# Jamboree Linear Regression Case Study - Problem Statement

**Context**

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

**Jamboree 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.**

The goal is to help Jamboree in understanding what factors are important in graduate admissions and how these factors are interrelated among themselves and also to help in predicting one's chances of admission given the rest of the variables.

**Additional View**

- Lin Reg. will also help predict one's chances of admission given the rest of the variables.
- GRE Score, TOEFL Score & CGPA are most important attributes as per Indian Perspective.

# Installing Dependencies

In [367…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

## About the Dataset

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

**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 Dataset

In [368…

```
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
jamboree=pd.read_csv('Jamboree_Admission.csv')
#df.head()
```

In [369…

```
jamboree.head(5)
```

Out[369]:

| | 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 |

### Removing unwanted column from the dataset

In [370…

```
jamboree.drop(["Serial No."], axis = 1, inplace = True)
```

In [371…

```
jamboree.shape
```

Out[371]:

```
(500, 8)
```

In [372…

```
jamboree.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   GRE Score          500 non-null    int64  
 1   TOEFL Score        500 non-null    int64  
 2   University Rating  500 non-null    int64  
 3   SOP                500 non-null    float64
 4   LOR                500 non-null    float64
 5   CGPA               500 non-null    float64
 6   Research           500 non-null    int64  
 7   Chance of Admit    500 non-null    float64
dtypes: float64(4), int64(4)
memory usage: 31.4 KB
```

In [373…

```
jamboree.dtypes
```

Out[373]:

```
GRE Score              int64
TOEFL Score            int64
University Rating      int64
SOP                  float64
LOR                  float64
CGPA                 float64
Research               int64
Chance of Admit      float64
dtype: object
```

- **All the features are numerical**

In [374…

```
jamboree.isnull().sum()
```

Out[374]:

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

- **From above it is evident that there are no null values**

In [375…

```
jamboree.describe().T
```

Out[375]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **GRE Score** | 500.0 | 316.47200 | 11.295148 | 290.00 | 308.0000 | 317.00 | 325.00 | 340.00 |
| **TOEFL Score** | 500.0 | 107.19200 | 6.081868 | 92.00 | 103.0000 | 107.00 | 112.00 | 120.00 |
| **University Rating** | 500.0 | 3.11400 | 1.143512 | 1.00 | 2.0000 | 3.00 | 4.00 | 5.00 |
| **SOP** | 500.0 | 3.37400 | 0.991004 | 1.00 | 2.5000 | 3.50 | 4.00 | 5.00 |
| **LOR** | 500.0 | 3.48400 | 0.925450 | 1.00 | 3.0000 | 3.50 | 4.00 | 5.00 |
| **CGPA** | 500.0 | 8.57644 | 0.604813 | 6.80 | 8.1275 | 8.56 | 9.04 | 9.92 |
| **Research** | 500.0 | 0.56000 | 0.496884 | 0.00 | 0.0000 | 1.00 | 1.00 | 1.00 |
| **Chance of Admit** | 500.0 | 0.72174 | 0.141140 | 0.34 | 0.6300 | 0.72 | 0.82 | 0.97 |

- **While Observing the mean and 50% percentile of data there is no significant difference observed**
- **We can conclude there are no outliers in the dataset.**

In [376…

```
jamboree.duplicated().sum()
```

Out[376]: 0

- **There are no duplicated values in the dataset**

### Renaming columns - Removing extra whitespace "LOR " & "Chance of Admit "

In [377…

```
#jamboree.columns = map(lambda x: x.strip(), jamboree.columns)
jamboree.rename(columns={"LOR ": "LOR", "Chance of Admit ": "Chance of Admit"},
```

# Non-Graphical Analysis

In [378…

```
jamboree["University Rating"].value_counts(normalize=True)
```

Out[378]:

```
3    0.324
2    0.252
4    0.210
5    0.146
1    0.068
Name: University Rating, dtype: float64
```

In [379…

```
jamboree['Research'].unique()
```

Out[379]: array([1, 0], dtype=int64)

In [380…

```
jamboree['University Rating'].unique()
```

```
Out[380]: array([4, 3, 2, 5, 1], dtype=int64)
```

- **While observing the university rating. Most of universities average rated.**

**Research and University rating are categorical variables**

```
In [381…  jamboree["SOP"].value_counts(normalize=True)
```

```
Out[381]: 4.0    0.178
3.5    0.176
3.0    0.160
2.5    0.128
4.5    0.126
2.0    0.086
5.0    0.084
1.5    0.050
1.0    0.012
Name: SOP, dtype: float64
```

```
In [382…  jamboree["Research"].value_counts(normalize=True)
```

```
Out[382]: 1    0.56
0    0.44
Name: Research, dtype: float64
```

- **Above stats shows there are almost equal distribution among students who did research**

# Utility Functions - Used during Analysis

## Missing Value - Calculator

```
In [383…  def missingValue(df):
    #Identifying Missing data. Already verified above. To be sure again checking
    total_null = df.isnull().sum().sort_values(ascending = False)
    percent = ((df.isnull().sum()/df.isnull().count())*100).sort_values(ascendin
    print("Total records = ", df.shape[0])

    md = pd.concat([total_null,percent.round(2)],axis=1,keys=['Total Missing','I
    return md
```

## Categorical Variable Analysis

**Bar plot** - Frequency of feature in percentage

**Pie Chart**

```
In [451…  # Frequency of each feature in percentage.
def cat_analysis(df, colnames, nrows=2,mcols=2,width=20,height=30, sortbyindex=F
    fig , ax = plt.subplots(nrows,mcols,figsize=(width,height))
    fig.set_facecolor(color = 'lightgrey')
    string = "Frequency of "
    rows = 0
```

```
    for colname in colnames:
        count = (df[colname].value_counts(normalize=True)*100)
        string += colname + ' in (%)'
        if sortbyindex:
                count = count.sort_index()
        count.plot.bar(color=sns.color_palette("deep"),ax=ax[rows][0])
        ax[rows][0].set_ylabel(string, fontsize=14,family = "Comic Sans MS")
        ax[rows][0].set_xlabel(colname, fontsize=14,family = "Comic Sans MS")
        count.plot.pie(colors = sns.color_palette("pastel"),autopct='%0.0f%%',
                       textprops={'fontsize': 14,'family':"Comic Sans MS",'color
        string = "Frequency of "
        rows += 1
```

## Function for Outlier detection

Box plot - for checking range of outliers distplot - For checking skewness

In [385…

```
def outlier_detect(df,colname,nrows=2,mcols=2,width=20,height=15):
    fig , ax = plt.subplots(nrows,mcols,figsize=(width,height))
    fig.set_facecolor("lightgrey")
    rows = 0
    for var in colname:
        ax[rows][0].set_title("Boxplot for Outlier Detection ", fontweight="bold
        plt.ylabel(var, fontsize=12,family = "Comic Sans MS")
        sns.boxplot(y = df[var],color='b',ax=ax[rows][0])

        # plt.subplot(nrows,mcols,pltcounter+1)
        sns.distplot(df[var],color='y',ax=ax[rows][1])
        ax[rows][1].axvline(df[var].mean(), color='r', linestyle='--', label="Me
        ax[rows][1].axvline(df[var].median(), color='m', linestyle='-', label="M
        ax[rows][1].axvline(df[var].mode()[0], color='royalblue', linestyle='-',
        ax[rows][1].set_title("Outlier Detection ", fontweight="bold")
        ax[rows][1].legend({'Mean':df[var].mean(),'Median':df[var].median(),'Mod
        rows += 1
    plt.show()
```

## Boxplot for Categorical variables

In [386…

```
#Function to plot a list of categorical variables together
def box_plot(colname,y):
    fig = plt.figure(figsize=(18, 14))
    fig.set_facecolor("darkgrey")
    for var in colname:
        plt.subplot(2,2,colname.index(var)+1)
        sns.boxplot(x = var, y = y, data = jamboree)
        plt.title("Box plot of " + var, fontweight="bold")
    plt.show()
```

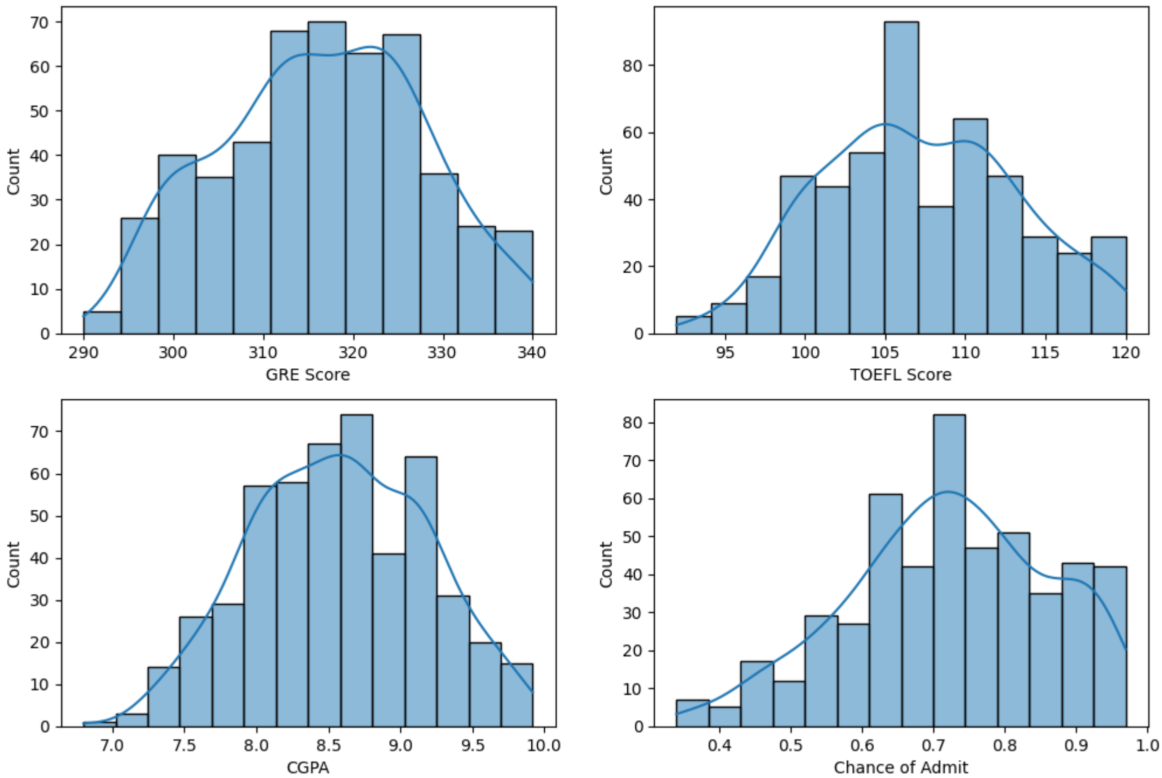
# Univariate Analysis

In [387…

```
cat_cols = ['University Rating', 'SOP', 'LOR', 'Research']
num_cols = ['GRE Score', 'TOEFL Score', 'CGPA']
target = 'Chance of Admit'
```

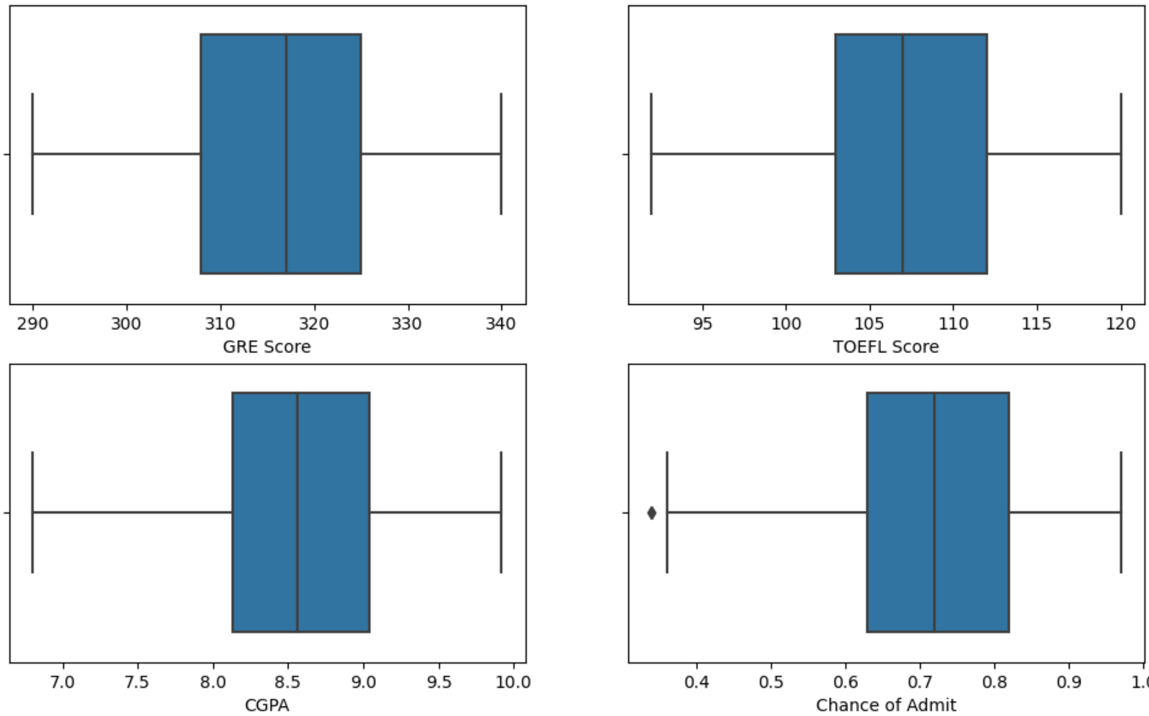
In [388…

```
# check distribution of each numerical variable
rows, cols = 2, 2
fig, axs = plt.subplots(rows,cols, figsize=(12, 8))
index = 0
for row in range(rows):
    for col in range(cols):
        sns.histplot(jamboree[num_cols[index]], kde=True, ax=axs[row,col])
        index += 1
    break

sns.histplot(jamboree[num_cols[-1]], kde=True, ax=axs[1,0])
sns.histplot(jamboree[target], kde=True, ax=axs[1,1])
plt.show()
```



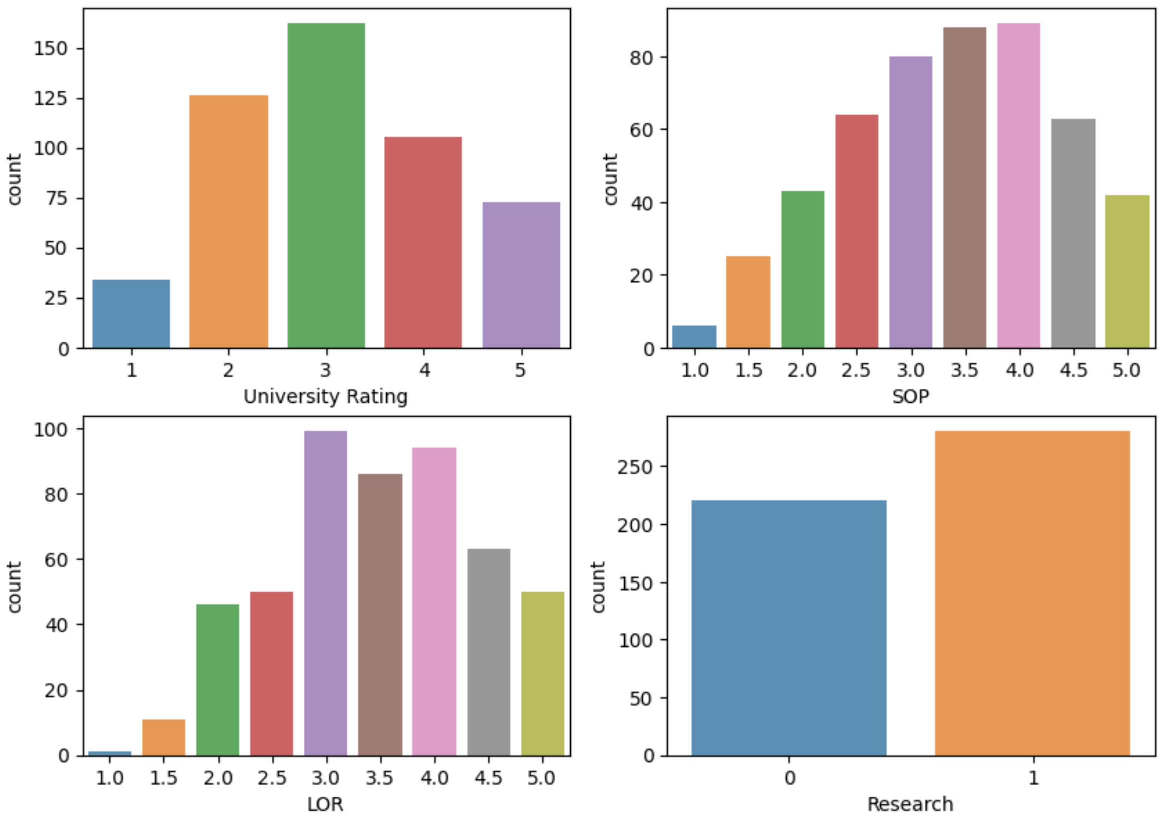

In [389…

```
# check for outliers using boxplots
rows, cols = 2, 2
fig, axs = plt.subplots(rows, cols, figsize=(12, 7))

index = 0
for col in range(cols):
    sns.boxplot(x=num_cols[index], data=jamboree, ax=axs[0,index])
    index += 1

sns.boxplot(x=num_cols[-1], data=jamboree, ax=axs[1,0])
sns.boxplot(x=target, data=jamboree, ax=axs[1,1])
plt.show()
```




In [390…

```
# countplots for categorical variables
cols, rows = 2, 2
fig, axs = plt.subplots(rows, cols, figsize=(10, 7))

index = 0
for row in range(rows):
    for col in range(cols):
        sns.countplot(x=cat_cols[index], data=jamboree, ax=axs[row, col], alpha=
        index += 1

plt.show()
```



In [ ]:

## DataType Validation

In [391…

```
jamboree.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   GRE Score          500 non-null    int64  
 1   TOEFL Score        500 non-null    int64  
 2   University Rating  500 non-null    int64  
 3   SOP                500 non-null    float64
 4   LOR                500 non-null    float64
 5   CGPA               500 non-null    float64
 6   Research           500 non-null    int64  
 7   Chance of Admit    500 non-null    float64
dtypes: float64(4), int64(4)
memory usage: 31.4 KB
```

In [392…

```
jamboree['SOP'] = jamboree['SOP'].astype("category")
jamboree['LOR'] = jamboree['LOR'].astype("category")
jamboree['Research'] = jamboree['Research'].astype("category")
jamboree['University Rating'] = jamboree['University Rating'].astype("category")
```

In [393…

```
jamboree.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype   
---  ------             --------------  -----   
 0   GRE Score          500 non-null    int64   
 1   TOEFL Score        500 non-null    int64   
 2   University Rating  500 non-null    category
 3   SOP                500 non-null    category
 4   LOR                500 non-null    category
 5   CGPA               500 non-null    float64 
 6   Research           500 non-null    category
 7   Chance of Admit    500 non-null    float64 
dtypes: category(4), float64(2), int64(2)
memory usage: 18.8 KB
```

## Statistical Summary

In [394…

```
jamboree.describe().T
```

Out[394]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **GRE Score** | 500.0 | 316.47200 | 11.295148 | 290.00 | 308.0000 | 317.00 | 325.00 | 340.00 |
| **TOEFL Score** | 500.0 | 107.19200 | 6.081868 | 92.00 | 103.0000 | 107.00 | 112.00 | 120.00 |
| **CGPA** | 500.0 | 8.57644 | 0.604813 | 6.80 | 8.1275 | 8.56 | 9.04 | 9.92 |
| **Chance of Admit** | 500.0 | 0.72174 | 0.141140 | 0.34 | 0.6300 | 0.72 | 0.82 | 0.97 |

In [395…

```
jamboree.describe(include=['object','category']).T
```

Out[395]:

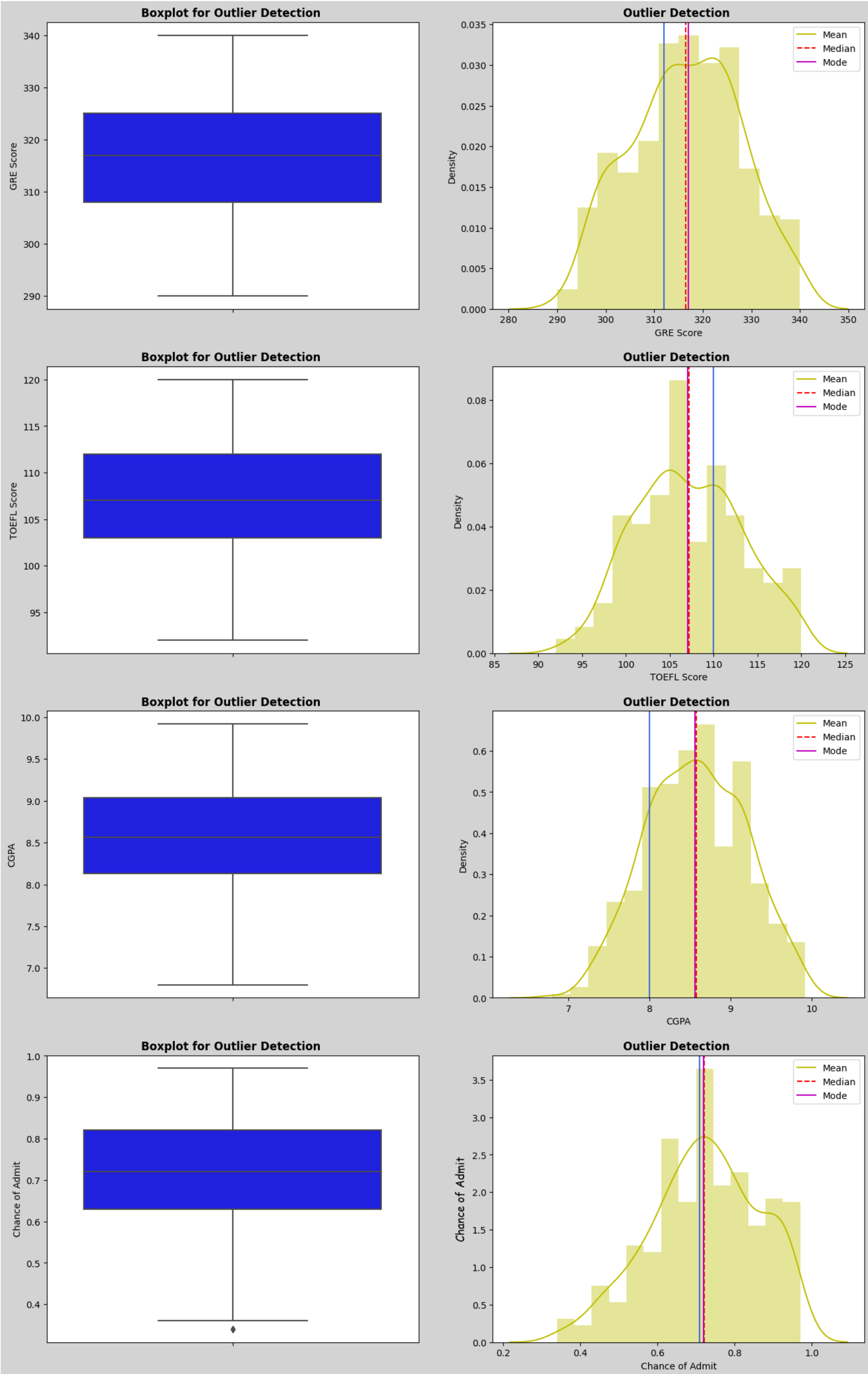
| | count | unique | top | freq |
|---|---|---|---|---|
| **University Rating** | 500.0 | 5.0 | 3.0 | 162.0 |
| **SOP** | 500.0 | 9.0 | 4.0 | 89.0 |
| **LOR** | 500.0 | 9.0 | 3.0 | 99.0 |
| **Research** | 500.0 | 2.0 | 1.0 | 280.0 |

## Numerical Variables - Outlier detection

- GRE Score
- TOEFL Score
- CGPA

In [396…

```
col_num = [ 'GRE Score', 'TOEFL Score', 'CGPA','Chance of Admit']
outlier_detect(jamboree,col_num,4,2,16,26)
```

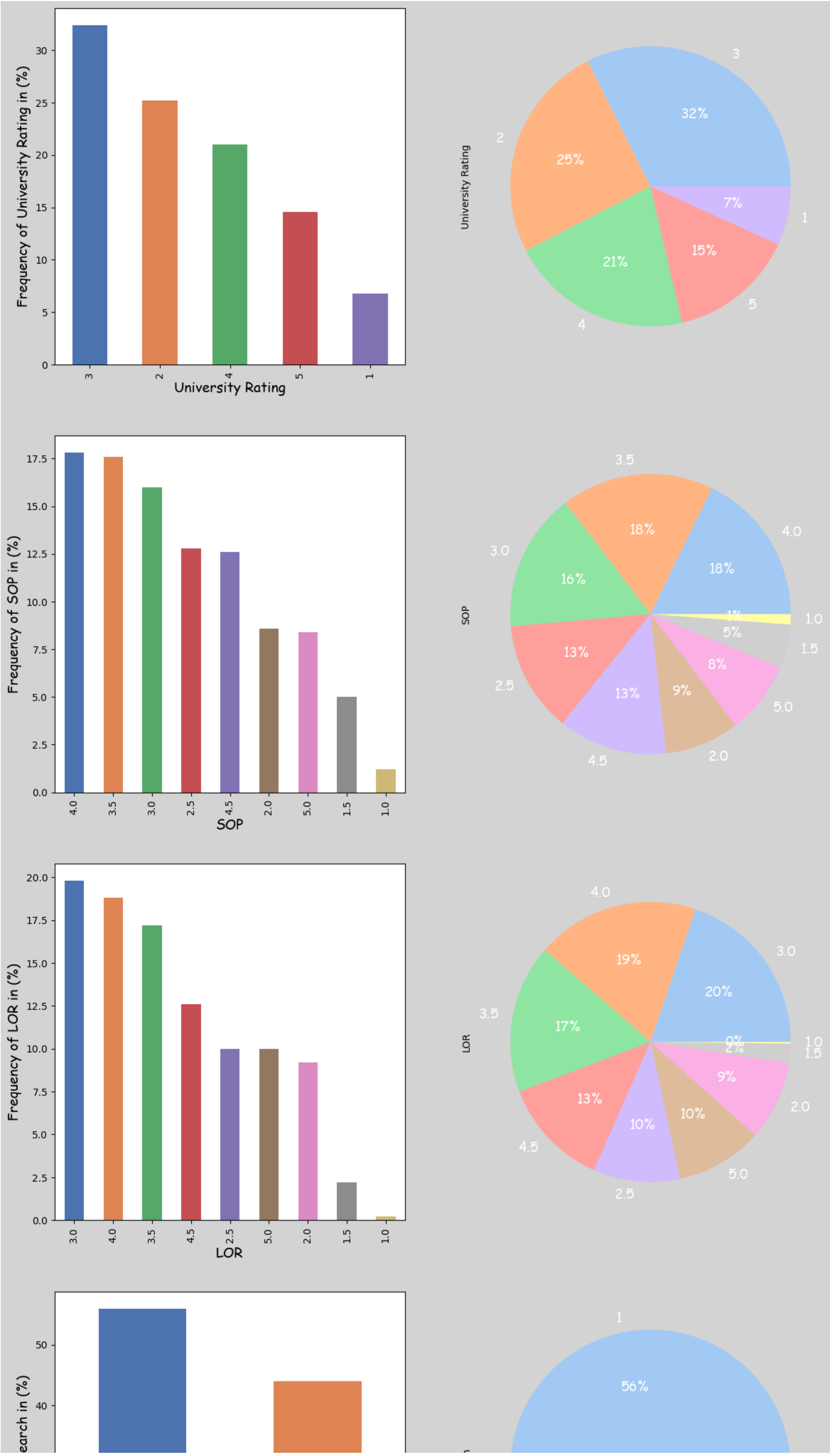

Boxplot for Outlier Detection
GRE Score
Outlier Detection
Mean
Median
Mode
Density
GRE Score
Boxplot for Outlier Detection
TOEFL Score
Outlier Detection
Mean
Median
Mode
Density
TOEFL Score
Boxplot for Outlier Detection
CGPA
Outlier Detection
Mean
Median
Mode
Density
CGPA
Boxplot for Outlier Detection
Chance of Admit
Outlier Detection
Mean
Median
Mode
Chance of Admit
Chance of Admit

# Inferences

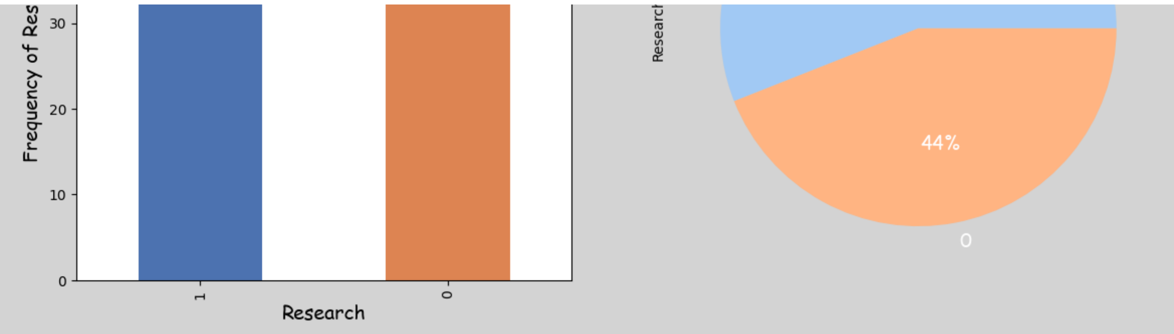
- Based on the above graph we donot have outliers for "'GRE Score', 'TOEFL Score' & 'CGPA'.
- 'Chance of Admit' is slightly left screwed. Since 'Chance of Admit' is a slightly left skewed, we don't have to handle it.

In [452…

```
cat_cols = ['University Rating', 'SOP','LOR', 'Research']
cat_analysis(jamboree,cat_cols,4,2,14,30)
```



Frequency of University Rating in (%)
University Rating
3
2
4
5
1
32%
25%
21%
15%
7%
Frequency of SOP in (%)
SOP
4.0
3.5
3.0
2.5
4.5
2.0
5.0
1.5
1.0
18%
18%
16%
13%
13%
9%
8%
5%
1%
Frequency of LOR in (%)
LOR
3.0
4.0
3.5
4.5
2.5
5.0
2.0
1.5
1.0
20%
19%
17%
13%
10%
10%
9%
2%
0%
earch in (%)
56%




### Inferences

- Among students who have done research vs those who did not, 56 % said yes and 44 % said no
- More than 50% of the data has a university rating of 3 or 2
- A majority of students (56%) have letter of recommendation values between 3.0 and 4.5

## Bivariate Analysis

### Numerical variables

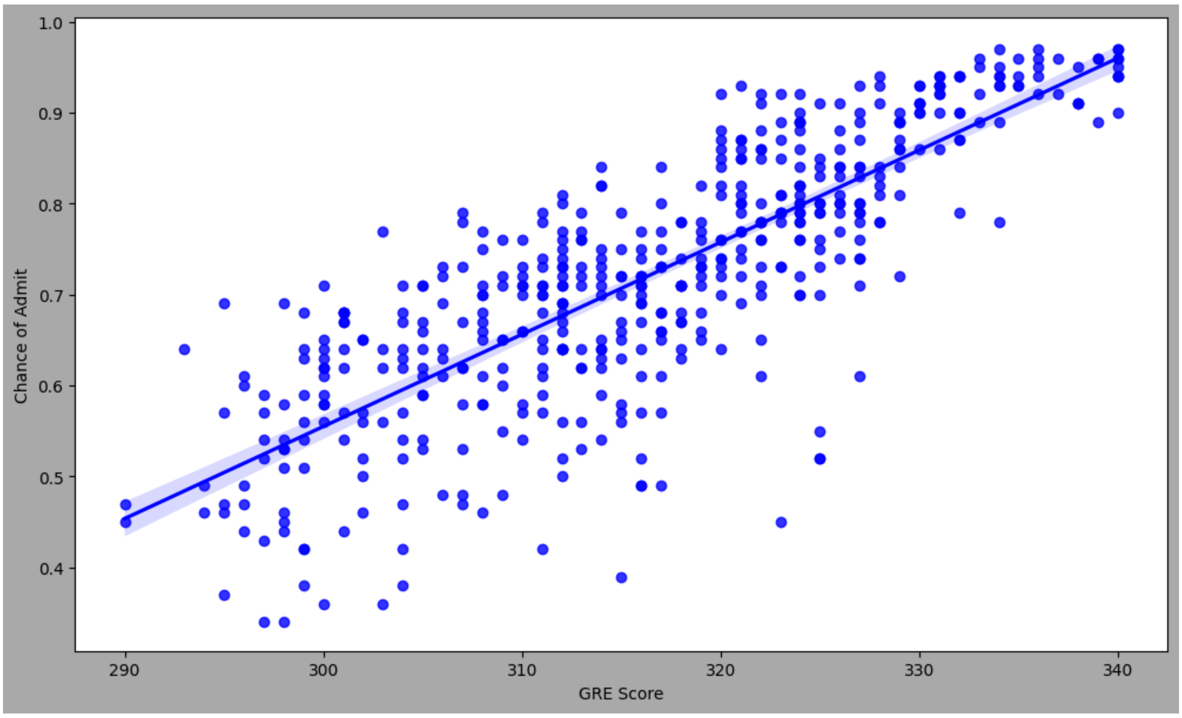
- 'GRE Score' vs 'Chance of Admit'
- 'TOEFL Score' vs 'Chance of Admit'
- 'CGPA' vs 'Chance of Admit'

### Categorical variables

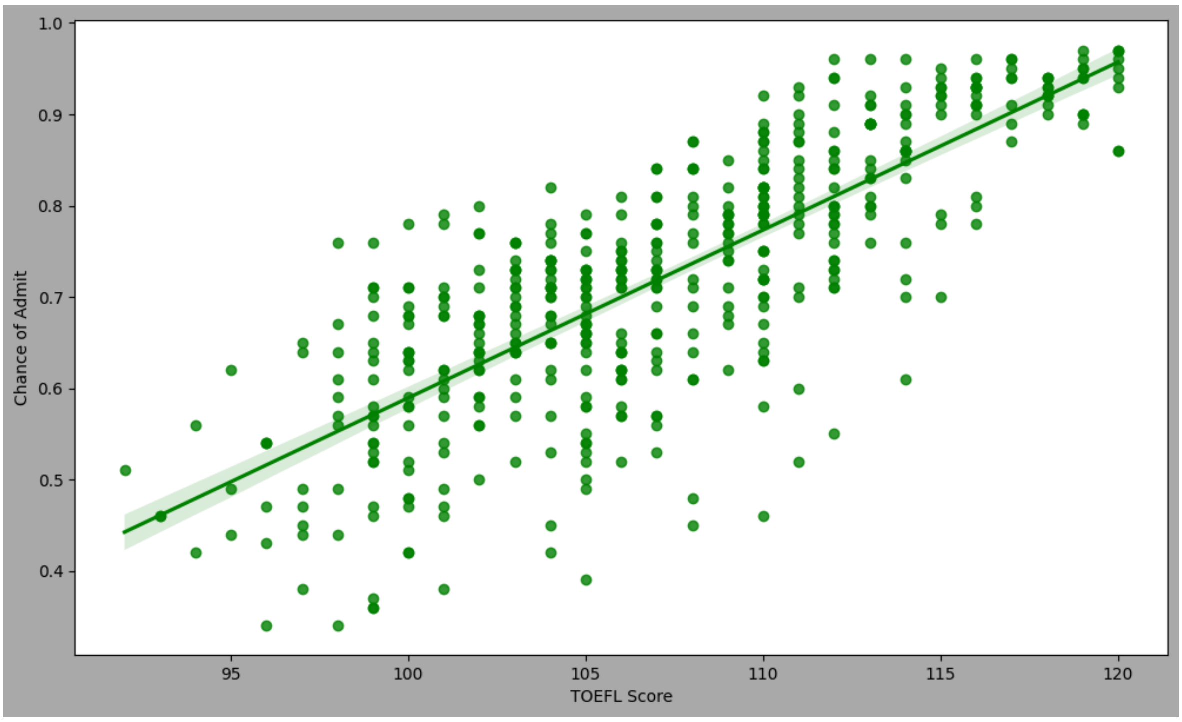
- 'Research' vs 'Chance of Admit'
- 'Univarsity rating' vs 'Chance of Admit'
- 'LOR' vs 'Chance of Admit'
- 'SOP' vs 'Chance of Admit'

In [446…

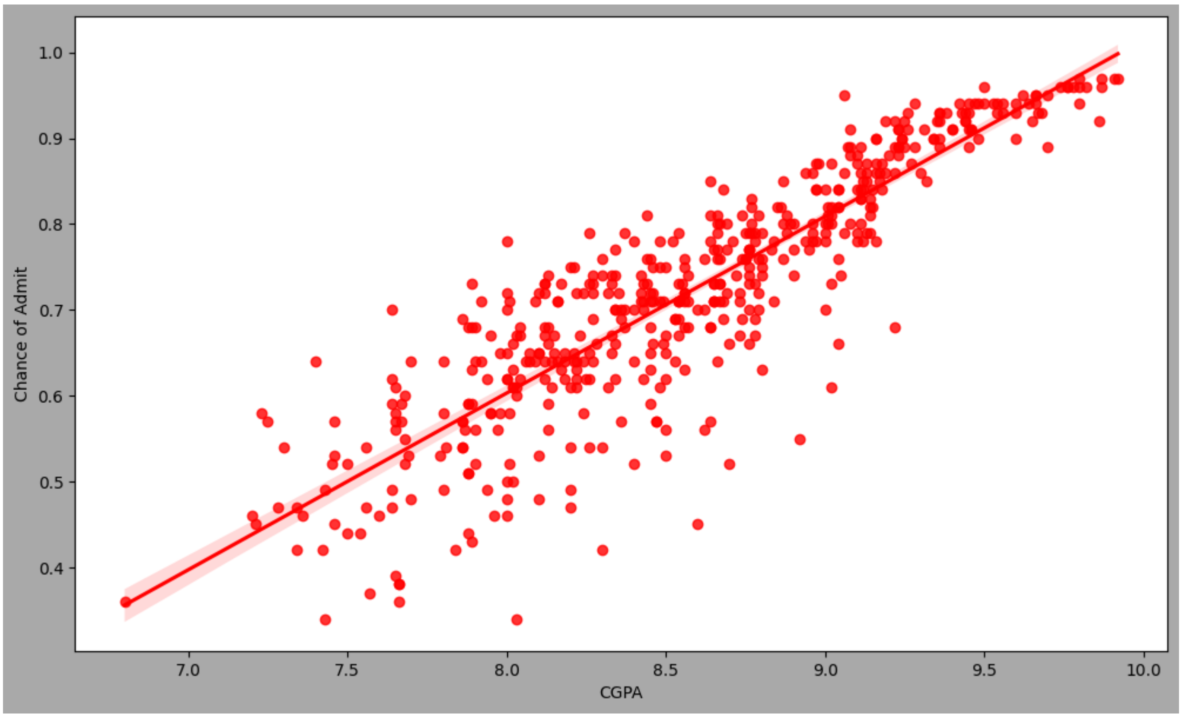
```
fig = plt.figure(figsize=(12, 7))
fig.set_facecolor(color = 'darkgrey')
sns.regplot(x='GRE Score',y='Chance of Admit',color="b",data=jamboree);

fig = plt.figure(figsize=(12, 7))
fig.set_facecolor(color = 'darkgrey')
sns.regplot(x='TOEFL Score',y='Chance of Admit',color="g",data=jamboree);

fig = plt.figure(figsize=(12, 7))
fig.set_facecolor(color = 'darkgrey')
sns.regplot(x='CGPA',y='Chance of Admit',color="r",data=jamboree);
```



Chance of Admit
GRE Score

Chance of Admit
TOEFL Score

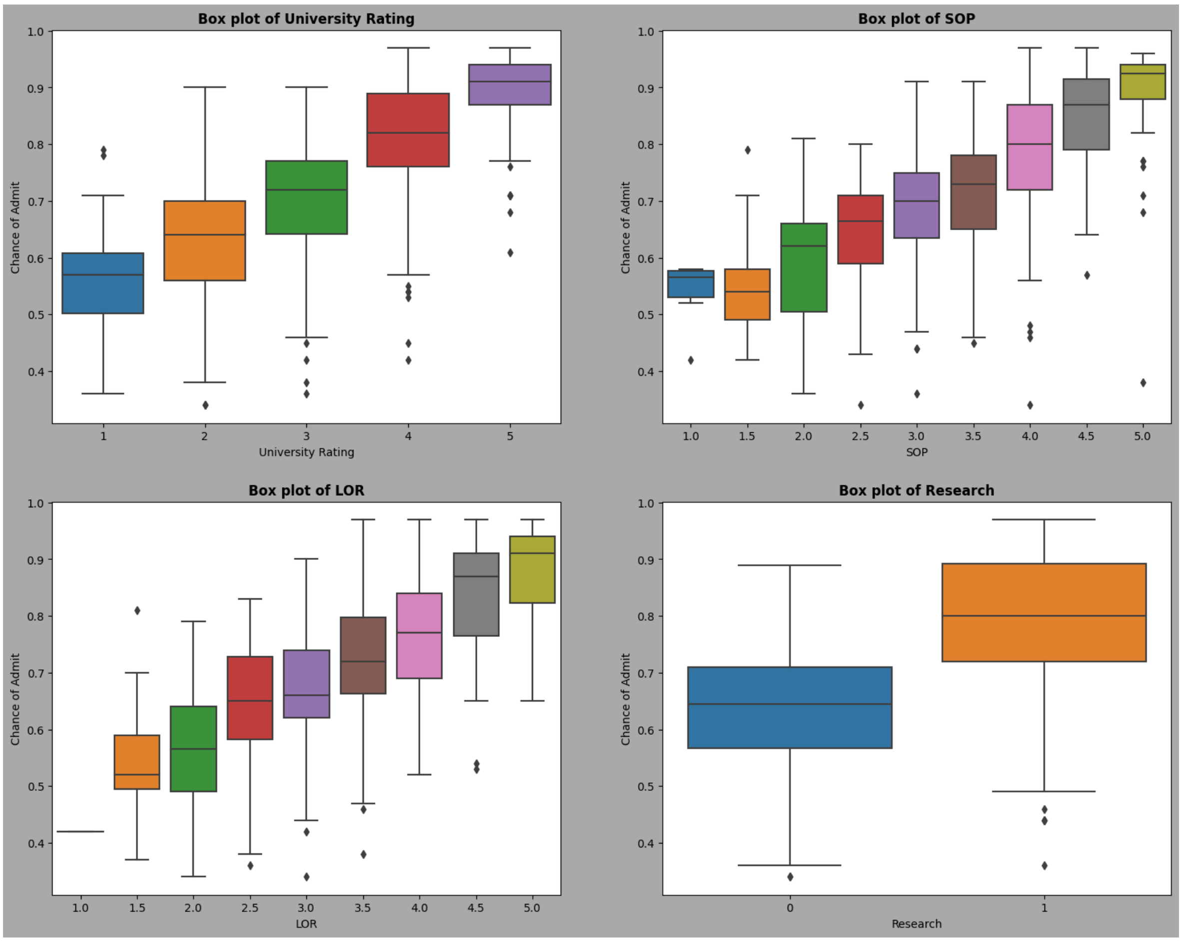



## Inferences

- A strong positive relationship exists between Chance of admit and numerical variables (GRE & TOEFL score and CGPA).

In [399…

```
box_plot(['University Rating', 'SOP', 'LOR','Research'],'Chance of Admit')
```




## Inferences

- The graph above shows an upward trend for each categorical variable. A higher rating or value increases the chance of admission

- GRE vs Chance of Admit Analysis
  - There is linear relationship between GRE and Chance of Admission
  - Higher the GRE -> Higher the chance of admission
- TOEFL vs Chance of Admit Analysis
  - There is linear relationship between TOEFL and Chance of Admission
  - Higher the TOEFL -> Higher the chance of admission
- LOR / SOP / University Rating vs Chance of Admit Analysis
  - There is no significant linear relationship between TOEFL and Chance of Admission
- CGPA vs Chance of Admit Analysis
  - There is linear relationship between TOEFL and Chance of Admission
  - Higher the CGPA -> Higher the chance of admission

- **People with higher GRE Scores also have higher TOEFL Scores which is justified because both TOEFL and GRE have a verbal section which although not similar are relatable**
- **Although there are exceptions, people with higher CGPA usually have higher GRE scores maybe because they are smart or hard working**

- **LORs are not that related with CGPA so it is clear that a persons LOR is not dependent on that persons academic excellence.**
- **Having research experience is usually related with a good LOR which might be justified by the fact that supervisors have personal interaction with the students performing research which usually results in good LORs**

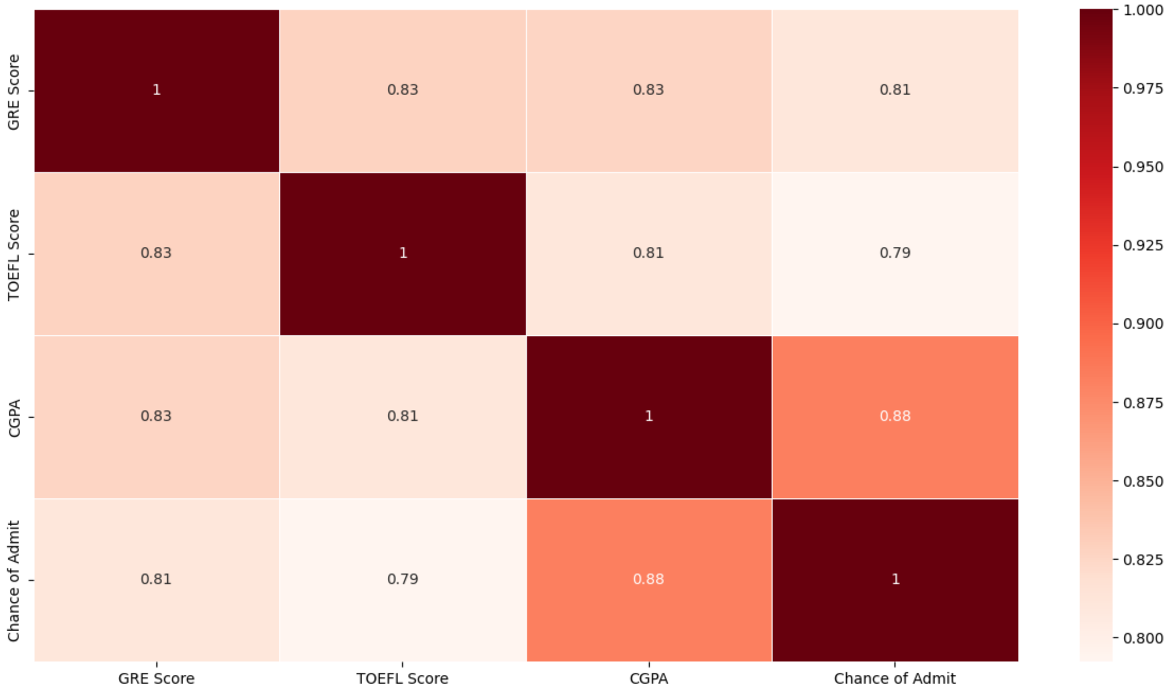
- **GRE scores and LORs are also not that related. People with different kinds of LORs have all kinds of GRE scores**

- **CGPA and SOP are not that related because Statement of Purpose is related to academic performance, but since people with good CGPA tend to be more hard working so they have good things to say in their SOP which might explain the slight move towards higher CGPA as along with good SOPs**
- **Similary, GRE Score and SOP is only slightly related**
- **Applicants with different kinds of SOP have different kinds of TOEFL Score. So the quality of SOP is not always related to the applicants English skills.**

# Correlation Analysis: Heat Map

In [444…

```
fig = plt.figure(figsize = (15, 8))
corr = jamboree.corr()
sns.heatmap(corr, linewidths=.5, annot=True, cmap="Reds")
plt.show()
```



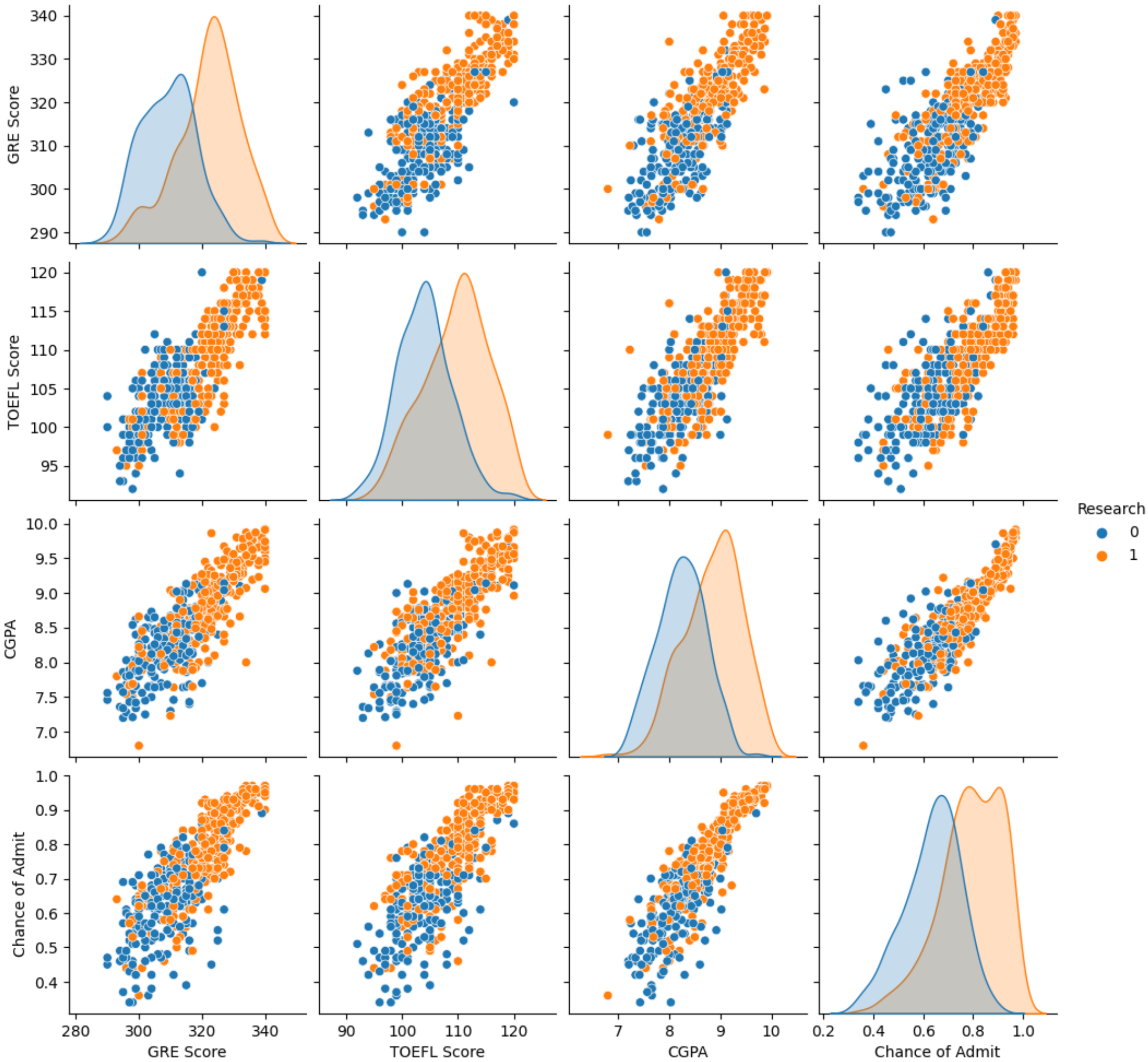

- **High Correlation**
    1. GRE Score vs TOEFL Score
    2. CGPA vs TOEFL Score
    3. CGPA vs GRE Score
    4. Chance of Admit vs CGPA
    5. GRE Score vs Chance of Admit

## Pair Plot

In [403…

```
sns.pairplot(jamboree,hue='Research' )
```

Out[403]: <seaborn.axisgrid.PairGrid at 0x14f4f0ef370>




# Inferences

### We can already see some multicolinearity between the independent variables.

- **GRE Score** , **TOFEL Score** and **CGPA** are highly correlated **(0.80)**. We should drop two of these.

## Considered only Significant variables

- When multiple features are highly correlated (above 0.80), only one feature is considered

In [404…

```
# Creating the new dataframe with only significant variables.
significant_colname = ['GRE Score', 'University Rating', 'SOP', 'LOR', 'Research
sig_jamboree = jamboree[significant_colname]
sig_jamboree.shape
```

Out[404]: (500, 6)

# Data Preprocessing

# Duplicate Value Check

In [405…
```
np.any(jamboree.duplicated())
```

Out[405]:
```
False
```

# Missing Value Check

In [406…
```
jamboree.isnull().sum()
```

Out[406]:
```
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

# Outlier Check

In [407…
```
col_num = [ 'GRE Score', 'TOEFL Score', 'CGPA','Chance of Admit']
for i in col_num:
    print("======" * 10)
    print("Mean of {}: ".format(i), jamboree[i].mean())
    print("Median of {}: ".format(i), jamboree[i].median())
```

```
============================================================
Mean of GRE Score:  316.472
Median of GRE Score:  317.0
============================================================
Mean of TOEFL Score:  107.192
Median of TOEFL Score:  107.0
============================================================
Mean of CGPA:  8.576439999999998
Median of CGPA:  8.56
============================================================
Mean of Chance of Admit:  0.72174
Median of Chance of Admit:  0.72
```

- **No outliers detected. As each and every feature overlaps its mean and median**

# Feature Engineering & Data Modelling

In [408…
```
cat_cols = ['University Rating', 'SOP', 'LOR', 'Research']
num_cols = ['GRE Score', 'TOEFL Score', 'CGPA']
target = 'Chance of Admit'
```

### Data preparation for model building

In [409…

```
from sklearn.model_selection import train_test_split

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

print("X shape: {}".format(X.shape))
print("y shape: {}".format(y.shape))
```

```
X shape: (500, 7)
y shape: (500,)
```

# Train & Test Split

In [410…

```
#X_train, X_test, y_train, y_test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

print("X_train shape: {}".format(X_train.shape))
print("X_test shape: {}".format(X_test.shape))
print("y_train shape: {}".format(y_train.shape))
print("y_test shape: {}".format(y_test.shape))
```

```
X_train shape: (400, 7)
X_test shape: (100, 7)
y_train shape: (400,)
y_test shape: (100,)
```

In [411…

```
X_train.head(5)
```

Out[411]:

| | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research |
|---|---|---|---|---|---|---|---|
| **238** | 310 | 104 | 3 | 2.0 | 3.5 | 8.37 | 0 |
| **438** | 318 | 110 | 1 | 2.5 | 3.5 | 8.54 | 1 |
| **475** | 300 | 101 | 3 | 3.5 | 2.5 | 7.88 | 0 |
| **58** | 300 | 99 | 1 | 3.0 | 2.0 | 6.80 | 1 |
| **380** | 322 | 104 | 3 | 3.5 | 4.0 | 8.84 | 1 |

In [412…

```
y_train
```

Out[412]:

```
238    0.70
438    0.67
475    0.59
58     0.36
380    0.78
       ...
255    0.79
72     0.93
396    0.84
235    0.88
37     0.58
Name: Chance of Admit, Length: 400, dtype: float64
```

# Feature standardization

In [413…
```python
# Standarization
from sklearn.preprocessing import StandardScaler
# standardize the dataset
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
#X_train, X_test, y_train, y_test
```

In [414…
```python
X_train
```

Out[414]:
```
array([[-0.53736015, -0.51949116, -0.05463584, ...,  0.00933125,
        -0.32658176, -1.11114215],
       [ 0.16363964,  0.44925692, -1.8029826 , ...,  0.00933125,
        -0.04593523,  0.89997486],
       [-1.41360989, -1.0038652 , -0.05463584, ..., -1.05709751,
        -1.13550409, -1.11114215],
       ...,
       [ 0.77701445, -0.03511712, -0.05463584, ...,  0.00933125,
         0.89505605,  0.89997486],
       [ 0.86463943,  0.61071493,  1.69371093, ...,  0.54254563,
         1.09315948,  0.89997486],
       [-1.41360989, -0.35803314, -1.8029826 , ..., -1.59031189,
        -1.26757304, -1.11114215]])
```

In [415…
```python
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(400, 7) (400,)
(100, 7) (100,)
```

# Model Building

In [416…
```python
def adjusted_r2(r2, p, n):
    """
    n: no of samples
    p: no of predictors
    r2: r2 score
    """
    adj_r2 = 1 - ((1-r2)*(n-1) / (n-p-1))
    return adj_r2

def get_metrics(y_true, y_pred, p=None):
    n = y_true.shape[0]
    mse = np.sum((y_true - y_pred)**2) / n
    rmse = np.sqrt(mse)
    mae = np.mean(np.abs(y_true - y_pred))
    score = r2_score(y_true, y_pred)
    adj_r2 = None
    if p is not None:
        adj_r2 = adjusted_r2(score, p, n)

    res = {
        "mean_absolute_error": round(mae, 2),
        "rmse": round(rmse, 2),
        "r2_score": round(score, 2),
        "adj_r2": round(adj_r2, 2)
```

```
    }
    return res
```

In [417…

```
def train_model(X_train, y_train, X_test, y_test,cols, model_name="linear", alph
    model = None
    if model_name == "lasso":
        model = Lasso(alpha=alpha)
    elif model_name == "ridge":
        model = Ridge(alpha=alpha)
    else:
        model = LinearRegression()

    model.fit(X_train, y_train)
    y_pred_train = model.predict(X_train)
    y_pred_test = model.predict(X_test)
    p = X_train.shape[1]
    train_res = get_metrics(y_train, y_pred_train, p)
    test_res = get_metrics(y_test, y_pred_test, p)

    print(f"\n----   {model_name.title()} Regression Model  ----\n")
    print(f"Train MAE: {train_res['mean_absolute_error']} Test MAE: {test_res['m
    print(f"Train RMSE: {train_res['rmse']} Test RMSE: {test_res['rmse']}")
    print(f"Train R2_score: {train_res['r2_score']} Test R2_score: {test_res['r2
    print(f"Train Adjusted_R2: {train_res['adj_r2']} Test Adjusted_R2: {test_res
    print(f"Intercept: {model.intercept_}")
    #print(len(df.columns), len(model.coef_))
    coef_df = pd.DataFrame({"Column": cols, "Coef": model.coef_})
    print(coef_df)
    print("-"*50)
    return model
```

In [418…

```
train_model(X_train, y_train, X_test, y_test,jamboree.columns[:-1], "linear")
train_model(X_train, y_train, X_test, y_test,jamboree.columns[:-1], "ridge")
train_model(X_train, y_train, X_test, y_test,jamboree.columns[:-1], "lasso", 0.0
```

```
----   Linear Regression Model  ----

Train MAE: 0.04 Test MAE: 0.04
Train RMSE: 0.06 Test RMSE: 0.06
Train R2_score: 0.82 Test R2_score: 0.82
Train Adjusted_R2: 0.82 Test Adjusted_R2: 0.81
Intercept: 0.7209250000000001
              Column      Coef
0          GRE Score  0.020910
1        TOEFL Score  0.019658
2  University Rating  0.007011
3                SOP  0.003049
4                LOR  0.013528
5               CGPA  0.070693
6           Research  0.009890
--------------------------------------------------

----   Ridge Regression Model  ----

Train MAE: 0.04 Test MAE: 0.04
Train RMSE: 0.06 Test RMSE: 0.06
Train R2_score: 0.82 Test R2_score: 0.82
Train Adjusted_R2: 0.82 Test Adjusted_R2: 0.81
Intercept: 0.7209250000000001
              Column      Coef
0          GRE Score  0.021109
1        TOEFL Score  0.019777
2  University Rating  0.007094
3                SOP  0.003201
4                LOR  0.013571
5               CGPA  0.070053
6           Research  0.009907
--------------------------------------------------

----   Lasso Regression Model  ----

Train MAE: 0.04 Test MAE: 0.04
Train RMSE: 0.06 Test RMSE: 0.06
Train R2_score: 0.82 Test R2_score: 0.82
Train Adjusted_R2: 0.82 Test Adjusted_R2: 0.81
Intercept: 0.7209250000000001
              Column      Coef
0          GRE Score  0.020837
1        TOEFL Score  0.019397
2  University Rating  0.006829
3                SOP  0.002824
4                LOR  0.013077
5               CGPA  0.070842
6           Research  0.009299
--------------------------------------------------
```

Out[418]:

```
Lasso
Lasso(alpha=0.001)
```

- Since model is not overfitting, Results for Linear, Ridge and Lasso are the same.
- R2_score and Adjusted_r2 are almost the same. Hence there are no unnecessary independent variables in the data.

# Linear Regression Model - Assumption Test

## Mutlicollinearity Check

In [419…

```
def vif(newdf):
    # VIF dataframe
    vif_data = pd.DataFrame()
    vif_data["feature"] = newdf.columns

    # calculating VIF for each feature
    vif_data["VIF"] = [variance_inflation_factor(newdf.values, i)
                              for i in range(len(newdf.columns))]
    return vif_data
```

In [420…

```
res = vif(jamboree.iloc[:,:-1])
res
```

Out[420]:

| | feature | VIF |
|---|---|---|
| **0** | GRE Score | 1308.061089 |
| **1** | TOEFL Score | 1215.951898 |
| **2** | University Rating | 20.933361 |
| **3** | SOP | 35.265006 |
| **4** | LOR | 30.911476 |
| **5** | CGPA | 950.817985 |
| **6** | Research | 2.869493 |

In [421…

```
# drop GRE Score and again calculate the VIF
res = vif(jamboree.iloc[:, 1:-1])
res
```

Out[421]:

| | feature | VIF |
|---|---|---|
| **0** | TOEFL Score | 639.741892 |
| **1** | University Rating | 19.884298 |
| **2** | SOP | 33.733613 |
| **3** | LOR | 30.631503 |
| **4** | CGPA | 728.778312 |
| **5** | Research | 2.863301 |

In [422...

```
# # drop TOEFL Score and again calculate the VIF
res = vif(jamboree.iloc[:,2:-1])
res
```

Out[422]:

| | feature | VIF |
|---|---|---|
| 0 | University Rating | 19.777410 |
| 1 | SOP | 33.625178 |
| 2 | LOR | 30.356252 |
| 3 | CGPA | 25.101796 |
| 4 | Research | 2.842227 |

In [423...

```
# Now lets drop the SOP and again calculate VIF
res = vif(jamboree.iloc[:,2:-1].drop(columns=['SOP']))
res
```

Out[423]:

| | feature | VIF |
|---|---|---|
| 0 | University Rating | 15.140770 |
| 1 | LOR | 26.918495 |
| 2 | CGPA | 22.369655 |
| 3 | Research | 2.819171 |

In [424...

```
# Lets drop the LOR as well
newdf = jamboree.iloc[:,2:-1].drop(columns=['SOP'])
newdf = newdf.drop(columns=['LOR'], axis=1)
res = vif(newdf)
res
```

Out[424]:

| | feature | VIF |
|---|---|---|
| 0 | University Rating | 12.498400 |
| 1 | CGPA | 11.040746 |
| 2 | Research | 2.783179 |

In [425...

```
# drop the University Rating
newdf = newdf.drop(columns=['University Rating'])
res = vif(newdf)
res
```

Out[425]:

| | feature | VIF |
|---|---|---|
| 0 | CGPA | 2.455008 |
| 1 | Research | 2.455008 |

In [426...

```
# now again train the model with these only two features
X = jamboree[['CGPA', 'Research']]
sc = StandardScaler()
```

```
X = sc.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
```

In [427…

```
model = train_model(X_train, y_train, X_test, y_test, ['CGPA', 'Research'], "lin
train_model(X_train, y_train, X_test, y_test, ['CGPA', 'Research'], "ridge")
train_model(X_train, y_train, X_test, y_test, ['CGPA', 'Research'], "lasso", 0.0
```

```
----   Linear Regression Model  ----

Train MAE: 0.05 Test MAE: 0.05
Train RMSE: 0.06 Test RMSE: 0.07
Train R2_score: 0.78 Test R2_score: 0.81
Train Adjusted_R2: 0.78 Test Adjusted_R2: 0.81
Intercept: 0.7247774222727991
     Column      Coef
0      CGPA  0.112050
1  Research  0.020205
--------------------------------------------------

----   Ridge Regression Model  ----

Train MAE: 0.05 Test MAE: 0.05
Train RMSE: 0.06 Test RMSE: 0.07
Train R2_score: 0.78 Test R2_score: 0.81
Train Adjusted_R2: 0.78 Test Adjusted_R2: 0.81
Intercept: 0.7247830300095277
     Column      Coef
0      CGPA  0.111630
1  Research  0.020362
--------------------------------------------------

----   Lasso Regression Model  ----

Train MAE: 0.05 Test MAE: 0.05
Train RMSE: 0.06 Test RMSE: 0.07
Train R2_score: 0.78 Test R2_score: 0.81
Train Adjusted_R2: 0.78 Test Adjusted_R2: 0.81
Intercept: 0.7247713356661623
     Column      Coef
0      CGPA  0.111344
1  Research  0.019571
--------------------------------------------------
```

Out[427]:

```
▾ Lasso
Lasso(alpha=0.001)
```

After removing collinear features using VIF and using only two features. R2_score and Adjusted_r2 are still the same as before the testing dataset.

### Mean of Residuals

It is clear from RMSE that Mean of Residuals is almost zero.

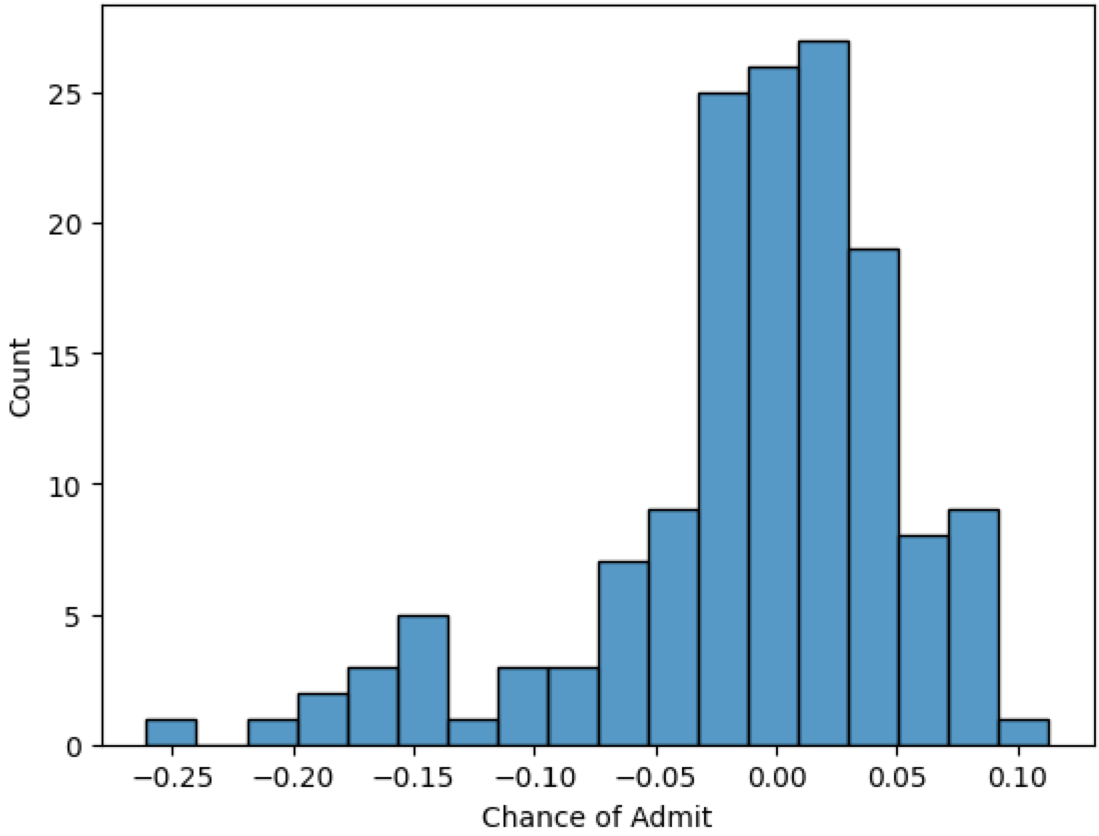
### Linearity of variables

It is quite clear from EDA that independent variables are linearly dependent on the target variables.

### Normality of Residuals

In [428…

```
y_pred = model.predict(X_test)
residuals = (y_test - y_pred)
sns.histplot(residuals)
plt.show()
```



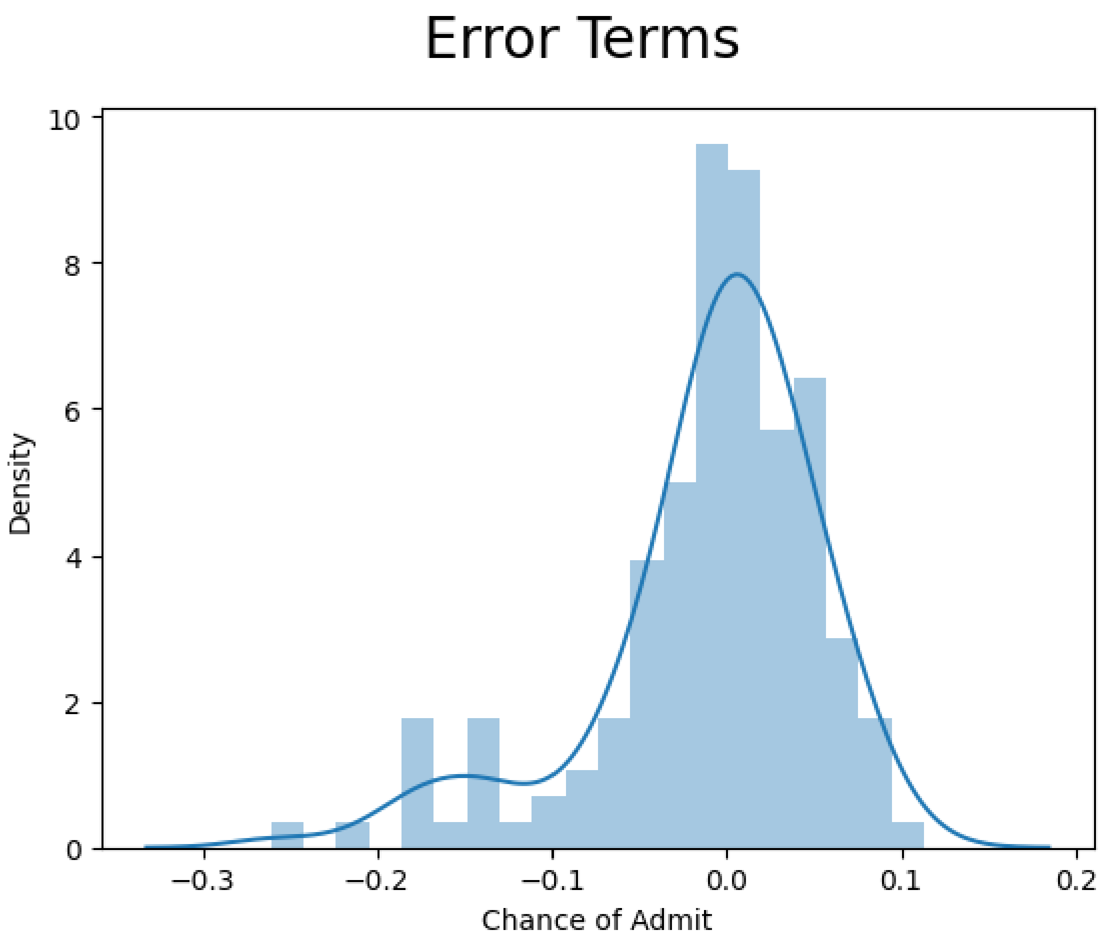

In [ ]:

```

```

# Residual Analysis of the Model

In [429…

```
# Plot the histogram of the error terms
fig = plt.figure()
sns.distplot((y_test - y_pred), bins = 20)
fig.suptitle('Error Terms', fontsize = 20)    # Plot heading
plt.show()
```




**Inferences**

- Error terms seem to be approximately normally distributed, so the assumption on the linear modeling seems to be fulfilled.

In [430…

```
(y_test - y_pred).mean()
```

Out[430]: -0.010124740909331383

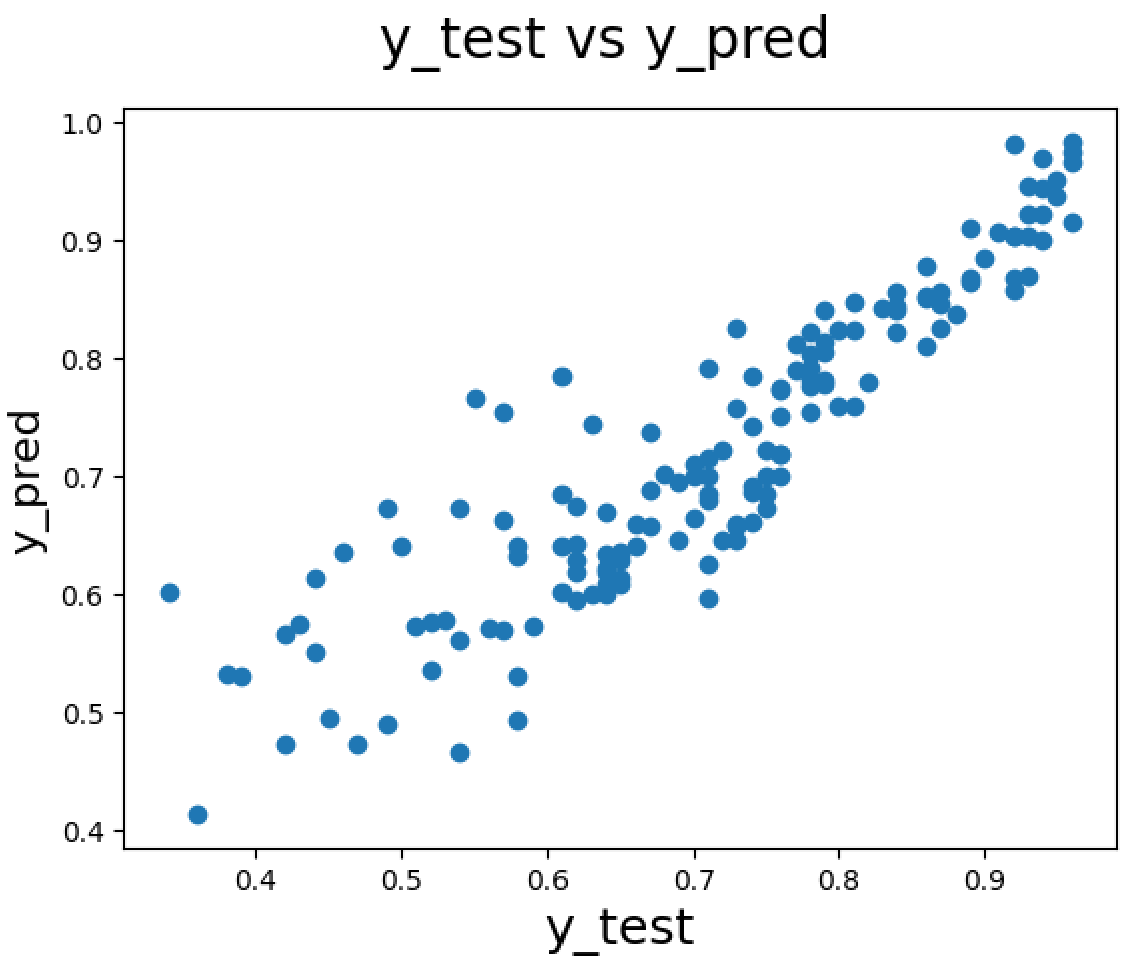
**Mean of Residuals:** `-0.010124740909331383`

# Linearity of Variables

In [431…

```
# Plotting y_test and y_pred to understand the spread.
fig = plt.figure()
plt.scatter(y_test,y_pred)
fig.suptitle('y_test vs y_pred', fontsize=20)              # Plot heading
plt.xlabel('y_test', fontsize=18)                          # X-label
plt.ylabel('y_pred', fontsize=16)
```

Out[431]: Text(0, 0.5, 'y_pred')

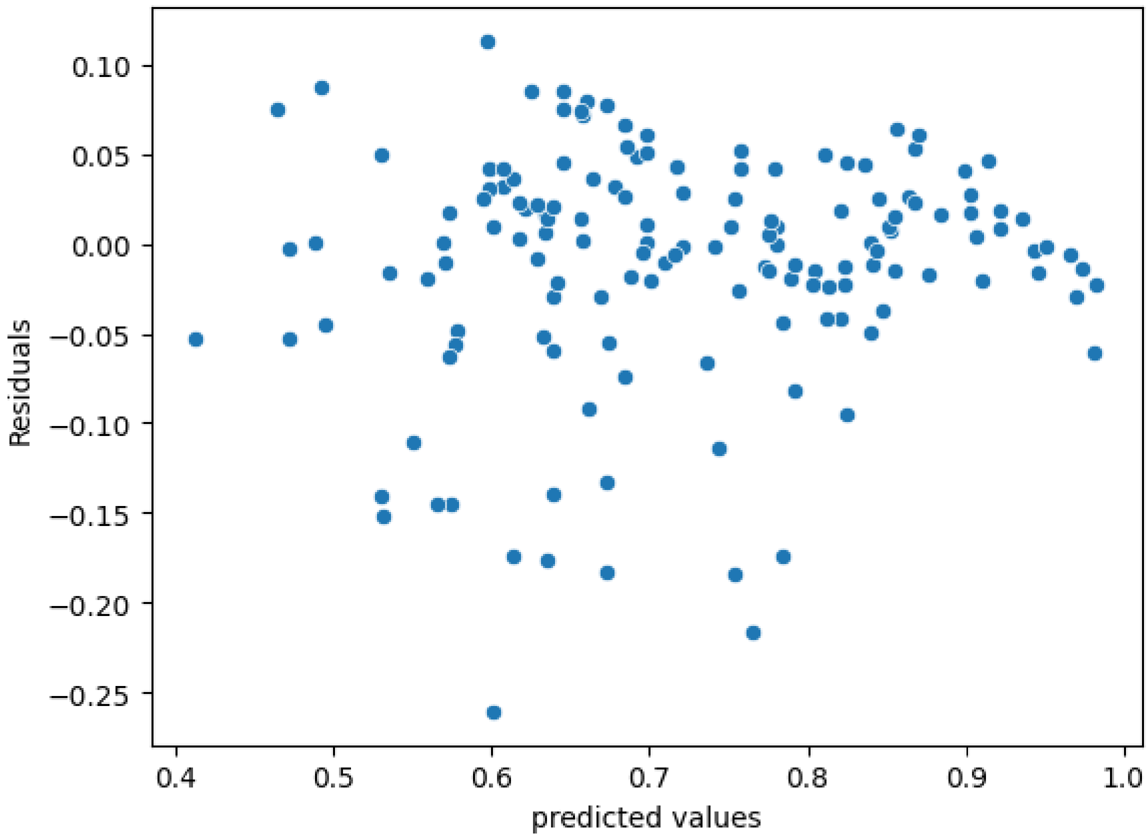



## Test for Homoscedasticity

In [432…

```
residuals = y_test - y_pred
p = sns.scatterplot(x=y_pred,y=residuals)
plt.xlabel('predicted values')
plt.ylabel('Residuals')
# plt.ylim(-0.4,0.4)
# plt.xlim(0,1)
```

Out[432]: Text(0, 0.5, 'Residuals')




In [433…
```
import statsmodels.stats.api as sas
from statsmodels.compat import lzip
name=['F statistics','p-value']
test=sas.het_goldfeldquandt(residuals,X_test)
lzip(name,test)
```

Out[433]:
```
[('F statistics', 1.386773717455843), ('p-value', 0.08241454308683209)]
```

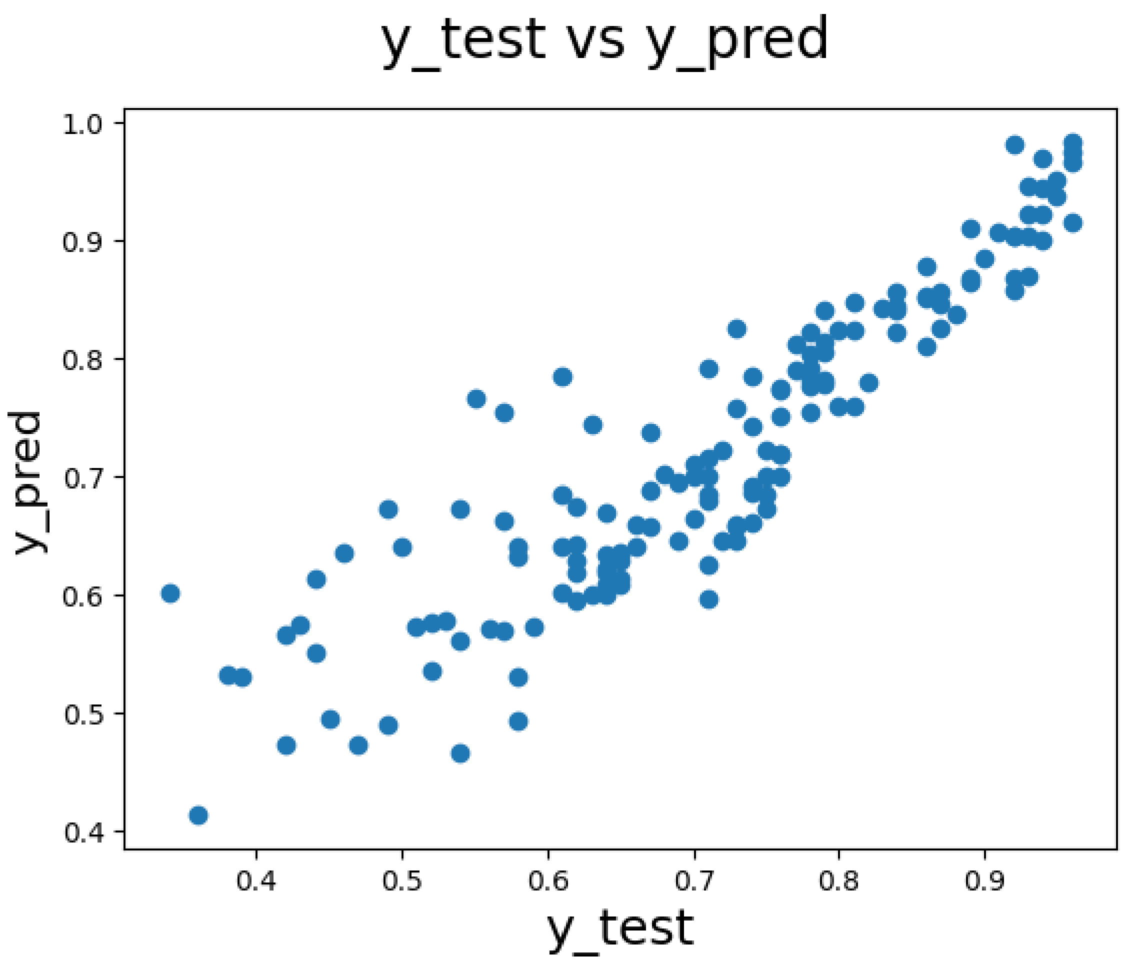
**Inferences**

- Here null hypothesis is - error terms are homoscedastic and since p-values >0.05, we fail to reject the null hypothesis

# Normality of Residual

In [434…
```
# Plotting y_test and y_pred to understand the spread.
fig = plt.figure()
plt.scatter(y_test.values, y_pred)
fig.suptitle('y_test vs y_pred', fontsize=20)              # Plot heading
plt.xlabel('y_test', fontsize=18)                          # X-label
plt.ylabel('y_pred', fontsize=16)
```

Out[434]:
```
Text(0, 0.5, 'y_pred')
```

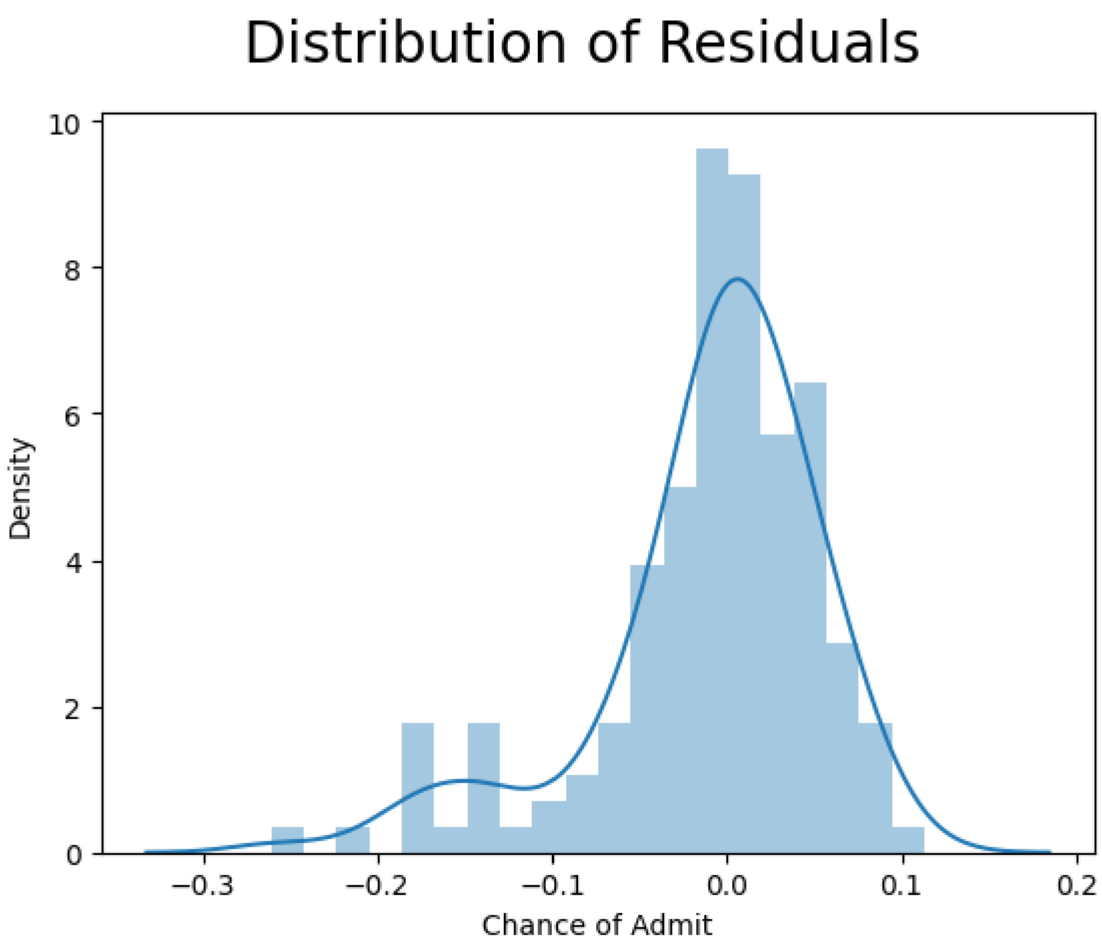



**Inferences**

- y_test and y_pred overlaps for the most of the datapoints

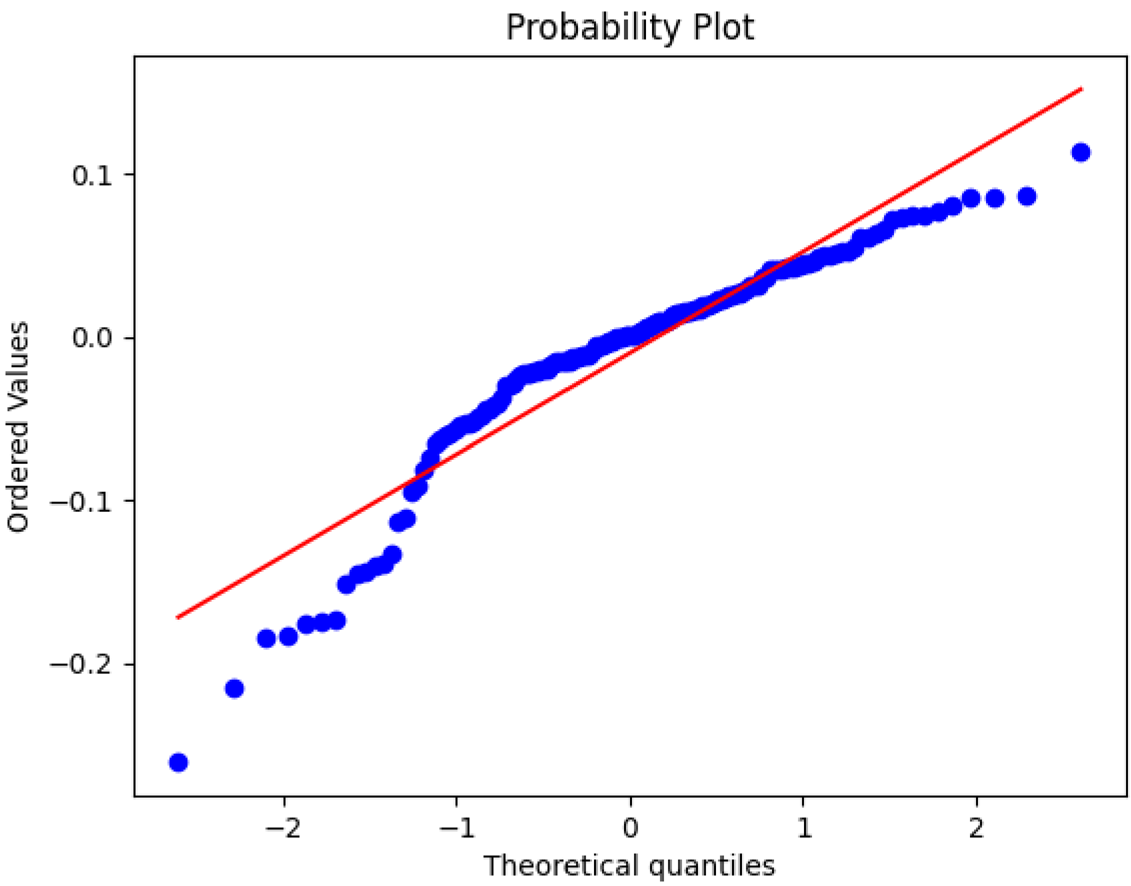
In [435…

```
fig = plt.figure()
sns.distplot(residuals, bins = 20)
fig.suptitle('Distribution of Residuals', fontsize = 20)
plt.show()
```




In [436…

```
from scipy import stats
stats.probplot(residuals, plot=plt)
plt.show()
```



**Inferences**

- QQ Plots suggest majority of the data points fit the regression line.

# Model performance evaluation

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

In [437…

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

print("======" * 10)
print('Mean Absolute Error: ',mean_absolute_error(y_test.values,y_pred))
print("======" * 10)
print('Root Mean Square Error: ',np.sqrt(mean_squared_error(y_test.values,y_pred
print("======" * 10)
r2Score = r2_score(y_test, y_pred)
print('R2 Score: ', r2Score)
print("======" * 10)
aR2Score = 1 - (1-r2Score/(len(y_test)-X_test_new.shape[1]-1))
print('Adjusted. R2 Score: ', r2Score)
print("======" * 10)
```

```
============================================================
Mean Absolute Error:  0.045288366763772 49
============================================================
Root Mean Square Error:  0.06551830407461516
============================================================
R2 Score:  0.8083230375560084
============================================================
Adjusted. R2 Score:  0.8083230375560084
============================================================
```

## Inference

### Error term

An error term appears in a statistical model, like a regression model, to indicate the uncertainty in the model.

- R-Squared (Accuracy Score) - 0.72
  - This statistic indicates the percentage of the variance in the dependent variable that the independent variables explain collectively.As seen above our residual plot looks good, which means we donot have any bias in our model.
  - R-squared does not indicate if a regression model provides an adequate fit to your data. **A good model can have a low R2 value.** On the other hand, a biased model can have a high R2 value
- Mean Absolute Error - 0.42
  - MAE describes the typical magnitude of the residuals. Small MAE suggests the model is great at prediction, while a large MAE suggests that your model may have trouble in certain areas. There is scope of improvement.

- Root Mean Square Error - 0.54
  - RMSE is defined as the square root of the average squared difference between the predicted and the actual score. The lower the RMSE, the better a model fits a dataset
  - A huge difference between the RMSE and MAE indicates outliers. A smaller difference indicates less outliers in our case.
- Mean Square Error - 0.29
  - MSE equation is most apparent with the presence of outliers in our data.
  - While each residual in MAE contributes proportionally to the total error, the error grows quadratically in MSE.This means that outliers in our data will contribute to much higher total error in the MSE than they would the MAE.
- Mean Absolute Percentage Error - 2%
  - MAPE is biased towards predictions that are systematically less than the actual values themselves.MAPE will be lower when the prediction is lower than the actual compared to a prediction that is higher by the same amount

# Performance test Train & Test Dataset

In [ ]:

```
print("======" * 10)
Trainr2Score = r2_score(y_train, y_train_pred)
print('Train R2 Score: ', Trainr2Score)
print("======" * 10)
Testr2Score = r2_score(y_test, y_pred)
print('Test R2 Score: ', Testr2Score)
```

- **Train R2_score: 0.78**
- **Test R2_score: 0.81**

In [442…

```
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.724
Model:                            OLS   Adj. R-squared:                  0.722
Method:                 Least Squares   F-statistic:                     345.8
Date:                Wed, 06 Dec 2023   Prob (F-statistic):          3.32e-110
Time:                        23:02:07   Log-Likelihood:                 470.65
No. Observations:                 400   AIC:                            -933.3
Df Residuals:                     396   BIC:                            -917.3
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.7148      0.004    190.667      0.000       0.707       0.722
GRE Score      0.0869      0.005     17.366      0.000       0.077       0.097
LOR            0.0409      0.004      9.184      0.000       0.032       0.050
Research       0.0110      0.005      2.386      0.017       0.002       0.020
==============================================================================
Omnibus:                       43.888   Durbin-Watson:                   2.011
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               64.076
Skew:                          -0.741   Prob(JB):                     1.22e-14
Kurtosis:                       4.283   Cond. No.                         2.27
==============================================================================

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

# Actionable Insights and Recommendations

## Insights

1. Multicollinearity is present in the data.
2. After removing collinear features there are only two variables which are important in making predictions for the target variables.
3. Indepedent variables are linearly correlated with dependent variables.

## Recommendations

1. `CGPA and Research` are the only two variables which are important in making the prediction for `Chance of Admit`.

2. `CGPA` is the most important varibale in making the prediction for the `Chance of Admit`.

3. Following are the final model results on the test data:

   - **RMSE:** 0.07
   - **MAE:** 0.05
   - **R2_score:** 0.81
   - **Adjusted_R2:** 0.81

4. R-sqaured and Adjusted R-squared (extent of fit) - 0.83 and 0.82 - 85% variance explained.

5. F-stats and Prob(F-stats) (overall model fit) - 387.9 and 1.03e-149(approx. 0.0) - Model fit is significant and explained 82% variance is just not by chance.

6. p-values - p-values for all the coefficients seem to be less than the significance level of 0.05. - meaning that all the predictors are statistically significant.

7. There is lot of chance for the model improvement by tunining the parameters.

8. Currently this models attains accuracy around 80%. This can be improved further by doing some feature engg.

9. As the dataset is strictly provided for the Indian perspective. This model is not generalized, there is scope for the generalization of this model.

10. LogLikelihood is around 570 which indicates model is significantly fit.

11. Performance of trainning and test data is almost same indicates the model will work significantly on unseen data.

12. While observing the model and according to test assumptions - We can infer errors are homoscedasticity according to p-value

13. While observing the linearity of residual there is no significant pattern found which indicates the residual plots are not correlated

14. While observing the normality of residual - the distribution resembles like bell-shaped and the reg. line fits almost every point

## Suggestions

### Graduation Admission - Can use the above model to create new feature where students/learners can come to their website and check their probability of getting into the IVY league college.

#### Key features which influence the chance of Admit are

- GRE Score
- TOEFL Score
- CGPA
- LOR greater or equal to than 4.5
- --> GRE , TOEFL and CGPA are highly correlated and CGPA alone can explain the model performance alone
- --> Research Feature is also important

### A higher University rating will increases the chance of admission

In [ ]:

```

```