Problem Statement

Jamboree has helped thousands of students 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.

Objective of analysis is:

- To help Jamboree in understanding what factors are important in graduate admissions.
- How these factors are interrelated among themselves.
- Help predict one's chances of admission given the rest of the variables.

Column Profiling:

- Serial No. (Unique row ID)
- GRE Scores (out of 340)
- TOEFL Scores (out of 120)
- University Rating (out of 5)
- Statement of Purpose and Letter of Recommendation Strength (out of 5)
- Undergraduate GPA (out of 10)
- Research Experience (either 0 or 1)
- Chance of Admit (ranging from 0 to 1)

Loading dependencies and dataset

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

from scipy.stats import levene, f_oneway, kruskal
from scipy.stats import ttest_ind
from scipy.stats import chi2_contingency
from statsmodels.graphics.gofplots import qqplot

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

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

In [2]:
```
df = pd.read_csv('./data/jamboree.csv')
df.head()
```

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

Basic Checks on the data

In [3]:
```
# Removing spaces from column names
print(df.columns)
print('-'*100)
```

```
df.columns = [x.strip() for x in df.columns]
print(df.columns)
```

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

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

Out[4]:
```
(500, 9)
```

In [5]:
```
# All features are numeric
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]:
```
# We will drop the feature 'Serial No.' since it does not add any practical value to our problem st
df.drop(labels='Serial No.', axis=1, inplace=True)
df.head()
```

Out[6]:

	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 [7]:
```
# Detect missing values in data
df.isna().sum()
```

Out[7]:
```
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

In [8]:
```
# Number of unique values per feature
for col in df.columns:
    print(col, ':', df[col].nunique())
```

```
GRE Score : 49
TOEFL Score : 29
University Rating : 5
SOP : 9
LOR : 9
CGPA : 184
Research : 2
Chance of Admit : 61
```

In [9]:
```
# Descriptive statistics on the features in our data
df.describe()
```

Out[9]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.00000
mean	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	0.72174
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	0.14114
min	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.34000
25%	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.63000
50%	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.72000
75%	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.82000
max	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.97000

EDA

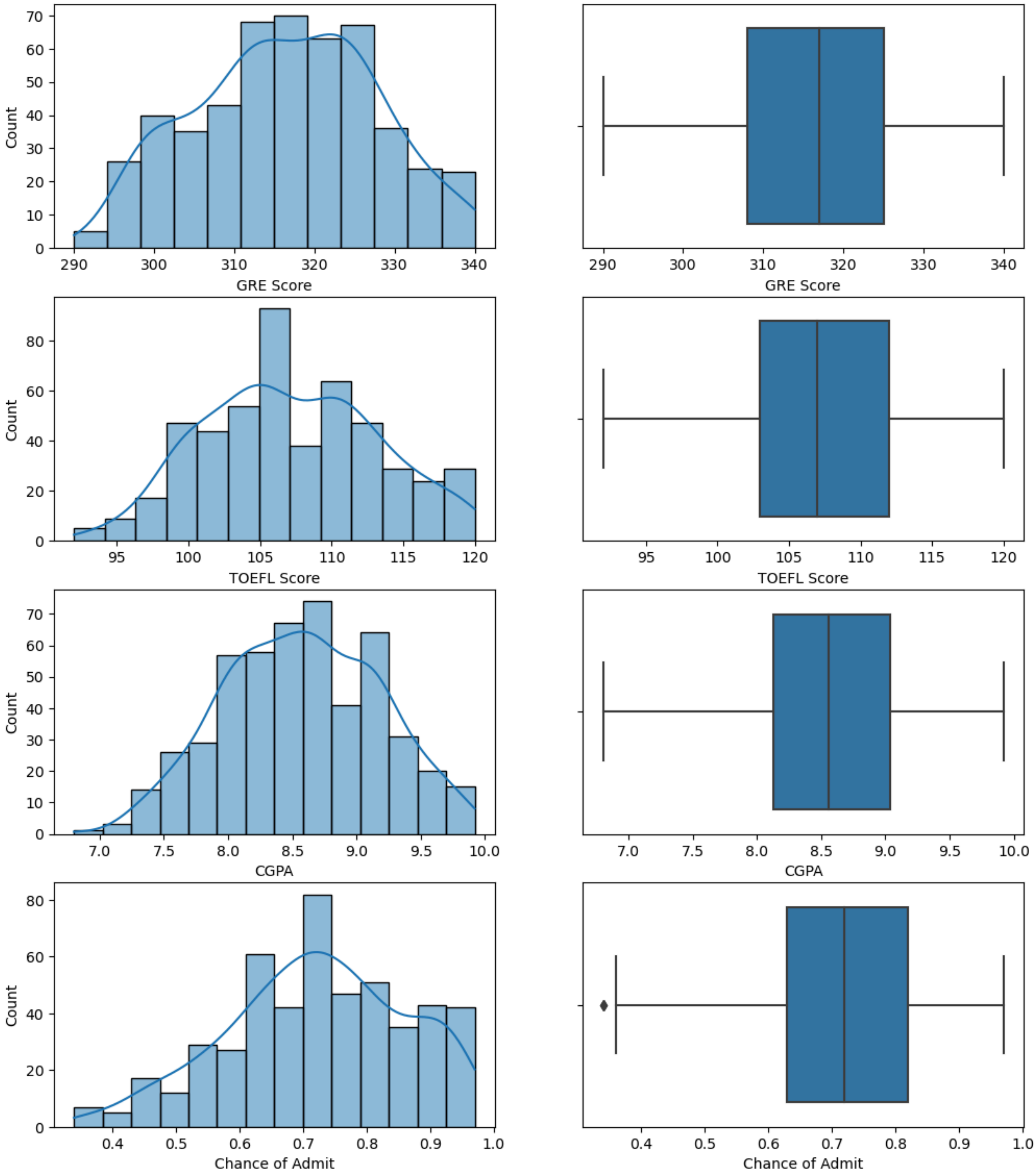
Univariate Analysis

In [10]:

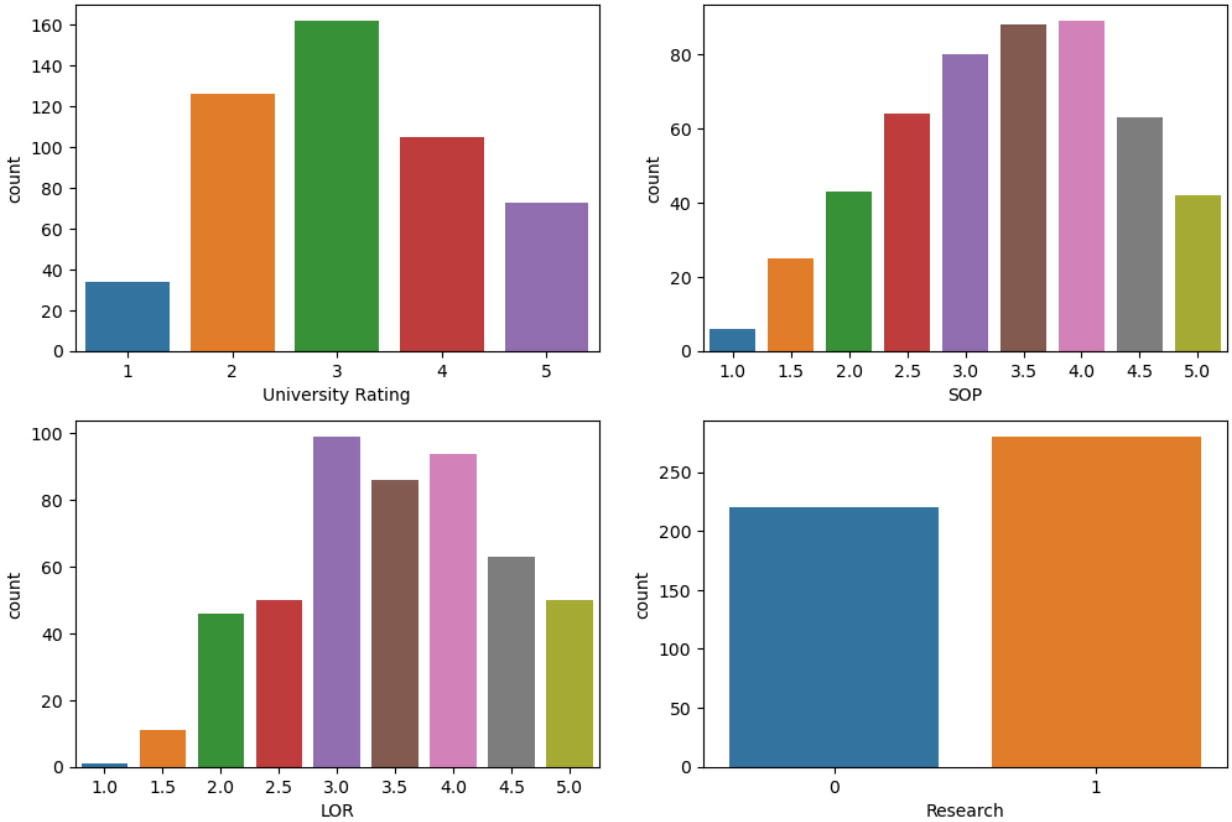
```
cols_cont = ['GRE Score', 'TOEFL Score', 'CGPA', 'Chance of Admit']
cols_disc = ['University Rating', 'SOP', 'LOR', 'Research']
```

