1. Introduction

What is Jamboree?

Jamboree has been instrumental in helping thousands of students achieve their dreams of studying at top colleges abroad. They specialize in test preparation for exams like GMAT, GRE, and SAT, using unique problem-solving methods that ensure maximum scores with minimal effort. Recently, Jamboree launched a feature on their website where students can check their probability of getting into Ivy League colleges. This feature estimates the chances of graduate admission from an Indian perspective.

Issue at Hand

Jamboree aims to understand the key factors that influence graduate admissions and how these factors interrelate. They also want to build a predictive model to estimate an applicant's likelihood of admission based on the available features.

Objective

- Analyze the provided dataset to derive valuable insights into the factors affecting graduate admissions.
- Build a predictive model to estimate an applicant's chance of admission given their profile.

Features of the Dataset:

Feature	Description
Serial No.	Unique row ID
GRE Scores	GRE score out of 340
TOEFL Scores	TOEFL score out of 120
University Rating	Rating of the university (out of 5)
SOP	Strength of the Statement of Purpose (out of 5)
LOR	Strength of the Letters of Recommendation (out of 5)
CGPA	Undergraduate GPA (out of 10)
Research	Research experience (0: No, 1: Yes)
Chance of Admit	Probability of admission (ranging from 0 to 1)

2. Exploratory Data Analysis :

In [127...

```
# Importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from scipy import stats # For Hypothesis Testing
import plotly.express as px # For Interactive Plots
import plotly.graph_objects as go # For Interactive Plots
import plotly.subplots as sp
from datetime import datetime as dt # For datetime manipulation
```

Dataset Link - https://drive.google.com/file/d/1s2qm9nn8H_sdfUxJ7Ya-HPiKlZbaycAL/view?usp=sharing

In [128...

```
# Importing Data Set
file_path = r'C:\Users\Bijayalaxmi Sahoo\Desktop\jamboree_admission.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(file_path)
```

In [129...

```
df
```

Out[129]:

	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 [130...

```
# Shape of the Data Set
df.shape
```

Out[130]:

```
(500, 9)
```

In [131...

```
# Columns of the Data Set
df.columns
```

Out[131]:

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

In [132…
```
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
```

Insight:

- The dataset appears to be clean with no missing values (**non-null count for each column is 500**).
- This dataset is suitable for building predictive models, such as linear regression, to predict the **'Chance of Admit'** based on other features like GRE score, TOEFL score, university rating, etc.

3. Statistical Summary

In [133…
```
df.describe()
```

Out[133]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Resea
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000
mean	250.500000	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560
std	144.481833	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000
25%	125.750000	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000
50%	250.500000	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000
75%	375.250000	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000
max	500.000000	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000

4. Duplicate Detection

In [134…
```
df.duplicated().value_counts()
```

Out[134]:
```
False    500
dtype: int64
```

Insights

- There are NO Duplicate entries in the dataset.

5.Unique Values

In [135…

```
# Check the number of unique values for each column

unique_counts = df.nunique()
print(unique_counts)
```

```
Serial No.           500
GRE Score             49
TOEFL Score           29
University Rating      5
SOP                    9
LOR                    9
CGPA                 184
Research               2
Chance of Admit       61
dtype: int64
```

6. Basic Data Cleaning and Exploration

Handling Missing Values in the Data Set

In [136…

```
# Identify missing values
missing_values = df.isnull().sum()
print(missing_values)
```

```
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
```

Insights

- There are no missing values in any of the columns in the dataset.

7.Dropping the irrelevant column 'Serial No.'

In [137…

```
df = df.drop(columns=['Serial No.'])
```

In [138…

```
# Check the shape of the dataset
print(f"Shape of the dataset: {df.shape}")
```

```
Shape of the dataset: (500, 8)
```

In [139…

```python
print("Data types of each column:\n", df.dtypes)
```

```
Data types of each column:
 GRE Score              int64
TOEFL Score             int64
University Rating       int64
SOP                   float64
LOR                   float64
CGPA                  float64
Research                int64
Chance of Admit       float64
dtype: object
```

In [140…

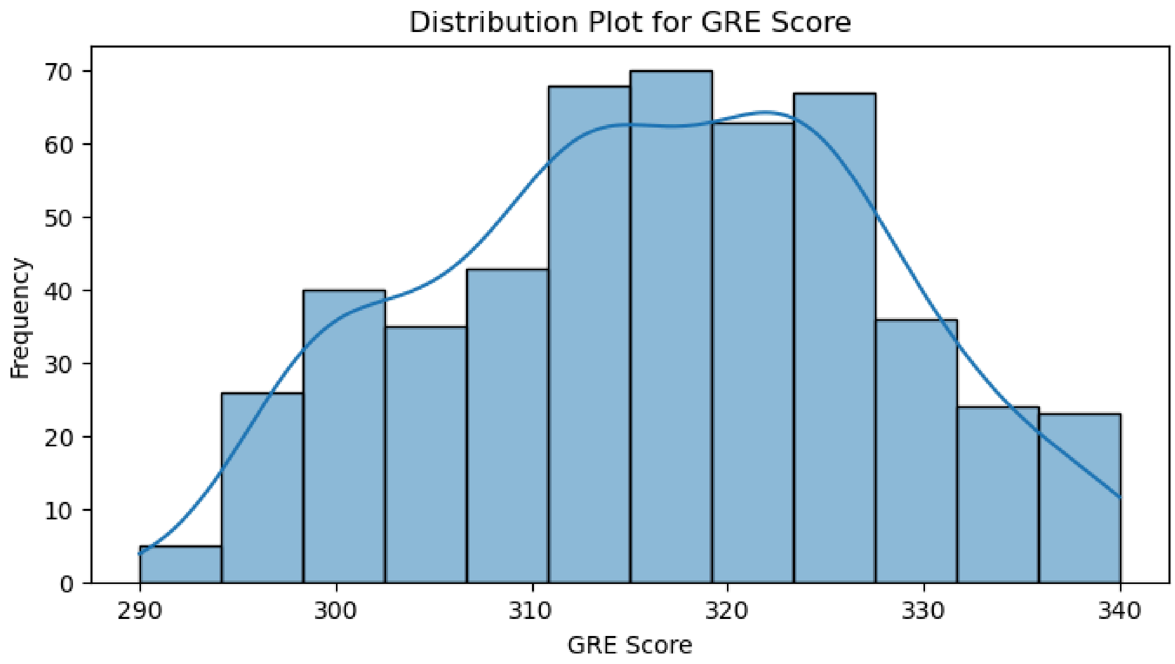
```python
# The statistical summary of the entire dataset

print("Statistical summary:\n", df.describe())
```

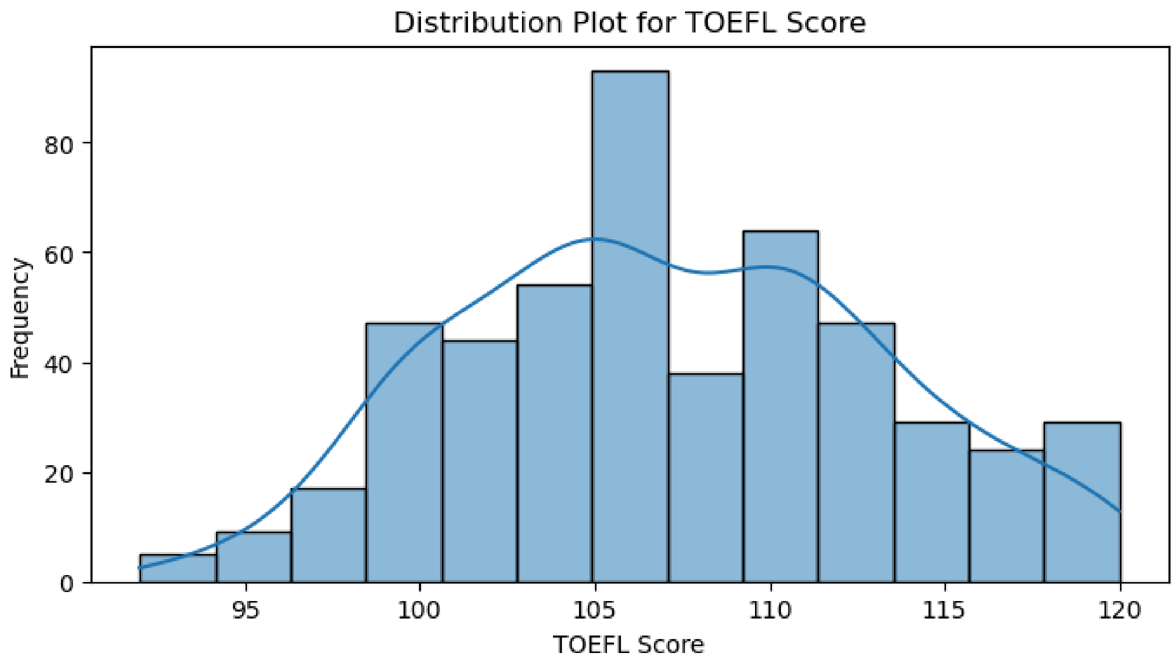
```
Statistical summary:
          GRE Score  TOEFL Score  University Rating         SOP        LOR   \
count  500.000000   500.000000           500.000000  500.000000  500.00000
mean   316.472000   107.192000             3.114000    3.374000    3.48400
std     11.295148     6.081868             1.143512    0.991004    0.92545
min    290.000000    92.000000             1.000000    1.000000    1.00000
25%    308.000000   103.000000             2.000000    2.500000    3.00000
50%    317.000000   107.000000             3.000000    3.500000    3.50000
75%    325.000000   112.000000             4.000000    4.000000    4.00000
max    340.000000   120.000000             5.000000    5.000000    5.00000

             CGPA    Research  Chance of Admit
count  500.000000  500.000000         500.00000
mean     8.576440    0.560000           0.72174
std      0.604813    0.496884           0.14114
min      6.800000    0.000000           0.34000
25%      8.127500    0.000000           0.63000
50%      8.560000    1.000000           0.72000
75%      9.040000    1.000000           0.82000
max      9.920000    1.000000           0.97000
```

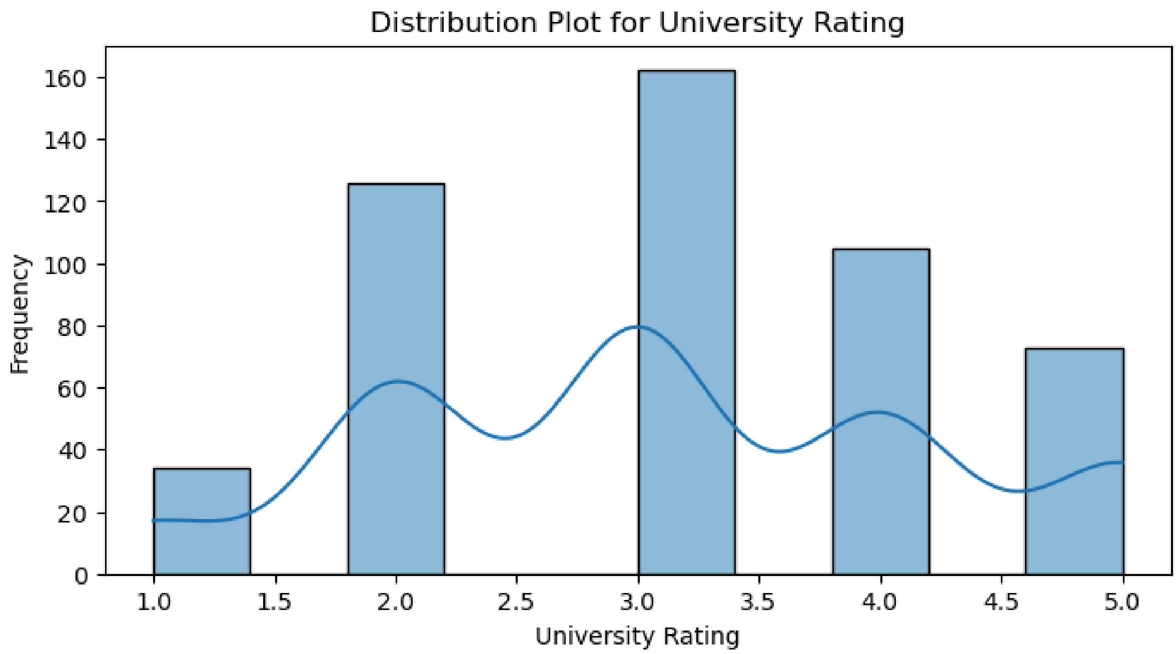
8. Univariate Analysis

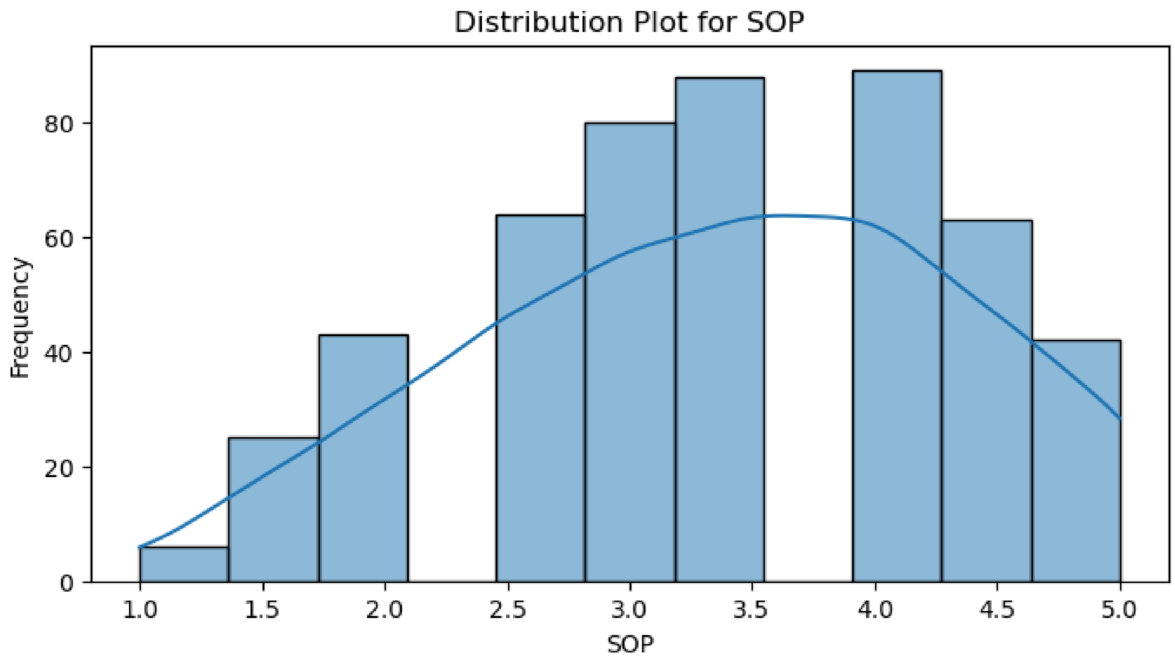
In [141…

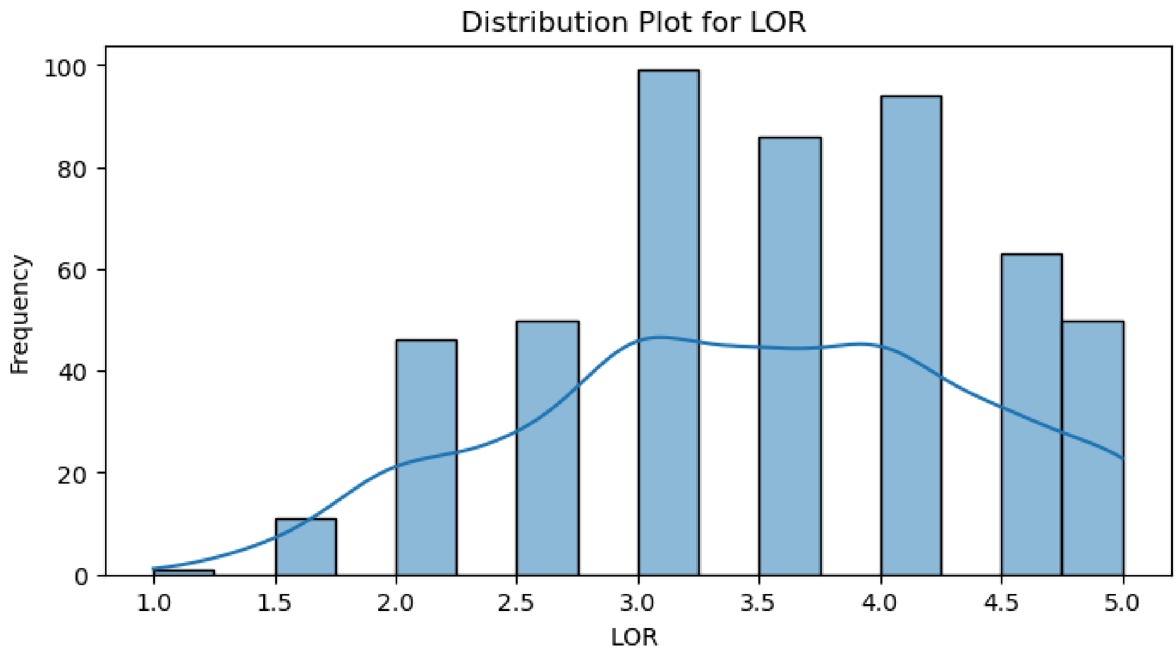
```python
# Continuous variables

continuous_columns = ['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR
for col in continuous_columns:
    plt.figure(figsize=(8, 4))
    sns.histplot(df[col], kde=True)
    plt.title(f'Distribution Plot for {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()
```

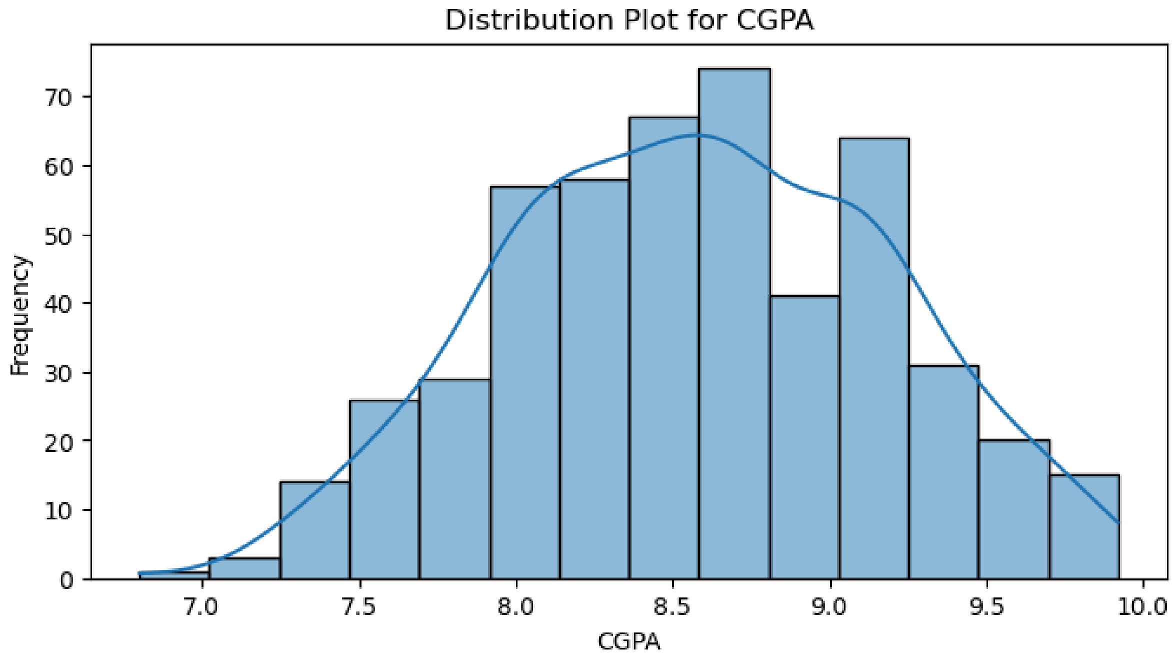

Distribution Plot for GRE Score
Frequency
0
10
20
30
40
50
60
70
290
300
310
320
330
340
GRE Score

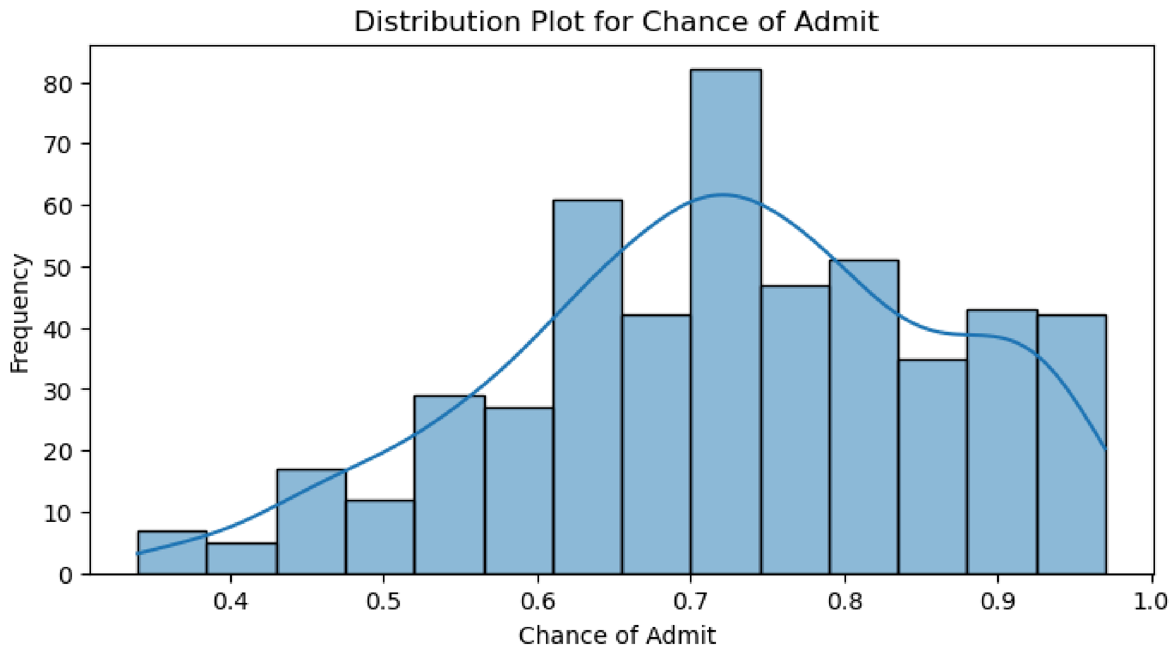

Distribution Plot for TOEFL Score
Frequency
0
20
40
60
80
95
100
105
110
115
120
TOEFL Score


Distribution Plot for University Rating
Frequency
0
20
40
60
80
100
120
140
160
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
University Rating


Distribution Plot for SOP
Frequency
SOP
0
20
40
60
80
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

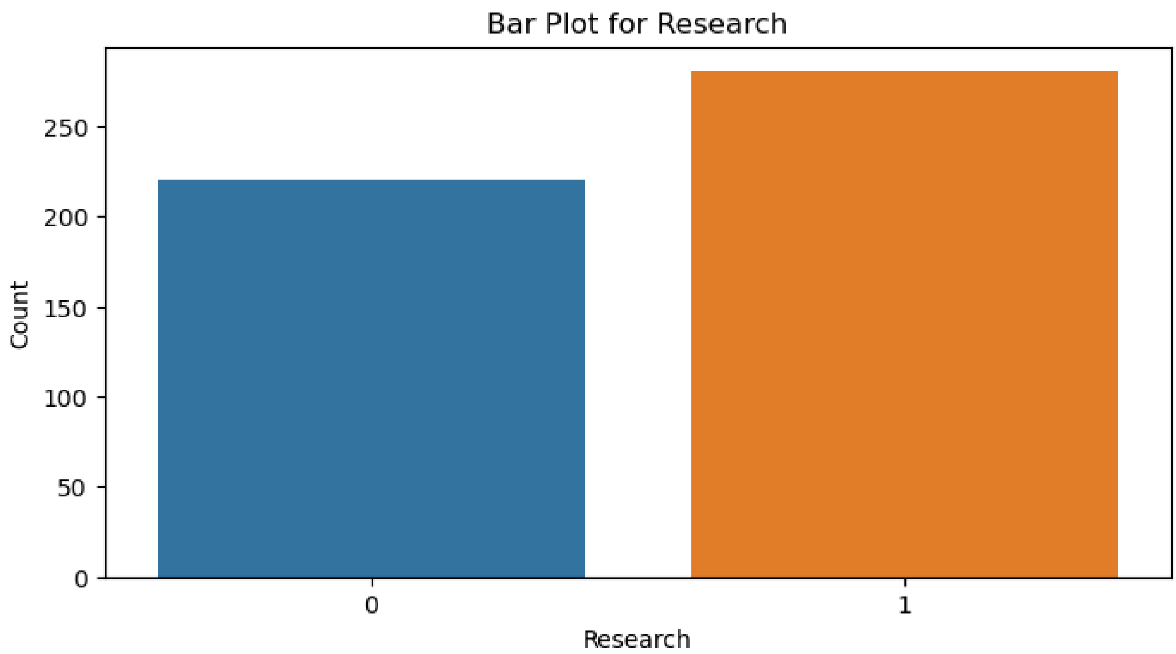

Distribution Plot for LOR
Frequency
LOR
0
20
40
60
80
100
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Distribution Plot for CGPA
Frequency
CGPA
0
10
20
30
40
50
60
70
7.0
7.5
8.0
8.5
9.0
9.5
10.0



In [142…

```
# Categorical variables
categorical_columns = ['Research']
for col in categorical_columns:
    plt.figure(figsize=(8, 4))
    sns.countplot(x=df[col])
    plt.title(f'Bar Plot for {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.show()
```



Insights from Univariate Analysis

- **Continuous Variables**: The distribution plots indicate that most continuous variables (GRE Score, TOEFL Score, CGPA, etc.) are approximately normally distributed with some skewness.
- **Categorical Variables**: The majority of students have participated in research, as indicated by the count plot of the 'Research' variable.

9. Bivariate analysis

In [143…

```
# Define continuous and categorical columns
continuous_columns = ['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR
categorical_columns = ['Research']
```

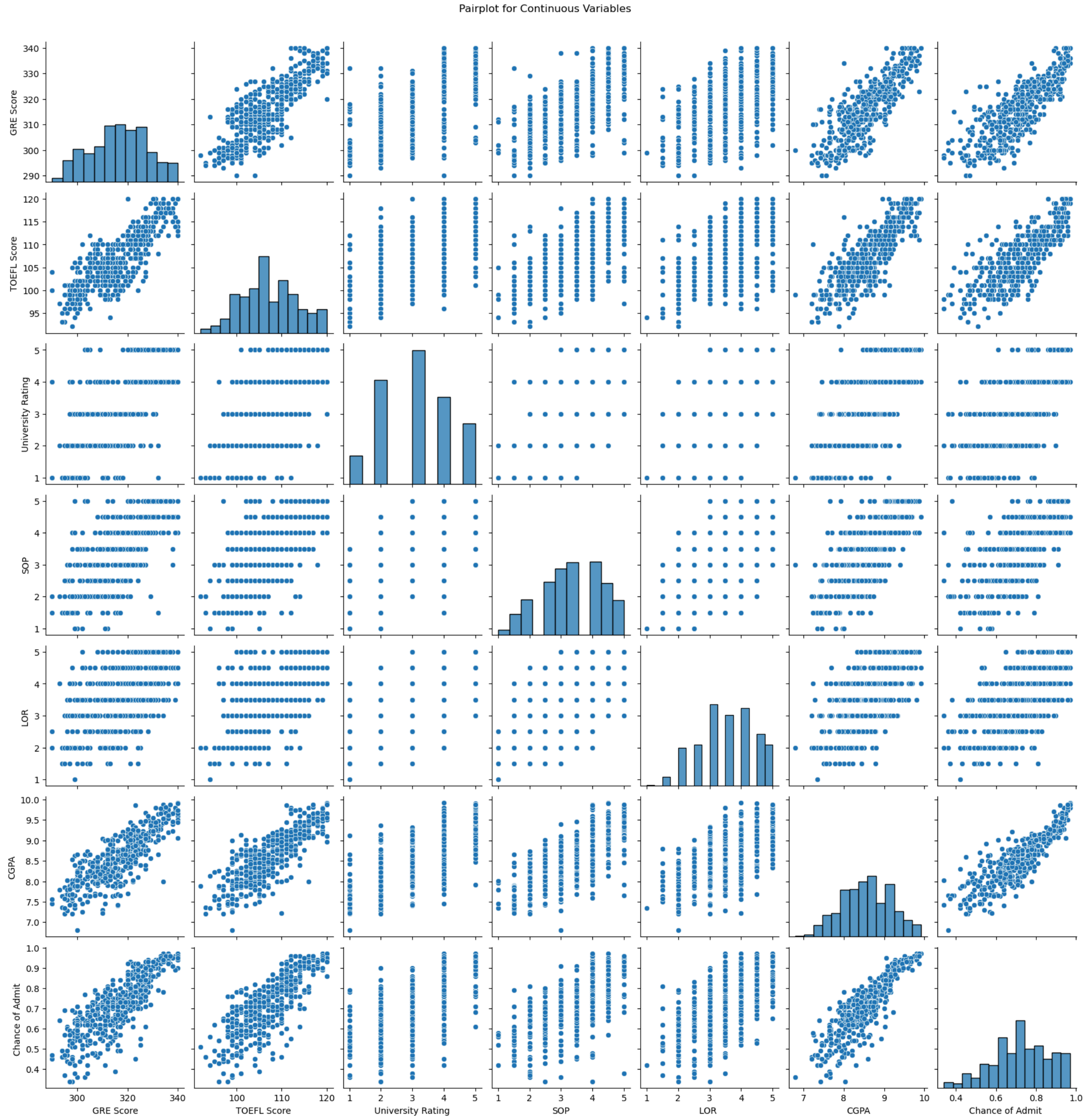
In [144…

```
# Non-graphical analysis: Correlation matrix for continuous variables
correlation_matrix = df[continuous_columns].corr()
print("Correlation Matrix:\n", correlation_matrix)
```

```
Correlation Matrix:
                    GRE Score  TOEFL Score  University Rating       SOP  \
GRE Score           1.000000     0.827200           0.635376  0.613498
TOEFL Score         0.827200     1.000000           0.649799  0.644410
University Rating   0.635376     0.649799           1.000000  0.728024
SOP                 0.613498     0.644410           0.728024  1.000000
LOR                 0.524679     0.541563           0.608651  0.663707
CGPA                0.825878     0.810574           0.705254  0.712154
Chance of Admit     0.810351     0.792228           0.690132  0.684137

                        LOR      CGPA  Chance of Admit
GRE Score          0.524679  0.825878         0.810351
TOEFL Score        0.541563  0.810574         0.792228
University Rating  0.608651  0.705254         0.690132
SOP                0.663707  0.712154         0.684137
LOR                1.000000  0.637469         0.645365
CGPA               0.637469  1.000000         0.882413
Chance of Admit    0.645365  0.882413         1.000000
```

In [145…

```
# Graphical analysis: Pairplot for continuous variables
sns.pairplot(df[continuous_columns])
plt.suptitle('Pairplot for Continuous Variables', y=1.02)
plt.show()
```

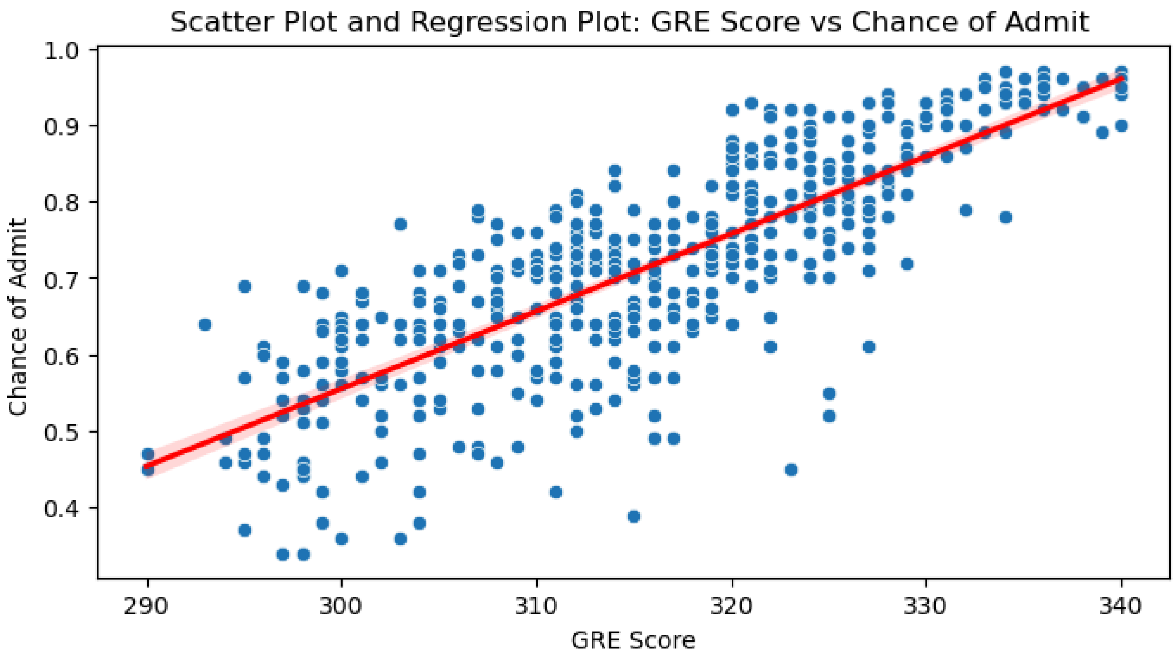


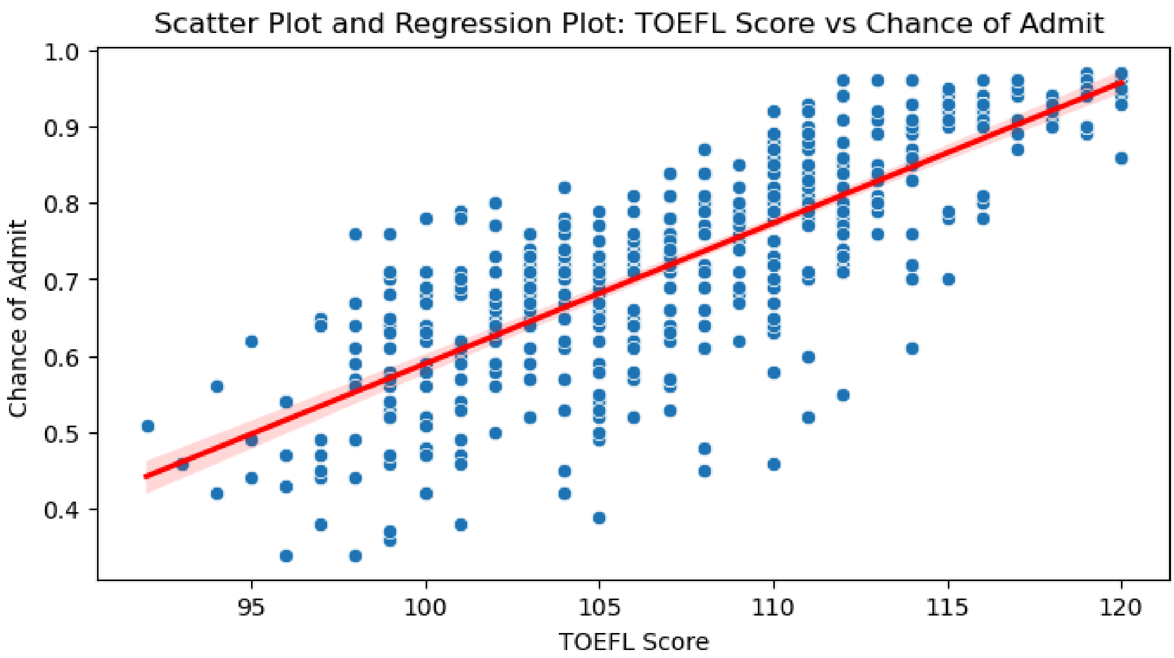
In [146…

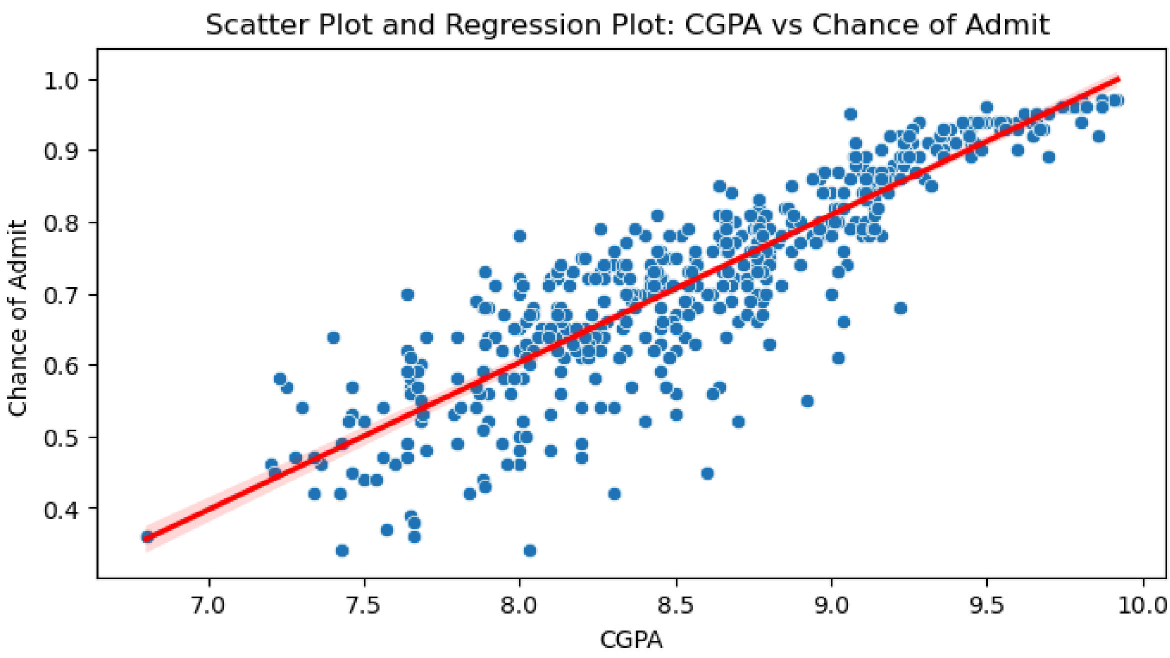
```
# Graphical analysis: Scatter plot and regression plot for selected pairs
pairs_to_plot = [
    ('GRE Score', 'Chance of Admit '),
    ('TOEFL Score', 'Chance of Admit '),
    ('CGPA', 'Chance of Admit '),
    ('University Rating', 'CGPA'),
    ('SOP', 'CGPA')
]
```

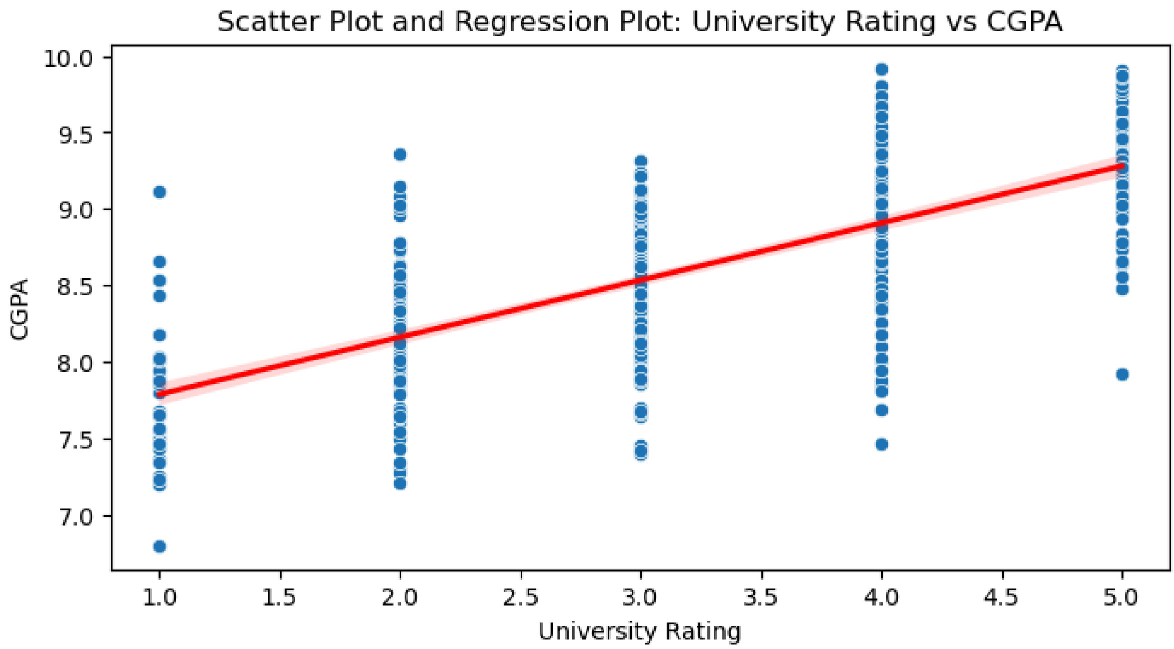
In [147…

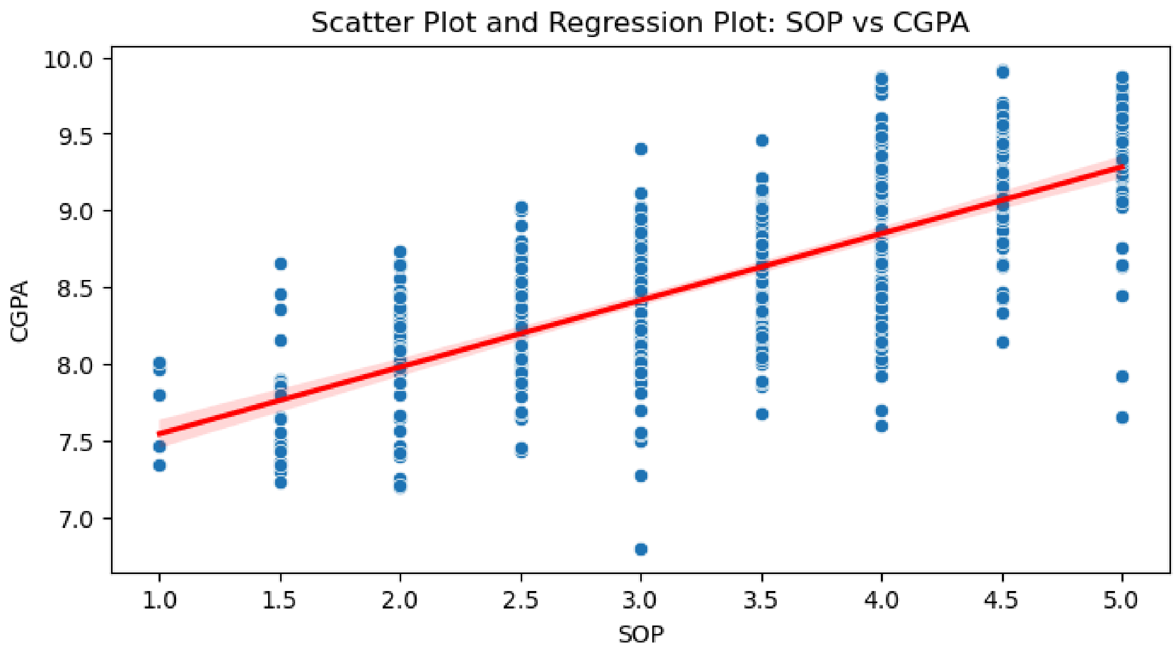
```
for x, y in pairs_to_plot:
    plt.figure(figsize=(8, 4))
    sns.scatterplot(x=df[x], y=df[y])
    sns.regplot(x=df[x], y=df[y], scatter=False, color='red')
    plt.title(f'Scatter Plot and Regression Plot: {x} vs {y}')
    plt.xlabel(x)
    plt.ylabel(y)
    plt.show()
```


Scatter Plot and Regression Plot: GRE Score vs Chance of Admit
Chance of Admit
GRE Score
1.0
0.9
0.8
0.7
0.6
0.5
0.4
290
300
310
320
330
340


Scatter Plot and Regression Plot: TOEFL Score vs Chance of Admit
Chance of Admit
TOEFL Score
1.0
0.9
0.8
0.7
0.6
0.5
0.4
95
100
105
110
115
120

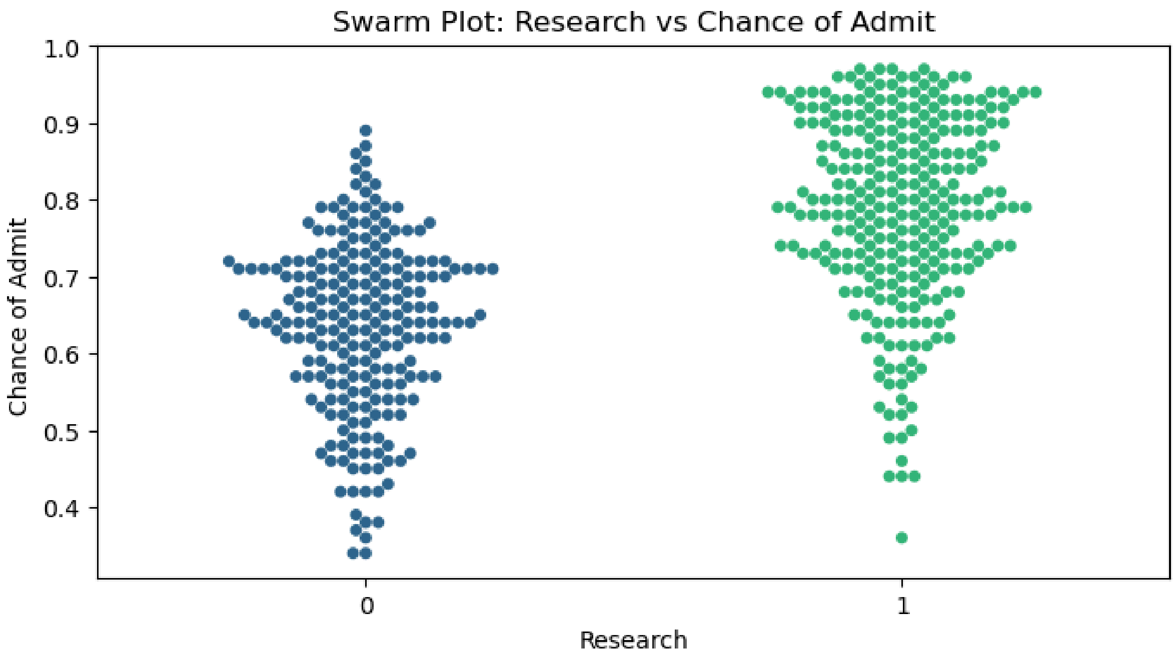

Scatter Plot and Regression Plot: CGPA vs Chance of Admit
Chance of Admit
CGPA
1.0
0.9
0.8
0.7
0.6
0.5
0.4
7.0
7.5
8.0
8.5
9.0
9.5
10.0





In [148…

```
# Graphical analysis: Swarmplot for categorical variables
for col in categorical_columns:
    plt.figure(figsize=(8, 4))
    sns.swarmplot(x=df[col], y=df['Chance of Admit '], palette='viridis')
    plt.title(f'Swarm Plot: {col} vs Chance of Admit')
    plt.show()
```



Insights from Bivariate Analysis

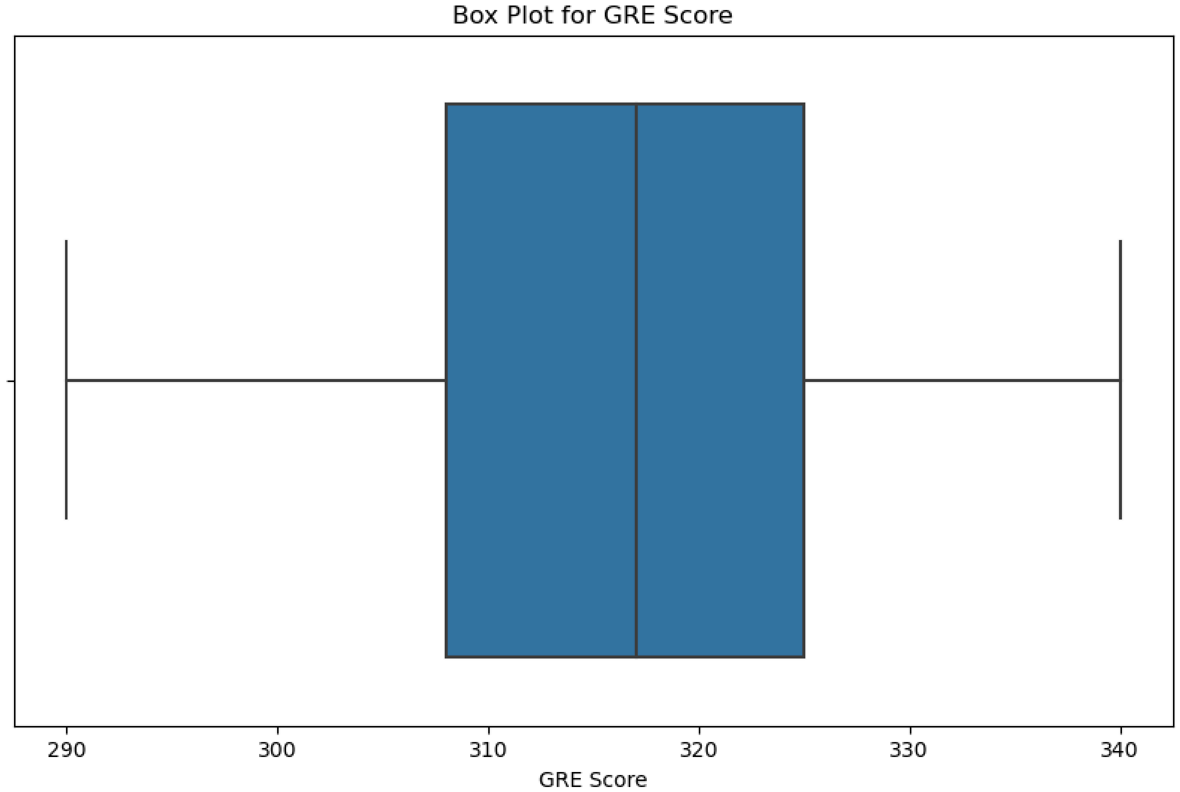
- **Correlation with Chance of Admit**: Higher GRE Scores, TOEFL Scores, and CGPA are positively correlated with a higher Chance of Admit, as shown by scatter plots and regression lines.
- **Research Impact**: Students with research experience tend to have a higher Chance of Admit, as indicated by the swarmplot.

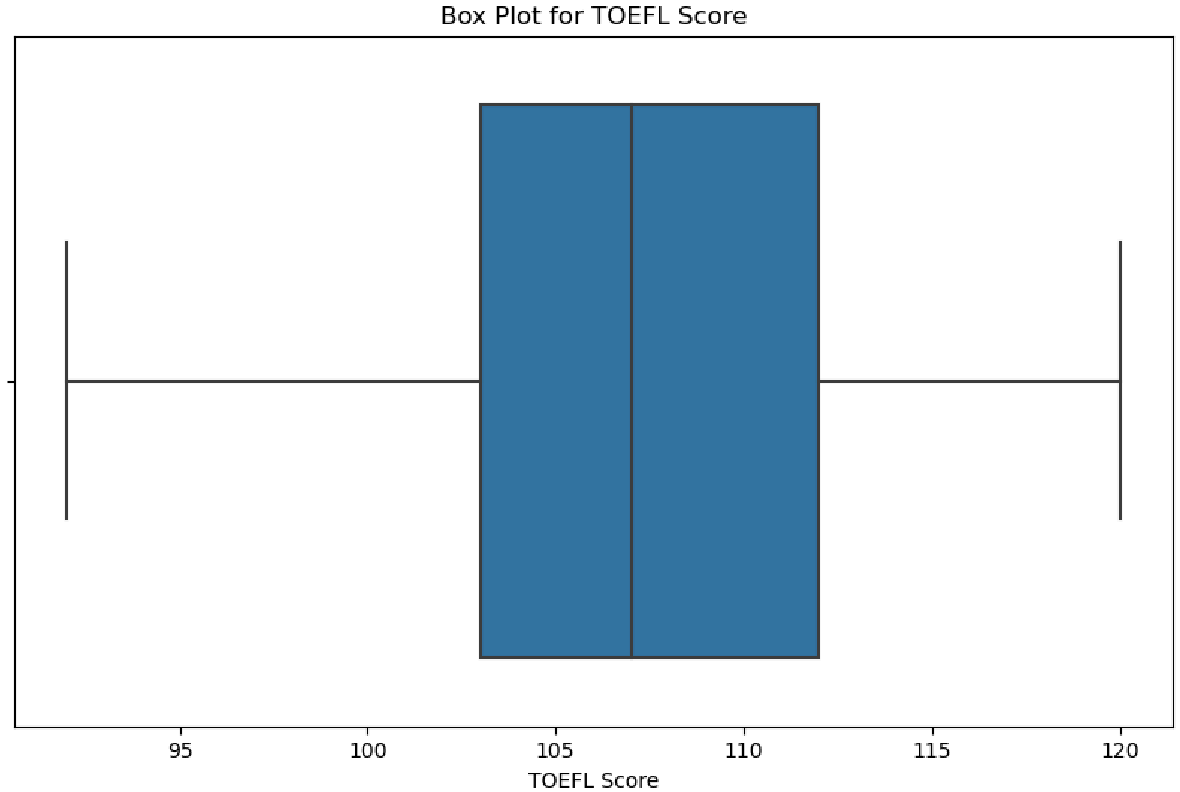
10. Data Preprocessing Steps

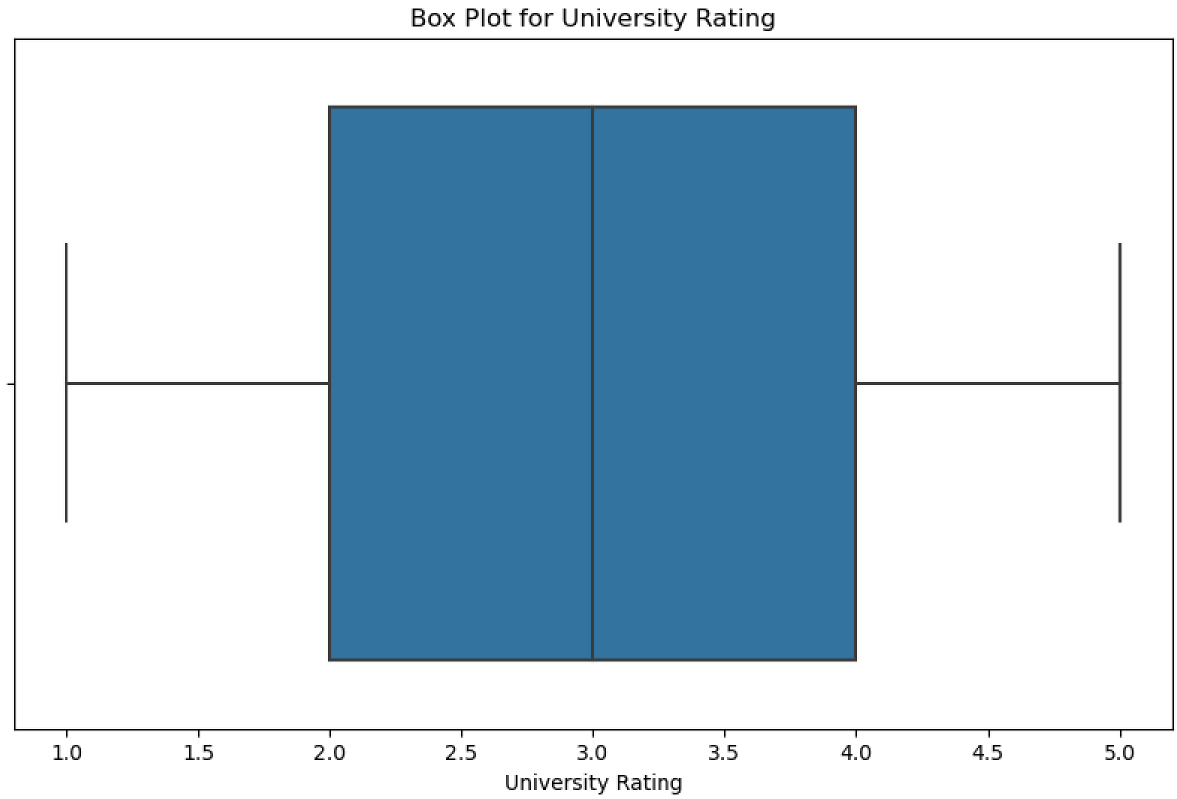
In [149…

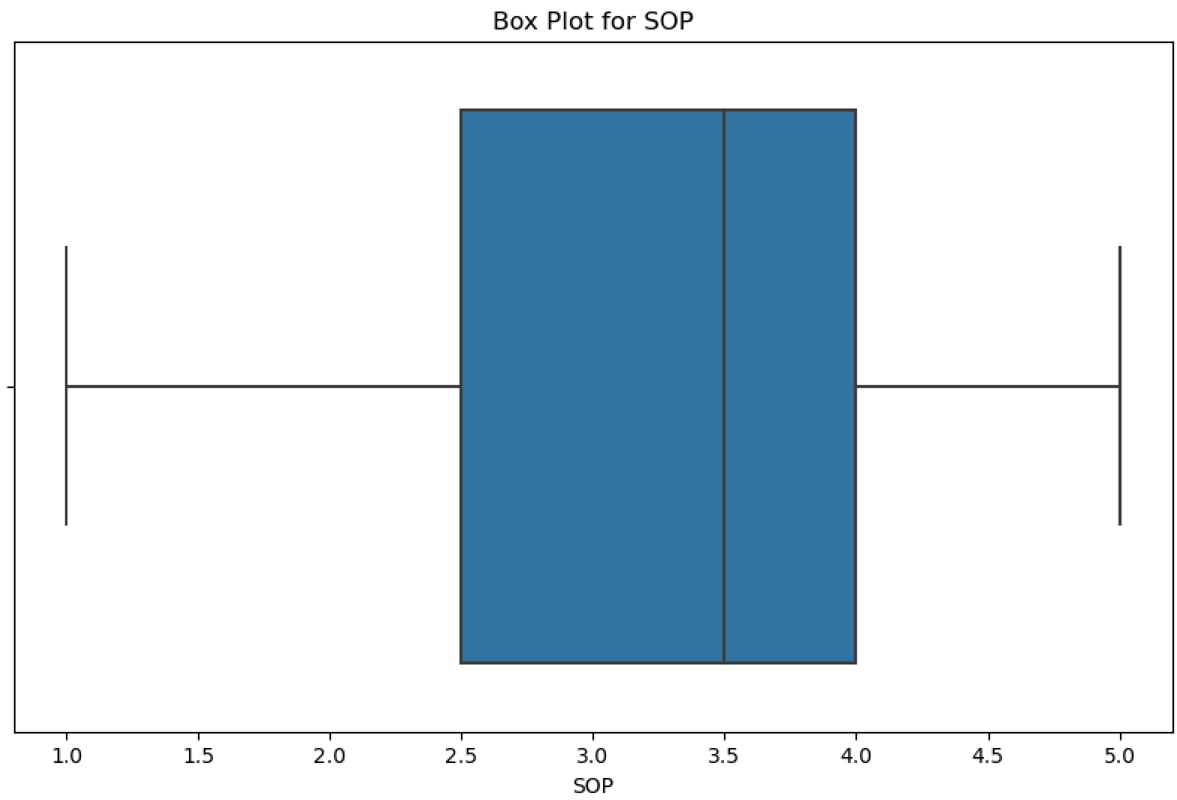
```
# Step 1: Check for Outliers
continuous_columns = ['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR

for col in continuous_columns:
    plt.figure(figsize=(10, 6))
    sns.boxplot(x=df[col])
    plt.title(f'Box Plot for {col}')
    plt.xlabel(col)
    plt.show()

# Step 2: Treat Outliers using IQR Method:
def treat_outliers(df, col):
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df[col] = np.where(df[col] < lower_bound, lower_bound, df[col])
    df[col] = np.where(df[col] > upper_bound, upper_bound, df[col])

for col in continuous_columns:
    treat_outliers(df, col)

# Step 3: Normalize/Scale the Data
scaler = StandardScaler()
df[continuous_columns] = scaler.fit_transform(df[continuous_columns])
```

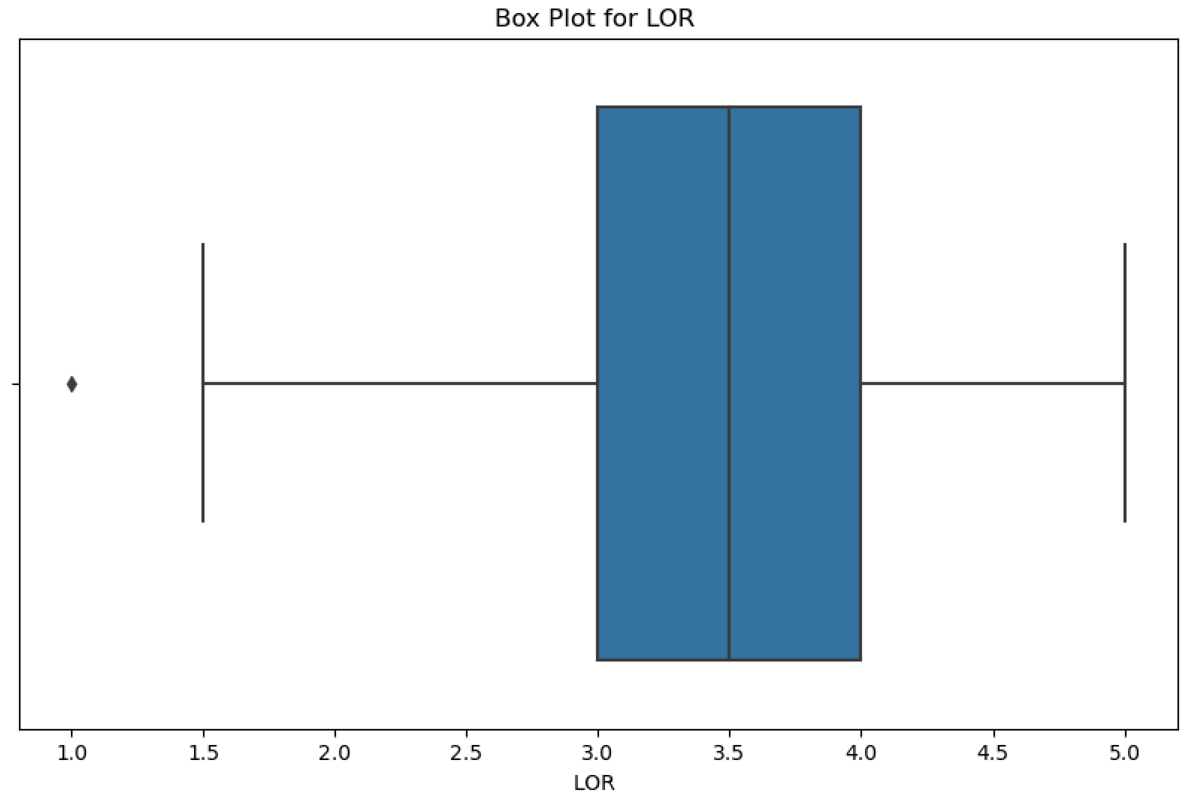
```
# Display the cleaned data
df.head()
```

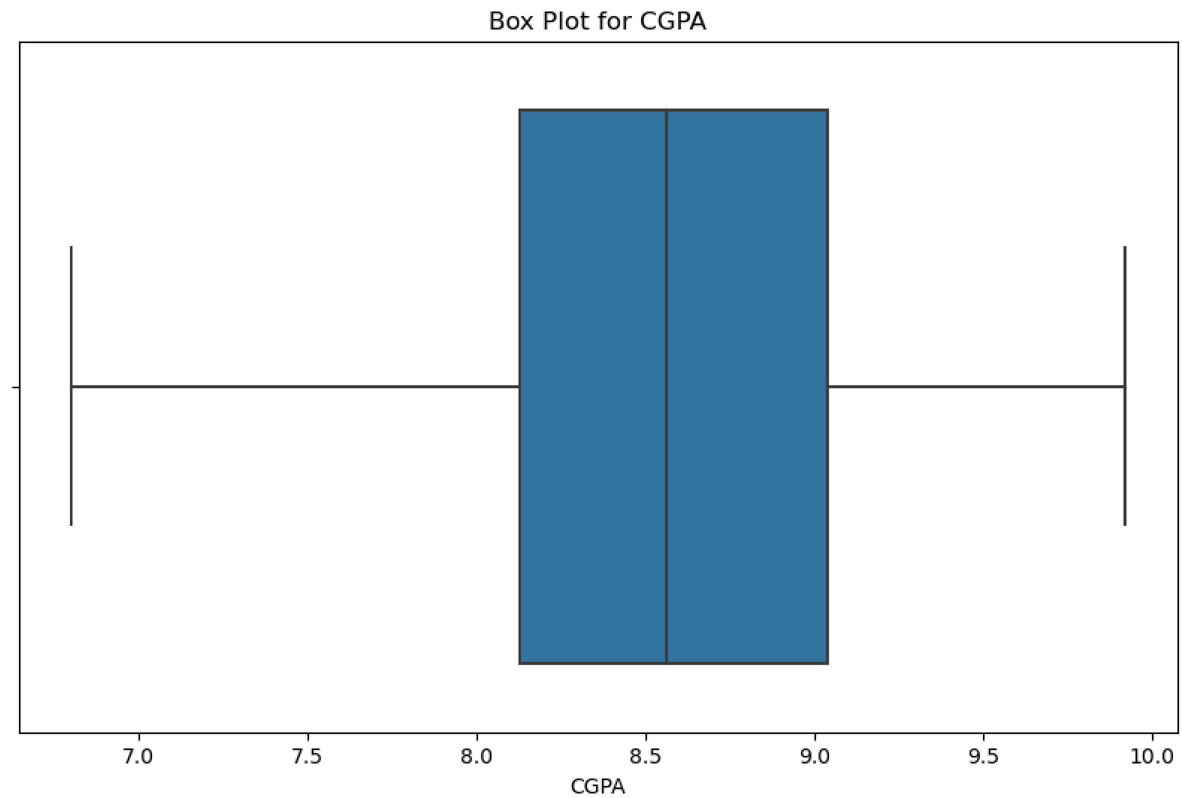


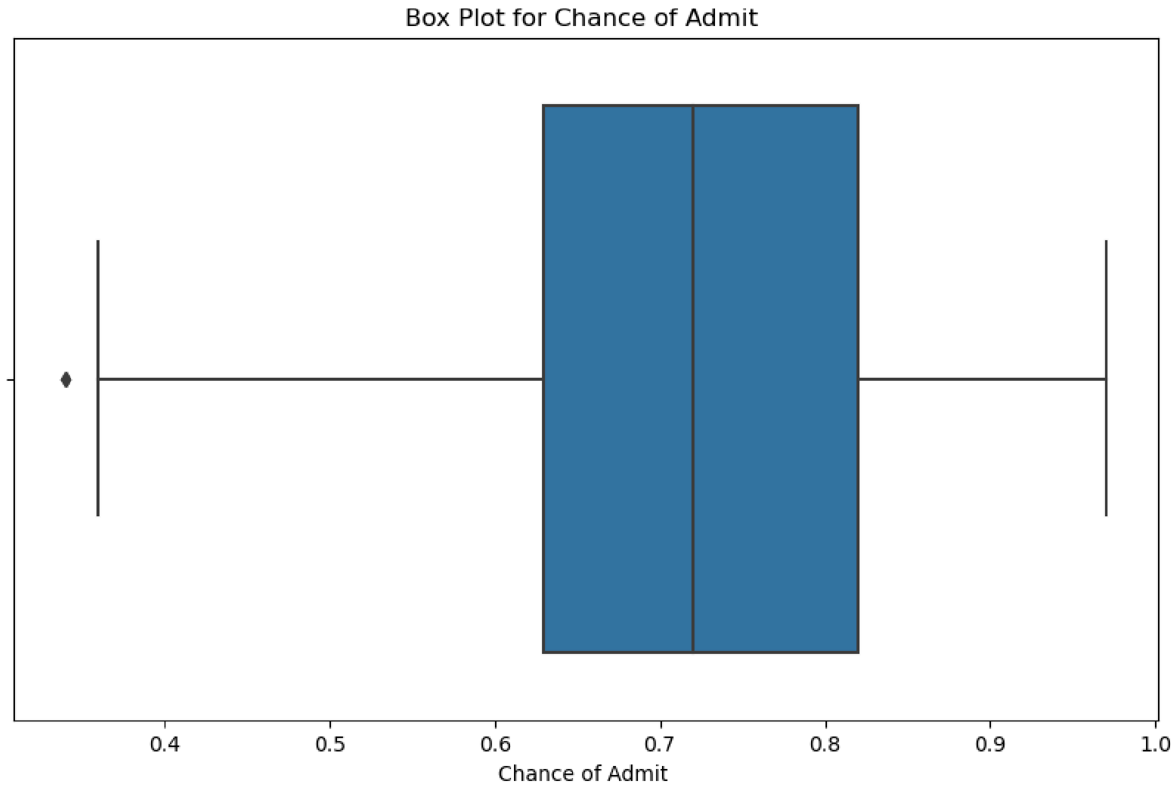



Box Plot for University Rating
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
University Rating


Box Plot for SOP
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
SOP


Box Plot for LOR
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
LOR


Box Plot for CGPA
7.0
7.5
8.0
8.5
9.0
9.5
10.0
CGPA



Out[149]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1.819238	1.778865	0.775582	1.137360	1.100744	1.776806	1	1.406502
1	0.667148	-0.031601	0.775582	0.632315	1.100744	0.485859	1	0.271311
2	-0.041830	-0.525364	-0.099793	-0.377773	0.016267	-0.954043	1	-0.012487
3	0.489904	0.462163	-0.099793	0.127271	-1.068210	0.154847	1	0.555109
4	-0.219074	-0.689952	-0.975168	-1.387862	-0.525971	-0.606480	0	-0.509133

11. Correlation among independent variables

Non-Graphical Analysis

Correlation Matrix:

In [150…

```
correlation_matrix = df.corr()
print(correlation_matrix)
```

	GRE Score	TOEFL Score	University Rating	SOP \
GRE Score	1.000000	0.827200	0.635376	0.613498
TOEFL Score	0.827200	1.000000	0.649799	0.644410
University Rating	0.635376	0.649799	1.000000	0.728024
SOP	0.613498	0.644410	0.728024	1.000000
LOR	0.524377	0.540630	0.608241	0.662848
CGPA	0.825878	0.810574	0.705254	0.712154
Research	0.563398	0.467012	0.427047	0.408116
Chance of Admit	0.810421	0.792292	0.690257	0.684380

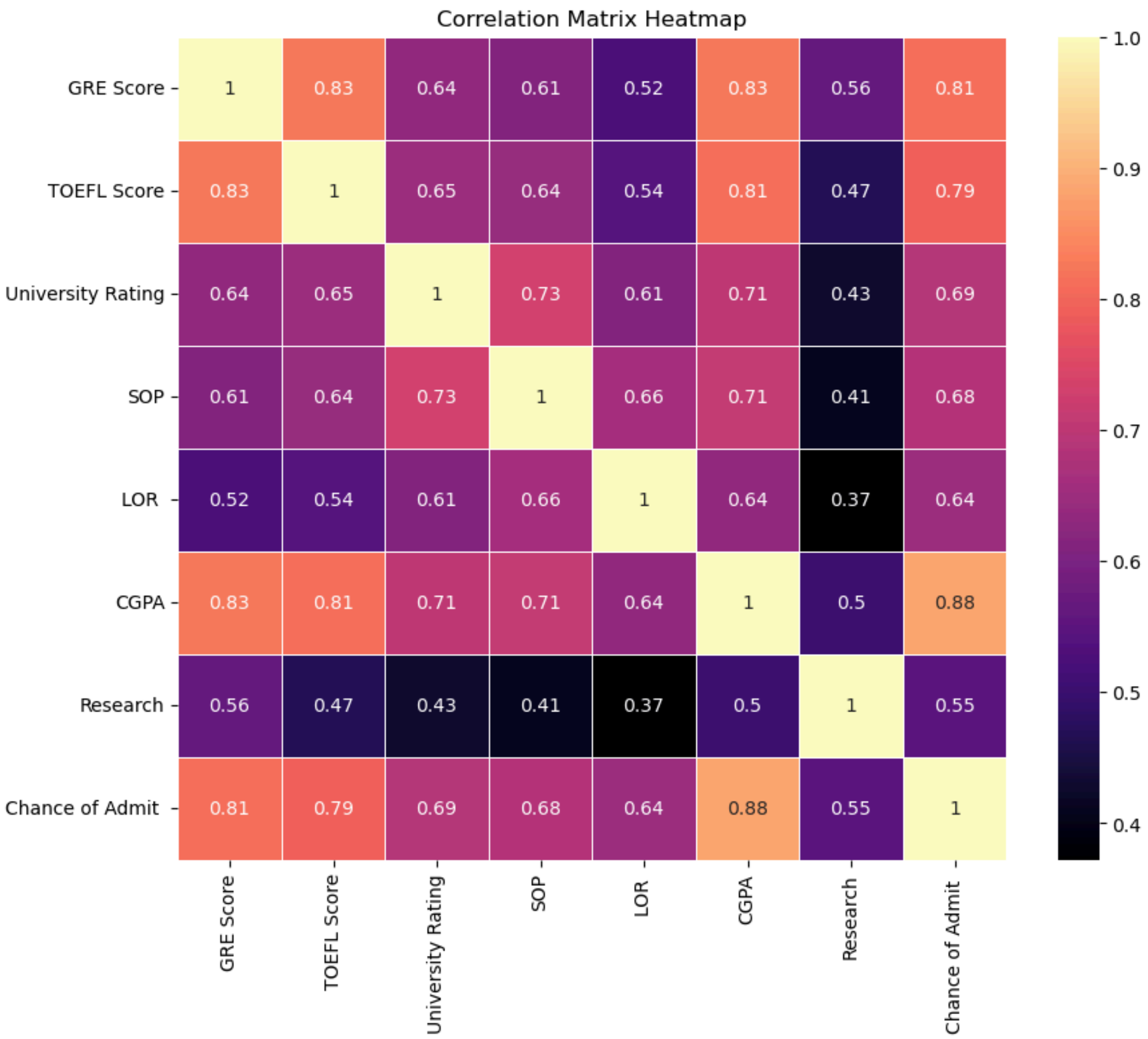
	LOR	CGPA	Research	Chance of Admit
GRE Score	0.524377	0.825878	0.563398	0.810421
TOEFL Score	0.540630	0.810574	0.467012	0.792292
University Rating	0.608241	0.705254	0.427047	0.690257
SOP	0.662848	0.712154	0.408116	0.684380
LOR	1.000000	0.636923	0.372280	0.644832
CGPA	0.636923	1.000000	0.501311	0.882551
Research	0.372280	0.501311	1.000000	0.545919
Chance of Admit	0.644832	0.882551	0.545919	1.000000

Graphical Analysis

Heatmap:

In [151…

```
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='magma', linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```

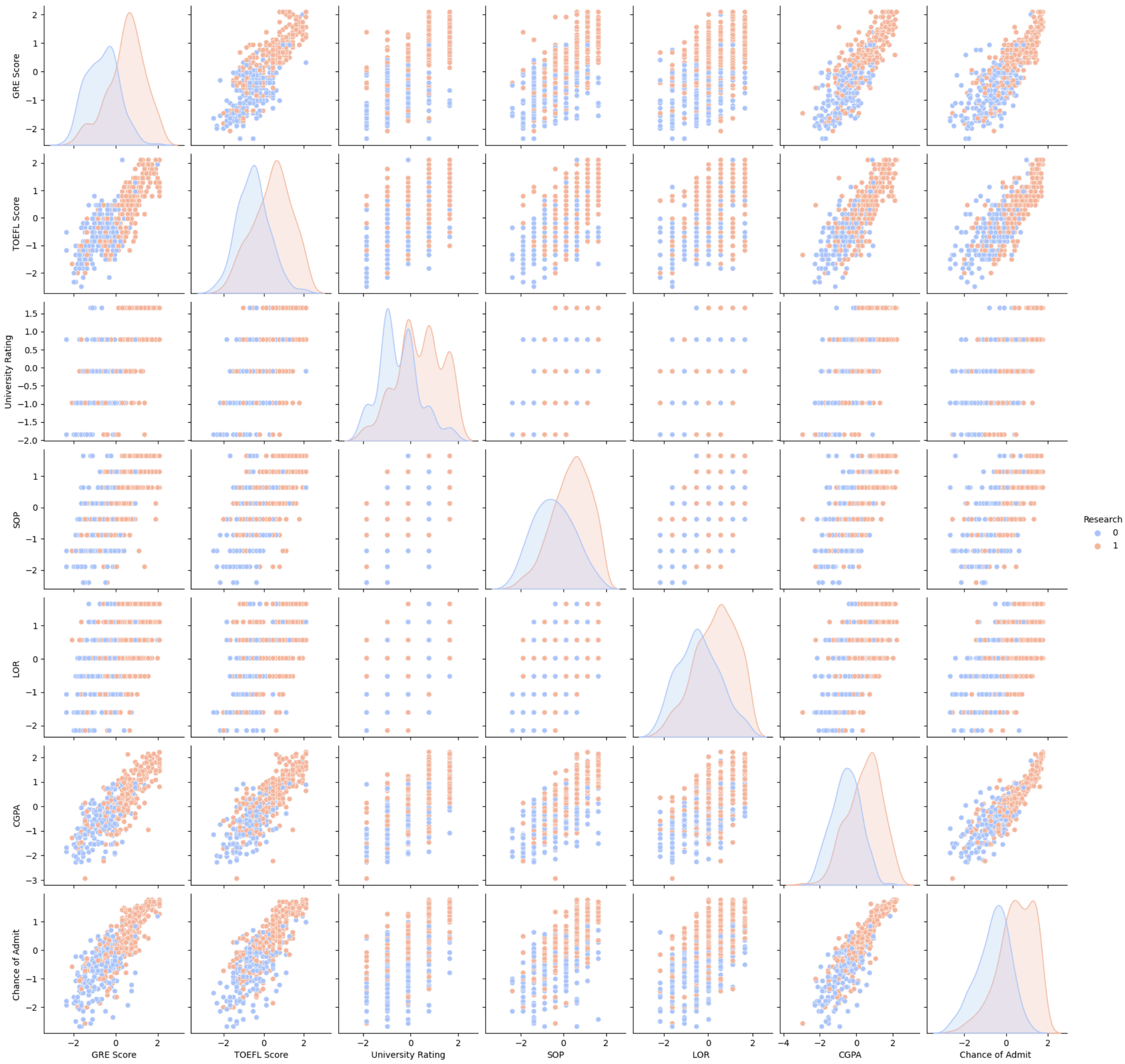


Pairplot

To visualize relationships between pairs of variables and the distribution of single variables

In [152…

```
sns.pairplot(df, vars=continuous_columns, hue='Research', palette='coolwarm')
plt.show()
```



Insights from Non-Graphical and Graphical Analysis

Correlation Among Variables:

- The correlation matrix and heatmap show that 'GRE Score', 'TOEFL Score', and 'CGPA' have strong positive correlations with 'Chance of Admit', indicating that higher scores in these areas increase the chances of admission.
- 'University Rating', 'SOP', and 'LOR' also show positive correlations with 'Chance of Admit', but to a lesser extent.

Interaction Between Variables:

- The pairplot reveals that students with research experience generally have higher GRE Scores, TOEFL Scores, CGPA, and 'Chance of Admit'.
- The pairplot also shows that there are some interactions between 'SOP', 'LOR', and 'University Rating', where higher ratings in these variables are associated with higher GRE and TOEFL Scores.

12. To build a Linear Regression model

In [153…

```
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 mean_squared_error, r2_score
```

In [154…

```
# Step 1: Encoding the Categorical Variables
# 'Research' is already numeric (0 or 1)

# Step 2: Performing the Train-Test Split
X = df.drop(columns=['Chance of Admit '])
y = df['Chance of Admit ']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [155…

```
# Step 3: Perform Data Normalization/Standardization
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [156…

```
# Step 4: Build Linear Regression Model
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

y_train_pred = lin_reg.predict(X_train)
y_test_pred = lin_reg.predict(X_test)

print("Linear Regression Model")
print("Training set performance:")
print(f"RMSE: {np.sqrt(mean_squared_error(y_train, y_train_pred))}")
print(f"R^2: {r2_score(y_train, y_train_pred)}")
print("\nTest set performance:")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_test_pred))}")
print(f"R^2: {r2_score(y_test, y_test_pred)}")

print("\nCoefficients:")
for name, coef in zip(X.columns, lin_reg.coef_):
    print(f"{name}: {coef}")
```

```
Linear Regression Model
Training set performance:
RMSE: 0.42080917084027697
R^2: 0.8213379717854047

Test set performance:
RMSE: 0.43197740546771624
R^2: 0.8187280986509647

Coefficients:
GRE Score: 0.18901042315872205
TOEFL Score: 0.12906415996670353
University Rating: 0.020802455864565528
SOP: 0.013168993396438886
LOR : 0.11236867888594886
CGPA: 0.479448428933418
Research: 0.08467506819210967
```

In [157…

```
# Step 5: Ridge and Lasso Regression
ridge_reg = Ridge(alpha=1.0)
ridge_reg.fit(X_train, y_train)

lasso_reg = Lasso(alpha=0.1)
lasso_reg.fit(X_train, y_train)
```

Out[157]:

```
▾ Lasso
Lasso(alpha=0.1)
```

In [158…

```
# Ridge Regression
y_train_pred_ridge = ridge_reg.predict(X_train)
y_test_pred_ridge = ridge_reg.predict(X_test)

print("\nRidge Regression Model")
print("Training set performance:")
print(f"RMSE: {np.sqrt(mean_squared_error(y_train, y_train_pred_ridge))}")
print(f"R^2: {r2_score(y_train, y_train_pred_ridge)}")
print("\nTest set performance:")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_test_pred_ridge))}")
print(f"R^2: {r2_score(y_test, y_test_pred_ridge)}")

print("\nCoefficients:")
for name, coef in zip(X.columns, ridge_reg.coef_):
    print(f"{name}: {coef}")
```

```
Ridge Regression Model
Training set performance:
RMSE: 0.42081387866192205
R^2: 0.821333974183989

Test set performance:
RMSE: 0.43204483307788083
R^2: 0.818671504556598

Coefficients:
GRE Score: 0.18985430553835583
TOEFL Score: 0.12997634729126578
University Rating: 0.021553868561648426
SOP: 0.014221212701226919
LOR : 0.11256239579017405
CGPA: 0.4754056890264306
Research: 0.08473778712703486
```

In [159…

```python
# Lasso Regression
y_train_pred_lasso = lasso_reg.predict(X_train)
y_test_pred_lasso = lasso_reg.predict(X_test)

print("\nLasso Regression Model")
print("Training set performance:")
print(f"RMSE: {np.sqrt(mean_squared_error(y_train, y_train_pred_lasso))}")
print(f"R^2: {r2_score(y_train, y_train_pred_lasso)}")
print("\nTest set performance:")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_test_pred_lasso))}")
print(f"R^2: {r2_score(y_test, y_test_pred_lasso)}")

print("\nCoefficients:")
for name, coef in zip(X.columns, lasso_reg.coef_):
    print(f"{name}: {coef}")
```

```
Lasso Regression Model
Training set performance:
RMSE: 0.4399394976190073
R^2: 0.8047244916671988

Test set performance:
RMSE: 0.44611503769231337
R^2: 0.8066687084939047

Coefficients:
GRE Score: 0.1848883510044935
TOEFL Score: 0.09737256560136835
University Rating: 0.0
SOP: 0.0
LOR : 0.06342447056861057
CGPA: 0.4916972518494586
Research: 0.026117914136180698
```

Insights from the Model Statistics

1. Linear Regression Model:

- Evaluate RMSE and R^2 scores for both training and test sets.
- Analyze the coefficients to understand the importance of each feature. ### 2.Ridge and Lasso Regression Models:

- Compare the RMSE and R^2 scores to those of the Linear Regression model.

- Ridge and Lasso can help handle multicollinearity and feature selection respectively, indicated by different coefficients.

13. To test the assumptions of the linear regression model:

1. Multicollinearity Check using VIF:

- Drop variables one-by-one till none has VIF > 5.

2. Mean of Residuals:

- Check if the mean of residuals is nearly zero.

3. Linearity of Variables:

- Check for no pattern in the residual plot.

4. Homoscedasticity:

- Check for constant variance in the residual plot.

5. Normality of Residuals:

- Check if the residuals follow a normal distribution and points in the QQ plot lie on the line.

In [160…

```
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import norm, probplot
```

In [161…

```
X = df.drop(columns=['Chance of Admit '])
y = df['Chance of Admit ']

# Splitting the dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Adding constant to the model
X_train_sm = sm.add_constant(X_train)

# Fitting the model
model = sm.OLS(y_train, X_train_sm).fit()

# VIF Calculation
vif_data = pd.DataFrame()
vif_data["feature"] = X_train.columns
vif_data["VIF"] = [variance_inflation_factor(X_train.values, i) for i in range(len(
```

In [162…

```
# Dropping variables with high VIF one by one (example)
while vif_data['VIF'].max() > 5:
    drop_var = vif_data.sort_values('VIF', ascending=False).iloc[0]['feature']
    X_train = X_train.drop(columns=[drop_var])
    X_test = X_test.drop(columns=[drop_var])
    X_train_sm = sm.add_constant(X_train)
    model = sm.OLS(y_train, X_train_sm).fit()
    vif_data = pd.DataFrame()
    vif_data["feature"] = X_train.columns
    vif_data["VIF"] = [variance_inflation_factor(X_train.values, i) for i in range(

print(vif_data)

# Predictions
y_train_pred = model.predict(X_train_sm)

# Residuals
residuals = y_train - y_train_pred
```

```
                feature       VIF
0             GRE Score  4.227737
1           TOEFL Score  3.658212
2     University Rating  2.558111
3                   SOP  2.785328
4                   LOR  1.977583
5                  CGPA  4.654023
6              Research  1.195546
```

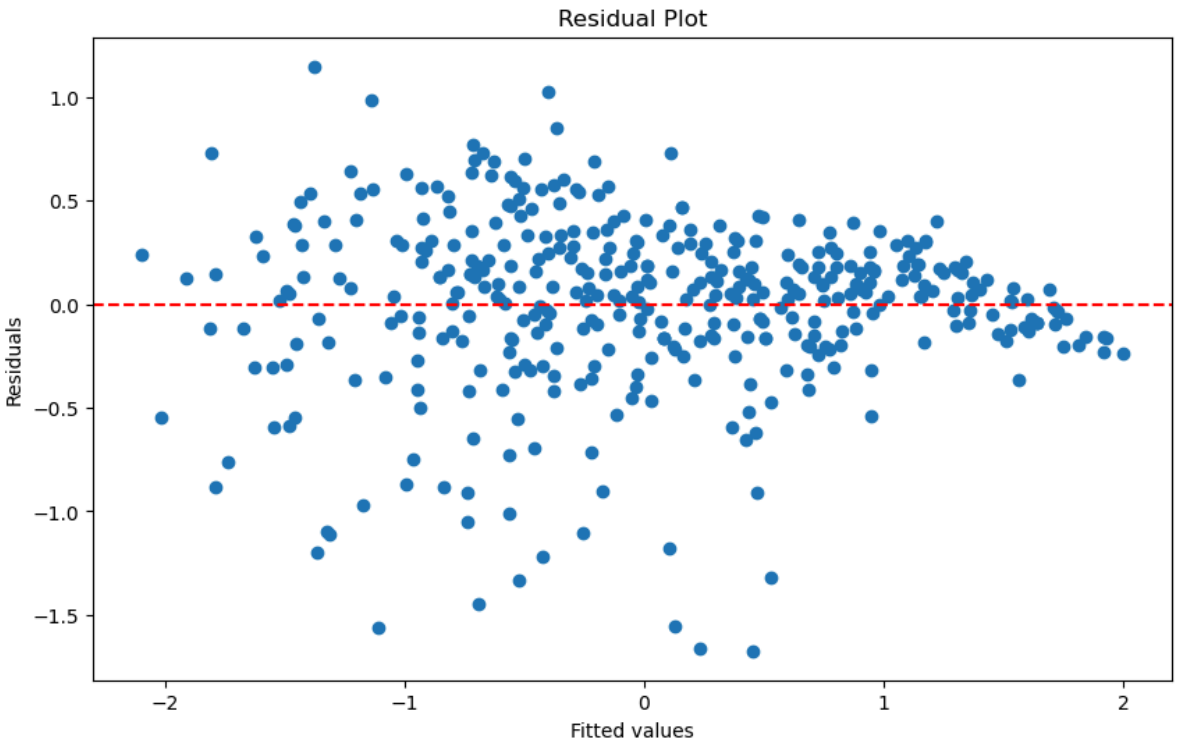
In [163…

```
# Mean of Residuals
mean_residuals = np.mean(residuals)
print(f"Mean of Residuals: {mean_residuals}")
```

```
Mean of Residuals: 9.992007221626408e-18
```

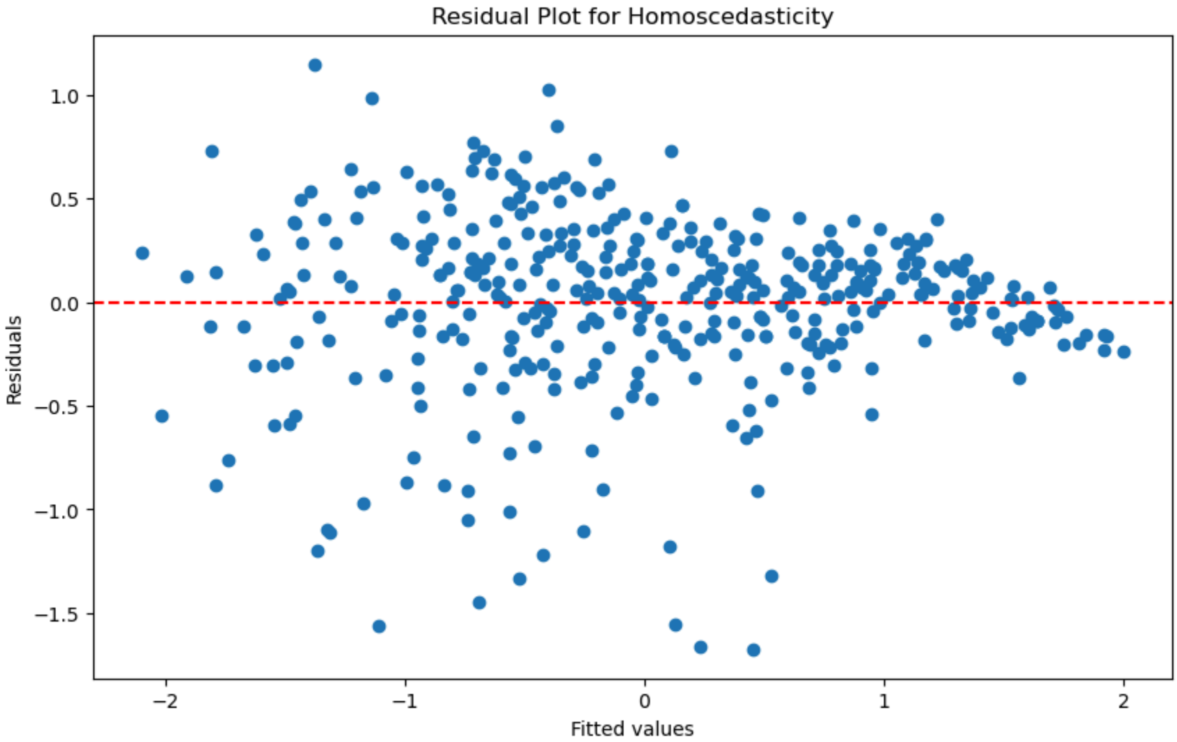
In [164…

```
# Residual Plot for Linearity
plt.figure(figsize=(10, 6))
plt.scatter(y_train_pred, residuals)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Fitted values')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.show()
```



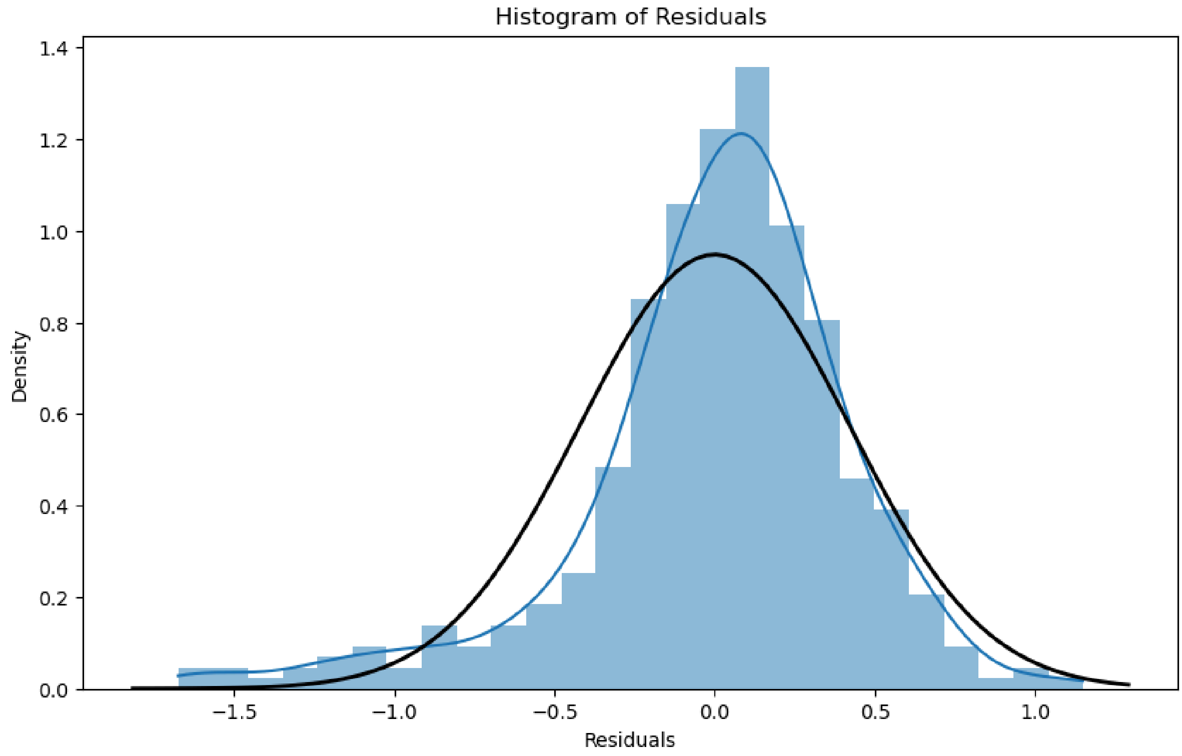
In [165…

```
# Homoscedasticity
plt.figure(figsize=(10, 6))
plt.scatter(y_train_pred, residuals)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Fitted values')
plt.ylabel('Residuals')
plt.title('Residual Plot for Homoscedasticity')
plt.show()
```



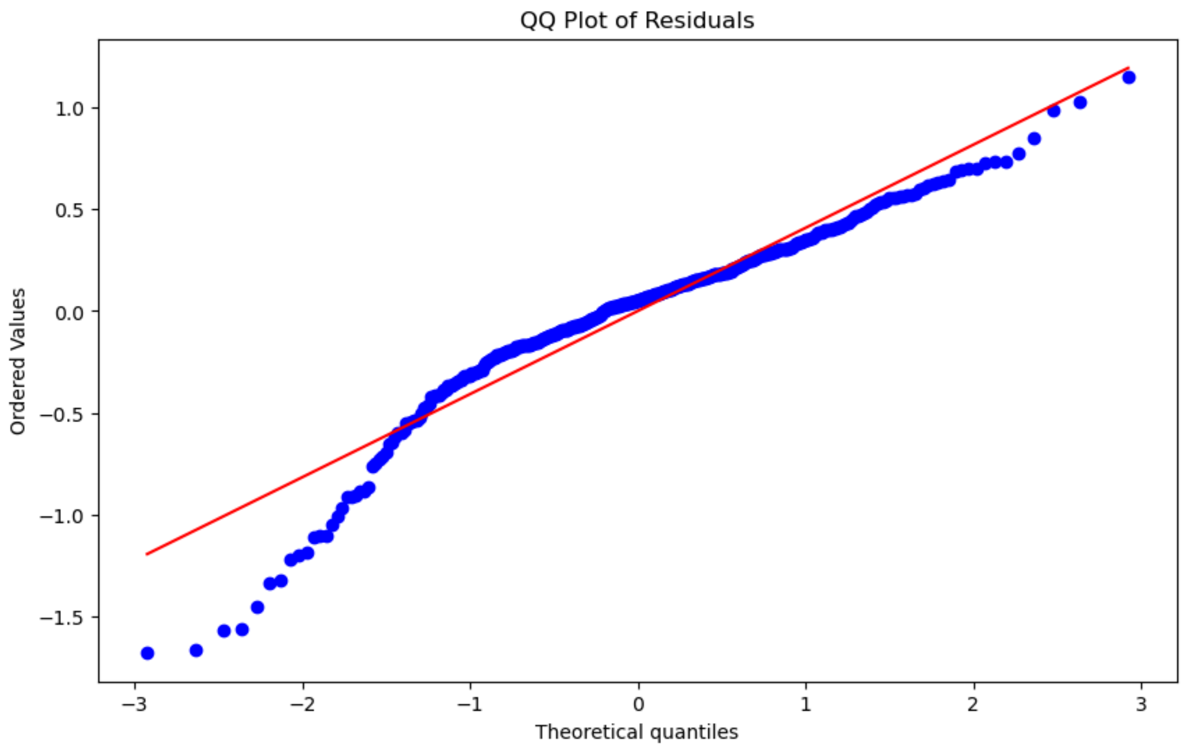
In [166…

```
# Plot histogram of residuals with a KDE
plt.figure(figsize=(10, 6))
sns.histplot(residuals, kde=True, stat="density", linewidth=0)
plt.title('Histogram of Residuals')

# Fit a normal distribution to the residuals
mu, std = norm.fit(residuals)
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, mu, std)

# Plot the normal distribution fit
plt.plot(x, p, 'k', linewidth=2)
plt.xlabel('Residuals')
plt.ylabel('Density')
plt.show()
```



In [167…

```
# QQ Plot for Normality
plt.figure(figsize=(10, 6))
probplot(residuals, dist="norm", plot=plt)
plt.title('QQ Plot of Residuals')
plt.show()
```



Insights:

1. Multicollinearity Check using VIF:

- Variables with high VIF were iteratively removed to ensure no multicollinearity among predictors.

2. Mean of Residuals:

- The mean of residuals is nearly zero, indicating that the model's errors are centered around zero.

3. Linearity of Variables:

- The residual plot shows no obvious patterns, suggesting that the relationship between predictors and the target is linear.

4. Homoscedasticity:

- The residual plot indicates constant variance of residuals, confirming homoscedasticity.

5. Normality of Residuals:

- The histogram and QQ plot indicate that the residuals are approximately normally distributed.

14. Model performance

In [168…

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

In [169…

```
# Add a constant to the predictors to include an intercept in the model
X_train_const = sm.add_constant(X_train)
X_test_const = sm.add_constant(X_test)

# Fit the model again with the constant term
model = sm.OLS(y_train, X_train_const).fit()

# Function to calculate adjusted R-squared
def adjusted_r2_score(r2, n, p):
    return 1 - ((1 - r2) * (n - 1) / (n - p - 1))

# Predict on train and test sets
y_train_pred = model.predict(X_train_const)
y_test_pred = model.predict(X_test_const)
```

In [170…

```
# Calculate metrics for train set
mae_train = mean_absolute_error(y_train, y_train_pred)
rmse_train = mean_squared_error(y_train, y_train_pred, squared=False)
r2_train = r2_score(y_train, y_train_pred)
adj_r2_train = adjusted_r2_score(r2_train, X_train_const.shape[0], X_train_const.sh
```

In [171…

```
# Calculate metrics for test set
mae_test = mean_absolute_error(y_test, y_test_pred)
rmse_test = mean_squared_error(y_test, y_test_pred, squared=False)
r2_test = r2_score(y_test, y_test_pred)
adj_r2_test = adjusted_r2_score(r2_test, X_test_const.shape[0], X_test_const.shape[
```

In [172…

```
# Display metrics
print(f'Train Set Metrics:')
print(f'MAE: {mae_train}')
print(f'RMSE: {rmse_train}')
```

```
print(f'R²: {r2_train}')
print(f'Adjusted R²: {adj_r2_train}')

print(f'\nTest Set Metrics:')
print(f'MAE: {mae_test}')
print(f'RMSE: {rmse_test}')
print(f'R²: {r2_test}')
print(f'Adjusted R²: {adj_r2_test}')
```

```
Train Set Metrics:
MAE: 0.301555030186361
RMSE: 0.420809170840277
R²: 0.8213379717854047
Adjusted R²: 0.8176824827170753

Test Set Metrics:
MAE: 0.3036410106226582
RMSE: 0.4319774054677161
R²: 0.8187280986509647
Adjusted R²: 0.8027921073235771
```

In [173…

```
# Comment on the performance
if abs(r2_train - r2_test) < 0.1:
    print("\nThe model has similar performance on the train and test sets, indicati
else:
    print("\nThe model has different performance on the train and test sets, indica

if r2_test < 0.7:
    print("The model may need improvement, as the R² value on the test set is relat
else:
    print("The model performs well on the test set with a good R² value.")
```

```
The model has similar performance on the train and test sets, indicating good gene
ralization.
The model performs well on the test set with a good R² value.
```

Insights from the Evaluation:

1. Model Generalization:

- By comparing the R^2 values for the train and test sets, you can assess whether the model generalizes well to unseen data.

2. Model Accuracy:

- High R^2 and adjusted R^2 values on the test set indicate a good fit, while low values suggest the need for model improvement.

15. Actionable Insights & Recommendations

1. Significance of Predictor Variables

- **GRE Score, TOEFL Score, CGPA:** These variables have the highest positive correlation with the chance of admission. Students with higher scores in these areas have a higher probability of being admitted.

- **University Rating, SOP, LOR:** These also positively correlate with the chance of admission but are less significant compared to the above variables. They still play a crucial role in the holistic evaluation of the applicant.
- **Research:** Having research experience has a notable impact on admission chances, indicating the importance of research work in the evaluation process.

2. Additional Data Sources for Model Improvement

- **Work Experience:** Including data on the applicant's work experience could provide insights into how professional experience impacts admission chances.
- **Extracurricular Activities:** Information about involvement in extracurricular activities can help assess the overall profile strength of the applicant.
- **Personal Statement Analysis:** Text analysis of personal statements could provide qualitative insights into the applicant's motivation and fit for the program.
- **Letter of Recommendation Content:** The content and sentiment analysis of letters of recommendation can add depth to the evaluation of LOR scores.

3. Model Implementation in Real World

- **Automated Application Screening:** Implementing the model in an automated system can help universities quickly filter and prioritize applications, saving time and resources in the admissions process.
- **Personalized Feedback:** The model can provide personalized feedback to applicants, highlighting areas for improvement (e.g., suggesting retaking standardized tests or gaining research experience).
- **Data-Driven Decision Making:** Admissions committees can use the model to make more informed, data-driven decisions, ensuring a fair and objective evaluation process.

4. Potential Business Benefits from Improving the Model

- **Enhanced Efficiency:** Automating the initial screening process allows admissions officers to focus on more detailed aspects of applications, improving overall efficiency.
- **Improved Applicant Experience:** By providing clear feedback and transparent evaluation criteria, the model can enhance the applicant experience, leading to higher satisfaction and potentially attracting more high-quality applicants.
- **Data-Backed Strategies:** Universities can leverage insights from the model to develop targeted marketing and recruitment strategies, improving the quality and diversity of the applicant pool.
- **Increased Admission Rates of Qualified Candidates:** By accurately identifying the most promising candidates, the university can improve its admission rates of highly qualified students, enhancing the institution's reputation and academic standards.

In [ ]:
```

```