax=ax[rows][0])

        # Plot distplot to get distribution for continuous numerical features
        sns.distplot(df[var], color='m', ax=ax[rows][1])

        # Get mean vertical line
        ax[rows][1].axvline(df[var].mean(), color='r', linestyle='--', label="Mean

        # Get median vertical line
        ax[rows][1].axvline(df[var].median(), color='g', linestyle='-', label="Med

        # Get mode vertical line
        ax[rows][1].axvline(df[var].mode()[0], color='royalblue', linestyle='-', la

        # set the title
        ax[rows][1].set_title("Outlier Detection ", fontweight="bold")

        # add the legend
        ax[rows][1].legend({'Mean':df[var].mean(), 'Median':df[var].median(), 'Mod
        rows += 1
    plt.show()
```

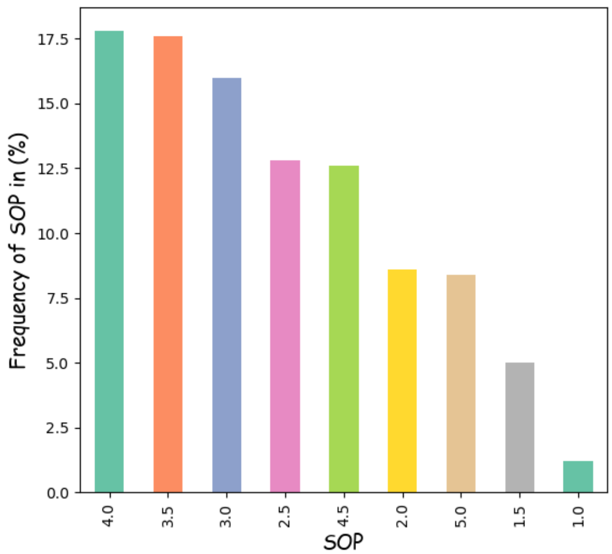
In [14]:
```
def num_cat_bi(df, col_cat, col_num, nrows=1, mcols=2, width=15, height=6):
    fig , ax = plt.subplots(nrows,mcols,figsize=(width,height),squeeze=False)
    sns.set(style='white')
    fig.set_facecolor("lightgrey")
    rows = 0
    i = 0
    while rows < nrows:

        # box plot with x and y
        sns.boxplot(x = col_cat[i],y = col_num, data = df,ax=ax[rows][0],palette="
        ax[rows][0].set_xlabel(col_cat[i], fontweight="bold",fontsize=14,family =
        ax[rows][0].set_ylabel(col_num,fontweight="bold", fontsize=14,family = "Co
        i += 1

        # box plot with x and y
        sns.boxplot(x = col_cat[i],y = col_num, data = df,ax=ax[rows][1],palette="
        ax[rows][1].set_xlabel(col_cat[i], fontweight="bold",fontsize=14,family =
        ax[rows][1].set_ylabel(col_num,fontweight="bold", fontsize=14,family = "Co
        i += 1
        rows += 1
    plt.show()
```

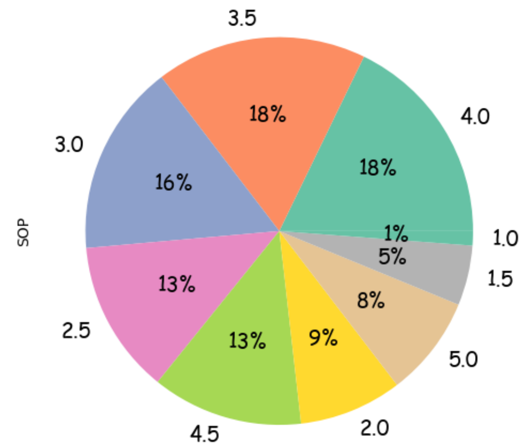
Univariate Analysis

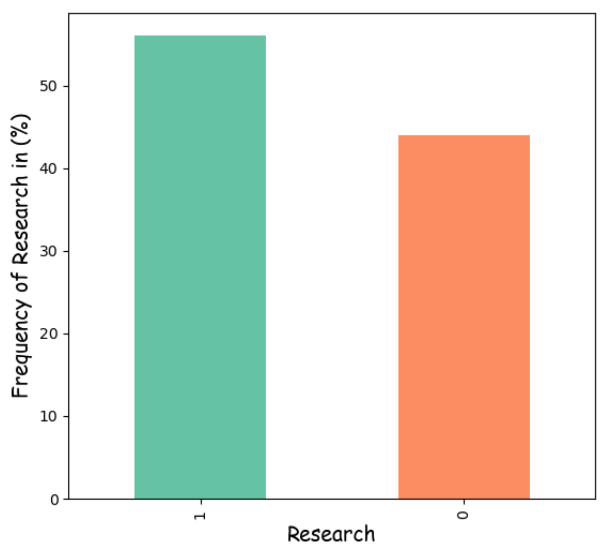
In [15]:
```
cat_cols = ['University Rating', 'SOP', 'Research']
get_analysis_of_univariate_cat(JDF_data, cat_cols, 3, 2, 14, 20)
```

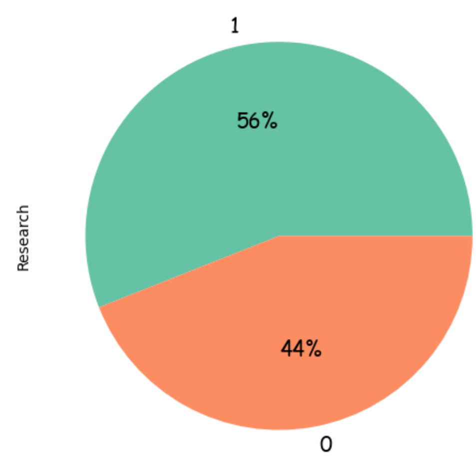






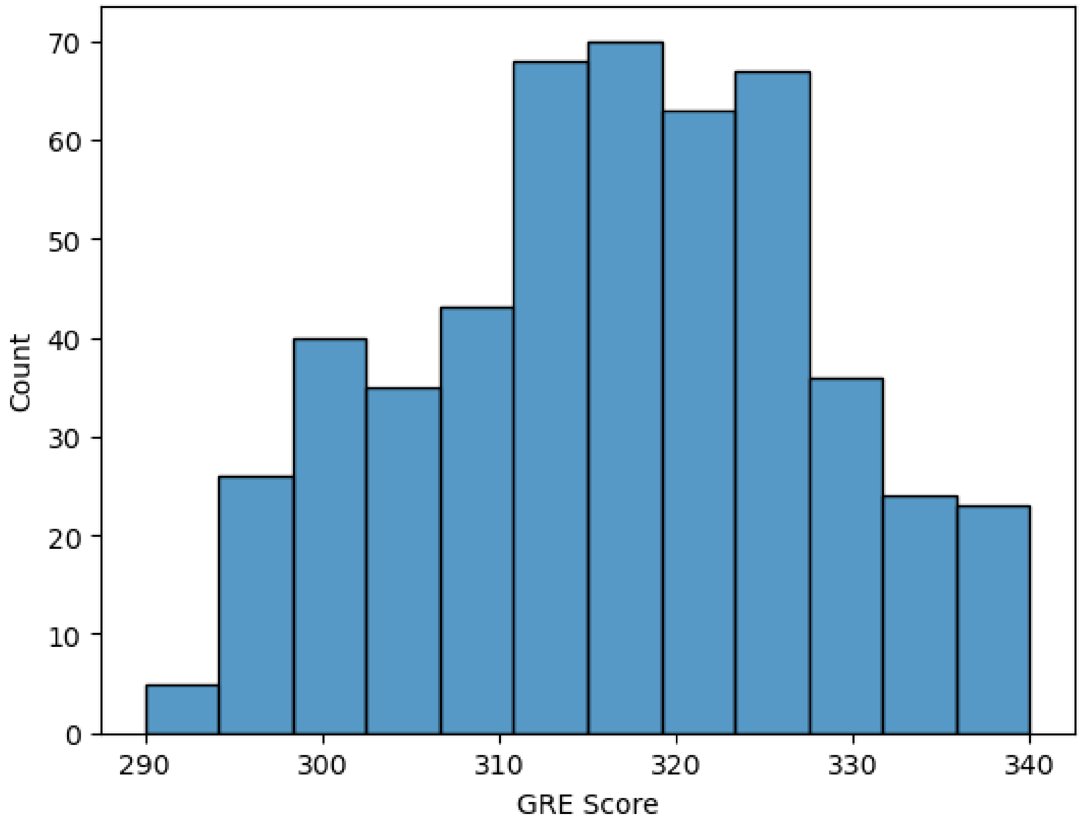




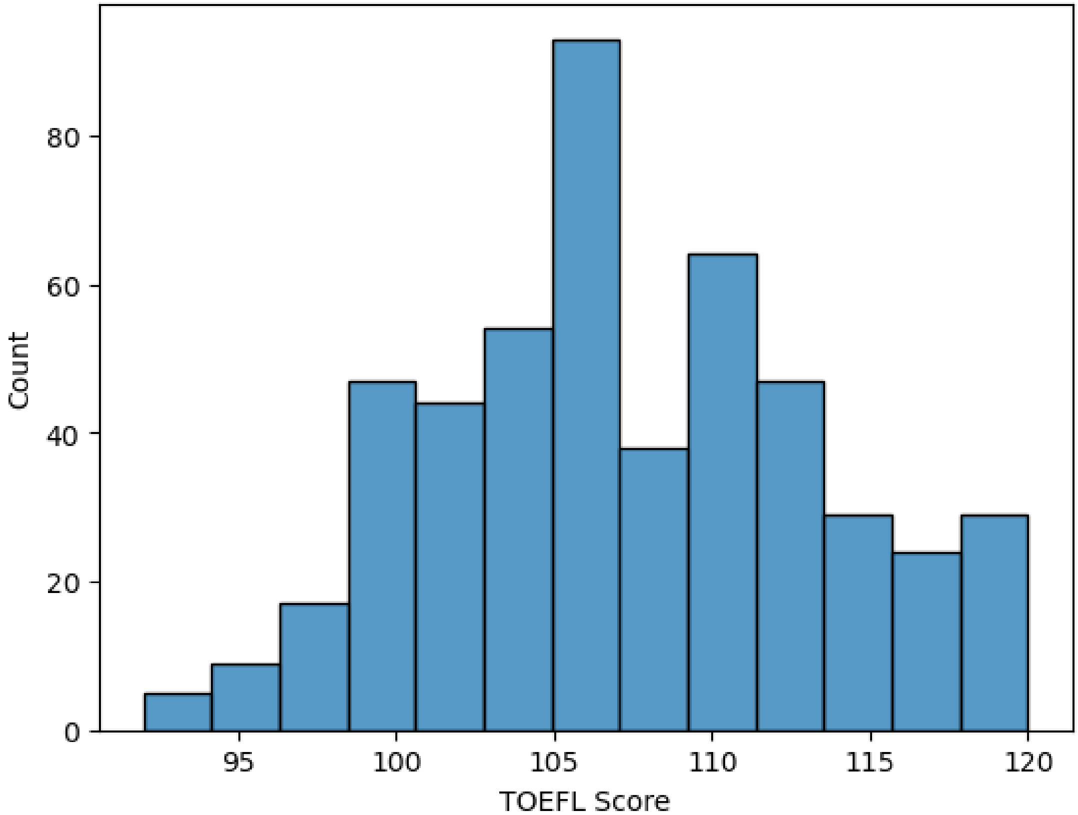


For Numerical features

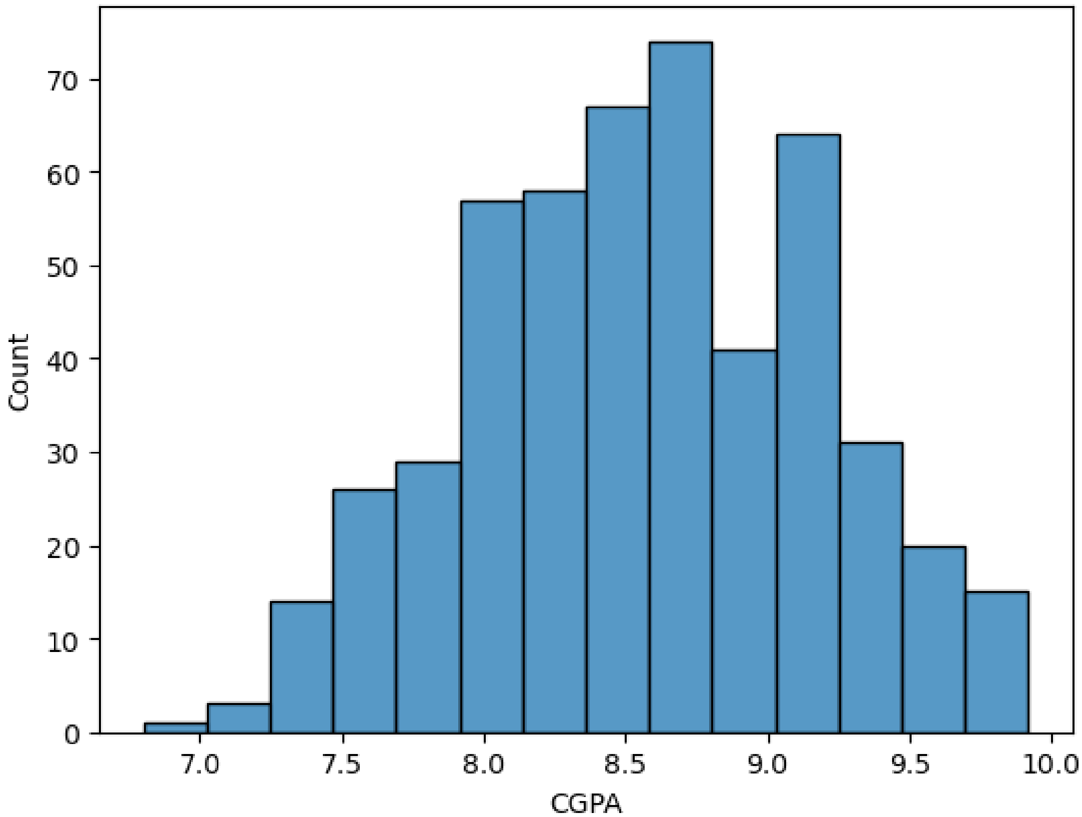
In [16]:
```
sns.histplot(x = 'GRE Score', data = JDF_data)
plt.show()
```



In [17]:
```
sns.histplot(x = 'TOEFL Score', data = JDF_data)
plt.show()
```



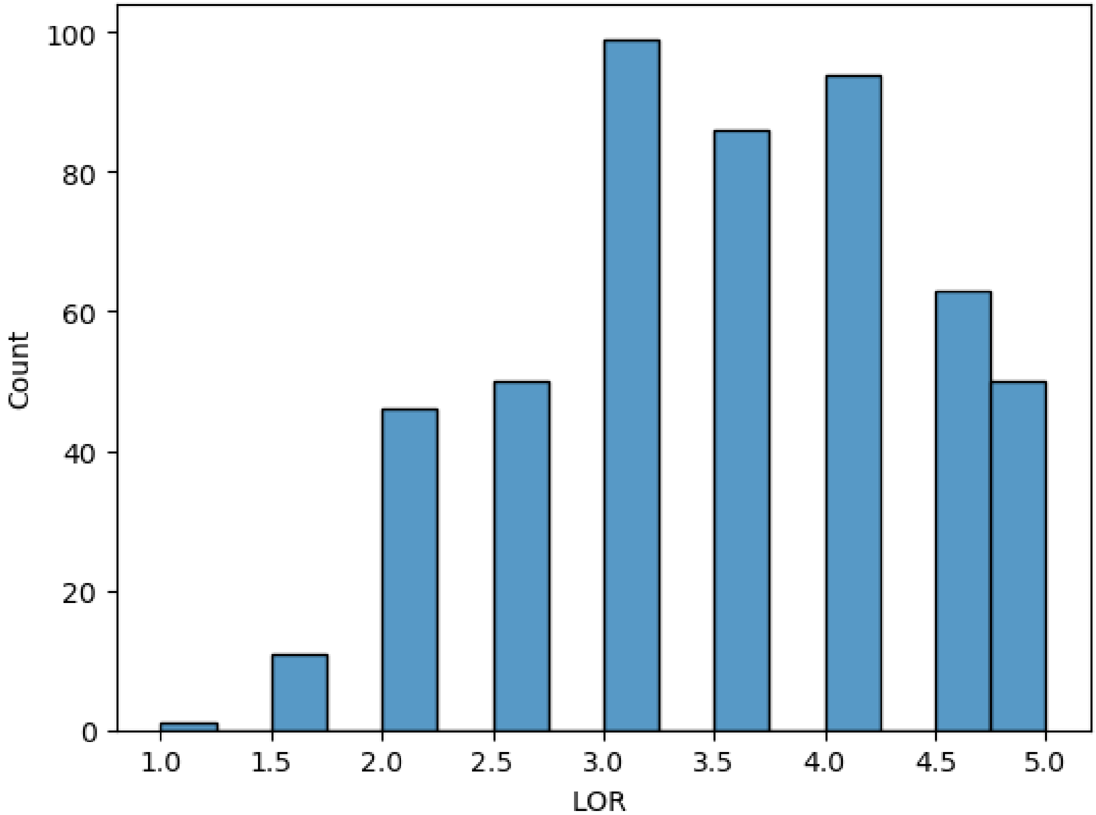
In [18]:
```
sns.histplot(x = 'CGPA', data = JDF_data)
plt.show()
```



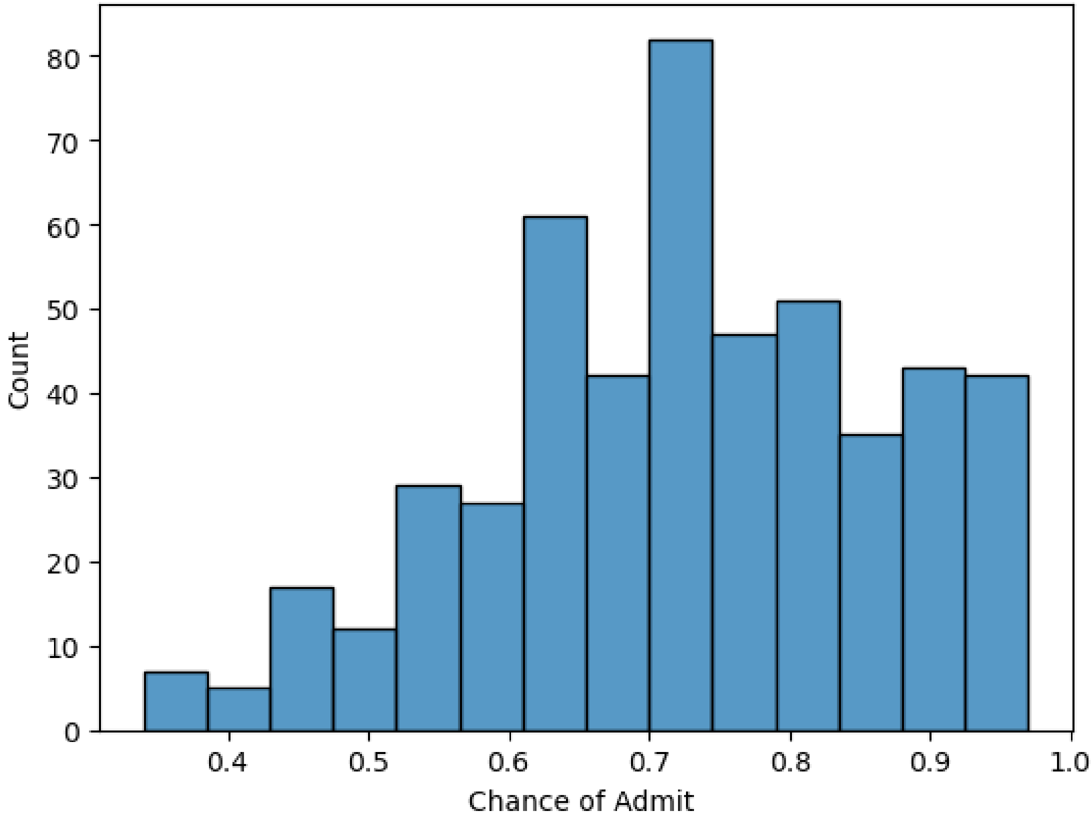
In [19]:
```
JDF_data.columns
```

Out[19]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

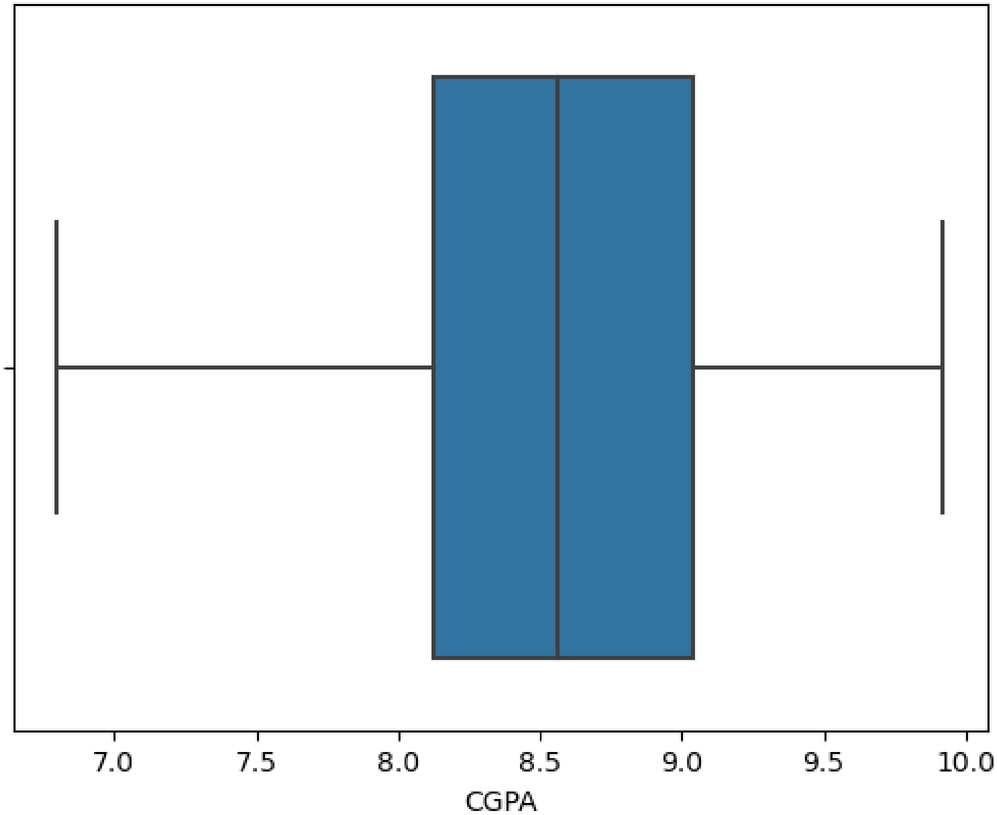
In [20]:
```
sns.histplot(x = 'LOR ', data = JDF_data)
plt.show()
```



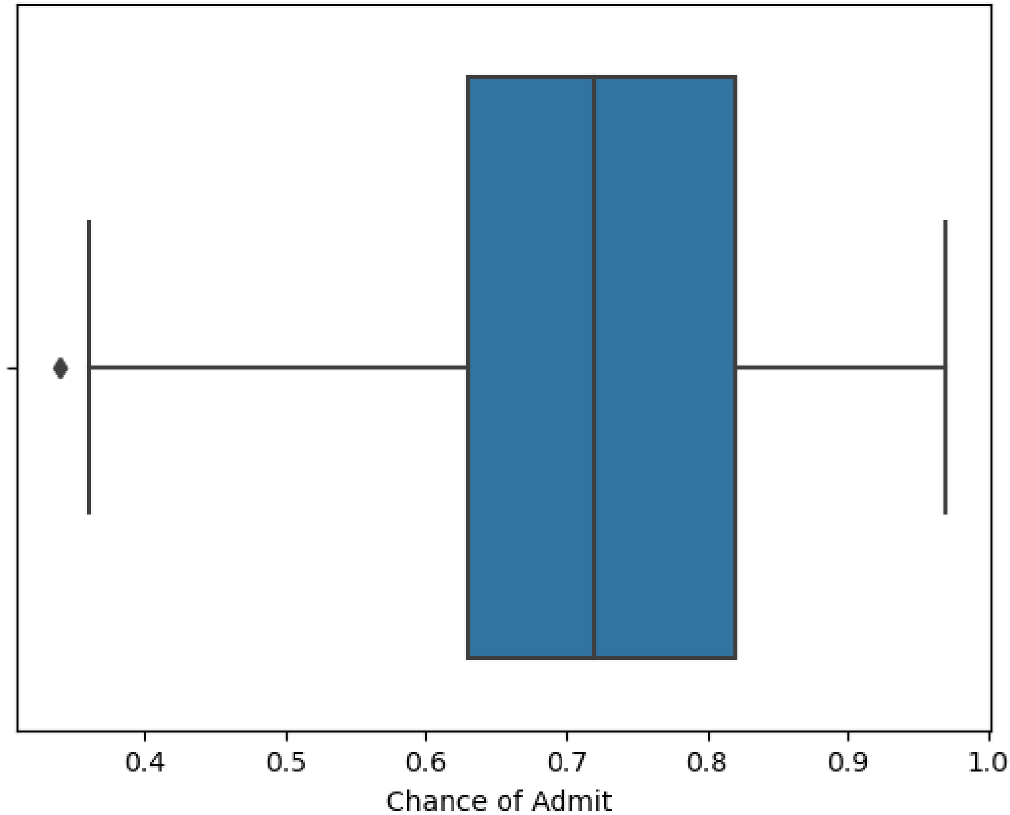
In [21]:
```
sns.histplot(x = 'Chance of Admit ', data = JDF_data)
plt.show()
```



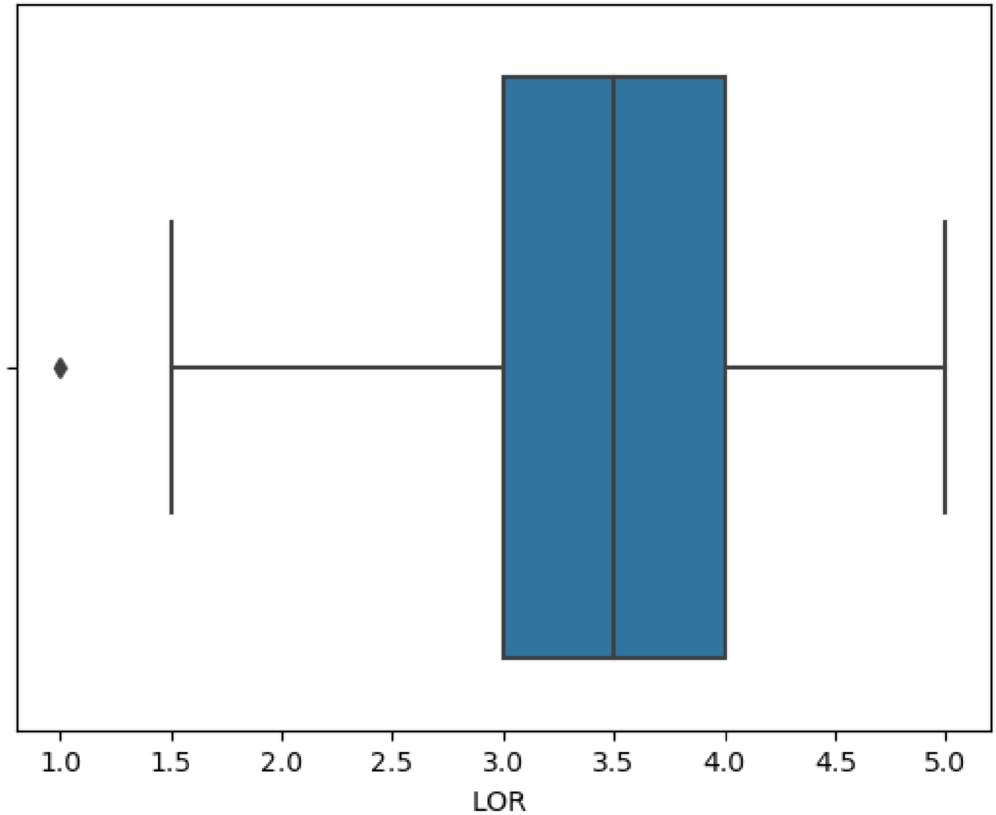
In [22]:
```
sns.boxplot(x = 'CGPA', data = JDF_data)
plt.show()
```



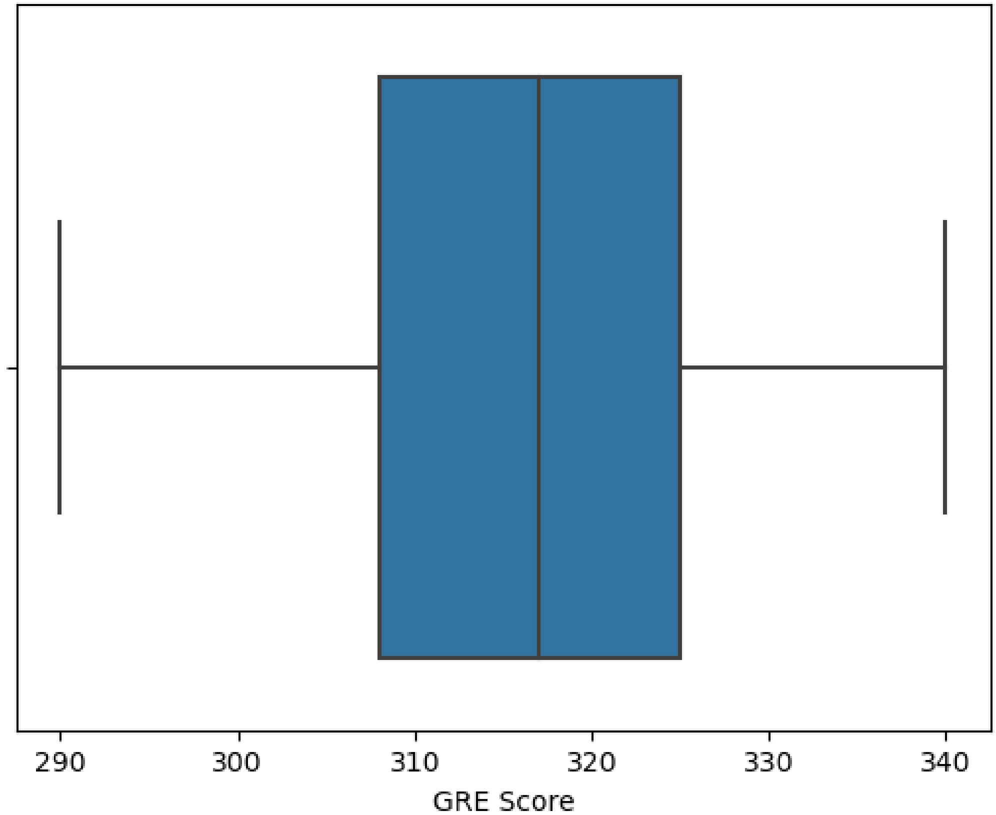
In [23]:
```
sns.boxplot(x = 'Chance of Admit ', data = JDF_data)
plt.show()
```



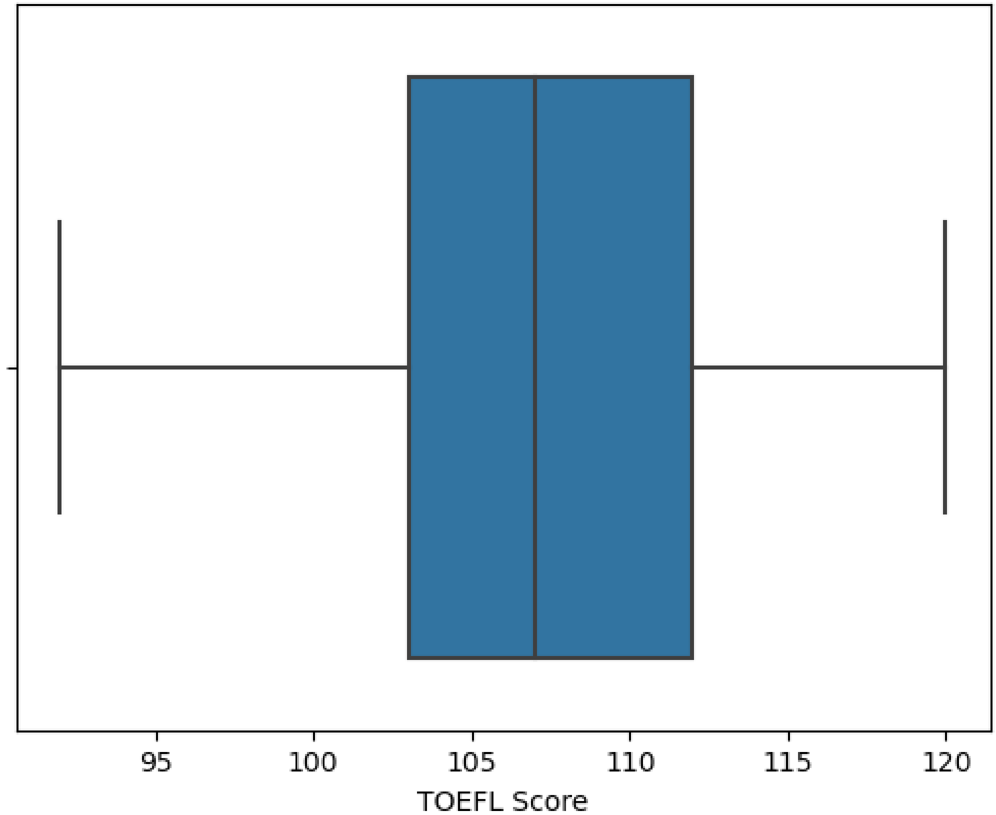
In [24]:
```
sns.boxplot(x = 'LOR ', data = JDF_data)
plt.show()
```



In [25]:
```
sns.boxplot(x = 'GRE Score', data = JDF_data)
plt.show()
```



In [26]:
```
sns.boxplot(x = 'TOEFL Score' , data = JDF_data)
plt.show()
```



Observation:

from univariate analysis, it is quite evident that University with rating 3 has maximum students, following other universities with rating 2 and 4.

SOP with 3.5 and 4.0 has maximum number of students.

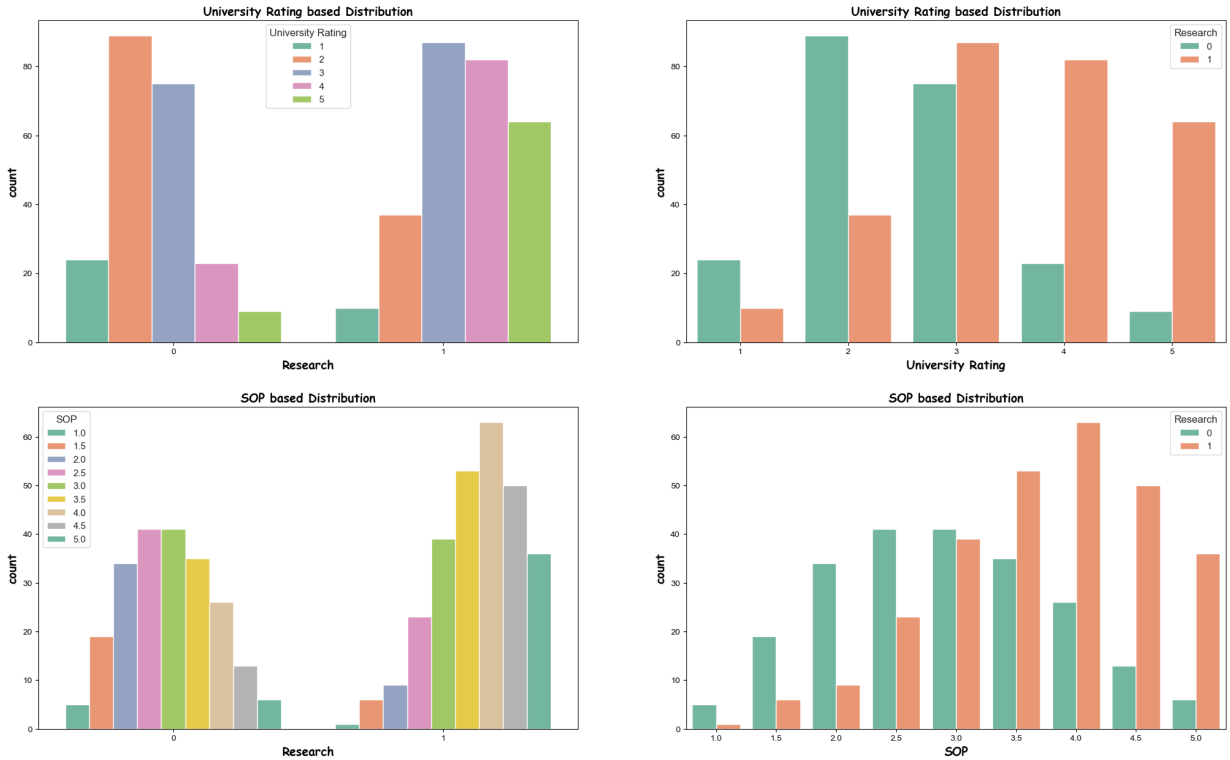
from the data there are more students whith research profile than non researchers.

we still need to see how these predictors effects our target variables for deriving any conclusions.

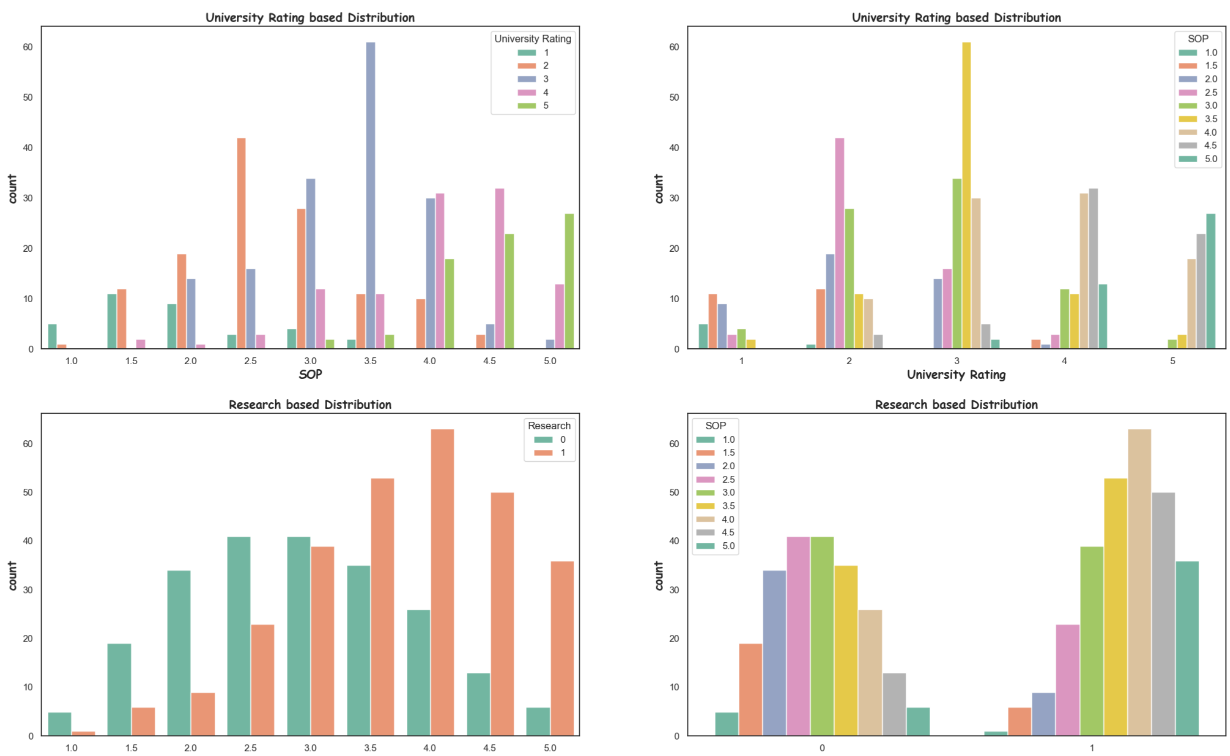
Bi-variate Analysis

Research vs University Rating and SOP

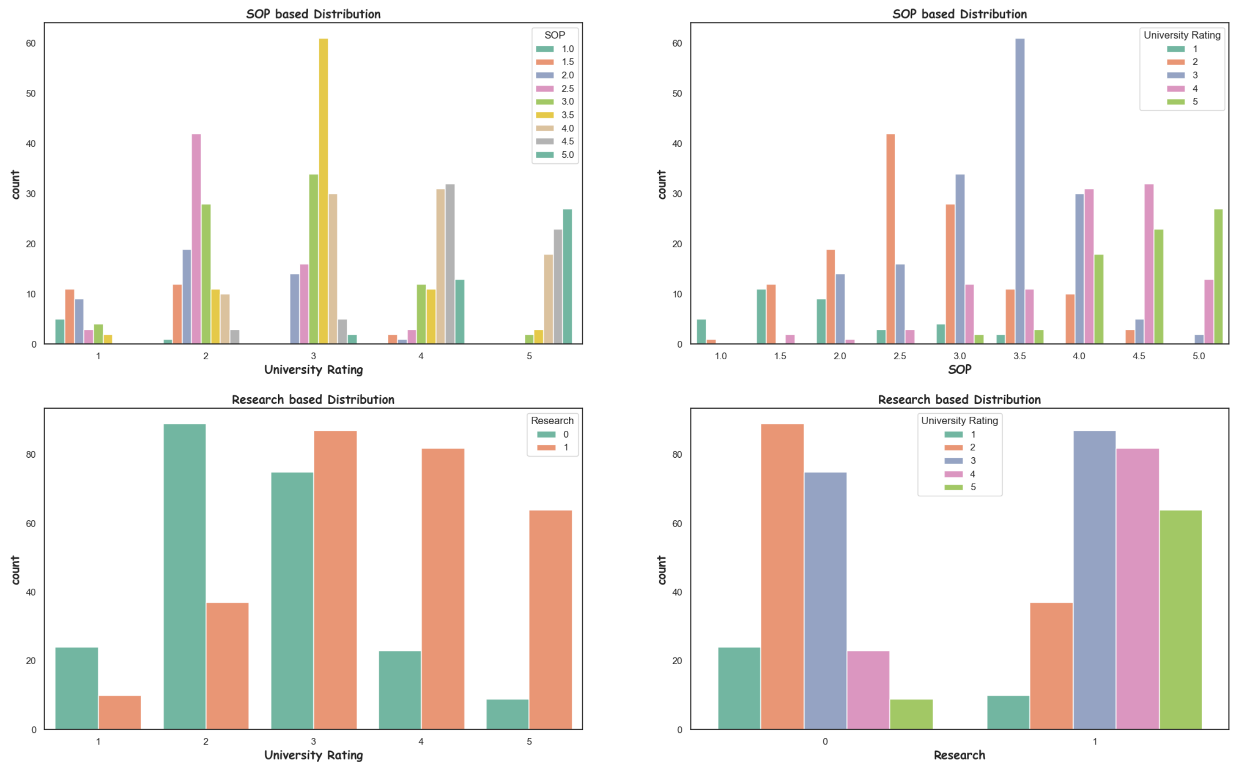
In [27]:
```
cat_cols = ['University Rating', 'SOP']
get_analysis_of_bivariate_cat(JDF_data, cat_cols, 'Research', 2, 2, 25, 15 )
```



In [28]:
```
cat_cols = ['University Rating', 'Research']
get_analysis_of_bivariate_cat(JDF_data, cat_cols, 'SOP', 2, 2, 25, 15 )
```



In [29]:
```
cat_cols = ['SOP', 'Research']
get_analysis_of_bivariate_cat(JDF_data, cat_cols, 'University Rating', 2, 2, 25, 1
```



Observation:

from the visual analysis, we can see that students from tier 3, 4 and 5 are with research profiles.

SOP ratings are higher for students with research profiles.

SOP with 3.5 ratings has the majortity and they are mostly from tier 3 universities.

from tier 1 and 2 universities, Non researchers are higher than researchers.

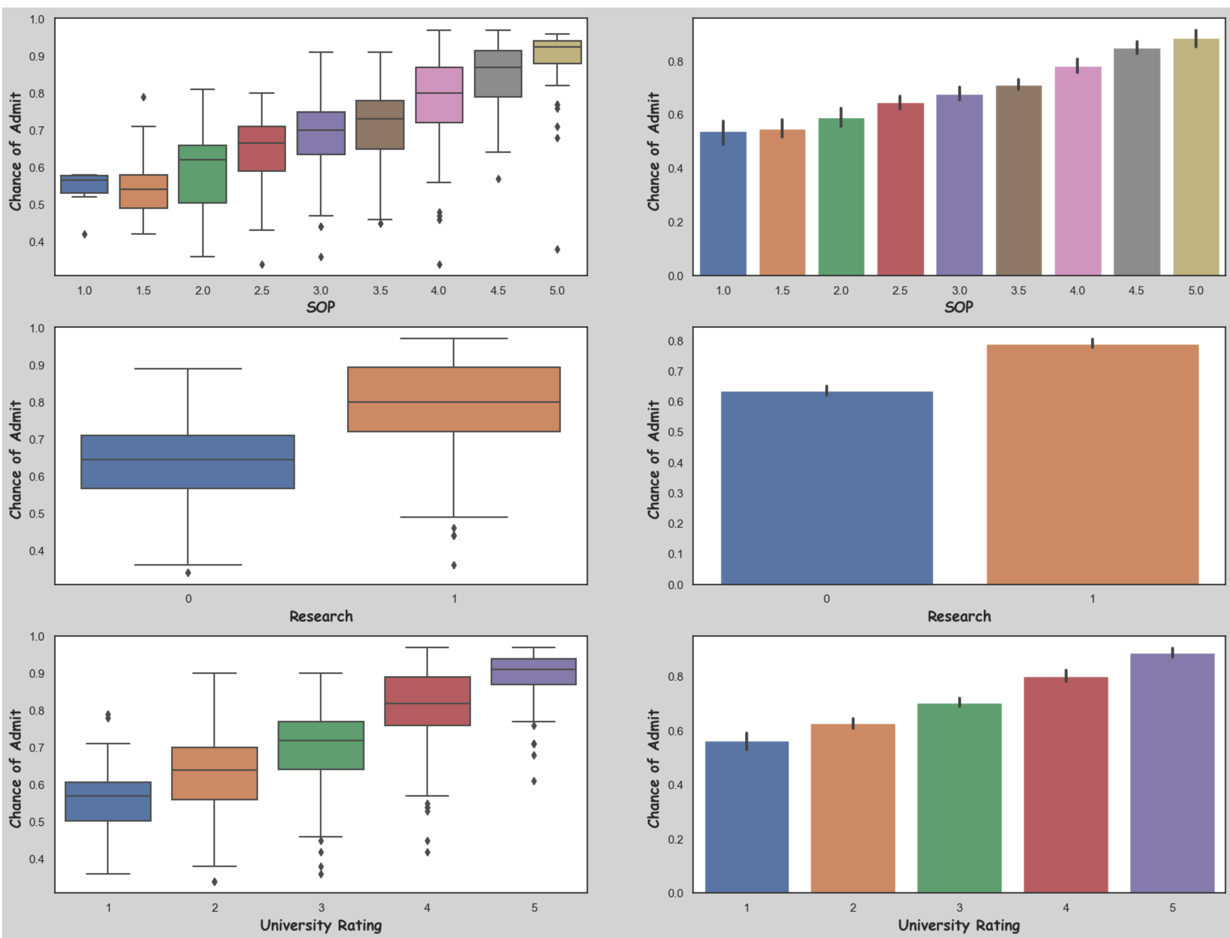
In [30]:
```
JDF_data.columns
```

Out[30]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

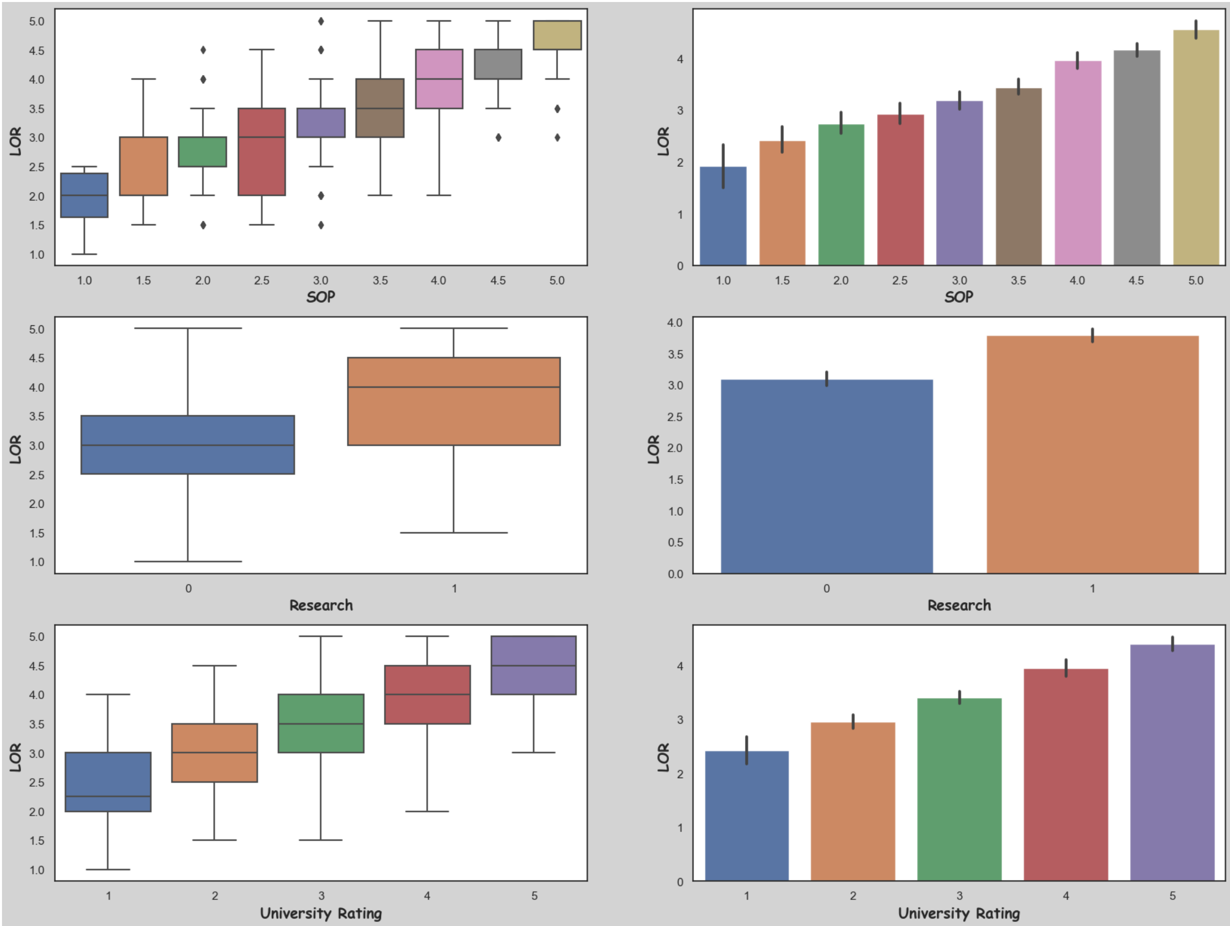
In [31]:
```
# bi-variate analysis for Categotical vs Numerical features
def get_bi_var_analysis_of_cat_num(df, colname, category, nrows=1, mcols=2, width=
    fig , ax = plt.subplots(nrows, mcols, figsize=(width,height), squeeze=False)
    sns.set(style='white')
    fig.set_facecolor("lightgrey")
    rows = 0
    for var in colname:

        # box plot with x, y and hue
        sns.boxplot(x = var, y = category, data = df, ax=ax[rows][0])

        # lineplot with x , y and hue
        sns.barplot(x = var, y = category, data = df, ax=ax[rows][1])

        ax[rows][0].set_ylabel(category, fontweight="bold", fontsize=14, family =
        ax[rows][0].set_xlabel(var, fontweight="bold", fontsize=14, family = "Comi
        ax[rows][1].set_ylabel(category, fontweight="bold", fontsize=14, family =
        ax[rows][1].set_xlabel(var, fontweight="bold", fontsize=14, family = "Comi
        rows += 1
    plt.show()
```

Categorical vs Numerical

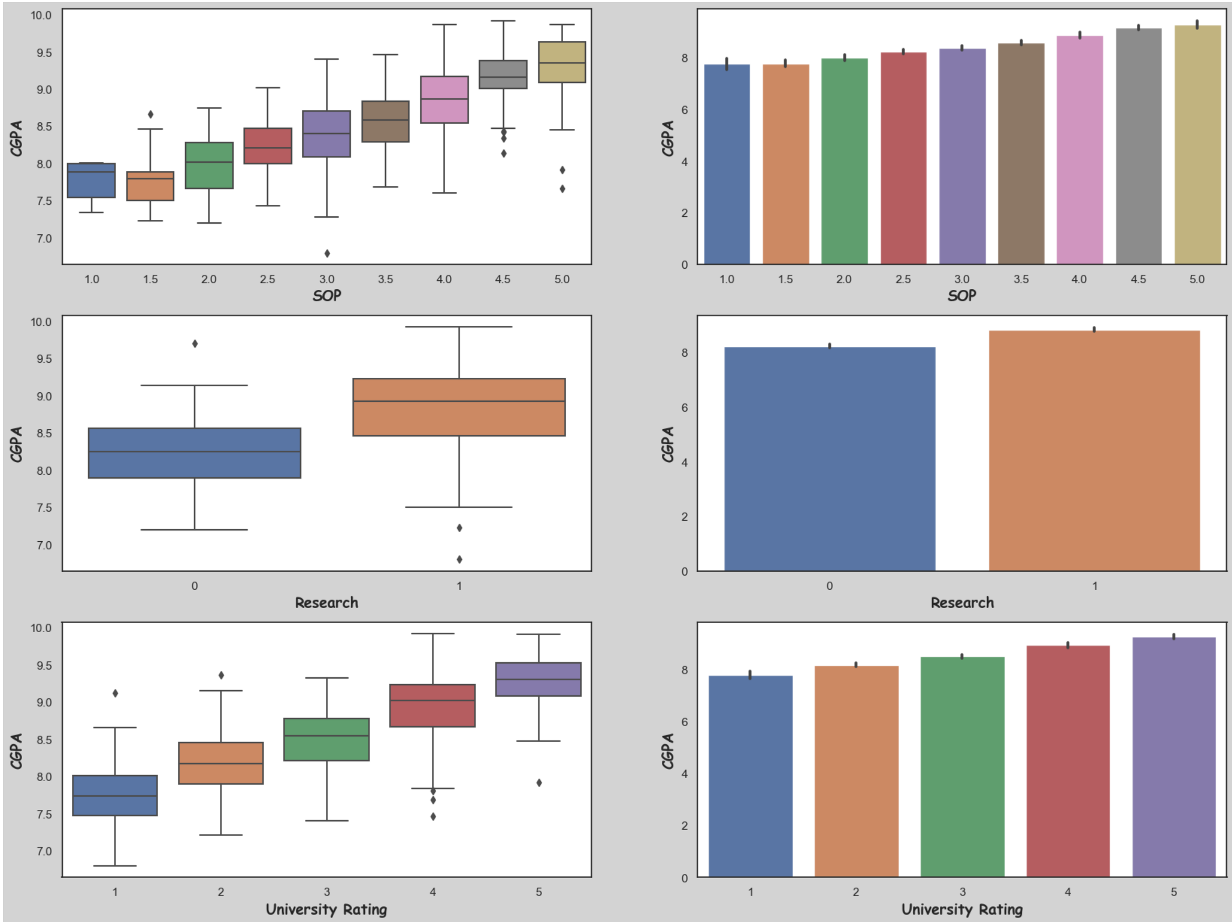
In [32]:
```
cat_cols = ['SOP', 'Research', 'University Rating']
get_bi_var_analysis_of_cat_num(JDF_data, cat_cols, 'Chance of Admit ', 3, 2, 20, 1
```



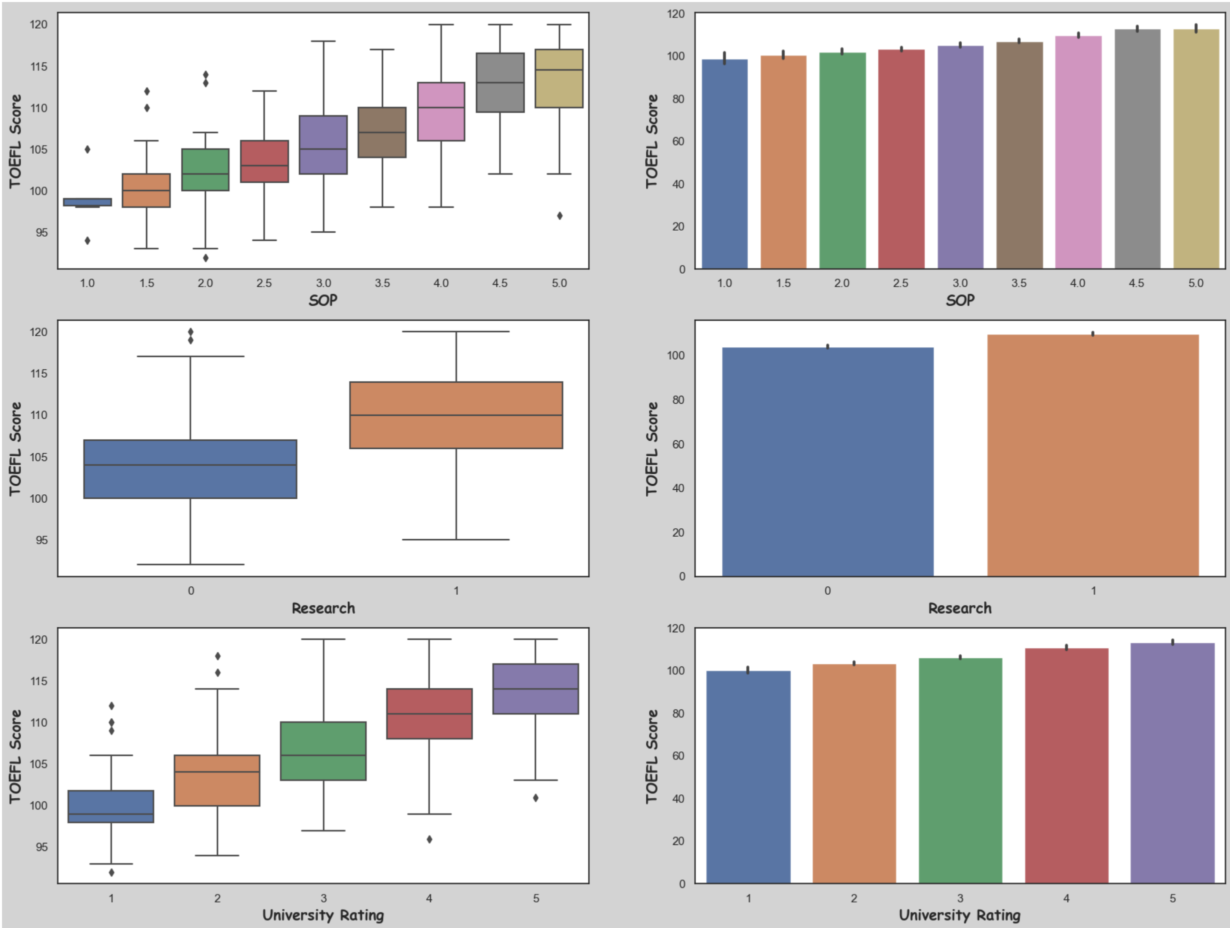
In [33]:
```
cat_cols = ['SOP', 'Research', 'University Rating']
get_bi_var_analysis_of_cat_num(JDF_data, cat_cols, 'LOR ', 3, 2, 20, 15)
```



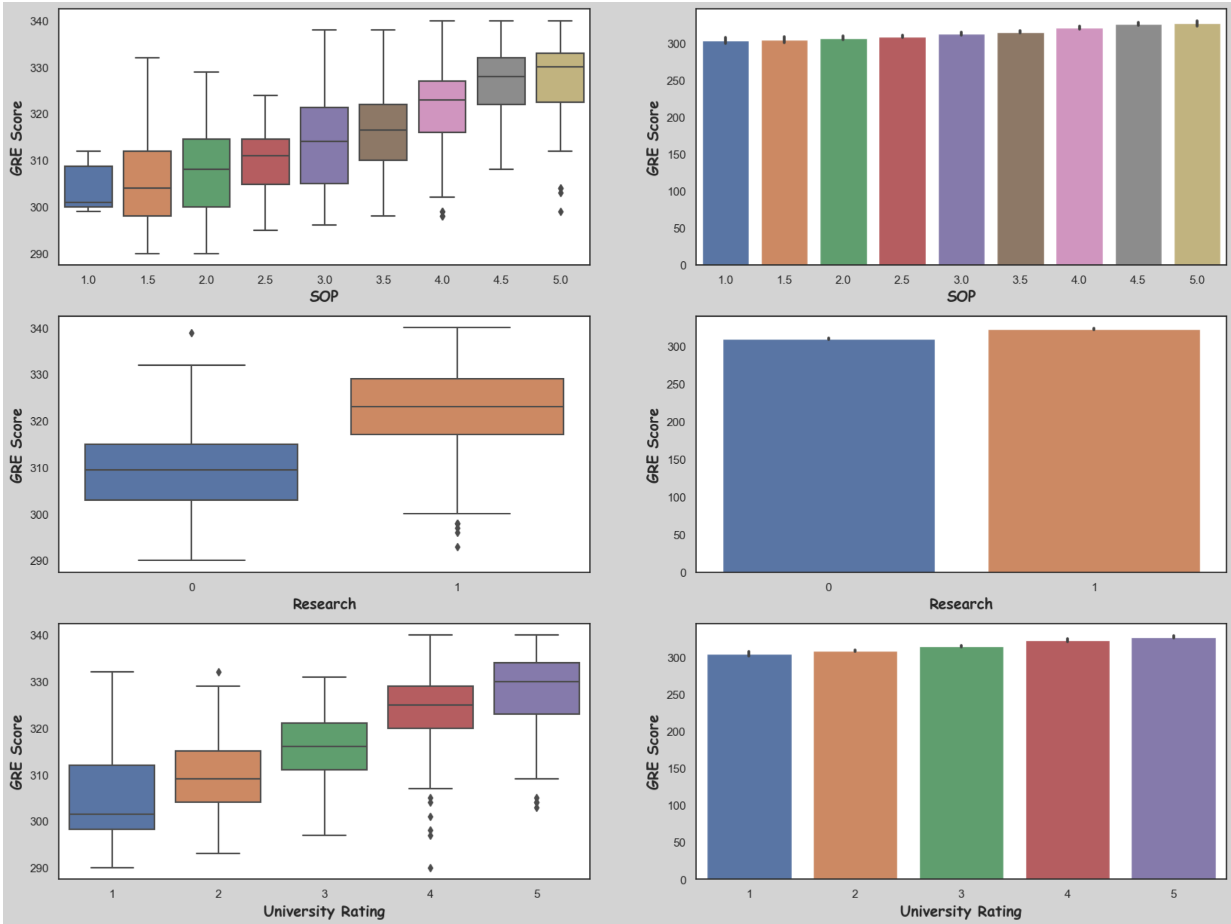
In [34]:
```
cat_cols = ['SOP', 'Research', 'University Rating']
get_bi_var_analysis_of_cat_num(JDF_data, cat_cols, 'CGPA', 3, 2, 20, 15)
```



In [35]:
```
cat_cols = ['SOP', 'Research', 'University Rating']
get_bi_var_analysis_of_cat_num(JDF_data, cat_cols, 'TOEFL Score', 3, 2, 20, 15)
```



In [36]:
```
cat_cols = ['SOP', 'Research', 'University Rating']
get_bi_var_analysis_of_cat_num(JDF_data, cat_cols, 'GRE Score', 3, 2, 20, 15)
```



Observations:

Its quites clear from analysis that higher the SOP higher are the chances of getting the profiles shortlisted for admission. so SOP seems to place very important role which is infact true.

Researcher profile's selection has higher median than Non researchers for chance of admission ratio.

University rating also seems to very important feature, as university rating increases higher of chance of getting to international college admission.

Higher the CGPA higher the SOP, seems there is positive correlation between these 2 features, we will find out later using Heat map.

CGPA and university ratings also has linear positive relationships. Students from tier 5 universities has highest score of CGPA. Researcher profile's students has higher TOEFL scores.

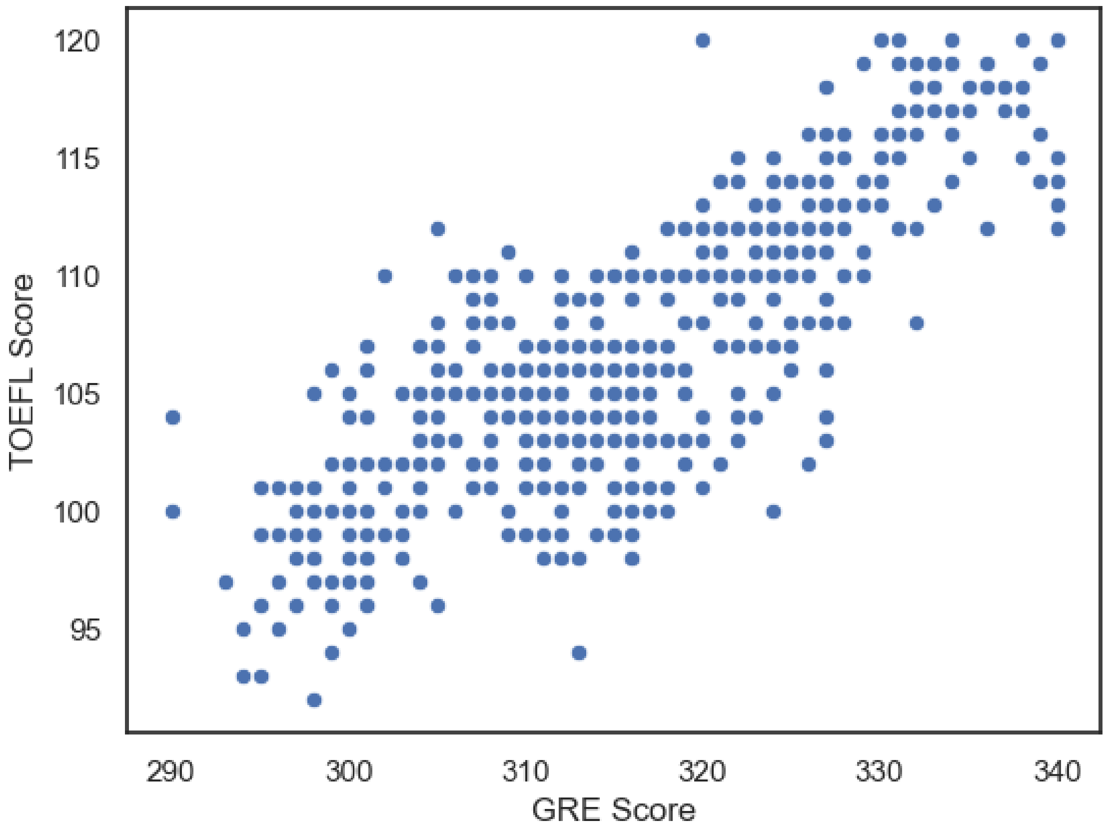
Higher scores in TOEFL belongs to tier 5 universties, again seems like positive correlationship.

Students with research profiles has scores higher in GRE exams.

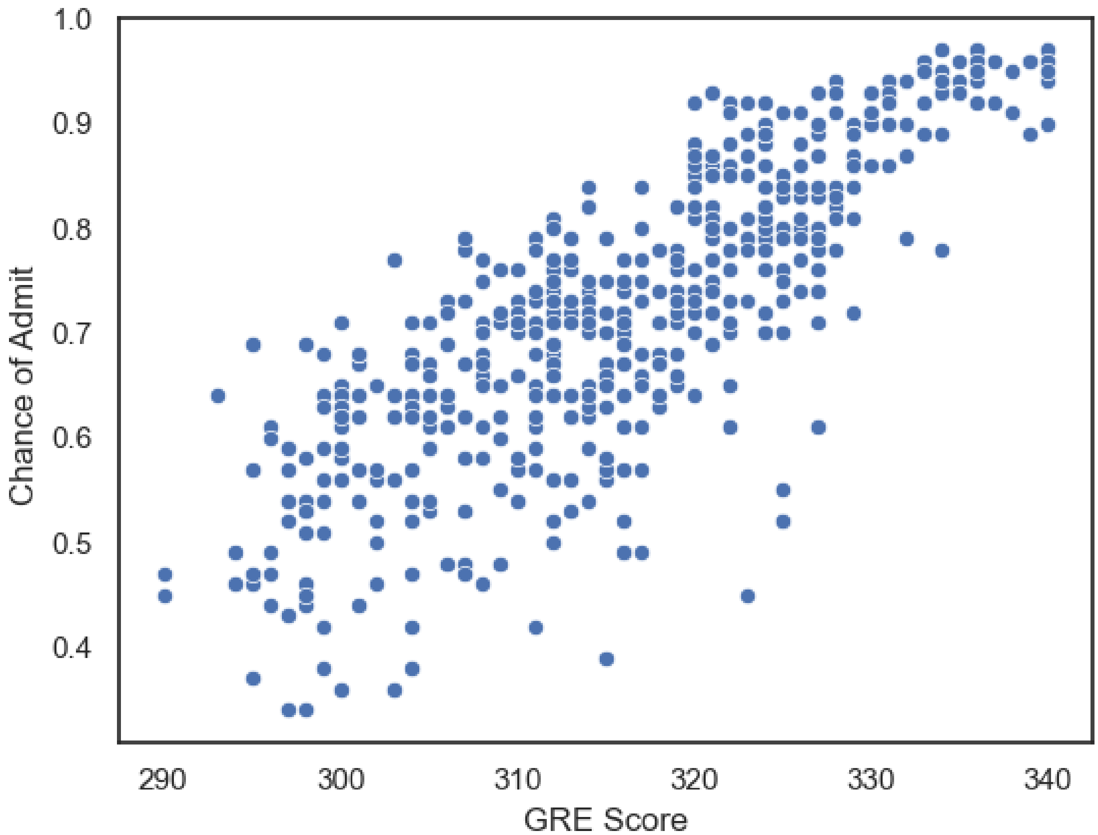
GRE score and universities rating also has linear relationship.

Numerical vs Numerical

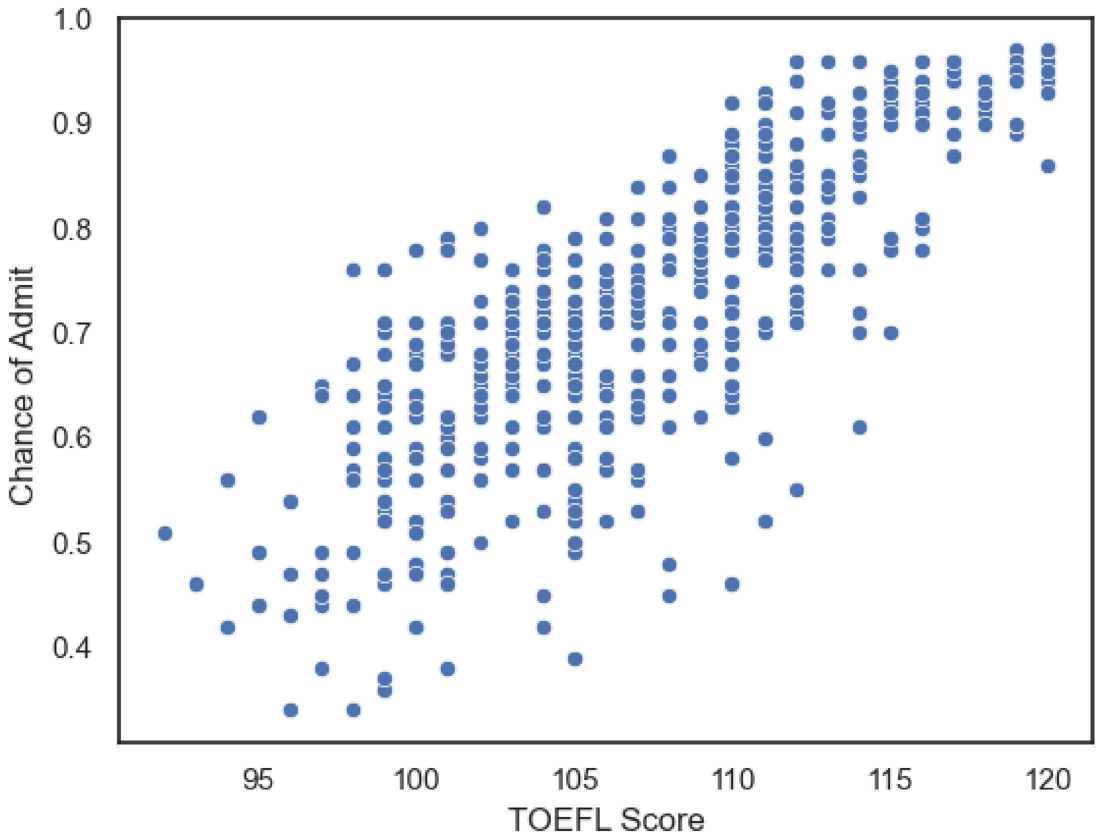
In [37]:
```