In [11]:

```
plt.figure(figsize=(12, 14))
i = 1
for col in cols_cont:
    plt.subplot(4, 2, i)
    sns.histplot(df[col], kde=True)
    plt.subplot(4, 2, i+1)
    sns.boxplot(x=df[col])
    i += 2
plt.show()
```

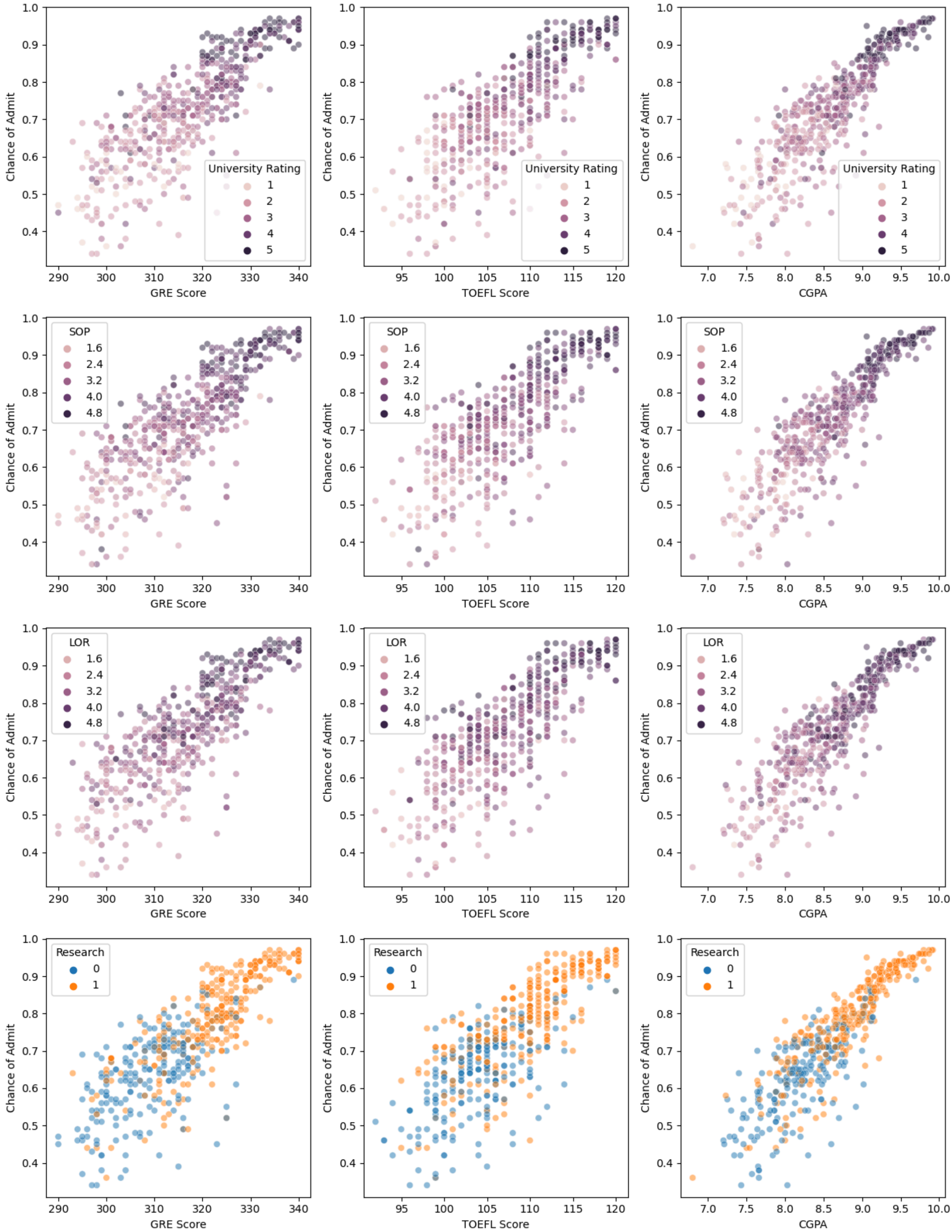


In [12]:
```
plt.figure(figsize=(12, 8))
i = 1
for col in cols_disc:
    plt.subplot(2, 2, i)
    sns.countplot(data=df, x=col)
    i += 1
plt.show()
```

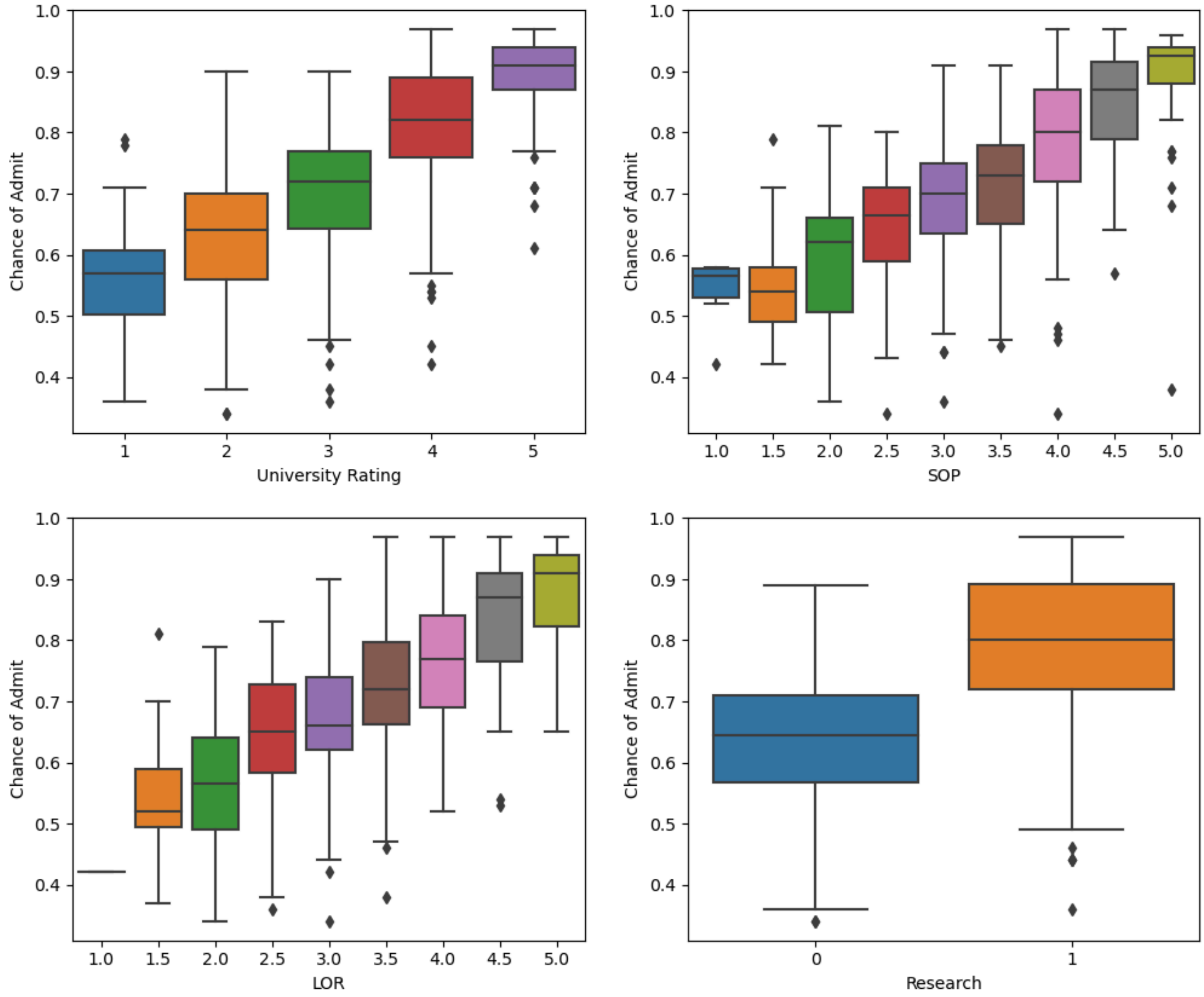


Bivariate Analysis

In [13]:
```
plt.figure(figsize=(15, 20))
i = 1
for col_d in cols_disc:
    for col_c in cols_cont[:-1]:
        plt.subplot(4, 3, i)
        sns.scatterplot(x=df[col_c], y=df['Chance of Admit'], alpha = 0.5, hue=df[col_d])
        i += 1
plt.show()
```



In [14]:
```
plt.figure(figsize=(12, 10))
i = 1
for col_d in cols_disc:
    plt.subplot(2, 2, i)
    sns.boxplot(x=df[col_d], y=df['Chance of Admit'])
    i += 1
plt.show()
```



EDA Insights

- Among the numerical features: GRE score, TOEFL score and CGPA has a strong correlation with chance of admission.
- Among the categorical features: University Rating, SOP, LOR & Research all have postive correlation with chance of admission.
- From above plots we can also see strong correlation b/w the independent variables as well, we will qunatify the strength of multi-collinearity in our below analysis.

Data Pre-processing

Duplicate value check, Missing-Values, Outlier-Treament

In [15]:
```
# Duplicate value check: No duplicate rows
df.loc[df[['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR', 'CGPA','Research']].dupl
```

Out[15]:

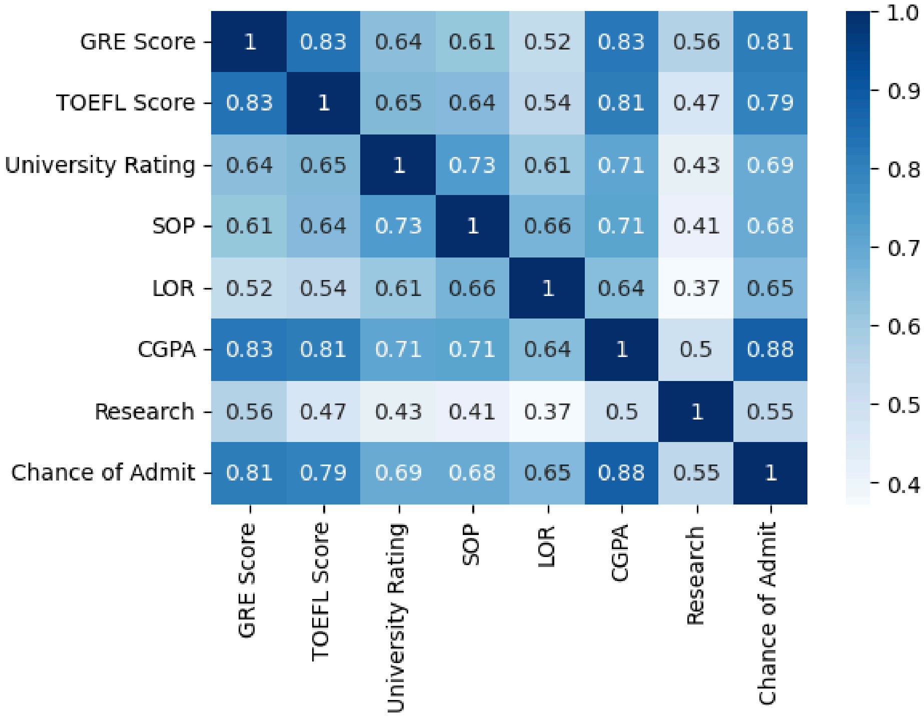
GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit

Missing-values, Outlier Treatment:

- From EDA, we have seen there are no missing values for the independent features
- From EDA, we have also seen that the independent features are not having any outliers (using boxplots which employ the 1.5xIQR rule)

Feature Engineering

In [16]:
```
# Correlation matrix
plt.figure(figsize=(6,4))
sns.heatmap(df.corr(), annot=True, cmap='Blues')
plt.show()
```

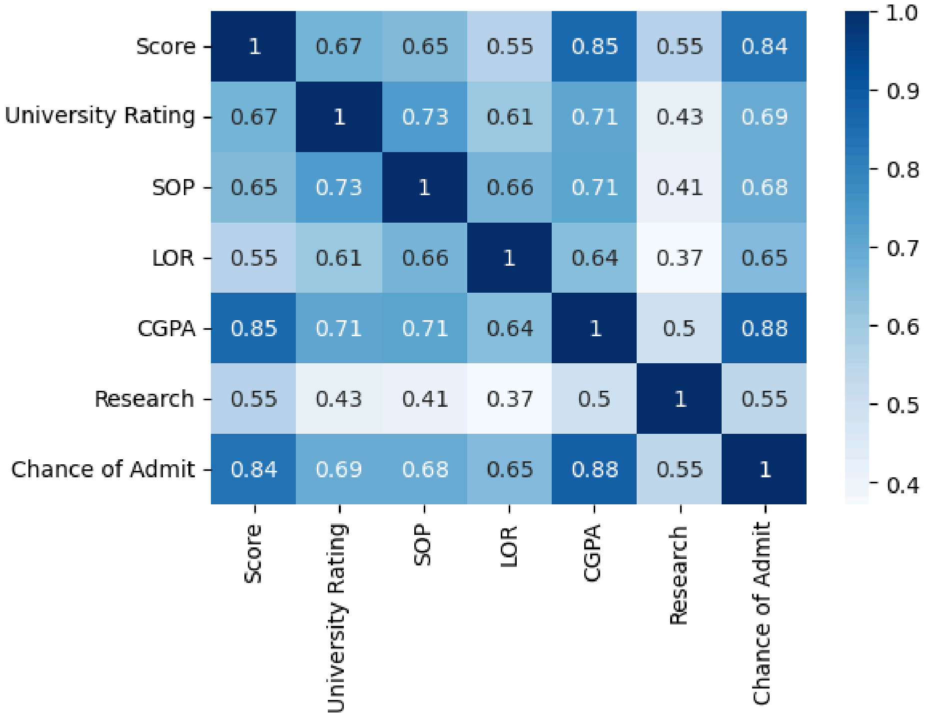


Feature creation #1:

- GRE & TOEFL score can be clubbed together and normalized and scaled up on a base of 100
    - From EDA, we have seen that GRE & TOEFL score havesimilar scatter plot with our target variable.
    - Even from the above correlation matrix, we see that the correaltion coefficients are very similar

In [17]:
```
# Introducing new feature
df_final1 = df.copy()
df_final1['Score'] = 100*((df_final1['GRE Score'] + df_final1['TOEFL Score'])/460)
df_final1 = df_final1[['Score', 'University Rating', 'SOP', 'LOR', 'CGPA', 'Research', 'Chance of A

# Plot new correlation matrix
plt.figure(figsize=(6,4))
sns.heatmap(df_final1.corr(), annot=True, cmap='Blues')
plt.show()
```



In [18]:
```
# Check correlation with target variable
df_final1.corr()['Chance of Admit'].sort_values()
```

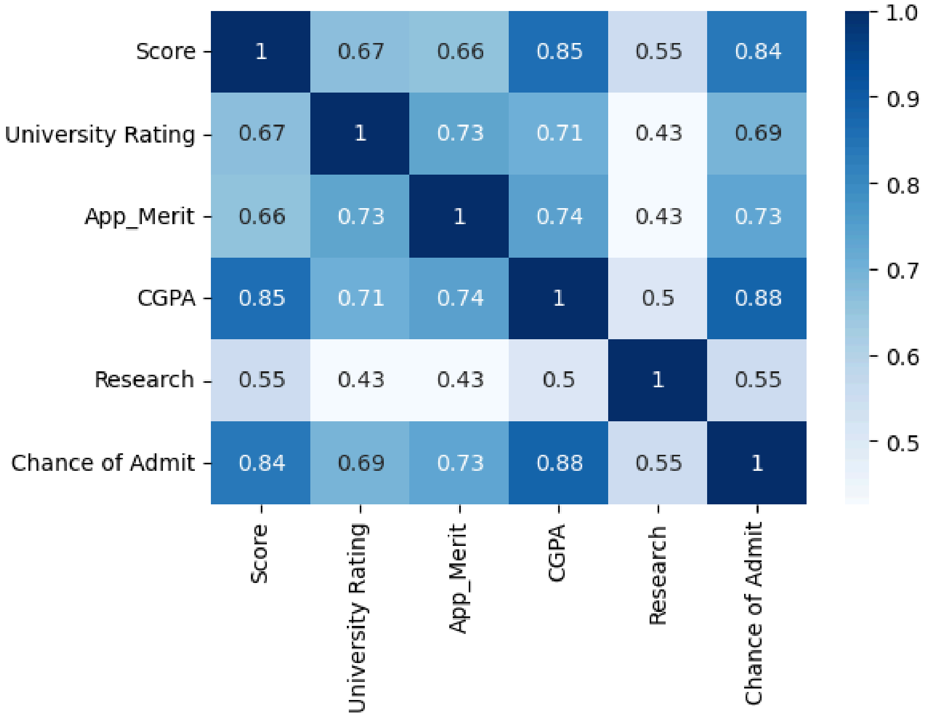
Out[18]:
```
Research             0.545871
LOR                  0.645365
SOP                  0.684137
University Rating    0.690132
Score                0.837609
CGPA                 0.882413
Chance of Admit      1.000000
Name: Chance of Admit, dtype: float64
```

Feature creation #2:

- SOP & LOR can be clubbed together and normalized & scaled up on a base of 5
  - From EDA, we have seen that SOP & LOR score similar scatter plot with our target variable.
  - Even from the above correlation matrix, we see that the correaltion coefficients are very similar

In [19]:
```
# Introducing new feature
df_final2 = df_final1.copy()
df_final2['App_Merit'] = 5*((df_final1['SOP'] + df_final1['LOR'])/10)
df_final2 = df_final2[['Score', 'University Rating', 'App_Merit', 'CGPA', 'Research', 'Chance of Ac

# Plot new correlation matrix
plt.figure(figsize=(6,4))
sns.heatmap(df_final2.corr(), annot=True, cmap='Blues')
plt.show()
```



In [20]:
```
# Check correlation with target variable
df_final2.corr()['Chance of Admit'].sort_values()
```

Out[20]:
```
Research             0.545871
University Rating    0.690132
App_Merit            0.729486
Score                0.837609
CGPA                 0.882413
Chance of Admit      1.000000
Name: Chance of Admit, dtype: float64
```

Base Model using Linear Regression

In [21]:
```
# df_final = df
df_final = df_final2
df_final.head()
```

Out[21]:

	Score	University Rating	App_Merit	CGPA	Research	Chance of Admit
0	98.913043	4	4.50	9.65	1	0.92
1	93.695652	4	4.25	8.87	1	0.76
2	91.304348	3	3.25	8.00	1	0.72
3	93.913043	3	3.00	8.67	1	0.80
4	90.652174	2	2.50	8.21	0	0.65

Train-Test Split, Feature Scaling

In [22]:
```
y = df_final[['Chance of Admit']]
X = df_final.drop(labels='Chance of Admit', axis = 1)
```

In [23]:
```
# Train-Test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=81)
print('Train-Shape:', X_train.shape, X_test.shape)

# Perform Standardisation
scaler = StandardScaler() # mean 0 and standard deviation of 1
X_train_scaled = scaler.fit_transform(X_train) # learn parameters and transform/convert
X_test_scaled = scaler.transform(X_test) # convert using parameters learnt from Training Data
```

```
Train-Shape: (400, 5) (100, 5)
```

Using Sklearn's LinearRegression

In [24]:
```
# # Fit Data using Lin_Reg
# sklr = LinearRegression()
# sklr.fit(X_train_scaled,y_train)

# # R2 score on train set
# train_r2_score = sklr.score(X_train_scaled,y_train)
# print('Train R2_score:', train_r2_score)
```

In [25]:
```
# # See coefficients
# print(sklr.coef_)
# print(sklr.intercept_)
```

In [26]:
```
# # Predict on test set
# y_pred = sklr.predict(X_test_scaled)
# print('Test-Shape:', y_pred.shape, X_test.shape, y_test.shape)

# # R2-score of test_set
# test_r2_score = r2_score(y_test, y_pred)
# print('Test R2_score:', test_r2_score)
```

Using StatsModel's OLS

In [27]:
```
# to include and learn bias
X_train_scaled_cons = sm.add_constant(X_train_scaled)
lr_sm=sm.OLS(y_train,X_train_scaled_cons)

# triggers the training process
fitted_model =  lr_sm.fit() # triggers the training process

# detailed summary
print(fitted_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:         Chance of Admit   R-squared:                       0.820
Model:                             OLS   Adj. R-squared:                  0.817
Method:                  Least Squares   F-statistic:                     358.1
Date:                 Tue, 11 Jun 2024   Prob (F-statistic):          4.45e-144
Time:                         18:17:37   Log-Likelihood:                 558.18
No. Observations:                  400   AIC:                            -1104.
Df Residuals:                      394   BIC:                            -1080.
Df Model:                            5
Covariance Type:             nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.7259      0.003    240.371      0.000       0.720       0.732
x1             0.0336      0.006      5.680      0.000       0.022       0.045
x2             0.0072      0.005      1.434      0.152      -0.003       0.017
x3             0.0156      0.005      2.993      0.003       0.005       0.026
x4             0.0738      0.006     11.494      0.000       0.061       0.086
x5             0.0119      0.004      3.307      0.001       0.005       0.019
==============================================================================
Omnibus:                      111.865   Durbin-Watson:                   2.028
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              295.290
Skew:                          -1.345   Prob(JB):                     7.56e-65
Kurtosis:                       6.238   Cond. No.                         5.03
==============================================================================

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

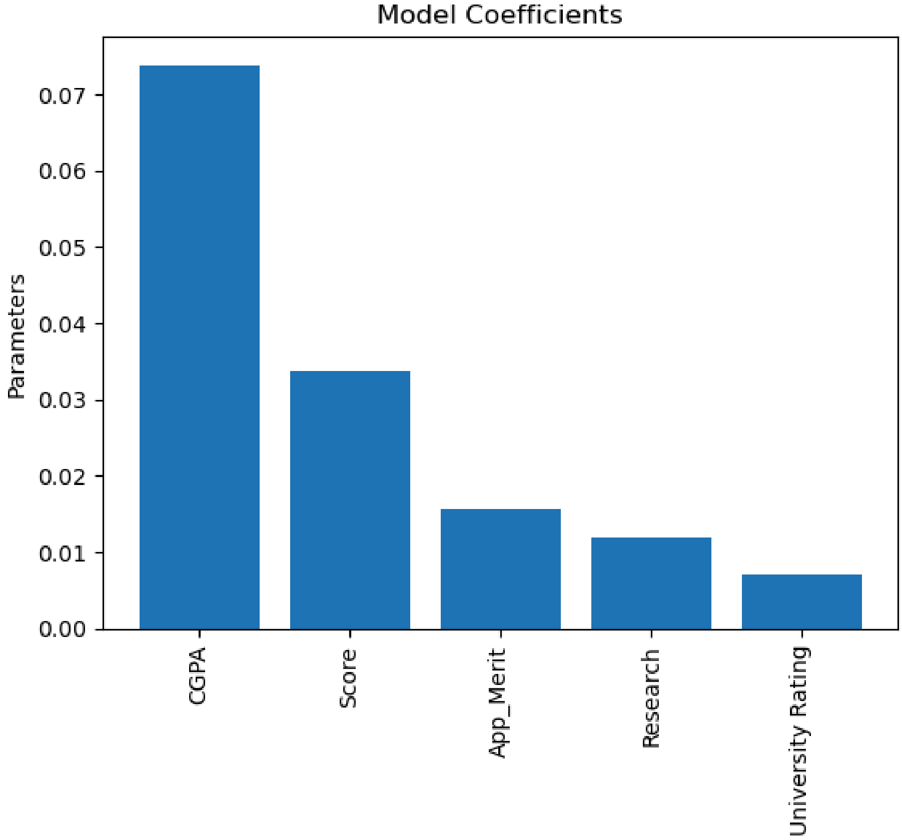
In [28]:
```
# Plotting Parameters against Features
# features = ['Intercept']
# features += list((X_train.columns))

features = list((X_train.columns))
parameters = fitted_model.params.values[1:]
feat_param = list(zip(features, parameters))
feat_param.sort(key=lambda x:x[1], reverse=True)

feat = [i[0] for i in feat_param]
param = [i[1] for i in feat_param]

plt.bar(x=feat, height=param)
plt.title('Model Coefficients')
plt.ylabel('Parameters')
```

```
plt.xticks(rotation=90)
plt.show();
```



In [29]:
```
lin_reg_param = pd.DataFrame()
lin_reg_param['Features'] = ['Intercept'] + list(X_train.columns)
lin_reg_param['Parameters'] = fitted_model.params.values
lin_reg_param
```

Out[29]:

	Features	Parameters
0	Intercept	0.725875
1	Score	0.033629
2	University Rating	0.007156
3	App_Merit	0.015593
4	CGPA	0.073827
5	Research	0.011855

In [30]:
```
# Predict on test set
X_test_scaled_cons = sm.add_constant(X_test_scaled)
sm_pred=fitted_model.predict(X_test_scaled_cons)
sm_pred

# R2-score of test_set
test_r2_score= r2_score(y_test, sm_pred)
print('Test R2_score:', test_r2_score)
```

```
Test R2_score: 0.8176851788128836
```

In [31]:
```
def adj_r2_score(r2_score, n, d):
    term = ((1-r2_score)*(n-1))/(n-d-1)
    return 1 - term


print('test_adj_r2_score:', adj_r2_score(test_r2_score, X_test_scaled.shape[0],  X_test_scaled.shap
```

```
test_adj_r2_score: 0.8079875819412284
```

Regularized Models: Ridge & Lasso

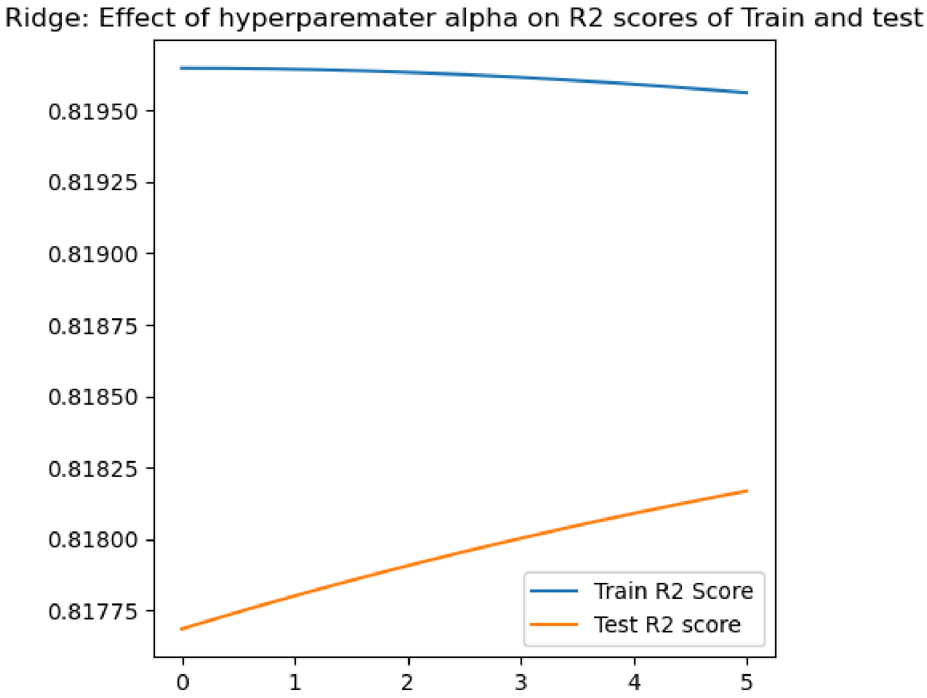
In [32]:
```
from sklearn.linear_model import Ridge  # L2 regualrization
from sklearn.linear_model import Lasso  # L1 regualrization
```

Ridge

In [33]:
```
## Hyperparameter Tuning : for appropriate lambda value :
ridge_train_r2_score = []
ridge_test_r2_score = []
lambdas = []
ridge_train_test_difference_r2 =  []
lambda_ = 0
while lambda_ <= 5:
    lambdas.append(lambda_)
    RidgeModel = Ridge(lambda_)
    RidgeModel.fit(X_train_scaled,y_train)
    trainR2 = RidgeModel.score(X_train_scaled,y_train)
    testR2 = RidgeModel.score(X_test_scaled,y_test)
    ridge_train_r2_score.append(trainR2)
    ridge_test_r2_score.append(testR2)

    lambda_ += 0.01
```

In [34]:
```
plt.figure(figsize = (5,5))
plt.plot(lambdas, ridge_train_r2_score)
plt.plot(lambdas, ridge_test_r2_score)
plt.legend(['Train R2 Score','Test R2 score'])
plt.title("Ridge: Effect of hyperparemater alpha on R2 scores of Train and test")
plt.show()
```



We can try Ridge Regression, although it does not give any significant better performance than simple Linear Regression

In [35]:
```
# Varying alpha doesn't change much, we will proceed with alpha=0.1
RidgeModel = Ridge(alpha = 0.1)
RidgeModel.fit(X_train_scaled,y_train)
ridge_trainR2 = RidgeModel.score(X_train_scaled,y_train)
ridge_testR2 = RidgeModel.score(X_test_scaled,y_test)
```

In [36]:
```
ridge_model_param = pd.DataFrame()
ridge_model_param['Features'] = ['Intercept'] + list(X_train.columns)
ridge_model_param['Parameters'] = np.append(RidgeModel.intercept_, RidgeModel.coef_)
ridge_model_param
```

Out [36]:

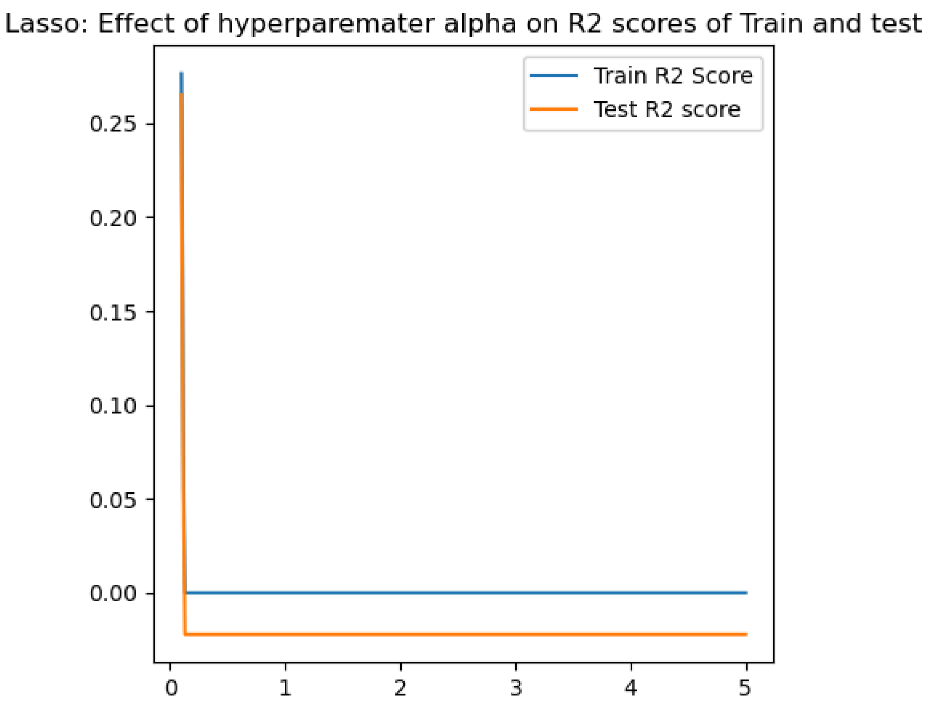
	Features	Parameters
0	Intercept	0.725875
1	Score	0.033648
2	University Rating	0.007168
3	App_Merit	0.015607
4	CGPA	0.073772
5	Research	0.011857

Lasso

In [37]:
```
## Hyperparameter Tuning : for appropriate lambda value :
lasso_train_r2_score = []
lasso_test_r2_score = []
lambdas = []
lasso_train_test_difference_r2 =  []
lambda_ = 0.1
while lambda_ <= 5:
    lambdas.append(lambda_)
    LassoModel = Lasso(lambda_)
    LassoModel.fit(X_train_scaled,y_train)
    trainR2 = LassoModel.score(X_train_scaled,y_train)
    testR2 = LassoModel.score(X_test_scaled,y_test)
    lasso_train_r2_score.append(trainR2)
    lasso_test_r2_score.append(testR2)

    lambda_ += 0.01
```

In [38]:
```
plt.figure(figsize = (5,5))
plt.plot(lambdas, lasso_train_r2_score)
plt.plot(lambdas, lasso_test_r2_score)
plt.legend(['Train R2 Score','Test R2 score'])
plt.title("Lasso: Effect of hyperparemater alpha on R2 scores of Train and test")
plt.show()
```



Clearly from the above plot, Lasso Regression is not a solution

Assumptions for Linear Regression

- Target variable is linearly dependent on independent variables
- No multicollinearity between independent variables
- Mean of Residuals should approximately zero

- Errors(Residuals) should be normally distributed
- Heteroscedasticity (b/2 residuals and y_pred) should not exist

In [39]:
```
df_final
```

Out[39]:

	Score	University Rating	App_Merit	CGPA	Research	Chance of Admit
0	98.913043	4	4.50	9.65	1	0.92
1	93.695652	4	4.25	8.87	1	0.76
2	91.304348	3	3.25	8.00	1	0.72
3	93.913043	3	3.00	8.67	1	0.80
4	90.652174	2	2.50	8.21	0	0.65
...	...	...	...	...	...	...
495	95.652174	5	4.25	9.02	1	0.87
496	98.695652	5	5.00	9.87	1	0.96
497	97.826087	5	4.75	9.56	1	0.93
498	90.217391	4	4.50	8.43	0	0.73
499	95.652174	4	4.50	9.04	0	0.84

500 rows × 6 columns

Target variable is linearly dependent on independent variables

- Linearity of variables refers to the assumption that there is a linear relationship between the independent variables and the dependent variable in a regression model. It means that the effect of the independent variables on the dependent variable is constant across different levels of the independent variables.
- When we talk about "no pattern in the residual plot" in the context of linearity, we are referring to the plot of the residuals (the differences between the observed and predicted values of the dependent variable) against the predicted values or the independent variables.
- Ideally, in a linear regression model, the residuals should be randomly scattered around zero, without any clear patterns or trends. This indicates that the model captures the linear relationships well and the assumption of linearity is met.

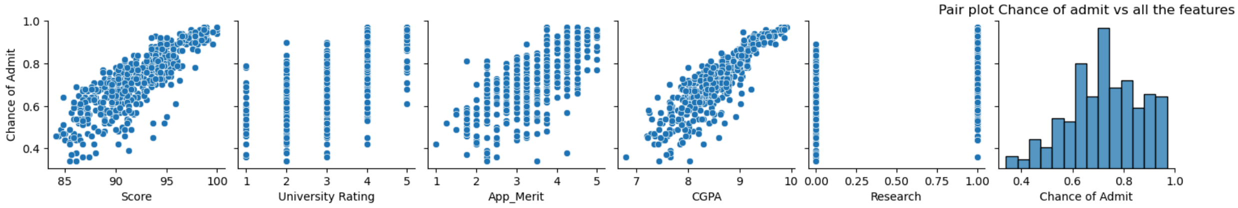
- If a pattern is observed in the residual plot, it may indicate that the linear regression model is not appropriate, and nonlinear regression or other modeling techniques should be considered. Additionally, transformations of variables, adding interaction terms, or using polynomial terms can sometimes help capture nonlinear relationships and improve linearity in the residual plot.

Common patterns that indicate non-linearity

If there is a visible pattern in the residual plot, it suggests a violation of the linearity assumption. Common patterns that indicate non-linearity include:

- Curved or nonlinear shape: The residuals form a curved or nonlinear pattern instead of a straight line.
- U-shaped or inverted U-shaped pattern: The residuals show a U-shape or inverted U-shape, indicating a nonlinear relationship.
- Funnel-shaped pattern: The spread of residuals widens or narrows as the predicted values or independent variables change, suggesting heteroscedasticity.
- Clustering or uneven spread: The residuals show clustering or uneven spread across different levels of the predicted values or independent variables.

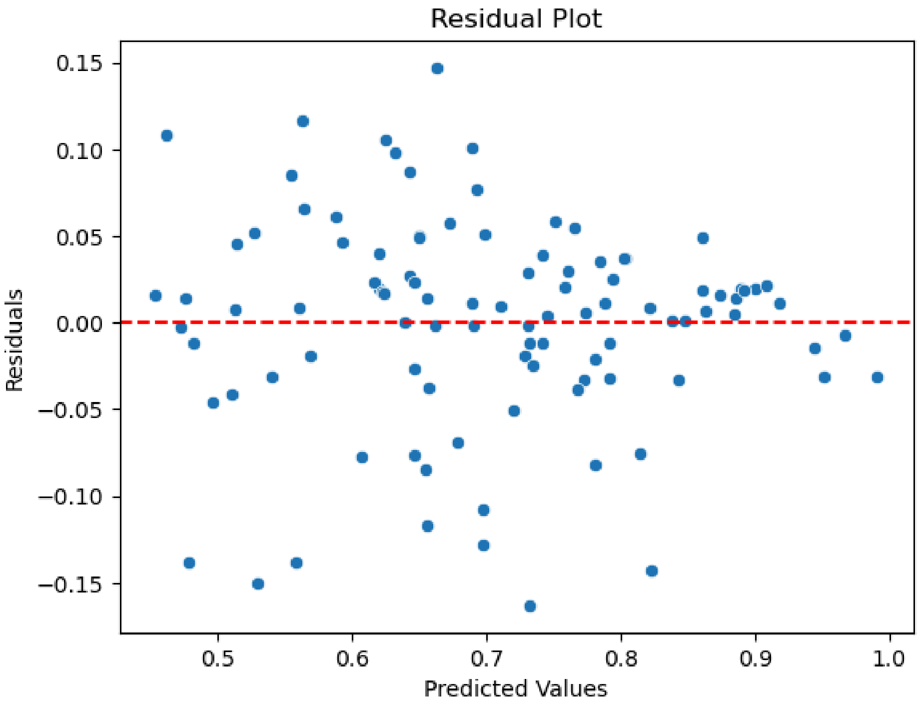
In [40]:
```
sns.pairplot(df_final, y_vars='Chance of Admit')
plt.title("Pair plot Chance of admit vs all the features")
plt.show()
```



In [41]:
```python
print(sm_pred.shape, y_test.shape)
residuals = y_test['Chance of Admit'].values - sm_pred
```

```
(100,) (100, 1)
```

In [42]:
```python
sns.scatterplot(x = sm_pred, y=residuals)
plt.title('Residual Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.show();
```



From the above plots, we can safely pass this assumption

No multicollinearity between independent variables

- VIF (Variance Inflation Factor) is a measure that quantifies the severity of multicollinearity in a regression analysis.
- It assesses how much the variance of the estimated regression coefficient is inflated due to collinearity.

In [43]:
```python
def calculate_vif(data):
    # VIF dataframe
    vif_data = pd.DataFrame()
    vif_data["feature"] = data.columns

    # calculating VIF for each feature
    vif_data["VIF_Score"] = [variance_inflation_factor(data.values, i) for i in range(data.shape[1
    vif_data.sort_values(by='VIF_Score', ascending=False, inplace=True)
    return vif_data
```

In [44]:
```python
calculate_vif(X_train)
```

Out[44]:

	feature	VIF_Score
3	CGPA	908.426415
0	Score	738.058095
2	App_Merit	49.066195
1	University Rating	21.058491
4	Research	2.918342

In [45]:
```
# Drop CGPA and recalculate VIF
calculate_vif(X_train.drop(labels='CGPA', axis=1))
```

Out[45]:

	feature	VIF_Score
2	App_Merit	42.313808
0	Score	20.747072
1	University Rating	20.494255
3	Research	2.906263

As predicted from our EDA, there is a lot of correlation amongst the independent variables themselves which is not a good thing to have when using Linear Regression.

Re-training on Reduced features (Experimental)

In [46]:
```
# y_red = df_final[['Chance of Admit']]
# X_red = df_final[['Score', 'Research', 'App_Merit', 'Research']]
```

In [47]:
```
# # Train-Test split
# Xr_train, Xr_test, yr_train, yr_test = train_test_split(X_red, y_red, test_size=0.2, random_state
# print('Train-Shape:', Xr_train.shape, Xr_test.shape)

# # Perform Standardisation
# scaler = StandardScaler() # mean 0 and standard deviation of 1
# Xr_train_scaled = scaler.fit_transform(Xr_train) # learn parameters and transform/convert
# Xr_test_scaled = scaler.transform(Xr_test) # convert using parameters learnt from Training Data
```

In [48]:
```
# # Fit Data using Lin_Reg
# sklr = LinearRegression()
# sklr.fit(Xr_train_scaled,yr_train)

# # R2 score on train set
# train_r2_score_red = sklr.score(Xr_train_scaled,yr_train)
# print('Train R2_score:', train_r2_score_red)


# # Predict on test set
# yr_pred = sklr.predict(Xr_test_scaled)
# print('Test-Shape:', yr_pred.shape, Xr_test.shape, yr_test.shape)

# # R2-score of test_set
# test_r2_score_red = r2_score(yr_test, yr_pred)
# print('Test R2_score:', test_r2_score_red)
```

Mean of Residuals should be approximately zero

- The mean of residuals represents the average of residual values in a regression model. Residuals are the discrepancies or errors between the observed values and the values predicted by the regression model.
- The mean of residuals is useful to assess the overall bias in the regression model. If the mean of residuals is close to zero, it indicates that the model is unbiased on average. However, if the mean of residuals is significantly different from zero, it suggests that the model is systematically overestimating or underestimating the observed values.

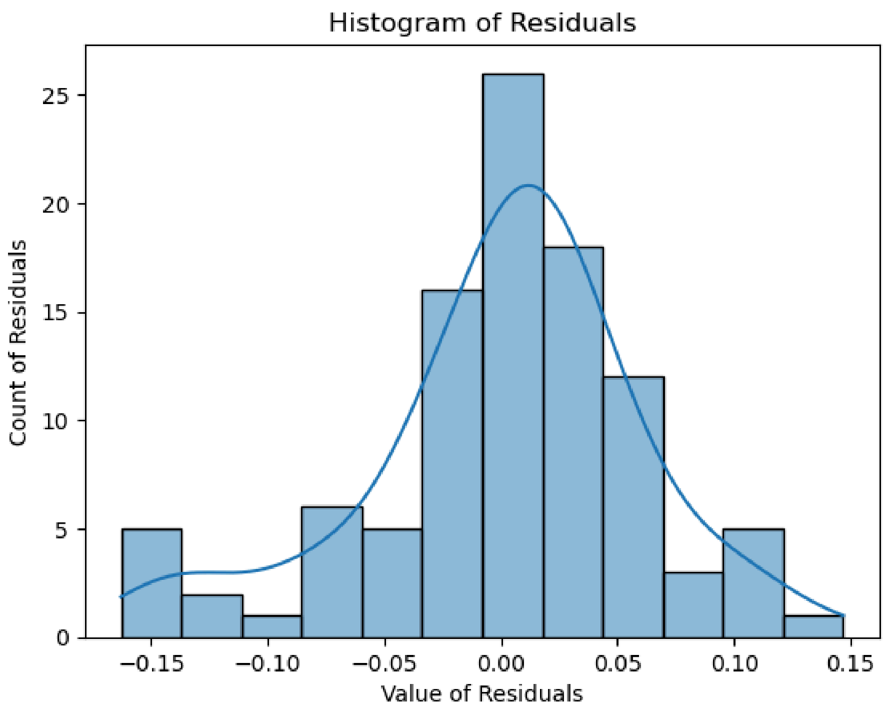
In [49]:
```
print('Mean of residuals:', residuals.mean())
```

```
Mean of residuals: 0.001101391660017783
```

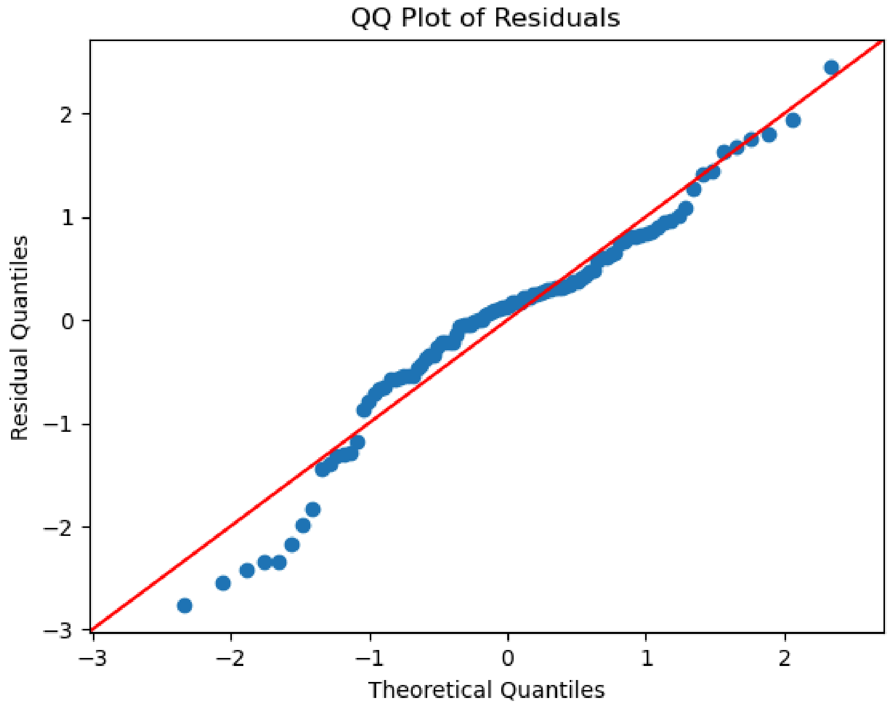
Errors(Residuals) should be normally distributed

- Normality of residuals refers to the assumption that the residuals (or errors) in a statistical model are normally distributed. Residuals are the differences between the observed values and the predicted values from the model.
- The assumption of normality is important in many statistical analyses because it allows for the application of certain statistical tests and the validity of confidence intervals and hypothesis tests. When residuals are normally distributed, it implies that the errors are random, unbiased, and have consistent variability.
  - To check for the normality of residuals, you can follow these steps:
  - Residual Histogram: Create a histogram of the residuals and visually inspect whether the shape of the histogram resembles a bell-shaped curve. If the majority of the residuals are clustered around the mean with a symmetric distribution, it suggests normality.
  - Q-Q Plot (Quantile-Quantile Plot): This plot compares the quantiles of the residuals against the quantiles of a theoretical normal distribution. If the points in the Q-Q plot are reasonably close to the diagonal line, it indicates that the residuals are normally distributed. Deviations from the line may suggest departures from normality.
  - Shapiro-Wilk Test: This is a statistical test that checks the null hypothesis that the residuals are normally distributed. The Shapiro-Wilk test calculates a test statistic and provides a p-value. If the p-value is greater than the chosen significance level (e.g., 0.05), it suggests that the residuals follow a normal distribution. However, this test may not be reliable for large sample sizes.
  - Skewness and Kurtosis: Calculate the skewness and kurtosis of the residuals. Skewness measures the asymmetry of the distribution, and a value close to zero suggests normality. Kurtosis measures the heaviness of the tails of the distribution compared to a normal distribution, and a value close to zero suggests similar tail behavior.

In [50]:
```
#Histogram of Residuals
sns.histplot(residuals, kde=True)
plt.title('Histogram of Residuals')
plt.xlabel('Value of Residuals')
plt.ylabel('Count of Residuals')
plt.show();
```



In [51]:
```
# QQ-Plot of residuals
sm.qqplot(residuals, fit=True, line='45')
plt.title('QQ Plot of Residuals')
plt.ylabel('Residual Quantiles')
plt.show();
```



We see that the distribution is close to normal, there is some deviation in the left portion of the distribution though

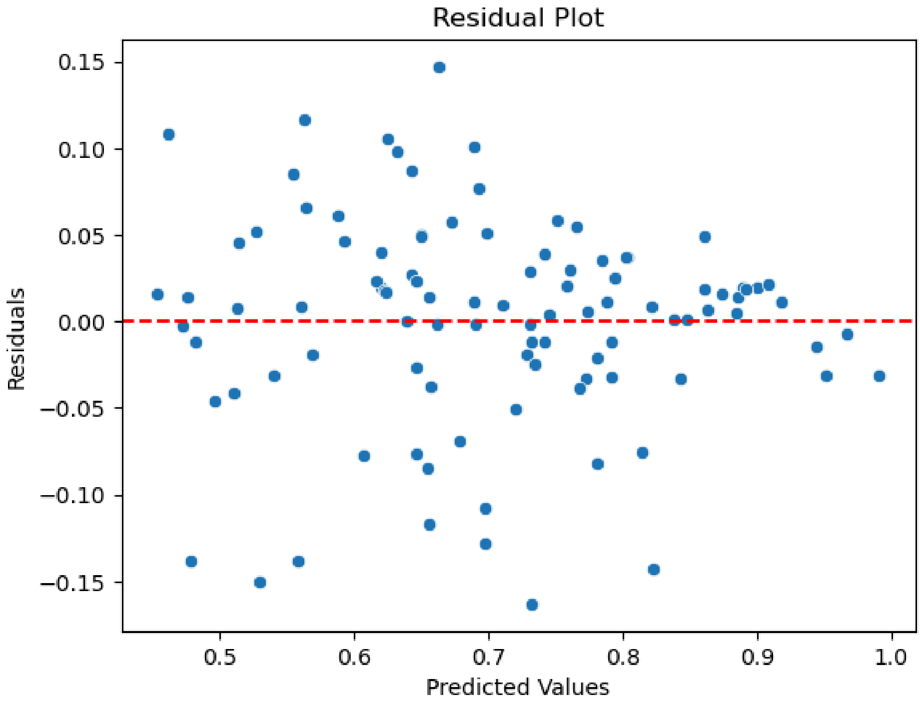
Heteroscedasticity (b/w residuals and y_pred) should not exist => Homoscedasticity should exist

- Homoscedasticity refers to the assumption in regression analysis that the variance of the residuals (or errors) should be constant across all levels of the independent variables. In simpler terms, it means that the spread of the residuals should be similar across different values of the predictors.
- When homoscedasticity is violated, it indicates that the variability of the errors is not consistent across the range of the predictors, which can lead to unreliable and biased regression estimates.

Tests for homoscedasticity (there are several graphical and statistical methods):

- Residual plot: Plot the residuals against the predicted values or the independent variables. Look for any systematic patterns or trends in the spread of the residuals. If the spread appears to be consistent across all levels of the predictors, then homoscedasticity is likely met.
- Scatterplot: If you have multiple independent variables, you can create scatter plots of the residuals against each independent variable separately. Again, look for any patterns or trends in the spread of the residuals.
- Breusch-Pagan Test: This is a statistical test for homoscedasticity. It involves regressing the squared residuals on the independent variables and checking the significance of the resulting model. If the p-value is greater than a chosen significance level (e.g., 0.05), it suggests homoscedasticity. However, this test assumes that the errors follow a normal distribution.
- Goldfeld-Quandt Test: This test is used when you suspect heteroscedasticity due to different variances in different parts of the data. It involves splitting the data into two subsets based on a specific criterion and then comparing the variances of the residuals in each subset. If the difference in variances is not significant, it suggests homoscedasticity.

In [52]:
```
sns.scatterplot(x = sm_pred, y=residuals)
plt.title('Residual Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.show();
```



Visually, it looks like that the variance of errors keep on decreasing as the chance os admission is increasing

Model Performance Evaluation

- We tried 3 approaches: Linear, Ridge and Lasso Regressions
- Simple Linear Reg & Ridge Reg gave us R2 scores of about ~82%
- We have document the metrics for those 2 approaches

In [53]:
```
n = X_test_scaled.shape[0]
d = X_test_scaled.shape[1]

summary = pd.DataFrame()
summary['Metrics'] = ['MAE', 'RMSE', 'Train_R2', 'Train_Adj_R2', 'Test_R2', 'Test_AdjR2']
summary['Linear_Reg'] = [mean_absolute_error(y_test, sm_pred), np.sqrt(mean_squared_error(y_test, s
                         fitted_model.rsquared, adj_r2_score(fitted_model.rsquared,n,d),
                         test_r2_score,  adj_r2_score(test_r2_score,n,d)]
summary['Ridge_Reg'] = [mean_absolute_error(y_test, RidgeModel.predict(X_test_scaled)), np.sqrt(mea
                        ridge_trainR2, adj_r2_score(ridge_trainR2,n,d),
                        ridge_testR2, adj_r2_score(ridge_testR2,n,d)]
summary
```

Out[53]:

	Metrics	Linear_Reg	Ridge_Reg
0	MAE	0.043883	0.043883
1	RMSE	0.059430	0.059428
2	Train_R2	0.819649	0.819649
3	Train_Adj_R2	0.810056	0.810056
4	Test_R2	0.817685	0.817697
5	Test_AdjR2	0.807988	0.808000

Insights and Recommendations

Insights:

- The distribution of target variable (chances of admit) is left-skewed.
- Exam scores (CGPA/GRE/TOEFL) have a strong positive correlation with chance of admit. These variables are also highly correlated amongst themselves.
- The categorical variables such as university ranking, research, quality of SOP and LOR also show an upward trend for chances of admit.

- We had created 2 features: 'Score' which encapsulates GRE & TOEFL scores, 'App_Merit' which encapsulates SOP & LOR quality.
- From the model coefficients (parameters), we can conclude that CGPA is the most significant predictor variable while University Rating/Research are the least significant.
- Both Linear Regression and Ridge Regression models, which are our best models, have captured upto 82% of the variance in the target variable (chance of admit). Due to high colinearity among the predictor variables, it is difficult to achieve better results.
- Other than multicolinearity, the predictor variables have grossly met the conditions required for Linear Regression - mean of residuals is close to 0, linearity of variables, normality of residuals and homoscedasticity is established.

Recommendations:

- Since all the exam scores are highly correlated, it is recommended to add more independent features for better prediction.
- Since our R2 score is close to ~80%, there is a lot of scope for improvement.
- Adding more relevant independent variables may be useful in capturing more variance in our target.
- Examples of other independent variables could be work experience, internships, mock interview performance, extracurricular activities or diversity variables.

In [ ]:

In [ ]:

In [ ]: