# Jamboree Education - Linear Regression Case Study.

## Problem Statement

**Jamboree aims to help students estimate their probability of gaining admission to Ivy League colleges by leveraging their unique problem-solving methods. To enhance this feature, a detailed analysis is required to identify and understand the key factors influencing graduate admissions. By developing a predictive model, Jamboree seeks to provide personalized insights to students from an Indian perspective, thereby increasing their chances of successful applications.**

## Data Importing and Exploration

In [ ]:

```
# importing libraries
import pandas as pd

# loading dataset
df = pd.read_csv("/content/jamboree_admission.csv")

# data view
df.head()
```

Out[ ]:

| | 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 [ ]:

```
# basic data information
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 [ ]:

```
# describing the data
```

```
df.describe(include = "all")
```

Out[ ]:

| | 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 |

In [ ]:

```
# as per problem statement, the 'Serial No.' feature is not much usefull so let's drop it
.
df.drop(columns = ["Serial No."], inplace = True)
```

In [ ]:

```
df.head()
```

Out[ ]:

| | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research | Chance of Admit |
|---|---|---|---|---|---|---|---|---|
| **0** | 337 | 118 | 4 | 4.5 | 4.5 | 9.65 | 1 | 0.92 |
| **1** | 324 | 107 | 4 | 4.0 | 4.5 | 8.87 | 1 | 0.76 |
| **2** | 316 | 104 | 3 | 3.0 | 3.5 | 8.00 | 1 | 0.72 |
| **3** | 322 | 110 | 3 | 3.5 | 2.5 | 8.67 | 1 | 0.80 |
| **4** | 314 | 103 | 2 | 2.0 | 3.0 | 8.21 | 0 | 0.65 |

# Exploratory Data Analysis

**Here we will be visualising the data by performing univariate and bivariate analysis.**

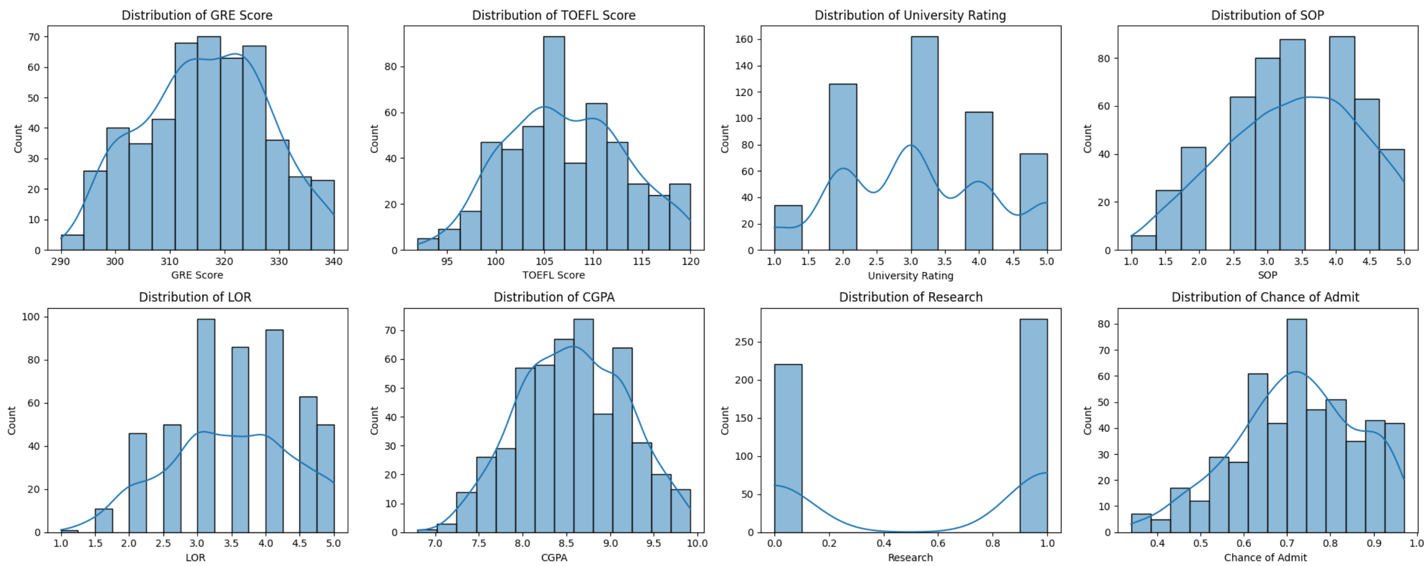
In [ ]:

```
# importing libraries for data visualization.
import matplotlib.pyplot as plt
import seaborn as sns

# Let's create a function to perform analysis on the data
# Univariate Analysis
def univariate_analysis(data):
    num_columns = df.shape[1]
    num_rows = (num_columns + 3) // 4   # Calculate the number of rows needed

    # Create a figure with the required number of subplots
    fig, axes = plt.subplots(num_rows, 4, figsize=(20, num_rows * 4))
    axes = axes.flatten()   # Flatten the 2D array of axes to 1D for easy iteration

    for i, column in enumerate(df.columns):
        sns.histplot(df[column], kde=True, ax=axes[i])
        axes[i].set_title(f'Distribution of {column}')

    # Remove any empty subplots
    for j in range(i + 1, len(axes)):
```

```python
            fig.delaxes(axes[j])

    plt.tight_layout()
    plt.show()

# Bivariate Analysis
def bivariate_analysis(df):
    sns.pairplot(df)
    plt.show()
```

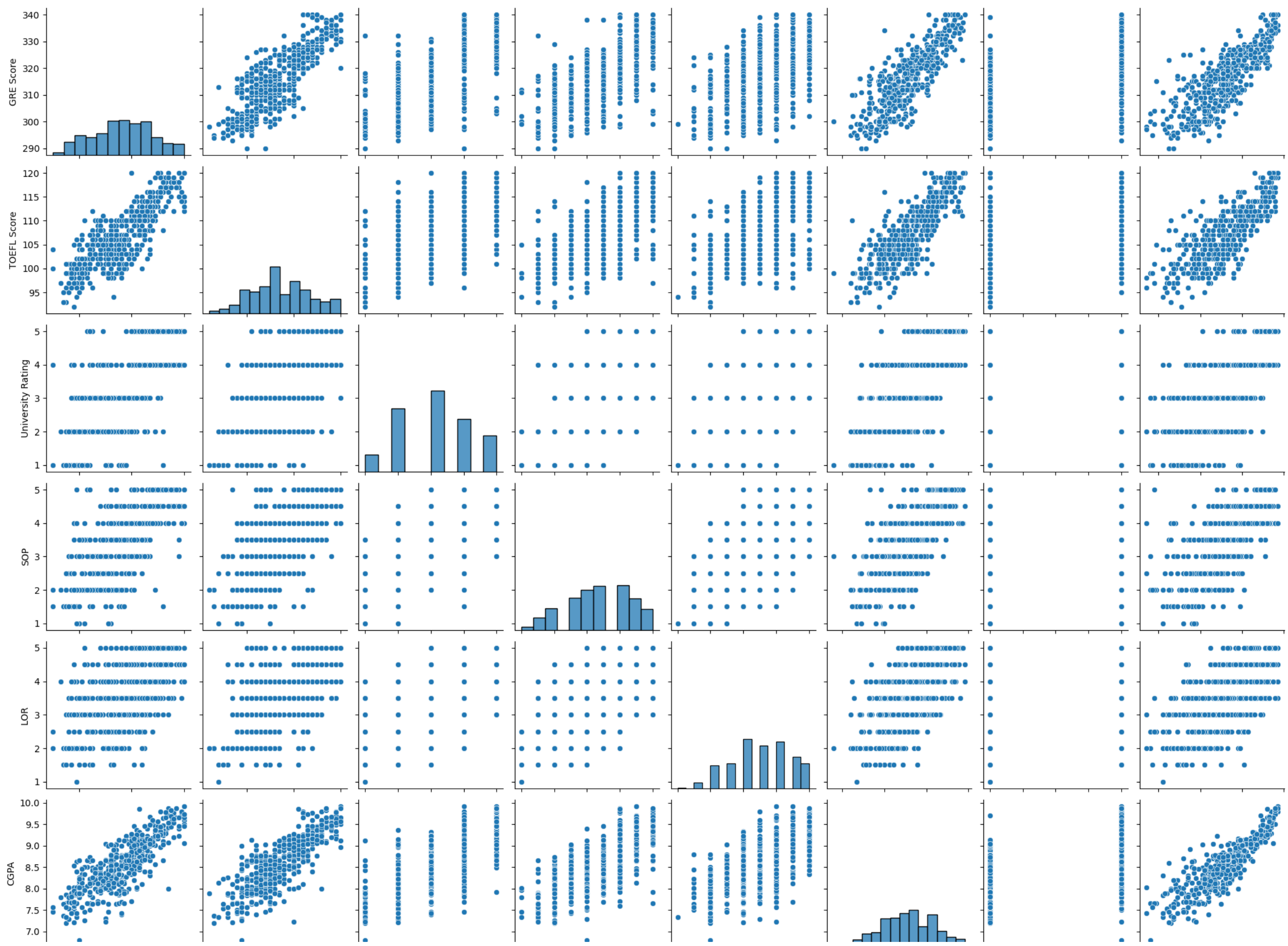
In [ ]:

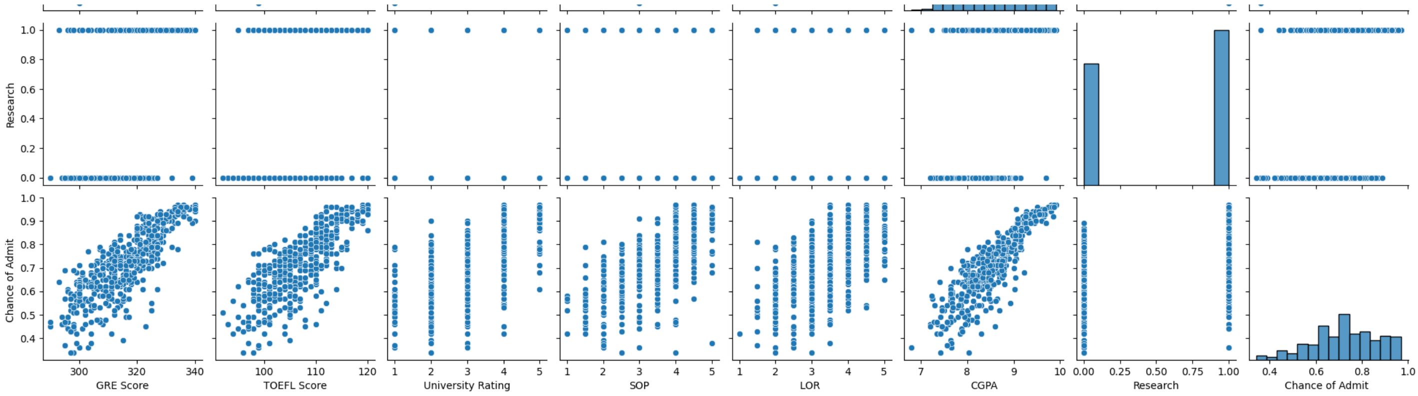
```python
# Calling EDA functions
univariate_analysis(df)
```




In [ ]:

```python
bivariate_analysis(df)
```






**OBSERVATION:**

**Univariate Analysis**

- **All the features nearly looks normally distributed, no such major skewness is observed except the research feature where it is showing the extreme value distribution. But, as we can see in the pairplot, it is not much affecting the target variable so we can ignore the its distribution.**
- **Majority of students have scores centered around the middle value range of their respective score features.**

**Bivariate Analysis**

- **GRE Score, TOEFl Score, CGPA, and Chance of Admit are positively correlated with each other. This means the scores of an individal are highly responsible for getting one admitted into the abroad colleges.**
- **There are no such traces of outliers.**
- **The feature 'Research' doesn't affect an individual's chance of getting admission.**

# Data Preprocessing

**Now, we will check for duplicate values, outliers, etc.**

In [ ]:

```
df.duplicated().sum()
```

Out[ ]:

0

In [ ]:

```
def treat_outliers(data, column):
    # Calculate Q1 (25th percentile) and Q3 (75th percentile)
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)

    # Calculate the Interquartile Range (IQR)
    IQR = Q3 - Q1

    # Calculate the lower and upper bound
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Print bounds for debugging
    print(f"Column: {column}, Lower Bound: {lower_bound}, Upper Bound: {upper_bound}")

    # Treat outliers by capping them to the lower and upper bounds
    df[column] = df[column].apply(lambda x: lower_bound if x < lower_bound else (upper_b
ound if x > upper_bound else x))

    return df
```

In [ ]:

```
for column in df.columns:
```

```
    if pd.api.types.is_numeric_dtype(df[column]):
      data = treat_outliers(df, column)

data.head()
```

```
Column: GRE Score, Lower Bound: 282.5, Upper Bound: 350.5
Column: TOEFL Score, Lower Bound: 89.5, Upper Bound: 125.5
Column: University Rating, Lower Bound: -1.0, Upper Bound: 7.0
Column: SOP, Lower Bound: 0.25, Upper Bound: 6.25
Column: LOR , Lower Bound: 1.5, Upper Bound: 5.5
Column: CGPA, Lower Bound: 6.758750000000045, Upper Bound: 10.408749999999996
Column: Research, Lower Bound: -1.5, Upper Bound: 2.5
Column: Chance of Admit , Lower Bound: 0.3450000000000001, Upper Bound: 1.105
```

Out[ ]:

| | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research | Chance of Admit |
|---|---|---|---|---|---|---|---|---|
| 0 | 337 | 118 | 4 | 4.5 | 4.5 | 9.65 | 1 | 0.92 |
| 1 | 324 | 107 | 4 | 4.0 | 4.5 | 8.87 | 1 | 0.76 |
| 2 | 316 | 104 | 3 | 3.0 | 3.5 | 8.00 | 1 | 0.72 |
| 3 | 322 | 110 | 3 | 3.5 | 2.5 | 8.67 | 1 | 0.80 |
| 4 | 314 | 103 | 2 | 2.0 | 3.0 | 8.21 | 0 | 0.65 |

In [ ]:

```
df.describe()
```

Out[ ]:

| | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research | Chance of Admit |
|---|---|---|---|---|---|---|---|---|
| count | 500.000000 | 500.000000 | 500.000000 | 500.000000 | 500.000000 | 500.000000 | 500.000000 | 500.000000 |
| mean | 316.472000 | 107.192000 | 3.114000 | 3.374000 | 3.485000 | 8.576440 | 0.560000 | 0.721760 |
| std | 11.295148 | 6.081868 | 1.143512 | 0.991004 | 0.923027 | 0.604813 | 0.496884 | 0.141087 |
| min | 290.000000 | 92.000000 | 1.000000 | 1.000000 | 1.500000 | 6.800000 | 0.000000 | 0.345000 |
| 25% | 308.000000 | 103.000000 | 2.000000 | 2.500000 | 3.000000 | 8.127500 | 0.000000 | 0.630000 |
| 50% | 317.000000 | 107.000000 | 3.000000 | 3.500000 | 3.500000 | 8.560000 | 1.000000 | 0.720000 |
| 75% | 325.000000 | 112.000000 | 4.000000 | 4.000000 | 4.000000 | 9.040000 | 1.000000 | 0.820000 |
| max | 340.000000 | 120.000000 | 5.000000 | 5.000000 | 5.000000 | 9.920000 | 1.000000 | 0.970000 |

**OBSERVATION:**

**As we can clearly see after comparing the before and after data, we do not see any kind of major outliers. So the data is pretty good for model building.**

# Model Building

**Now as the data is ready, we can move further towards the model building part. As in here we will be using linear regression model to train and test our data.**

**The reason behind using linear regression is that it is used to predict continous values or simply numbers.**

In [ ]:

```
# importing libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# extracting independent features and target variable
X = df.drop(columns = ['Chance of Admit '])
```

```
y = data['Chance of Admit ']

# splitting data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_state =
24)

# scaling down the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# model building and fitting the training data into the model
model = LinearRegression()
model.fit(X_train_scaled, y_train)
```

Out[ ]:

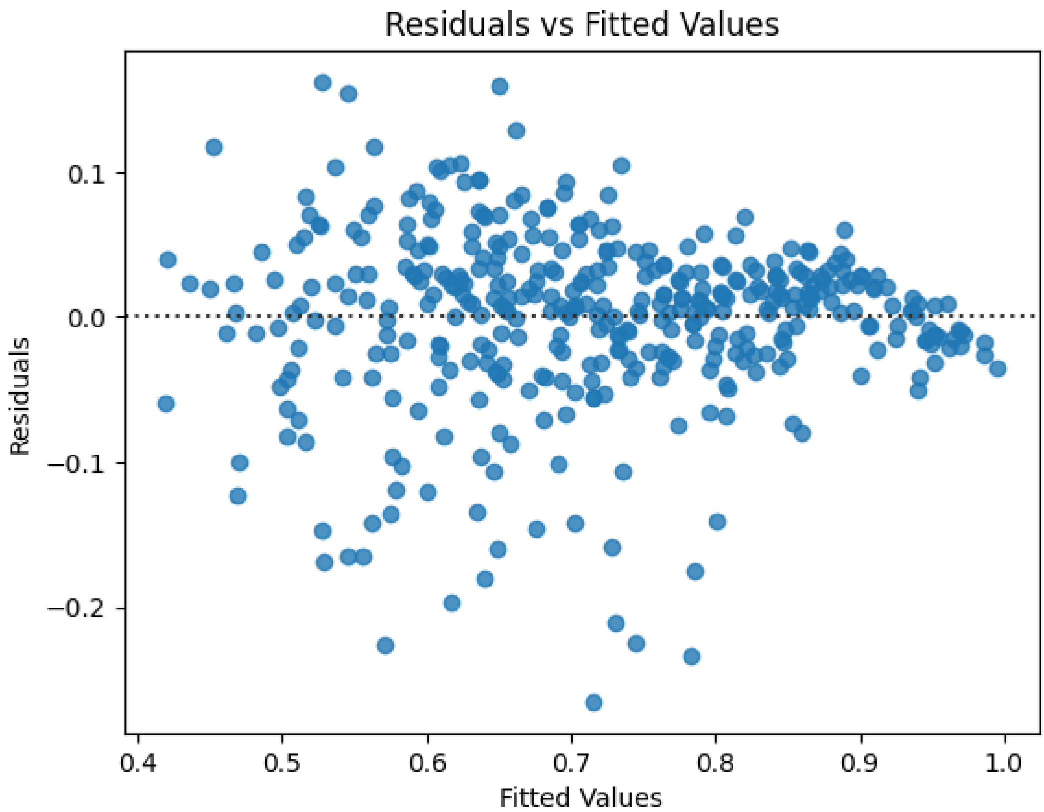
```
▾ LinearRegression
LinearRegression()
```

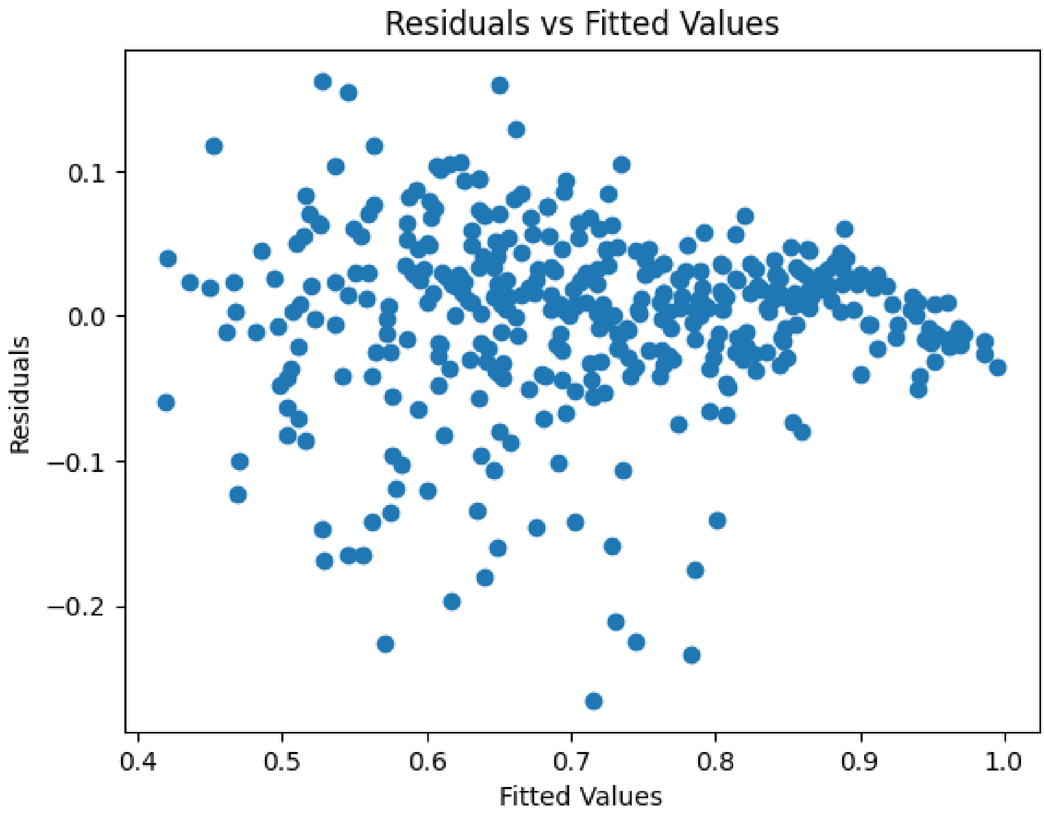
In [ ]:

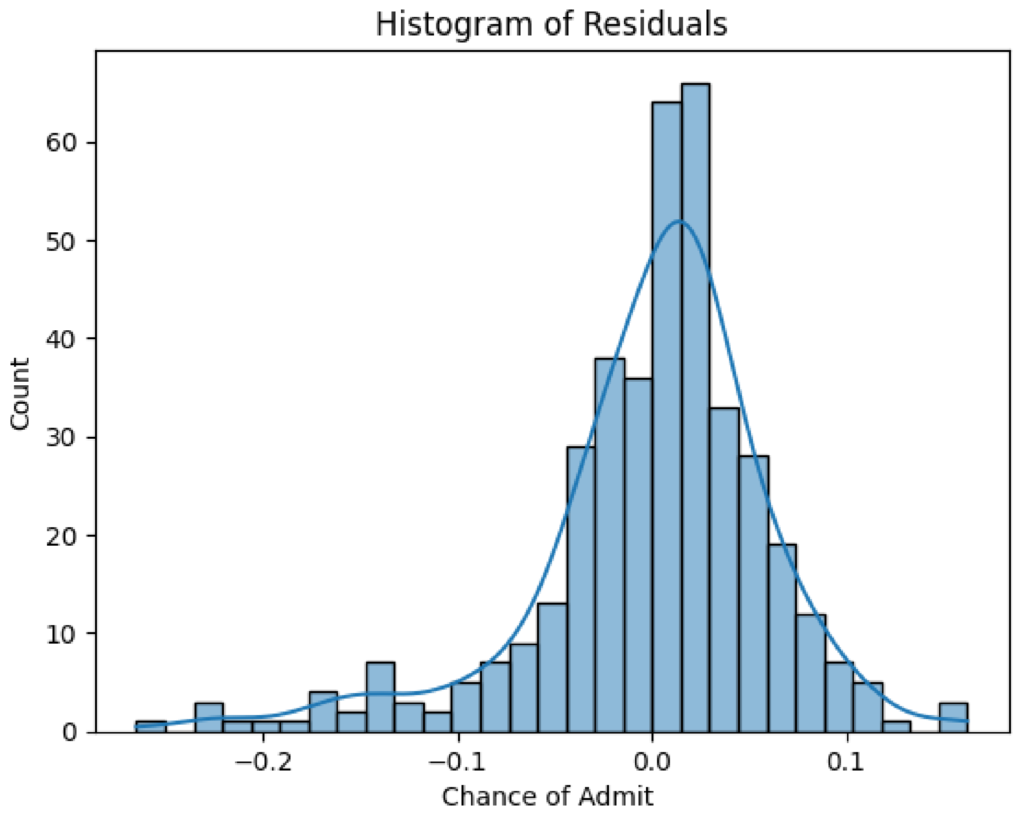
```
import numpy as np
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Convert the scaled array back to a DataFrame
X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=X_train.columns)

# Multicollinearity check (VIF)
def check_vif(X):
    vif = pd.DataFrame()
    vif["Features"] = X_train_scaled_df.columns
    vif["VIF"] = [variance_inflation_factor(X_train_scaled_df.values, i) for i in range(
X_train_scaled_df.shape[1])]
    return vif

vif_df = check_vif(X_train_scaled)
print("VIF for each feature:")
print(vif_df)

# Calculate residuals
y_pred_train = model.predict(X_train_scaled)
residuals_train = y_train - y_pred_train

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

# Linearity check (Residuals vs Fitted Values)
sns.residplot(x=y_pred_train, y=residuals_train)
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs Fitted Values")
plt.show()

# Homoscedasticity test (Residuals vs Fitted Values)
# Scatter plot of residuals vs fitted values is a simple way to visualize homoscedasticit
y
plt.scatter(x=y_pred_train, y=residuals_train)
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs Fitted Values")
plt.show()

# Normality of residuals
# Histogram of residuals
sns.histplot(residuals_train, kde=True)
plt.title("Histogram of Residuals")
```

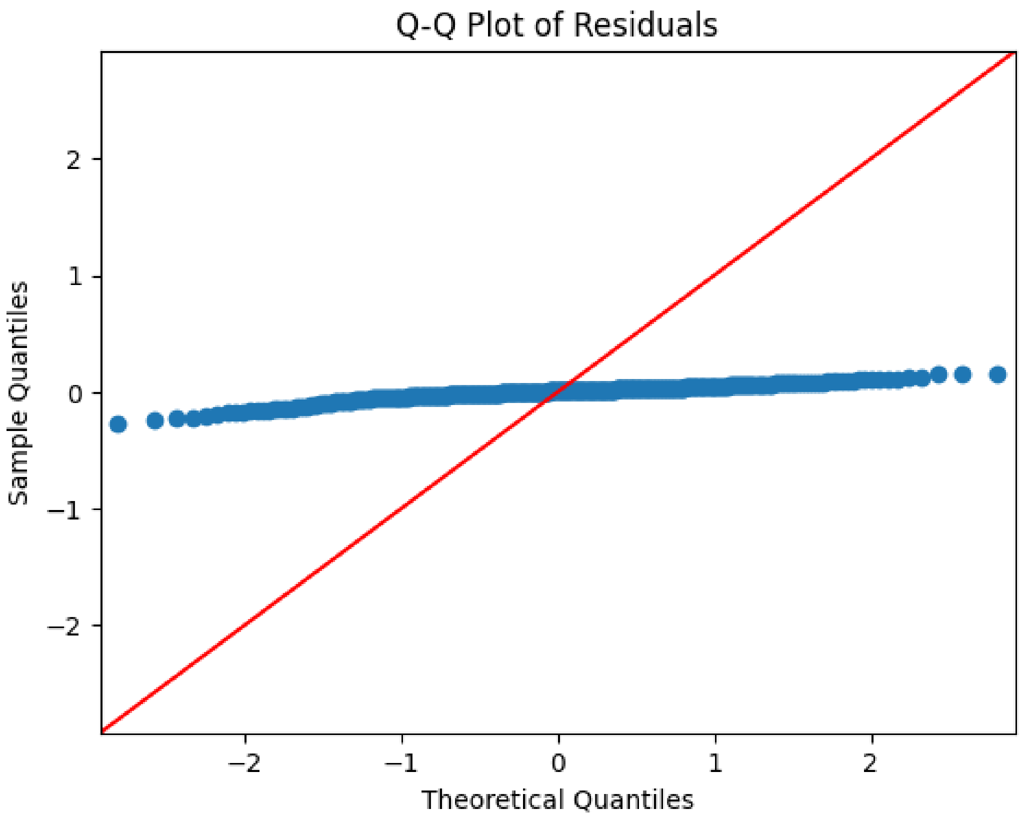
```
plt.show()

# Q-Q plot of residuals
sm.qqplot(residuals_train, line='45')
plt.title("Q-Q Plot of Residuals")
plt.show()
```

```
VIF for each feature:
            Features       VIF
0          GRE Score  4.509742
1        TOEFL Score  3.963875
2  University Rating  2.499077
3                SOP  2.857948
4               LOR   2.016254
5               CGPA  4.972822
6           Research  1.488824
Mean of residuals: -1.2406742300186124e-16
```






**OBSERVATION:**

**VIF Interpretation:**

- **VIF values close to 1 indicate low multicollinearity, meaning the variable is not highly correlated with other predictors.**
- **VIF values above 5 or 10 suggest high multicollinearity, indicating that the variable may be redundant and could cause instability in the regression coefficients.**

**Interpretation of VIF Results:**

1. **GRE Score: VIF = 4.51 - Moderate multicollinearity, but not too high.**
2. **TOEFL Score: VIF = 3.96 - Moderate multicollinearity, similar to GRE Score.**

3. **University Rating**: VIF = 2.50 - Low multicollinearity, the variable is not highly correlated with others.
4. **SOP**: VIF = 2.86 - Low to moderate multicollinearity.
5. **LOR**: VIF = 2.02 - Low multicollinearity.
6. **CGPA**: VIF = 4.97 - Moderate multicollinearity, similar to GRE and TOEFL scores.
7. **Research**: VIF = 1.49 - Very low multicollinearity, the variable is not highly correlated with others.

**Mean of Residuals: The mean of residuals being close to zero (-1.24e-16) indicates that, on average, the residuals are balanced around zero, which is a good sign. It suggests that the model is unbiased and does not systematically overestimate or underestimate the target variable.**

**Overall interpretation...**

**The VIF values suggest that while there is some multicollinearity present in the data, it is not severe enough to warrant dropping any predictors. However, it's essential to keep an eye on variables with higher VIF values, such as GRE Score, TOEFL Score, and CGPA, and consider potential model refinement if multicollinearity becomes problematic during further analysis.**

In [ ]:

```
X_train_scaled_df
```

Out[ ]:

| | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research |
|---|---|---|---|---|---|---|---|
| 0 | 1.028965 | 0.462435 | 0.791672 | 1.681798 | 0.596998 | 0.945387 | 0.909112 |
| 1 | 2.081390 | 1.288949 | 1.678699 | 1.165909 | 1.139723 | 1.458420 | 0.909112 |
| 2 | -0.374269 | -0.694685 | -0.095355 | 0.134131 | 0.596998 | 0.349607 | -1.099975 |
| 3 | -1.777503 | -1.355896 | -0.982382 | -0.381758 | 0.054273 | -2.132810 | -1.099975 |
| 4 | 0.590454 | 0.958343 | -0.095355 | 0.650020 | -0.488453 | 1.243277 | 0.909112 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 395 | -0.111162 | -0.364079 | -0.982382 | -0.381758 | -0.488453 | -0.378569 | -1.099975 |
| 396 | -0.988184 | -0.694685 | -0.982382 | -0.897647 | 0.054273 | -0.726107 | -1.099975 |
| 397 | 0.502752 | 1.123646 | 1.678699 | 1.165909 | 0.596998 | 0.614398 | 0.909112 |
| 398 | -0.812779 | -0.364079 | -0.982382 | -1.413536 | 0.054273 | -0.775755 | -1.099975 |
| 399 | -0.637375 | 0.627737 | -0.982382 | -0.897647 | 0.596998 | -0.891601 | -1.099975 |

400 rows × 7 columns

# Model Performance Check

## R2 score

In [ ]:

```
# Model's train and test performance

train_score = model.score(X_train_scaled, y_train)
test_score = model.score(X_test_scaled, y_test)

print("The training score: ", train_score)
print("The testing score: ", test_score)
```

```
The training score:  0.8210171036855565
The testing score:  0.8173726889383743
```

**The training R-squared score is 0.821, indicating that approximately 82.1% of the variance in the target variable (chance of admission) is explained by the model on the training data.**

**Similarly, the testing R-squared score is 0.817, suggesting that approximately 81.7% of the variance in the target**

Similarly, the testing R-squared score is 0.817, suggesting that approximately 81.7% of the variance in the target variable is explained by the model on the test data.

These R-squared scores indicate that the model performs relatively well on both the training and testing data, as they are close to each other and reasonably high.

### MSE - mean squared error

In [ ]:

```
from sklearn.metrics import mean_squared_error

train_mse = mean_squared_error(y_train, model.predict(X_train_scaled))
test_mse = mean_squared_error(y_test, model.predict(X_test_scaled))

print("The training score: ", train_mse)
print("The testing score: ", test_mse)
```

```
The training score:  0.003701405560155573
The testing score:  0.0029719654575571484
```

### RMSE - root mean squared error

In [ ]:

```
train_rmse = np.sqrt(train_mse)
test_rmse = np.sqrt(test_mse)

print("The training score: ", train_mse)
print("The testing score: ", test_mse)
```

```
The training score:  0.003701405560155573
The testing score:  0.0029719654575571484
```

### Adjusted R2 score

In [ ]:

```
# Adjusted R-squared for training set
n = len(X_train_scaled)
p = X_train_scaled.shape[1]
adj_train_score = 1 - ((1 - train_score) * (n - 1) / (n - p - 1))

# Adjusted R-squared for test set
n = len(X_test_scaled)
p = X_test_scaled.shape[1]
adj_test_score = 1 - ((1 - test_score) * (n - 1) / (n - p - 1))

print("The adjusted training R-squared score: ", adj_train_score)
print("The adjusted testing R-squared score: ", adj_test_score)
```

```
The adjusted training R-squared score:  0.8178209805370842
The adjusted testing R-squared score:  0.8034771326619463
```

## Insights and Recommendations

1. **Significance of Predictor Variables:**

  - **GRE Score, TOEFL Score, and CGPA** : These variables exhibit moderate multicollinearity but are crucial predictors of admission chances. Students should focus on improving these scores to enhance their likelihood of acceptance.
  - **University Rating, SOP, and LOR** : While these factors also contribute to admission decisions, they exhibit lower multicollinearity and thus provide additional insights beyond academic performance.
  - **Research Experience:** Although it has a low VIF and does not significantly impact admission chances, research experience can still be a distinguishing factor, particularly for candidates applying to research-

research experience can still be a distinguishing factor, particularly for candidates applying to research-oriented programs.

2. **Additional Data Sources for Model Improvement:**

    - Additional features such as extracurricular activities, internships, and personal statements could provide valuable insights into applicants' holistic profiles.
    - Gathering data on admission criteria specific to Ivy League colleges, such as interview performance and recommendations from alumni, could offer deeper insights into the decision-making process.

3. **Model Implementation in the Real World :**

    - Jamboree can integrate this predictive model into its existing platform, offering students personalized insights into their admission probabilities.
    - By providing actionable recommendations based on the model's insights, Jamboree can guide students on areas for improvement and strategies to enhance their profiles.

4. **Potential Business Benefits:**

    - Improved student outcomes: By helping students optimize their application profiles, Jamboree can increase the likelihood of successful admissions to prestigious colleges, enhancing its reputation and attracting more clients.
    - Differentiation in the market: Offering a predictive model tailored to Ivy League admissions distinguishes Jamboree from competitors, positioning it as a leader in providing data-driven solutions for academic success.
    - Enhanced customer satisfaction: Providing personalized insights and actionable recommendations demonstrates Jamboree's commitment to student success, fostering long-term relationships and customer loyalty.