In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
```

In [2]:
```python
df=pd.read_csv("Enter the path)
df.head(5)
```

Out[2]:

	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 [3]:
```python
df.shape
```

Out[3]:
```
(500, 9)
```

The dataset has **500 records** and **9 features**

In [4]:
```python
df.dtypes
```

Out[4]:
```
Serial No.              int64
GRE Score               int64
TOEFL Score             int64
University Rating       int64
SOP                   float64
LOR                   float64
CGPA                  float64
Research                int64
Chance of Admit       float64
dtype: object
```

The data types looks good, no need for any manipulation here

In [5]:
```python
df.describe()
```

Out[5]:

	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 [6]:
```python
df.isna().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
```

no missing values in this data

In [7]:
```python
# Drop the 'Serial No.' column if it's a unique row identifier
if 'Serial No.' in df.columns:
    df.drop('Serial No.', axis=1, inplace=True)
    print("\n'Serial No.' column dropped successfully.")
else:
    print("\nNo unique row identifier found.")
```

```
'Serial No.' column dropped successfully.
```

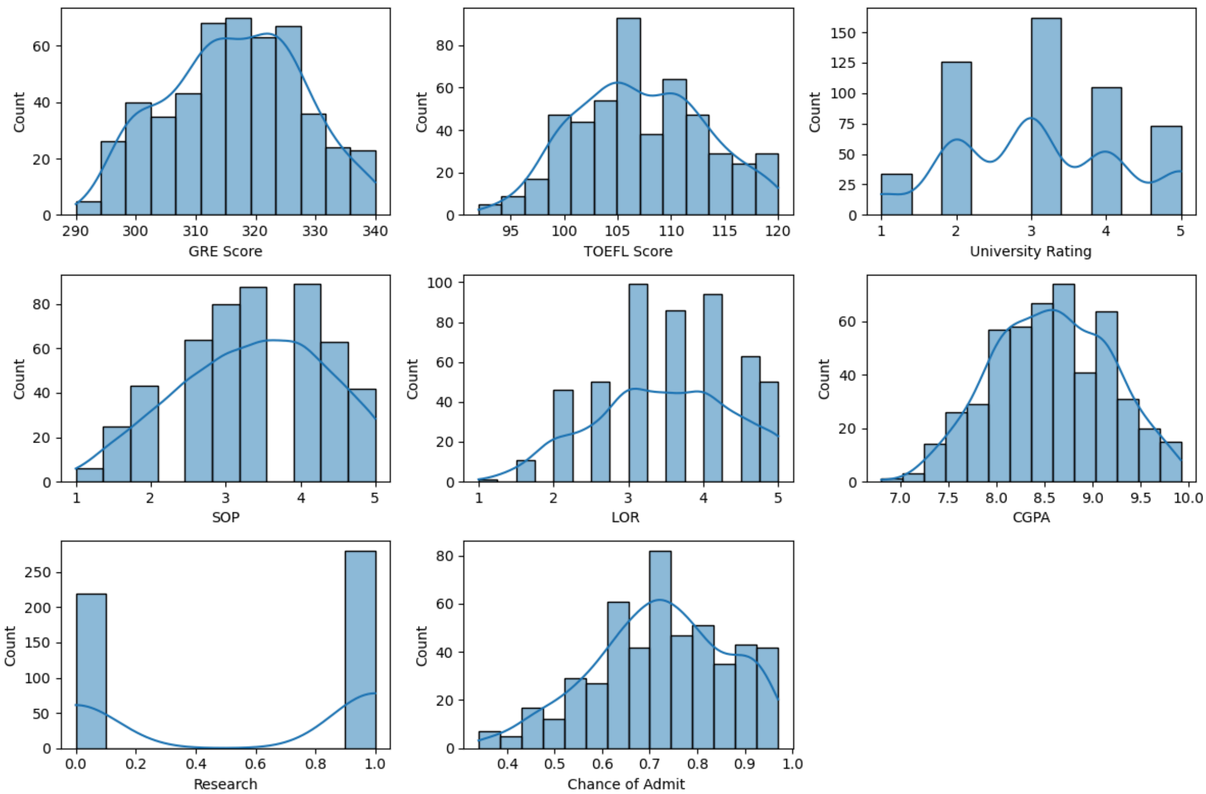
In [8]:
```python
df.head(2)
```

Out[8]:

	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

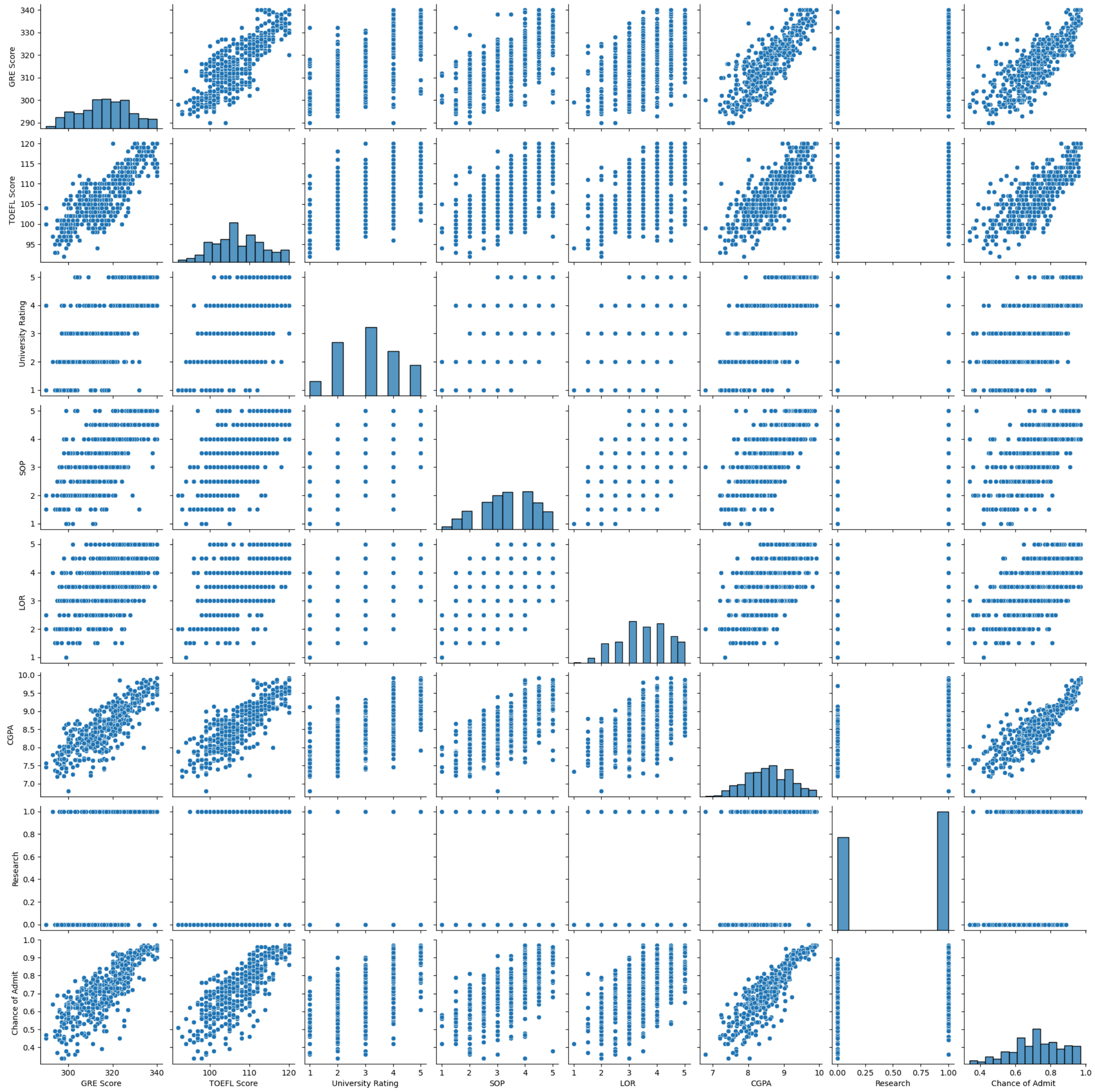
Univariate Analysis

In [9]:
```python
# Plot distributions of continuous variables
plt.figure(figsize=(12, 8))
for i, column in enumerate(df.select_dtypes(include=['int64', 'float64'])):
    plt.subplot(3, 3, i + 1)
    sns.histplot(df[column], kde=True)
    #plt.title(column)
plt.tight_layout()
plt.show()
```



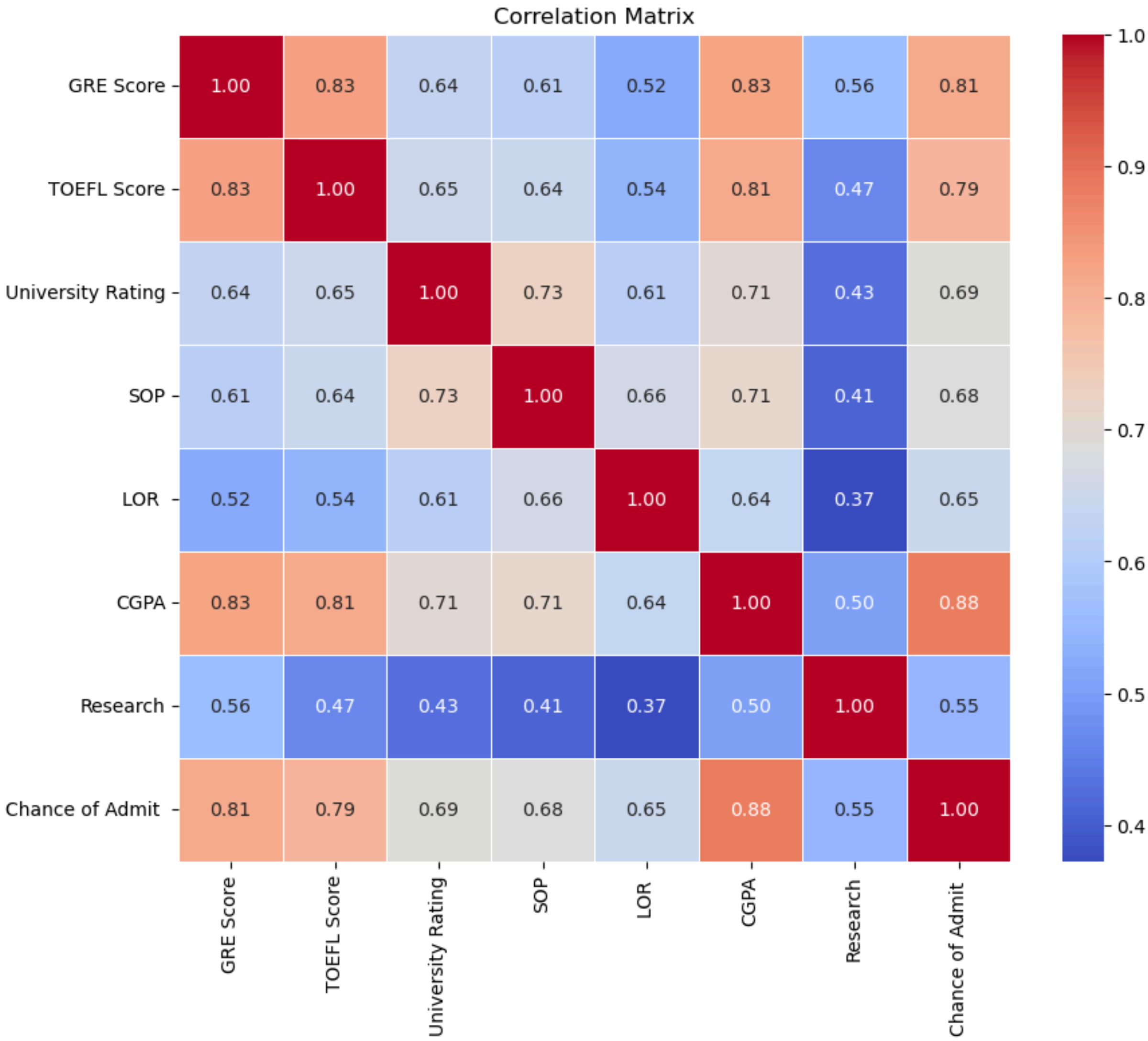
GRE score,**TOEFL score** and **CGPA** are have normal distribution tendency from the plot.**Chance of admit** is right skewed

In [10]:
```
# Pairplot for relationships between variables
sns.pairplot(df)
plt.show()
```



GRE score,**TOEFL score** and **CGPA** have linear relationship with**Chance of admit** but rest of the columns, being discrete have an axis parallel relationship with the target column

In [11]:
```
# Calculate correlation matrix
correlation_matrix = df.corr()

# Plot correlation matrix as a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



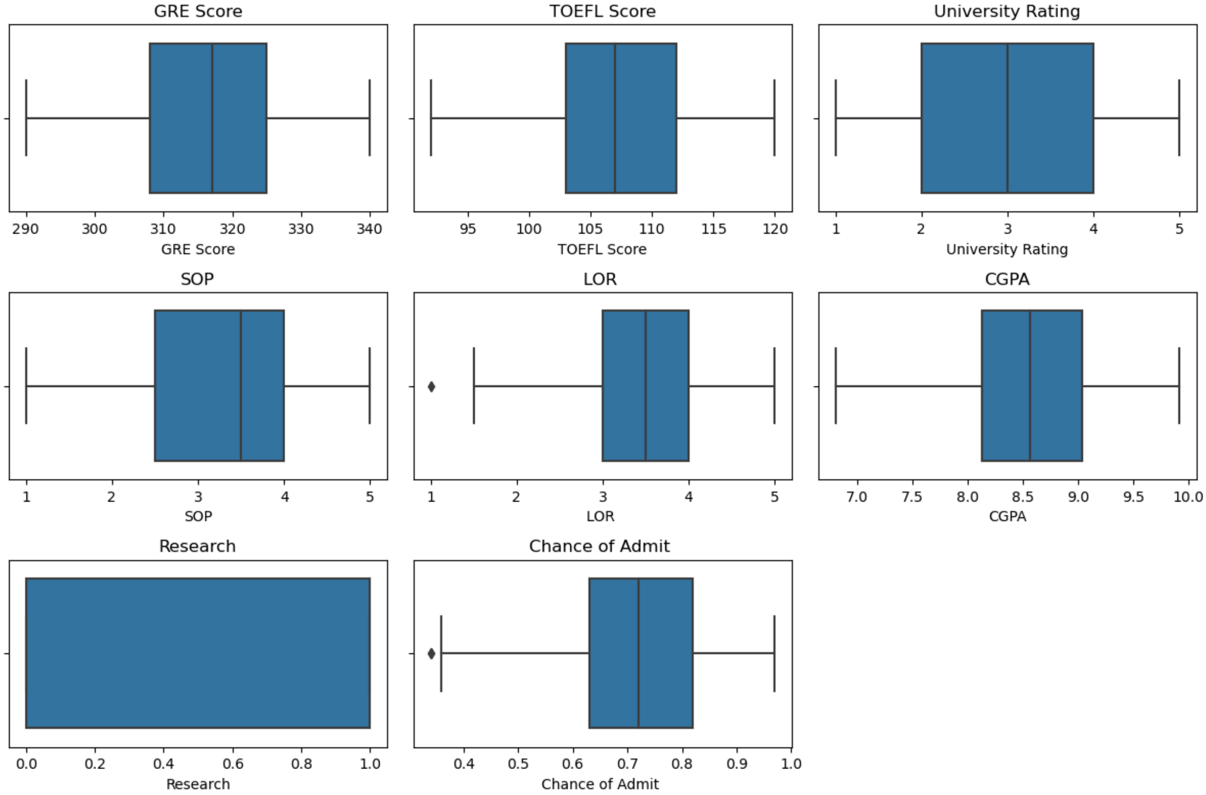
In [12]:
```
# Check for duplicate rows
duplicate_rows = df[df.duplicated()]
if len(duplicate_rows)>0:
    print(duplicate_rows.head())
else:
    print("No duplicate rows")
```

```
No duplicate rows
```

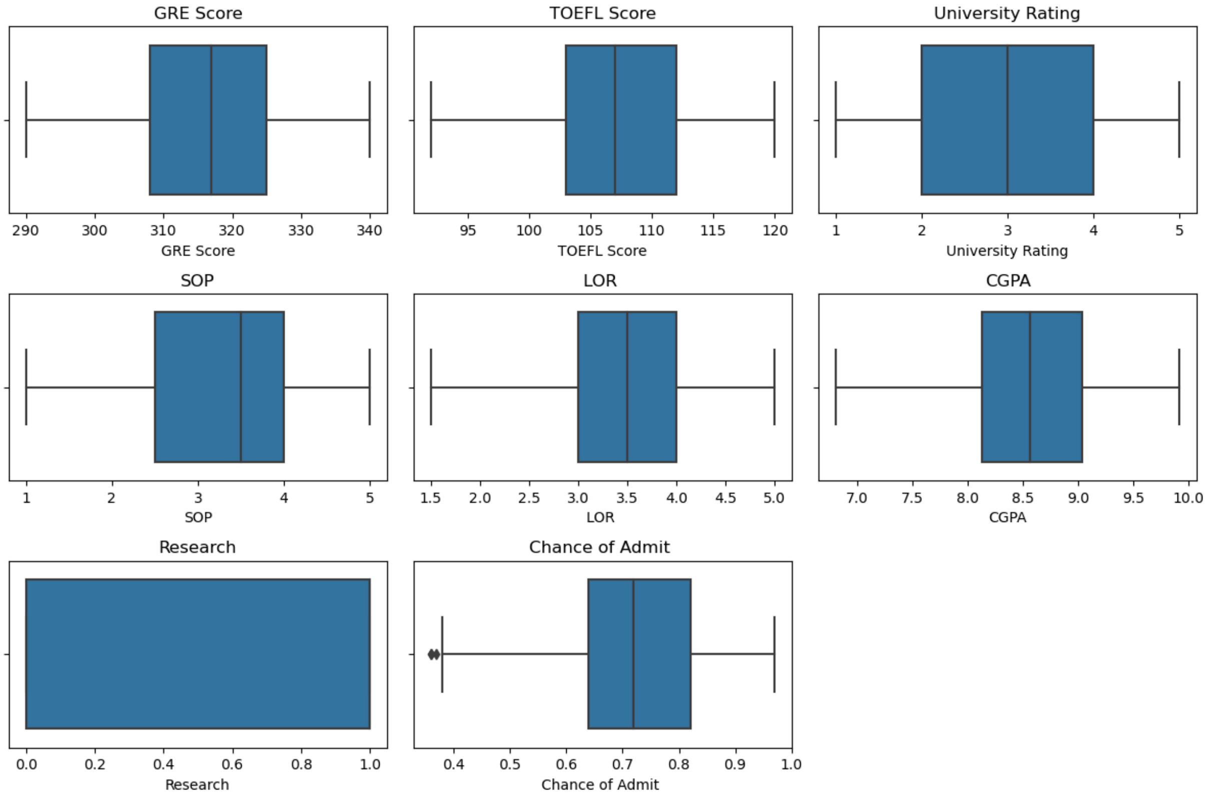
In [13]:
```
warnings.filterwarnings('ignore')
# Detect outliers using IQR method
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
outliers = ((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)

# Count the number of outliers detected
print("Number of outliers detected:", outliers.sum())

# Plot boxplots before outlier removal
plt.figure(figsize=(12, 8))
for i, column in enumerate(df.select_dtypes(include=['int64', 'float64'])):
    plt.subplot(3, 3, i + 1)
    sns.boxplot(df[column])
    plt.title(column)
plt.tight_layout()
plt.show()


# Remove outliers
df = df[~outliers]
print("df after removing outliers:", df.shape)
```

```python
# Plot boxplots after outlier removal
plt.figure(figsize=(12, 8))
for i, column in enumerate(df.select_dtypes(include=['int64', 'float64'])):
    plt.subplot(3, 3, i + 1)
    sns.boxplot(df[column])
    plt.title(column)
plt.tight_layout()
plt.show()
```

```
Number of outliers detected: 3
```




```
df after removing outliers: (497, 8)
```



In [14]:

```python
# Standardize the features using StandardScaler
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

```python
X_scaled = scaler.fit_transform(df)

# Convert scaled features back to DataFrame
X_scaled_df = pd.DataFrame(X_scaled, columns=df.columns)
```

In [15]:
```python
print(X_scaled_df.columns)
```

```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

In [16]:
```python
X_scaled_df.head(5)
```

Out[16]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1.818719	1.781161	0.769761	1.136549	1.097138	1.777188	0.880341	1.414372
1	0.660668	-0.043044	0.769761	0.629489	1.097138	0.478836	0.880341	0.260470
2	-0.051979	-0.540555	-0.107696	-0.384631	0.007672	-0.969326	0.880341	-0.028006
3	0.482506	0.454466	-0.107696	0.122429	-1.081793	0.145925	0.880341	0.548945
4	-0.230140	-0.706391	-0.985153	-1.398751	-0.537061	-0.619770	-1.135924	-0.532838

In [17]:
```python
# Split the data into train and test sets
from sklearn.model_selection import train_test_split

X_scaled_df.rename(columns={'Chance of Admit ': 'Chance of Admit'}, inplace=True)

y = X_scaled_df['Chance of Admit']
X = X_scaled_df.drop('Chance of Admit', axis=1)

# Split data into train and test sets (80% train, 20% test)
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Further split train_val set into train and validation sets (75% train, 25% validation)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.25, random_state

# Print the shapes of the splits
print("Train set shape:", X_train.shape, y_train.shape)
print("Validation set shape:", X_val.shape, y_val.shape)
print("Test set shape:", X_test.shape, y_test.shape)
```

```
Train set shape: (297, 7) (297,)
Validation set shape: (100, 7) (100,)
Test set shape: (100, 7) (100,)
```

base models

In [18]:
```python
def get_adj_r2_score(r2_score,n,p):
    adj_r2_score = 1 - (1 - r2_score) * (n - 1) / (n - p - 1)
    return adj_r2_score
```

In [19]:
```python
# Build a Linear Regression model
from sklearn.linear_model import LinearRegression

# Initialize the Linear Regression model
linear_reg_model = LinearRegression()

# Fit the model on the training data
linear_reg_model.fit(X_train, y_train)

# Print model statistics
print("Linear Regression Model Statistics:")
print("Intercept:", linear_reg_model.intercept_)
print("Coefficients:")
for feature, coef in zip(X_train.columns, linear_reg_model.coef_):
    print(feature, ':', coef)

r2_score=linear_reg_model.score(X_val, y_val)
print("R2 score:",r2_score)

n = len(X_val)
```

```python
p = X_val.shape[1]
adj_r2_score=get_adj_r2_score(r2_score,n,p)
print("Adjusted R2 score:",adj_r2_score)
```

```
Linear Regression Model Statistics:
Intercept: 0.011753764690O9573
Coefficients:
GRE Score : 0.10511533647985655
TOEFL Score : 0.14188530708062613
University Rating : 0.011667718864517866
SOP : 0.06406358090721308
LOR  : 0.11550749200156747
CGPA : 0.4890914853118642
Research : 0.11078670467307958
R2 score: 0.8395696825267027
Adjusted R2 score: 0.8273630279363431
```

In [20]:

```python
# Build a Ridge Regression model
from sklearn.linear_model import Ridge

# Initialize the Ridge Regression model
ridge_model = Ridge(alpha=1.0)  # You can adjust the alpha parameter for regularization

# Fit the model on the training data
ridge_model.fit(X_train, y_train)

# Print model statistics
print("\nRidge Regression Model Statistics:")
print("Intercept:", ridge_model.intercept_)
print("Coefficients:")
for feature, coef in zip(X_train.columns, ridge_model.coef_):
    print(feature, ':', coef)

r2_score=ridge_model.score(X_val, y_val)
print("R2 score:",r2_score)

n = len(X_val)
p = X_val.shape[1]
adj_r2_score=get_adj_r2_score(r2_score,n,p)
print("Adjusted R2 score:",adj_r2_score)
```

```
Ridge Regression Model Statistics:
Intercept: 0.011701440755419446
Coefficients:
GRE Score : 0.1077424836229993
TOEFL Score : 0.14269214935850066
University Rating : 0.012887047128115765
SOP : 0.06509694403238281
LOR  : 0.11618071545969225
CGPA : 0.4827665927607097
Research : 0.1103672083927703
R2 score: 0.8395726944496715
Adjusted R2 score: 0.827366269027364
```

In [21]:

```python
# Build a Lasso Regression model
from sklearn.linear_model import Lasso

# Initialize the Lasso Regression model
lasso_model = Lasso(alpha=0.1)  # You can adjust the alpha parameter for regularization

# Fit the model on the training data
lasso_model.fit(X_train, y_train)

# Print model statistics
print("\nLasso Regression Model Statistics:")
print("Intercept:", lasso_model.intercept_)
print("Coefficients:")
for feature, coef in zip(X_train.columns, lasso_model.coef_):
    print(feature, ':', coef)


r2_score=lasso_model.score(X_val, y_val)
print("R2 score:",r2_score)

n = len(X_val)
p = X_val.shape[1]
```

```python
adj_r2_score=get_adj_r2_score(r2_score,n,p)
print("Adjusted R2 score:",adj_r2_score)
```

```
Lasso Regression Model Statistics:
Intercept: 0.014372420496825765
Coefficients:
GRE Score : 0.09969327977059905
TOEFL Score : 0.09923351704042649
University Rating : 0.0
SOP : 0.04181378073365192
LOR  : 0.07580516053697718
CGPA : 0.5042921986345623
Research : 0.054965250901576765
R2 score: 0.8281720778545867
Adjusted R2 score: 0.8150982142130878
```

Using statsmodel OLS method to check for assumptions of linear regression

In [22]:
```python
#VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Calculate VIF scores
vif = pd.DataFrame()
vif["Features"] = X_train.columns
vif["VIF Score"] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]


# Drop variables with VIF > 5
high_vif_features = vif[vif["VIF Score"] > 5]["Features"]
print("No of high VIF featues:",len(high_vif_features))
print("high VIF features:",high_vif_features)
X_train.drop(high_vif_features, axis=1, inplace=True)
X_val.drop(high_vif_features, axis=1, inplace=True)
X_test.drop(high_vif_features, axis=1, inplace=True)
```

```
No of high VIF featues: 0
high VIF features: Series([], Name: Features, dtype: object)
```

In [ ]:

using elastic net regression (combination of L1 and L2 regression) and hyperparameter tuning

In [23]:
```python
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import GridSearchCV

# Define the Elastic Net model
elastic_net = ElasticNet()

# Define the hyperparameters to tune
param_grid = {
    'alpha': [0.007,0.04,0.06,0.07,0.13,0.145],  # Regularization strength
    'l1_ratio': [0.09,0.1,0.11,0.13,0.14]  # Mix ratio between L1 and L2 penalties
}

# Perform GridSearchCV
grid_search = GridSearchCV(estimator=elastic_net, param_grid=param_grid, scoring='r2', cv=5)
grid_search.fit(X_train_val, y_train_val)

# Print the best hyperparameters
print("Best hyperparameters:", grid_search.best_params_)

# Get the best model
best_elastic_net = grid_search.best_estimator_

# Evaluate the best model on the validation set
best_elastic_net_r2_score = best_elastic_net.score(X_val, y_val)
n = len(X_val)
p = X_val.shape[1]
best_elastic_net_adj_r2_score = 1 - (1 - best_elastic_net_r2_score) * (n - 1) / (n - p - 1)

print("\nBest Elastic Net Regression Model Scores:")
```

```
print("R^2 Score:", best_elastic_net_r2_score)
print("Adjusted R^2 Score:", best_elastic_net_adj_r2_score)
```

```
Best hyperparameters: {'alpha': 0.007, 'l1_ratio': 0.09}

Best Elastic Net Regression Model Scores:
R^2 Score: 0.84862396737448
Adjusted R^2 Score: 0.8371062257616687
```

In [24]:

```
# Use the hyperparameters obtained from GridSearchCV
best_hyperparams = grid_search.best_params_
alpha = best_hyperparams['alpha']
l1_ratio = best_hyperparams['l1_ratio']

# Define the best Elastic Net model with the hyperparameters
best_model = ElasticNet(alpha=alpha, l1_ratio=l1_ratio)

# Fit the model on the training data
best_model.fit(X_train, y_train)

# Evaluate the best model on the validation set
r2_score = best_model.score(X_val, y_val)
print("R2 score:", r2_score)

n = len(X_val)
p = X_val.shape[1]
adj_r2_score = 1 - (1 - r2_score) * (n - 1) / (n - p - 1)
print("Adjusted R2 score:", adj_r2_score)
```

```
R2 score: 0.839577611359208
Adjusted R2 score: 0.8273715600495826
```

In [25]:

```
# best model on test data
best_hyperparams = grid_search.best_params_
alpha = best_hyperparams['alpha']
l1_ratio = best_hyperparams['l1_ratio']

# Define the best Elastic Net model with the hyperparameters
best_model = ElasticNet(alpha=alpha, l1_ratio=l1_ratio)

# Fit the model on the training data
best_model.fit(X_train, y_train)

# Evaluate the best model on the validation set
r2_score = best_model.score(X_test, y_test)
print("R2 score:", r2_score)

n = len(X_val)
p = X_val.shape[1]
adj_r2_score = 1 - (1 - r2_score) * (n - 1) / (n - p - 1)
print("Adjusted R2 score:", adj_r2_score)
```

```
R2 score: 0.795242957635302
Adjusted R2 score: 0.7796636174553793
```

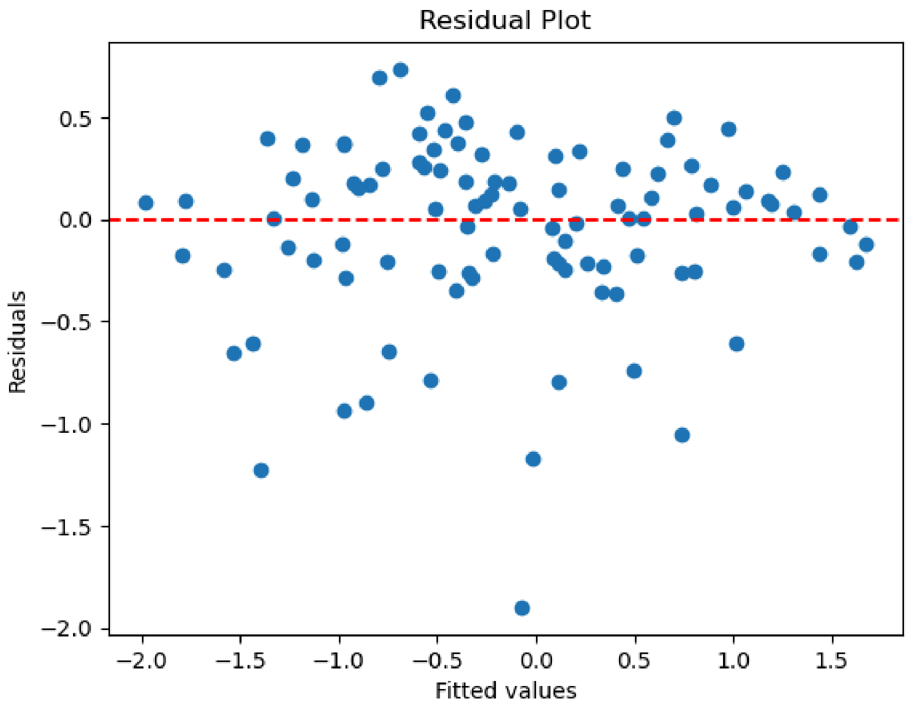
In [26]:

```
# Calculate residuals
residuals = y_test - best_model.predict(X_test)

# Check mean of residuals
mean_residuals = np.mean(residuals)
print("Mean of Residuals:", mean_residuals)
```

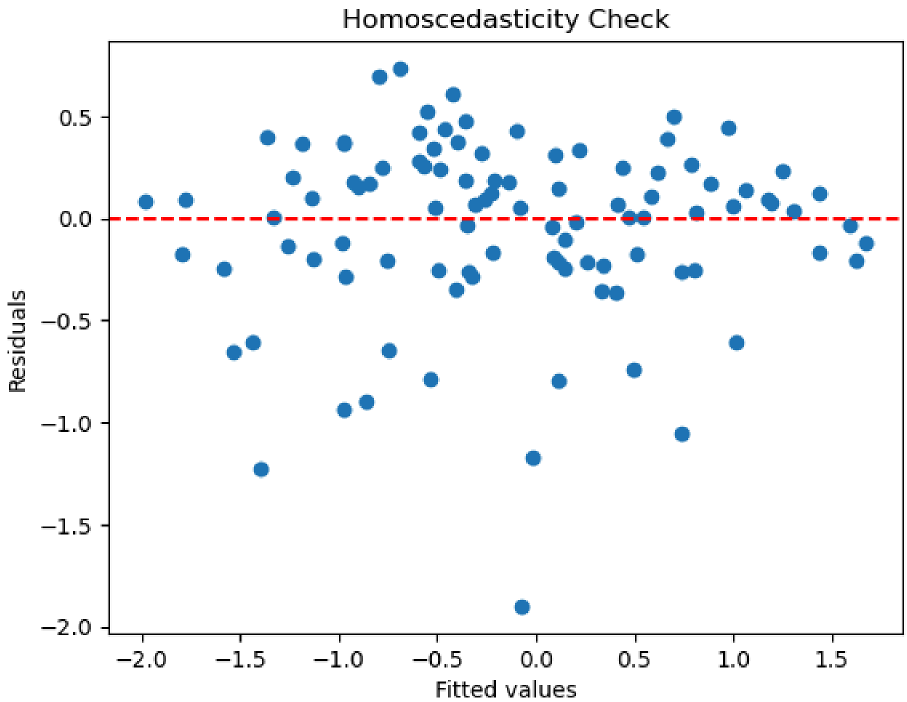
```
Mean of Residuals: -0.041755034726234534
```

In [27]:

```
# Plot residual plots
plt.scatter(best_model.predict(X_test), residuals)
plt.xlabel("Fitted values")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```

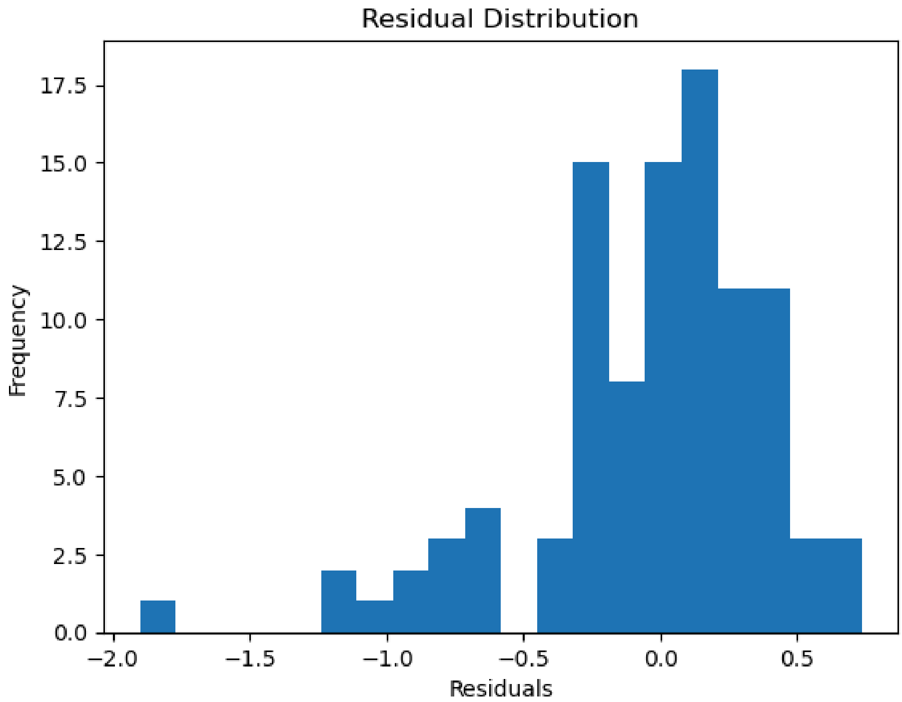


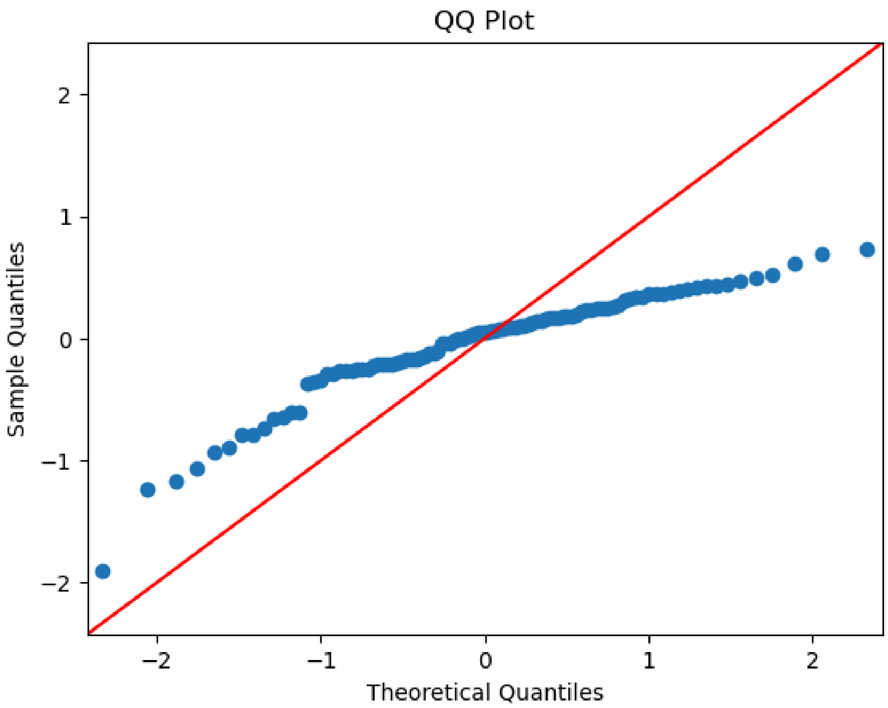
In [28]:
```
# Plot residuals vs. fitted values
plt.scatter(best_model.predict(X_test), residuals)
plt.xlabel("Fitted values")
plt.ylabel("Residuals")
plt.title("Homoscedasticity Check")
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



In [29]:
```
# Plot histogram of residuals
plt.hist(residuals, bins=20)
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.title("Residual Distribution")
plt.show()

# QQ plot
import statsmodels.api as sm
sm.qqplot(residuals, line ='45')
```

```
plt.title("QQ Plot")
plt.show()
```





Insights:

- The dataset has 500 records and 9 features
- The final regression model after hyperparameter tuning had an R2 score of 0.7952 and an adjusted R2 score of 0.779 compared to the validation results of an R2 score of 0.839 and an adjusted R2 score of 0.827, which is not very off.
- GRE score, TOEFL score, and CGPA have a normal distribution tendency from the plot. Chance of admit is right-skewed.
- The mean of residuals is around -0.041.
- The residuals satisfy homoscedasticity.
- The residuals plot/distribution is right-skewed and does not seem to follow a normal distribution perfectly. This is due to the lesser number of datapoints, which makes it harder to ascertain the distribution. Moreover, the data was split into test, train, and validation datasets for hyperparameter tuning, which might have contributed to it.

Business Recommendations:

- Given the model's performance with an R2 score of 0.7952 and an adjusted R2 score of 0.779, it suggests that the selected features are reasonably effective in predicting the chance of admission. We can Consider leveraging these features for future admissions decision-making processes.
- Although the model's performance is relatively good, there is still room for improvement, as indicated by the higher R2 score and adjusted R2 score obtained during validation. Explore additional features or refine existing ones to enhance the model's predictive accuracy.
- Given that GRE score, TOEFL score, and CGPA demonstrate a normal distribution tendency, they appear to be reliable predictors of admission chances. Consider placing more emphasis on these factors during the admissions evaluation process.
- Take into account the right-skewed distribution of the chance of admission variable. This suggests that a significant portion of applicants may have higher admission probabilities, potentially indicating a competitive applicant pool or the need for stricter admission criteria.
- Although the residuals satisfy homoscedasticity, the right-skewed distribution of residuals indicates potential deviations from normality. To improve model robustness, consider collecting more data to ensure a more representative sample and conducting further analyses to confirm the distribution's characteristics.
- When making admissions decisions, consider not only the individual predictors but also their combined effects. Evaluate applicants holistically, taking into account all relevant factors identified by the model.