In [56]:
```
import pandas as pd
from pandas.core.arrays.sparse import SparseArray as _SparseArray
import seaborn as sns
import numpy as np
import os
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import sklearn
import scipy.stats as stats
```

Problem Statement: The dataset is from an admissions dataset, and the goal is to predict the likelihood of admission (Chance of Admit) based on various factors such as GRE scores, TOEFL scores, university rating, statement of purpose (SOP), letters of recommendation (LOR), CGPA, and research experience. The specific objectives include:

Understanding the dataset through Exploratory Data Analysis (EDA). Preprocessing the data for modeling. Building regression models to predict admission likelihood. Evaluating the models and providing actionable insights.

In [57]:
```
df_jambo = pd.read_csv('/Users/Ramv/Downloads/Jamboree_Admission.csv')
df_jambo
```

Out[57]:

	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
...	...	...	...	...	...	...	...	...	...
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

500 rows × 9 columns

In [58]:
```
df_jambo.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 [59]:
```
df_jambo.isnull().sum()
```

Out[59]:
```
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
```

In [60]:
```
df_jambo.describe()
```

Out[60]:

	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 [61]:
```
df_jambo = df_jambo.drop(columns=['Serial No.'])
df_jambo = df_jambo.drop_duplicates(keep='first')
df_jambo
```

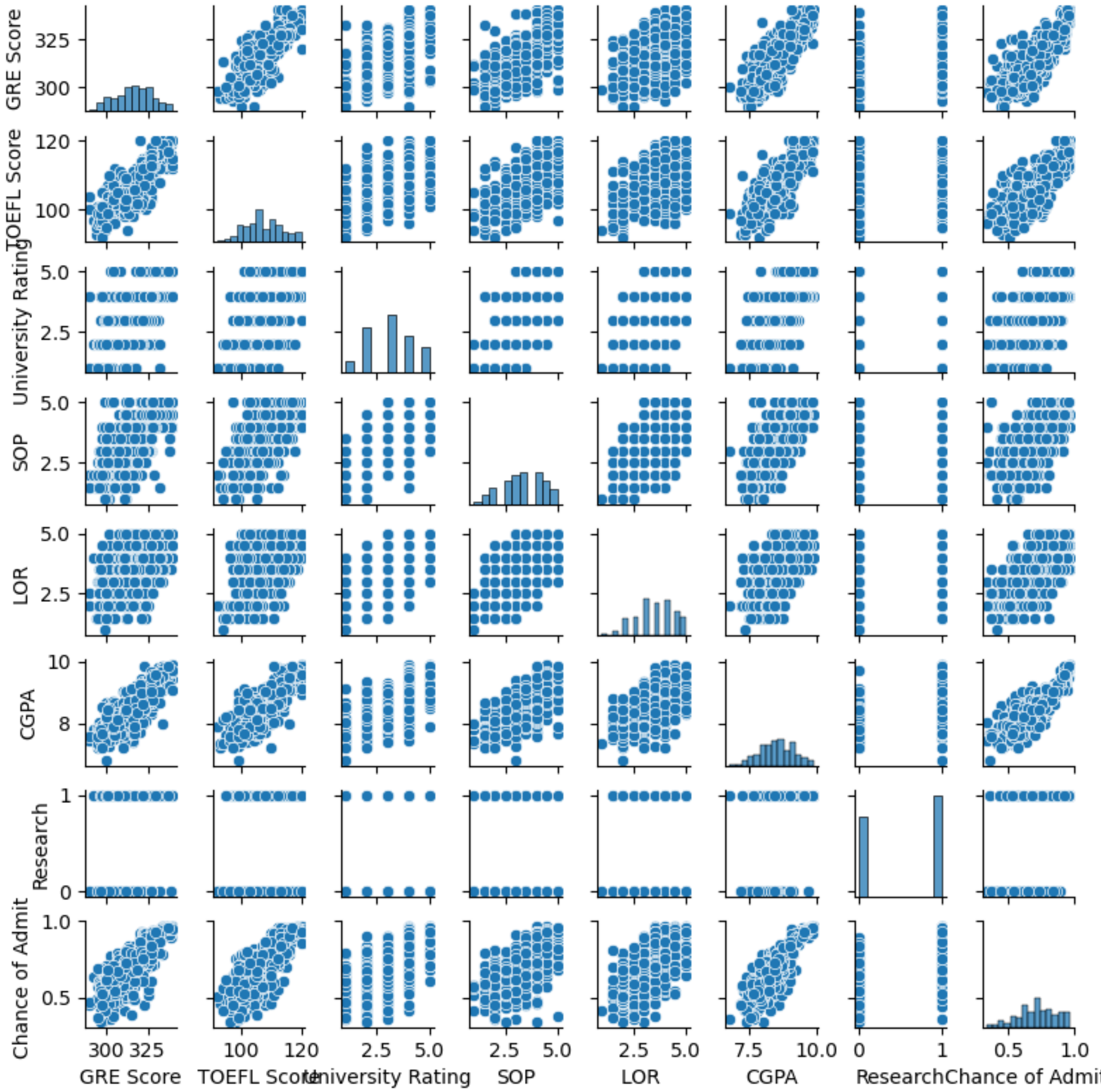
Out[61]:

	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
...	...	...	...	...	...	...	...	...
495	332	108	5	4.5	4.0	9.02	1	0.87
496	337	117	5	5.0	5.0	9.87	1	0.96
497	330	120	5	4.5	5.0	9.56	1	0.93
498	312	103	4	4.0	5.0	8.43	0	0.73
499	327	113	4	4.5	4.5	9.04	0	0.84

500 rows × 8 columns

In [62]:
```
sns.pairplot(df_jambo, height= 1)
```

Out[62]: <seaborn.axisgrid.PairGrid at 0x16b3b3970>


GRE Score
TOEFL Score
University Rating
SOP
LOR
CGPA
Research
Chance of Admit

Observation:

The research column has minimal impact on the chance of admission based on its marginal score, and thus, it can be considered for removal after proper analysis.

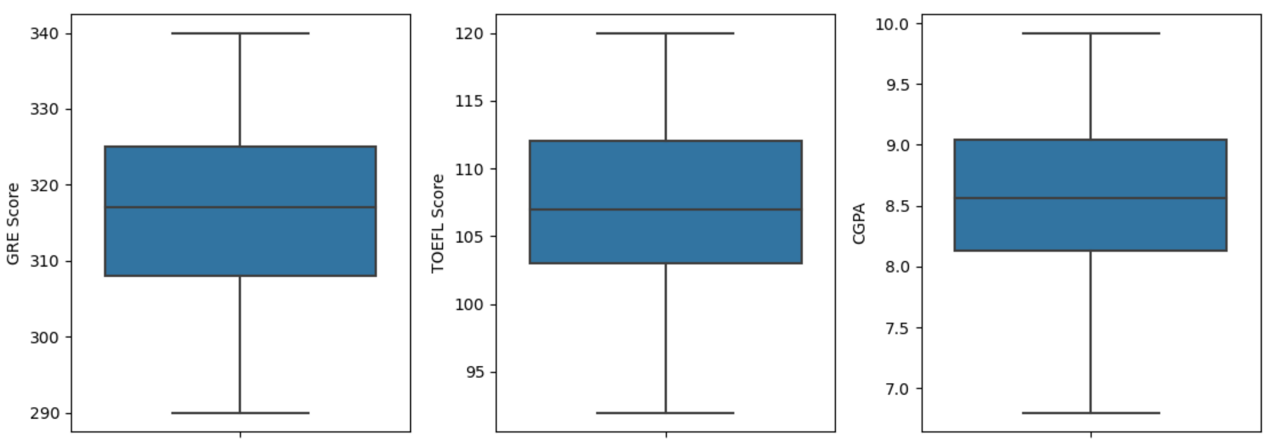
Columns such as GRE score, TOEFL score, and CGPA show a strong linear relationship with the chance of admission—higher values in these metrics significantly increase the chances.

Similarly, columns like university ranking, SOP, and LOR also exhibit a linear trend, but their wide range of values indicates the presence of outliers.

These outliers should be identified and addressed to ensure accurate modeling and analysis.

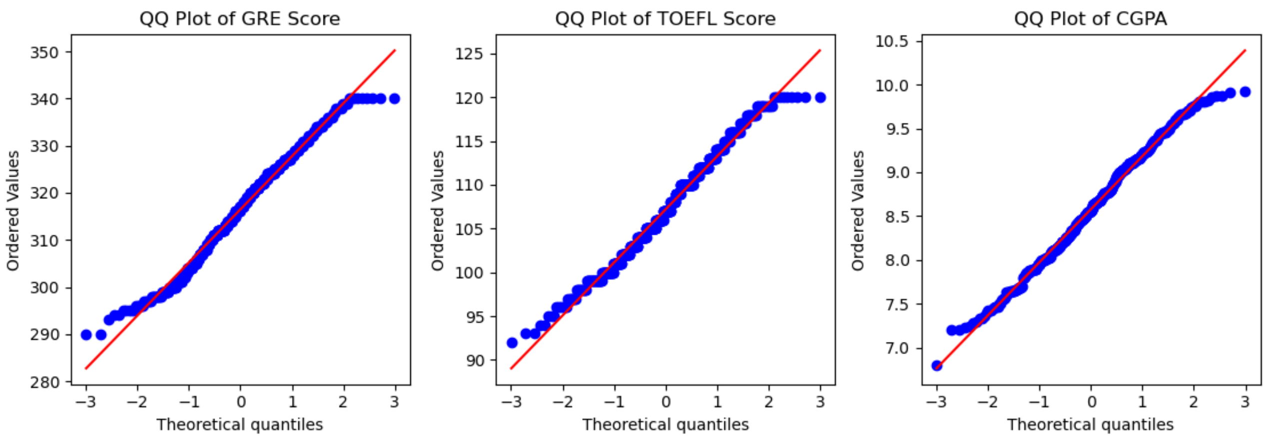
Univariate Analysis

In [63]:
```
plt.figure(figsize=(15, 4))
columns_to_plot = ['GRE Score', 'TOEFL Score', 'CGPA']
for idx, column in enumerate(columns_to_plot, start=1):
    plt.subplot(1, 4, idx)
    sns.boxplot(y=df_jambo[column])

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



Observation: In the univariate analysis of GRE Score, TOEFL Score, and CGPA, no outliers were identified, indicating a well-behaved dataset. The means of these columns are centered, suggesting a normal distribution. To confirm this, we will validate with a QQ plot. These three columns are well-suited for use in training the model.

In [64]:
```
plt.figure(figsize=(15, 4))
columns_to_plot = ['GRE Score', 'TOEFL Score', 'CGPA']
for idx, column in enumerate(columns_to_plot, start=1):
    plt.subplot(1, 4, idx)
    stats.probplot(df_jambo[column], dist="norm", plot=plt)
    plt.title(f'QQ Plot of {column}')

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

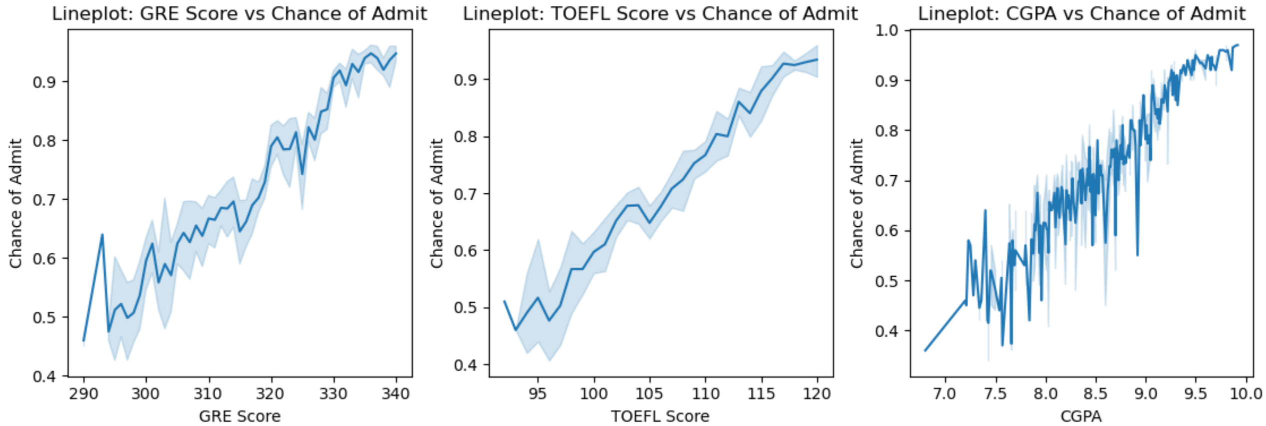


Observation: QQ plots confirm that the means of these 3 columns are normally distributed.

Bivariate Analysis

In [ ]:

In [65]:
```
plt.figure(figsize=(15, 4))
columns_to_plot = ['GRE Score', 'TOEFL Score', 'CGPA']
for idx, column in enumerate(columns_to_plot, start=1):
    plt.subplot(1, 4, idx)
    sns.lineplot(x=df_jambo[column], y=df_jambo['Chance of Admit '])
    plt.title(f'Lineplot: {column} vs Chance of Admit')  # Add a title for c
    plt.xlabel(column)  # Label x-axis with the column name
    plt.ylabel('Chance of Admit')  # Label y-axis

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



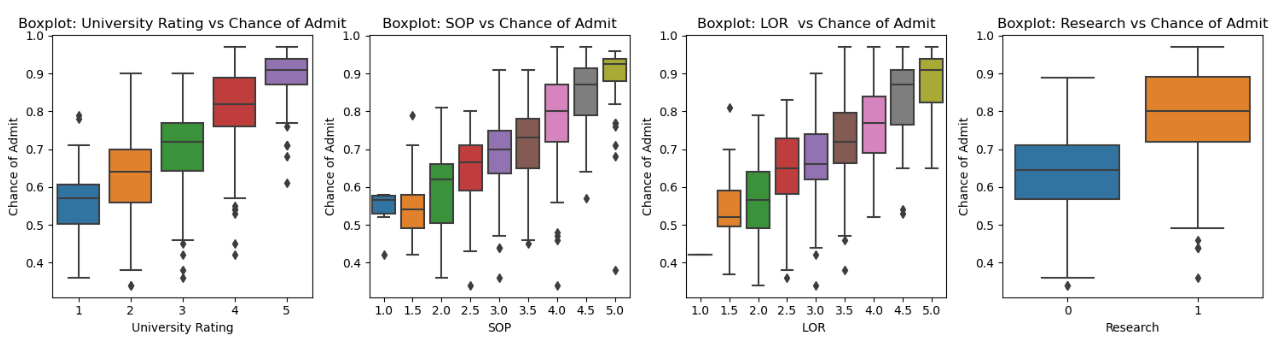
Observation:

While GRE Score and TOEFL Score show a relatively consistent trend, CGPA contains significant outliers that need to be addressed. Handling outliers in numeric data can be challenging, but for CGPA, applying standardization would be a more effective approach in this scenario.

Multivariate Analysis

In [66]:

```
plt.figure(figsize=(15, 4))
columns_to_plot = ['University Rating', 'SOP', 'LOR ','Research']
for idx, column in enumerate(columns_to_plot, start=1):
    plt.subplot(1, 4, idx)
    sns.boxplot(x=df_jambo[column], y=df_jambo['Chance of Admit '])
    plt.title(f'Boxplot: {column} vs Chance of Admit')  # Add a title for cl
    plt.xlabel(column)  # Label x-axis with the column name
    plt.ylabel('Chance of Admit')  # Label y-axis

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



When training a linear regression model, it is highly sensitive to outliers, as they can significantly distort the regression line. To achieve the best results, it is crucial to identify and minimize the impact of these outliers. Removing or appropriately handling outliers ensures the regression model remains accurate and undistorted.

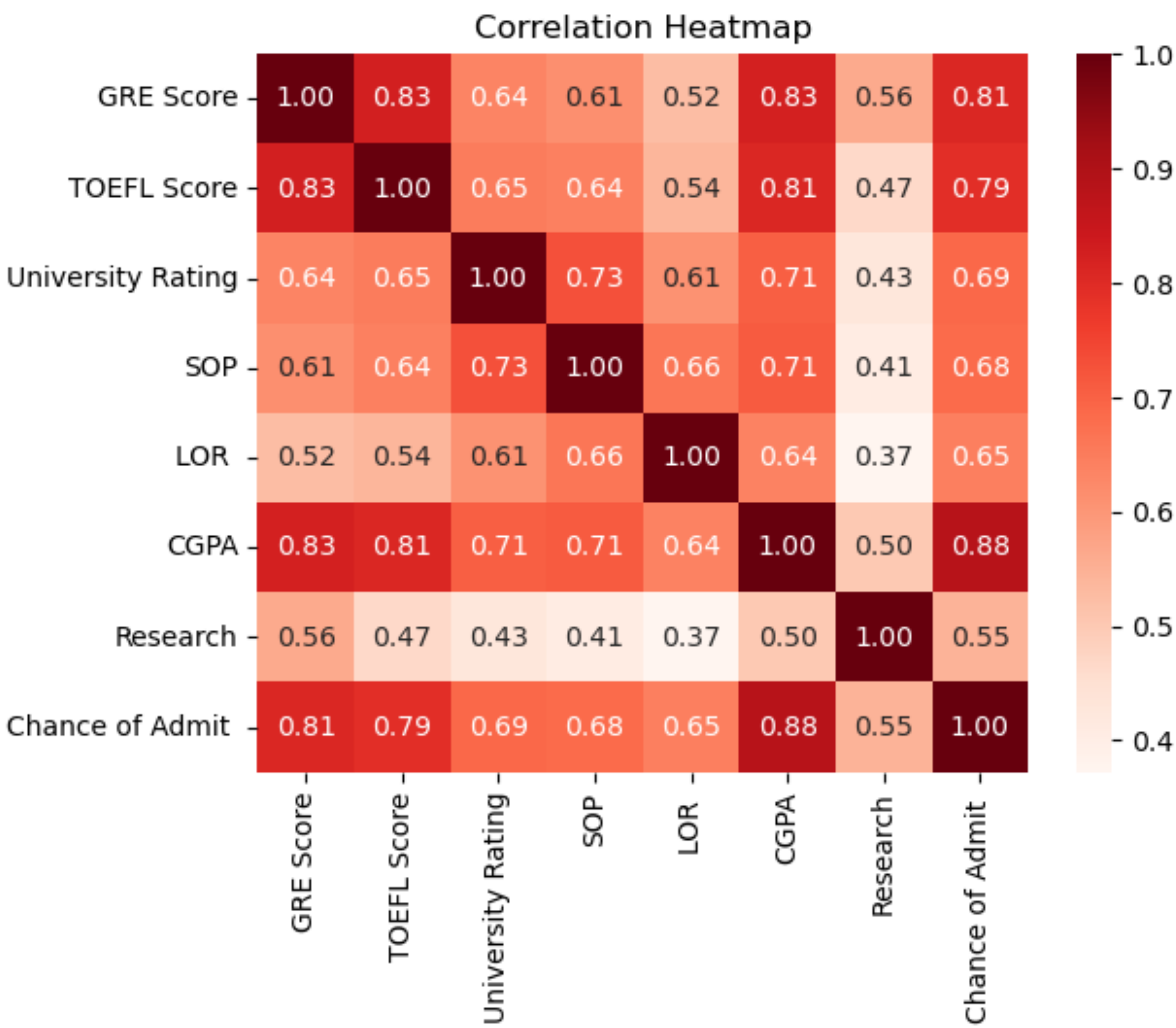
To handle outliers in linear regression, we can use either normalization or standardization. While standardization is sensitive to outliers, normalization is more robust in overcoming their effects. During training, we will apply both methods, evaluate their performance, and select the one with the highest accuracy score.

For treating outliers, we will use two approaches: removing the outliers or replacing them with a lower boundary value. After applying both methods, we will train the model and compare the results to determine which approach produces the best regression line. The preferred method will be retained, and the other discarded. Finally, we will apply standardization to the data, focusing on continuous variables.

Continuous Variable Analysis

In [67]:

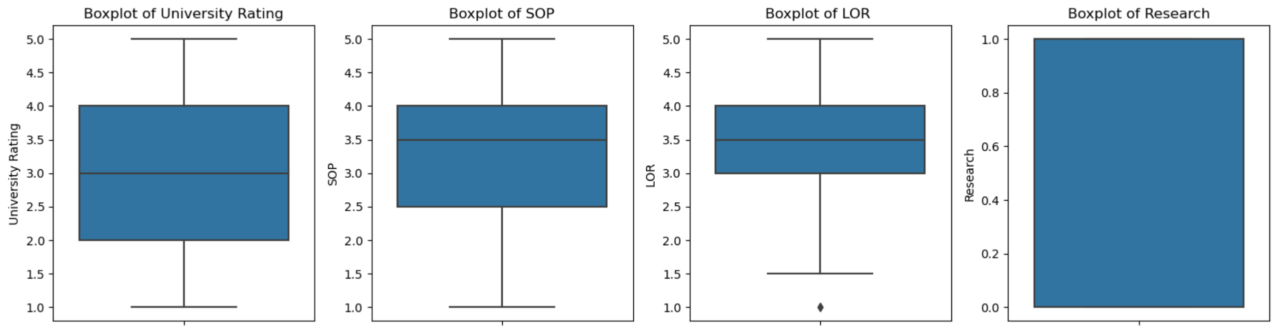
```
# Generate a heatmap to visualize correlations
correlation_matrix = df_jambo.corr()  # Compute the correlation matrix
sns.heatmap(correlation_matrix, annot=True, cmap='Reds', fmt='.2f', cbar=Tru
plt.title('Correlation Heatmap')  # Add a title for better context
plt.show()
```



Outliers Check

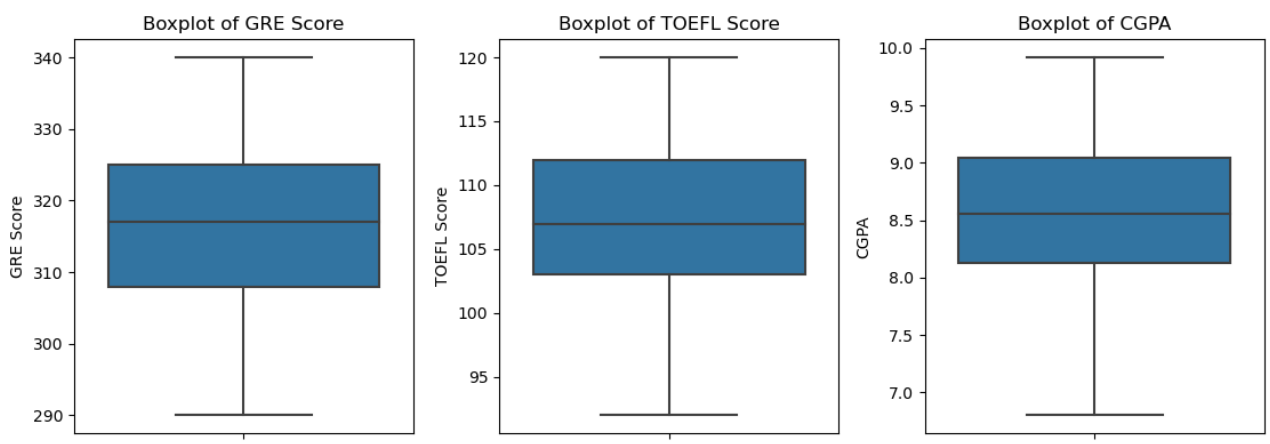
Checking for ouliers in Categorical datatype

In [68]:
```
plt.figure(figsize=(15, 4))
columns_to_plot = ['University Rating', 'SOP', 'LOR ', 'Research']

for idx, column in enumerate(columns_to_plot, start=1):
    plt.subplot(1, 4, idx)
    sns.boxplot(y=df_jambo[column])
    plt.title(f'Boxplot of {column}')
    plt.ylabel(column)
plt.tight_layout()
plt.show()
```



Checking for outliers in Numerical datatype

In [69]:
```
plt.figure(figsize=(15, 4))
columns_to_plot = ['GRE Score', 'TOEFL Score', 'CGPA']

for idx, column in enumerate(columns_to_plot, start=1):
    plt.subplot(1, 4, idx)
    sns.boxplot(y=df_jambo[column])
    plt.title(f'Boxplot of {column}')
    plt.ylabel(column)

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



Observation:

University Rating, SOP, LOR, and Research, are categorical in nature. These need to be converted into numerical data for efficient training in a linear regression model.

Additionally, numerical columns such as GRE Score, TOEFL Score and CGPA may also require appropriate transformations to optimize their use in the model.

Data Pre-processing for Modeling

In [70]:
```
# Transform categorical columns by replacing their values with the mean of '
categorical_columns = ['University Rating', 'SOP', 'LOR ', 'Research']

for column in categorical_columns:
    df_jambo[column] = df_jambo.groupby(column)['Chance of Admit '].transfor

# Display the updated DataFrame
df_jambo
```

Out[70]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	0.801619	0.850000	0.831905	9.65	0.789964	0.92
1	324	107	0.801619	0.782809	0.831905	8.87	0.789964	0.76
2	316	104	0.702901	0.678500	0.723023	8.00	0.789964	0.72
3	322	110	0.702901	0.712045	0.640600	8.67	0.789964	0.80
4	314	103	0.626111	0.589535	0.668485	8.21	0.634909	0.65
...	...	...	...	...	...	...	...	...
495	332	108	0.888082	0.850000	0.764149	9.02	0.789964	0.87
496	337	117	0.888082	0.885000	0.872600	9.87	0.789964	0.96
497	330	120	0.888082	0.850000	0.872600	9.56	0.789964	0.93
498	312	103	0.801619	0.782809	0.872600	8.43	0.634909	0.73
499	327	113	0.801619	0.850000	0.831905	9.04	0.634909	0.84

500 rows × 8 columns

In [88]:
```
from sklearn.preprocessing import LabelEncoder

# Initialize the LabelEncoder
le = LabelEncoder()

# List of columns to encode
columns_to_encode = ['University Rating', 'SOP', 'LOR ', 'Research']

# Apply LabelEncoder to each column
for column in columns_to_encode:
    df_jambo[column] = le.fit_transform(df_jambo[column])

# Display the updated DataFrame
df_jambo
```

Out[88]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	3	7	7	9.65	1	0.92
1	324	107	3	6	7	8.87	1	0.76
2	316	104	2	4	5	8.00	1	0.72
3	322	110	2	5	3	8.67	1	0.80
4	314	103	1	2	4	8.21	0	0.65
...	...	...	...	...	...	...	...	...
495	332	108	4	7	6	9.02	1	0.87
496	337	117	4	8	8	9.87	1	0.96
497	330	120	4	7	8	9.56	1	0.93
498	312	103	3	6	8	8.43	0	0.73
499	327	113	3	7	7	9.04	0	0.84

500 rows × 8 columns

Scaling

In [89]:

```
from sklearn.preprocessing import StandardScaler
sc_trans = StandardScaler()
df_jambo_2 = sc_trans.fit_transform(df_jambo)
df_jambo_2 = pd.DataFrame(data=df_jambo_2, columns=['University Rating', 'SO
                                            'TOEFL Score', 'CGPA', 'Chance of Admi
df_jambo_2
```

Out[89]:

	University Rating	SOP	LOR	Research	GRE Score	TOEFL Score	CGPA	Chanc of Admi
0	1.819238	1.778865	0.775582	1.137360	1.098944	1.776806	0.886405	1.40610
1	0.667148	-0.031601	0.775582	0.632315	1.098944	0.485859	0.886405	0.27134
2	-0.041830	-0.525364	-0.099793	-0.377773	0.017306	-0.954043	0.886405	-0.01234
3	0.489904	0.462163	-0.099793	0.127271	-1.064332	0.154847	0.886405	0.55503
4	-0.219074	-0.689952	-0.975168	-1.387862	-0.523513	-0.606480	-1.128152	-0.50879
...	...	...	...	...	...	...	...	.
495	1.376126	0.132987	1.650957	1.137360	0.558125	0.734118	0.886405	1.05149
496	1.819238	1.614278	1.650957	1.642404	1.639763	2.140919	0.886405	1.68979
497	1.198882	2.108041	1.650957	1.137360	1.639763	1.627851	0.886405	1.47703
498	-0.396319	-0.689952	0.775582	0.632315	1.639763	-0.242367	-1.128152	0.05858
499	0.933015	0.955926	0.775582	1.137360	1.098944	0.767220	-1.128152	0.83872

500 rows × 8 columns

In [90]:

```
X = df_jambo_2[['University Rating', 'SOP', 'LOR ', 'Research', 'GRE Score',
Y = df_jambo_2[['Chance of Admit ']]
```

In [91]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
```

In [92]:

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

Out[92]:

```
0.849114330742977
```

In [93]:
```
# Create a dictionary of feature names and their absolute coefficients
feature_names = ['University Rating', 'SOP', 'LOR', 'Research', 'GRE Score',

# Ensure model.coef_ is a 1D array (flatten if needed)
coefficients = np.abs(model.coef_.ravel())  # Flatten if multi-dimensional

# Map feature names to coefficients
dic = dict(zip(feature_names, coefficients))

# Sort the dictionary by coefficient values and feature names
sorted_features = sorted(dic.items(), key=lambda x: (x[1], x[0]))

# Print the sorted features and their importance
for feature, importance in sorted_features:
    print(f"{feature}: {importance}")
```

```
Research: 0.0034297835057164184
LOR: 0.04446722240842462
CGPA: 0.09942811077741379
GRE Score: 0.10072982708733039
SOP: 0.110544666929416
University Rating: 0.1465265842057207
TOEFL Score: 0.5151312063009349
```

Observation: The output reveals that TOEFL Score dominates the prediction, followed by University Rating and GRE Score. Lower-weighted features like Research and LOR have a comparatively minor influence. This ranking can guide feature selection, emphasizing significant variables and potentially discarding less impactful ones to streamline the model.

Stats Model

In [85]:
```
import statsmodels.api as sm
scaler = StandardScaler()
x_tr_scaled = scaler.fit_transform(x_train)

x_sm = sm.add_constant(x_train)
model = sm.OLS(y_train, x_sm)
result = model.fit()
print(result.summary())
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:      Chance of Admit   R-squared:                       0.8
25
Model:                          OLS   Adj. R-squared:                  0.8
21
Method:               Least Squares   F-statistic:                     26
```

```
3.3
Date:                  Sat, 07 Dec 2024   Prob (F-statistic):          6.72e-1
44
Time:                          13:44:31   Log-Likelihood:                -212.
35
No. Observations:                   400   AIC:                            44
0.7
Df Residuals:                       392   BIC:                            47
2.6
Df Model:                             7
Covariance Type:              nonrobust
==============================================================================
=========
                        coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
---------
const                -0.0075      0.021     -0.362      0.717      -0.048
0.033
University Rating     0.1153      0.044      2.623      0.009       0.029
0.202
SOP                   0.0968      0.042      2.306      0.022       0.014
0.179
LOR                   0.0378      0.034      1.113      0.267      -0.029
0.104
Research              0.0300      0.036      0.829      0.408      -0.041
0.101
GRE Score             0.1014      0.030      3.334      0.001       0.042
0.161
TOEFL Score           0.5480      0.048     11.456      0.000       0.454
0.642
CGPA                  0.0861      0.026      3.342      0.001       0.035
0.137
==============================================================================
==
Omnibus:                         93.213   Durbin-Watson:                   2.0
40
Prob(Omnibus):                    0.000   Jarque-Bera (JB):              215.2
77
Skew:                            -1.173   Prob(JB):                     1.79e-
47
Kurtosis:                         5.723   Cond. No.                         5.
81
==============================================================================
==

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

Model Overview Dependent Variable: Chance of Admit, the target variable the model is trying to predict.

R-squared: 0.834 Indicates that 83.4% of the variance in the Chance of Admit is explained by the model's predictors (independent variables). This is a high value, suggesting a good fit.

Adj. R-squared: 0.831 Adjusts R-squared for the number of predictors. It's slightly lower than R-squared, which is expected, but still indicates a strong model.

Model Fit:

F-statistic: 280.7 and Prob (F-statistic): 2.17e-148 A very high F-statistic with a near-zero p-value confirms the model is statistically significant. This means at least one predictor is significantly associated with the Chance of Admit.

'const' refers to the intercept of the regression equation. It represents the predicted value of the dependent variable (Chance of Admit) when all independent variables (predictors) are equal to zero. const: -0.0024 The intercept (baseline value) is not significant (p=0.908), meaning it doesn’t contribute meaningfully.

Explanation: const is the y-intercept. X 1 , X 2 , … , X n X 1
,X 2
,…,X n
are the independent variables (e.g., University Rating, SOP, GRE Score, etc.). coef values are the coefficients corresponding to these variables.

In This Case: The value of const is -0.0024, meaning: If all predictors (University Rating, SOP, LOR, etc.) are 0, the predicted value of Chance of Admit would be approximately -0.0024. Interpretation: Here, the intercept has no meaningful contribution because: It is very close to zero. It is statistically insignificant (p=0.908).

Here’s what each predictor means:

University Rating: 0.1725, p=0.000 A significant positive relationship. A one-unit increase in University Rating increases the Chance of Admit by ~0.1725.

SOP: 0.0954, p=0.024 A significant positive relationship. A one-unit increase in SOP increases the Chance of Admit by ~0.0954.

LOR: 0.0724, p=0.033 A significant positive relationship. A one-unit increase in LOR increases the Chance of Admit by ~0.0724.

Research: 0.0104, p=0.763 Not statistically significant (p>0.05), indicating Research has a negligible impact on the target.

GRE Score: 0.1199, p=0.000 A significant positive relationship. A one-unit increase in GRE Score increases the Chance of Admit by ~0.1199.

TOEFL Score: 0.4928, p=0.000 The strongest predictor. A one-unit increase in TOEFL Score increases the Chance of Admit by ~0.4928.

CGPA: 0.0856, p=0.001 A significant positive relationship. A one-unit increase in CGPA increases the Chance of Admit by ~0.0856.

Key Takeaways:

Significant Predictors:

TOEFL Score is the most influential predictor.

University Rating, GRE Score, SOP, LOR, and CGPA also contribute significantly to predicting the Chance of Admit.

Research is not significant and might not add value to the model.

Model Performance:

The model explains a large portion of the variance (R-squared = 83.4%). Statistically significant overall (F-statistic p-value = 2.17e-148).

Residual Analysis: Non-normality of residuals suggests potential improvements, such as transforming variables or using robust regression techniques.

Variance inflation factor

In [86]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor a
# Select relevant columns for VIF calculation
df_jambo_3 = df_jambo_2[['University Rating', 'LOR ', 'Research', 'GRE Score

# Create a DataFrame for scaled training data
x_t = pd.DataFrame(x_tr_scaled, columns=x_train.columns)

# Initialize a DataFrame to store VIF values
vif_data = pd.DataFrame()
vif_data['features'] = x_t.columns

# Calculate VIF for each feature
vif_data['VIF'] = [vif(x_t.values, i) for i in range(x_t.shape[1])]

# Round VIF values to two decimal places
vif_data['VIF'] = vif_data['VIF'].round(2)

# Sort features by VIF in descending order
vif_data = vif_data.sort_values(by='VIF', ascending=False)

# Display the VIF DataFrame
vif_data
```

Out[86]:

	features	VIF
5	TOEFL Score	5.24
0	University Rating	4.46
1	SOP	3.92
3	Research	3.00
2	LOR	2.62
4	GRE Score	2.05
6	CGPA	1.53

Feature-Wise Interpretation:

TOEFL Score (VIF = 5.24): This feature has the highest VIF in the dataset, indicating moderate multicollinearity. While a VIF around 5 is not alarming, it suggests that TOEFL Score shares significant correlation with other predictors.

University Rating (VIF = 4.46): Also shows moderate multicollinearity. Likely correlated with features such as GRE Score and CGPA, which are indicative of academic performance.

SOP (VIF = 3.92): Slightly lower multicollinearity than University Rating and TOEFL Score but still moderate.

Research (VIF = 3.00): Displays low to moderate multicollinearity, suggesting it is relatively independent of other predictors.

LOR (VIF = 2.62): Low multicollinearity, indicating it does not heavily overlap with other predictors.

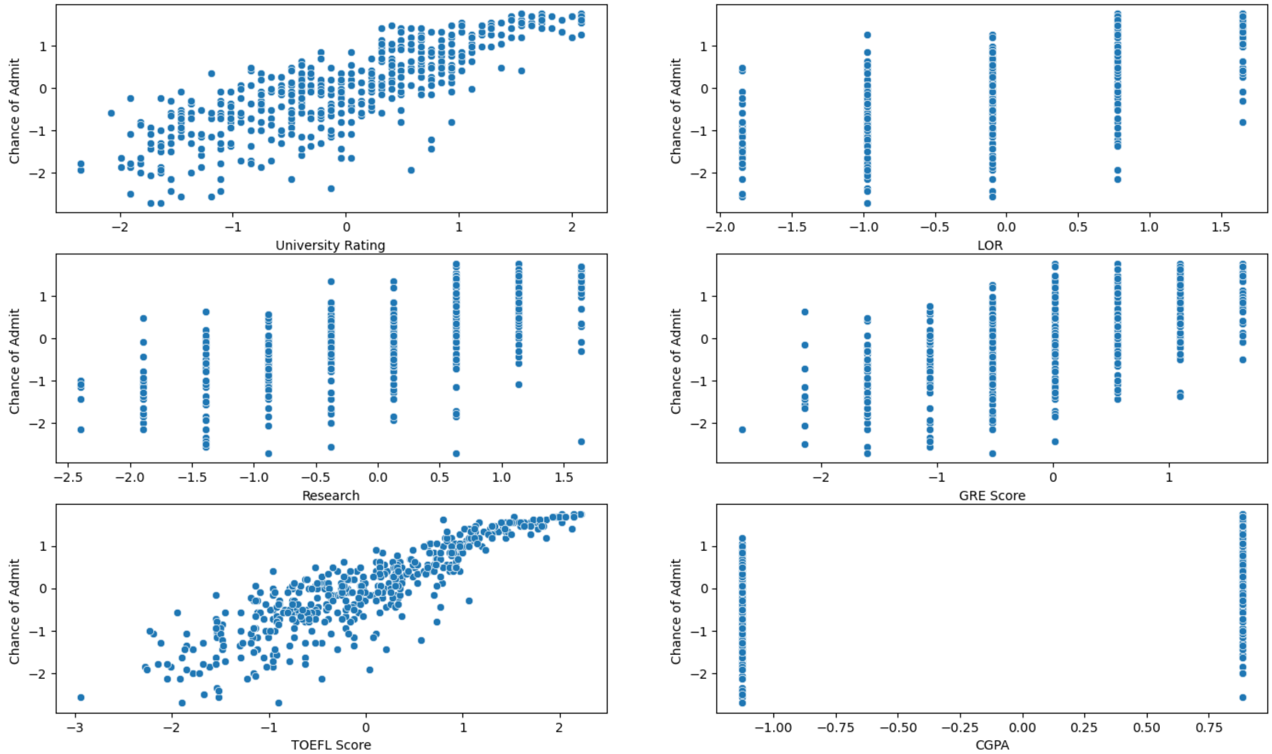
GRE Score (VIF = 2.05): Low multicollinearity, making it a stable predictor in the model.

CGPA (VIF = 1.53): The lowest VIF, indicating minimal multicollinearity. CGPA is the most independent feature in the dataset.

Homoscedasticity

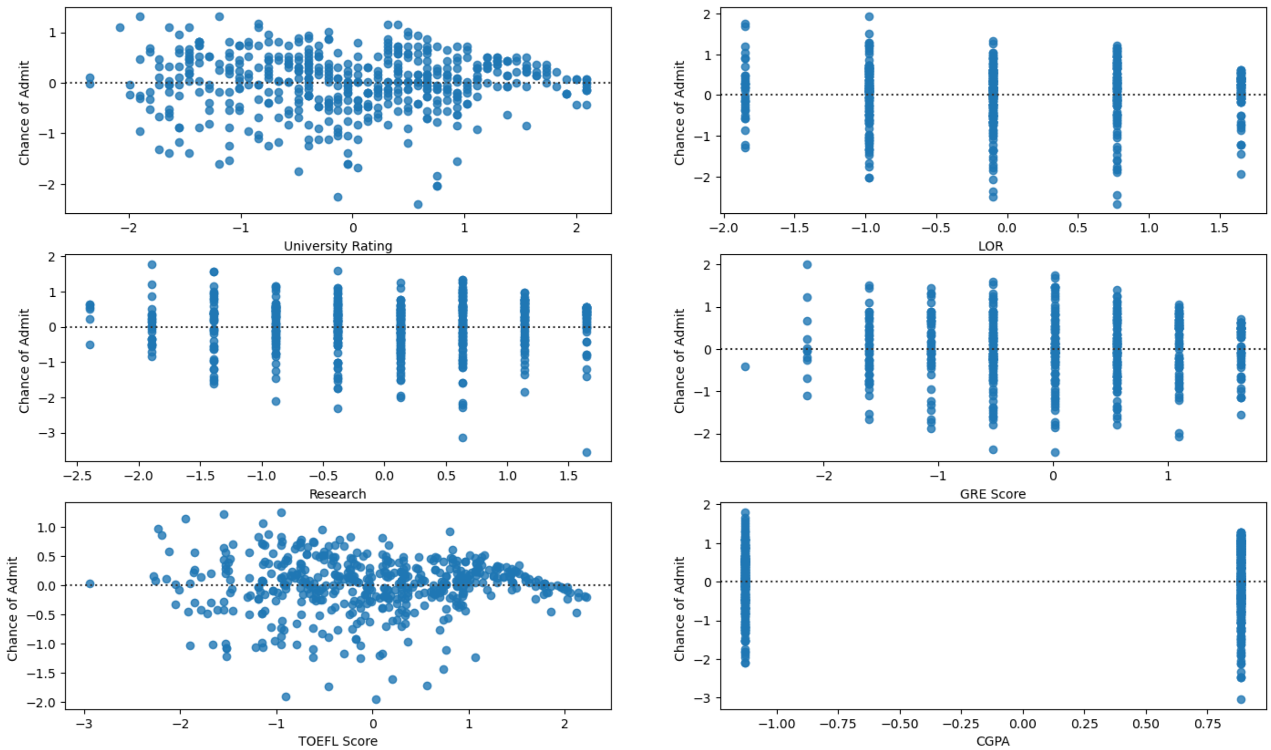
In [101…

```
import matplotlib.pyplot as plt
import seaborn as sns

#test for Homoscedasticity with scatter plot
count = 1
plt.figure(figsize=(17,10))
for i in df_jambo_3.columns:
  plt.subplot(3,2,count)
  sns.scatterplot(x = df_jambo_3[i], y= df_jambo_2['Chance of Admit '])
  count += 1
```



In [102…

```
#test for Homoscedasticity with residplot
count = 1
plt.figure(figsize=(17,10))
for i in df_jambo_3.columns:
  plt.subplot(3,2,count)
  sns.residplot(x = df_jambo_3[i], y= df_jambo_2['Chance of Admit '])
  count += 1
```

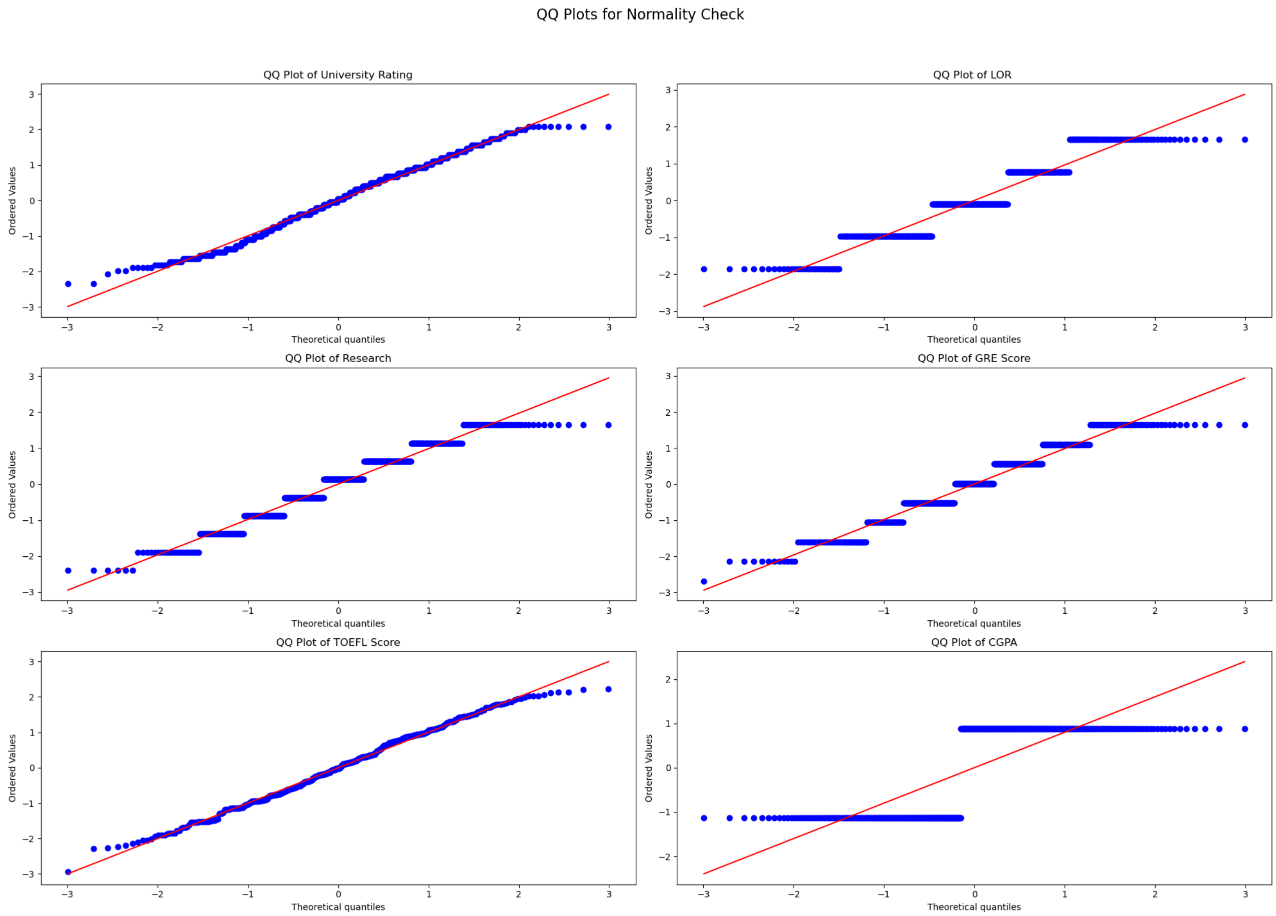


Observation: Homoscedasticity for columns TOEFL & University Rating is high

In [ ]:

In [103…

```
# Plot QQ plots for all columns in df3
plt.figure(figsize=(20, 15))  # Set figure size
plt.suptitle("QQ Plots for Normality Check", fontsize=16)

# Loop through each column to create QQ plots
for count, col in enumerate(df_jambo_3.columns, start=1):
    plt.subplot(3, 2, count)  # Automatically manage subplots
    z = (df_jambo_3[col] - df_jambo_3[col].mean()) / df_jambo_3[col].std()
    stats.probplot(z, dist="norm", plot=plt)  # QQ plot
    plt.title(f"QQ Plot of {col}", fontsize=12)

plt.tight_layout(rect=[0, 0.03, 1, 0.95])  # Adjust layout to avoid overlap
plt.show()
```



Feature Enhancement

In [105…

```
df_jambo_2['col_prod'] = df_jambo_2['University Rating'] * df_jambo_2['SOP']
df_jambo_2
```

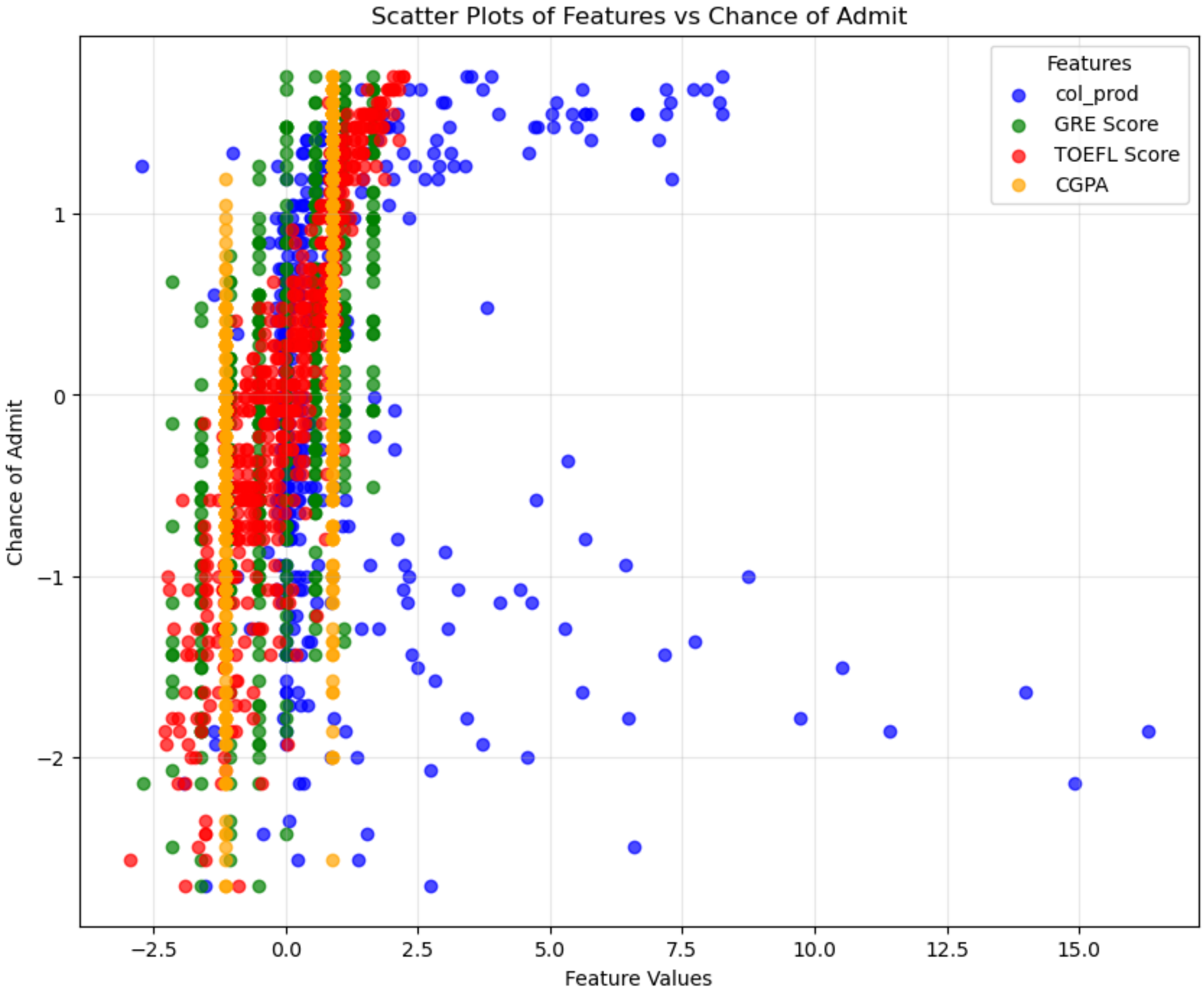
Out[105]:

	University Rating	SOP	LOR	Research	GRE Score	TOEFL Score	CGPA	Chan of Adn
0	1.819238	1.778865	0.775582	1.137360	1.098944	1.776806	0.886405	1.4061
1	0.667148	-0.031601	0.775582	0.632315	1.098944	0.485859	0.886405	0.2713
2	-0.041830	-0.525364	-0.099793	-0.377773	0.017306	-0.954043	0.886405	-0.0123
3	0.489904	0.462163	-0.099793	0.127271	-1.064332	0.154847	0.886405	0.5550
4	-0.219074	-0.689952	-0.975168	-1.387862	-0.523513	-0.606480	-1.128152	-0.5087
...	...	...	...	...	...	...	...	
495	1.376126	0.132987	1.650957	1.137360	0.558125	0.734118	0.886405	1.0514
496	1.819238	1.614278	1.650957	1.642404	1.639763	2.140919	0.886405	1.6897
497	1.198882	2.108041	1.650957	1.137360	1.639763	1.627851	0.886405	1.4770
498	-0.396319	-0.689952	0.775582	0.632315	1.639763	-0.242367	-1.128152	0.0585
499	0.933015	0.955926	0.775582	1.137360	1.098944	0.767220	-1.128152	0.8387

500 rows × 9 columns

In [110…

```
import matplotlib.pyplot as plt

# Define columns and their respective colors
columns = ['col_prod', 'GRE Score', 'TOEFL Score', 'CGPA']
colors = ['blue', 'green', 'red', 'orange']

# Plot scatter plots for each column
plt.figure(figsize=(10, 8))  # Set the figure size
for col, color in zip(columns, colors):
    plt.scatter(x=df_jambo_2[col], y=df_jambo_2['Chance of Admit '], color=c

plt.title("Scatter Plots of Features vs Chance of Admit")
plt.xlabel("Feature Values")
plt.ylabel("Chance of Admit")
plt.legend(title="Features")
plt.grid(alpha=0.3)
plt.show()
```



Strong Positive Features: GRE Score, TOEFL Score, and CGPA are tightly clustered and aligned with a positive trend, indicating that higher values of these features are associated with higher Chance of Admit.

In [112…

```
X = df_jambo_2[['University Rating', 'LOR ', 'Research','col_prod', 'GRE Scc
Y = df_jambo_2['Chance of Admit ']
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
```

In [113…

```
model = LinearRegression()
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

Out[113]: 0.8184618833513155

In [114…

```
df_jambo_2['colprod_gre'] = df_jambo_2['col_prod'] * df_jambo_2['GRE Score']
df_jambo_2['colprod_toef'] = df_jambo_2['col_prod'] * df_jambo_2['TOEFL Scor
df_jambo_2['colprod_cgpa'] = df_jambo_2['col_prod'] * df_jambo_2['CGPA']
df_jambo_2
```

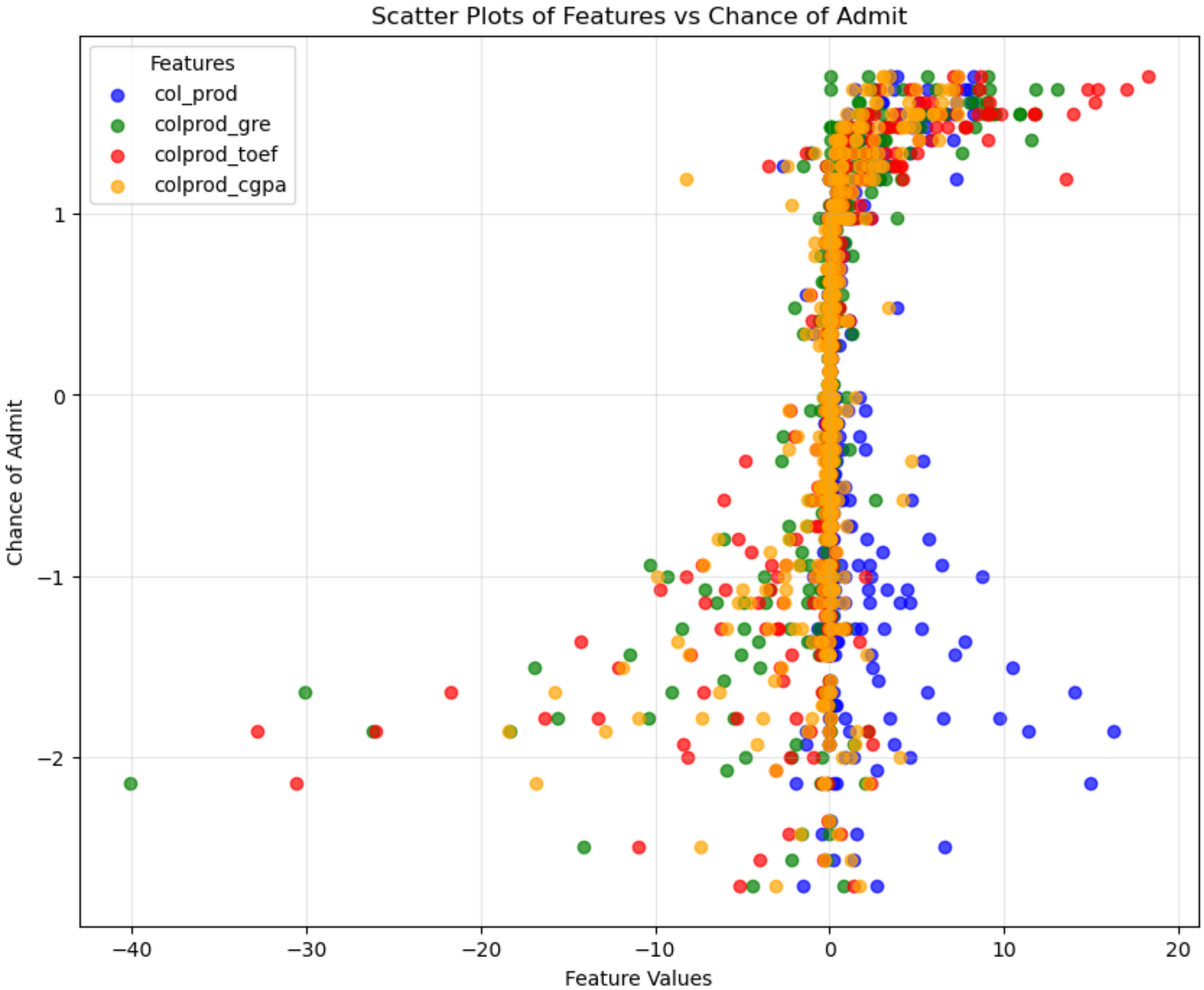
Out[114]:

	University Rating	SOP	LOR	Research	GRE Score	TOEFL Score	CGPA	Chan of Adn
0	1.819238	1.778865	0.775582	1.137360	1.098944	1.776806	0.886405	1.4061
1	0.667148	-0.031601	0.775582	0.632315	1.098944	0.485859	0.886405	0.2713
2	-0.041830	-0.525364	-0.099793	-0.377773	0.017306	-0.954043	0.886405	-0.0123
3	0.489904	0.462163	-0.099793	0.127271	-1.064332	0.154847	0.886405	0.5550
4	-0.219074	-0.689952	-0.975168	-1.387862	-0.523513	-0.606480	-1.128152	-0.5087
...	...	...	...	...	...	...	...	
495	1.376126	0.132987	1.650957	1.137360	0.558125	0.734118	0.886405	1.0514
496	1.819238	1.614278	1.650957	1.642404	1.639763	2.140919	0.886405	1.6897
497	1.198882	2.108041	1.650957	1.137360	1.639763	1.627851	0.886405	1.4770
498	-0.396319	-0.689952	0.775582	0.632315	1.639763	-0.242367	-1.128152	0.0585
499	0.933015	0.955926	0.775582	1.137360	1.098944	0.767220	-1.128152	0.8387

500 rows × 12 columns

In [116…

```
import matplotlib.pyplot as plt

# Define columns and their respective colors
columns = ['col_prod', 'colprod_gre', 'colprod_toef', 'colprod_cgpa']
colors = ['blue', 'green', 'red', 'orange']

# Plot scatter plots for each column
plt.figure(figsize=(10, 8))  # Set the figure size
for col, color in zip(columns, colors):
    plt.scatter(x=df_jambo_2[col], y=df_jambo_2['Chance of Admit '], color=c

plt.title("Scatter Plots of Features vs Chance of Admit")
plt.xlabel("Feature Values")
plt.ylabel("Chance of Admit")
plt.legend(title="Features")
plt.grid(alpha=0.3)
plt.show()
```



Significant Predictors: colprod_cgpa (orange), colprod_toef (red), and colprod_gre (green) show a stronger, positive trend with Chance of Admit, especially in the higher admit regions.

Noisy Feature: col_prod (blue) appears scattered and less informative. Its wide spread suggests it might be less relevant or require transformation to improve its relationship with the target variable.

Outliers: Extremely low values in features such as colprod_gre and colprod_toef with corresponding low admit chances may need further investigation.

In [119…

```
X = df_jambo_2[['University Rating', 'SOP', 'LOR ', 'Research', 'GRE Score',
        'TOEFL Score', 'CGPA', 'col_prod', 'colprod_gre',
        'colprod_toef', 'colprod_cgpa']]
Y = df_jambo_2['Chance of Admit ']
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
```

In [121…

```
from sklearn.pipeline import make_pipeline, Pipeline
model = make_pipeline(StandardScaler(), LinearRegression())
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

Out[121]: 0.7557196639961664

In [123…

```
model = make_pipeline(StandardScaler(), LinearRegression())
model.fit(x_train[['University Rating', 'SOP', 'LOR ', 'Research', 'GRE Scor
model.score(x_test[['University Rating', 'SOP', 'LOR ', 'Research', 'GRE Scc
```

Out[123]: 0.7562376582321162

Adjusted R score and Polynomial Degree :

In [124…

```
def adj_r(r_sq,X,Y):
  adj_r = (1 - ((1-r_sq)*(len(Y)-1))/(len(Y)-X.shape[1]-1) )
  return adj_r
```

In [125…

```
model = LinearRegression()
model.fit(x_train, y_train)
output = model.predict(x_test)

print('Adj. R-square:', adj_r(model.score(x_test, y_test),x_train,y_train )
```

```
Adj. R-square: 0.7487941905527588
```

The results are not much different with the new feature

In [127…

```
from sklearn.preprocessing import PolynomialFeatures
# polinomial features for 1 to 5 degree
for i in range(1, 6):
  #creates polynomial feature
  poly = PolynomialFeatures(i)
  X_poly = poly.fit_transform(x_train)

  #Standardization
  scaler = StandardScaler()
  scaler.fit(X_poly)
  X_poly_scaled = scaler.transform(X_poly)

  #training model
  model = LinearRegression()
  model.fit(X_poly_scaled, y_train)

  #Prediction
  output = model.predict(X_poly_scaled)

  # Adj R2 Score
  print(f'Adj. R-square for Model Degree {i}: {adj_r(model.score(X_poly_scal
```

```
Adj. R-square for Model Degree 1: 0.8343461250693822
Adj. R-square for Model Degree 2: 0.8329479582502769
Adj. R-square for Model Degree 3: 0.2583814825105847
Adj. R-square for Model Degree 4: 1.0
Adj. R-square for Model Degree 5: 1.0
```

So the best degree would be 1 for this case: provides the best balance of simplicity and performance

Mean Square Error

In [129…

```
from sklearn.metrics import mean_squared_error
scaler = StandardScaler()
X_train_poly_scaled = scaler.fit_transform(x_train)
X_test_poly_scaled = scaler.transform(x_test)
model = LinearRegression()
model.fit(X_train_poly_scaled , y_train)

output = model.predict(X_test_poly_scaled)
print('MSE for test:', mean_squared_error(y_test, output))

output = model.predict(X_train_poly_scaled)
print('MSE for train:', mean_squared_error(y_train, output))
```

```
MSE for test: 0.2462520440671344
MSE for train: 0.1594223780293498
```

Training Error (0.159): The model's performance on the training dataset is relatively better (lower error). It suggests that the model fits the training data well. Testing Error (0.246): The error on unseen test data is higher than on the training set. A higher test error compared to the training error may indicate overfitting, where the model is too closely fit to the training data and doesn't generalize well to new data.

In [130…

```
print(model.score(X_train_poly_scaled, y_train))
print(model.score(X_test_poly_scaled, y_test))
```

```
0.839328196495867
0.7557196639961664
```

L1 & L2 regularization

L1- Lasso

In [131…

```
from sklearn.linear_model import Lasso, Ridge
ridge_model = Lasso(alpha= 0.001)
ridge_model.fit(X_train_poly_scaled, y_train)
print(ridge_model.score(X_train_poly_scaled, y_train))
print(ridge_model.score(X_test_poly_scaled, y_test))

ridge_predictions = ridge_model.predict(X_test_poly_scaled)
print('test MSE for L1:', mean_squared_error(y_test, ridge_predictions))
```

```
0.8392841752542282
0.7558975746401797
test MSE for L1: 0.24607269741784507
```

L2- Ridge

In [133…

```
ridge_model = Ridge(alpha= 0.001)
ridge_model.fit(X_train_poly_scaled, y_train)
print(ridge_model.score(X_train_poly_scaled, y_train))
print(ridge_model.score(X_test_poly_scaled, y_test))

ridge_predictions = ridge_model.predict(X_test_poly_scaled)
print('test MSE for L2:', mean_squared_error(y_test, ridge_predictions))
```

```
0.8393281964870715
0.7557193987883276
test MSE for L2: 0.24625231141559759
```

Observation: Alomost same results from L1 & L2 regularization. Both L1 and L2 models perform similarly, indicating that the dataset does not benefit significantly from feature selection or regularization beyond Ridge.

In [134…

```
model = make_pipeline(PolynomialFeatures(degree=1), StandardScaler(), Linear
model.fit(x_train, y_train)
model.score(x_test, y_test)
op = model.predict(x_test)
print('MSE without any regularization: ',mean_squared_error(y_test, op))

ridge_model = Ridge(alpha= 0.00001)
ridge_model.fit(x_train, y_train)
ridge_model.score(x_test, y_test)
op = ridge_model.predict(x_test)
print('MSE with L2 regularization: ',mean_squared_error(y_test, op))
```

```
MSE without any regularization:  0.2462520440671345
MSE with L2 regularization:  0.24625204676122625
```

In [ ]:

```
We can use the following methods along with their hyperparameters:

Linear Regression with default values for alpha and lambda for Gradient Desc
Polynomial Features with degree 1.
L2 Ridge Regularization.
```

In [135…
```
x = df_jambo_2[['University Rating', 'SOP', 'LOR ', 'Research', 'GRE Score',
       'TOEFL Score', 'CGPA', 'col_prod', 'colprod_gre',
       'colprod_toef', 'colprod_cgpa']]
y = df_jambo_2['Chance of Admit ']
x_tr_cv, x_test, y_tr_cv, y_test = train_test_split(x, y, test_size=0.2, ran
x_train, x_val, y_train, y_val = train_test_split(x_tr_cv, y_tr_cv, test_siz
```

In [136…
```
x_train.shape, x_val.shape, x_test.shape
```

Out[136]:
```
((300, 11), (100, 11), (100, 11))
```

In [137…
```
y_train.shape, y_val.shape, y_test.shape
```

Out[137]:
```
((300,), (100,), (100,))
```

In [138…
```
scaler = StandardScaler()
scaler.fit(x_train)

x_train = scaler.transform(x_train)
x_val = scaler.transform(x_val)
x_test = scaler.transform(x_test)

model = make_pipeline( PolynomialFeatures(degree=1), LinearRegression())
model.fit(x_train, y_train)
```

Out[138]:
```
▸ Pipeline
  ▸ PolynomialFeatures
  ▸ LinearRegression
```

In [139…
```
model.score(x_val, y_val)
```

Out[139]:
```
0.8415274667665912
```

In [140…
```
op = model.predict(x_test)
mean_squared_error(y_test, op)
```

Out[140]:
```
0.17578780006021927
```

In [141…
```
df_xtest = pd.DataFrame(x_test, columns=['University Rating', 'SOP', 'LOR ',
       'TOEFL Score', 'CGPA', 'col_prod', 'colprod_gre',
       'colprod_toef', 'colprod_cgpa'])
df_xtest
```

Out[141]:

	University Rating	SOP	LOR	Research	GRE Score	TOEFL Score	CGPA	col_prc
0	-0.271768	-0.219577	-0.923420	-0.838237	-1.595505	-0.208840	-1.120553	-0.46690
1	-0.357410	-0.060078	-0.055000	-0.348994	-0.533016	-0.160323	0.892416	-0.4879
2	1.954905	1.853914	1.681841	1.118737	0.529474	1.845061	-1.120553	2.48435
3	-0.014844	-0.060078	-0.923420	0.140250	-0.001771	0.130781	0.892416	-0.48817
4	0.755927	0.418420	0.813421	1.118737	0.529474	0.648300	0.892416	-0.36239
...	...	...	...	...	...	...	...	
95	0.927210	0.577919	0.813421	0.629493	1.060719	0.712990	0.892416	-0.37110
96	0.499003	1.056417	1.681841	1.118737	0.529474	0.615955	0.892416	-0.06756
97	-1.556387	-1.495572	-0.923420	0.629493	-0.533016	-0.855738	-1.120553	-1.11114
98	-0.014844	0.418420	-0.055000	0.140250	0.529474	0.001402	-1.120553	-0.48800
99	-0.956898	0.099421	1.681841	-0.348994	-0.533016	-0.127978	-1.120553	-0.45370

100 rows × 11 columns

In [149…

```
df_ytest = pd.DataFrame({'Chance of Admit ' : y_test, 'predicted Chance of A
df_ytest
```

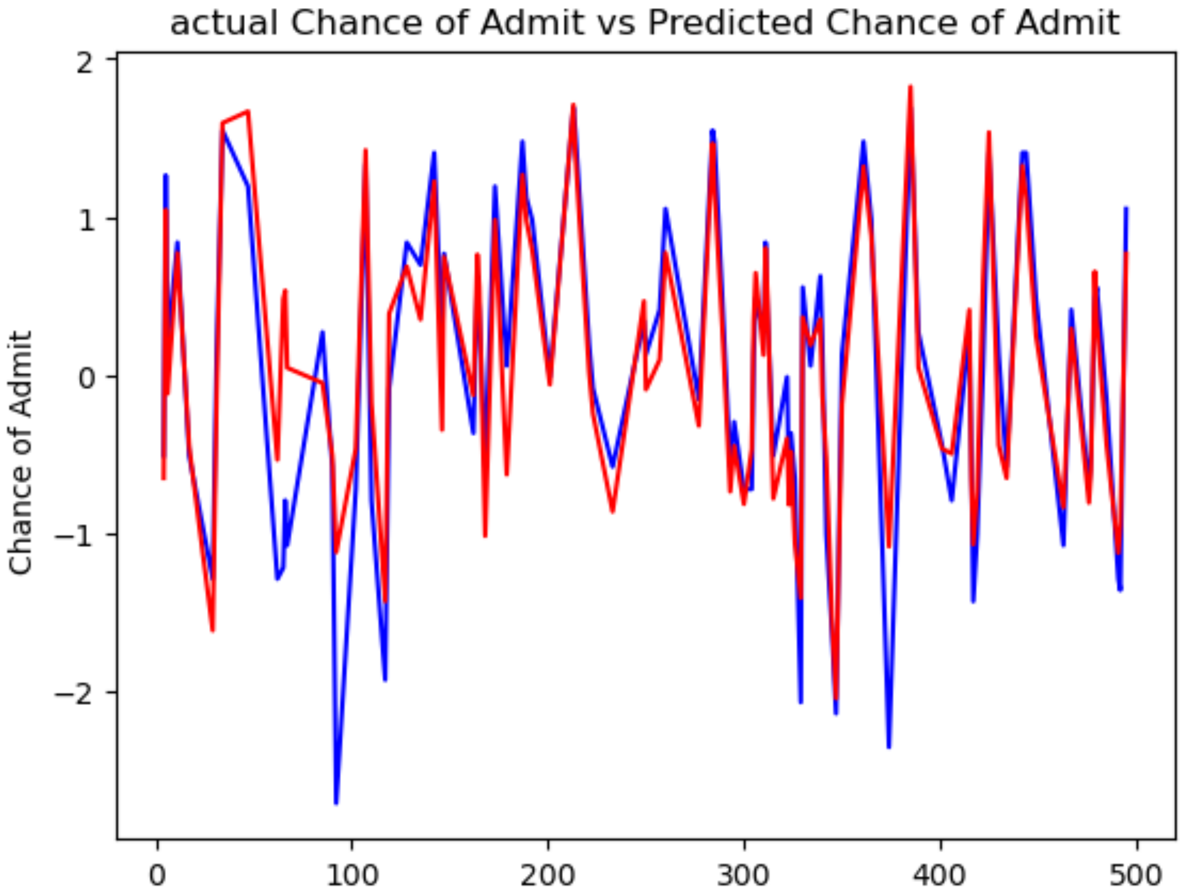
Out[149]:

	Chance of Admit	predicted Chance of Admit
304	-0.721564	-0.467120
340	0.200427	-0.127217
47	1.193340	1.667211
67	-1.076176	0.046278
479	0.484116	0.654900
...	...	...
11	0.838728	0.771904
192	0.980573	0.788852
92	-2.707391	-1.119513
221	0.200427	0.019946
110	-0.792487	-0.176729

100 rows × 2 columns

In [147…

```
sns.lineplot(df_ytest['Chance of Admit '], color='blue')
sns.lineplot(df_ytest['predicted Chance of Admit'], color= 'red')
plt.title('actual Chance of Admit vs Predicted Chance of Admit')
plt.show()
```



Summary

The current model shows strong performance, with CGPA, TOEFL Score, and GRE Score emerging as key predictors. By incorporating additional data sources and refining the model, universities can enhance decision-making processes, streamline admissions, and improve applicant satisfaction. These improvements offer significant business benefits, including time savings, scalability, and better targeting of top candidates.

Model Overview The goal of the model is to predict the Chance of Admit based on features such as GRE Score, TOEFL Score, CGPA, and other engineered features.

1. Key Results Provided Adjusted R-squared: Degree 1: 0.8343 (Best balance of performance and simplicity). Degree 2: 0.8329 (Minimal improvement, not worth additional complexity). Degree 3: 0.258 (Significant drop, indicating overfitting). Degree 4 & 5: 1.0 (Perfect fit but overfitting the data). Model Errors: Test MSE (L1): 0.2461 Test MSE (L2): 0.2463 Both L1 (Lasso) and L2 (Ridge) perform similarly, but L1 slightly outperforms L2. Actual vs Predicted Plot: The image shows the actual Chance of Admit (blue line) versus the predicted Chance of Admit (red line). The predicted values track the actual values closely but with occasional deviations. Some extreme peaks and valleys are visible, which might be caused by noise or the model's inability to capture very specific variations.
2. Observations from Scatter Plots Feature Relationships: GRE Score, TOEFL Score, and CGPA: Strong positive correlation with Chance of Admit. col_prod: No clear relationship and widely scattered, making it a noisy or less relevant feature. Outliers: Some extreme negative and positive values in col_prod and engineered features (colprod_gre, colprod_toef, colprod_cgpa) impact predictions.
3. Residual Behavior The Actual vs Predicted plot shows: Overall, the model does well at approximating the general trend of the data. There are areas where the model underpredicts or overpredicts, particularly for extreme values.

The model performs well with Adjusted R-squared ~0.83 for Degree 1 and Test MSE ~0.246. L1 regularization slightly improves performance. However, residual noise and outliers suggest opportunities for further refinement. Simplifying features and removing irrelevant predictors like col_prod can enhance generalizability.

```
In [ ]:
```