Define Problem Statement and perform Exploratory Data Analysis

Context

Jamboree has helped thousands of students like you make it to top colleges abroad. Be it GMAT, GRE or SAT, their unique problem-solving methods ensure maximum scores with minimum effort. They recently launched a feature where students/learners can come to their website and check their probability of getting into the IVY league college. This feature estimates the chances of graduate admission from an Indian perspective.

Problem Statement

Your analysis will help Jamboree in understanding what factors are important in graduate admissions and how these factors are interrelated among themselves. It will also help predict one's chances of admission given the rest of the variables.

► Observations on shape of data, data types of all the attributes, conversion of categorical attributes to 'category' (If required) , missing value detection, statistical summary.

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
df = pd.read_csv("C:\Jamboree_Admission_csv.csv")
df.sample(5)
```

Out[3]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
179	180	307	102	3	3.0	3.0	8.27	0	0.73
122	123	310	106	4	1.5	2.5	8.36	0	0.57
406	407	322	103	4	3.0	2.5	8.02	1	0.61
290	291	307	105	2	2.5	3.0	7.65	0	0.58
373	374	321	109	3	3.0	3.0	8.54	1	0.79

In [4]:
```
df.shape
```

Out[4]: (500, 9)

In [5]:
```
1 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 [6]:
```
1 df.isnull().sum()
```

Out[6]:
```
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 [7]:
```
1 df.describe()
```

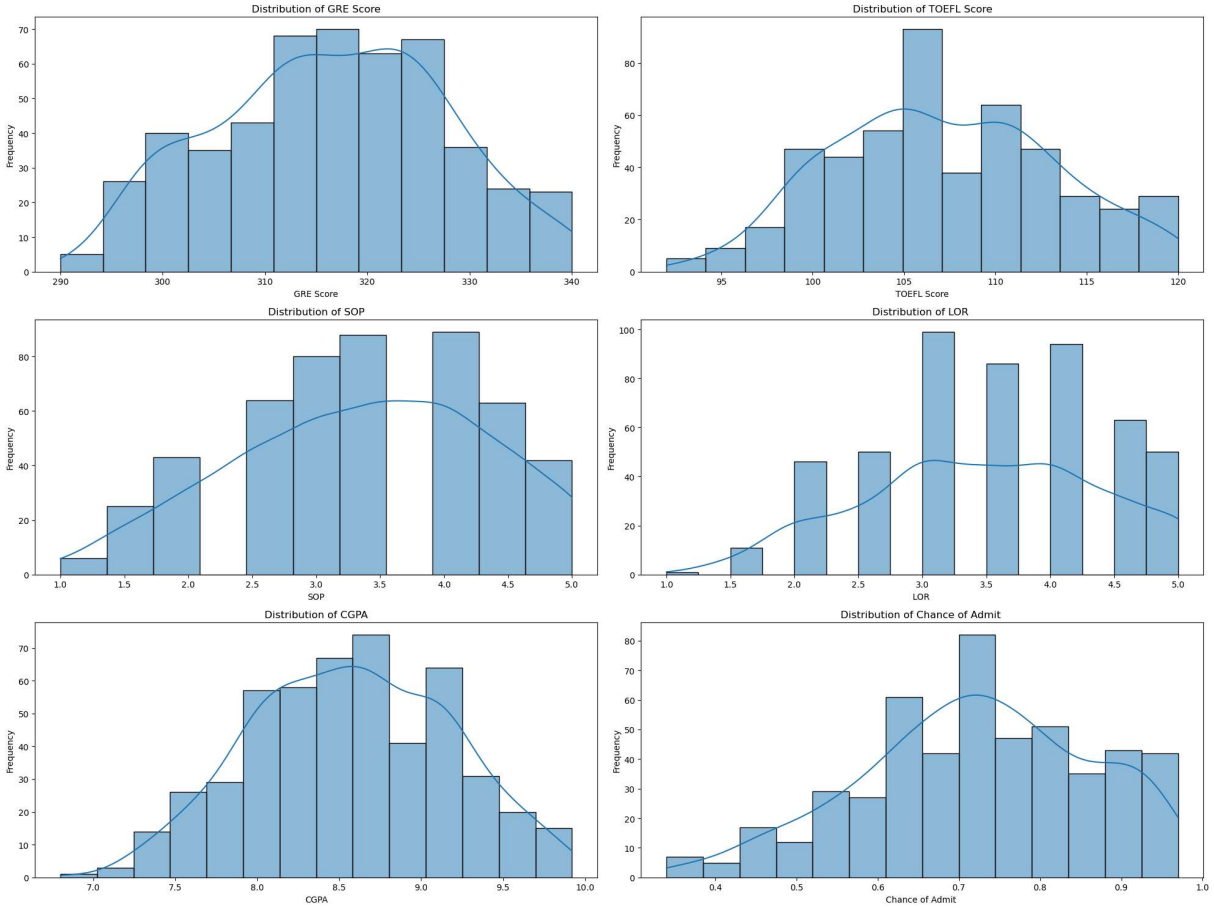
Out[7]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.00000
mean	250.500000	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	0.72174
std	144.481833	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	0.14114
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.34000
25%	125.750000	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.63000
50%	250.500000	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.72000
75%	375.250000	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.82000
max	500.000000	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.97000

► Univariate Analysis (distribution plots of all the continuous variable(s) barplots/countplots of all the categorical variables)

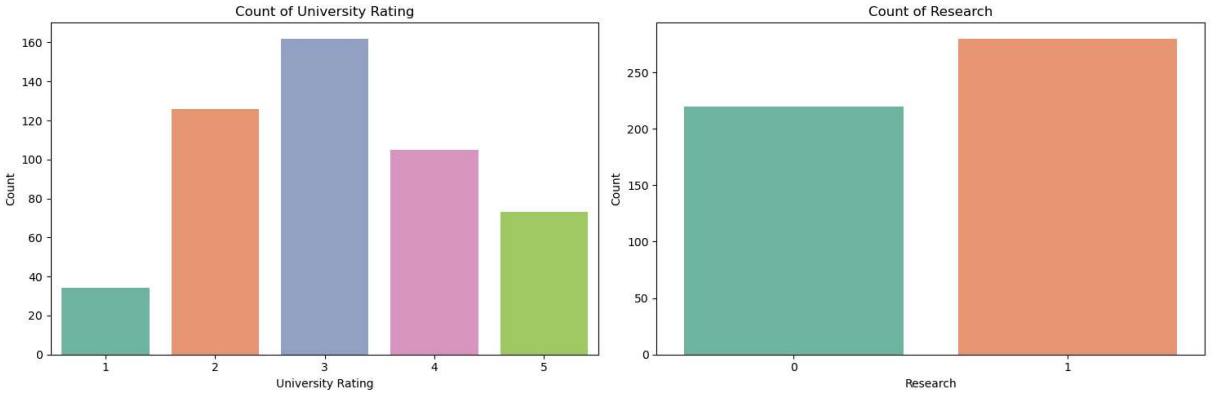
In [8]:

```
# Continuous Variables
continuous_vars = ['GRE Score', 'TOEFL Score', 'SOP', 'LOR ', 'CGPA', 'Chance of Admit '

# Create distribution plots for continuous variables
plt.figure(figsize=(20, 15))
for i, var in enumerate(continuous_vars, 1):
    plt.subplot(3, 2, i)
    sns.histplot(df[var], kde=True)
    plt.title(f'Distribution of {var}')
    plt.xlabel(var)
    plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```



In [9]:

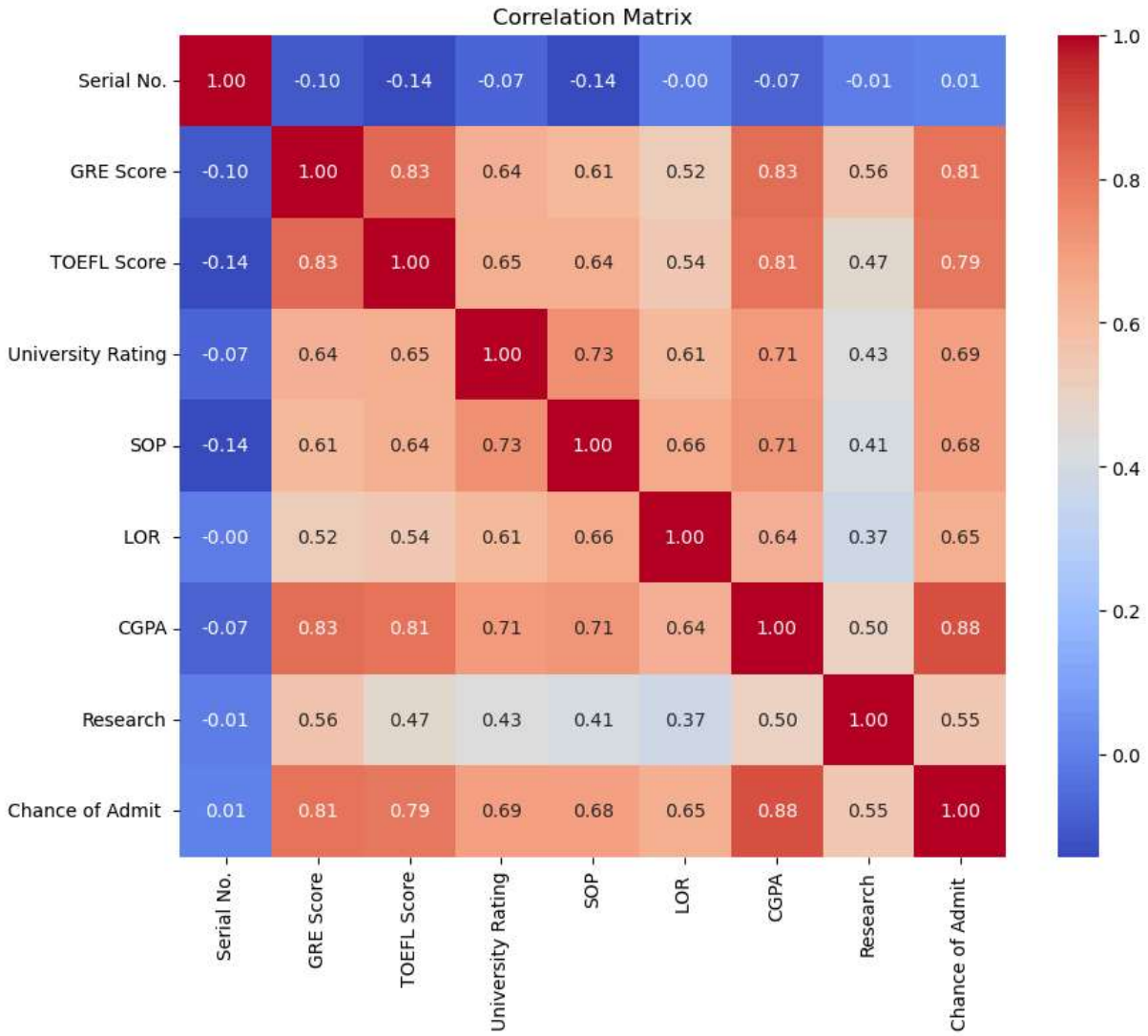
```
# Create count plots for categorical variables
categorical_vars = ['University Rating', 'Research']

plt.figure(figsize=(15, 5))
for i, var in enumerate(categorical_vars, 1):
    plt.subplot(1, 2, i)
    sns.countplot(x=var, data=df, palette='Set2')
    plt.title(f'Count of {var}')
    plt.xlabel(var)
    plt.ylabel('Count')
plt.tight_layout()
plt.show()
```



► Bivariate Analysis (Relationships between important variables

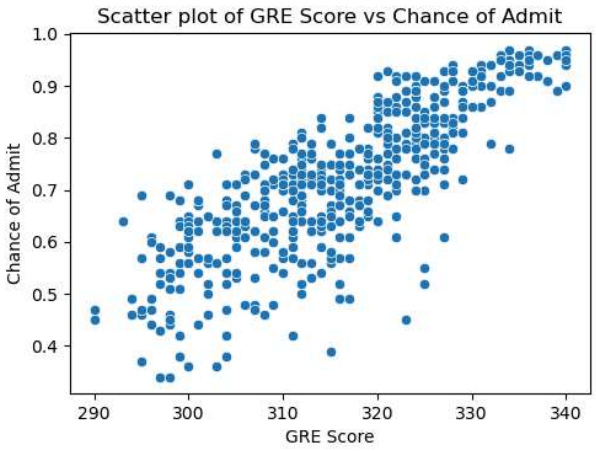
In [10]:

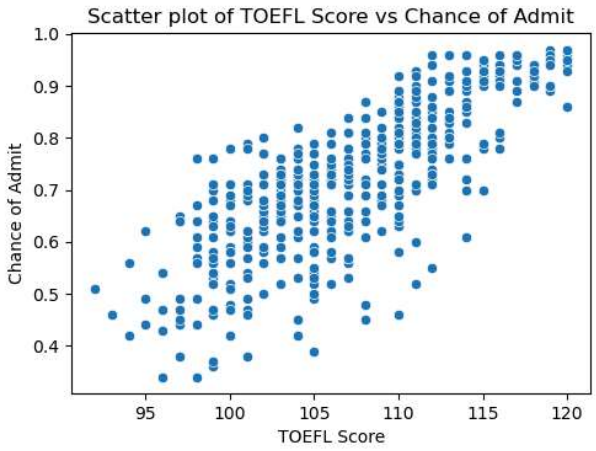
```
correlation_matrix = df.corr()

# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

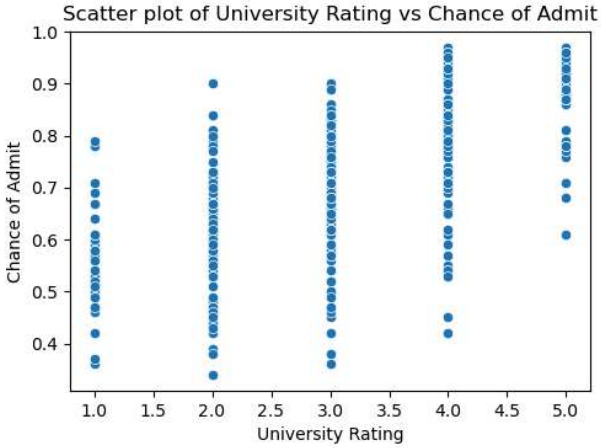


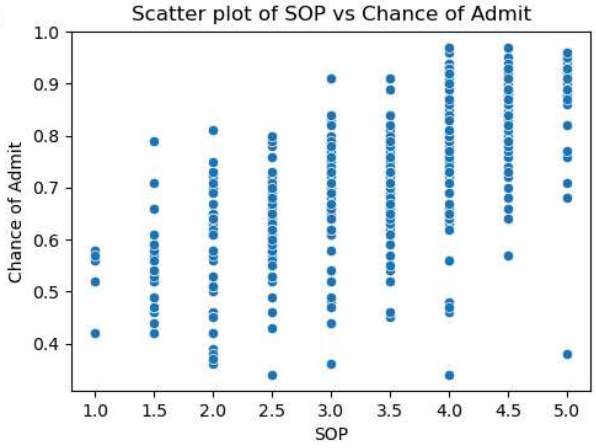
In [65]:

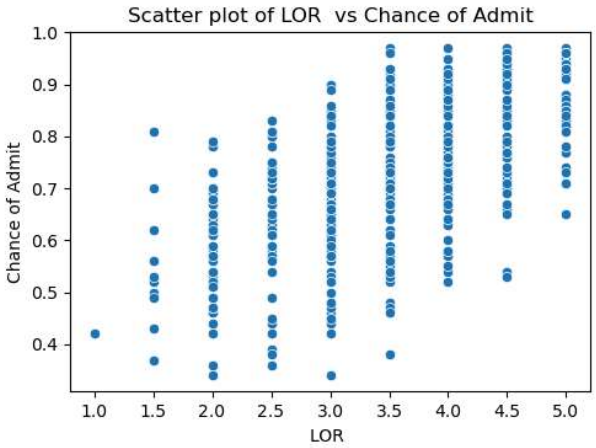
```
independent_col = ['GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research']

plt.figure(figsize=(10,15))
for i, col in enumerate(independent_col,1):
    plt.subplot(4,2,i)
    sns.scatterplot(x=col,y='Chance of Admit ',data=df)
    plt.title(f'Scatter plot of {col} vs Chance of Admit')
    plt.xlabel(col)
    plt.ylabel('Chance of Admit')

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

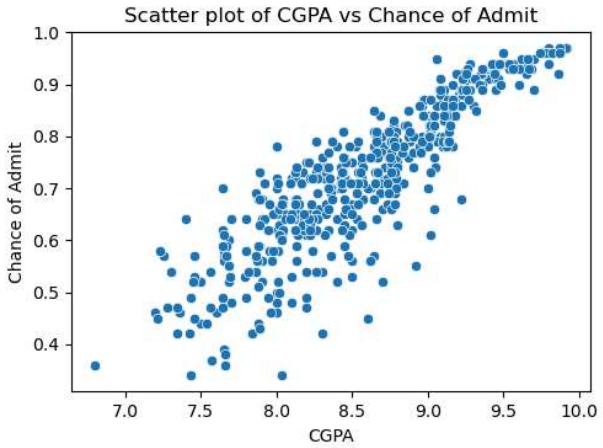

Scatter plot of GRE Score vs Chance of Admit
Chance of Admit
GRE Score

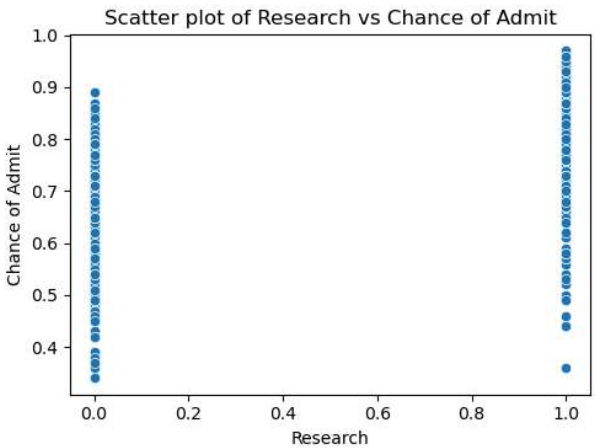

Scatter plot of TOEFL Score vs Chance of Admit
Chance of Admit
TOEFL Score


Scatter plot of University Rating vs Chance of Admit
Chance of Admit
University Rating

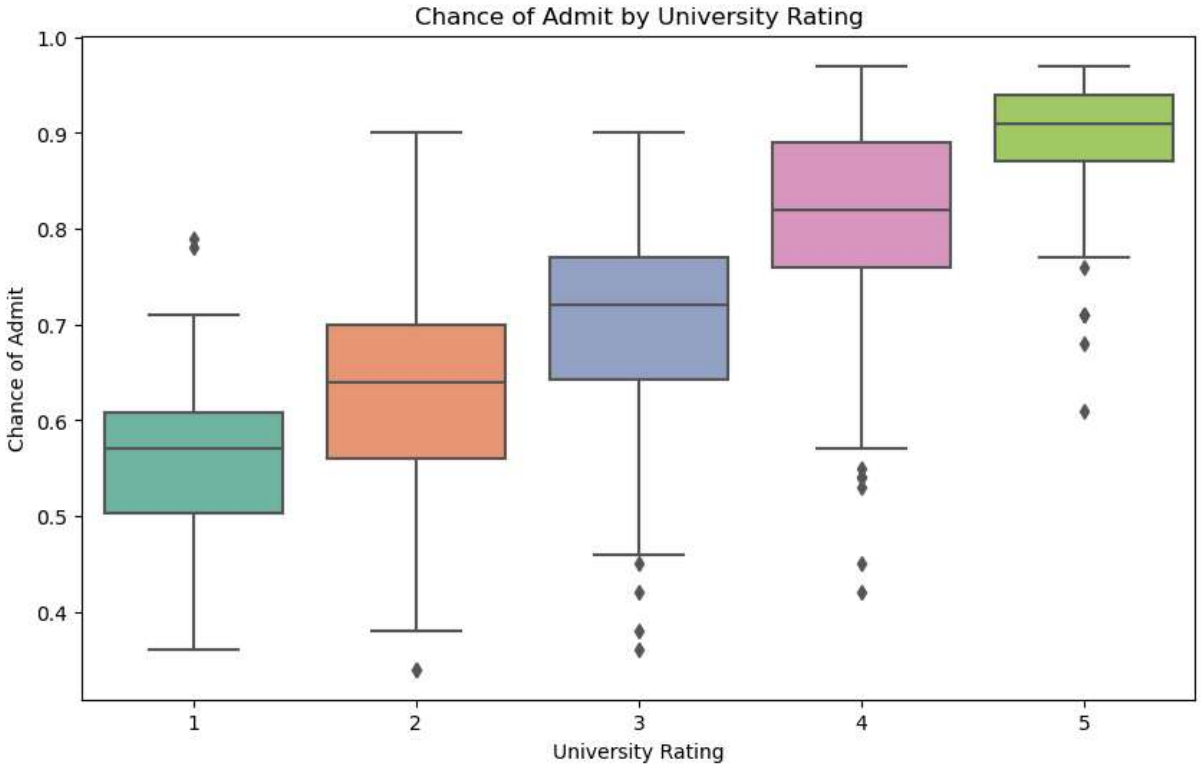

Scatter plot of SOP vs Chance of Admit
Chance of Admit
SOP


Scatter plot of LOR vs Chance of Admit
Chance of Admit
LOR


Scatter plot of CGPA vs Chance of Admit
Chance of Admit
CGPA


Scatter plot of Research vs Chance of Admit
Chance of Admit
Research

In [12]:
```
# Box plot of Chance of Admit by University Rating
plt.figure(figsize=(10, 6))
sns.boxplot(x='University Rating', y='Chance of Admit ', data=df, palette='Set2')
plt.title('Chance of Admit by University Rating')
plt.xlabel('University Rating')
plt.ylabel('Chance of Admit')
plt.show()
```



► Illustrate the insights based on EDA

Comments on range of attributes, outliers of various attributes . Comments on the distribution of the variables and relationship between them . Comments for each univariate and bivariate plots

GRE Score: Indicating that higher GRE scores correlate with a higher chance of admission.

TOEFL Score: Positively impacts admission chances, suggesting proficiency in English is important for applicants.

CGPA: The past academic performance is a strong predictor of future success.

LOR (Letter of Recommendation): Suggesting that strong endorsements positively influence admission chances.

Research: Significant : Indicating that research experience adds value to the application.

University Rating : Indicating it may not have a strong impact on admission chances.

SOP (Statement of Purpose): Also not significant, suggesting that the quality of the SOP might not be as impactful as other factors.

Data Preprocessing

► Duplicate value check , Missing value treatment , Outlier treatment , Feature engineering , Data preparation for modeling .

In [14]:
```
1 df.duplicated().sum()
```

Out[14]: 0

In [15]:
```
1 df.isnull().sum()
```

Out[15]:
```
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
```

Model building

► Build the Linear Regression model and comment on the model statistics

In [16]:
```
1 from sklearn.model_selection import train_test_split
2
3 X = df.drop(['Chance of Admit '], axis=1)
4 y = df['Chance of Admit ']
```

In [17]:
```
1 import statsmodels.api as sm
2 from sklearn.preprocessing import StandardScaler
3
4 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
```

In [18]:
```
1 X_train
```

Out[18]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
249	250	321	111	3	3.5	4.0	8.83	1
433	434	316	111	4	4.0	5.0	8.54	0
19	20	303	102	3	3.5	3.0	8.50	0
322	323	314	107	2	2.5	4.0	8.27	0
332	333	308	106	3	3.5	2.5	8.21	1
...	...	...	...	...	...	...	...	...
106	107	329	111	4	4.5	4.5	9.18	1
270	271	306	105	2	2.5	3.0	8.22	1
348	349	302	99	1	2.0	2.0	7.25	0
435	436	309	105	2	2.5	4.0	7.68	0
102	103	314	106	2	4.0	3.5	8.25	0

400 rows × 8 columns

In [19]:
```
1 y_train
```

Out[19]:
```
249    0.77
433    0.71
19     0.62
322    0.72
332    0.75
       ...
106    0.87
270    0.72
348    0.57
435    0.55
102    0.62
Name: Chance of Admit , Length: 400, dtype: float64
```

In [20]:
```
1 #Standardization
2 from sklearn.preprocessing import StandardScaler
3
4 X_train_columns=X_train.columns
5 std=StandardScaler()
6 X_train_std=std.fit_transform(X_train)
7 X_train_std
```

Out[20]:
```
array([[ 0.00692641,  0.38998634,  0.6024183 , ...,  0.56498381,
         0.4150183 ,  0.89543386],
       [ 1.31742529, -0.06640493,  0.6024183 , ...,  1.65149114,
        -0.06785154, -1.11677706],
       [-1.63119718, -1.25302222, -0.87691722, ..., -0.52152352,
        -0.13445427, -1.11677706],
       ...,
       [ 0.71203178, -1.34430047, -1.37002906, ..., -1.60803084,
        -2.2157898 , -1.11677706],
       [ 1.33166984, -0.7053527 , -0.38380538, ...,  0.56498381,
        -1.49981038, -1.11677706],
       [-1.04004823, -0.24896144, -0.21943477, ...,  0.02173015,
        -0.55072138, -1.11677706]])
```

In [21]:
```
1 X_train = pd.DataFrame(X_train_std, columns=X_train_columns)
2 X_train
```

Out[21]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	0.006926	0.389986	0.602418	-0.098298	0.126796	0.564984	0.415018	0.895434
1	1.317425	-0.066405	0.602418	0.775459	0.633979	1.651491	-0.067852	-1.116777
2	-1.631197	-1.253022	-0.876917	-0.098298	0.126796	-0.521524	-0.134454	-1.116777
3	0.526853	-0.248961	-0.055064	-0.972054	-0.887570	0.564984	-0.517420	-1.116777
4	0.598075	-0.796631	-0.219435	-0.098298	0.126796	-1.064777	-0.617324	0.895434
...	...	...	...	...	...	...	...	...
395	-1.011559	1.120212	0.602418	0.775459	1.141162	1.108237	0.997792	0.895434
396	0.156494	-0.979187	-0.383805	-0.972054	-0.887570	-0.521524	-0.600673	0.895434
397	0.712032	-1.344300	-1.370029	-1.845810	-1.394754	-1.608031	-2.215790	-1.116777
398	1.331670	-0.705353	-0.383805	-0.972054	-0.887570	0.564984	-1.499810	-1.116777
399	-1.040048	-0.248961	-0.219435	-0.972054	0.633979	0.021730	-0.550721	-1.116777

400 rows × 8 columns

In [23]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

X = df.drop(columns=['Chance of Admit ', 'Serial No.'])
y = df['Chance of Admit ']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42

lr_model = LinearRegression()

lr_model.fit(X_train, y_train)

y_pred = lr_model.predict(X_test)

# Model Evaluation
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5
r2 = r2_score(y_test, y_pred)

print("Model Evaluation Metrics:")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2 Score): {r2}")

```

```
Model Evaluation Metrics:
Mean Absolute Error (MAE): 0.042722654277053636
Mean Squared Error (MSE): 0.00370465539878841
Root Mean Squared Error (RMSE): 0.060865880415783113
R-squared (R2 Score): 0.8188432567829629
```

▶ Display model coefficients with column names

In [24]:
```
# To also see the coefficients
print("\nCoefficients of the Linear Regression model:")
for col, coef in zip(X.columns, lr_model.coef_):
    print(f"{col}: {coef}")

# Intercept of the model
print(f"\nIntercept: {lr_model.intercept_}")
```

```
Coefficients of the Linear Regression model:
GRE Score: 0.0024344383948367533
TOEFL Score: 0.0029958733715185724
University Rating: 0.00256879774352657
SOP: 0.0018136900128124102
LOR : 0.017237983661425423
CGPA: 0.11252708444059892
Research: 0.024026787646206117

Intercept: -1.421447071901646
```

▶ Try out Ridge and Lasso regression

In [25]:

```
from sklearn.linear_model import Ridge, Lasso

ridge_model = Ridge(alpha=1.0)
lasso_model = Lasso(alpha=0.01)

# Train the ridge model
ridge_model.fit(X_train, y_train)
y_pred_ridge = ridge_model.predict(X_test)

# Train the Lasso model
lasso_model.fit(X_train, y_train)
y_pred_lasso = lasso_model.predict(X_test)

# Evaluate Ridge model
mae_ridge = mean_absolute_error(y_test, y_pred_ridge)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
rmse_ridge = mse_ridge ** 0.5
r2_ridge = r2_score(y_test, y_pred_ridge)

# Evaluate Lasso model
mae_lasso = mean_absolute_error(y_test, y_pred_lasso)
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
rmse_lasso = mse_lasso ** 0.5
r2_lasso = r2_score(y_test, y_pred_lasso)

# Print Ridge Regression results
print("Ridge Regression Evaluation Metrics:")
print(f"Mean Absolute Error (MAE): {mae_ridge}")
print(f"Mean Squared Error (MSE): {mse_ridge}")
print(f"Root Mean Squared Error (RMSE): {rmse_ridge}")
print(f"R-squared (R2 Score): {r2_ridge}")

# Print Lasso Regression results
print("\nLasso Regression Evaluation Metrics:")
print(f"Mean Absolute Error (MAE): {mae_lasso}")
print(f"Mean Squared Error (MSE): {mse_lasso}")
print(f"Root Mean Squared Error (RMSE): {rmse_lasso}")
print(f"R-squared (R2 Score): {r2_lasso}")
```

```
Ridge Regression Evaluation Metrics:
Mean Absolute Error (MAE): 0.04287834886976052
Mean Squared Error (MSE): 0.0037223223199539745
Root Mean Squared Error (RMSE): 0.06101083772539084
R-squared (R2 Score): 0.8179793486575073

Lasso Regression Evaluation Metrics:
Mean Absolute Error (MAE): 0.05402131794750673
Mean Squared Error (MSE): 0.0055803893977300345
Root Mean Squared Error (RMSE): 0.07470200397399011
R-squared (R2 Score): 0.7271203228493871
```

Testing the assumptions of the linear regression model

► Multicollinearity check by VIF score (variables are dropped one-by-one till none has VIF>5)

In [26]:

```
import statsmodels.api as sm

X_train = sm.add_constant(X_train)
model = sm.OLS(y_train.values, X_train).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.821
Model:                            OLS   Adj. R-squared:                  0.818
Method:                 Least Squares   F-statistic:                     257.0
Date:                Sun, 13 Oct 2024   Prob (F-statistic):          3.41e-142
Time:                        09:56:17   Log-Likelihood:                561.91
No. Observations:                 400   AIC:                            -1108.
Df Residuals:                     392   BIC:                            -1076.
Df Model:                           7
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
const               -1.4214      0.123    -11.549      0.000      -1.663      -1.179
GRE Score            0.0024      0.001      4.196      0.000       0.001       0.004
TOEFL Score          0.0030      0.001      3.174      0.002       0.001       0.005
University Rating    0.0026      0.004      0.611      0.541      -0.006       0.011
SOP                  0.0018      0.005      0.357      0.721      -0.008       0.012
LOR                  0.0172      0.005      3.761      0.000       0.008       0.026
CGPA                 0.1125      0.011     10.444      0.000       0.091       0.134
Research             0.0240      0.007      3.231      0.001       0.009       0.039
==============================================================================
Omnibus:                       86.232   Durbin-Watson:                   2.050
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              190.099
Skew:                          -1.107   Prob(JB):                     5.25e-42
Kurtosis:                       5.551   Cond. No.                     1.37e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.37e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Note :- If any variable has a VIF greater than 5, it is removed from the dataset. The VIF values are recalculated for the remaining variables. This process repeats until no variable has a VIF > 5.

In [27]:

```
X_train_new=X_train.drop(columns='SOP')


model1 = sm.OLS(y_train.values, X_train_new).fit()
print(model1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.821
Model:                            OLS   Adj. R-squared:                  0.818
Method:                 Least Squares   F-statistic:                     300.4
Date:                Sun, 13 Oct 2024   Prob (F-statistic):          2.01e-143
Time:                        09:58:01   Log-Likelihood:                561.85
No. Observations:                 400   AIC:                            -1110.
Df Residuals:                     393   BIC:                            -1082.
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const             -1.4272      0.122    -11.708      0.000      -1.667      -1.188
GRE Score          0.0024      0.001      4.192      0.000       0.001       0.004
TOEFL Score        0.0030      0.001      3.240      0.001       0.001       0.005
University Rating  0.0031      0.004      0.779      0.437      -0.005       0.011
LOR                0.0177      0.004      4.056      0.000       0.009       0.026
CGPA               0.1133      0.011     10.730      0.000       0.093       0.134
Research           0.0241      0.007      3.240      0.001       0.009       0.039
==============================================================================
Omnibus:                       85.621   Durbin-Watson:                   2.047
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              188.163
Skew:                          -1.101   Prob(JB):                     1.38e-41
Kurtosis:                       5.539   Cond. No.                     1.36e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.36e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [28]:
```
X_train_new=X_train_new.drop(columns='University Rating')


model1 = sm.OLS(y_train.values, X_train_new).fit()
print(model1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.821
Model:                            OLS   Adj. R-squared:                  0.818
Method:                 Least Squares   F-statistic:                     360.8
Date:                Sun, 13 Oct 2024   Prob (F-statistic):          1.36e-144
Time:                        10:22:48   Log-Likelihood:                561.54
No. Observations:                 400   AIC:                            -1111.
Df Residuals:                     394   BIC:                            -1087.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          -1.4555      0.116    -12.517      0.000      -1.684      -1.227
GRE Score       0.0025      0.001      4.245      0.000       0.001       0.004
TOEFL Score     0.0031      0.001      3.391      0.001       0.001       0.005
LOR             0.0187      0.004      4.465      0.000       0.010       0.027
CGPA            0.1150      0.010     11.147      0.000       0.095       0.135
Research        0.0246      0.007      3.328      0.001       0.010       0.039
==============================================================================
Omnibus:                       84.831   Durbin-Watson:                   2.053
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              185.096
Skew:                          -1.094   Prob(JB):                     6.41e-41
Kurtosis:                       5.514   Cond. No.                     1.30e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.3e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

note :- This iterative process helps ensure that the remaining variables in the model have low multicollinearity (VIF ≤ 5). Since we elementaed all the columns with VIF > 5.

► The mean of residuals is nearly zero (10 Points)

In [34]:

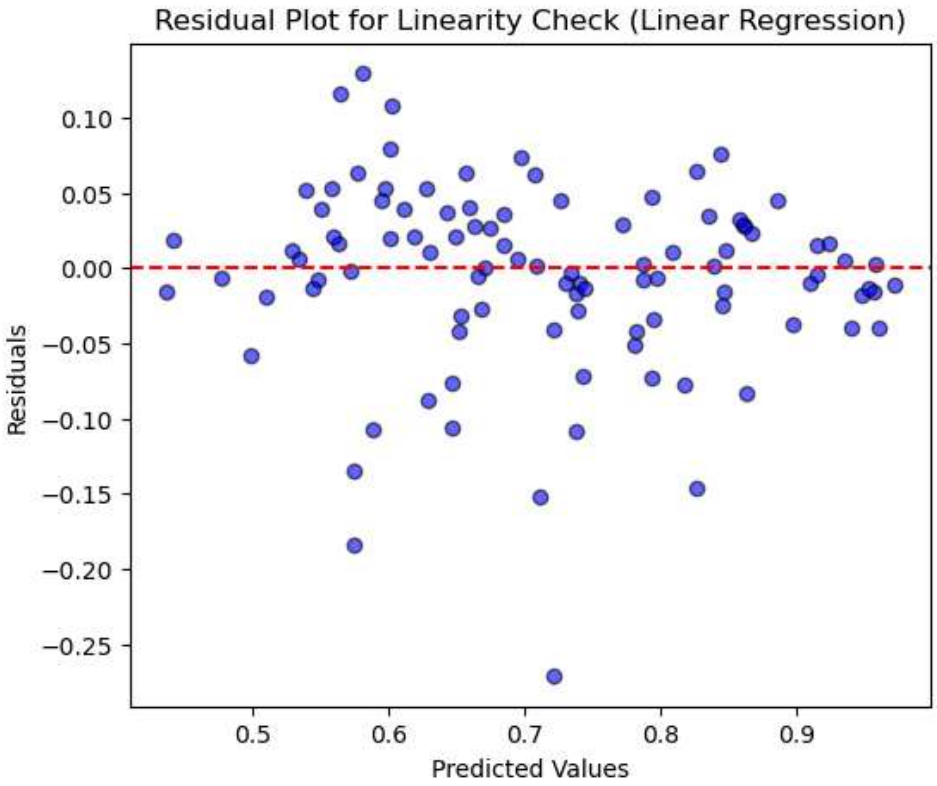
```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error

# Ensure the same columns are used for training and testing
X_train_new = X_train.drop(['SOP', 'University Rating'], axis=1)
X_test_new = X_test.drop(['SOP', 'University Rating'], axis=1)

X_train_new = sm.add_constant(X_train_new)
X_test_new = sm.add_constant(X_test_new)

# Re-fit the models to training data with consistent features
linear_model.fit(X_train_new, y_train)
ridge_model.fit(X_train_new, y_train)
lasso_model.fit(X_train_new, y_train)

# Predict on the updated test data
y_pred_linear = linear_model.predict(X_test_new)
y_pred_ridge = ridge_model.predict(X_test_new)
y_pred_lasso = lasso_model.predict(X_test_new)

# Calculate residuals for each model
residuals_linear = y_test - y_pred_linear
residuals_ridge = y_test - y_pred_ridge
residuals_lasso = y_test - y_pred_lasso

# Calculate mean of residuals for each model
mean_residuals_linear = np.mean(residuals_linear)
mean_residuals_ridge = np.mean(residuals_ridge)
mean_residuals_lasso = np.mean(residuals_lasso)

# Print the mean of residuals for each model
print(f"Mean of residuals (Linear Regression): {mean_residuals_linear}")
print(f"Mean of residuals (Ridge Regression): {mean_residuals_ridge}")
print(f"Mean of residuals (Lasso Regression): {mean_residuals_lasso}")

# Calculate RMSE for each model
rmse_linear = np.sqrt(mean_squared_error(y_test, y_pred_linear))
rmse_ridge = np.sqrt(mean_squared_error(y_test, y_pred_ridge))
rmse_lasso = np.sqrt(mean_squared_error(y_test, y_pred_lasso))
print()

print(f"RMSE (Linear Regression): {rmse_linear}")
print(f"RMSE (Ridge Regression): {rmse_ridge}")
print(f"RMSE (Lasso Regression): {rmse_lasso}")
```

```
Mean of residuals (Linear Regression): -0.0053059479423483435
Mean of residuals (Ridge Regression): -0.005191662831442836
Mean of residuals (Lasso Regression): -0.00020658690848106452

RMSE (Linear Regression): 0.06142491974041873
RMSE (Ridge Regression): 0.061623608610484375
RMSE (Lasso Regression): 0.07529715710456189
```

► Linearity of variables (no pattern in the residual plot)

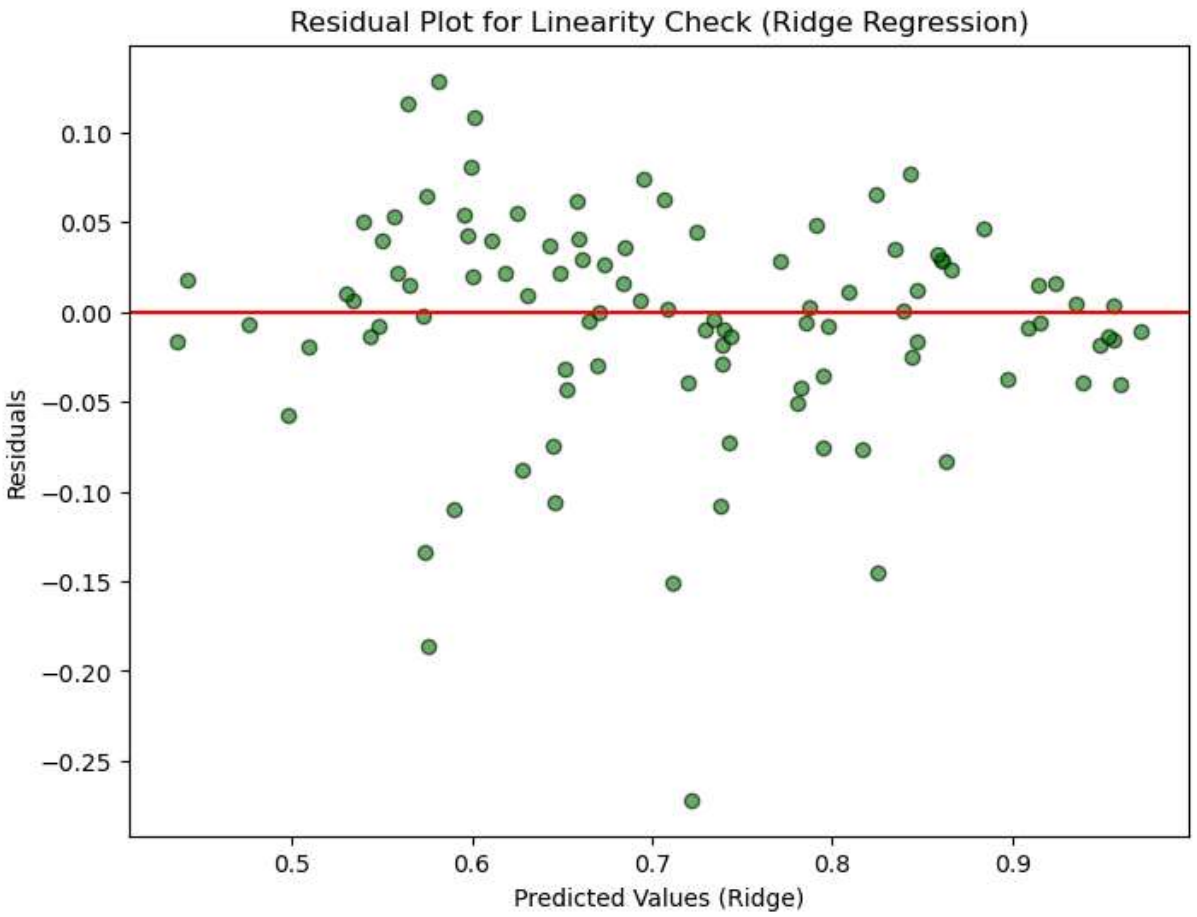
In [84]:

```
# Calculate residuals for Linear Regression
residuals_linear = y_test - y_pred_linear

# Plot residuals
plt.figure(figsize=(6, 5))
plt.scatter(y_pred_linear, residuals_linear, color='blue', edgecolor='k', alpha=0.6)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residual Plot for Linearity Check (Linear Regression)')
plt.show()
```

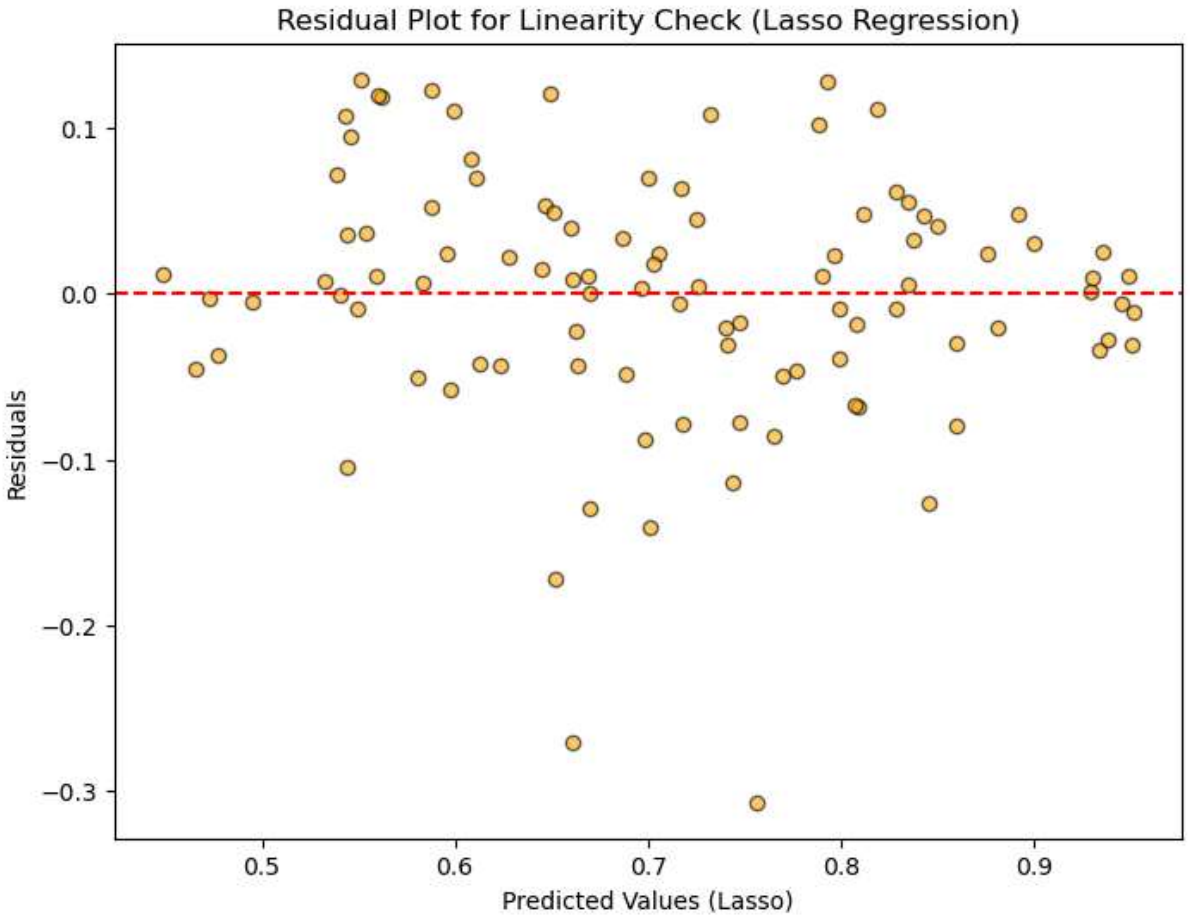


In [89]:

```
# Residual plot for Ridge Regression
residuals_ridge = y_test - y_pred_ridge
plt.figure(figsize=(8, 6))
plt.scatter(y_pred_ridge, residuals_ridge, color='green', edgecolor='k', alpha=0.6)
plt.axhline(y=0, color='r', linestyle='-')
plt.xlabel('Predicted Values (Ridge)')
plt.ylabel('Residuals')
plt.title('Residual Plot for Linearity Check (Ridge Regression)')
plt.show()
```



In [87]:
```
# Residual plot for Lasso Regression
residuals_lasso = y_test - y_pred_lasso
plt.figure(figsize=(8, 6))
plt.scatter(y_pred_lasso, residuals_lasso, color='orange', edgecolor='k', alpha=0.6)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel('Predicted Values (Lasso)')
plt.ylabel('Residuals')
plt.title('Residual Plot for Linearity Check (Lasso Regression)')
plt.show()
```



We can clearly see that points are randomly scattered around the horizontal line at 0 , the linearity assumption holds.

In [90]:
```
import statsmodels.api as sm
from statsmodels.stats.diagnostic import het_breuschpagan

#Perform Breusch-Pagan test for homoscedasticity
_, pval, __, f_pval = het_breuschpagan(residuals_linear, sm.add_constant(X_test))

print(f"Breusch-Pagan test p-value for residuals_linear: {pval}")
```

```
Breusch-Pagan test p-value for residuals_linear: 0.27678666848656575
```

In [91]:
```
import statsmodels.api as sm
from statsmodels.stats.diagnostic import het_breuschpagan

#Perform Breusch-Pagan test for homoscedasticity
_, pval, __, f_pval = het_breuschpagan(residuals_ridge, sm.add_constant(X_test))

#Print p-value result
print(f"Breusch-Pagan test p-value residuals_ridge: {pval}")
```

```
Breusch-Pagan test p-value residuals_ridge: 0.27186243137735644
```

In [92]:
```
import statsmodels.api as sm
from statsmodels.stats.diagnostic import het_breuschpagan

#Perform Breusch-Pagan test for homoscedasticity
_, pval, __, f_pval = het_breuschpagan(residuals_lasso, sm.add_constant(X_test))

#Print p-value result
print(f"Breusch-Pagan test p-value residuals_lasso: {pval}")
```

```
Breusch-Pagan test p-value residuals_lasso: 0.25076969438695207
```

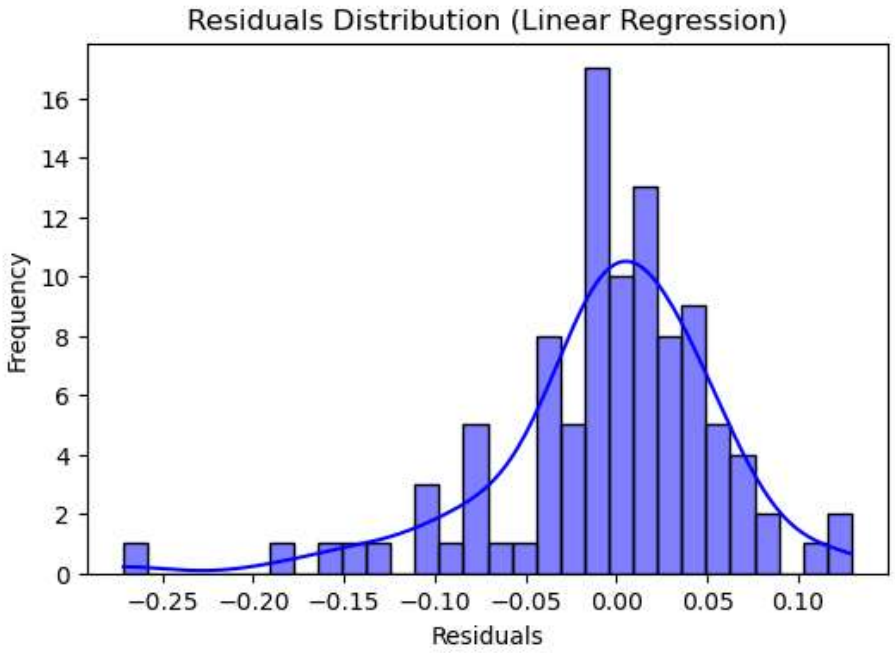
If the p-value is small (typically < 0.05), it suggests that heteroscedasticity is present (violation of the assumption). but in our all 3 cases p-value > 0.05 which means heteroscedasticity is not present.

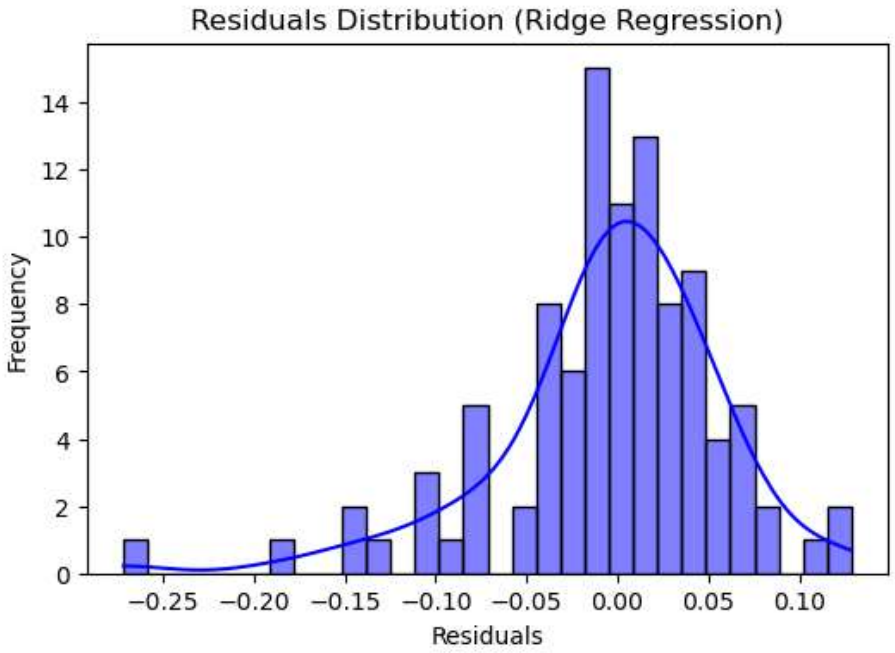
► Normality of residuals (almost bell-shaped curve in residuals distribution, points in QQ plot are almost all on the line)

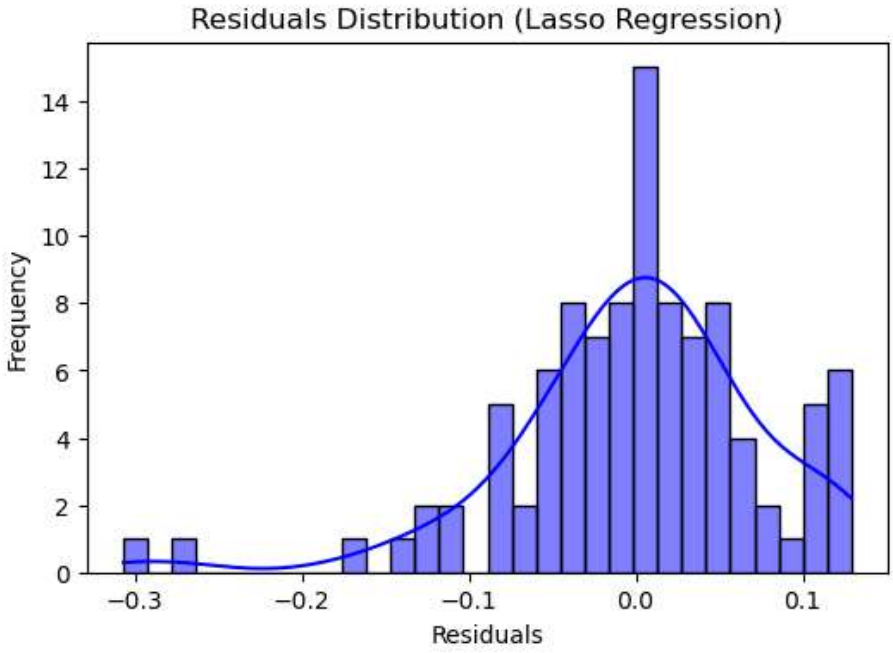
To check the normality of residuals, there are two common methods: plotting the distribution of the residuals and generating a Q-Q plot (Quantile-Quantile plot).

In [95]:

```
# List of residuals
residuals_of_models = [residuals_linear, residuals_ridge, residuals_lasso]
model_names = ['Linear Regression', 'Ridge Regression', 'Lasso Regression']

for i, residuals in enumerate(residuals_of_models):
    plt.figure(figsize=(6, 4))
    sns.histplot(residuals, kde=True, color='blue', bins=30)
    plt.title(f'Residuals Distribution ({model_names[i]})')
    plt.xlabel('Residuals')
    plt.ylabel('Frequency')
    plt.show()
```



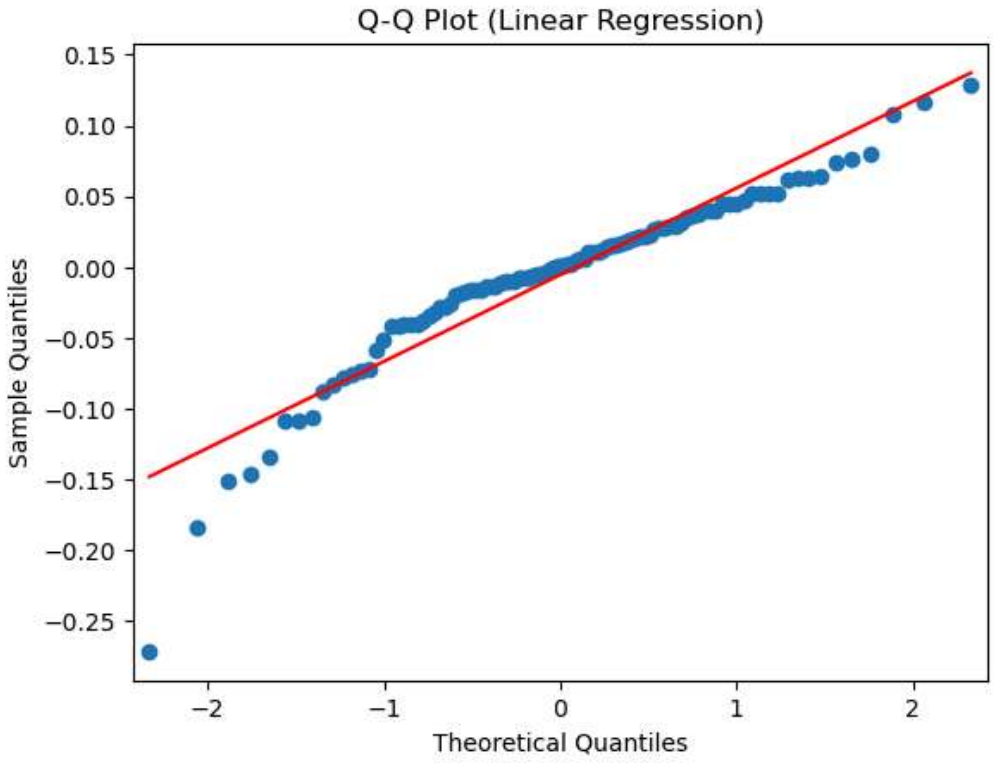



Residuals Distribution (Lasso Regression)
Frequency
Residuals
0
2
4
6
8
10
12
14
−0.3
−0.2
−0.1
0.0
0.1

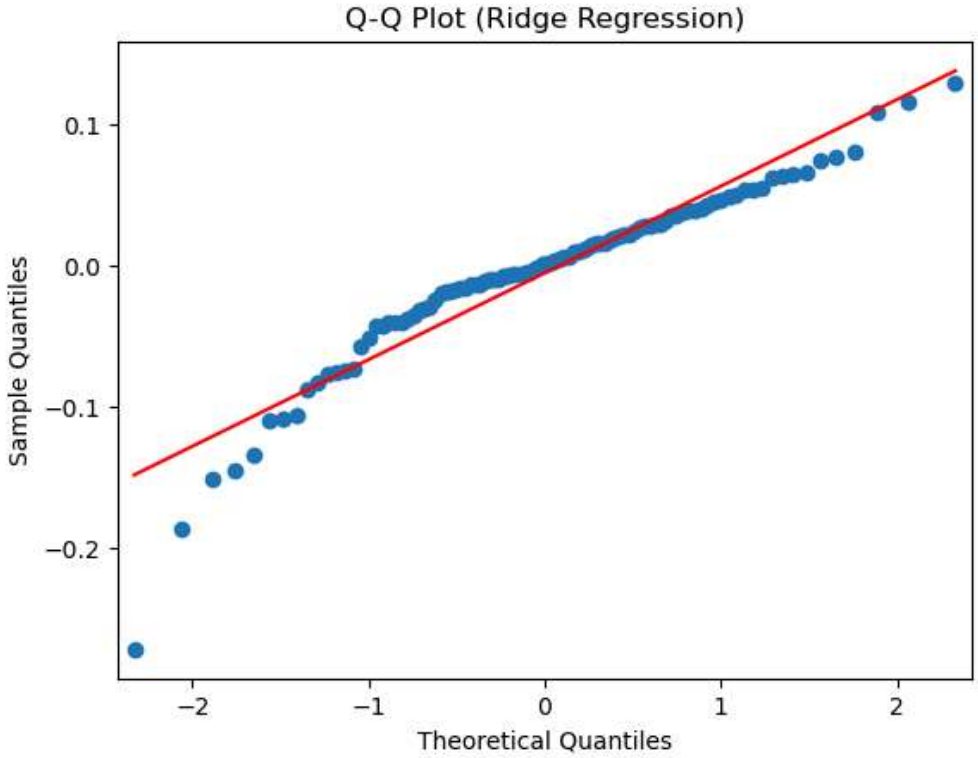
In [96]:

```
import statsmodels.api as sm

for i, residuals in enumerate(residuals_of_models):
    plt.figure(figsize=(6, 6))
    sm.qqplot(residuals, line='s')
    plt.title(f'Q-Q Plot ({model_names[i]})')
    plt.show()
```

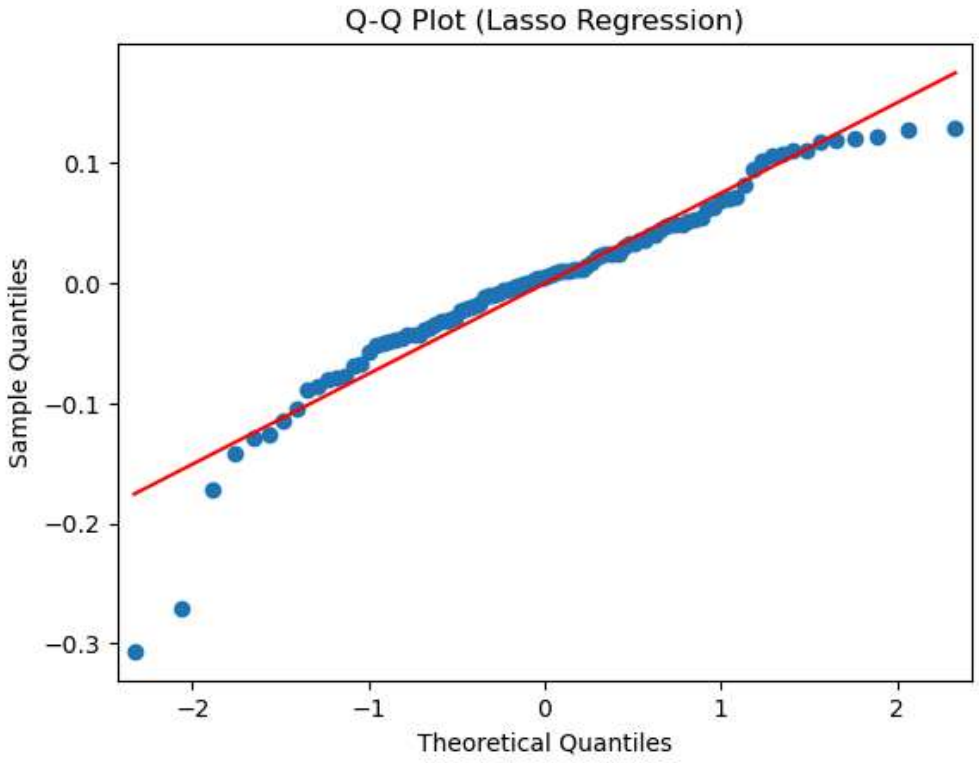
```
<Figure size 600x600 with 0 Axes>
```




```
<Figure size 600x600 with 0 Axes>
```



```
<Figure size 600x600 with 0 Axes>
```



Model performance evaluation

► Metrics checked - MAE, RMSE, R2, Adj R2

In [59]:

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

# Initialize lists to store metrics for each model
metrics = []

# Number of observations and features
n = X_test_new.shape[0]
k = X_test_new.shape[1]

# Evaluate each model and store metrics
for model_name, y_pred in zip(['Linear', 'Ridge', 'Lasso'], [y_pred_linear, y_pred_ridge
    mae = mean_absolute_error(y_test, y_pred)
    rmse = mean_squared_error(y_test, y_pred, squared = False)
    r2 = r2_score(y_test, y_pred)
    adj_r2 =  1 - (((1 - r2) * (n - 1)) / (n - k - 1))

    metrics.append({'Model': model_name,'MAE': mae,'RMSE': rmse,'R2': r2,'Adj R2': adj_r

metrics_df = pd.DataFrame(metrics)
metrics_df
```

Out[59]:

	Model	MAE	RMSE	R2	Adj R2
0	Linear	0.042923	0.061425	0.815500	0.803597
1	Ridge	0.043113	0.061624	0.814305	0.802324
2	Lasso	0.054308	0.075297	0.722755	0.704868

► Train and test performances are checked

The Linear and Ridge regression models have very similar MAE values, indicating they are making relatively accurate predictions. The Lasso model has a higher MAE, suggesting it is less accurate than the other two.

RMSE values are also lowest for Linear and Ridge models, confirming their better performance. The Lasso regression has a noticeably higher RMSE, indicating it may be less reliable in making predictions.

The R^2 values for Linear and Ridge regression are quite high (above 0.8), suggesting that they explain a significant portion of the variance in the target variable. The Lasso regression shows a much lower R^2, indicating it explains less variance in the target variable, which may be due to over-regularization or the nature of the dataset.

The adjusted R^2 values mirror the R^2 results, indicating that the Linear and Ridge models perform well while the Lasso model's performance significantly drops when considering the number of predictors.

Comments on the performance measures and if there is any need to improve the model or not

► Model Selection : Given the higher performance of Linear and Ridge models, they are more suitable for this dataset. The Lasso regression might need further tuning of the alpha parameter or reconsideration of its feature selection impact.

► Feature Engineering: Investigate potential interactions or polynomial features that could enhance model performance. The inclusion of relevant features could improve predictions further.

► Model Complexity: If the dataset allows, consider trying more complex models (e.g., Decision Trees, Random Forests, or Gradient Boosting) to see if they can outperform the linear models.

► Cross-Validation: Implement cross-validation to ensure that the model performance is robust and not overly fitted to the training set.

Actionable Insights & Recommendations

► A) Comments on significance of predictor variables

Key Predictors:

GRE Score (p-value < 0.001) : The positive coefficient (0.0024) indicates a significant positive impact on the dependent variable. Higher GRE scores lead to better outcomes, making it a critical factor for admission decisions.

TOEFL Score (p-value = 0.002) : Similar to GRE, TOEFL scores significantly influence outcomes. The coefficient (0.0030) suggests that higher language proficiency correlates with improved performance.

LOR (Letter of Recommendation) (p-value < 0.001) : With a coefficient of 0.0172, a strong LOR positively influences outcomes. This variable's significance underscores the importance of recommendation quality in the application process.

CGPA (Cumulative Grade Point Average) (p-value < 0.001) : The high coefficient (0.1125) indicates CGPA's strong

Research (p-value = 0.001) : The positive coefficient (0.0240) shows that research experience enhances applicant profiles, highlighting its value in admissions.

Insignificant Predictors:

University Rating (p-value = 0.541) : This variable is not statistically significant, indicating that it may not be a strong predictor of outcomes in the model.

SOP (Statement of Purpose) (p-value = 0.721) : The low significance suggests that SOP may not be adequately capturing the candidates' intentions or motivations, warranting a review of how this variable is assessed.

► B) Comments on Additional Data Sources for Model Improvement:

Additional Data Sources:

Engagement Metrics : Consider incorporating data on student engagement, such as participation in extracurricular activities or online learning platforms, to assess its influence on success.

Alumni Outcomes : Gather data on alumni performance post-graduation to refine predictions based on long-term success, providing a comprehensive view of how admissions decisions impact career trajectories.

Model Implementation in the Real World:

Decision Support Systems : Integrate the predictive model into admissions software to assist committees in evaluating applications based on data-driven insights, ensuring a more objective selection process.

Continuous Learning : Implement a feedback loop where the model is regularly updated with new data, allowing it to adapt and improve over time.

Potential Business Benefits:

Optimized Admissions Strategy : Enhanced predictive capabilities enable institutions to refine their selection criteria, leading to better alignment between admitted students and program success rates.

Increased Retention Rates : By identifying key success factors, institutions can develop targeted support programs for students, improving retention and overall satisfaction.

Competitive Advantage : Leveraging data to inform admissions strategies positions institutions as leaders in the field, potentially attracting higher-quality applicants and enhancing institutional reputation.

By focusing on these insights, the model can transition from a good solution to an excellent one, driving significant improvements in the admissions process and student success