sns.scatterplot(x = 'GRE Score', y = 'TOEFL Score', data = JDF_data)
plt.show()
```



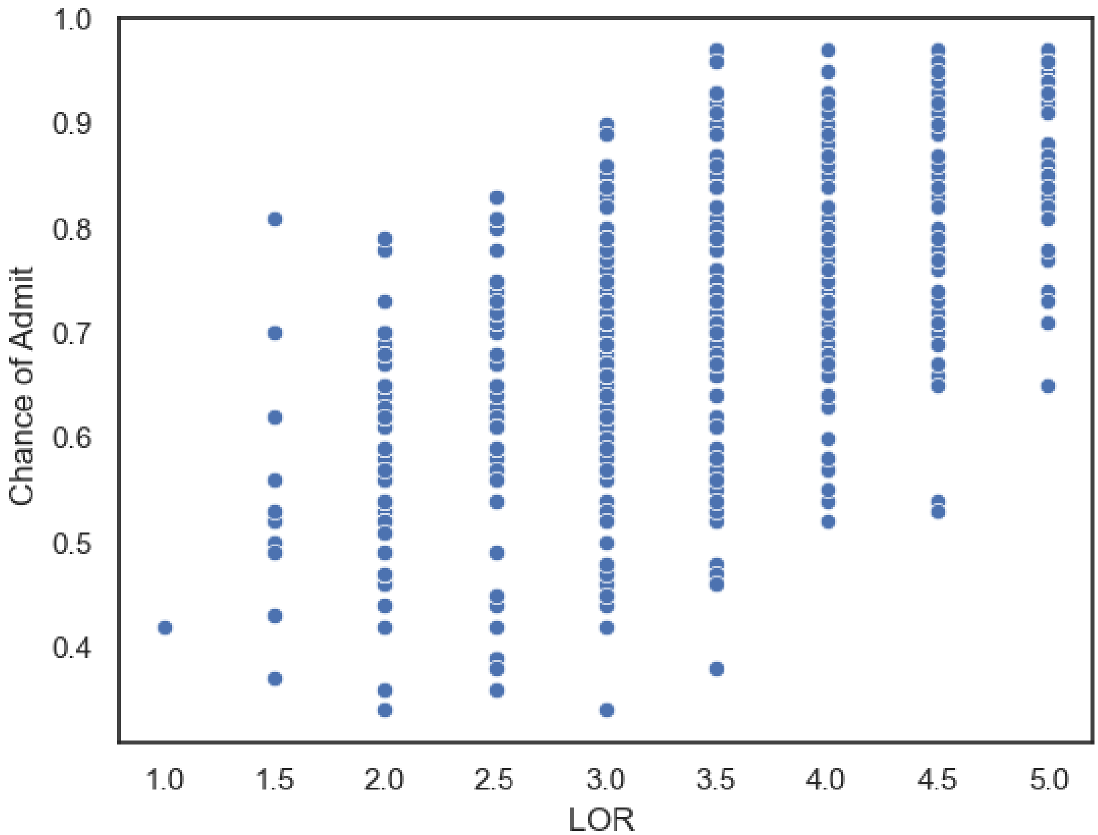
In [38]:
```
sns.scatterplot(x = 'GRE Score', y = 'Chance of Admit ', data = JDF_data)
plt.show()
```



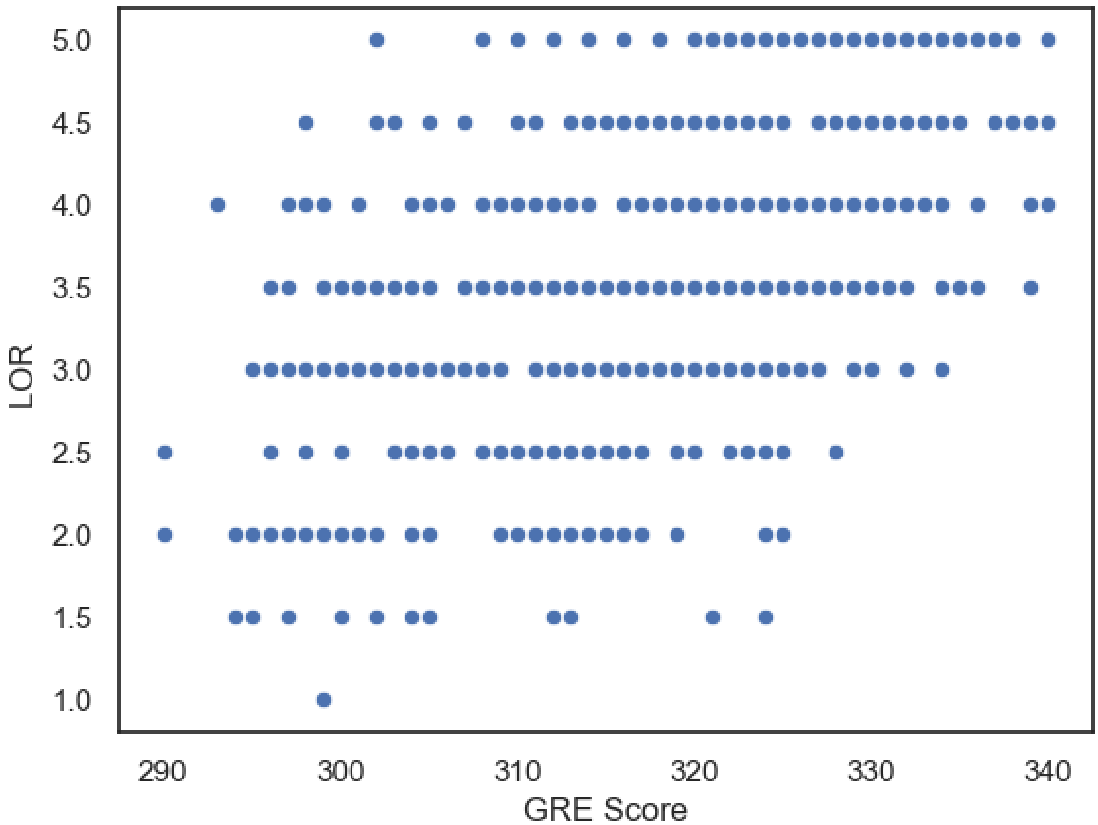
In [39]:
```
sns.scatterplot(x = 'TOEFL Score', y = 'Chance of Admit ', data = JDF_data)
plt.show()
```



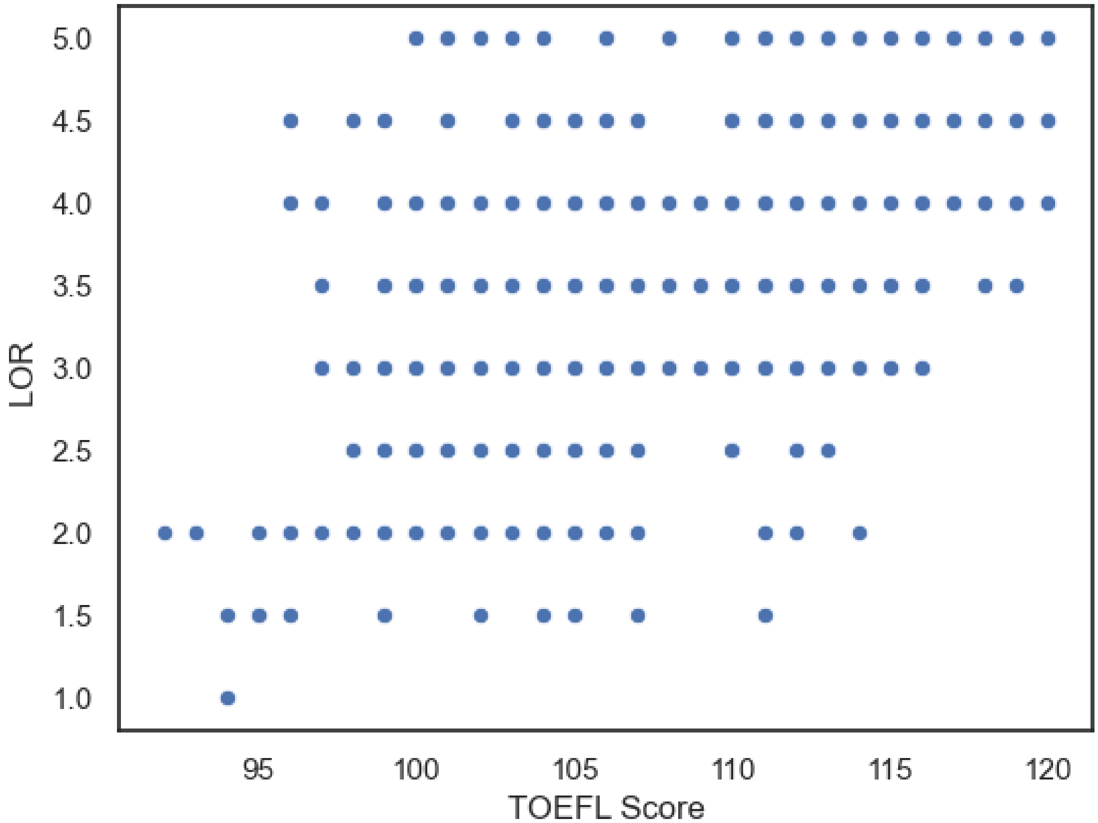
In [40]:
```
sns.scatterplot(x = 'LOR ', y = 'Chance of Admit ', data = JDF_data)
plt.show()
```



In [41]:
```
sns.scatterplot(x = 'GRE Score', y = 'LOR ', data = JDF_data)
plt.show()
```

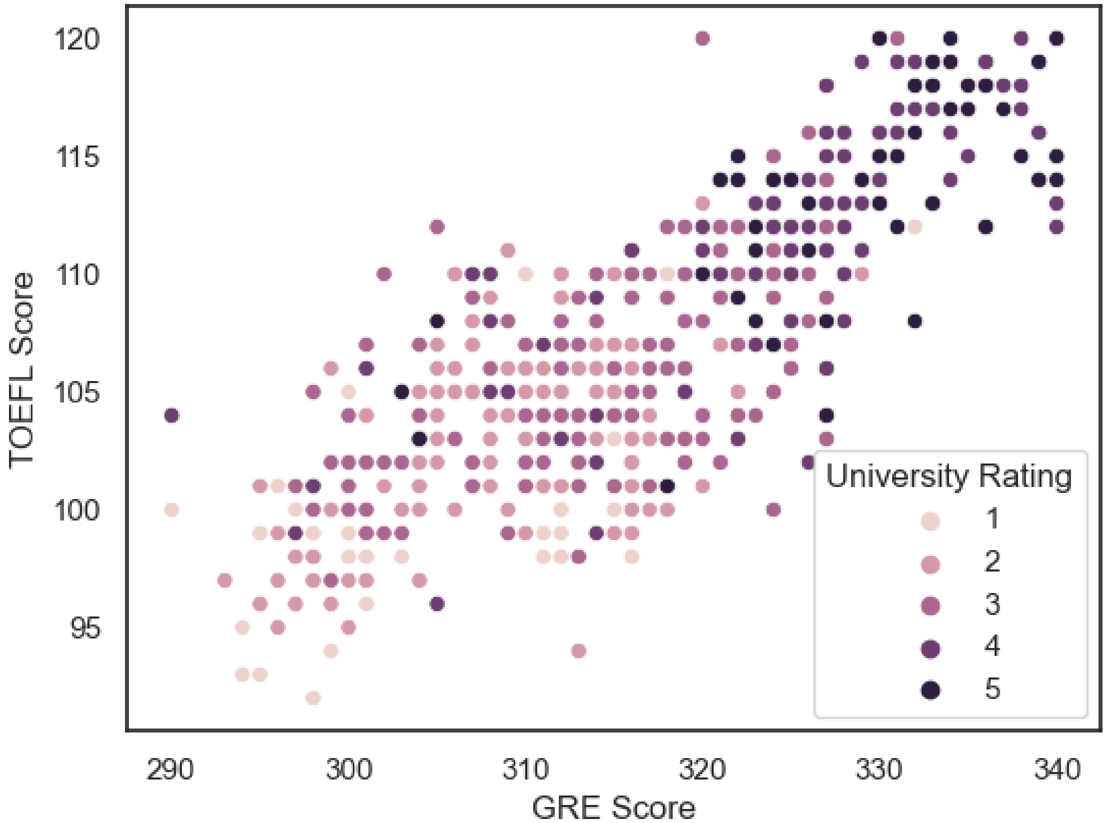


In [42]:
```
sns.scatterplot(x = 'TOEFL Score', y = 'LOR ', data = JDF_data)
plt.show()
```

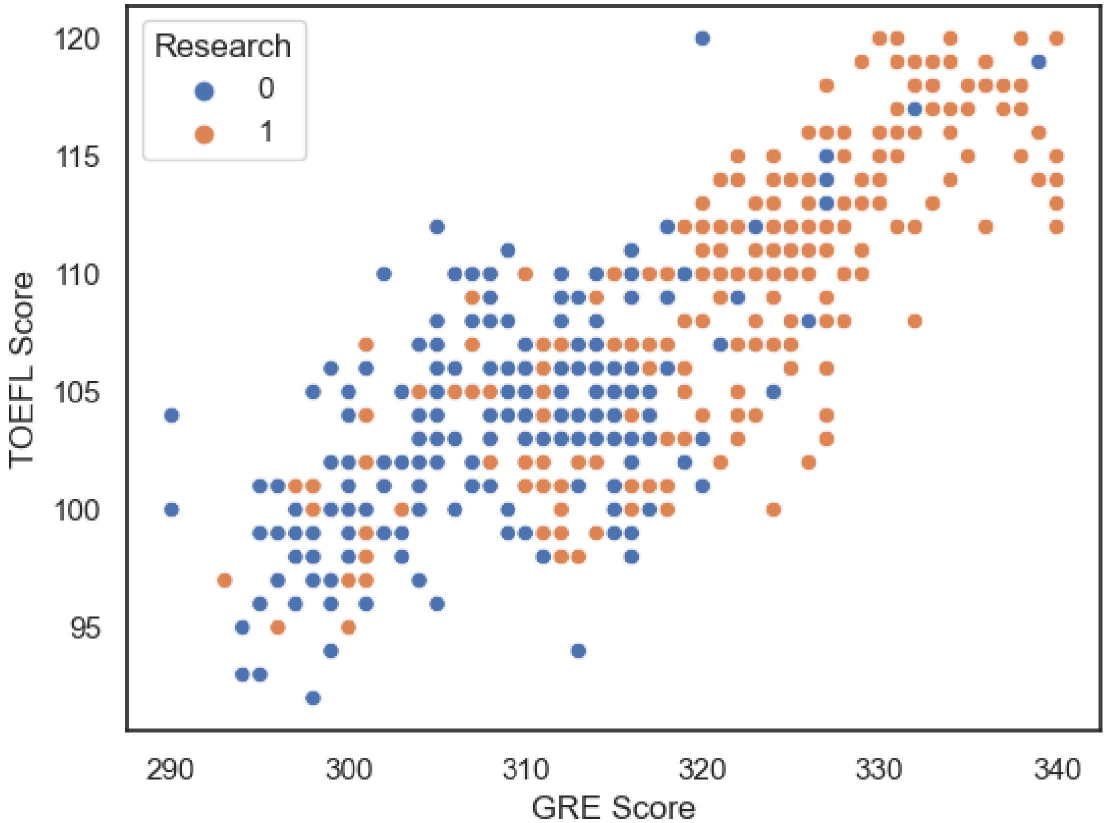


In [43]:
```
sns.scatterplot(x = 'GRE Score', y = 'TOEFL Score', hue = 'SOP', data = JDF_data)
plt.show()
```

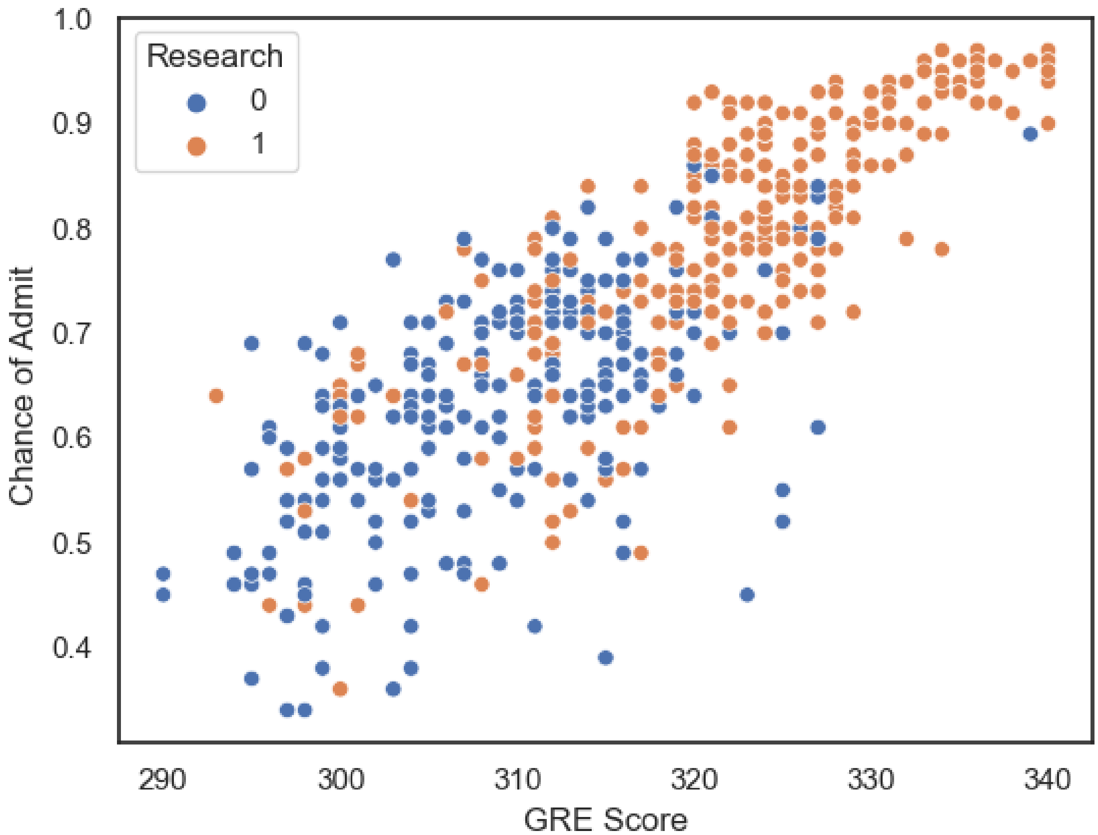
In [44]:
```
sns.scatterplot(x = 'GRE Score', y = 'TOEFL Score', hue = 'University Rating', data
plt.show()
```



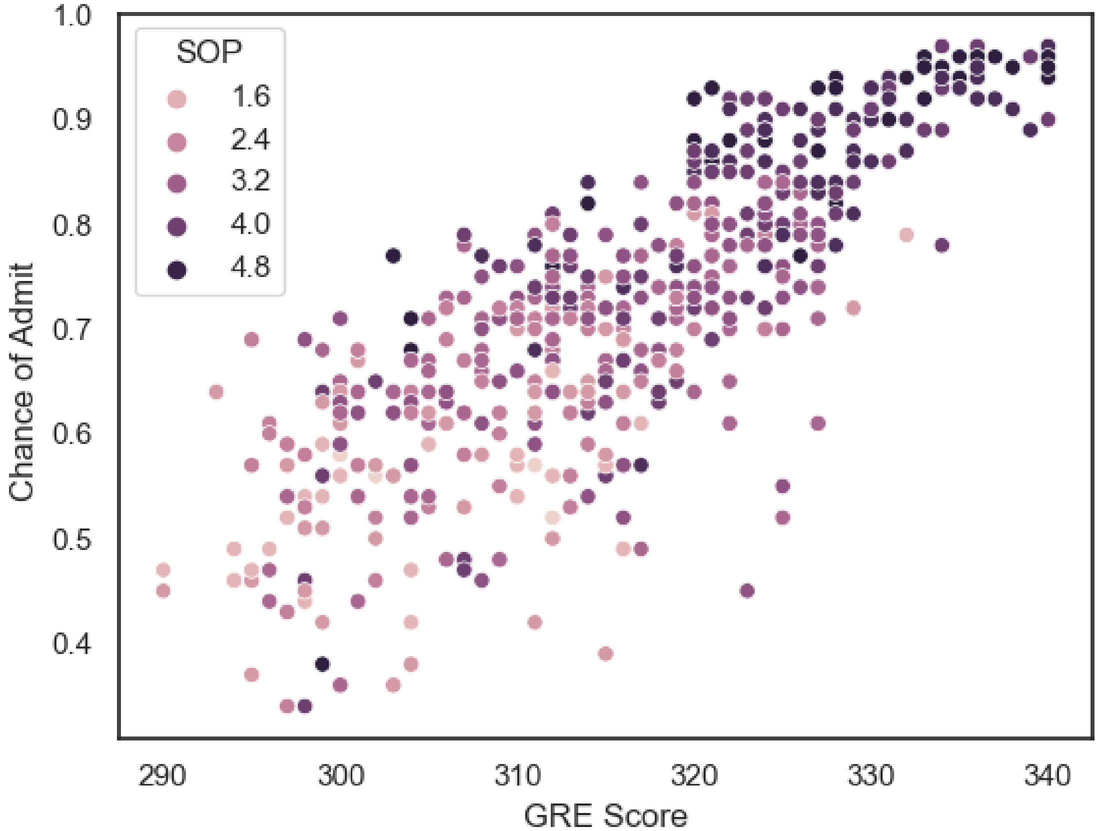
In [45]:
```
sns.scatterplot(x = 'GRE Score', y = 'TOEFL Score', hue = 'Research', data = JDF_d
plt.show()
```



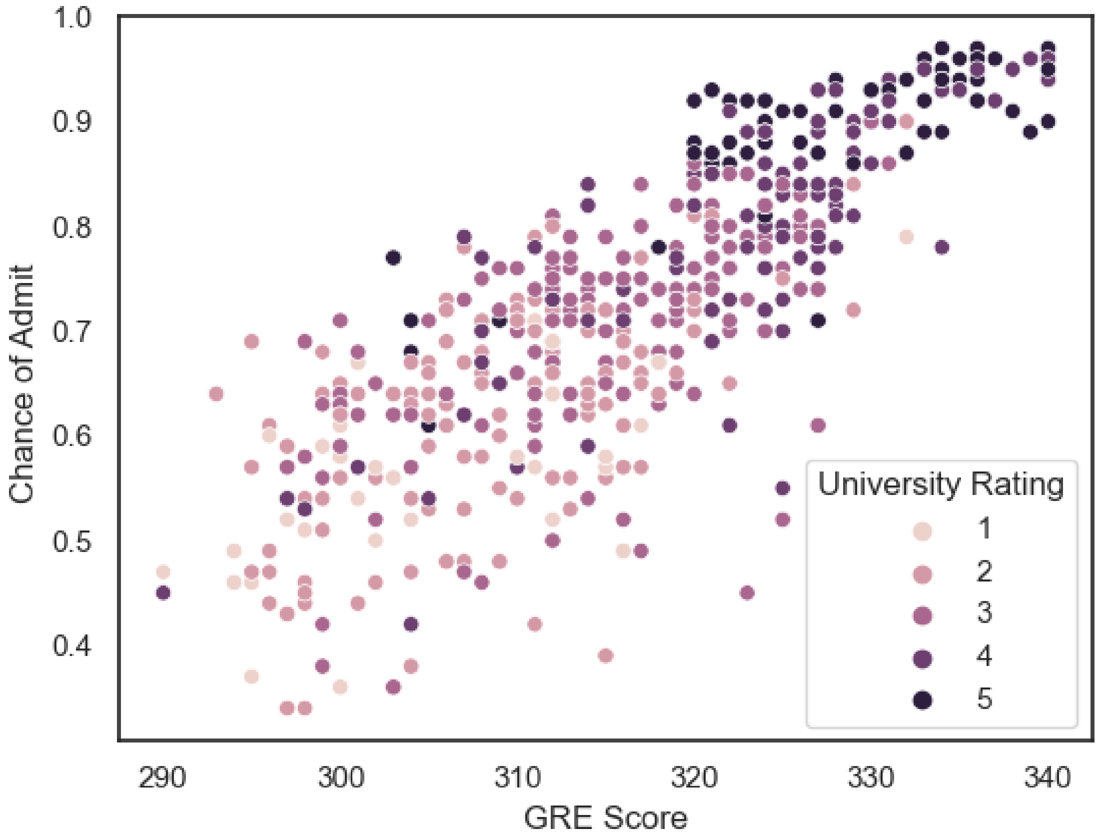
In [46]:
```
sns.scatterplot(x = 'GRE Score', y = 'Chance of Admit ', hue = 'Research', data = 
plt.show()
```



In [47]:
```
sns.scatterplot(x = 'GRE Score', y = 'Chance of Admit ', hue = 'SOP', data = JDF_da
plt.show()
```



In [48]:
```
sns.scatterplot(x = 'GRE Score', y = 'Chance of Admit ', hue = 'University Rating'
plt.show()
```



Observations:

from scatter plot, we can see linear relation between GRE and TOEFL score.

GRE score and chance of Admission has positive relation.

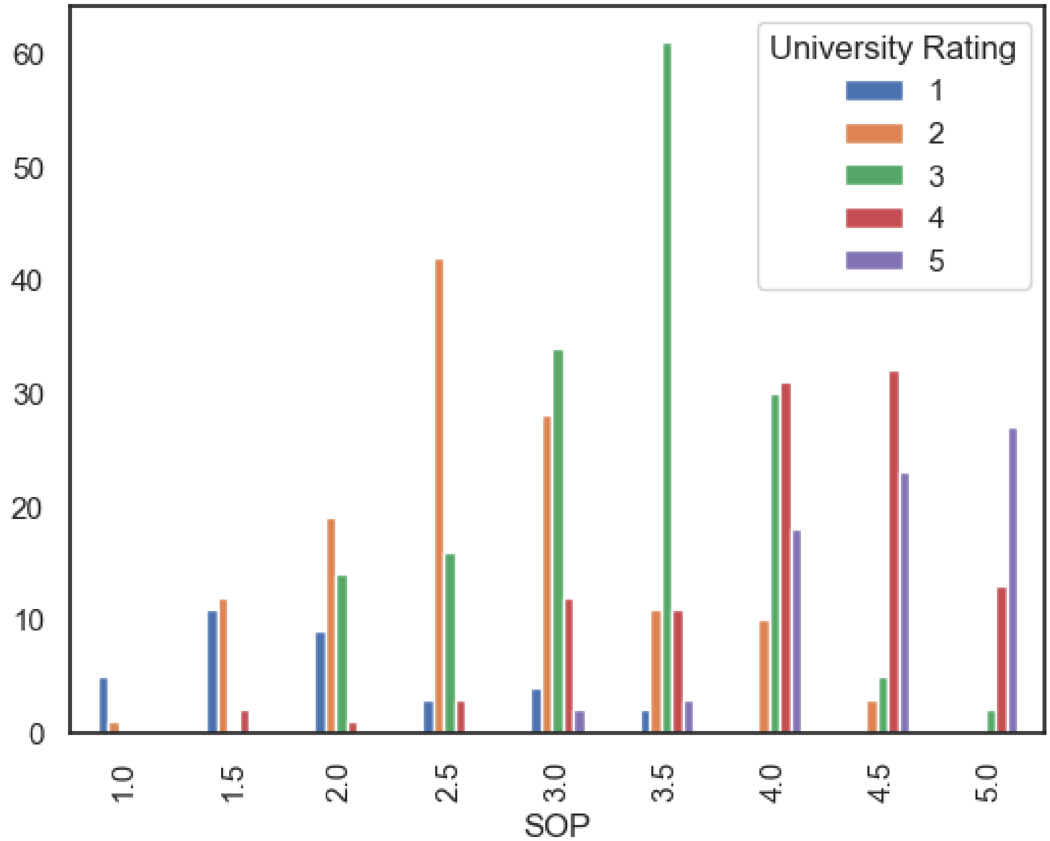
TOEFL and chance of Admit has positive relatiion but not in straight line.

Students with Higher GRE score and research profiles has higher chance of admission as per the Multivariate analysis.

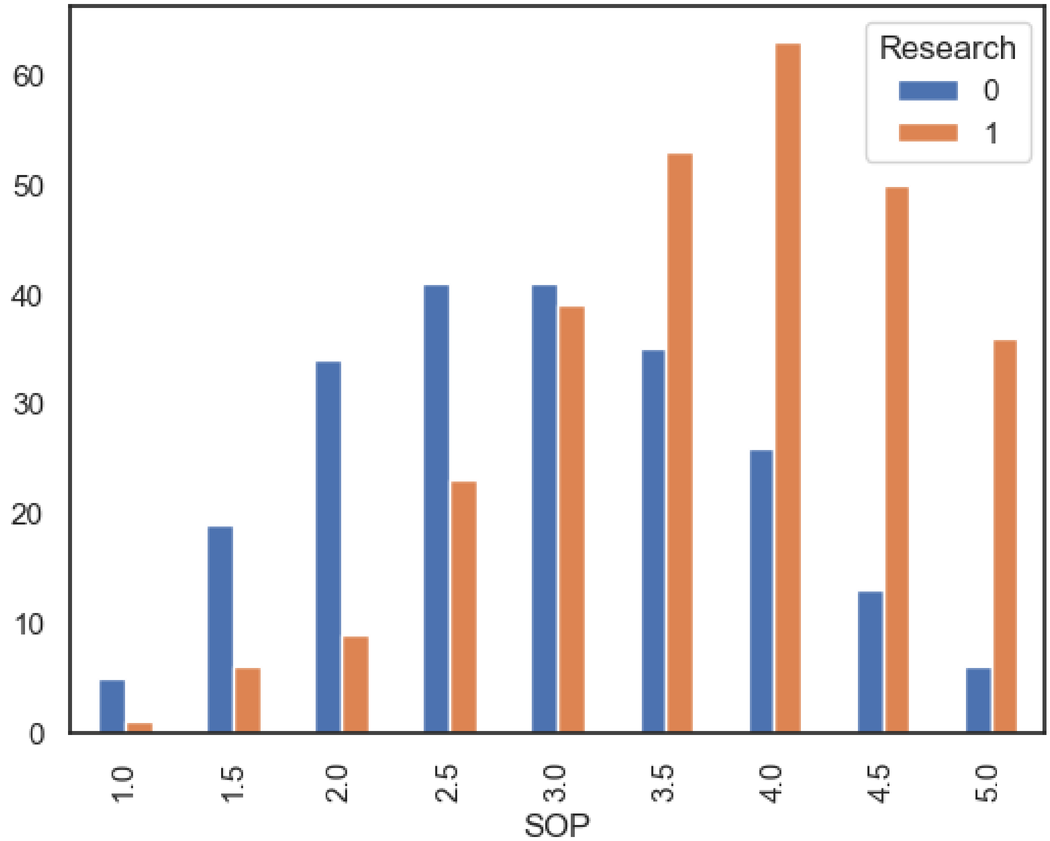
Students with Higher GRE score and higher SOP has higher chance of admission as per the Multivariate analysis but not always.

Categorical vs Categorical

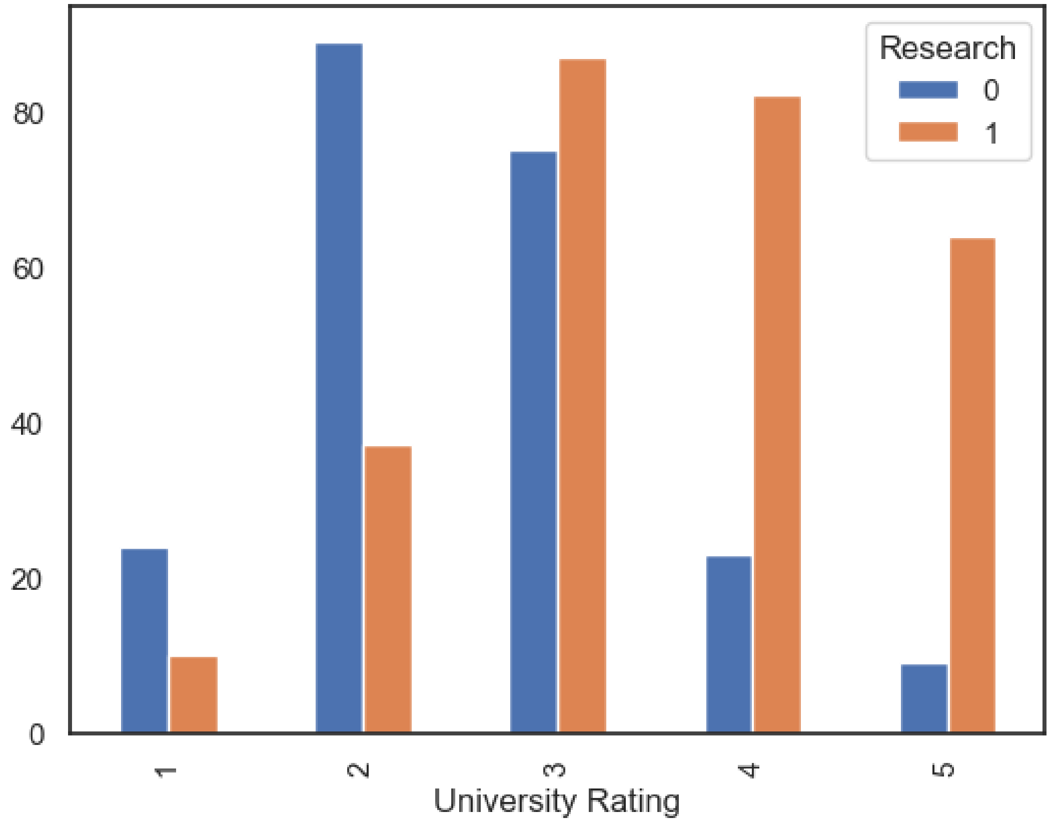
In [49]:
```
JDF_cross = pd.crosstab(JDF_data['SOP'], JDF_data['University Rating'])
JDF_cross.plot(kind = 'bar')
plt.show()
```



In [50]:
```
JDF_cross = pd.crosstab(JDF_data['SOP'], JDF_data['Research'])
JDF_cross.plot(kind = 'bar')
plt.show()
```



In [51]:
```
JDF_cross = pd.crosstab(JDF_data['University Rating'], JDF_data['Research'])
JDF_cross.plot(kind = 'bar')
plt.show()
```



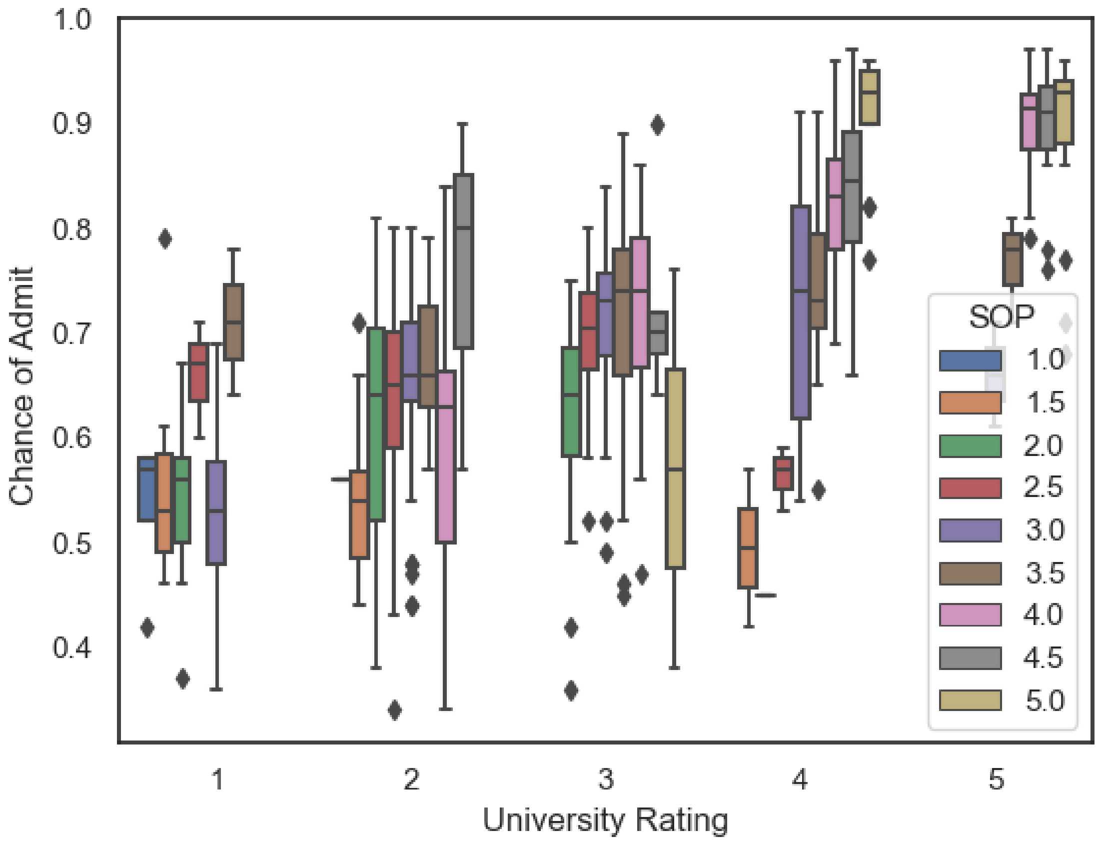
Observation:

Research profiles candidates are most likely from tier 3 , 4 or 5 rated universities.

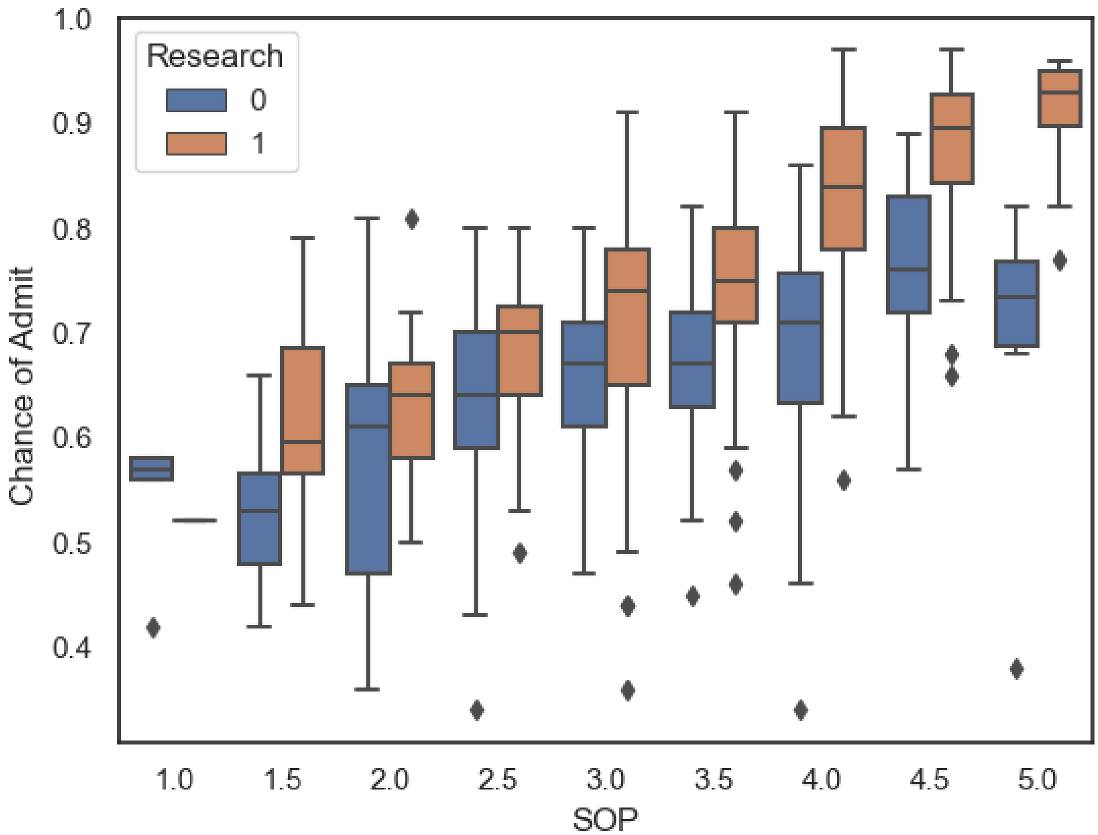
Candidates who has research profiles tends to have higher SOP as well.

Multivariate Analysis

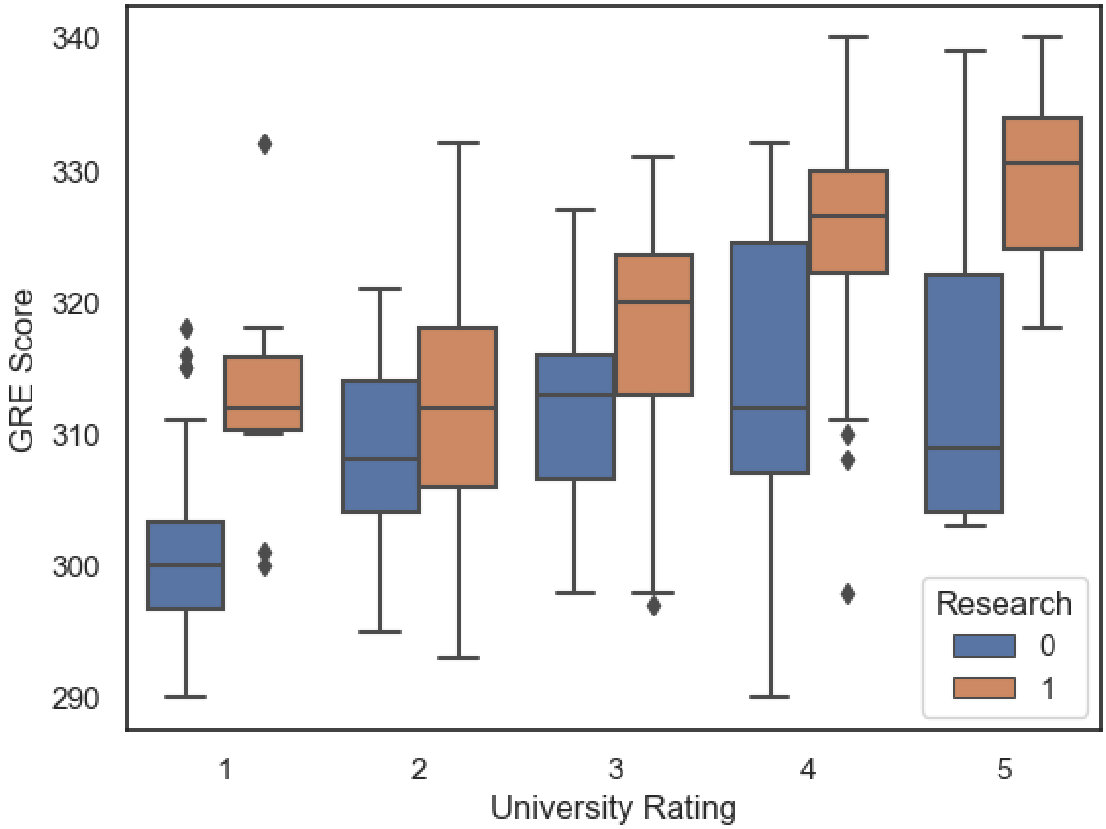
In [52]:
```
sns.boxplot(x = 'University Rating', y = 'Chance of Admit ', hue = 'SOP', data = J
plt.show()
```



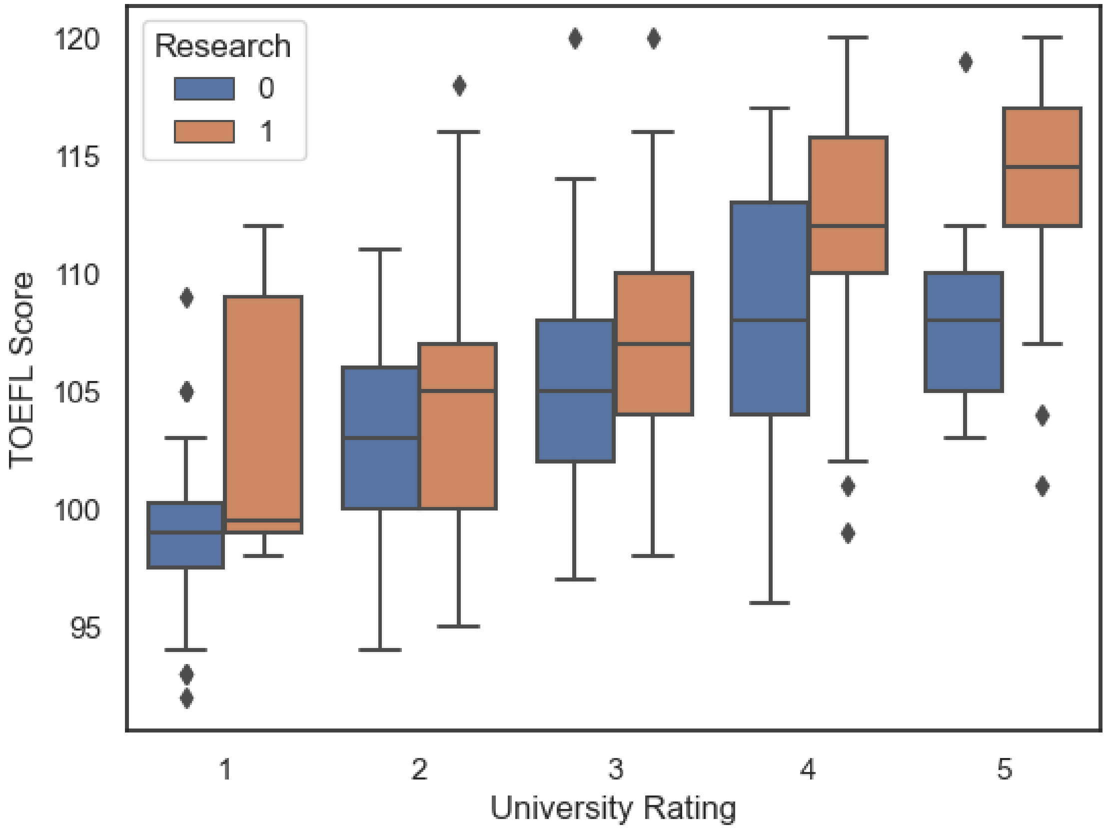
In [53]:
```
sns.boxplot(x = 'SOP', y = 'Chance of Admit ', hue = 'Research', data = JDF_data)
plt.show()
```



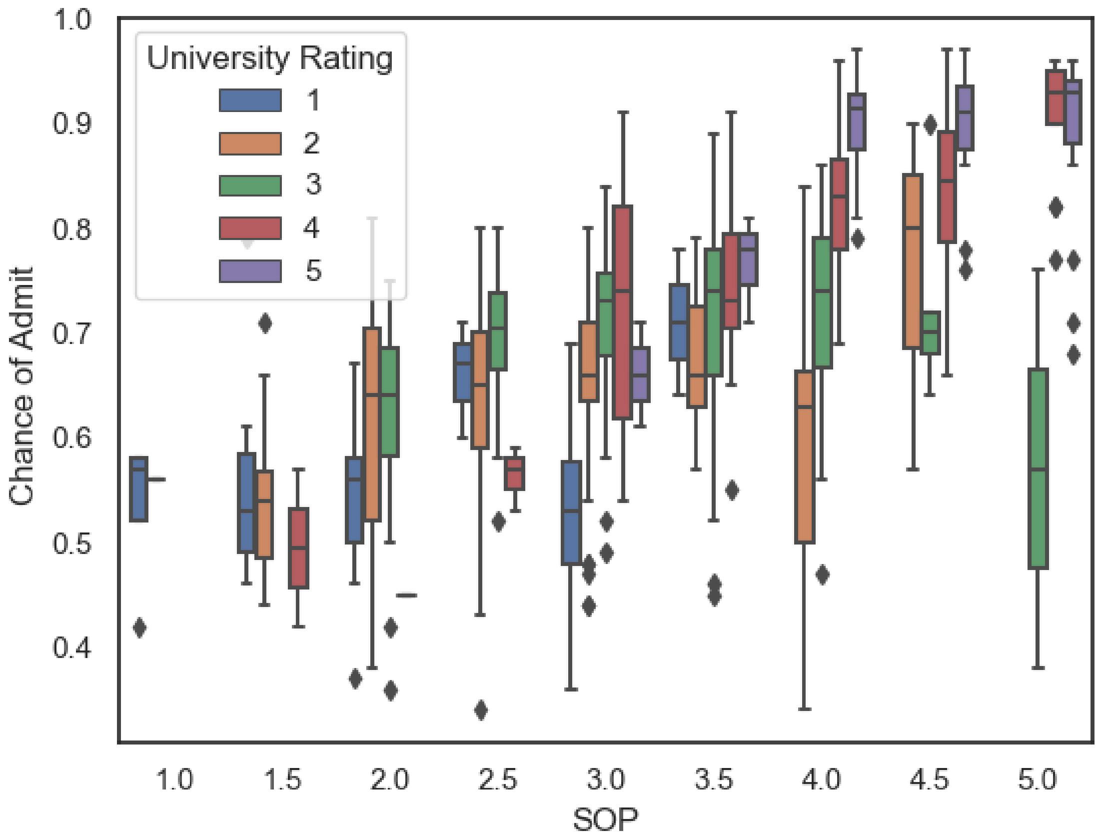
In [54]:
```
sns.boxplot(x = 'University Rating', y = 'GRE Score', hue = 'Research', data = JDF_
plt.show()
```



In [55]:
```
sns.boxplot(x = 'University Rating', y = 'TOEFL Score', hue = 'Research', data = J
plt.show()
```



In [56]:
```
sns.boxplot(x = 'SOP', y = 'Chance of Admit ', hue = 'University Rating', data = JI
plt.show()
```



Observation:

Candidates which has higher SOP (SOP >= 4.0), has higher chance of admission as per the data.

Candidates who has higher SOP but Non research profiles they are lower probabilities than students with slightly lower SOP but research profiles has higher prob to get the admission.

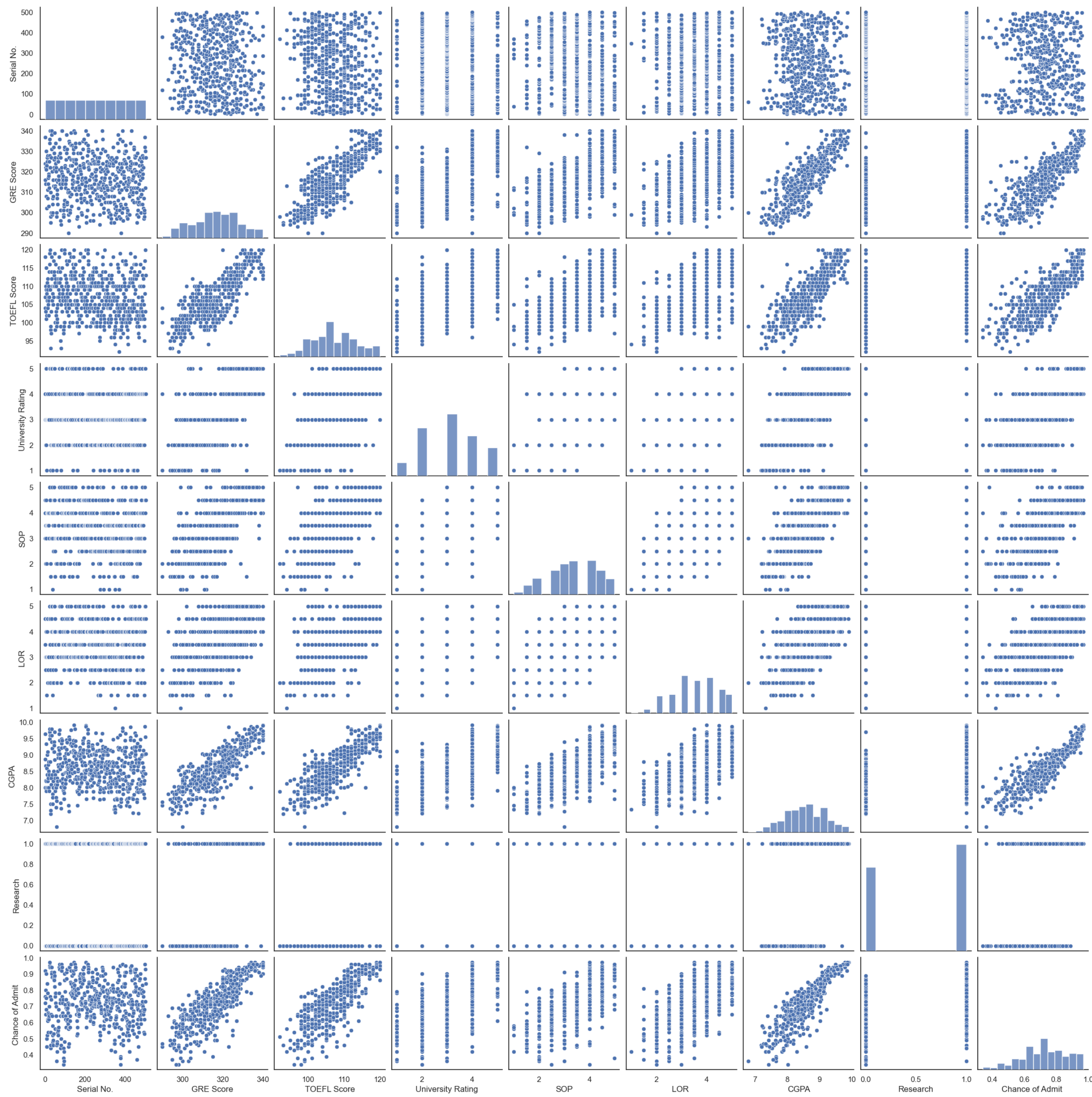
In all SOP categories, Non researcher's profiles has lower median of prob of getting selected to Internation college IVY leagues.

Candidates with tier 4 and 5 universities has higher chance of getting the admission.

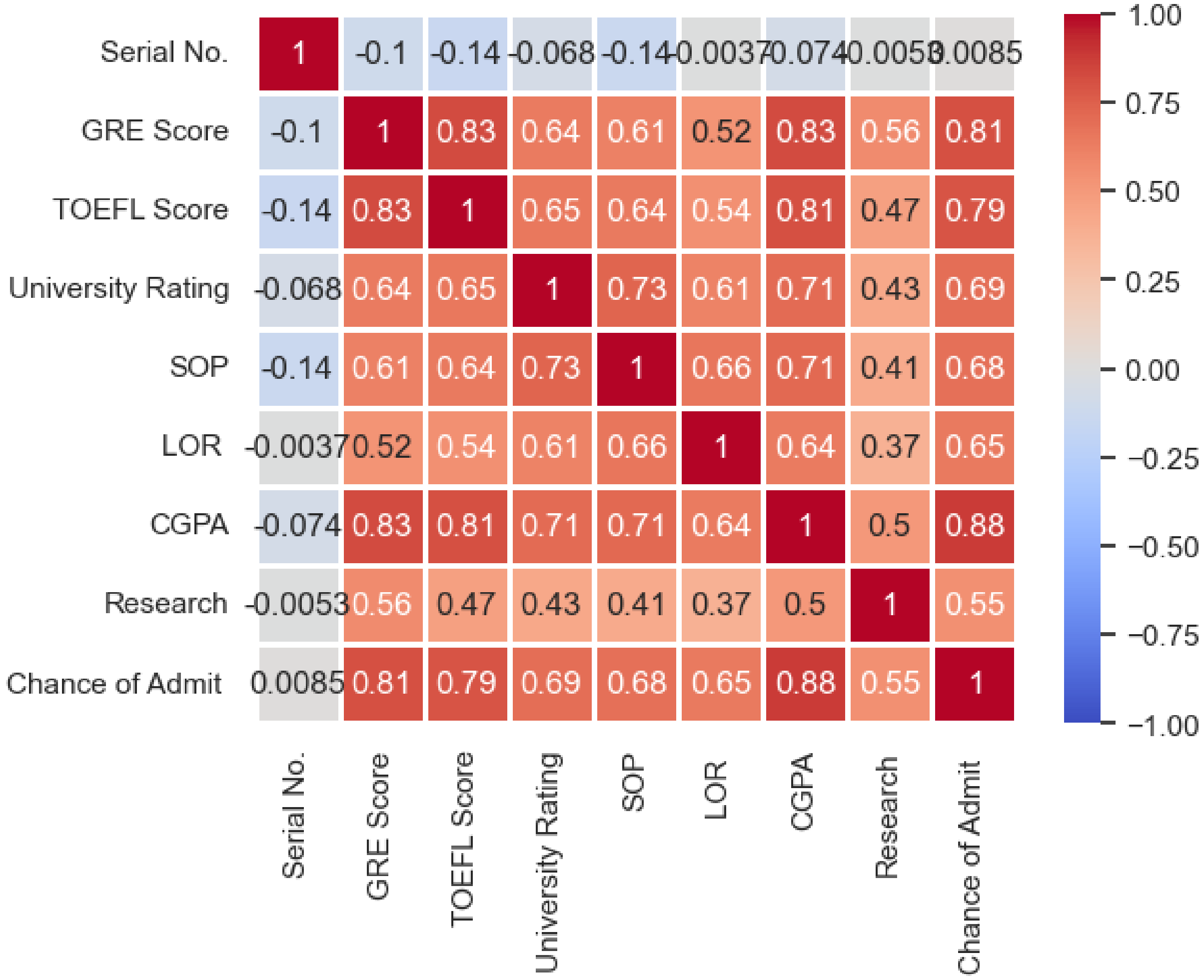
Researcher profiles tends to get higher GRE score than non researchers candidates.

There are candidates who are from tier 3 universities but they have huigher SOP score, could possibly selected for admission. tier 5 college candidates has more than 65% of chance of getting selected.

In [57]:
```
 sns.pairplot(JDF_data)
plt.show()
```



In [58]:
```
sns.heatmap(JDF_data.corr(), annot=True, vmin=-1, vmax = 1,cmap='coolwarm',linewid
plt.show()
```



Observations:

There are total 4 numerical/continuous and 5 numerical/descrete features. In total 9 independent features with 500 rows.

Missing data or Null values do not exist.

No duplicate data found in the data.

TOEFL and GRE score as very high correlation value as .83, which means they are positivel related.

Chance of Admit and GRE score also has very high correlation value as .81.

Chance of Admit and CGPA has high correlatiion score as .88.

TOEFL and Chance of Admit has high correlation score as .78.

Data Preprocessing

Duplicate Value Check

In [59]:
```
JDF_data[JDF_data.duplicated(keep=False)]
```

Out[59]:

Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit

Missing Values treatment

Numerical features

```
Univariate imputation (when we use mean,median,mode of the same column to
impute the missing data in that column)
    mean/median imputation
    Arbitrary imputation
    Random imputation

Multivariate Imputation (when we use data from diff column to impute the m
isisng value in the asked column)
```

Categorical features

```
Most Frequent imputation (Here we impute missing values with most frequent
data value of that column)
Missing category imputation (Here we can create another category called Mi
ssing in data)
```

In [60]:
```
missingValue(JDF_data)
```

```
Total number of records =  500
```

Out[60]:

	Total Missing values	In Percent
Serial No.	0	0.0
GRE Score	0	0.0
TOEFL Score	0	0.0
University Rating	0	0.0
SOP	0	0.0
LOR	0	0.0
CGPA	0	0.0
Research	0	0.0
Chance of Admit	0	0.0

Outliers treatment:

Outliers can be detected by boxplot analysis visually. There are 2 ways in which in general we deal with outliers: Trimming or Removing Capping

we have different techniques to deal with outliers.

we can use z-score technique if our data/column are normal distributed or approx normal distributed.

when we dont have normal distribution of data then we can use IQR method.

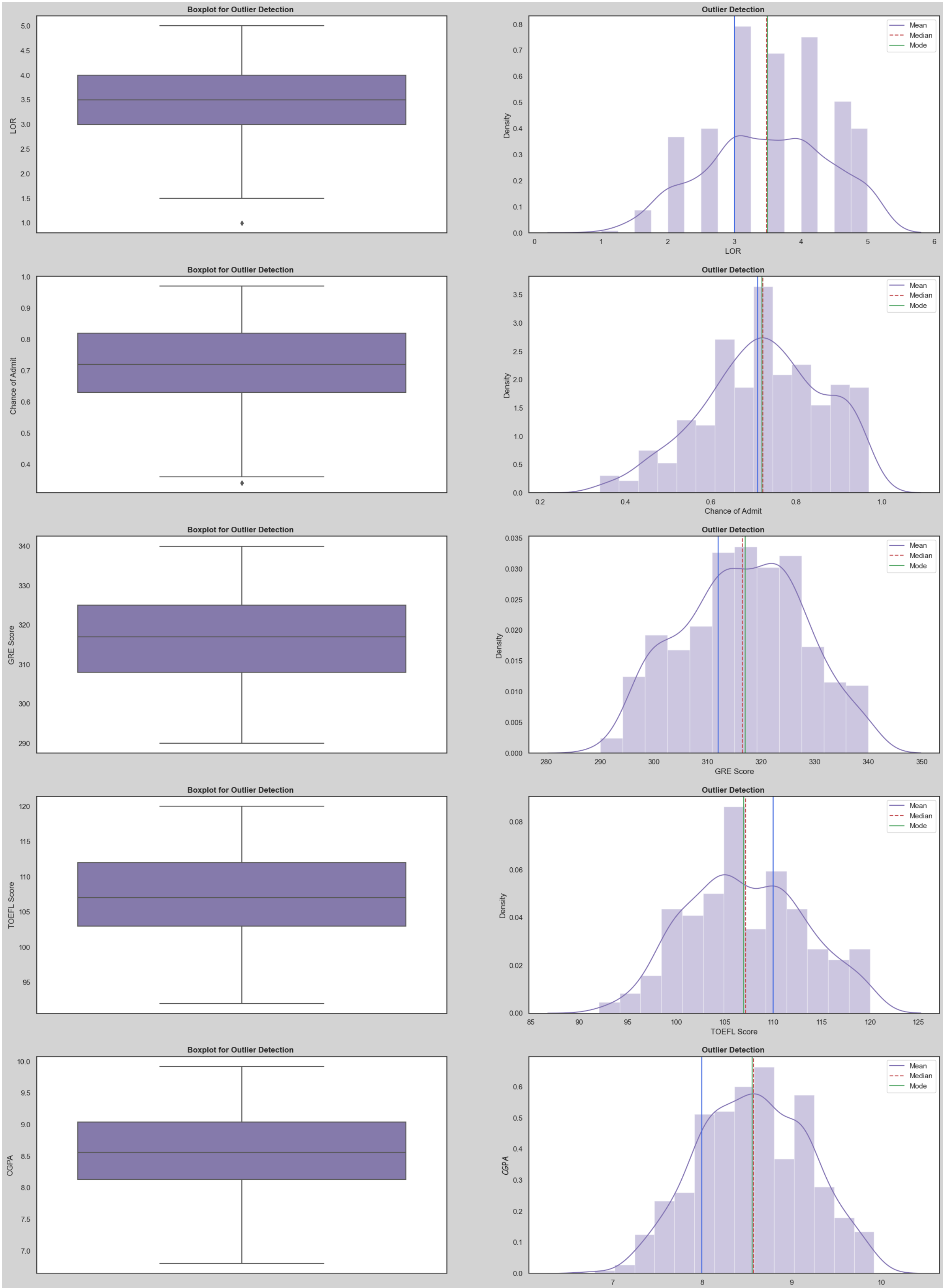
we can also use percentile method to remove or trim outliers.

In [61]:
```
JDF_data.columns
```

Out[61]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

Figuring out outliers visually

In [62]:
```
num_cols = ['LOR ', 'Chance of Admit ', 'GRE Score', 'TOEFL Score', 'CGPA']
get_outlier(JDF_data, num_cols, len(num_cols), 2, 25, 35)
```



Observation:

Since there was no missing value in the data so no missing value treatment is required.

No duplicate rows found hence no actions needs to be taken

There is no major outlier problem in data, so no outlier treatment is required.

In [64]:
```
JDF_data.columns
```

Out[64]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

Model building

In [69]:
```
# Lets scale the data, standardization
from sklearn.preprocessing import StandardScaler

X = JDF_data[JDF_data.columns.drop(['Chance of Admit ', 'Serial No.'])]
y = JDF_data["Chance of Admit "]

sc = StandardScaler()
cols = X.columns
X[cols] = sc.fit_transform(X[cols])

X_sm = sm.add_constant(X)  # Statmodels default is without intercept, to add interc

sm_model = sm.OLS(y, X_sm).fit()

print(sm_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       Chance of Admit   R-squared:                       0.822
Model:                           OLS   Adj. R-squared:                  0.819
Method:                Least Squares   F-statistic:                     324.4
Date:               Wed, 21 Jun 2023   Prob (F-statistic):          8.21e-180
Time:                       07:58:04   Log-Likelihood:                 701.38
No. Observations:                500   AIC:                            -1387.
Df Residuals:                    492   BIC:                            -1353.
Df Model:                          7
Covariance Type:           nonrobust
==================================================================================
====
                        coef    std err          t      P>|t|      [0.025      0.
975]
----------------------------------------------------------------------------------
----
const                 0.7217      0.003    269.039      0.000       0.716
0.727
GRE Score             0.0210      0.006      3.700      0.000       0.010
0.032
TOEFL Score           0.0169      0.005      3.184      0.002       0.006
0.027
University Rating     0.0068      0.004      1.563      0.119      -0.002
0.015
SOP                   0.0016      0.005      0.348      0.728      -0.007
0.010
LOR                   0.0156      0.004      4.074      0.000       0.008
0.023
CGPA                  0.0715      0.006     12.198      0.000       0.060
0.083
Research              0.0121      0.003      3.680      0.000       0.006
0.019
==============================================================================
Omnibus:                     112.770   Durbin-Watson:                   0.796
Prob(Omnibus):                 0.000   Jarque-Bera (JB):              262.104
Skew:                         -1.160   Prob(JB):                     1.22e-57
Kurtosis:                      5.684   Cond. No.                         5.65
==============================================================================

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

In [70]:

```
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,random_state=42
```

In [71]:

```
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
```

In [72]:

```
lr = LinearRegression()
```

In [73]:
```
X_train.shape, y_train.shape
```

Out[73]:
```
((400, 7), (400,))
```

In [74]:
```
X_train.head()
```

Out[74]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
249	0.401282	0.626751	-0.099793	0.127271	0.558125	0.419657	0.886405
433	-0.041830	0.626751	0.775582	0.632315	1.639763	-0.060310	-1.128152
19	-1.193919	-0.854540	-0.099793	0.127271	-0.523513	-0.126513	-1.128152
322	-0.219074	-0.031601	-0.975168	-0.882817	0.558125	-0.507177	-1.128152
332	-0.750808	-0.196189	-0.099793	0.127271	-1.064332	-0.606480	0.886405

In [78]:
```
lr.fit(X_train,y_train)
```

Out[78]:
```
▸ LinearRegression
```

In [76]:
```
y_pred = lr.predict(X_test)
```

In [77]:
```
lr.coef_
```

Out[77]:
```
array([0.02746983, 0.01820228, 0.00293451, 0.00179558, 0.01593692,
       0.06798973, 0.01192658])
```

In [79]:
```
lr.intercept_
```

Out[79]:
```
0.7228307247435319
```

In [80]:
```
r2_score(y_test,y_pred)
```

Out[80]:
```
0.8188432567829629
```

Applying Ridge regularization to our linear model

In [81]:
```
ridge = Ridge()
ridge.fit(X_train, y_train)
```

Out[81]:
```
▾ Ridge
Ridge()
```

In [82]:
```
y_pred_r = ridge.predict(X_test)
```

In [83]:
```
r2_score(y_test,y_pred_r)
```

Out[83]:
```
0.8187987385531805
```

Applying Lasso regularization to our linear model

In [84]:
```
lasso = Lasso()
lasso.fit(X_train, y_train)
```

Out[84]:
```
▾ Lasso
Lasso()
```

In [85]:
```
y_pred_l = lasso.predict(X_test)
```

In [86]:
```
r2_score(y_test, y_pred_l)
```

Out[86]:
```
-0.00724844132029312
```

In [87]:
```
print("MAE",mean_absolute_error(y_test,y_pred))
print("MSE",mean_squared_error(y_test,y_pred))
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred)))
print("R2 score",r2_score(y_test,y_pred))
```

```
MAE 0.042722654277053664
MSE 0.00370465539878841
RMSE 0.060865880415783113
R2 score 0.8188432567829629
```

Observations:

we can see that R2 score from statsmodel lib has higher score than scikit learn library.

After applying Ridge regularization to our model gives us the same result as normal Linear regression model.

But applying Lasso regression seem to be BIG NO NO, as R2 score turned out to be Negative.

One possible reason could be that Lasso won't perform good on data which has high collinear data by dropping them during L1 regularization process which also mean loss of information.

lasso would be good fit when we have huge dataset with lots of features to deal with, then lasso can help us by dropping useless feature's coeffecients to zero.

Testing the assumptions of the linear regression model

1. Linear relationship between dependent and indepdent variables¶

In [89]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train,y_train)

# Residual
y_pred = model.predict(X_test)
residual = y_test - y_pred
```

In [90]:
```
X_train.columns
```

Out[90]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research'],
      dtype='object')
```

In [93]:
```
fig, (ax1, ax2, ax3, ax4, ax5, ax6, ax7) = plt.subplots(ncols=7, figsize=(20, 5))

ax1.scatter(JDF_data['GRE Score'], JDF_data['Chance of Admit '])
ax1.set_title("GRE Score")

ax2.scatter(JDF_data['TOEFL Score'], JDF_data['Chance of Admit '])
ax2.set_title("TOEFL Score")

ax3.scatter(JDF_data['University Rating'], JDF_data['Chance of Admit '])
ax3.set_title("University Rating")

ax4.scatter(JDF_data['SOP'], JDF_data['Chance of Admit '])
ax4.set_title("SOP")

ax5.scatter(JDF_data['LOR '], JDF_data['Chance of Admit '])
ax5.set_title("LOR ")

ax6.scatter(JDF_data['CGPA'], JDF_data['Chance of Admit '])
ax6.set_title("CGPA")

ax7.scatter(JDF_data['Research'], JDF_data['Chance of Admit '])
ax7.set_title("Research")

plt.show()
```

In [94]:
```
# VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
X_t = X
vif['Features'] = X_t.columns
vif['VIF'] = [variance_inflation_factor(X_t.values, i) for i in range(X_t.shape[1]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[94]:

	Features	VIF
5	CGPA	4.78
0	GRE Score	4.46
1	TOEFL Score	3.90
3	SOP	2.84
2	University Rating	2.62
4	LOR	2.03
6	Research	1.49

Observartion:

we could see that VIF score of CGPA and GRE score are close to 5 .

we can drop these features and test the R2 sqaure of our model.

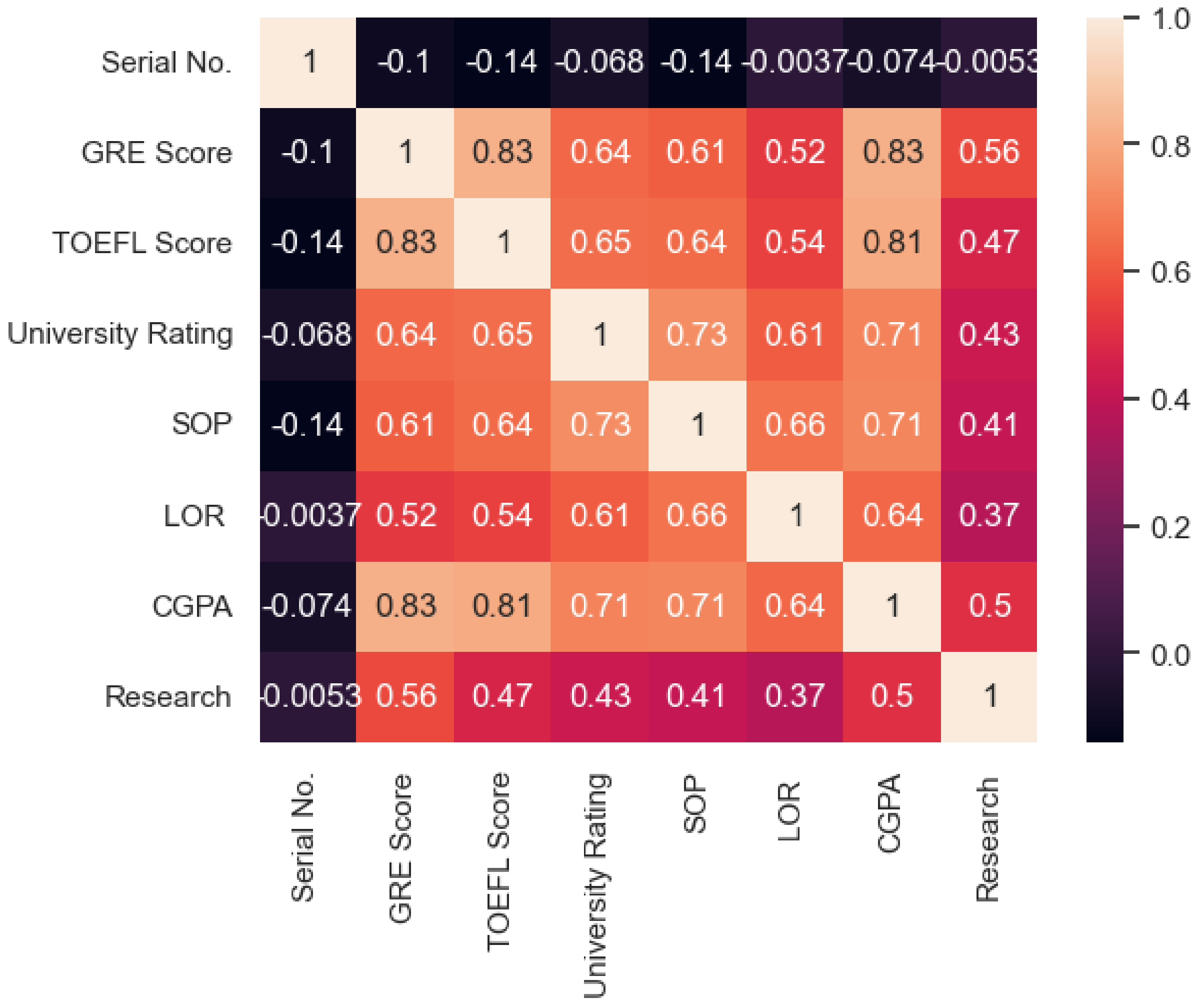
In [96]:
```
cols_2 = ['LOR ', 'University Rating', 'TOEFL Score', 'Research', 'SOP']
X2 = X[cols_2]

X2_sm = sm.add_constant(X2)  #Statmodels default is without intercept, to add inter

sm_model = sm.OLS(y, X2_sm).fit()

print(sm_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        Chance of Admit   R-squared:                       0.737
Model:                            OLS   Adj. R-squared:                  0.734
Method:                 Least Squares   F-statistic:                     276.6
Date:                Wed, 21 Jun 2023   Prob (F-statistic):         1.23e-140
Time:                        08:02:28   Log-Likelihood:                 603.72
No. Observations:                 500   AIC:                            -1195.
Df Residuals:                     494   BIC:                            -1170.
Df Model:                           5
Covariance Type:            nonrobust
==================================================================================
                        coef    std err          t      P>|t|      [0.025      0.
975]
----------------------------------------------------------------------------------
const             0.7217      0.003    221.752      0.000       0.715
0.728
LOR               0.0264      0.005      5.824      0.000       0.018
0.035
University Rating 0.0190      0.005      3.664      0.000       0.009
0.029
TOEFL Score       0.0657      0.005     13.998      0.000       0.057
0.075
Research          0.0228      0.004      6.065      0.000       0.015
0.030
SOP               0.0134      0.005      2.492      0.013       0.003
0.024
==============================================================================
Omnibus:                       69.852   Durbin-Watson:                   0.864
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              104.939
Skew:                          -0.919   Prob(JB):                     1.63e-23
Kurtosis:                       4.288   Cond. No.                         3.51
==============================================================================

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

Observation:

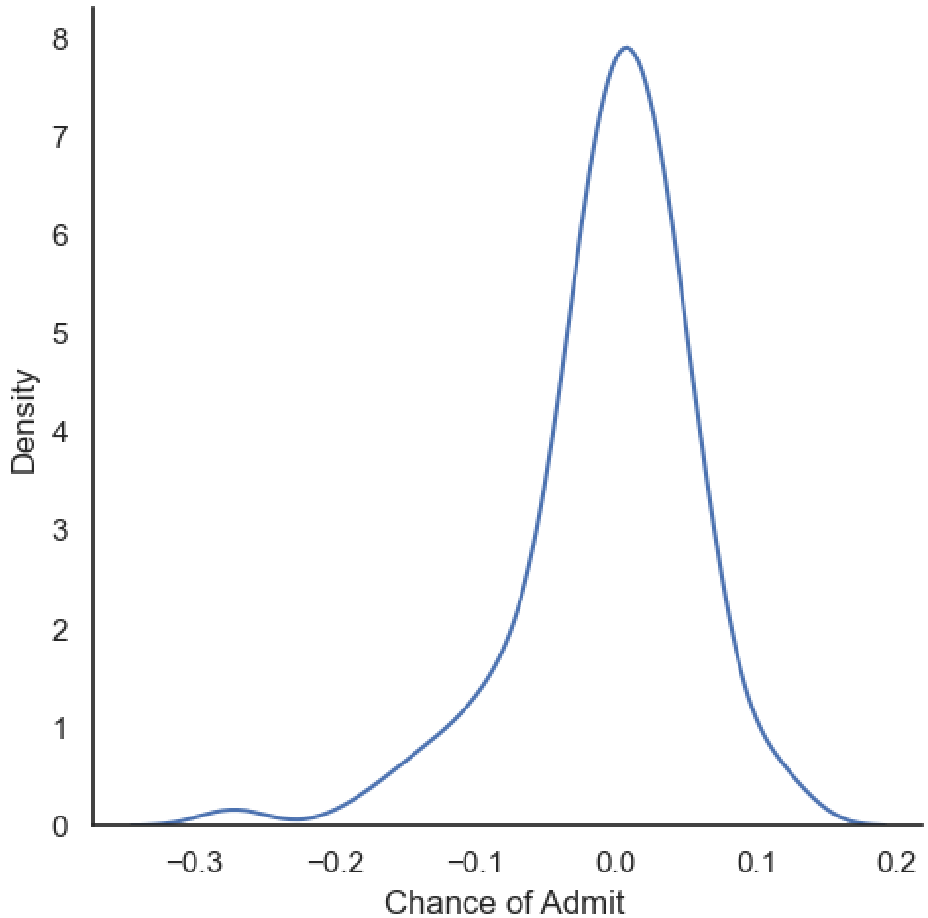
we could see that R2 squared went down by removing GRE Score and CGPA which implies we should not remove them.

In [99]:
```
# Another Technique
sns.heatmap(JDF_data.iloc[:,0:-1].corr(),annot=True)
plt.show()
```

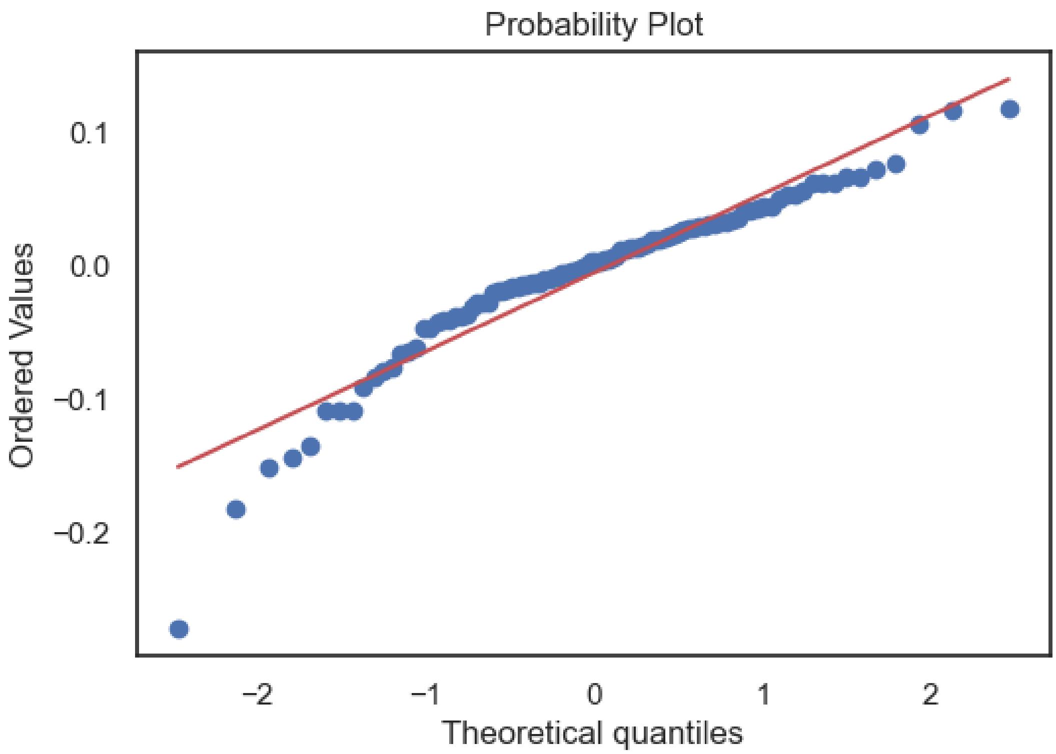


3. Normality of residual of Errors

In [100]:
```
sns.displot(residual,kind='kde')
plt.show()
```



In [101]:
```
# QQ Plot
import scipy as sp
fig, ax = plt.subplots(figsize=(6,4))
sp.stats.probplot(residual, plot=ax, fit=True)

plt.show()
```



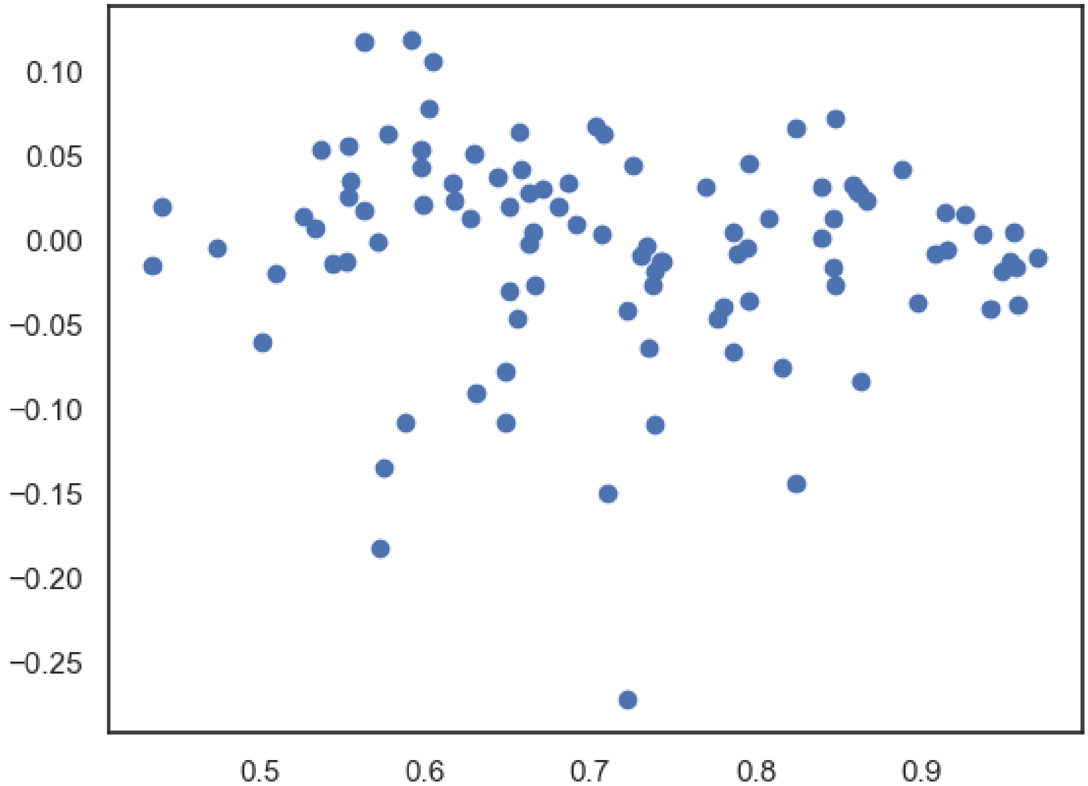
Observation:

we could see that from distplot and QQ plot that our residual follows Normal distribution.

Hence Mean of residuals are close to zero.

4. Homoscedasticity

In [102]:
```
plt.scatter(y_pred,residual)
plt.show()
```



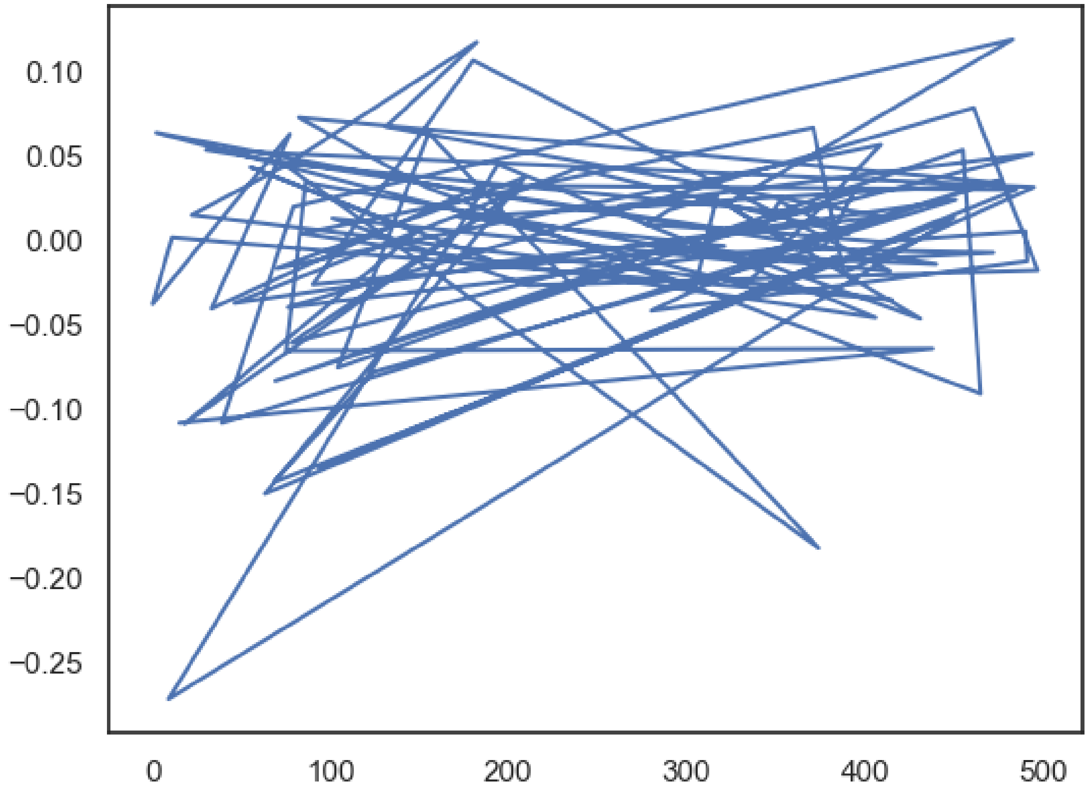
Observation:

we could see that not exactly but data follows homoscedasticity.

Points scattered evenly across mean. Hence does not showing any sign of Hetroscedasticity.

5. Autocorrelation of Residuals

In [103]:
```
plt.plot(residual)
plt.show()
```



Observations:

We can that Residuals does not follow any perticular pattern.

Hence Auto correlation of residuals assumption also hold true.

Model performance evaluation

Metrics Check

MAE

In [104]:
```
print("MAE",mean_absolute_error(y_test,y_pred))
```

```
MAE 0.042722654277053664
```

MSE

In [106]:
```
print("MSE",mean_squared_error(y_test,y_pred))
```

```
MSE 0.00370465539878841
```

RMSE

In [107]:
```
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred)))
```

```
RMSE 0.060865880415783113
```

R2

In [108]:
```
print("R2 score",r2_score(y_test,y_pred))
```

```
R2 score 0.8188432567829629
```

Adjusted R2 = 1 – [(1-R2)*(n-1)/(n-k-1)]

In [109]:
```
Adj_R2 = 1 - (1 - model.score(X_train, y_train))*(len(y_train)-1)/(len(y_train)- X_
print('Adj R2 ', Adj_R2)
```

```
Adj R2  0.8178719072345153
```

Train and test performances Checks

In [110]:
```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet

names = []
train_scores = []
test_scores = []

models = {
       'OLS': LinearRegression(),
       'Ridge': Ridge(),
       'Lasso': Lasso(),
       'ElasticN': ElasticNet()
        }

for name, model in models.items():
    name_model = model
    name_fit = name_model.fit(X_train, y_train)
    name_pred = name_model.predict(X_test)
    name_train_score = name_model.score(X_train, y_train).round(4)
    name_test_score = name_model.score(X_test, y_test).round(4)
    names.append(name)
    train_scores.append(name_train_score)
    test_scores.append(name_test_score)

score_df = pd.DataFrame(train_scores, test_scores)
score_df
```

Out[110]:

	0
0.8188	0.8211
0.8188	0.8211
-0.0072	0.0000
-0.0072	0.0000

In [111]:
```
score_df.reset_index(inplace = True, drop=False)
score_df
```

Out[111]:

	index	0
0	0.8188	0.8211
1	0.8188	0.8211
2	-0.0072	0.0000
3	-0.0072	0.0000

In [112]:
```
score_df.columns = ['train_score', 'test_score']
```

In [114]:
```
score_df.insert(0, 'models', names)
```

In [115]:
```
score_df
```

Out[115]:

	models	train_score	test_score
0	OLS	0.8188	0.8211
1	Ridge	0.8188	0.8211
2	Lasso	-0.0072	0.0000
3	ElasticN	-0.0072	0.0000

Observations:

from above table , it's quite evident that model is performing good with OLS and Ridge.

their test score and train scores also very close to each other.

Lasso and Elastic Net did not perform well in our case for Linear regression model build.

Actionable Insights & Recommendations

##Business Insights and Recommendations:

we able to create our first Linear regression model both with stats model and scikit library.

our model is performing good with the given the data.

we expect the performance to be higher when we have more data.

we can see that we can apply Linear regression as All 5 assumptions has been met.

Though we have multi collinearity within the features, but still we cant drop those features as they have high importance in model prediction.

we made our model less complex by figuring out those five features with the help of which we can able to predict our regression output.

Those 5 most important features that our model need to predict dependent varianble are LOR, CGPA, GRE Score, TOEFL Score, Research.

with the help of our model we can predict chance of admission of candidates if we get the above mentioned features to our model. we could improve the model prediction if we get more data for our model.

from whole analysis, it turned out to be that High CGPA, GRE score, TOEFL score with research profile candidates has higher chance to get admission.

from Business improvement, company can suggest to their candidates to focus more on these imprtant features as it can increase their chance of admission.

Company can offer seperate service to the candidates, whose chance of Admission are low by focusing on these areas.

Since due to addition of these model prediction features, candidates will feel transparent , which brings more business to Jamboree as students now know before applying what all needs to be focus if chance of admission turned out to be lower.

Due to adding new service for those students which has lower chance company will increase the revenue and due to addition of unbiased model prediction software , company will show case that they are with the latest technology in edtech business. This gives extra edge to the company.

In [ ]:

```

```