Importing the essential libraries

In [17]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
```

Data loading and EDA

In [18]:
```
df=pd.read_csv('/content/Jamboree - Jamboree.csv')
```

In [19]:
```
df.head(5)
```

Out[19]:

	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 [20]:
```
df.shape
```

Out[20]:
```
(500, 9)
```

In [21]:
```
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 [22]:
```
df1= df.copy()
```

In [23]:
```
df1.drop(columns=['Serial No.'], axis=0, inplace=True)
```

In [24]:
```
df1.head()
```

Out[24]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65

In [25]:
```
df1.rename(columns={'LOR ':'LOR', 'Chance of Admit ':'Chance of Admit'}, inpla
```

In [26]:
```
df1[['University Rating', 'SOP', 'LOR','Research']] = df1[['University Rating'
```

In [27]:
```
df1.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 [28]:
```
df1.describe(include='all')
```

Out[28]:

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

In [29]:
```
df1.duplicated().sum()
```

Out[29]:
```
np.int64(0)
```

In [30]:
```
df1.nunique()
```

Out[30]:

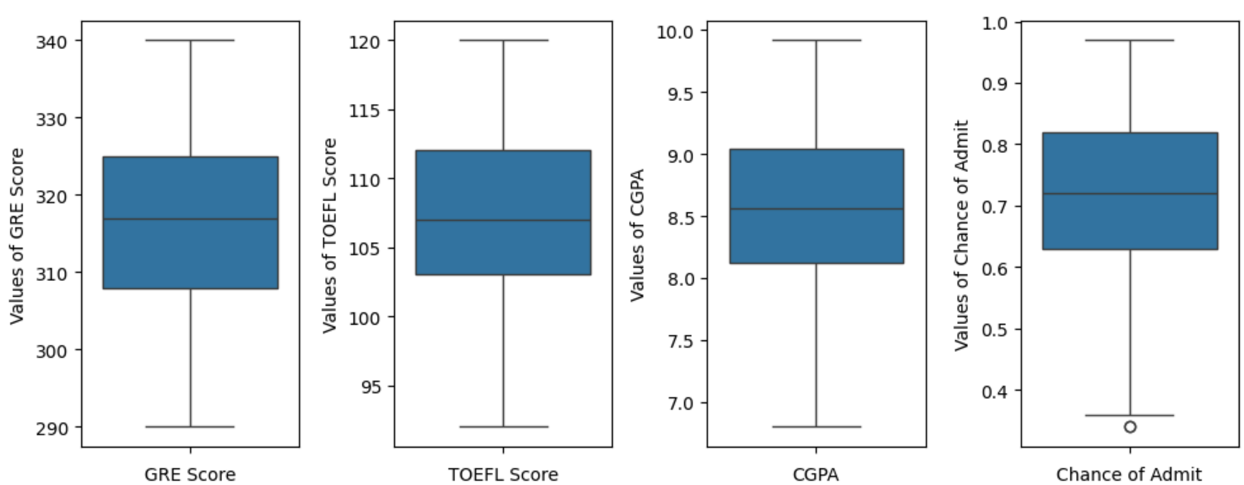
	0
GRE Score	49
TOEFL Score	29
University Rating	5
SOP	9
LOR	9
CGPA	184
Research	2
Chance of Admit	61

dtype: int64

In [31]:
```
numeric_cols = ['GRE Score', 'TOEFL Score', 'CGPA','Chance of Admit']
categorical_cols = ['University Rating', 'SOP', 'LOR','Research']
```

Checking for outliers

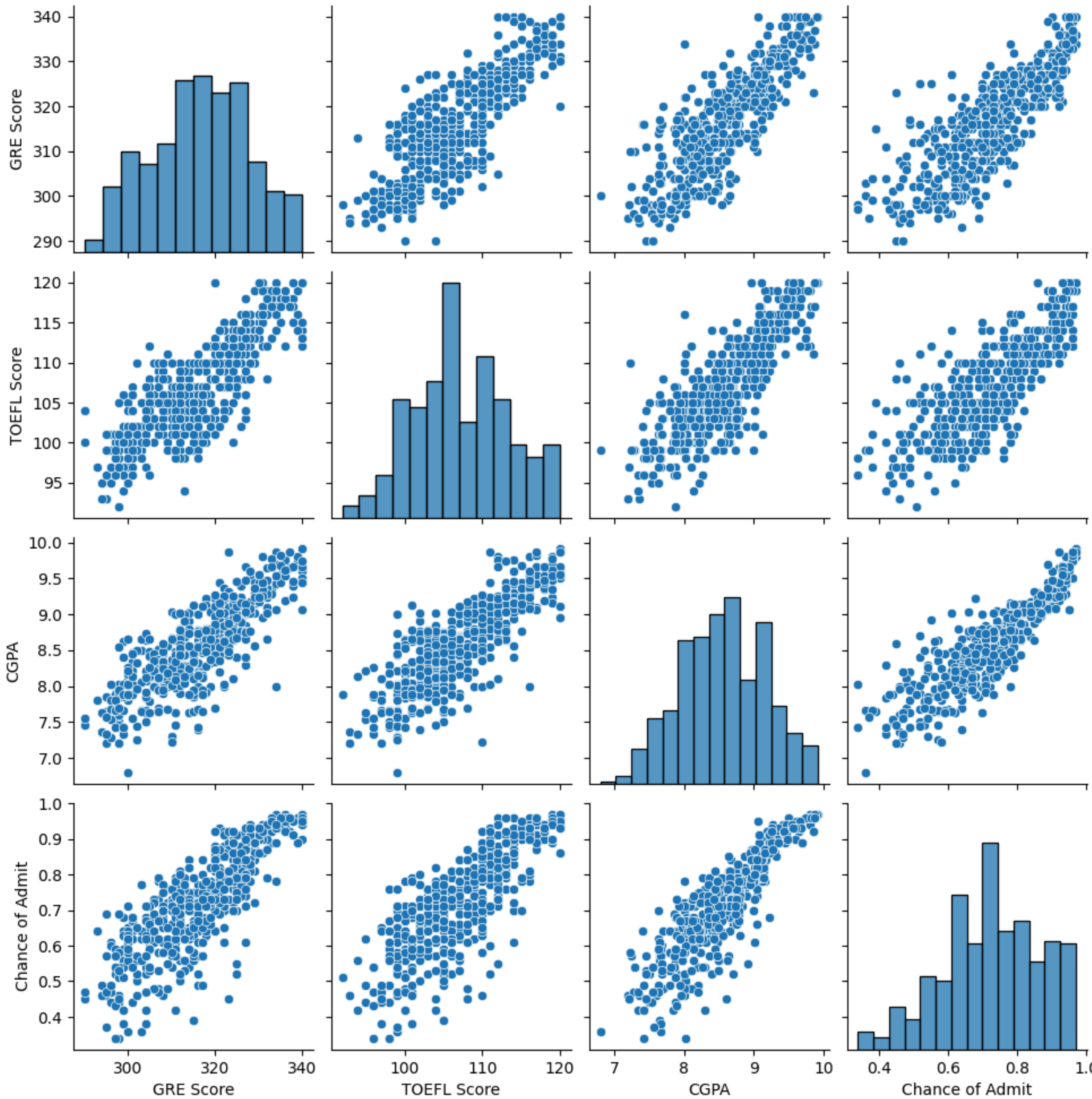
In [32]:
```
plt.figure(figsize=(10,4))
i=1
for col in numeric_cols:
  ax = plt.subplot(1,4,i)
  sns.boxplot(df1[col])
  #plt.title(col)
  plt.xlabel(col)
  plt.ylabel(f'Values of {col}')
  i+=1
plt.tight_layout()
plt.show()
```



There are hardly any outliers

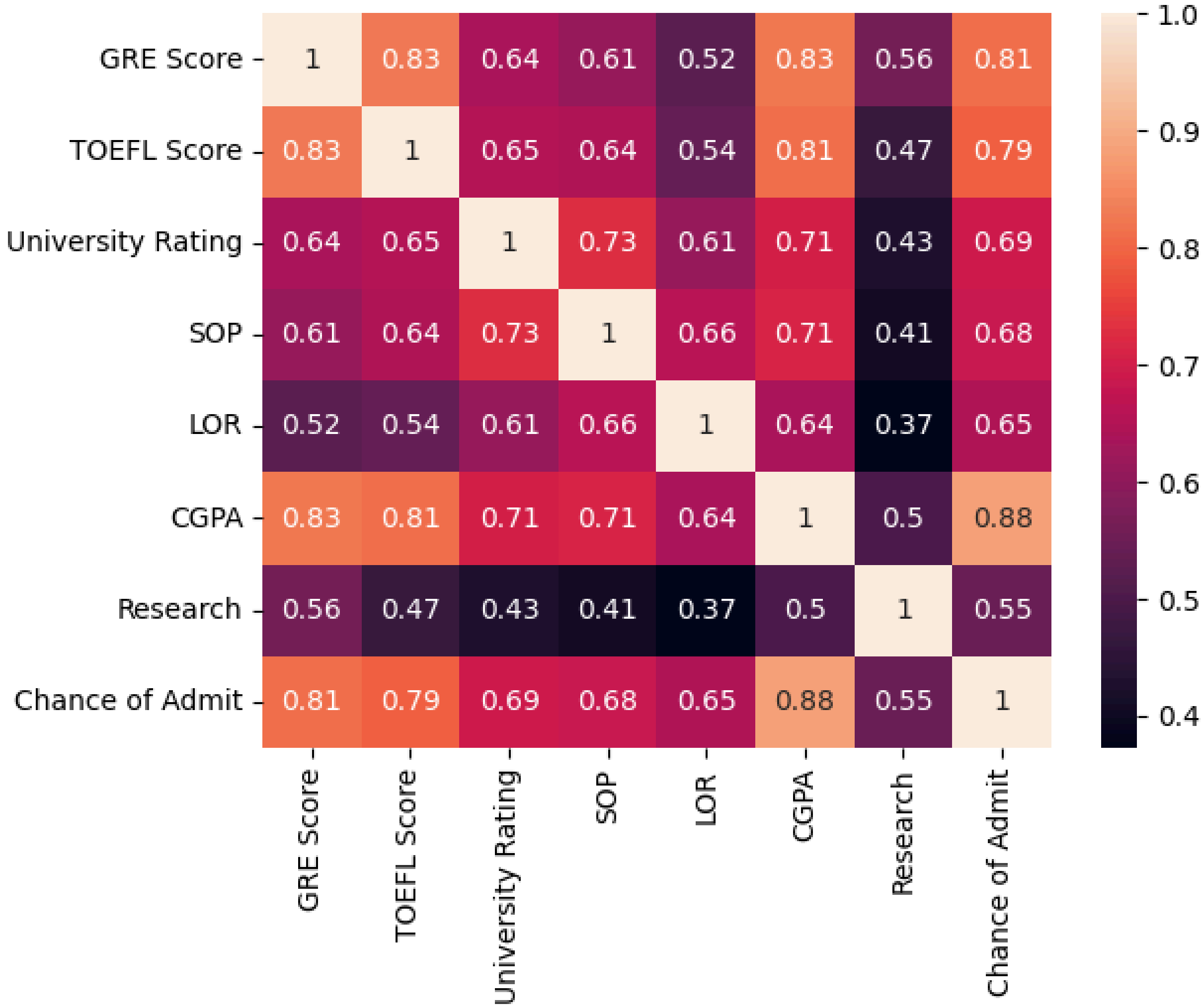
In [33]:
```
sns.pairplot(df1)
```

Out[33]: <seaborn.axisgrid.PairGrid at 0x7b50d40d9510>



In [34]:
```
sns.heatmap(df1.corr(),annot=True)
```

Out[34]: <Axes: >



Observations:

- CGPA, TOEFL Score and GRE Score show strong positive correlation to Chance of Admit
- University Rating show positive correlation to Chance of Admit, CGPA and all other factors except Research
- Chance of Admit show strong positive correlation to CGPA, GRE Score, TOEFL Score
- Research shows weak positive correlation to all other factors

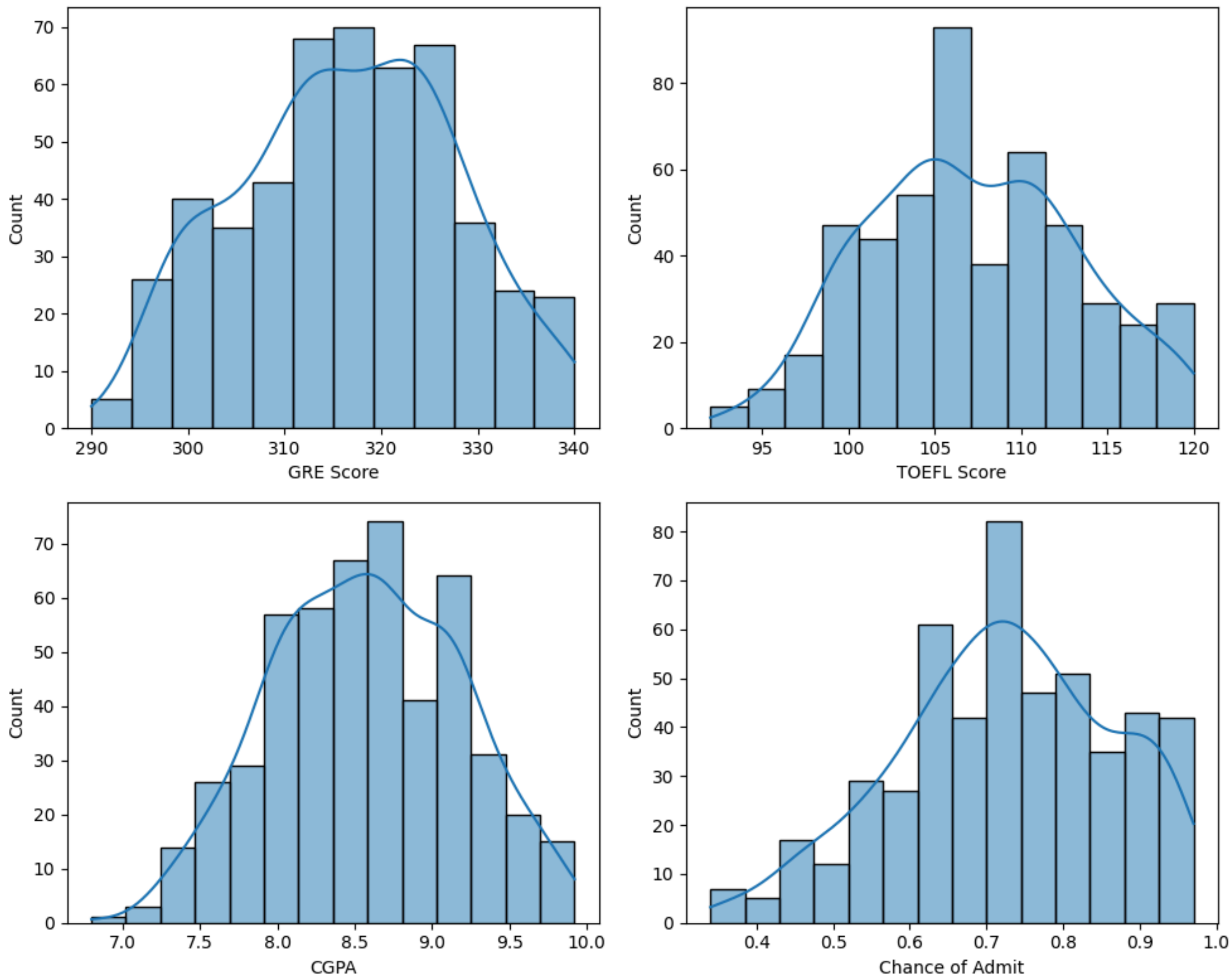
Distribution of Variables

- Univariate Analysis
- Bivariate Analysis

Distribution of Continuous Variables

In [35]:
```
plt.figure(figsize=(10,8))
i=1
for col in numeric_cols:
  ax=plt.subplot(2,2,i)
  sns.histplot(data=df1[col], kde=True)
  plt.xlabel(col)
  plt.ylabel('Count')
  i += 1

plt.tight_layout()
plt.show()
```

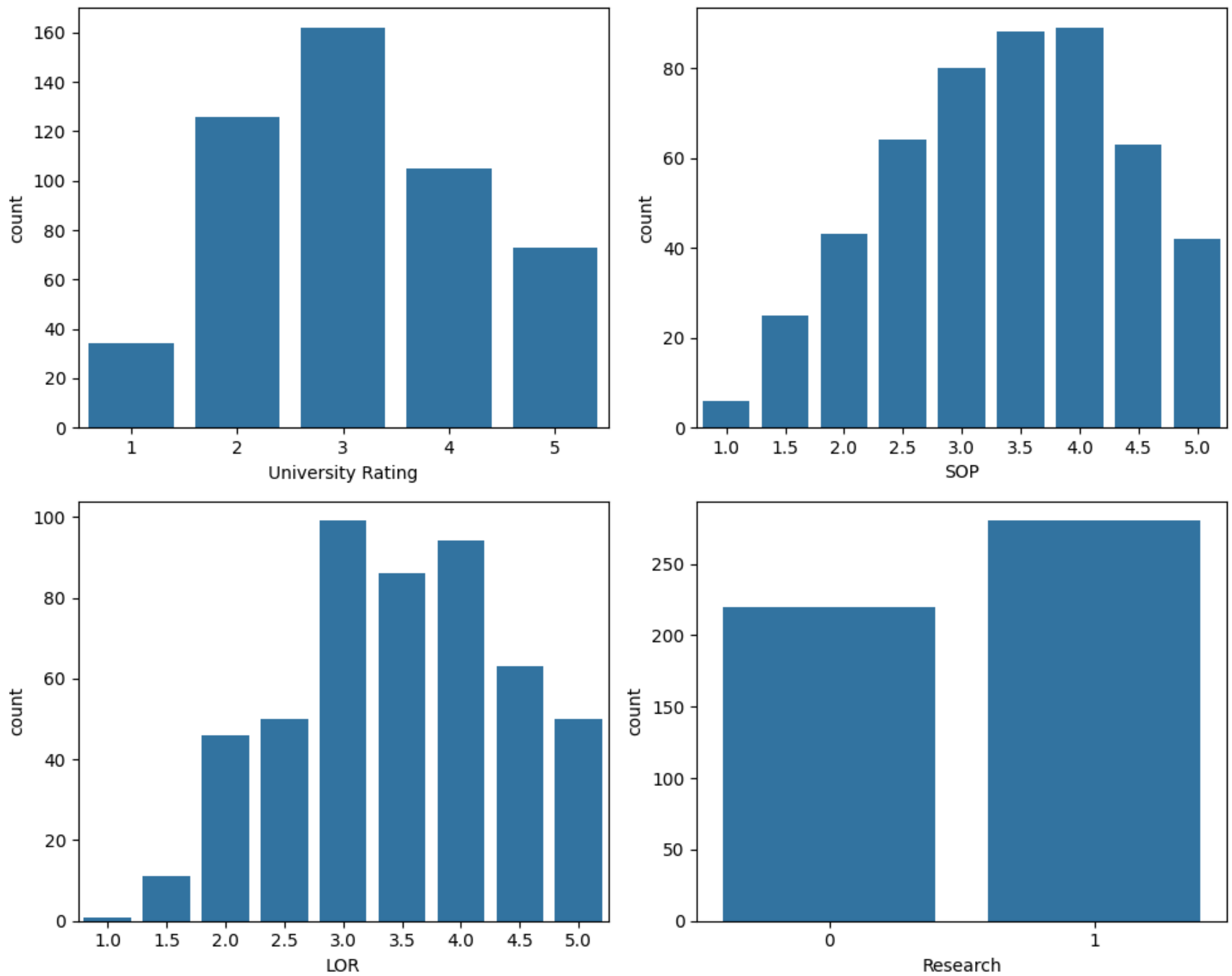


Distribution of categorical variables

In [36]:
```
plt.figure(figsize=(10,8))
i=1
for col in categorical_cols:
  ax=plt.subplot(2,2,i)
  sns.countplot(x=df1[col])
```

```
  plt.xlabel(col)
  plt.ylabel('count')
  i += 1

plt.tight_layout()
plt.show()
```



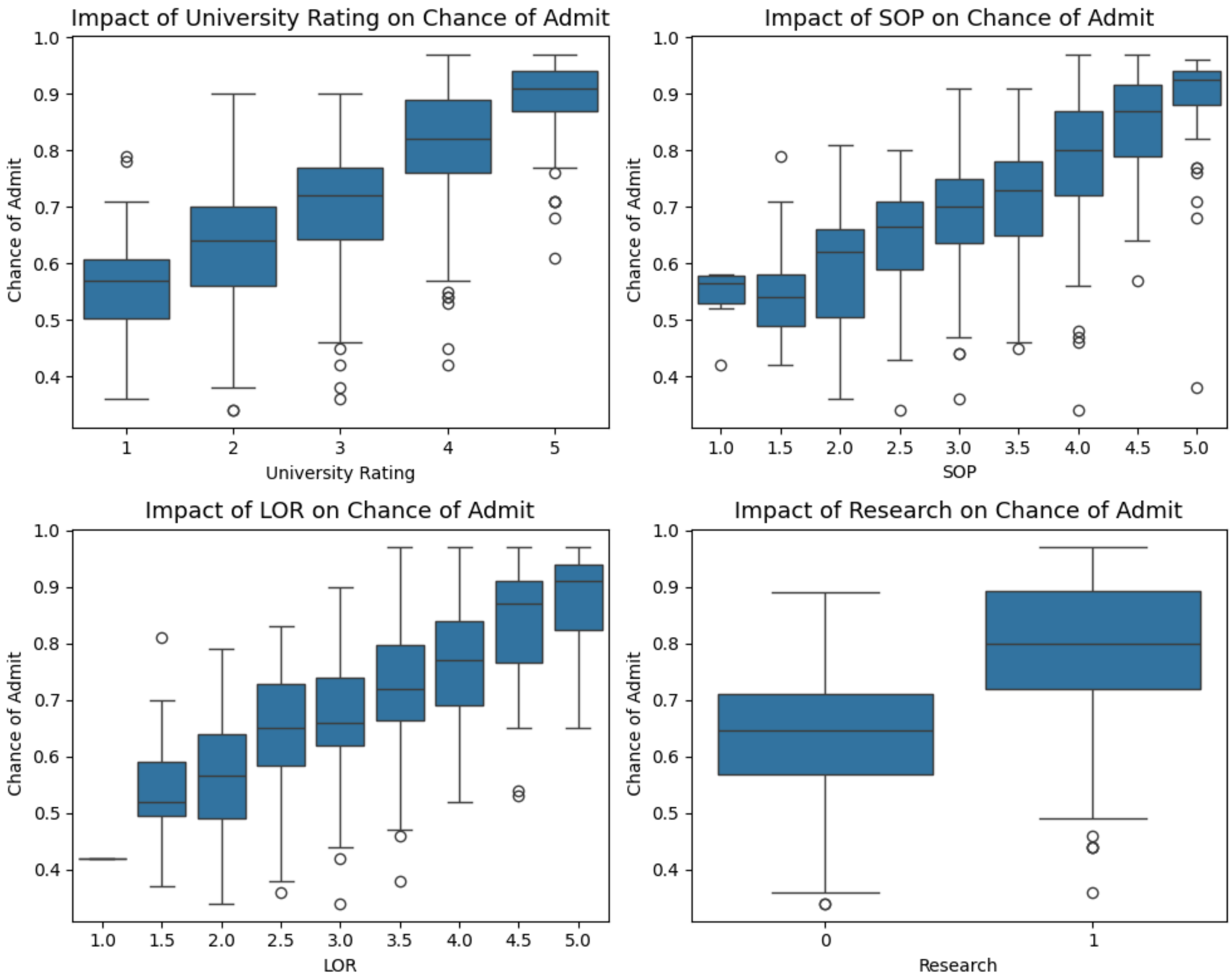
Bi-variate Analysis

impact of categorical columns on chance of admit

In [37]:

```
plt.figure(figsize=(10,8))
i=1
for col in categorical_cols:
  ax=plt.subplot(2,2,i)
  sns.boxplot(data=df1,x=col,y='Chance of Admit')
  plt.title(f"Impact of {col} on Chance of Admit",fontsize=13)
  plt.xlabel(col)
  plt.ylabel(" Chance of Admit")
```

```
  i=i+1
plt.tight_layout()
plt.show()
```



Preparation for modelling

In [38]:
```
df2=df.copy()
```

In [39]:
```
df2.head()
```

Out[39]:

	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 [40]:
```
df2.drop(columns=['Serial No.'], axis=0, inplace=True)
df2.rename(columns={'LOR ':'LOR', 'Chance of Admit ':'Chance of Admit'}, inpla
```

In [41]:
```
X=df2.drop(columns=['Chance of Admit'])
y=df2['Chance of Admit']
```

In [42]:
```
X.shape,y.shape
```

Out[42]:
```
((500, 7), (500,))
```

In [44]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.2,random_state=
```

In [45]:
```
print(f"X_train shape={X_train.shape}")
print(f"X_test shape={X_test.shape}")
print(f"Y_train shape{Y_train.shape}")
print(f"Y_test shape={Y_test.shape}")
```

```
X_train shape=(400, 7)
X_test shape=(100, 7)
Y_train shape(400,)
Y_test shape=(100,)
```

In [46]:
```
scaler=StandardScaler()
X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled=scaler.transform(X_test)
```

In [47]:
```
X_sm=sm.add_constant(X_train_scaled)
sm_model=sm.OLS(Y_train,X_sm).fit()
print(sm_model.summary())
```

```
                            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:                Fri, 06 Jun 2025   Prob (F-statistic):          3.41e-142
Time:                        19:35:10   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
x1             0.0267      0.006      4.196      0.000       0.014       0.039
x2             0.0182      0.006      3.174      0.002       0.007       0.030
x3             0.0029      0.005      0.611      0.541      -0.007       0.012
x4             0.0018      0.005      0.357      0.721      -0.008       0.012
x5             0.0159      0.004      3.761      0.000       0.008       0.024
x6             0.0676      0.006     10.444      0.000       0.055       0.080
x7             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 correctl
y specified.
```

In [48]:

```
column_names= list(X_train.columns)
model_parameters=list(sm_model.params[1:])
modelparam2=list(sm_model.pvalues[1:])
coefficients=pd.DataFrame({'Variables':column_names,'Coefficient':model_parame
print(coefficients)
```

```
            Variables  Coefficient       P_value
0           GRE Score     0.026671  3.357625e-05
1         TOEFL Score     0.018226  1.619658e-03
2   University Rating     0.002940  5.414408e-01
3                 SOP     0.001788  7.211636e-01
4                 LOR     0.015866  1.947965e-04
5                CGPA     0.067581  1.086636e-22
6            Research     0.011940  1.337508e-03
```

It is clear from above that 'University Rating' and 'SOP' have p-value>0.05 , signifying that these two features have no statistically significant effect on the dependent variable

Therefore, we shall remove these two features and re-train the model

In [49]:
```
df3=df.copy()
```

In [50]:
```
df3.rename(columns={'LOR ':'LOR', 'Chance of Admit ':'Chance of Admit'}, inpla
```

In [51]:
```
df3.head()
```

Out[51]:

	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 [52]:
```
df3.drop('Serial No.',inplace=True,axis=1)
df3.drop('University Rating',inplace=True,axis=1)
df3.drop('SOP',inplace=True,axis=1)
```

In [53]:
```
df3.head()
```

Out[53]:

	GRE Score	TOEFL Score	LOR	CGPA	Research	Chance of Admit
0	337	118	4.5	9.65	1	0.92
1	324	107	4.5	8.87	1	0.76
2	316	104	3.5	8.00	1	0.72
3	322	110	2.5	8.67	1	0.80
4	314	103	3.0	8.21	0	0.65

In [54]:
```
X=df3.drop(columns=['Chance of Admit'])
Y=df3['Chance of Admit']
```

In [80]:
```
Y
```

Out[80]:

	Chance of Admit
0	0.92
1	0.76
2	0.72
3	0.80
4	0.65
...	...
495	0.87
496	0.96
497	0.93
498	0.73
499	0.84

500 rows × 1 columns

dtype: float64

In [55]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.2,random_state=
```

In [56]:
```
print(f"X_train shape={X_train.shape}")
print(f"X_test shape={X_test.shape}")
print(f"Y_train shape{Y_train.shape}")
print(f"Y_test shape={Y_test.shape}")
```

```
X_train shape=(400, 5)
X_test shape=(100, 5)
Y_train shape(400,)
Y_test shape=(100,)
```

In [57]:
```
scaler=StandardScaler()
X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled=scaler.transform(X_test)
```

In [58]:
```
X_sm=sm.add_constant(X_train_scaled)
sm_model=sm.OLS(Y_train,X_sm).fit()
print(sm_model.summary())
```

```
                            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:                Fri, 06 Jun 2025   Prob (F-statistic):          1.36e-144
Time:                        19:36:55   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
x1             0.0269      0.006      4.245      0.000       0.014       0.039
x2             0.0191      0.006      3.391      0.001       0.008       0.030
x3             0.0172      0.004      4.465      0.000       0.010       0.025
x4             0.0691      0.006     11.147      0.000       0.057       0.081
x5             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 correctl
y specified.
```

In [59]:

```python
column_names= list(X_train.columns)
model_parameters=list(sm_model.params[1:])
modelparam2=list(sm_model.pvalues[1:])
coefficients=pd.DataFrame({'Variables':column_names,'Coefficient':model_parame
print(coefficients)
```

```
     Variables  Coefficient       P_value
0    GRE Score     0.026879  2.731841e-05
1  TOEFL Score     0.019106  7.667483e-04
2          LOR     0.017207  1.045150e-05
3         CGPA     0.069066  2.882599e-25
4     Research     0.012226  9.557871e-04
```

Observations:

- We will proceed with the selected **5 features**, as their p-values are less than `0.05`, indicating they have a **statistically significant effect** on the dependent variable.
- **R-squared** and **Adjusted R-squared** are nearly the same, suggesting that approximately **82% of the variance** in the dependent variable is explained by the independent variables.

- **Condition Number** is reduced to **4.76**, which is well below the threshold of 30, indicating that **there is no multicollinearity** present in the model.
- The **low Prob(F-statistic)** value confirms that the **overall model is statistically significant**.
- Among the features:
    - **CGPA** carries the **maximum weightage**
    - Followed by **GRE Score**, **TOEFL Score**, **LOR**, and **Research**,
      indicating the strength of their relationship with the dependent variable.

Test Assumptions of Linear Regression

Multicollinearity check

In [60]:
```
vif = pd.DataFrame()
X_t = pd.DataFrame(X_train_scaled, columns=X_train.columns)
vif['Features'] = X_t.columns
vif['VIF'] = [variance_inflation_factor(X_t.values, i) for i in range(X_t.shap
vif['VIF']=round(vif['VIF'],2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[60]:

	Features	VIF
0	GRE Score	4.47
3	CGPA	4.28
1	TOEFL Score	3.54
2	LOR	1.66
4	Research	1.50

VIF Value of all Features are well below 5 which means this model is good to go ahead

*** Mean of residue ***

In [61]:
```
X_test_sm=sm.add_constant(X_test_scaled)
Y_test_pred=sm_model.predict(X_test_sm)
```

In [62]:
```python
Y_test_values=Y_test.values.flatten()
```

In [63]:
```python
residual_test=Y_test_values-Y_test_pred
```

In [64]:
```python
mean_residual=np.mean(residual_test)
```

In [65]:
```python
print(f"Mean of Residuals={mean_residual}")
```

```
Mean of Residuals=-0.005305947942349201
```

◈ Residual Analysis:

- The **mean of residuals** on the test data is **-0.0053**, which is close to zero.
- This slight negative value suggests a **small tendency for underestimation**.
- However, the **magnitude is very small**, indicating that the model's predictions are generally **very close to the actual values**, and the bias is minimal.

Test for Homoscedasticity

In [66]:
```python
Y_hat=sm_model.predict(X_sm)
```

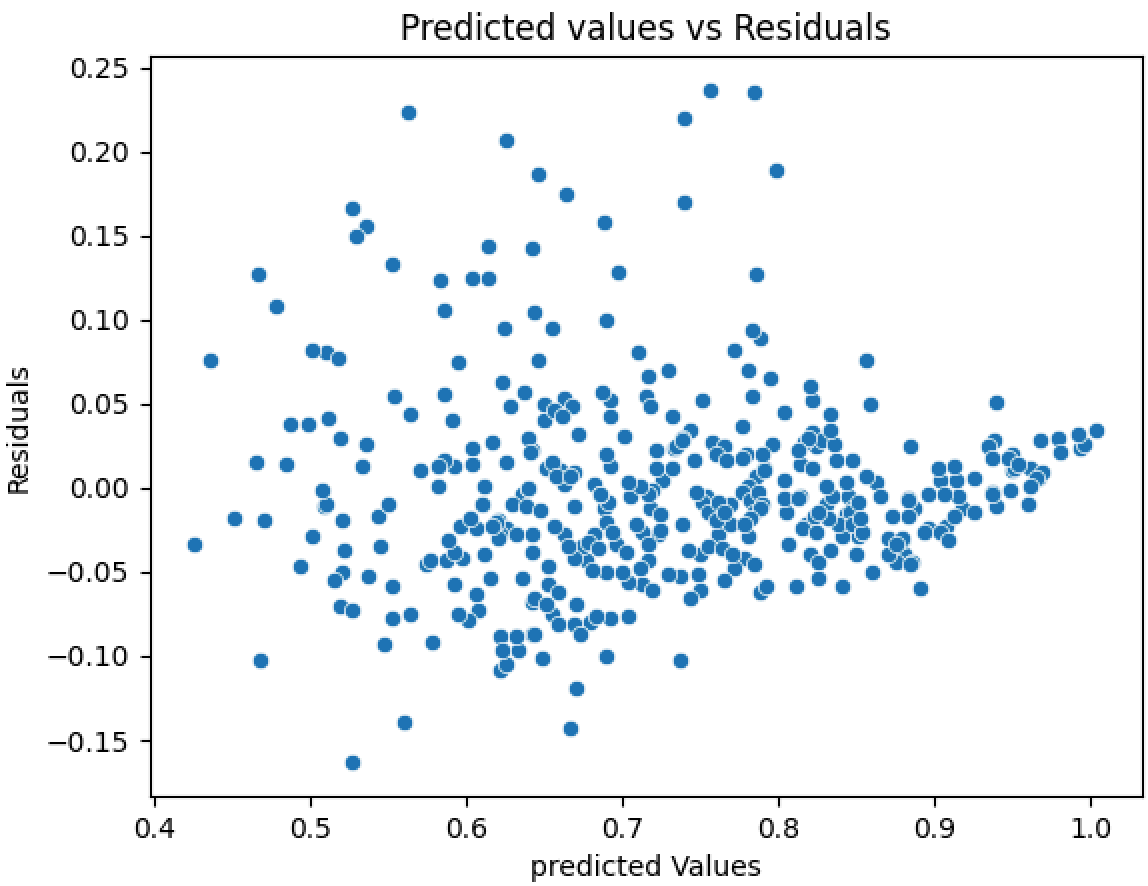
In [72]:
```python
if isinstance(Y_train, pd.Series):
    print("y_train is a Pandas Series")
elif isinstance(Y_train, pd.DataFrame):
    print("y_train is a Pandas DataFrame")
else:
    print("y_train is neither a Pandas Series nor a Pandas DataFrame")
```

```
y_train is a Pandas DataFrame
```

In [73]:
```python
Y_trains_columns=Y_train['Chance of Admit']
```

In [74]:
```python
errors=Y_hat-Y_trains_columns
```

In [75]:
```python
sns.scatterplot(x=Y_hat,y=errors)
plt.xlabel("predicted Values")
plt.ylabel("Residuals")
plt.title("Predicted values vs Residuals")
plt.show()
```



Using **Goldfeld Quandt Test** to check **homoskedacity**

In [144…
```
from statsmodels.compat import lzip
import statsmodels.stats.api as sms
```

In [145…
```
name=['F-statistics','P_value']
```

In [147…
```
test = sms.het_goldfeldquandt(Y_train, X_sm)
lzip(name, test)
```

Out[147…
```
[('F-statistics', np.float64(0.9592288620962849)),
 ('P_value', np.float64(0.6139024845884469))]
```

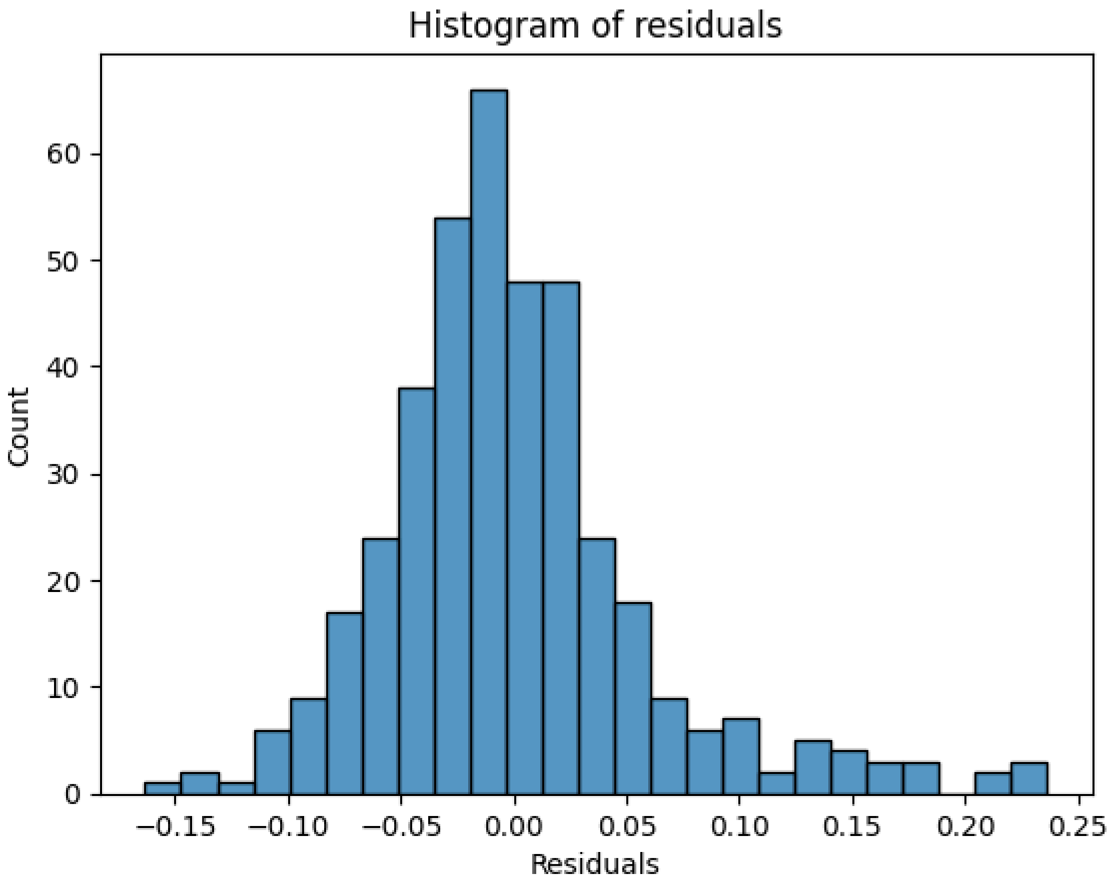
F Statistic comes out to be 0.959 => Implying minimal difference in variance between groups.

p-value of 0.613 indicates that this difference is statistically significant at conventional levels of significance (e.g., 0.05).

Therefore, we accept the null hypothesis of homoscedasticity, and conclude that there is no strong evidence of heteroscedasticity in the data.

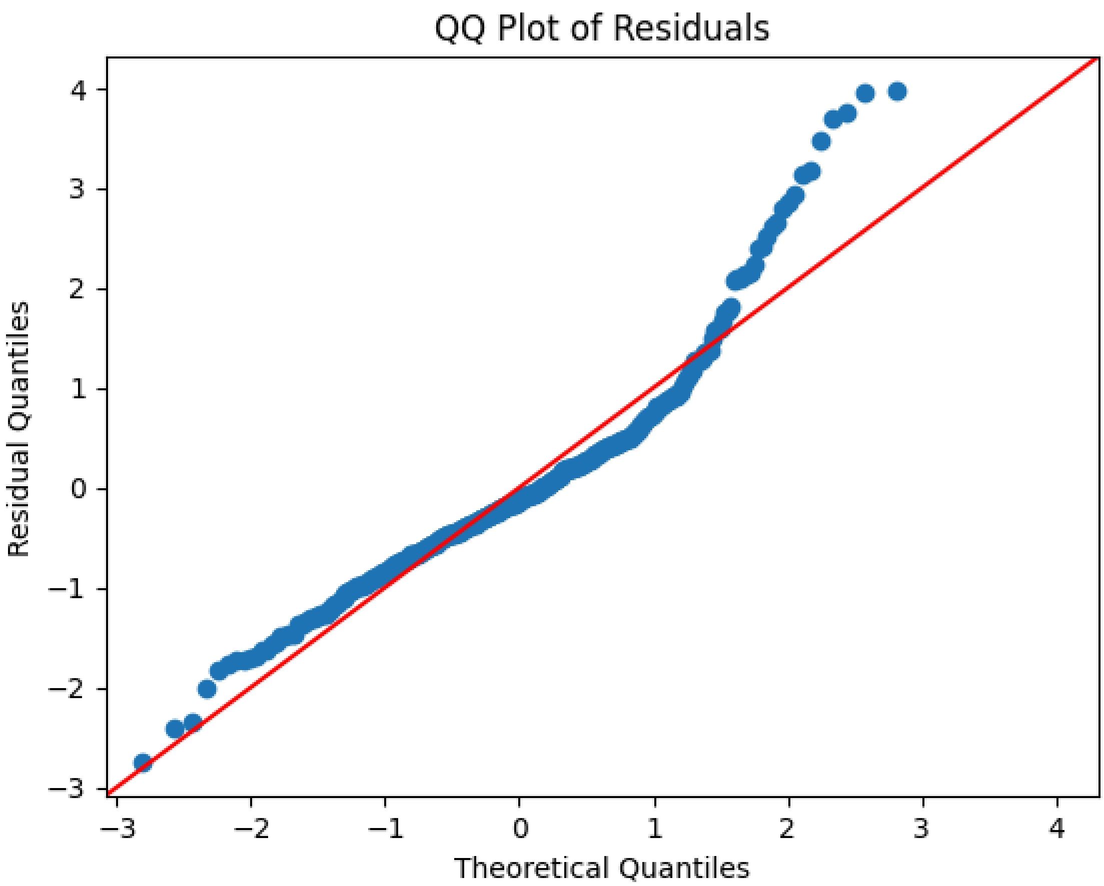
In [149…
```
sns.histplot(errors)
```

```
plt.title('Histogram of residuals')
plt.xlabel('Residuals')
plt.ylabel('Count')
plt.show()
```



In [150…

```
sm.qqplot(errors,line='45',fit=True)
plt.title('QQ Plot of Residuals')
plt.ylabel('Residual Quantiles')
plt.show()
```



Shapiro wilk Test of Normality

In [151…
```
from scipy import stats
```

In [155…
```
res=stats.shapiro(errors)
res.statistic
```

Out[155…
```
np.float64(0.931256678230213)
```

Since the value is closer to 1 denotes a high level of normality for the error distribuiton

Evaluate model performance

In [170…
```
def adjusted_r2_score(y_true, y_pred, n_features):
    n = len(y_true)
    r2 = r2_score(y_true, y_pred)
    adjusted_r2 = 1 - ((1 - r2) * (n - 1) / (n - n_features - 1))
    return adjusted_r2

# MAE (Mean Absolute Error)
train_mae = mean_absolute_error(Y_train, Y_hat)
```

```python
test_mae = mean_absolute_error(Y_test, Y_test_pred)

# RMSE (Root Mean Square Error)
train_rmse = np.sqrt(mean_squared_error(Y_train, Y_hat))
test_rmse = np.sqrt(mean_squared_error(Y_test, Y_test_pred))

# R-squared value
train_r2 = r2_score(Y_train, Y_hat)
test_r2 = r2_score(Y_test, Y_test_pred)

# Number of features in the model (assuming X_train is your feature matrix)
n_features = X_train_scaled.shape[1]

# Adjusted R-squared value
train_adj_r2 = adjusted_r2_score(Y_train, Y_hat, n_features)
test_adj_r2 = adjusted_r2_score(Y_test, Y_test_pred, n_features)

print("Training set:")
print("MAE:", train_mae)
print("RMSE:", train_rmse)
print("R-squared:", train_r2)
print("Adjusted R-squared:", train_adj_r2)

print("\nTesting set:")
print("MAE:", test_mae)
print("RMSE:", test_rmse)
print("R-squared:", test_r2)
print("Adjusted R-squared:", test_adj_r2)
```

```
Training set:
MAE: 0.04269126483606392
RMSE: 0.05944028044169098
R-squared: 0.8207326947514393
Adjusted R-squared: 0.8184577289487925

Testing set:
MAE: 0.042923455782657785
RMSE: 0.06142491974041883
R-squared: 0.8155002070847485
Adjusted R-squared: 0.8056863883126606
```

- The model performs well on both the training and testing sets, as indicated by small MAE and RMSE values and high R-squared and adjusted R-squared values.
- The model appears to generalize well to unseen data (testing set), as the performance metrics on the testing set are comparable to those on the training set.

Final Insights and Recommendations: Jamboree Admission Prediction Model

◈ Insights

- **Model Accuracy**: The Multiple Linear Regression model performs well with high R^2 and Adjusted R^2 values (~0.82), indicating that 82% of the variance in the target variable ('Chance of Admit') is explained by the model.
- **Key Influencing Factors**:
    - **CGPA**, **GRE Score**, and **TOEFL Score** have a strong positive correlation with the chance of admission.
    - **Letter of Recommendation (LOR)** and **Research Experience** also contribute meaningfully.
- **Statistically Insignificant Features**:
    - **University Rating** and **SOP Strength** have high p-values (> 0.05), suggesting no significant impact on the target variable. These were removed to improve model performance.
- **Data Distribution**:
    - Most students belong to universities with a rating of 3, followed by 2 and 4.
    - SOP strengths are most frequently at level 4, followed by 3.5 and 3.
    - Most students had a LOR strength of 3.
    - A significant portion of students had research experience.
- **Admission Likelihood**:
    - Highest for students with:
        - University Rating 5
        - SOP and LOR strengths of 5
        - Research experience
- **Residual Analysis**: Residuals are centered around zero, indicating unbiased and accurate predictions.
- **Multicollinearity**: Condition Number = 4.76 (well below 30), indicating that features are independent of one another.

⟐ Recommendations

1. **Focus on Key Predictors**

- Concentrate on improving and capturing CGPA, GRE, TOEFL, LOR, and Research in future models.
- Exclude non-significant features like SOP and University Rating to reduce noise.

2. **Enhance Data Collection**

- Consider adding more features such as:
    - Academic program details
    - Extracurriculars
    - Internship or work experience
    - Demographic and geographic diversity
- Collaborate with partner institutions to access more granular data.

3. **Build a Predictive Tool**

- Deploy this model on Jamboree’s website as a user-facing calculator.
- Allow students to input their data and receive a predicted chance of admission.
- Clearly explain which factors are most influential and why.

4. **Improve Student Support**

- Use predictions to recommend personalized test prep services or counseling.
- Focus on students in the medium-likelihood segment with targeted content and guidance.

5. **Business and Operational Impact**

- Boost credibility by offering a scientifically-backed prediction engine.
- Drive more engagement through interactive tools that offer genuine value.
- Increase test prep enrollments by showing students their weaknesses and how Jamboree can help.

6. **Model Maintenance**

- Retrain the model with new data every 6–12 months to adapt to trends in admissions.
- Incorporate user feedback to refine prediction accuracy and usability.
- Explore more advanced regression techniques or ensemble models as data volume grows.

By applying these insights, Jamboree can establish itself as a trusted education brand, offering not just test prep but intelligent, personalized admission guidance.

In [ ]:
```

```