About Jamboree

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.

Business Problem

To help Jamboree understand 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.

In [1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import r2_score

from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy import stats
```

In [2]:
```
# Loading dataset
df = pd.read_csv("/Users/bose/Downloads/Jamboree.csv")
```

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

Out[3]:

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

In [4]:
```
df.tail()
```

Out[4]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
495	496	332	108	5	4.5	4.0	9.02	1	0.87
496	497	337	117	5	5.0	5.0	9.87	1	0.96
497	498	330	120	5	4.5	5.0	9.56	1	0.93
498	499	312	103	4	4.0	5.0	8.43	0	0.73
499	500	327	113	4	4.5	4.5	9.04	0	0.84

In [5]:
```
df.info()
```

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

In [6]:
```
df.shape
```

Out[6]:
```
(500, 9)
```

In [7]:
```
df.dtypes
```

Out[7]:
```
Serial No.             int64
GRE Score              int64
TOEFL Score            int64
University Rating      int64
SOP                  float64
LOR                  float64
CGPA                 float64
Research               int64
Chance of Admit      float64
dtype: object
```

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

Out[8]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGI
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.00000	500.0000
mean	250.500000	316.472000	107.192000	3.114000	3.374000	3.48400	8.5764
std	144.481833	11.295148	6.081868	1.143512	0.991004	0.92545	0.6048
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.00000	6.8000
25%	125.750000	308.000000	103.000000	2.000000	2.500000	3.00000	8.1275
50%	250.500000	317.000000	107.000000	3.000000	3.500000	3.50000	8.5600
75%	375.250000	325.000000	112.000000	4.000000	4.000000	4.00000	9.0400
max	500.000000	340.000000	120.000000	5.000000	5.000000	5.00000	9.9200

In [9]:
```
#Checking for Null values -
df.isnull().sum()
```

Out[9]:
```
Serial No.           0
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

There are no null values in the dataset

In [10]:
```
#Removing column Serial No. -
df.drop(columns=['Serial No.'], inplace=True)
```

In [11]:
```
#Checking for duplicates
df.duplicated().sum()
```

Out[11]:
```
0
```

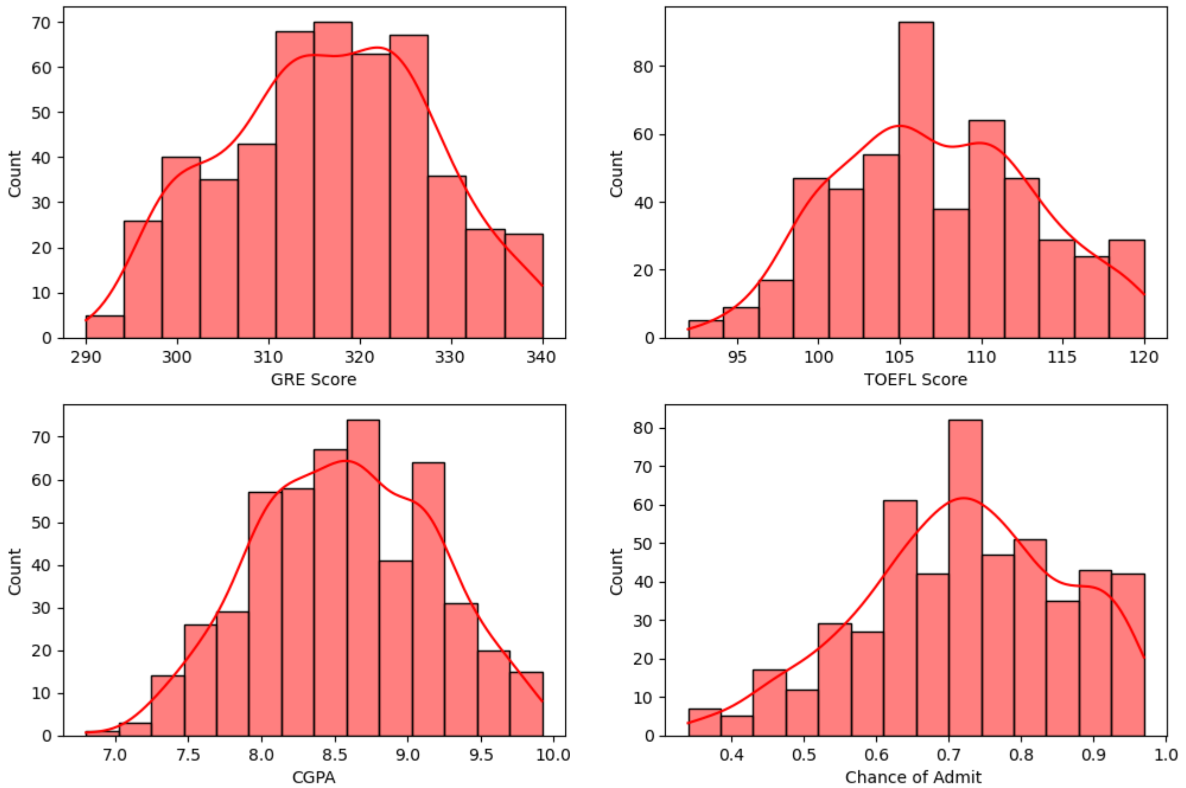
There are no duplicates in the given dataset

In [12]:
```
cat_cols = ['University Rating', 'SOP', 'LOR ', 'Research']
num_cols = ['GRE Score', 'TOEFL Score', 'CGPA']
target = 'Chance of Admit '
```

Univariate Analysis

In [13]:
```
#Distribution of Continuous numerical features
rows, cols = 2, 2
fig, axs = plt.subplots(rows,cols, figsize=(12, 8))
index = 0
for row in range(rows):
    for col in range(cols):
        sns.histplot(df[num_cols[index]], kde=True, ax=axs[row,col], color=
        index += 1
    break
```

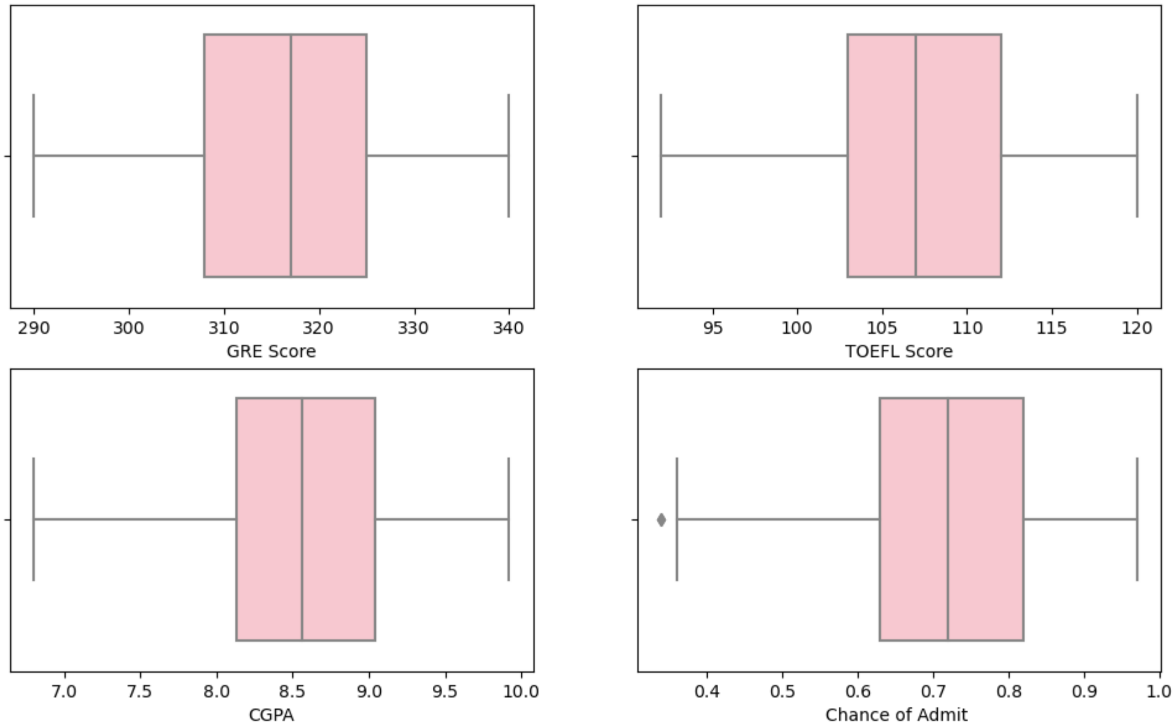
```
sns.histplot(df[num_cols[-1]], kde=True, ax=axs[1,0], color='r')
sns.histplot(df[target], kde=True, ax=axs[1,1], color='r')
plt.show()
```



Observation -

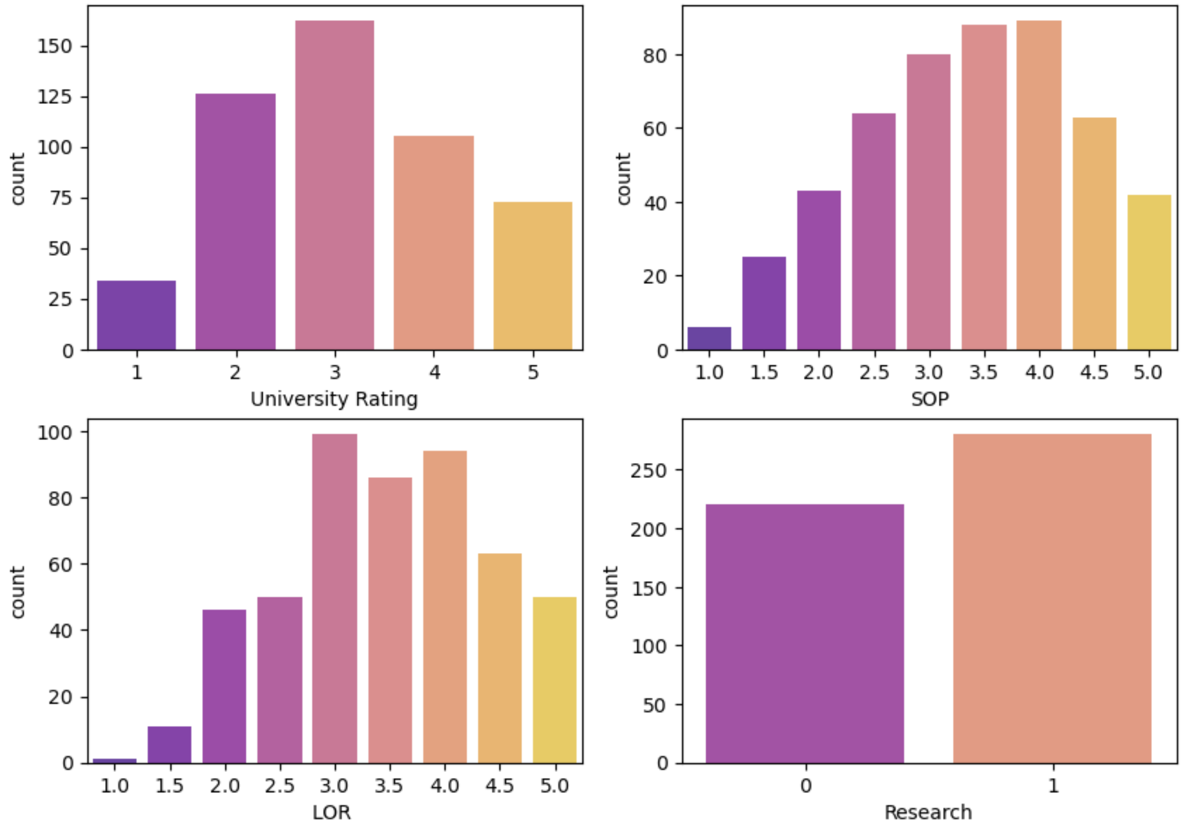
- Most students have GRE score in the range of (310-330)
- Most students have TOEFL score in the range (105-110)
- Majority of students have CGPA in the range of (8.0 - 9.0)
- Average Chance of Admit for the student comes out to 0.72

In [14]:
```
#Checking for outliers using boxplots
rows, cols = 2, 2
fig, axs = plt.subplots(rows, cols, figsize=(12, 7))

index = 0
for col in range(cols):
    sns.boxplot(x=num_cols[index], data=df, ax=axs[0,index],color='pink')
    index += 1

sns.boxplot(x=num_cols[-1], data=df, ax=axs[1,0],color='pink')
sns.boxplot(x=target, data=df, ax=axs[1,1],color='pink')
plt.show()
```



There are no outliers in the dataset

In [15]:
```
#Countplots for categorical variables
cols, rows = 2, 2
fig, axs = plt.subplots(rows, cols, figsize=(10, 7))

index = 0
for row in range(rows):
    for col in range(cols):
        sns.countplot(x=cat_cols[index], data=df, ax=axs[row, col], alpha=0.
        index += 1

plt.show()
```

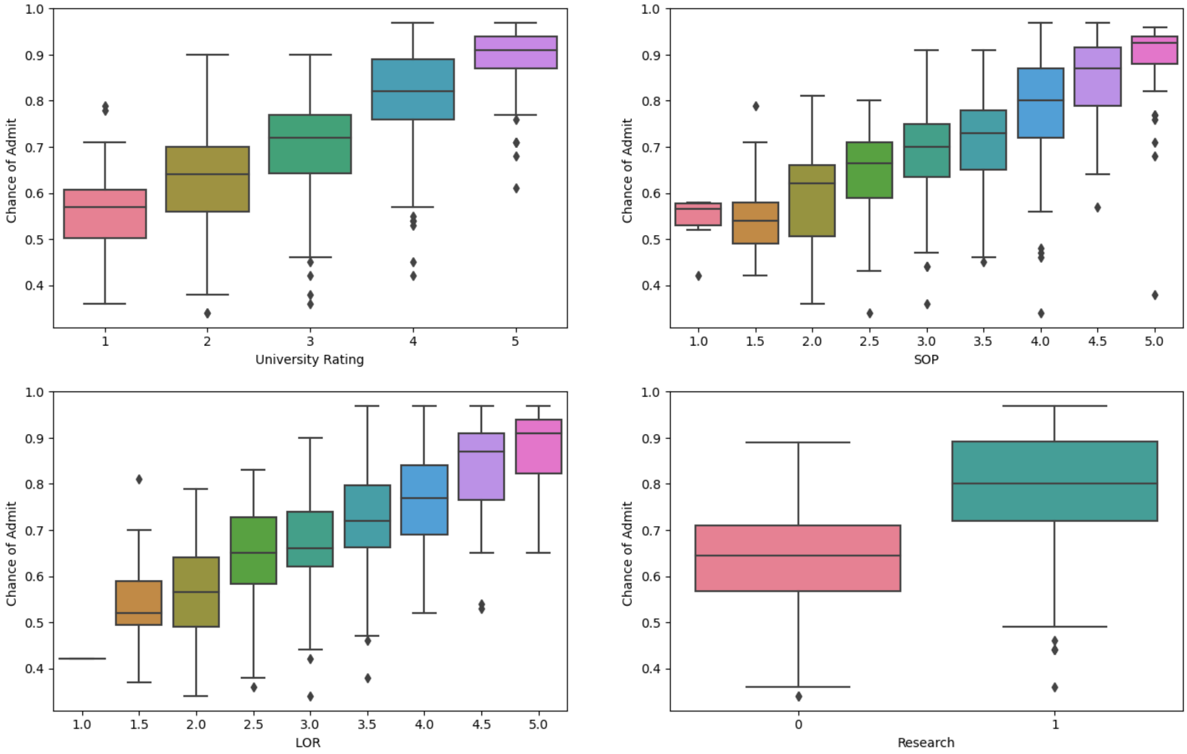


Observation -

- Most no of students have a University Ranking of 3
- SOP score of 3.5 and 4.0 is most common among students
- Highest number of students have an LOR score of 3.0
- No of students who have conducted research is more than the no of students who have not

Bivariate Analysis

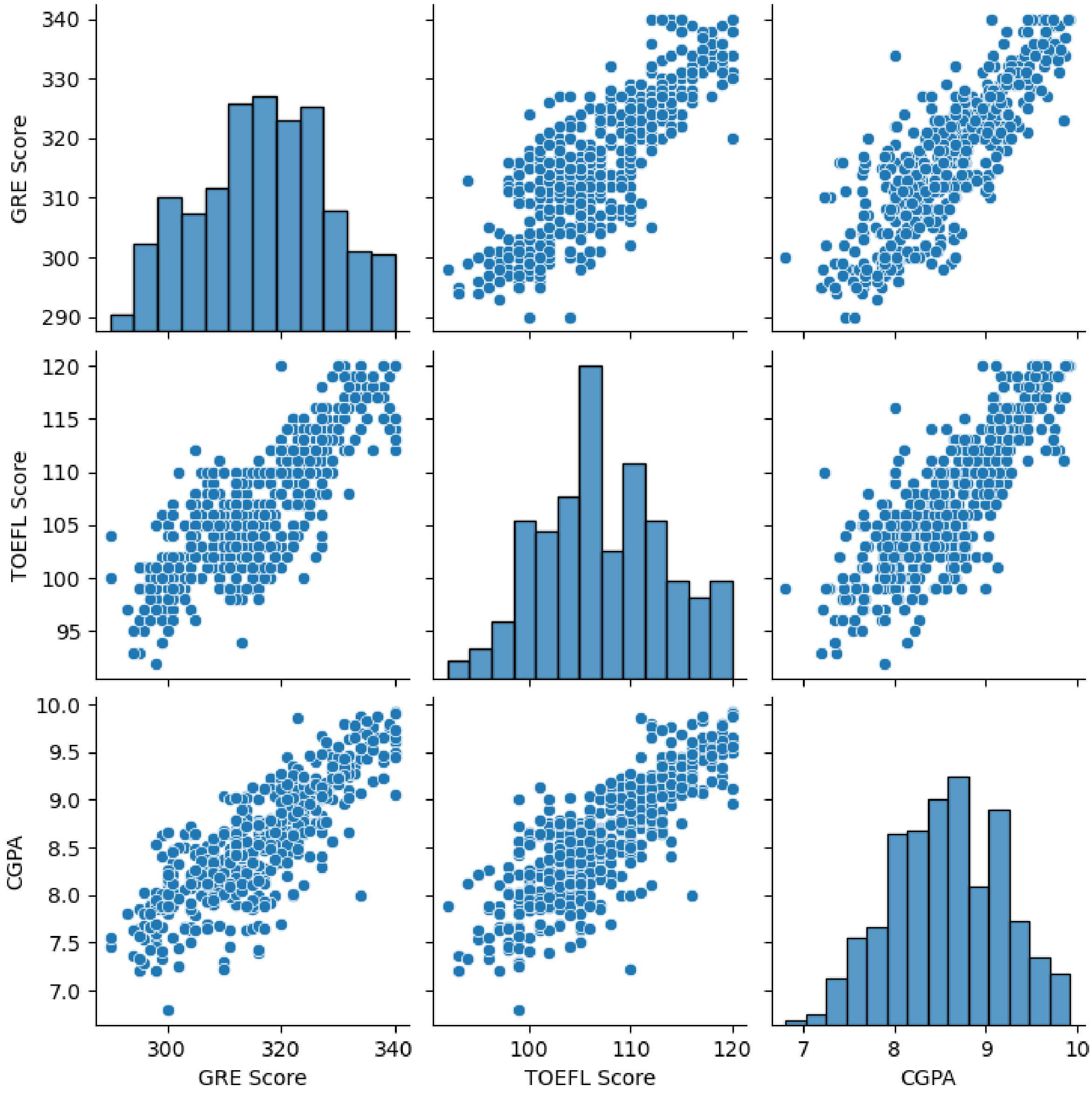
In [16]:
```
#Boxplot to represent the effect of categorical values on Chance of Admit
rows, cols = 2,2
fig, axs = plt.subplots(rows, cols, figsize=(16,10))

index = 0
for row in range(rows):
    for col in range(cols):
        sns.boxplot(x=cat_cols[index], y=target, data=df, ax=axs[row,col],
        index += 1
```



Observation -

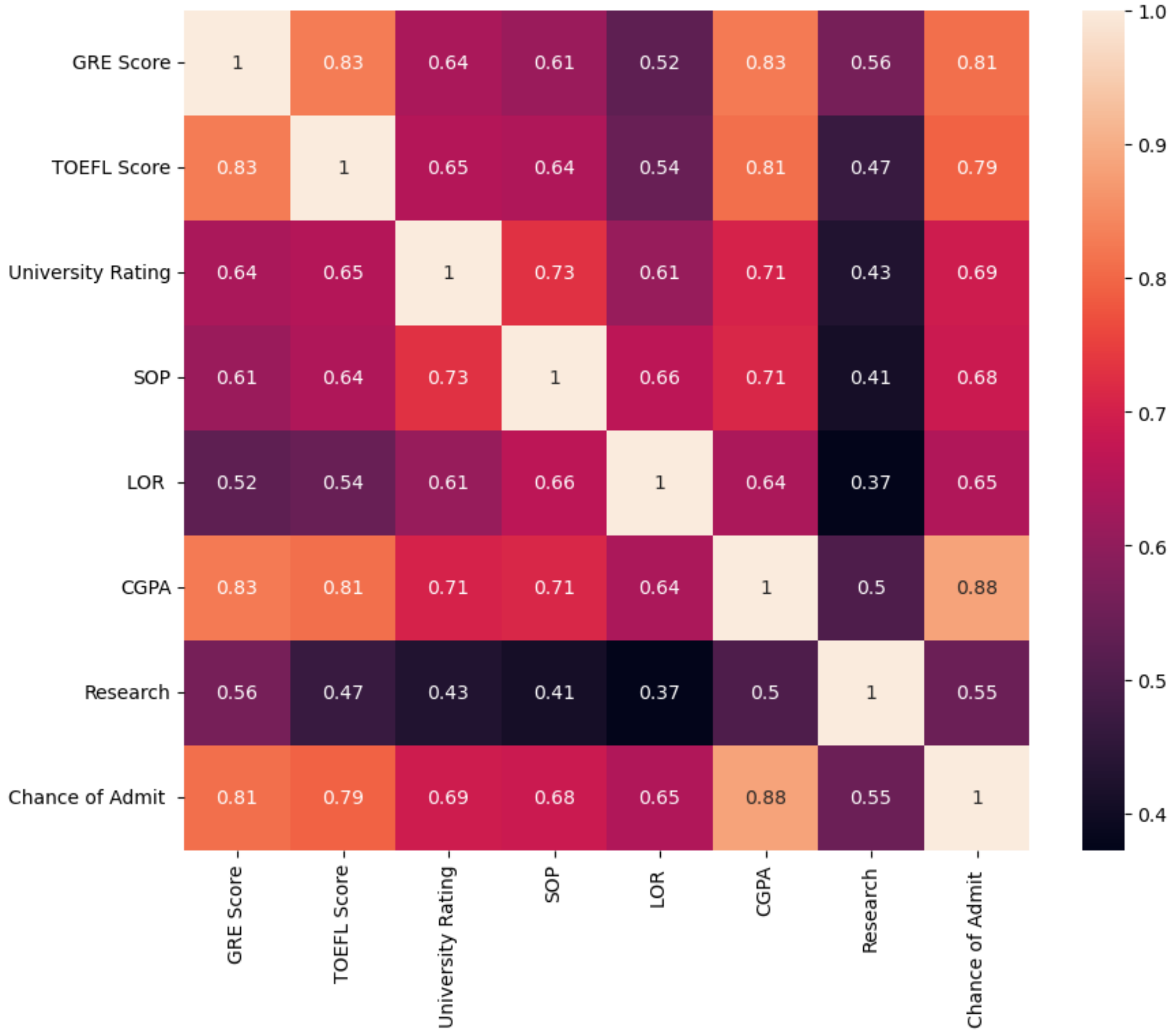
- As the University Rating increases, Chance of Admit also increases
- Same is the case with SOP and LOR. A Candidate with higher rating of SOP and LOR have a higher Chance of Admit
- A student with more research experience have higher Chance of Admit than a student without research experience

In [17]:
```
sns.pairplot(df[num_cols])
plt.show()
```



The numerical columns seems to have a correlation with each other

In [18]:
```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), annot=True)
plt.show()
```



Observation -

- **Chance of Admit** have a **high correlation** with **GRE Score, TOEFL Score and CGPA**
- Research seems to have low correlation with other features

Model Building

In [19]:
```
X = df.drop(columns=[target])
y = df[target]
```

In [20]:
```
#Standardize the dataset
sc = StandardScaler()
X = sc.fit_transform(X)
```

In [21]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ran
```

In [22]:
```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(350, 7) (350,)
(150, 7) (150,)
```

In [23]:
```
def adjusted_r2(r2, p, n):
    """
    n: no of samples
```

```
    p: no of predictors
    r2: r2 score
    """
    adj_r2 = 1 - ((1-r2)*(n-1) / (n-p-1))
    return adj_r2

def get_metrics(y_true, y_pred, p=None):
    n = y_true.shape[0]
    mse = np.sum((y_true - y_pred)**2) / n
    rmse = np.sqrt(mse)
    mae = np.mean(np.abs(y_true - y_pred))
    score = r2_score(y_true, y_pred)
    adj_r2 = None
    if p is not None:
        adj_r2 = adjusted_r2(score, p, n)

    res = {
        "mean_absolute_error": round(mae, 2),
        "rmse": round(rmse, 2),
        "r2_score": round(score, 2),
        "adj_r2": round(adj_r2, 2)
    }
    return res
```

In [24]:

```
def train_model(X_train, y_train, X_test, y_test,cols, model_name="linear",
    model = None
    if model_name == "lasso":
        model = Lasso(alpha=alpha)
    elif model_name == "ridge":
        model = Ridge(alpha=alpha)
    else:
        model = LinearRegression()

    model.fit(X_train, y_train)
    y_pred_train = model.predict(X_train)
    y_pred_test = model.predict(X_test)
    p = X_train.shape[1]
    train_res = get_metrics(y_train, y_pred_train, p)
    test_res = get_metrics(y_test, y_pred_test, p)

    print(f"\n----   {model_name.title()} Regression Model  ----\n")
    print(f"Train MAE: {train_res['mean_absolute_error']}  Test MAE: {test_r
    print(f"Train RMSE: {train_res['rmse']}  Test RMSE: {test_res['rmse']}")
    print(f"Train R2_score: {train_res['r2_score']}  Test R2_score: {test_re
    print(f"Train Adjusted_R2: {train_res['adj_r2']}  Test Adjusted_R2: {tes
    print(f"Intercept: {model.intercept_}")
    coef_df = pd.DataFrame({"Column": cols, "Coefficient": model.coef_})
    print(coef_df)
    print("-"*50)
    return model
```

In [25]:

```
train_model(X_train, y_train, X_test, y_test,df.columns[:-1], "linear")
train_model(X_train, y_train, X_test, y_test,df.columns[:-1], "ridge")
train_model(X_train, y_train, X_test, y_test,df.columns[:-1], "lasso", 0.001
```

```
----   Linear Regression Model  ----

Train MAE: 0.04  Test MAE: 0.04
Train RMSE: 0.06  Test RMSE: 0.06
Train R2_score: 0.82  Test R2_score: 0.82
Train Adjusted_R2: 0.82  Test Adjusted_R2: 0.81
Intercept: 0.724978121476996
              Column  Coefficient
0          GRE Score     0.018657
1        TOEFL Score     0.023176
2  University Rating     0.011565
3                SOP    -0.000999
4                LOR     0.012497
5               CGPA     0.064671
6           Research     0.013968
--------------------------------------------------

----   Ridge Regression Model  ----

Train MAE: 0.04  Test MAE: 0.04
Train RMSE: 0.06  Test RMSE: 0.06
Train R2_score: 0.82  Test R2_score: 0.82
Train Adjusted_R2: 0.82  Test Adjusted_R2: 0.81
Intercept: 0.7249823645841696
              Column  Coefficient
0          GRE Score     0.018902
1        TOEFL Score     0.023252
2  University Rating     0.011594
3                SOP    -0.000798
4                LOR     0.012539
5               CGPA     0.064004
6           Research     0.013990
--------------------------------------------------

----   Lasso Regression Model  ----

Train MAE: 0.04  Test MAE: 0.04
Train RMSE: 0.06  Test RMSE: 0.06
Train R2_score: 0.82  Test R2_score: 0.82
Train Adjusted_R2: 0.82  Test Adjusted_R2: 0.81
Intercept: 0.7249659139557142
              Column  Coefficient
0          GRE Score     0.018671
1        TOEFL Score     0.022770
2  University Rating     0.010909
3                SOP     0.000000
4                LOR     0.011752
5               CGPA     0.064483
6           Research     0.013401
--------------------------------------------------
```

Out[25]:
```
Lasso(alpha=0.001)
```

Observation -

- Since there is no difference between train and test scores, we can say that there is no overfitting.
- There are no unnecessary independant variables in the data, as the value of R2 and Adjusted_R2 are almost same.

Assumptions of Linear Regression Model

Multicollinearity Check

In [26]:
```
def vif(newdf):
    # VIF dataframe
    vif_data = pd.DataFrame()
    vif_data["feature"] = newdf.columns

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

In [27]:
```
res = vif(df.iloc[:,:-1])
res
```

Out[27]:

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

In [28]:
```
#Drop GRE Score and again calculate the VIF
res = vif(df.iloc[:, 1:-1])
res
```

Out[28]:

	feature	VIF
0	TOEFL Score	639.741892
1	University Rating	19.884298
2	SOP	33.733613
3	LOR	30.631503
4	CGPA	728.778312
5	Research	2.863301

In [29]:
```
#Drop TOEFL Score and again calculate the VIF
res = vif(df.iloc[:,2:-1])
res
```

Out[29]:

	feature	VIF
0	University Rating	19.777410
1	SOP	33.625178
2	LOR	30.356252
3	CGPA	25.101796
4	Research	2.842227

In [30]:
```
#Drop SOP and again calculate VIF
res = vif(df.iloc[:,2:-1].drop(columns=['SOP']))
res
```

Out[30]:

	feature	VIF
0	University Rating	15.140770
1	LOR	26.918495
2	CGPA	22.369655
3	Research	2.819171

In [31]:
```
#Drop LOR and again calculate VIF
newdf = df.iloc[:,2:-1].drop(columns=['SOP'])
newdf = newdf.drop(columns=['LOR '], axis=1)
res = vif(newdf)
res
```

Out[31]:

	feature	VIF
0	University Rating	12.498400
1	CGPA	11.040746
2	Research	2.783179

In [32]:
```
#Drop University Rating and again calculate VIF
newdf = newdf.drop(columns=['University Rating'])
res = vif(newdf)
res
```

Out[32]:

	feature	VIF
0	CGPA	2.455008
1	Research	2.455008

In [33]:
```
#Again train the model with only these two features
X = df[['CGPA', 'Research']]
sc = StandardScaler()
X = sc.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ran
```

In [34]:
```
model = train_model(X_train, y_train, X_test, y_test, ['CGPA', 'Research'],
train_model(X_train, y_train, X_test, y_test, ['CGPA', 'Research'], "ridge")
train_model(X_train, y_train, X_test, y_test, ['CGPA', 'Research'], "lasso",
```

```
----   Linear Regression Model  ----

Train MAE: 0.05  Test MAE: 0.05
Train RMSE: 0.06  Test RMSE: 0.07
Train R2_score: 0.78  Test R2_score: 0.81
Train Adjusted_R2: 0.78  Test Adjusted_R2: 0.81
Intercept: 0.7247774222727991
     Column  Coefficient
0      CGPA     0.112050
1  Research     0.020205
--------------------------------------------------

----   Ridge Regression Model  ----

Train MAE: 0.05  Test MAE: 0.05
Train RMSE: 0.06  Test RMSE: 0.07
Train R2_score: 0.78  Test R2_score: 0.81
Train Adjusted_R2: 0.78  Test Adjusted_R2: 0.81
Intercept: 0.7247830300095277
     Column  Coefficient
0      CGPA     0.111630
1  Research     0.020362
--------------------------------------------------

----   Lasso Regression Model  ----

Train MAE: 0.05  Test MAE: 0.05
Train RMSE: 0.06  Test RMSE: 0.07
Train R2_score: 0.78  Test R2_score: 0.81
Train Adjusted_R2: 0.78  Test Adjusted_R2: 0.81
Intercept: 0.7247713356661623
     Column  Coefficient
0      CGPA     0.111344
1  Research     0.019571
--------------------------------------------------
```

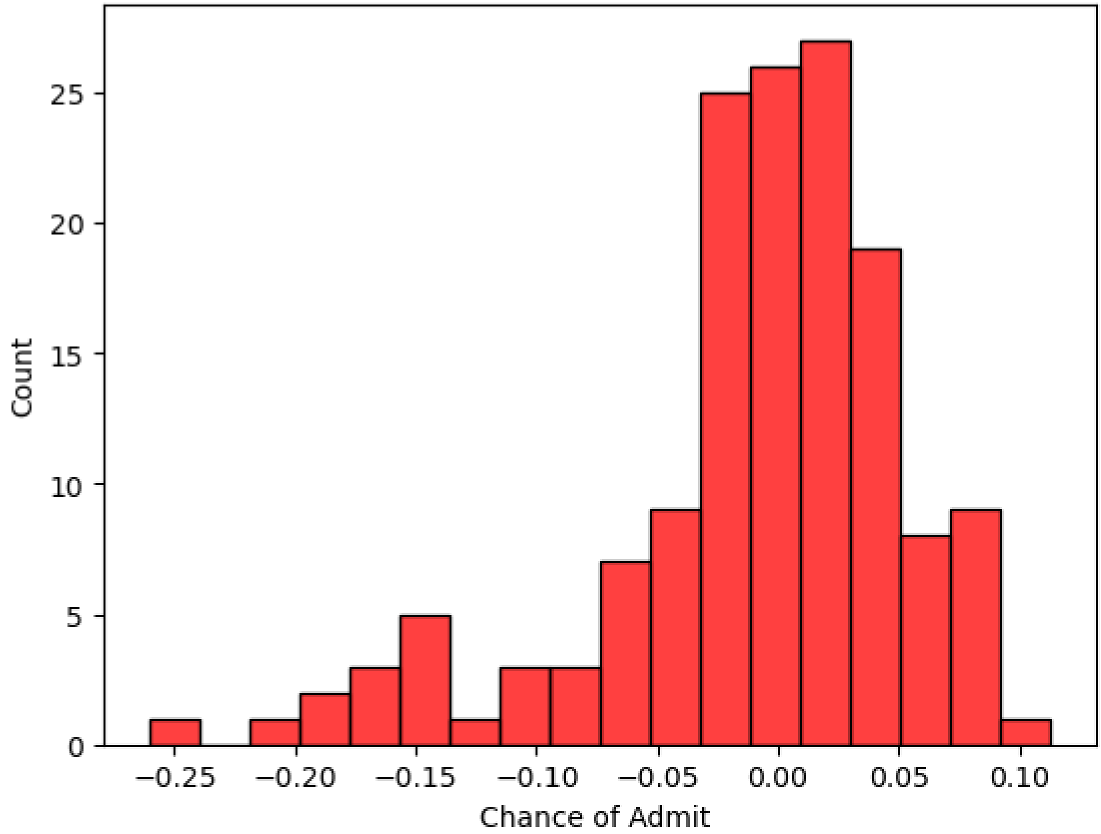
Out[34]:
```
Lasso(alpha=0.001)
```

Observation -

- The R2 score and Adjusted_R2 score are almost same, even after removing collinear features using VIF and using only two features.
- Mean of Residuals : From the RMSE score we can say that the Mean of Residuals is nearly zero
- Linearity of variable : Independant varibles are linearly dependant on the target variable

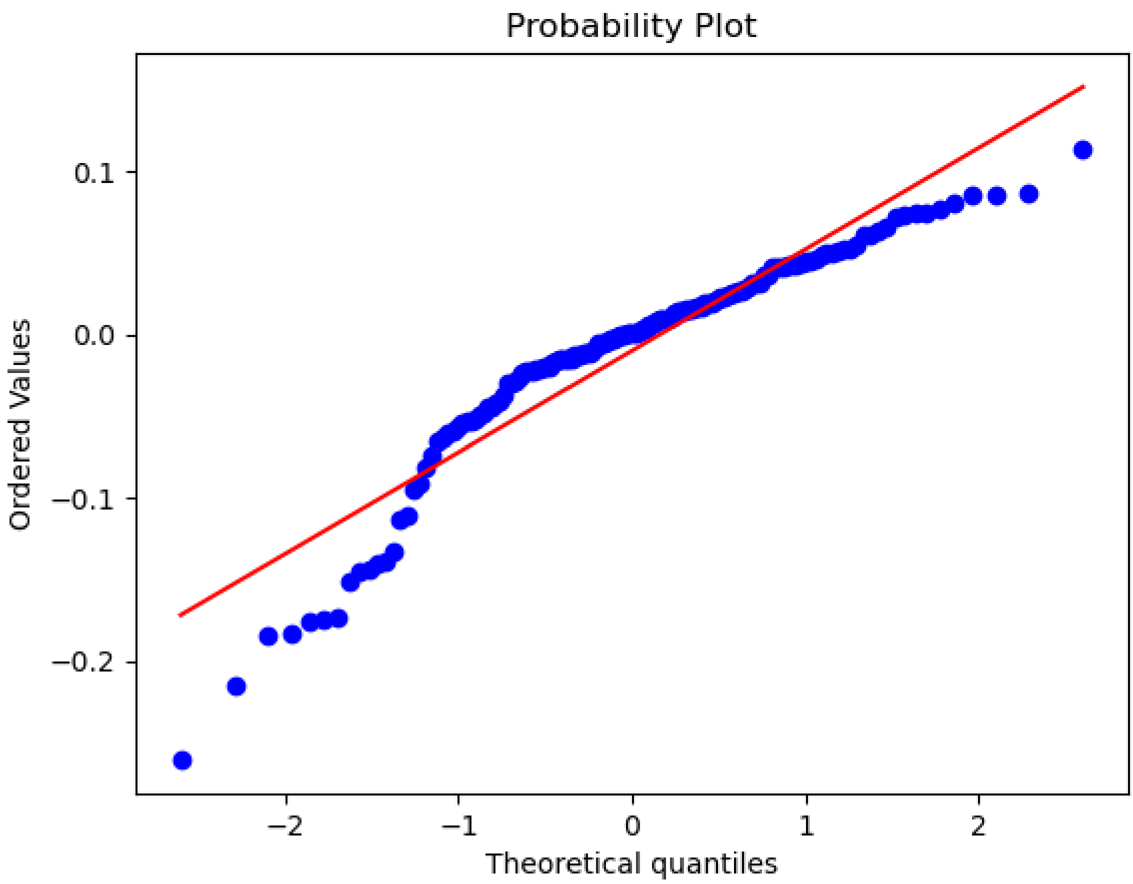
Normality of Residuals

In [35]:
```
y_pred = model.predict(X_test)
residuals = (y_test - y_pred)
sns.histplot(residuals, color='r')
plt.show()
```



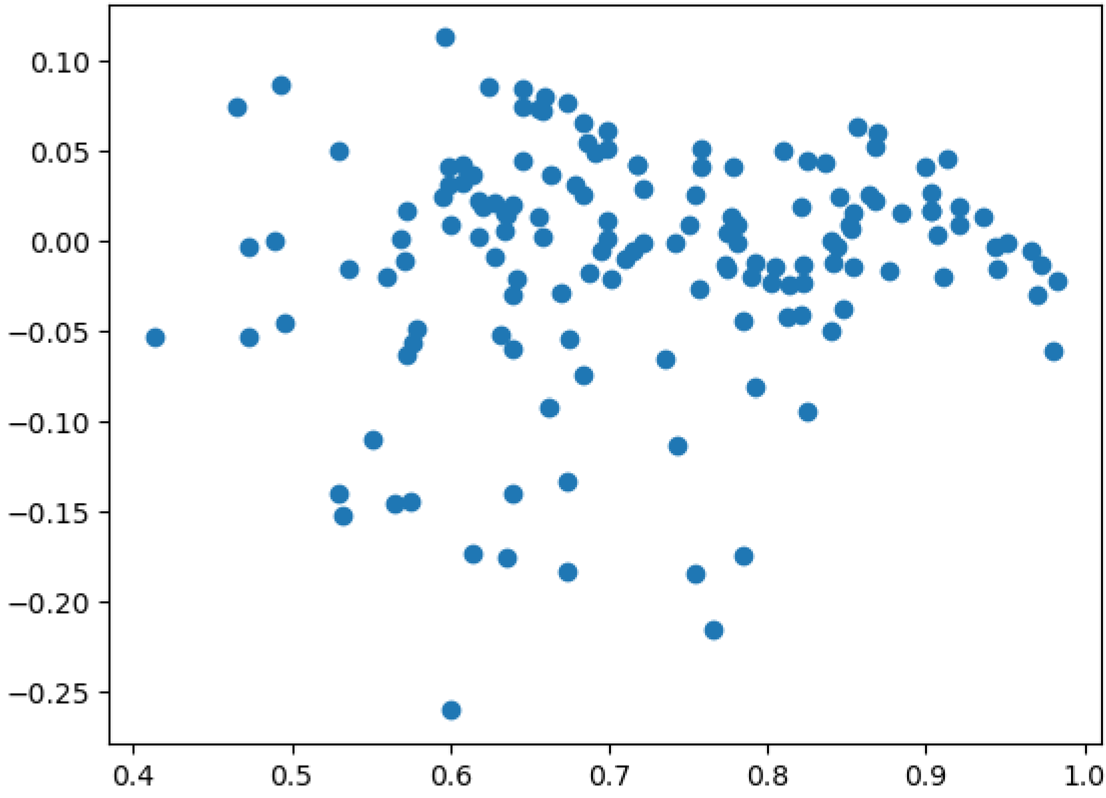
From the histogram we can see that there is a negative skew in the distribution of residuals but it is close to a normal distribution

In [36]:
```
stats.probplot(residuals, plot=plt)
plt.show()
```



Test for Homoscedasticity

In [37]:
```
plt.scatter(y_pred, residuals)
plt.show()
```



There is no homoscedasticity present in the data, because the plot is not creating a cone type shape

Insights -

- Multicollinearity is present in the data.
- The varibles CGPA, GRE Score and TOEFL Score have a strong relationship with the target variable(Chance of Admit). These variables are also highly correlated to themselves.
- Mean of residuals is close to 0.
- Indepedent variables are linearly correlated with dependent variables.

Recommendations -

- CGPA ,GRE Score, Toefl Score are important in making the prediction for Chance of Admit.
- All the exam scores are highly correlated. So it is recommende to add more independent features for better prediction.
- Independent variables like internships, projects completed can be added to the dataset to improve the prediction of the model.