Jamboree Case Study

OVERVIEW

Jamboree is a company which helps students to get admission in top collages abroad. Be it GMAT, GRE or SAT, their unique problem-solving methods ensure maximum scores with minimum effort.

Datset - The datset contains the unique record of each student and features lik GRE Score,TOEFL Score, University rating, SOP, LOR, CGPA, Research and Chance of Admit.

We have to understand 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.

- Did Exploratory Data Analysis and provided insights.
- Made the Linear Regression model to detect the chance of admission given the other feature.
- Tested all the assumption of linear regrssion like linearity, multicollineatiyt , homoscedacticity, mean of residuals and normaility of the residuals.

Imorting libraries

In [615]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.linear_model import LinearRegression,Lasso,Ridge
from sklearn.model_selection import  train_test_split,KFold
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
from tabulate import tabulate
import statsmodels.api as sm
```

Reading the dataset as pandas dataframe

In [213]:
```
df = pd.read_csv("d2beiqkhq929f0.cloudfront.net_public_assets_assets_000_001_
```

Exploratory Data Analysis

In [3]:
```
df.head()
```

Out[3]:

	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 [26]:
```
df.shape
```

Out[26]: (500, 9)

In [7]:
```
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
```

Insights

- No Null values in the data and all the datatypes are correct.

In [8]:
```
df.describe()
```

Out[8]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Re
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.
mean	250.500000	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.
std	144.481833	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.
25%	125.750000	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.
50%	250.500000	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.
75%	375.250000	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.
max	500.000000	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.

In [199]:
```
df.columns
```

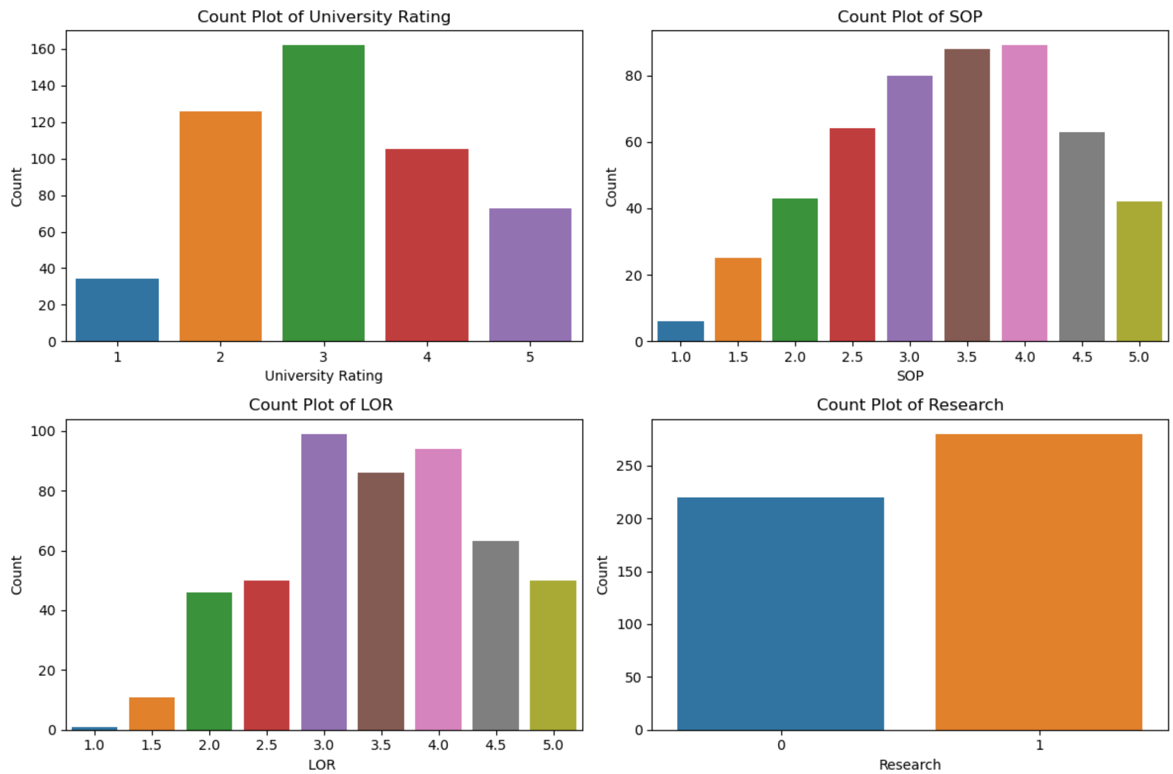
Out[199]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

In [204]:
```
categorical_columns = ['University Rating','SOP','LOR ','Research']
numerical_columns = ['GRE Score','TOEFL Score','CGPA','Chance of Admit ']
```

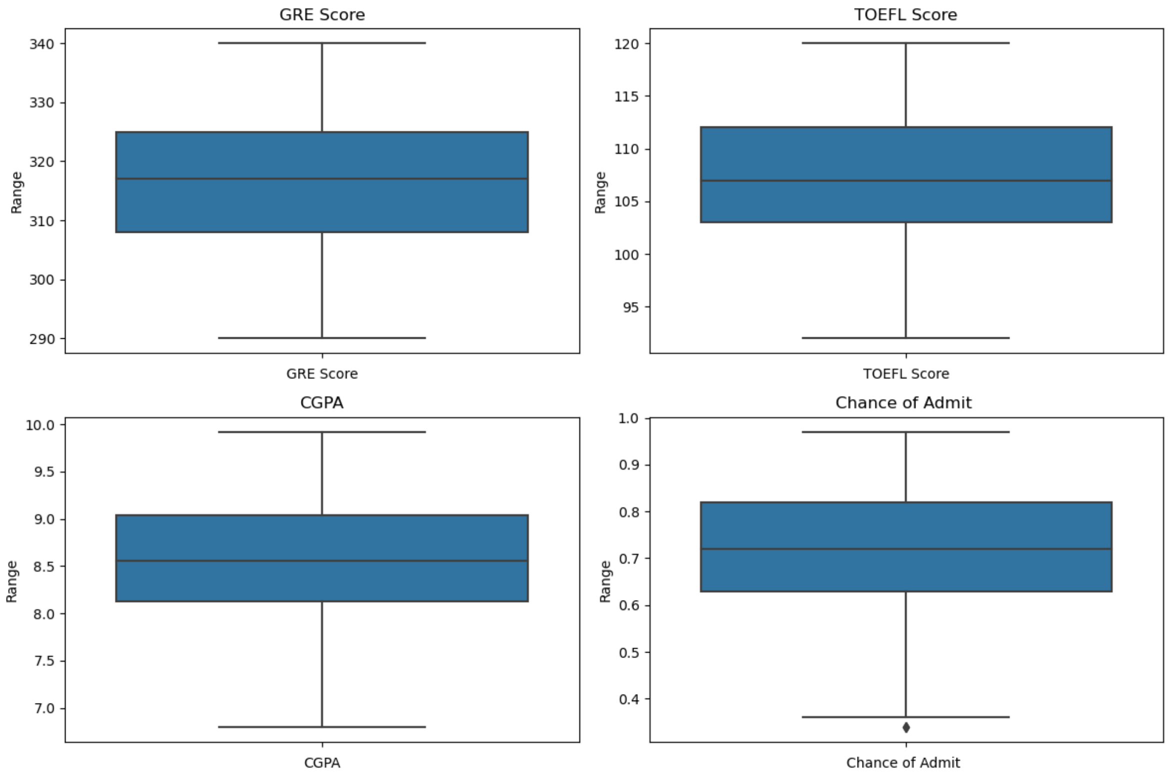
In [185]:
```
count_values = {}
for col in categorical_columns:
    count_values[col] = df[col].value_counts()

# Print the formatted table for multiple columns
for col, counts in count_values.items():
    print(f"Count values for '{col}':")
    print(tabulate(counts.reset_index(), headers=['Value', 'Count'], tablefmt
    print()
```

```
Count values for 'University Rating':
+---+-------+-------+
|   | Value | Count |
+---+-------+-------+
| 0 |   3   |  162  |
| 1 |   2   |  126  |
| 2 |   4   |  105  |
| 3 |   5   |  73   |
| 4 |   1   |  34   |
+---+-------+-------+

Count values for 'SOP':
+---+-------+-------+
|   | Value | Count |
+---+-------+-------+
| 0 |  4.0  | 89.0  |
| 1 |  3.5  | 88.0  |
| 2 |  3.0  | 80.0  |
| 3 |  2.5  | 64.0  |
| 4 |  4.5  | 63.0  |
| 5 |  2.0  | 43.0  |
| 6 |  5.0  | 42.0  |
| 7 |  1.5  | 25.0  |
| 8 |  1.0  |  6.0  |
+---+-------+-------+

Count values for 'LOR ':
+---+-------+-------+
|   | Value | Count |
+---+-------+-------+
| 0 |  3.0  | 99.0  |
| 1 |  4.0  | 94.0  |
| 2 |  3.5  | 86.0  |
| 3 |  4.5  | 63.0  |
| 4 |  2.5  | 50.0  |
| 5 |  5.0  | 50.0  |
| 6 |  2.0  | 46.0  |
| 7 |  1.5  | 11.0  |
| 8 |  1.0  |  1.0  |
+---+-------+-------+

Count values for 'Research':
+---+-------+-------+
|   | Value | Count |
+---+-------+-------+
| 0 |   1   |  280  |
| 1 |   0   |  220  |
+---+-------+-------+
```

In [173]:
```
plt.figure(figsize=(12,8))
for col,i in zip(categorical_columns,range(1,5)):
    plt.subplot(2,2,i)
    sns.countplot(data=df, x=col)
    plt.title(f'Count Plot of {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
plt.tight_layout()
plt.show()
```



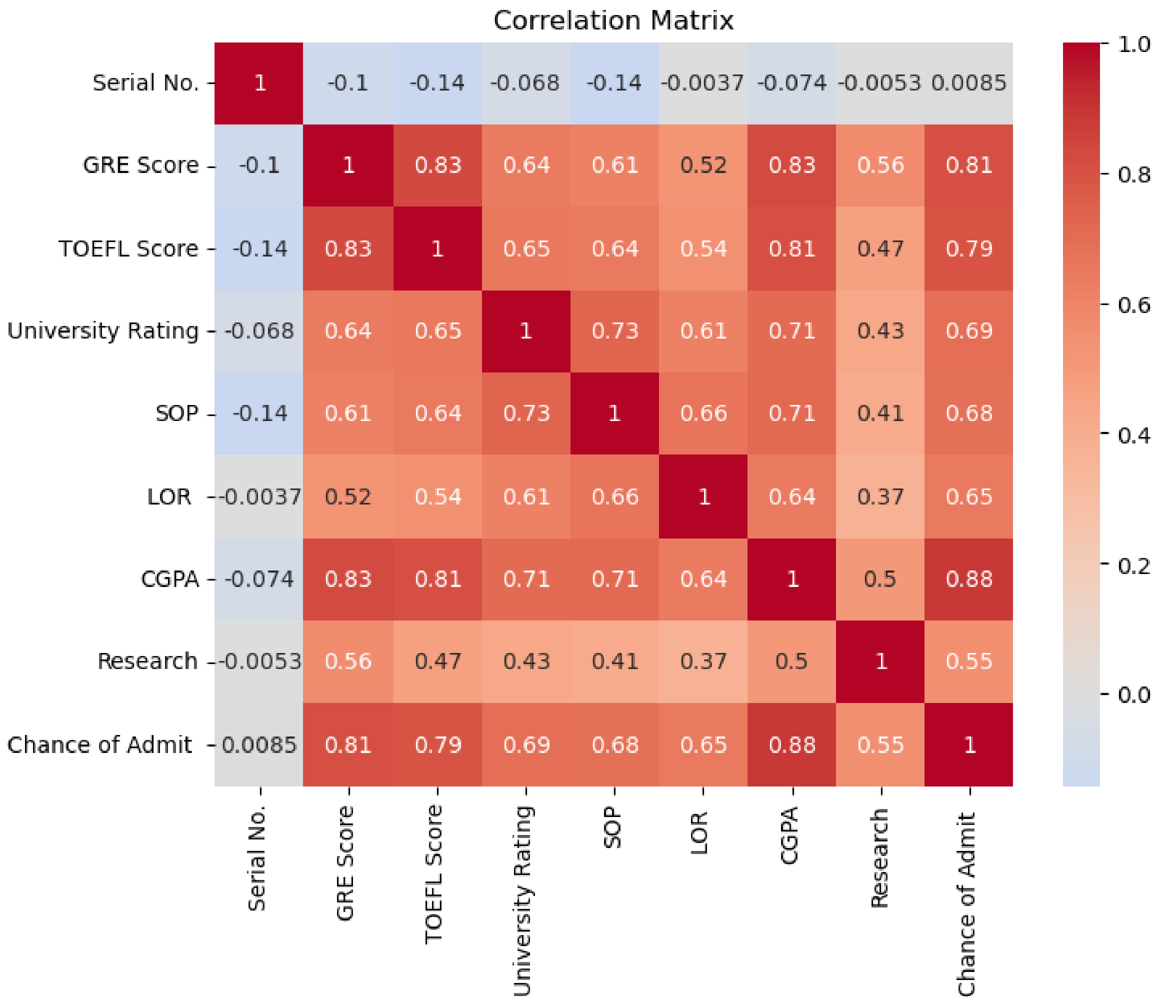
In [209]:
```
plt.figure(figsize=(12,8))
for col,i in zip(numerical_columns,range(1,5)):
    plt.subplot(2,2,i)
    sns.boxplot(data=df, y=col)
    plt.title(f'{col}')
    plt.xlabel(col)
    plt.ylabel('Range')
plt.tight_layout()
plt.show()
```



Insights

- No Outliers in all the continuous variables
- Most of the studnets have good CGPA which is grater than 6.5 and 75% have grater than 8.

In [10]:
```
correlation_matrix = df.corr()
# Create a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title("Correlation Matrix")
plt.show()
```



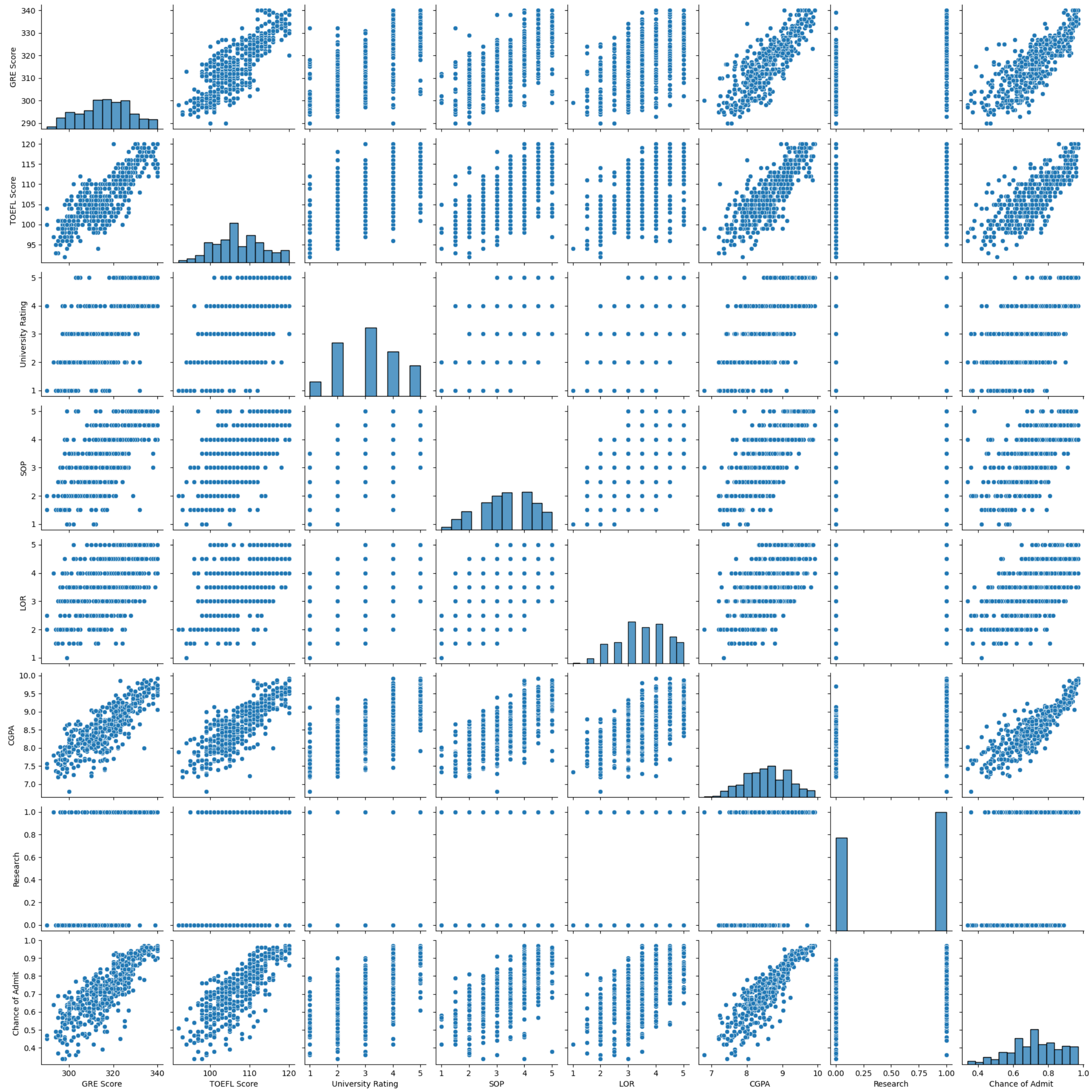
	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
Serial No.	1	-0.1	-0.14	-0.068	-0.14	-0.0037	-0.074	-0.0053	0.0085
GRE Score	-0.1	1	0.83	0.64	0.61	0.52	0.83	0.56	0.81
TOEFL Score	-0.14	0.83	1	0.65	0.64	0.54	0.81	0.47	0.79
University Rating	-0.068	0.64	0.65	1	0.73	0.61	0.71	0.43	0.69
SOP	-0.14	0.61	0.64	0.73	1	0.66	0.71	0.41	0.68
LOR	-0.0037	0.52	0.54	0.61	0.66	1	0.64	0.37	0.65
CGPA	-0.074	0.83	0.81	0.71	0.71	0.64	1	0.5	0.88
Research	-0.0053	0.56	0.47	0.43	0.41	0.37	0.5	1	0.55
Chance of Admit	0.0085	0.81	0.79	0.69	0.68	0.65	0.88	0.55	1

Insights

- All the variable are postively correlated to each othe.
- Chance of Admit is postively correlated with every feature but most correlated to CGPA by 0.88.
- If we just focus just on CGPA it is mostly correlated to every other feature more than any other feature

In [214]:
```
df.drop('Serial No.',axis = 1,inplace = True) # no need of SeiralNo. as it is
```

In [215]:
```
sns.pairplot(df)
plt.show()
```



Data Preprocessing

As we already checked there are no outliers and missing values. Lets check duplicate values.

In [217]:
```
df.duplicated().sum()  # No duplicate columns in the data
```

Out[217]:
```
0
```

In [220]:
```
target = df['Chance of Admit ']
```

In [221]:
```
scaler = MinMaxScaler()  # used min max scaler for scaling the data
```

In [222]:
```
df_ = pd.DataFrame(df_,columns = df.columns)
```

In [225]:
```
X = df_.drop(['Chance of Admit '], axis=1)   # Removed the target feature
```

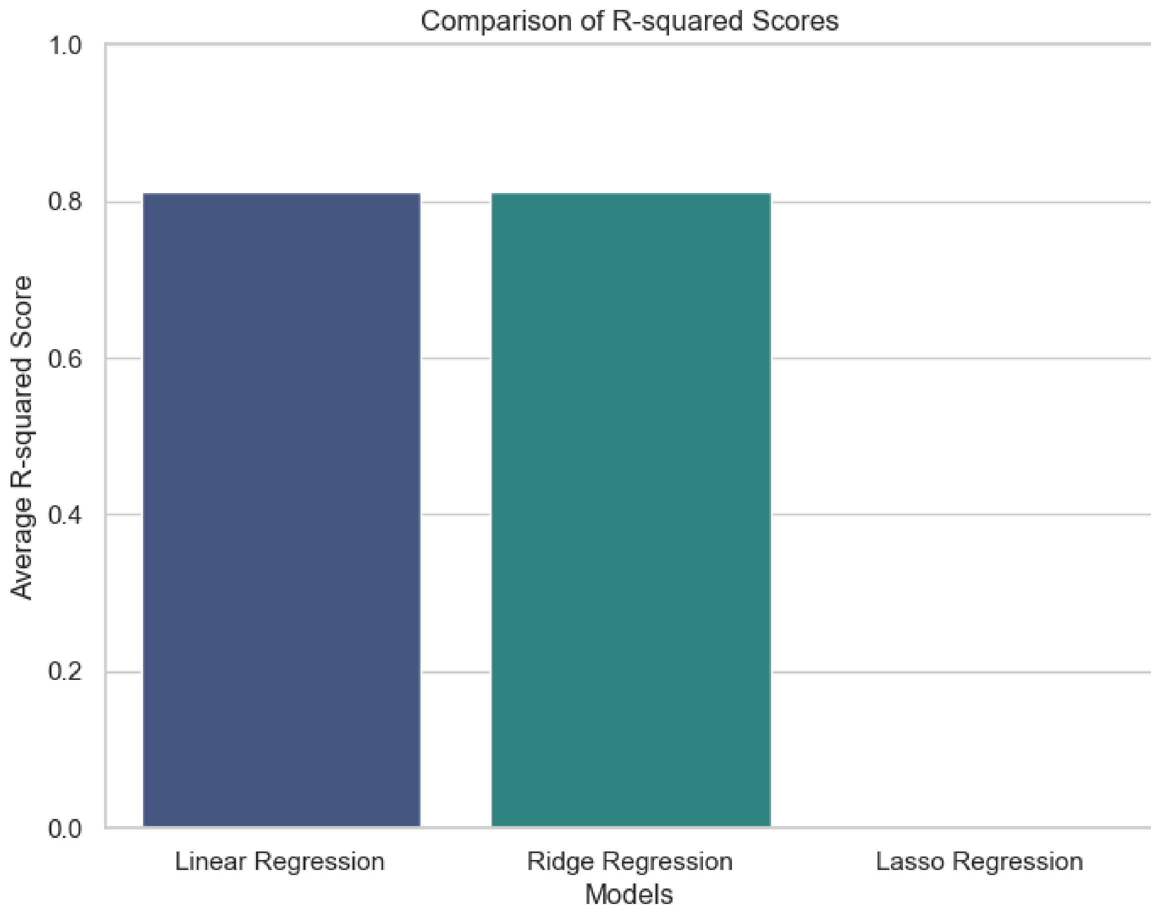
Model building

Used K-fold cross validation to access the performance and generalization of Model. Also the data contains only 500 datapoints therfore it become appropriate to use k-fold cross validation.

In [403]:
```
linear_model = LinearRegression()
ridge_model = Ridge(alpha=1.0)
lasso_model = Lasso(alpha=1.0)

# Perform k-fold cross-validation
k = 10  # Number of folds
kf = KFold(n_splits=k)

ridge_mse_scores = []
lasso_mse_scores = []
linear_mse_scores = []

linear_r2_scores = []
ridge_r2_scores = []
lasso_r2_scores = []

for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index,:],X.iloc[test_index,:]
    y_train, y_test = target[train_index], target[test_index]

    # Linear Regression
    linear_model.fit(X_train, y_train)
    linear_predictions = linear_model.predict(X_test)
    linear_r2 = r2_score(y_test, linear_predictions)
    linear_r2_scores.append(linear_r2)

    # Ridge Regression
    ridge_model.fit(X_train, y_train)
    ridge_predictions = ridge_model.predict(X_test)
    ridge_r2 = r2_score(y_test, ridge_predictions)
    ridge_r2_scores.append(ridge_r2)

    # Lasso Regression
    lasso_model.fit(X_train, y_train)
    lasso_predictions = lasso_model.predict(X_test)
    lasso_r2 = r2_score(y_test, lasso_predictions)
    lasso_r2_scores.append(lasso_r2)

linear_avg_r2 = np.mean(linear_r2_scores)
ridge_avg_r2 = np.mean(ridge_r2_scores)
lasso_avg_r2 = np.mean(lasso_r2_scores)


print("Average Linear Regression R-squared:", linear_avg_r2)
print("Average Ridge Regression R-squared:", ridge_avg_r2)
print("Average Lasso Regression R-squared:", lasso_avg_r2)

models = ['Linear Regression', 'Ridge Regression', 'Lasso Regression']
scores = [linear_avg_r2, ridge_avg_r2, lasso_avg_r2]

sns.set(style="whitegrid")
plt.figure(figsize=(8, 6))
sns.barplot(x=models, y=scores, palette="viridis")
plt.xlabel('Models')
plt.ylabel('Average R-squared Score')
plt.title('Comparison of R-squared Scores')
plt.ylim(0, 1)  # Set y-axis limit to range from 0 to 1 (R-squared range)
```

```
plt.show()
```

```
Average Linear Regression R-squared: 0.8131223770253243
Average Ridge Regression R-squared: 0.8113519941286784
Average Lasso Regression R-squared: -0.07653593730382247
```



We will go with Linear Regression as there is no much difference between the mean square value and r2_score.

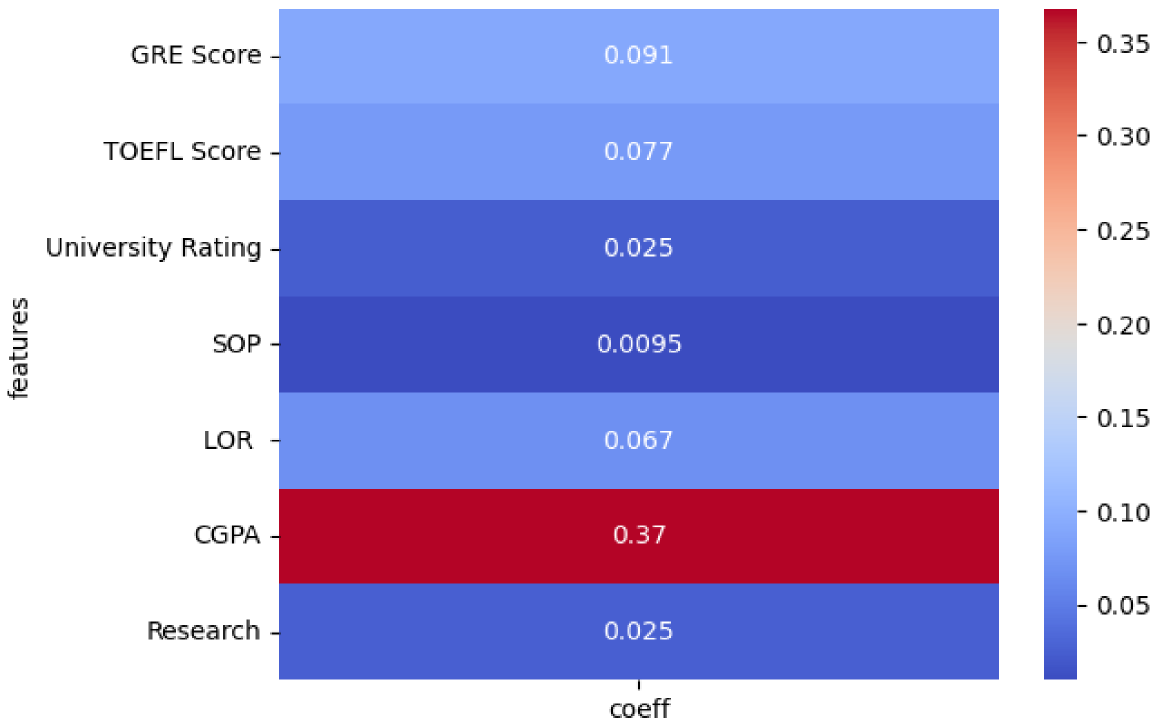
Now lets train the model on all the datapoints as we already tested the model using k-fold cross validation.

In [374]:
```
Model = LinearRegression()
Model.fit(X,target)
```

Out[374]:
```
LinearRegression()
```

In [386]:
```
coeff = pd.DataFrame({"features":X.columns,"coeff":model.coef_})
coeff.index = coeff['features']
coeff.drop('features',axis = 1,inplace = True)
sns.heatmap(coeff,annot = True,cmap = 'coolwarm')
```

Out[386]:
```
<AxesSubplot:ylabel='features'>
```



Got a good r2 score of 0.76. Lets also chek adj r2 score

In [256]:
```
def adj_r2(r2,X):
    return (1 - ((1-r2)*(len(X)-1))/(len(X)-X.shape[1]-1))
```

In [258]:
```
adj_r2(r2_score,X)  # Not much different from r2 score
```

Out[258]:
```
0.76174627830761
```

Testing the assumptions of the linear regression model

Multicollinearity check by VIF score

In [413]:
```
vif_data = pd.DataFrame()
vif_data["Feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.sh
vif_data.sort_values('VIF',inplace = True,ascending = False)
# Print VIF scores in a formatted table
print(vif_data)
```

```
             Feature        VIF
5               CGPA  41.461741
0          GRE Score  29.024693
1        TOEFL Score  28.124993
3                SOP  19.007718
4                LOR  15.048490
2  University Rating  11.101366
6           Research   3.344455
```

This shows high multicollinearity in most of rows. Now lets remove one by one.

In [415]:
```
X_= X.copy(deep = True)
X_.drop('CGPA',axis = 1,inplace = True)
vif_data = pd.DataFrame()
vif_data["Feature"] = X_.columns
vif_data["VIF"] = [variance_inflation_factor(X_.values, i) for i in range(X_.
vif_data.sort_values('VIF',inplace = True,ascending = False)
# Print VIF scores in a formatted table
print(vif_data)
```

```
             Feature        VIF
1        TOEFL Score  25.147554
0          GRE Score  24.107055
3                SOP  18.085424
4                LOR  13.279097
2  University Rating  11.024466
5           Research   3.343648
```

In [417]:
```
X_.drop('TOEFL Score',axis = 1,inplace = True)
vif_data = pd.DataFrame()
vif_data["Feature"] = X_.columns
vif_data["VIF"] = [variance_inflation_factor(X_.values, i) for i in range(X_.
vif_data.sort_values('VIF',inplace = True,ascending = False)
# Print VIF scores in a formatted table
print(vif_data)
```

```
             Feature        VIF
2                SOP  17.431031
3                LOR  12.970243
0          GRE Score  12.662108
1  University Rating  10.900990
4           Research   3.328578
```

In [418]:
```
X_.drop('SOP',axis = 1,inplace = True)
vif_data = pd.DataFrame()
vif_data["Feature"] = X_.columns
vif_data["VIF"] = [variance_inflation_factor(X_.values, i) for i in range(X_.
vif_data.sort_values('VIF',inplace = True,ascending = False)
# Print VIF scores in a formatted table
print(vif_data)
```

```
             Feature        VIF
0          GRE Score  11.714001
2                LOR   9.725113
1  University Rating   8.984274
3           Research   3.328459
```

In [419]:
```
X_.drop('GRE Score',axis = 1,inplace = True)
vif_data = pd.DataFrame()
vif_data["Feature"] = X_.columns
vif_data["VIF"] = [variance_inflation_factor(X_.values, i) for i in range(X_.
vif_data.sort_values('VIF',inplace = True,ascending = False)
# Print VIF scores in a formatted table
print(vif_data)
```

```
             Feature       VIF
0  University Rating  7.535720
1                LOR  7.363200
2           Research  2.848363
```

In [548]:
```
X_.drop('University Rating',axis = 1,inplace = True)
vif_data = pd.DataFrame()
vif_data["Feature"] = X_.columns
vif_data["VIF"] = [variance_inflation_factor(X_.values, i) for i in range(X_.
vif_data.sort_values('VIF',inplace = True,ascending = False)
# Print VIF scores in a formatted table
print(vif_data)
```

```
    Feature       VIF
0       LOR  2.632925
1  Research  2.632925
```

In [580]:
```
linear_r2_scores_VIF = []

for train_index, test_index in kf.split(X_):
    X_train, X_test = X_.iloc[train_index,:],X_.iloc[test_index,:]
    y_train, y_test = target[train_index], target[test_index]
    # Linear Regression
    linear_model.fit(X_train, y_train)
    linear_predictions = linear_model.predict(X_test)
    linear_r2 = r2_score(y_test, linear_predictions)
    linear_r2_scores_VIF.append(linear_r2)

print(np.mean(linear_r2_scores_VIF))
```

```
0.48760389349688377
```

After removing features having high multicollinearity.We got r2 score of 0.48.

we will make the model after removing all the multicollinearity features even if it gives less r2 score because stability matter s more than accuracy.

Lets train the model again

In [582]:
```
Model_ = LinearRegression()
Model_.fit(X_,target)
```

Out[582]:
```
LinearRegression()
```

In [584]:
```
Model_.coef_
```

Out[584]:
```
array([0.31309616, 0.10074623])
```

In [585]:
```
Model_.intercept_
```

Out[585]:
```
0.47088940046840483
```

Mean of Residuals

In [588]:
```
predicted_ = Model_.predict(X_)
```

In [586]:
```
residuals = target - Model_.predict(X_)
```

In [587]:
```
np.mean(residuals)
```

Out[587]:
```
-4.2521541843143496e-17
```

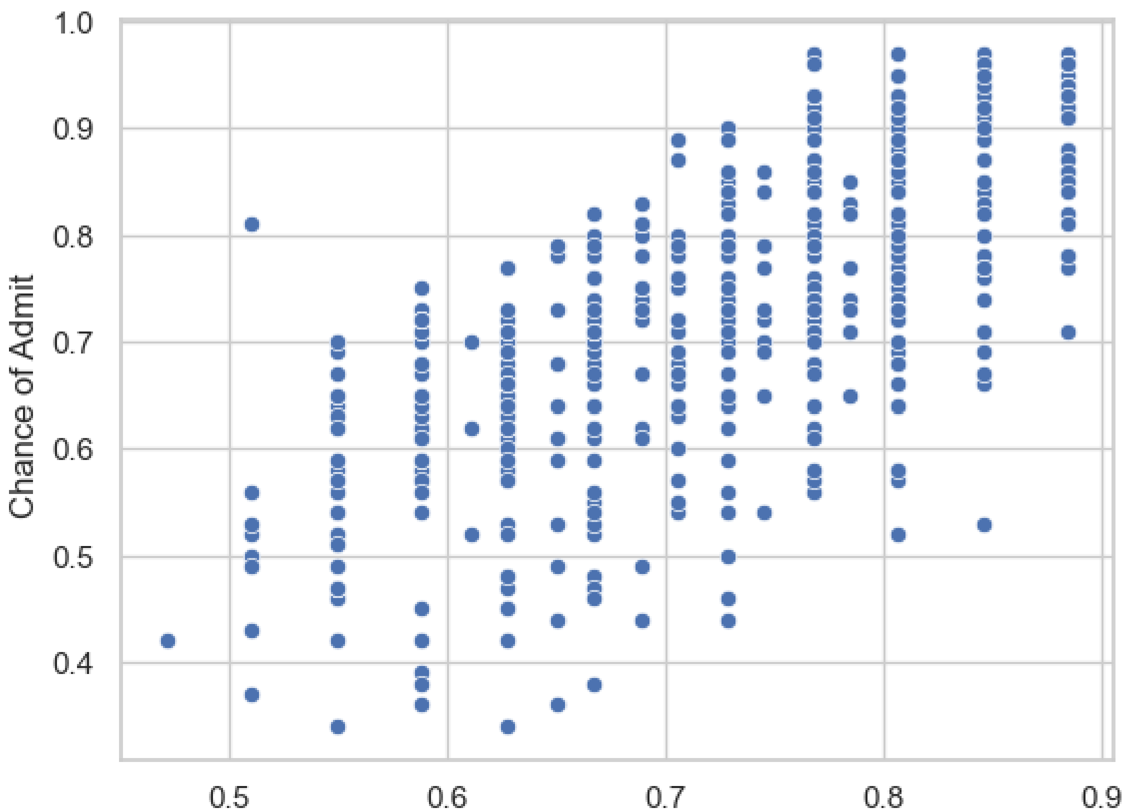
Threfore it satisfy the assumption of mean of reiduals near to zero

Linearity of variables

We have seen earlier that all the varibales are linearly correlated to the target variable

In [622]:
```
sns.scatterplot(x = predicted_,y = target)
```

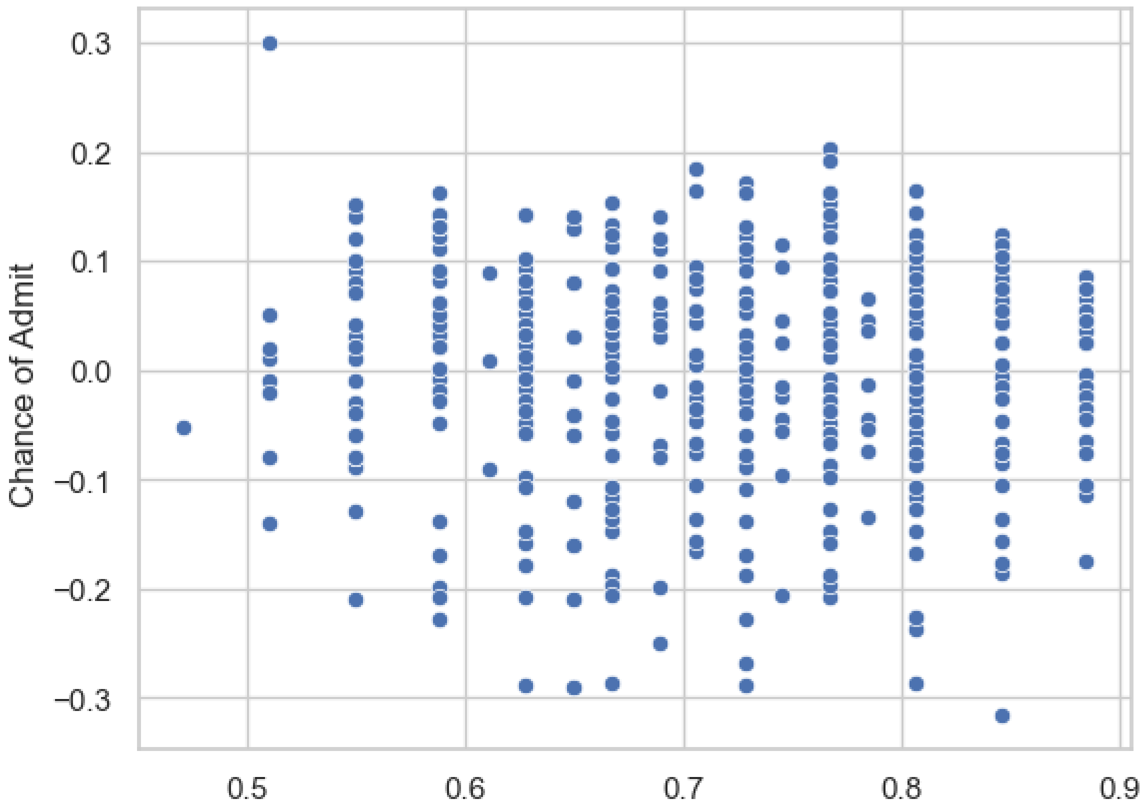
Out[622]: <AxesSubplot:ylabel='Chance of Admit '>



Test for Homoscedasticity

In [593]:
```
sns.scatterplot(y= residuals,x = predicted_)
```

Out[593]: <AxesSubplot:ylabel='Chance of Admit '>

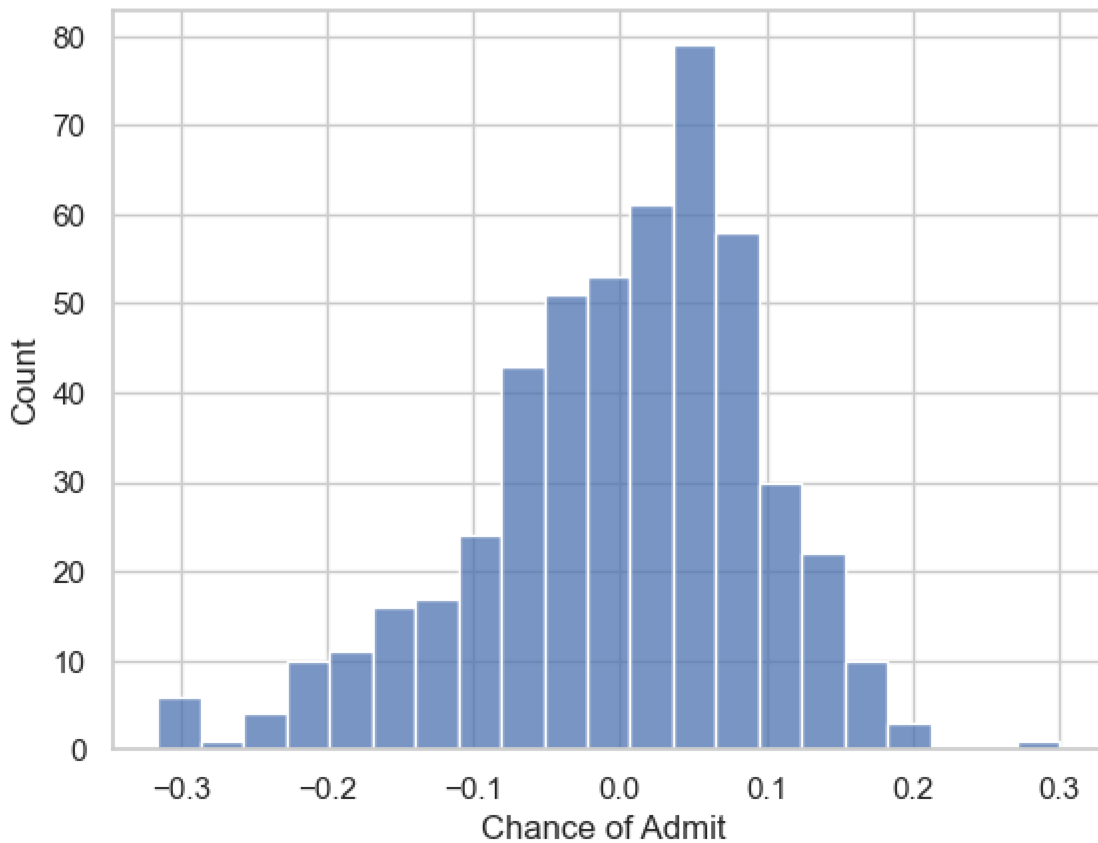


We can see from the graph that it follows homoscedsticity as teh mean is zero and equal varince across all teh predicted valiues.

Normality of residuals

In [596]:
```
sns.histplot(residuals)
```

Out[596]: <AxesSubplot:xlabel='Chance of Admit ', ylabel='Count'>



In [611]:
```
statistic, p_value = stats.shapiro(residuals)

# Set significance level
alpha = 0.05

# Check the p-value against the significance level
if p_value > alpha:
    print("Data follows a normal distribution (fail to reject H0)")
else:
    print("Data does not follow a normal distribution (reject H0)")
```

```
Data does not follow a normal distribution (reject H0)
```

Residual do not follow normality . This could be due to less no. of data points in the data.

Model performance evaluation

In [619]:
```
linear_r2_scores_VIF = []
mae= []
mse= []

for train_index, test_index in kf.split(X_):
    X_train, X_test = X_.iloc[train_index,:],X_.iloc[test_index,:]
    y_train, y_test = target[train_index], target[test_index]
    # Linear Regression
    linear_model.fit(X_train, y_train)
    linear_predictions = linear_model.predict(X_test)

    linear_r2 = r2_score(y_test, linear_predictions)
    linear_r2_scores_VIF.append(linear_r2)

    linear_mae = mean_absolute_error(y_test, linear_predictions)
    mae.append(linear_mae)

    linear_mse = mean_squared_error(y_test,linear_predictions)
    mse.append(linear_mse)

print("R2 Score:",np.mean(linear_r2_scores_VIF))
print("Mean Absolute Error:",np.mean(mae))
print("Mean Square Error:",np.mean(mse))
```

```
R2 Score: 0.48760389349688377
Mean Absolute Error: 0.0784072366645702
Mean Square Error: 0.009866383954050496
```

```
The model can be improve if we got more data.
```

Actionable Insights & Recommendations

- As we can see most of the feartures are coorelated to each other. which suggests that we can take only a few featueres and make a good model. Like we can also train the model by just using teh CGPA featuer as it is most postively correlated to the target variable.
- We selected LOR and Research as our final features for model training because of correlation between other features.This helps in making the model stable.

Recommedations-

- This model can be use by the company to get the chance of Admit for student even if we got few features data from the student.
- The model can improve more if we get more data points from students.
- If Fewer data points are not provided by student still we can detect the chance of admit to good accuracy as most of the features are highly correlated to each other.