In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import scipy as stats
```

In [3]:
```
df_jamboree=pd.read_csv('Jamboree.csv')
df_jamboree.sample(10)
```

Out[3]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
257	258	324	100	3	4.0	5.0	8.64	1	0.78
85	86	319	103	4	4.5	3.5	8.66	0	0.76
99	100	323	113	3	4.0	4.0	8.88	1	0.79
159	160	297	100	1	1.5	2.0	7.90	0	0.52
445	446	328	116	5	4.5	5.0	9.08	1	0.91
218	219	324	110	4	3.0	3.5	8.97	1	0.84
14	15	311	104	3	3.5	2.0	8.20	1	0.61
340	341	312	107	3	3.0	3.0	8.46	1	0.75
346	347	304	97	2	1.5	2.0	7.64	0	0.47
467	468	318	101	5	3.5	5.0	8.78	1	0.78

In [4]:
```
#shape of datasets
print("No of Rows:",df_jamboree.shape[0])
print("No of Columns:",df_jamboree.shape[1])
```

```
No of Rows: 500
No of Columns: 9
```

In [5]:
```
df_jamboree.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
```

Detect Null values and outliers

In [8]:
```
df_jamboree.isna().sum()
```

Out[8]:
```
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
```

checking duplicate values

In [11]:
```
duplicate_rows=df_jamboree[df_jamboree.duplicated()]
print(duplicate_rows.shape[0])
```

```
0
```

Data Exploration

In [13]:
```
# Statistical summary of the dataset -
df_jamboree.describe(include='all').T
```

Out[13]:

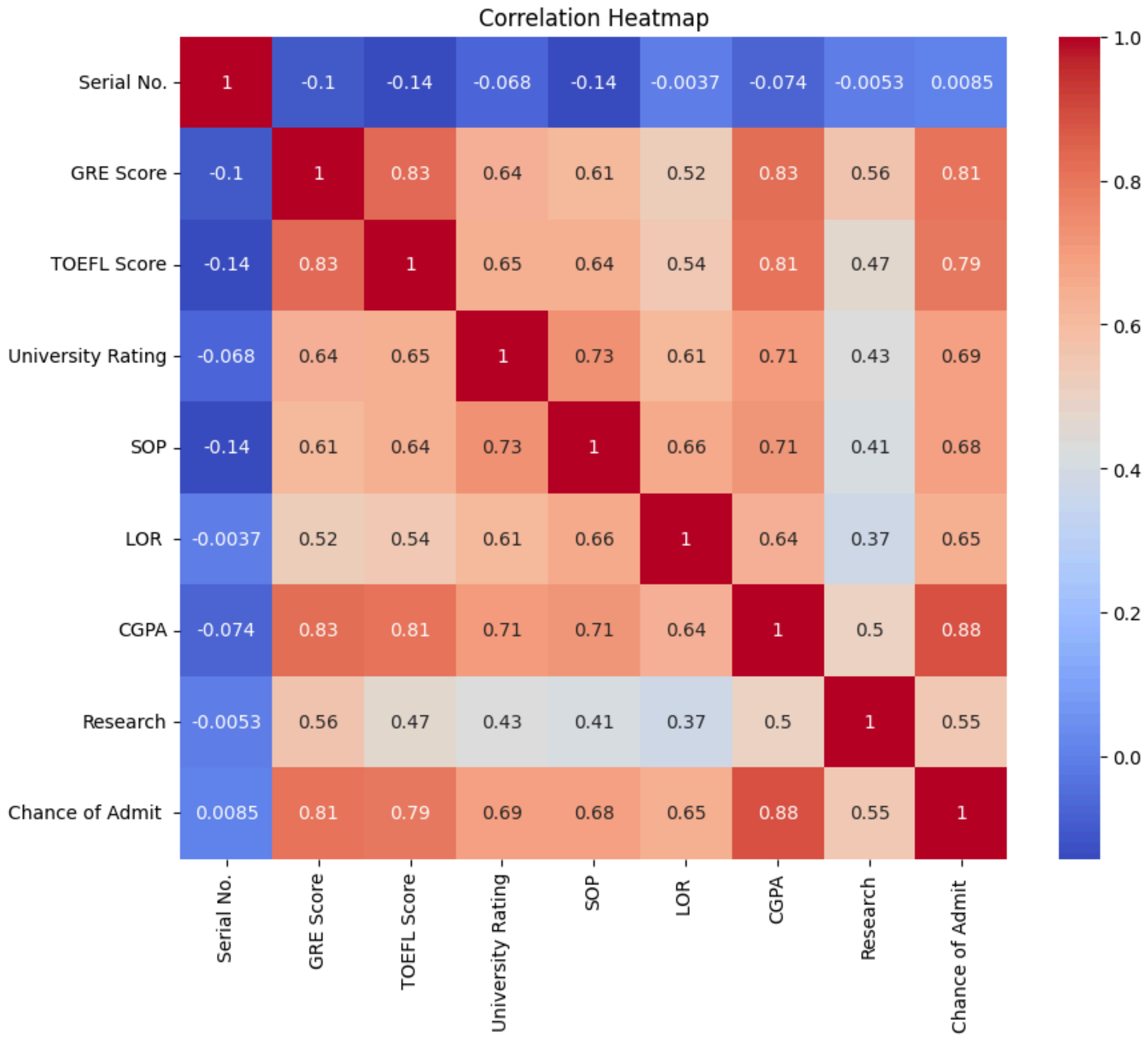
	count	mean	std	min	25%	50%	75%	max
Serial No.	500.0	250.50000	144.481833	1.00	125.7500	250.50	375.25	500.00
GRE Score	500.0	316.47200	11.295148	290.00	308.0000	317.00	325.00	340.00
TOEFL Score	500.0	107.19200	6.081868	92.00	103.0000	107.00	112.00	120.00
University Rating	500.0	3.11400	1.143512	1.00	2.0000	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 [14]:
```
df_jamboree.head()
```

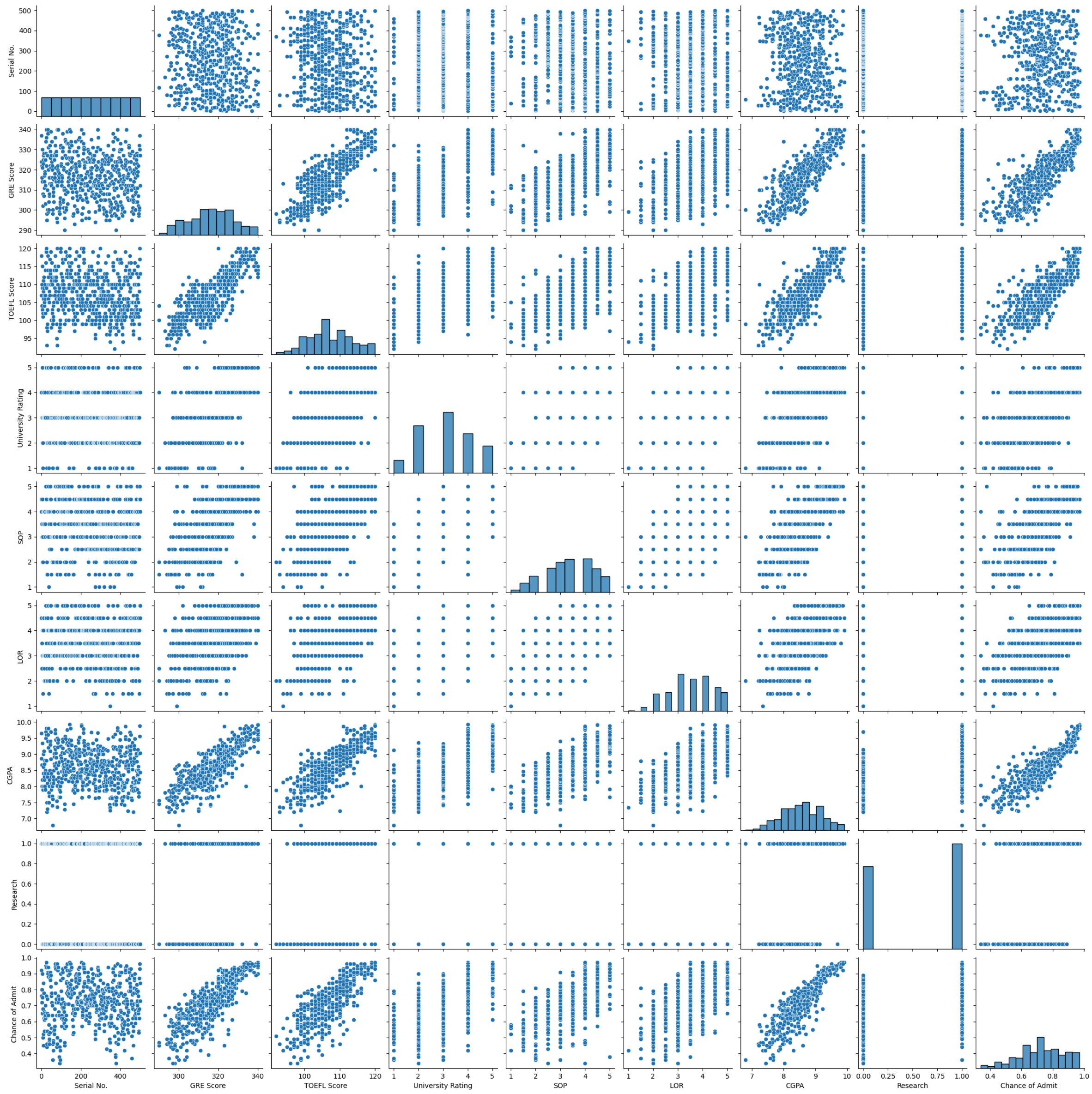
Out[14]:

	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 [15]:
```
# Step 5: Correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df_jamboree.corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```



In [17]:
```
# Step 6: Pairplot of dataset
sns.pairplot(df_jamboree)
plt.show()
```



In [20]:
```
df_jamboree.columns
```

Out[20]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

Test the assumptions of linear regression

In [35]:
```
#Multicollinearity check by VIF score
# Drop non-numeric or identifier columns
df_clean = df_jamboree.drop(columns=["Serial No."])  # drop Serial No.
```

In [37]:
```
# Independent variables only (exclude target)
X = df_clean.drop(columns=["Chance of Admit "])
```

In [39]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import StandardScaler
```

In [40]:
```
# Standardize the features (VIF is sensitive to scale)
scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
```

In [41]:
```
# Function to compute VIFs and drop variables > 5 iteratively
def calculate_vif(X):
    vif = pd.DataFrame()
    vif["Variable"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shap
    return vif
```

In [42]:
```
max_vif = 10  # start with any number > 5
while max_vif > 5:
    vif_data = calculate_vif(X_scaled)
    max_vif = vif_data["VIF"].max()
    if max_vif > 5:
        drop_variable = vif_data.sort_values("VIF", ascending=False).iloc[0]['
        print(f"Dropping '{drop_variable}' with VIF: {max_vif:.2f}")
        X_scaled = X_scaled.drop(columns=[drop_variable])
    else:
        print("All VIFs are below or equal to 5.")
```

```
All VIFs are below or equal to 5.
```

In [43]:
```
print("\nFinal variables after VIF filtering:")
print(X_scaled.columns.tolist())
```

```

Final variables after VIF filtering:
['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA', 'Re
search']
```

In [ ]:

In [44]:
```
# Step 7: Define features and target
X1 = df_jamboree[['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOF
y1 = df_jamboree['Chance of Admit ']
#Note Serial No is not significant feature so not considering for model traini
```

In [45]:
```
# Step 8: Split the data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size=0.2, rar
```

In [46]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
```

In [47]:
```
# Step 9: Train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[47]:
```
▾ LinearRegression
LinearRegression()
```

In [48]:
```
# Step 10: Predictions
y_pred = model.predict(X_test)
```

In [49]:
```
# Step 11: Model Evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

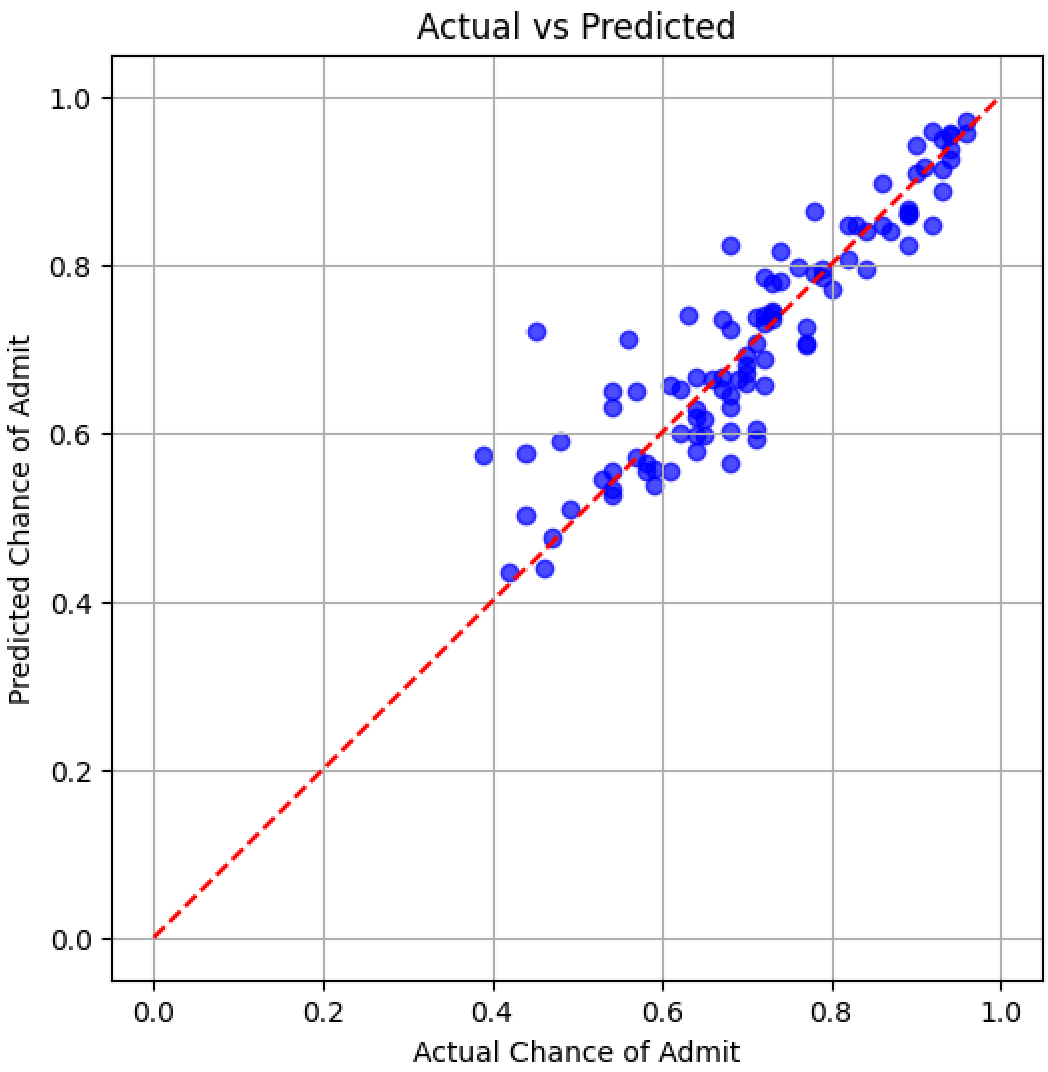
In [50]:
```
print(f"Mean Squared Error: {mse:.4f}")
print(f"R² Score: {r2:.4f}")
```

```
Mean Squared Error: 0.0037
R² Score: 0.8188
```

In [51]:
```
# Step 12: Coefficients of the model
coefficients = pd.DataFrame(model.coef_, X.columns, columns=['Coefficient'])
print(coefficients)
```

```
                   Coefficient
GRE Score             0.002434
TOEFL Score           0.002996
University Rating     0.002569
SOP                   0.001814
LOR                   0.017238
CGPA                  0.112527
Research              0.024027
```

In [34]:
```
# Step 13: Plotting Actual vs Predicted
plt.figure(figsize=(6, 6))
plt.scatter(y_test, y_pred, alpha=0.7, color="blue")
plt.xlabel("Actual Chance of Admit")
plt.ylabel("Predicted Chance of Admit")
plt.title("Actual vs Predicted")
plt.plot([0, 1], [0, 1], '--r')
plt.grid(True)
plt.show()
```



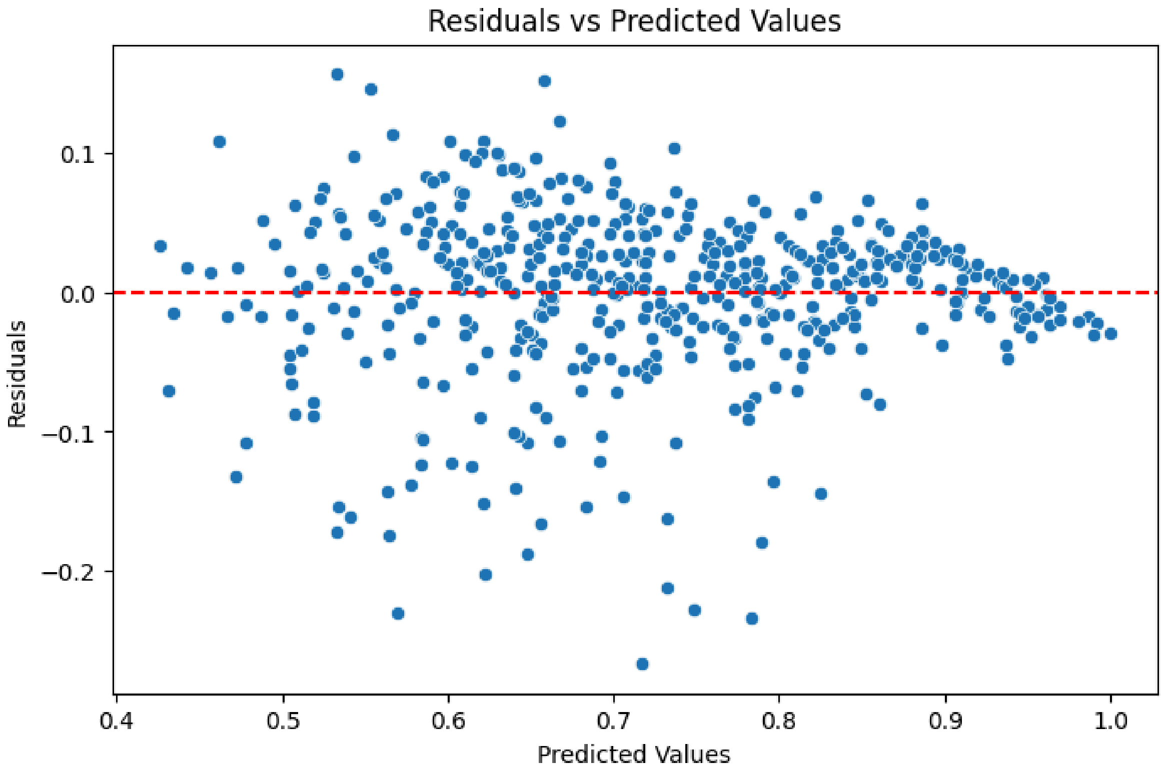
In [53]:
```
from statsmodels.stats.diagnostic import het_goldfeldquandt
import statsmodels.api as sm
```

In [54]:
```
# Add constant term for statsmodels
X_sm = sm.add_constant(X1)
```

In [55]:
```
# Fit the model
model = sm.OLS(y, X_sm).fit()
```

In [56]:
```
# Step 3: Get predicted values and residuals
predicted = model.predict(X_sm)
residuals = y - predicted
```

In [57]:
```
plt.figure(figsize=(8, 5))
sns.scatterplot(x=predicted, y=residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs Predicted Values")
plt.show()
```



In [58]:
```
# Step 4: Goldfeld-Quandt Test
gq_test = het_goldfeldquandt(residuals, X)
print(f"Goldfeld-Quandt Test F-statistic: {gq_test[0]:.4f}")
print(f"P-value: {gq_test[1]:.4f}")
```

```
Goldfeld-Quandt Test F-statistic: 0.4498
P-value: 1.0000
```

In [59]:
```
# Interpretation
if gq_test[1] > 0.05:
    print("✅ No strong evidence of heteroscedasticity (Homoscedasticity is p
else:
    print("⚠️ Evidence of heteroscedasticity found.")
```

```
✅ No strong evidence of heteroscedasticity (Homoscedasticity is present).
```

```
# 💡 Actionable Insights & Recommendations
1. 🎯 Focus on Improving CGPA
Insight: CGPA has the strongest influence.

Recommendation: Students should prioritize maintaining a high GPA
(preferably 8.5+).
```

2. 📘 Enhance GRE and TOEFL Scores
Insight: These scores moderately affect admission chances.

Recommendation: Aim for GRE > 320 and TOEFL > 105 to stay competitive.

3. 🏫 Target Higher-Ranked Universities
Insight: Applicants from better-rated universities are slightly favored.

Recommendation: If possible, enroll in better-rated institutions or leverage exchange programs to enhance profile.

4. ✍️ Write a Strong SOP
Insight: A compelling SOP has a small but noticeable effect.

Recommendation: Spend quality time tailoring your SOP to each program, highlighting your academic strengths and goals.

5. 🧑‍🔬 Engage in Research
Insight: Research experience adds a competitive edge.

Recommendation: Get involved in research projects, internships, or publish papers to demonstrate academic maturity.

6. 🤝 Get Strong Letters of Recommendation

Top Priority: CGPA, GRE, TOEFL

Medium Priority: SOP, University Rating

Bonus Factors: Research, LOR

In [ ]: