In [54]:
```
#importing necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_facto
from statsmodels.compat import lzip
import statsmodels.stats.api as sms
import matplotlib.pyplot as plt
from scipy.stats import shapiro
from statsmodels.formula.api import ols
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_s
from sklearn.preprocessing import StandardScaler
```

In [55]:
```
#Load the data set and create a data frame for the same.
df = pd.read_csv('Jamboree_Admission.csv')
df.head()
```

Out[55]:

	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 [56]:
```
#Checking the shape of the data set
df.shape
```

Out[56]: (500, 9)

In [57]:
```
#statistical summary of the data set . Serial No. is out of scope for thi
df_stat_summary = df.drop('Serial No.', axis=1).describe()
df_stat_summary
```

Out[57]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA
count	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000
mean	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813
min	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000
25%	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500
50%	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000
75%	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000
max	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000

In [58]:

```
#datatypes for the dataset
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 [59]:

```
#Missing values in the dataset
df.isnull().sum()
```

Out[59]:

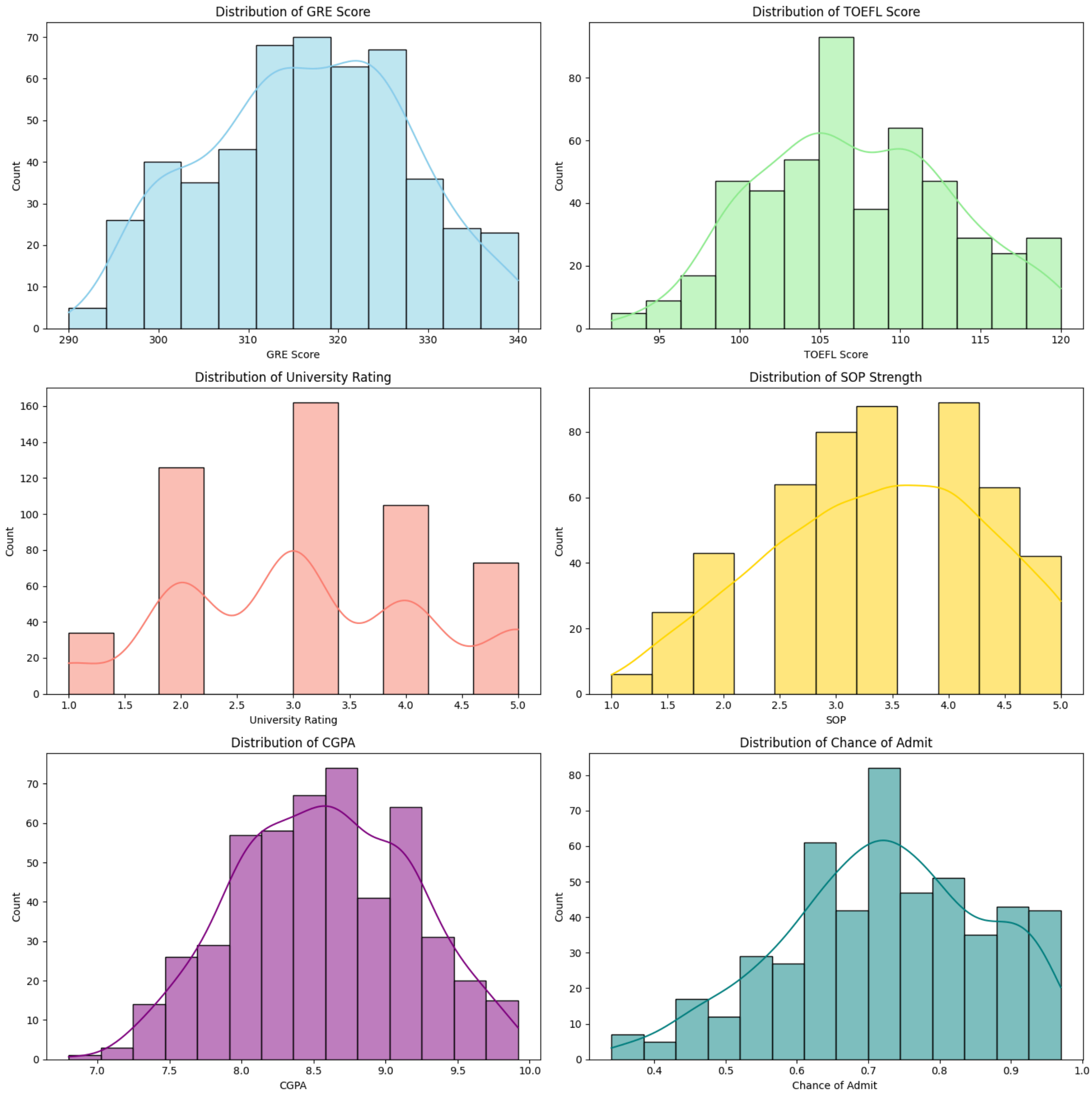
	0
Serial No.	0
GRE Score	0
TOEFL Score	0
University Rating	0
SOP	0
LOR	0
CGPA	0
Research	0
Chance of Admit	0

dtype: int64

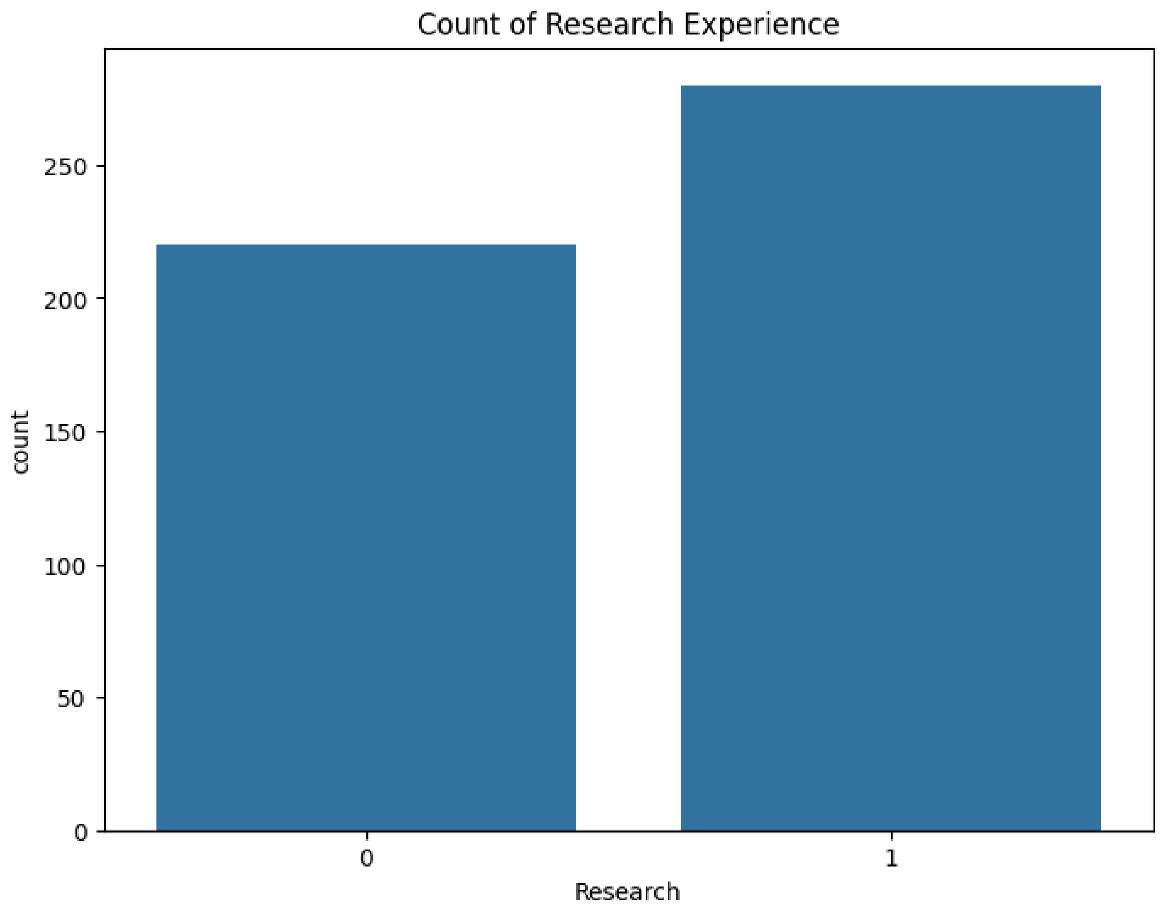
In [60]:
```
#Duplicate entry count in the data set
duplicate_count = df.duplicated().sum()
duplicate_count
```

Out[60]: 0

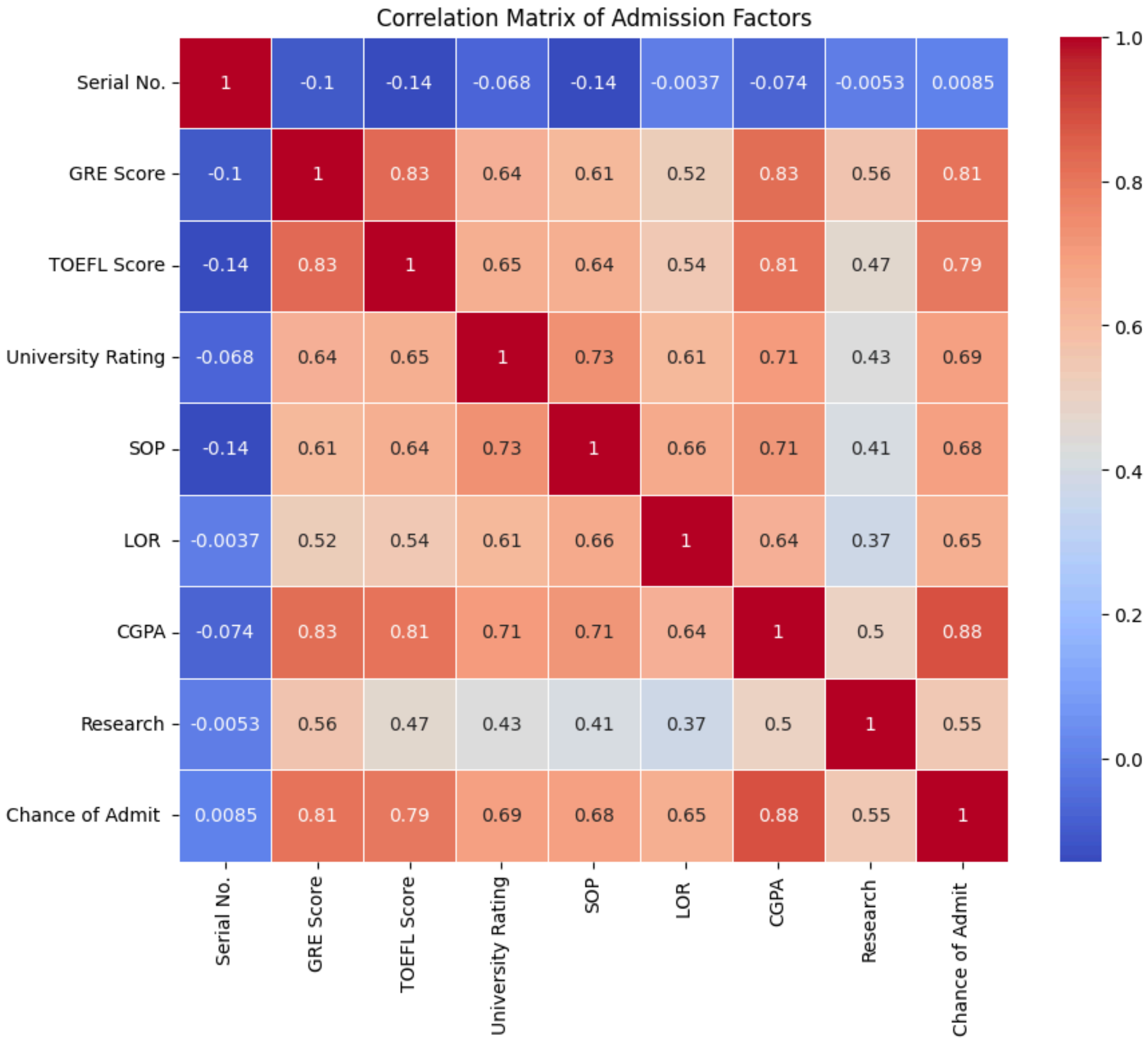
In [61]:
```
#Univariate Analysis
# Set up subplots
fig, axes = plt.subplots(3, 2, figsize=(15, 15))

# Plot distributions
sns.histplot(df['GRE Score'], kde=True, ax=axes[0, 0], color='skyblue')
axes[0, 0].set_title('Distribution of GRE Score')

sns.histplot(df['TOEFL Score'], kde=True, ax=axes[0, 1], color='lightgree
axes[0, 1].set_title('Distribution of TOEFL Score')

sns.histplot(df['University Rating'], kde=True, ax=axes[1, 0], color='sal
axes[1, 0].set_title('Distribution of University Rating')

sns.histplot(df['SOP'], kde=True, ax=axes[1, 1], color='gold')
axes[1, 1].set_title('Distribution of SOP Strength')

sns.histplot(df['CGPA'], kde=True, ax=axes[2, 0], color='purple')
axes[2, 0].set_title('Distribution of CGPA')

sns.histplot(df['Chance of Admit '], kde=True, ax=axes[2, 1], color='teal
axes[2, 1].set_title('Distribution of Chance of Admit')

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



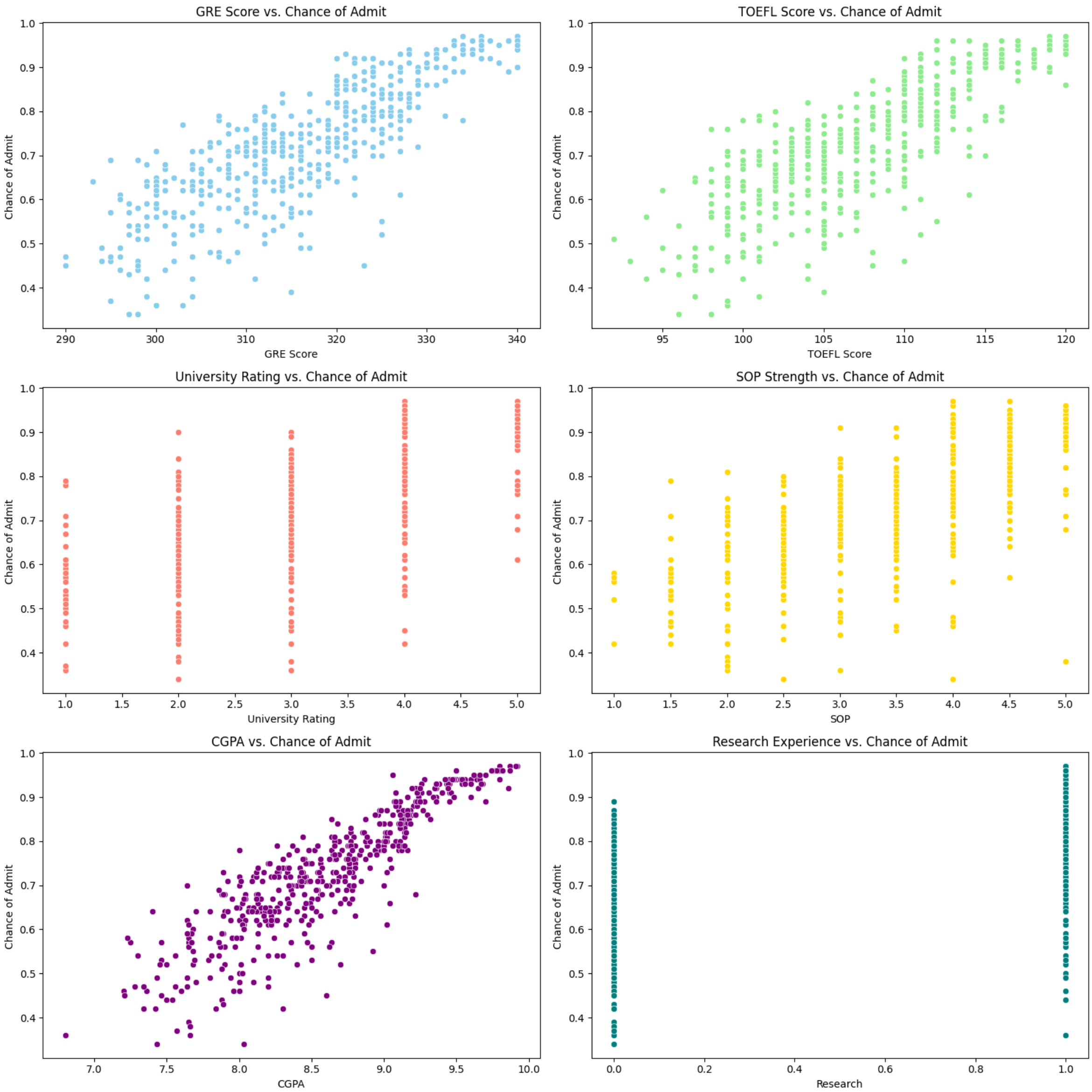
In [62]:
```
# count plot for continous variable
plt.figure(figsize=(8, 6))
sns.countplot(x=df['Research'])
plt.title('Count of Research Experience')
plt.show()
```



In [63]:
```
# Compute the correlation matrix
correlation_matrix = df.corr()

# Plot the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0
plt.title('Correlation Matrix of Admission Factors')
plt.show()
```



In [64]:
```
# Set up scatter plots for bivariate analysis
fig, axes = plt.subplots(3, 2, figsize=(15, 15))

# Plot scatter plots
sns.scatterplot(x=df['GRE Score'], y=df['Chance of Admit '], ax=axes[0, 0
axes[0, 0].set_title('GRE Score vs. Chance of Admit')

sns.scatterplot(x=df['TOEFL Score'], y=df['Chance of Admit '], ax=axes[0,
axes[0, 1].set_title('TOEFL Score vs. Chance of Admit')

sns.scatterplot(x=df['University Rating'], y=df['Chance of Admit '], ax=a
axes[1, 0].set_title('University Rating vs. Chance of Admit')

sns.scatterplot(x=df['SOP'], y=df['Chance of Admit '], ax=axes[1, 1], col
axes[1, 1].set_title('SOP Strength vs. Chance of Admit')

sns.scatterplot(x=df['CGPA'], y=df['Chance of Admit '], ax=axes[2, 0], co
axes[2, 0].set_title('CGPA vs. Chance of Admit')

sns.scatterplot(x=df['Research'], y=df['Chance of Admit '], ax=axes[2, 1]
axes[2, 1].set_title('Research Experience vs. Chance of Admit')

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



In [68]:
```
#Outlier treatment
# Define a function to identify outliers using the IQR method

def find_outliers_iqr(data, column, threshold=1.5):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - threshold * IQR
    upper_bound = Q3 + threshold * IQR
    outliers = data[(data[column] < lower_bound) | (data[column] > upper_
    return outliers

# Identify and treat outliers for GRE Score, TOEFL Score, and CGPA
for col in ['GRE Score', 'TOEFL Score', 'CGPA']:
    outliers = find_outliers_iqr(df, col)
    print(f'Number of outliers in {col}: {len(outliers)}')
```

```
Number of outliers in GRE Score: 0
Number of outliers in TOEFL Score: 0
Number of outliers in CGPA: 0
```

In [69]:
```
#feature Engineering
df.drop('Serial No.', axis=1, inplace=True)
```

In [71]:
```
# Prepare the data for modeling
X = df.drop('Chance of Admit ', axis=1)
y = df['Chance of Admit ']

# Scale the features using StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_siz

# Print the shapes of the training and testing sets
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 [73]:
```
#Linear Regression model
# Build the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Print model coefficients with column names
coefficients = pd.DataFrame({'Feature': X.columns, 'Coefficient': model.c
print('\nModel Coefficients:')
print(coefficients)

# Predict on the test set
y_pred = model.predict(X_test)
print(y_pred)
```

```
Model Coefficients:
             Feature  Coefficient
0          GRE Score     0.027470
1        TOEFL Score     0.018202
2  University Rating     0.002935
3                SOP     0.001796
4               LOR      0.015937
5               CGPA     0.067990
6           Research     0.011927
[0.91457473 0.79518127 0.57265986 0.70736968 0.81588282 0.86206561
 0.47459746 0.64850923 0.82378728 0.80741498 0.72193204 0.72589118
 0.65632227 0.93677168 0.8241518  0.50979177 0.83931942 0.59727295
 0.53339576 0.57155958 0.66548168 0.55305833 0.72232308 0.79506004
 0.78027648 0.60248654 0.94840363 0.84741471 0.62777011 0.74343096
 0.55533035 0.73004034 0.54474225 0.86116288 0.65713016 0.7371816
 0.55423839 0.95718977 0.64364267 0.71057279 0.97036982 0.57495143
 0.67075391 0.85830422 0.94112903 0.57793762 0.9583926  0.83902765
 0.79591651 0.92570648 0.88805969 0.56366238 0.70359711 0.52658929
 0.9536427  0.59746814 0.95600396 0.73916386 0.66256982 0.5012903
 0.62950759 0.68031188 0.59896721 0.59203806 0.44085868 0.58866369
 0.8667547  0.89783006 0.65831807 0.70667392 0.6176818  0.78587721
 0.69152566 0.56271019 0.5542953  0.65084583 0.84627224 0.86373777
 0.53729574 0.63142139 0.76958036 0.84812916 0.61693172 0.8471071
 0.73411583 0.6668525  0.60444455 0.73875671 0.78899999 0.66320147
 0.7428225  0.90802002 0.91576583 0.65056489 0.77694407 0.43563138
 0.68664259 0.78598826 0.73469446 0.64865736]
```

In [102…

```python
#Model Statistics
# Add a constant to the features
X_train_sm = sm.add_constant(X_train)

# Build the OLS model
model_sm = sm.OLS(y_train, X_train_sm)
model_fitted = model_sm.fit()

# Print the model summary
print('\nModel Summary:')
print(model_fitted.summary())
```

```
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:                       2
57.0
Date:                Tue, 18 Feb 2025   Prob (F-statistic):           3.41e
-142
Time:                        17:41:50   Log-Likelihood:                   56
1.91
No. Observations:                 400   AIC:                              -1
108.
Df Residuals:                     392   BIC:                              -1
076.
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const          0.7228      0.003    240.717      0.000       0.717
0.729
x1             0.0275      0.007      4.196      0.000       0.015
0.040
x2             0.0182      0.006      3.174      0.002       0.007
0.029
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             0.0680      0.007     10.444      0.000       0.055
0.081
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):                19
0.099
Skew:                          -1.107   Prob(JB):                       5.25
e-42
Kurtosis:                       5.551   Cond. No.
5.72
==============================================================================
====

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

In [77]:
```
# Ridge Regression
ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)
y_pred_ridge = ridge.predict(X_test)


# Lasso Regression
lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
y_pred_lasso = lasso.predict(X_test)
```

In [103…
```
# Calculate VIF scores
vif = pd.DataFrame()
vif['Feature'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.sha
print('\nVIF Scores:')
print(vif)
```

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

In [110…
```
# Drop variables with VIF > 5 iteratively
X_vif = X.copy()
dropped = True
while dropped:
    vif = pd.DataFrame()
    vif['Feature'] = X_vif.columns
    vif['VIF'] = [variance_inflation_factor(X_vif.values, i) for i in ran
    max_vif = vif['VIF'].max()
    if max_vif > 5:
        drop_feature = vif.loc[vif['VIF'].idxmax(), 'Feature']
        X_vif = X_vif.drop(drop_feature, axis=1)
        print(f'Dropped {drop_feature} due to high VIF')
    else:
        dropped = False

print('\nFeatures after VIF reduction:')
print(X_vif.columns)

# Scale the features using StandardScaler
scaler = StandardScaler()
X_vif_scaled = scaler.fit_transform(X_vif)

# Split the data into training and testing sets
X_train_vif, X_test_vif, y_train_vif, y_test_vif = train_test_split(X_vif

# Build the Linear Regression model
model_vif = LinearRegression()
model_vif.fit(X_train_vif, y_train_vif)

# Predict on the test set
y_pred_vif = model_vif.predict(X_test_vif)
```

```
Dropped GRE Score due to high VIF
Dropped CGPA due to high VIF
Dropped SOP due to high VIF
Dropped LOR  due to high VIF
Dropped University Rating due to high VIF

Features after VIF reduction:
Index(['TOEFL Score', 'Research'], dtype='object')
```

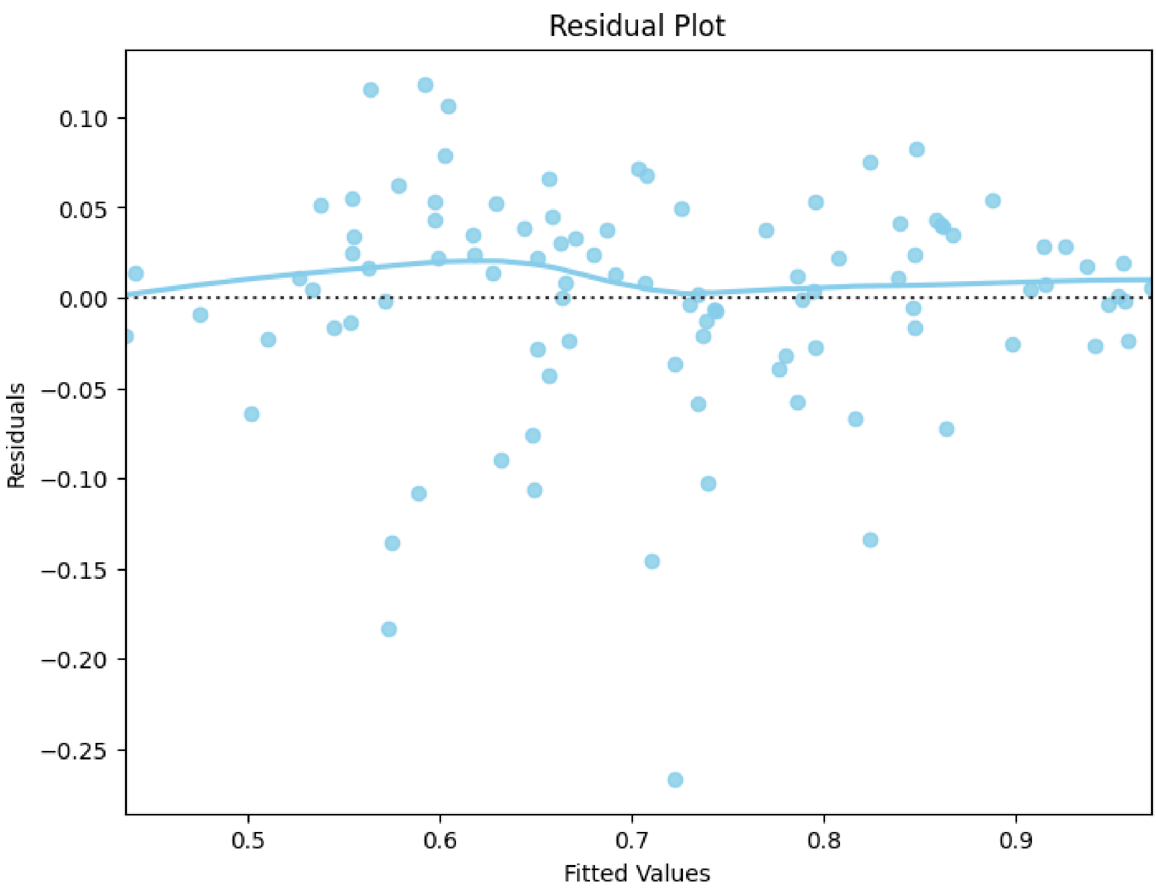
In [112…

```
#Mean of Residuals
# Calculate residuals
residuals = y_test - y_pred

# Check if the mean of residuals is nearly zero
mean_residuals = np.mean(residuals)
print(f'Mean of residuals: {mean_residuals}')
```

```
Mean of residuals: -0.0054536237176613465
```

In [113…

```
#Linearity of variables
# Plot residuals vs. fitted values
plt.figure(figsize=(8, 6))
sns.residplot(x=y_pred, y=residuals, lowess=True, color='skyblue')
plt.title('Residual Plot')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.show()
```



In [115…

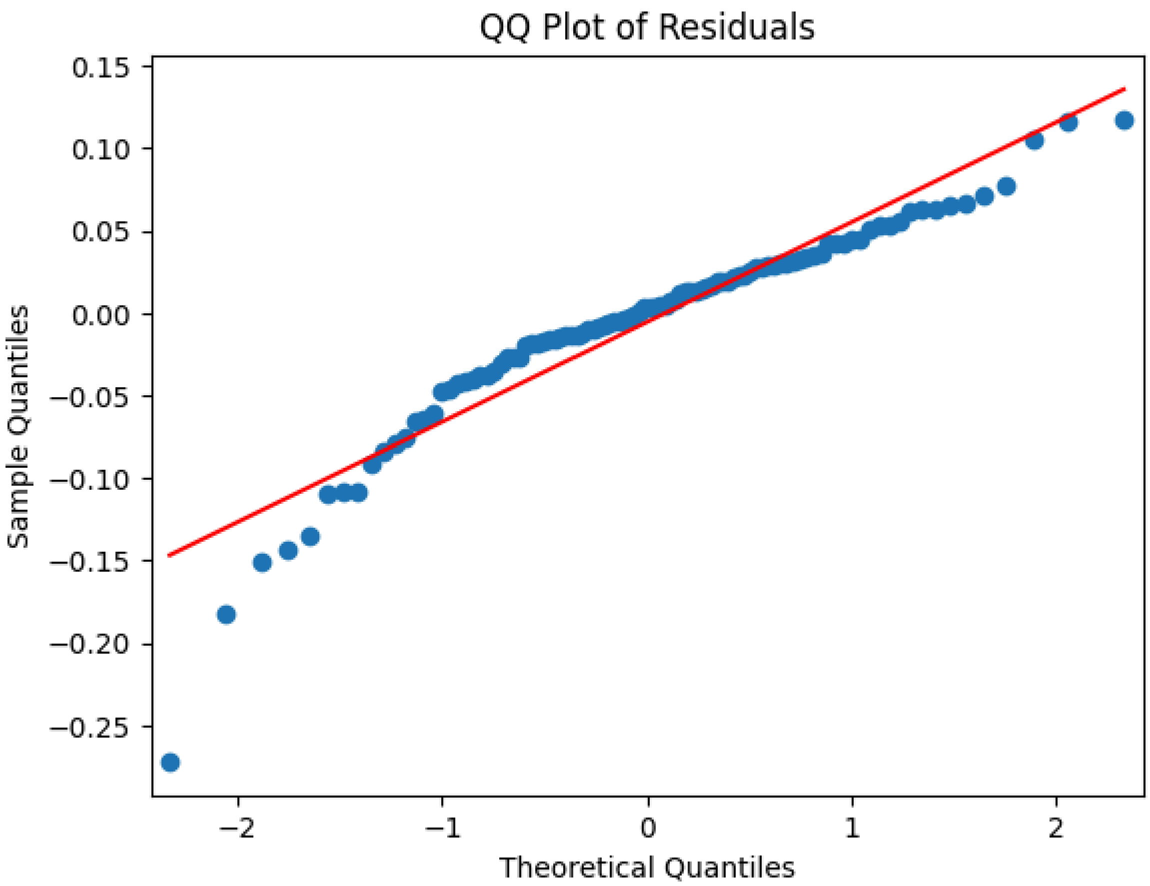
```
# Perform the Goldfeld-Quandt test for homoscedasticity
name = ['F statistic', 'p-value']
test = sms.het_goldfeldquandt(residuals, X_test)
print('Goldfeld-Quandt Test:')
print(lzip(name, test))
```

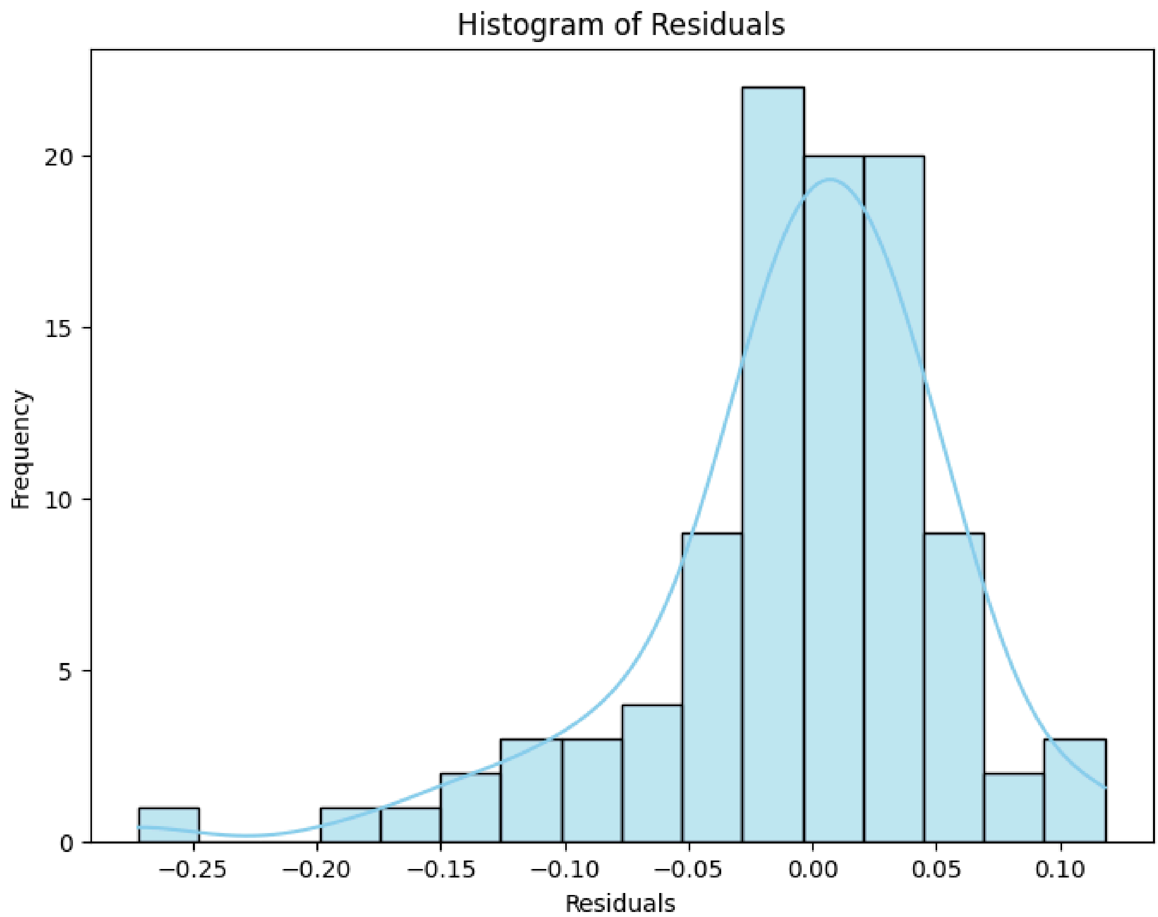
```
Goldfeld-Quandt Test:
[('F statistic', 0.49493874916994285), ('p-value', 0.9883892329316785)]
```

In [116…

```
#normality of residuals
# Shapiro-Wilk Test
shapiro_test = shapiro(residuals)
print(f'\nShapiro-Wilk Test: {shapiro_test}')

# QQ Plot
sm.qqplot(residuals, line='s')
plt.title('QQ Plot of Residuals')
plt.show()

# Histogram of Residuals
plt.figure(figsize=(8, 6))
sns.histplot(residuals, kde=True, color='skyblue')
plt.title('Histogram of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.show()
```

```
Shapiro-Wilk Test: ShapiroResult(statistic=0.9178703251544267, pvalue=1.08
69980466510536e-05)
```





In [118…

```
# Calculate evaluation metrics
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
adj_r2 = 1 - (1 - r2) * (len(y_test) - 1) / (len(y_test) - X_test.shape[1

print(f'MAE: {mae}')
print(f'RMSE: {rmse}')
print(f'R2 Score: {r2}')
print(f'Adjusted R2 Score: {adj_r2}')
```

```
MAE: 0.042722654277053664
RMSE: 0.060865880415783113
R2 Score: 0.8188432567829629
Adjusted R2 Score: 0.8050595915381884
```

In [122…

```
# Calculate evaluation metrics for Ridge Regression
mae_ridge = mean_absolute_error(y_test, y_pred_ridge)
rmse_ridge = np.sqrt(mean_squared_error(y_test, y_pred_ridge))
r2_ridge = r2_score(y_test, y_pred_ridge)
adj_r2_ridge = 1 - (1 - r2_ridge) * (len(y_test) - 1) / (len(y_test) - X_

print(f'Ridge Regression - MAE: {mae_ridge}')
print(f'Ridge Regression - RMSE: {rmse_ridge}')
print(f'Ridge Regression - R2 Score: {r2_ridge}')
print(f'Ridge Regression - Adjusted R2 Score: {adj_r2_ridge}')
```

```
Ridge Regression - MAE: 0.04274636477332955
Ridge Regression - RMSE: 0.06087335867674351
Ridge Regression - R2 Score: 0.8187987385531802
Ridge Regression - Adjusted R2 Score: 0.8050116860517917
```

In [121…

```
# Calculate evaluation metrics for Lasso Regression
mae_lasso = mean_absolute_error(y_test, y_pred_lasso)
rmse_lasso = np.sqrt(mean_squared_error(y_test, y_pred_lasso))
r2_lasso = r2_score(y_test, y_pred_lasso)
adj_r2_lasso = 1 - (1 - r2_lasso) * (len(y_test) - 1) / (len(y_test) - X_

print(f'Lasso Regression - MAE: {mae_lasso}')
print(f'Lasso Regression - RMSE: {rmse_lasso}')
print(f'Lasso Regression - R2 Score: {r2_lasso}')
print(f'Lasso Regression - Adjusted R2 Score: {adj_r2_lasso}')
```

```
Lasso Regression - MAE: 0.09858647394745274
Lasso Regression - RMSE: 0.12296387645200821
Lasso Regression - R2 Score: 0.2606300776476902
Lasso Regression - Adjusted R2 Score: 0.20437367051218847
```

In [124…

```
# Calculate evaluation metrics for VIF reduced model
mae_vif = mean_absolute_error(y_test_vif, y_pred_vif)
rmse_vif = np.sqrt(mean_squared_error(y_test_vif, y_pred_vif))
r2_vif = r2_score(y_test_vif, y_pred_vif)
adj_r2_vif = 1 - (1 - r2_vif) * (len(y_test_vif) - 1) / (len(y_test_vif)

print(f'VIF Reduced Model - MAE: {mae_vif}')
print(f'VIF Reduced Model - RMSE: {rmse_vif}')
print(f'VIF Reduced Model - R2 Score: {r2_vif}')
print(f'VIF Reduced Model - Adjusted R2 Score: {adj_r2_vif}')
```

```
VIF Reduced Model - MAE: 0.0623718521016161
VIF Reduced Model - RMSE: 0.08043761005252416
VIF Reduced Model - R2 Score: 0.6836083564321792
VIF Reduced Model - Adjusted R2 Score: 0.6770848173895437
```

In [ ]:

In [ ]: