[redacted]

August 4, 2024

1 Jamboree Education - Linear Regression Case Study

1.1 About Data

Jamboree is a renowned educational institution that has successfully assisted numerous students in gaining admission to top colleges abroad. With their proven problem-solving methods, they have helped students achieve exceptional scores on exams like GMAT, GRE, and SAT with minimal effort.

To further support students, Jamboree has recently introduced a new feature on their website. This feature enables students to assess their probability of admission to Ivy League colleges, considering the unique perspective of Indian applicants.

By conducting a thorough analysis, we can assist Jamboree in understanding the crucial factors impacting graduate admissions and their interrelationships. Additionally, we can provide predictive insights to determine an individual's admission chances based on various variables.

1.2 Objective

As a data scientist/ML engineer hired by Jamboree, your primary objective is toanalyze the given dataset and derive valuable insights from it. Additionally, utilize the dataset to construct a predictive model capable of estimating an applicant's likelihood of admission based on the available features.

Solving this business case holds immense importance for aspiring data scientists and ML engineers.

Building predictive models using machine learning is widely popular among the data scientists/ML engineers. By working through this case study, individuals gain hands-on experience and practical skills in the field.

Additionally, it will enhance one's ability to communicate with the stakeholders involved in data-related projects and help the organization take better, data-driven decisions.s.

1.3 Importing Libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

```
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from scipy import stats

import statsmodels.api as sm
import statsmodels.stats.api as sms

import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
Jamboree_data = pd.read_csv('Jamboree_Admission.csv')
```

[3]:
```
Jamboree_data
```

[3]:
```
     Serial No.  GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  \
0             1        337          118                  4  4.5   4.5  9.65
1             2        324          107                  4  4.0   4.5  8.87
2             3        316          104                  3  3.0   3.5  8.00
3             4        322          110                  3  3.5   2.5  8.67
4             5        314          103                  2  2.0   3.0  8.21
..          ...        ...          ...                ...  ...   ...   ...
495         496        332          108                  5  4.5   4.0  9.02
496         497        337          117                  5  5.0   5.0  9.87
497         498        330          120                  5  4.5   5.0  9.56
498         499        312          103                  4  4.0   5.0  8.43
499         500        327          113                  4  4.5   4.5  9.04

     Research  Chance of Admit
0           1              0.92
1           1              0.76
2           1              0.72
3           1              0.80
4           0              0.65
..        ...               ...
495         1              0.87
496         1              0.96
497         1              0.93
498         0              0.73
499         0              0.84

[500 rows x 9 columns]
```

[4]:
```
copy_Jamboree_data = Jamboree_data.copy()
```

[5]:
```
copy_Jamboree_data
```

[5]:

	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
..	…	…	…	…	…	…	…	…	…
495	496	332	108	5	4.5	4.0	9.02	1	0.87
496	497	337	117	5	5.0	5.0	9.87	1	0.96
497	498	330	120	5	4.5	5.0	9.56	1	0.93
498	499	312	103	4	4.0	5.0	8.43	0	0.73
499	500	327	113	4	4.5	4.5	9.04	0	0.84

[500 rows x 9 columns]

1.4 Exploratory Data Analysis

[6]:
```
#Get basic information about the DataFrame
copy_Jamboree_data.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
```

1.4.1 Drop any irrelevant column present in the dataset

[7]:
```
copy_Jamboree_data = copy_Jamboree_data.drop(columns = 'Serial No.')
```

First column i.e. 'Serial No.' which was observed as unique row identifier which was dropped and was not required for model buildingry.

1.4.2 Checking the Shape of Dataset

[8]:
```
# Finding the number of rows and columns given in the dataset
print(f"Total number of rows in the dataset : {copy_Jamboree_data.
 ↪shape[0]}\nTotal number of columns in the dataset : {copy_Jamboree_data.
 ↪shape[1]}")
```

```
Total number of rows in the dataset : 500
Total number of columns in the dataset : 8
```

1.4.3 Checking the Datatype of each column

[9]:
```
copy_Jamboree_data.dtypes
```

[9]:
```
GRE Score              int64
TOEFL Score            int64
University Rating      int64
SOP                  float64
LOR                  float64
CGPA                 float64
Research               int64
Chance of Admit      float64
dtype: object
```

1.4.4 Checking the range of attributes

[10]:
```
for _ in copy_Jamboree_data.columns:
    print()
    print(f'Range of {_} column is from {copy_Jamboree_data[_].min()} to␣
 ↪{copy_Jamboree_data[_].max()}')
    print('-'*120)
```

```
Range of GRE Score column is from 290 to 340
------------------------------------------------------------------------------------------
```

```
----------------------------------------

Range of TOEFL Score column is from 92 to 120
--------------------------------------------------------------------------------
----------------------------------------

Range of University Rating column is from 1 to 5
--------------------------------------------------------------------------------
----------------------------------------

Range of SOP column is from 1.0 to 5.0
--------------------------------------------------------------------------------
----------------------------------------

Range of LOR  column is from 1.0 to 5.0
--------------------------------------------------------------------------------
----------------------------------------

Range of CGPA column is from 6.8 to 9.92
--------------------------------------------------------------------------------
----------------------------------------

Range of Research column is from 0 to 1
--------------------------------------------------------------------------------
----------------------------------------

Range of Chance of Admit  column is from 0.34 to 0.97
--------------------------------------------------------------------------------
----------------------------------------
```

1.4.5 Statistical summary of the entire dataset

[11]:
```
# Summary statistics for all columns
copy_Jamboree_data.describe(include="all").T
```

[11]:

	count	mean	std	min	25%	50%	\
GRE Score	500.0	316.47200	11.295148	290.00	308.0000	317.00	
TOEFL Score	500.0	107.19200	6.081868	92.00	103.0000	107.00	
University Rating	500.0	3.11400	1.143512	1.00	2.0000	3.00	
SOP	500.0	3.37400	0.991004	1.00	2.5000	3.50	
LOR	500.0	3.48400	0.925450	1.00	3.0000	3.50	
CGPA	500.0	8.57644	0.604813	6.80	8.1275	8.56	
Research	500.0	0.56000	0.496884	0.00	0.0000	1.00	
Chance of Admit	500.0	0.72174	0.141140	0.34	0.6300	0.72	

	75%	max
GRE Score	325.00	340.00

```
TOEFL Score        112.00  120.00
University Rating    4.00    5.00
SOP                  4.00    5.00
LOR                  4.00    5.00
CGPA                 9.04    9.92
Research             1.00    1.00
Chance of Admit      0.82    0.97
```

[12]:
```
copy_Jamboree_data.cov()
```

[12]:
```
                   GRE Score  TOEFL Score  University Rating       SOP  \
GRE Score         127.580377    56.825026           8.206605  6.867206
TOEFL Score        56.825026    36.989114           4.519150  3.883960
University Rating   8.206605     4.519150           1.307619  0.825014
SOP                 6.867206     3.883960           0.825014  0.982088
LOR                 5.484521     3.048168           0.644112  0.608701
CGPA                5.641944     2.981607           0.487761  0.426845
Research            3.162004     1.411303           0.242645  0.200962
Chance of Admit     1.291862     0.680046           0.111384  0.095691

                       LOR       CGPA  Research  Chance of Admit
GRE Score         5.484521  5.641944  3.162004          1.291862
TOEFL Score       3.048168  2.981607  1.411303          0.680046
University Rating 0.644112  0.487761  0.242645          0.111384
SOP               0.608701  0.426845  0.200962          0.095691
LOR               0.856457  0.356807  0.171303          0.084296
CGPA              0.356807  0.365799  0.150655          0.075326
Research          0.171303  0.150655  0.246894          0.038282
Chance of Admit   0.084296  0.075326  0.038282          0.019921
```

[13]:
```
copy_Jamboree_data.columns
```

[13]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

[14]:
```
#Viewing and understanding few 5 rows of the Jamboree dataframe
copy_Jamboree_data.head()
```

[14]:
```
   GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  Research  \
0        337          118                  4  4.5   4.5  9.65         1
1        324          107                  4  4.0   4.5  8.87         1
2        316          104                  3  3.0   3.5  8.00         1
3        322          110                  3  3.5   2.5  8.67         1
4        314          103                  2  2.0   3.0  8.21         0

   Chance of Admit
```

```
0              0.92
1              0.76
2              0.72
3              0.80
4              0.65
```

1.4.6 Insights

University Rating , SOP and LOR strength and research are seems to be discrete random Variables , but also ordinal numeric data.

1.5 Data Preprocessing

1.5.1 Non - Graphical Analysis:

Checking Duplicate values in the dataset

[15]:
```
copy_Jamboree_data.duplicated().value_counts()
```

[15]:
```
False    500
Name: count, dtype: int64
```

Null Treatment:

- Check for the missing values and find the number of missing values in each column

[16]:
```
copy_Jamboree_data.isna().sum()
```

[16]:
```
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

Checking the unique values for columns

[17]:
```
for i in copy_Jamboree_data.columns:
    print(f'Unique values in {i} columns are : \n {copy_Jamboree_data[i].
 ↪unique()}\n\n')
```

```
Unique values in GRE Score columns are :
 [337 324 316 322 314 330 321 308 302 323 325 327 328 307 311 317 319 318
 303 312 334 336 340 298 295 310 300 338 331 320 299 304 313 332 326 329
 339 309 315 301 296 294 306 305 290 335 333 297 293]
```

```
Unique values in TOEFL Score columns are :
 [118 107 104 110 103 115 109 101 102 108 106 111 112 105 114 116 119 120
  98  93  99  97 117 113 100  95  96  94  92]


Unique values in University Rating columns are :
 [4 3 2 5 1]


Unique values in SOP columns are :
 [4.5 4.  3.  3.5 2.  5.  1.5 1.  2.5]


Unique values in LOR  columns are :
 [4.5 3.5 2.5 3.  4.  1.5 2.  5.  1. ]


Unique values in CGPA columns are :
 [9.65 8.87 8.   8.67 8.21 9.34 8.2  7.9  8.6  8.4  9.   9.1  8.3  8.7
 8.8  8.5  9.5  9.7  9.8  9.6  7.5  7.2  7.3  8.1  9.4  9.2  7.8  7.7
 9.3  8.85 7.4  7.6  6.8  8.92 9.02 8.64 9.22 9.16 9.64 9.76 9.45 9.04
 8.9  8.56 8.72 8.22 7.54 7.36 8.02 9.36 8.66 8.42 8.28 8.14 8.76 7.92
 7.66 8.03 7.88 7.84 8.96 9.24 8.88 8.46 8.12 8.25 8.47 9.05 8.78 9.18
 9.46 9.38 8.48 8.68 8.34 8.45 8.62 7.46 7.28 8.84 9.56 9.48 8.36 9.32
 8.71 9.35 8.65 9.28 8.77 8.16 9.08 9.12 9.15 9.44 9.92 9.11 8.26 9.43
 9.06 8.75 8.89 8.69 7.86 9.01 8.97 8.33 8.27 7.98 8.04 9.07 9.13 9.23
 8.32 8.98 8.94 9.53 8.52 8.43 8.54 9.91 9.87 7.65 7.89 9.14 9.66 9.78
 9.42 9.26 8.79 8.23 8.53 8.07 9.31 9.17 9.19 8.37 7.68 8.15 8.73 8.83
 8.57 9.68 8.09 8.17 7.64 8.01 7.95 8.49 7.87 7.97 8.18 8.55 8.74 8.13
 8.44 9.47 8.24 7.34 7.43 7.25 8.06 7.67 9.54 9.62 7.56 9.74 9.82 7.96
 7.45 7.94 8.35 7.42 8.95 9.86 7.23 7.79 9.25 9.67 8.86 7.57 7.21 9.27
 7.81 7.69]


Unique values in Research columns are :
 [1 0]


Unique values in Chance of Admit  columns are :
 [0.92 0.76 0.72 0.8  0.65 0.9  0.75 0.68 0.5  0.45 0.52 0.84 0.78 0.62
 0.61 0.54 0.66 0.63 0.64 0.7  0.94 0.95 0.97 0.44 0.46 0.74 0.91 0.88
 0.58 0.48 0.49 0.53 0.87 0.86 0.89 0.82 0.56 0.36 0.42 0.47 0.55 0.57
 0.96 0.93 0.38 0.34 0.79 0.71 0.69 0.59 0.85 0.77 0.81 0.83 0.67 0.73
 0.6  0.43 0.51 0.39 0.37]
```

[18]:
```
copy_Jamboree_data.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
```

Checking the total number of unique values for columns

[19]:
```
for i in copy_Jamboree_data.columns:
    print(f'Unique values in {i} columns are : {copy_Jamboree_data[i].
 ↪nunique()}\n')
```

```
Unique values in GRE Score columns are : 49

Unique values in TOEFL Score columns are : 29

Unique values in University Rating columns are : 5

Unique values in SOP columns are : 9

Unique values in LOR  columns are : 9

Unique values in CGPA columns are : 184

Unique values in Research columns are : 2

Unique values in Chance of Admit  columns are : 61
```

[20]:
```
numerical_columns = copy_Jamboree_data.columns
```

Outlier treatment Finding the outliers for every numerical variable in the dataset

[21]:
```
arr = {'5th percentile': 5, '25th percentile or Q1': 25, '50th percentile or␣
 ↪Q2': 50, '75th percentile or Q3': 75,
'95th percentile': 95}
```

[22]:
```
for key, value in arr.items():
    for column in numerical_columns:
        print(f'{column} : {key} -> {np.percentile(copy_Jamboree_data[column],␣
 ↪value):.2f}')

    print('_' * 100)
```

```
GRE Score : 5th percentile -> 298.00
TOEFL Score : 5th percentile -> 98.00
University Rating : 5th percentile -> 1.00
SOP : 5th percentile -> 1.50
LOR  : 5th percentile -> 2.00
CGPA : 5th percentile -> 7.64
Research : 5th percentile -> 0.00
Chance of Admit  : 5th percentile -> 0.47
----------------------------------------------------------------------------------
--------------------
GRE Score : 25th percentile or Q1 -> 308.00
TOEFL Score : 25th percentile or Q1 -> 103.00
University Rating : 25th percentile or Q1 -> 2.00
SOP : 25th percentile or Q1 -> 2.50
LOR  : 25th percentile or Q1 -> 3.00
CGPA : 25th percentile or Q1 -> 8.13
Research : 25th percentile or Q1 -> 0.00
Chance of Admit  : 25th percentile or Q1 -> 0.63
----------------------------------------------------------------------------------
--------------------
GRE Score : 50th percentile or Q2 -> 317.00
TOEFL Score : 50th percentile or Q2 -> 107.00
University Rating : 50th percentile or Q2 -> 3.00
SOP : 50th percentile or Q2 -> 3.50
LOR  : 50th percentile or Q2 -> 3.50
CGPA : 50th percentile or Q2 -> 8.56
Research : 50th percentile or Q2 -> 1.00
Chance of Admit  : 50th percentile or Q2 -> 0.72
----------------------------------------------------------------------------------
--------------------
GRE Score : 75th percentile or Q3 -> 325.00
TOEFL Score : 75th percentile or Q3 -> 112.00
University Rating : 75th percentile or Q3 -> 4.00
SOP : 75th percentile or Q3 -> 4.00
LOR  : 75th percentile or Q3 -> 4.00
CGPA : 75th percentile or Q3 -> 9.04
Research : 75th percentile or Q3 -> 1.00
Chance of Admit  : 75th percentile or Q3 -> 0.82
----------------------------------------------------------------------------------
--------------------
```

```
GRE Score : 95th percentile -> 335.00
TOEFL Score : 95th percentile -> 118.00
University Rating : 95th percentile -> 5.00
SOP : 95th percentile -> 5.00
LOR  : 95th percentile -> 5.00
CGPA : 95th percentile -> 9.60
Research : 95th percentile -> 1.00
Chance of Admit  : 95th percentile -> 0.94
--------------------------------------------------------------------------------
--------------------
```

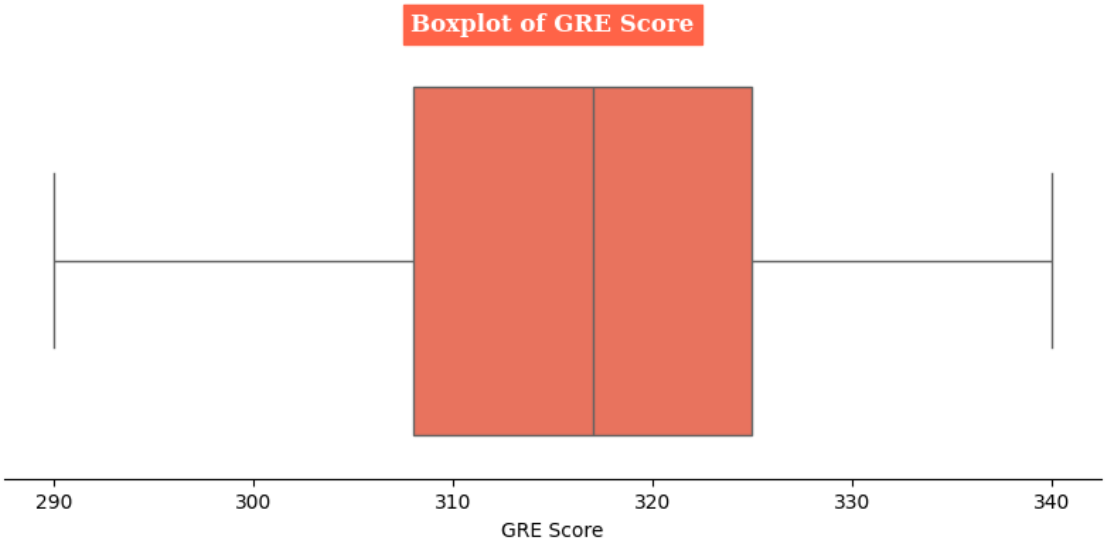
[23]:
```
# Detecting Outliers
for column in numerical_columns:
    Q1 = np.quantile(copy_Jamboree_data[column], 0.25)
    Q3 = np.quantile(copy_Jamboree_data[column], 0.75)
    IQR = Q3 - Q1
    LB = Q1 - 1.5 * IQR
    UB = Q3 + 1.5 * IQR
    outliers = copy_Jamboree_data[(copy_Jamboree_data[column] < LB) |␣
 ↪(copy_Jamboree_data[column] > UB)]
    print('Column :', column)
    print(f'Q1 : {Q1}')
    print(f'Q3 : {Q3}')
    print(f'IQR : {IQR}')
    print(f'LB : {LB}')
    print(f'UB : {UB}')
    print(f'Number of outliers : {outliers.shape[0]}')
    print('-----------------------------------')
```

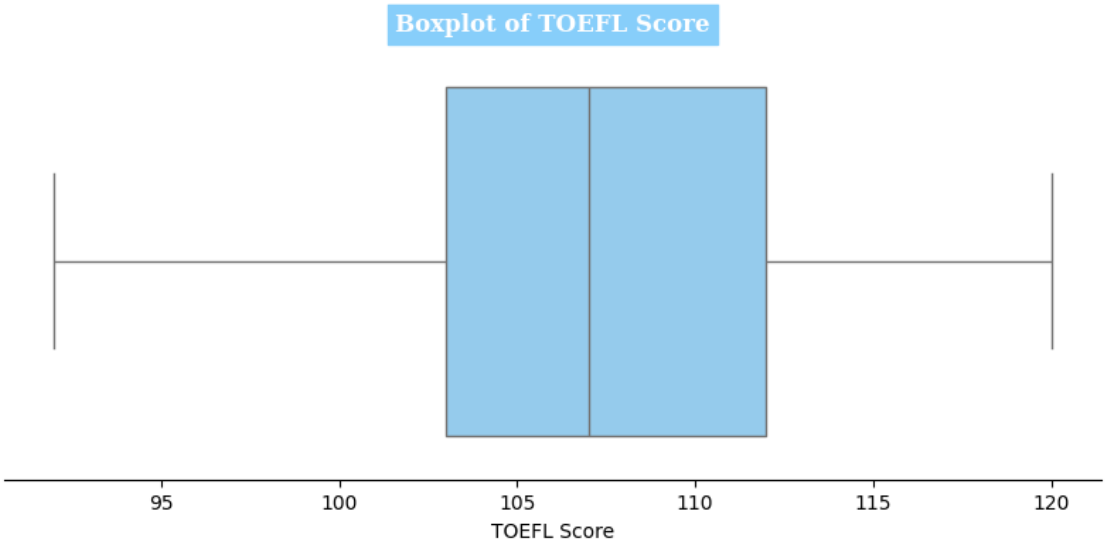
```
Column : GRE Score
Q1 : 308.0
Q3 : 325.0
IQR : 17.0
LB : 282.5
UB : 350.5
Number of outliers : 0
-----------------------------------
Column : TOEFL Score
Q1 : 103.0
Q3 : 112.0
IQR : 9.0
LB : 89.5
UB : 125.5
Number of outliers : 0
-----------------------------------
Column : University Rating
Q1 : 2.0
Q3 : 4.0
```

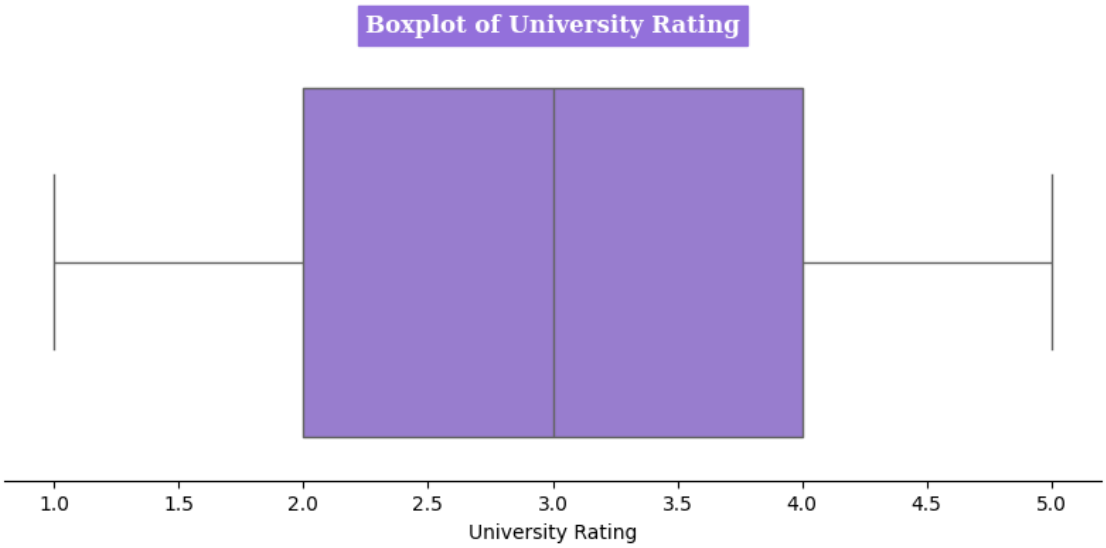
```
IQR : 2.0
LB : -1.0
UB : 7.0
Number of outliers : 0
---------------------------------
Column : SOP
Q1 : 2.5
Q3 : 4.0
IQR : 1.5
LB : 0.25
UB : 6.25
Number of outliers : 0
---------------------------------
Column : LOR
Q1 : 3.0
Q3 : 4.0
IQR : 1.0
LB : 1.5
UB : 5.5
Number of outliers : 1
---------------------------------
Column : CGPA
Q1 : 8.127500000000001
Q3 : 9.04
IQR : 0.9124999999999979
LB : 6.7587500000000045
UB : 10.408749999999996
Number of outliers : 0
---------------------------------
Column : Research
Q1 : 0.0
Q3 : 1.0
IQR : 1.0
LB : -1.5
UB : 2.5
Number of outliers : 0
---------------------------------
Column : Chance of Admit
Q1 : 0.63
Q3 : 0.82
IQR : 0.18999999999999995
LB : 0.3450000000000001
UB : 1.105
Number of outliers : 2
---------------------------------
```

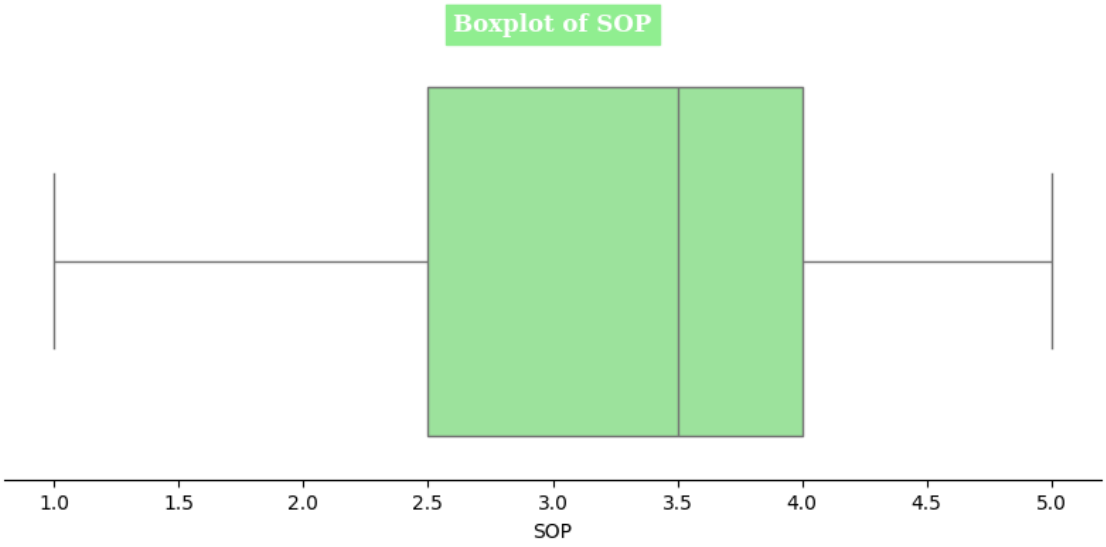
[24]:
```
colors = ["#FF6347", "#87CEFA", "#9370DB", "#90EE90", "#FFD700", "#FFA07A",␣
↪"#00FFFF",'salmon','tomato','red','dimgrey','tomato','dimgray','orangered','k','salmon']
```

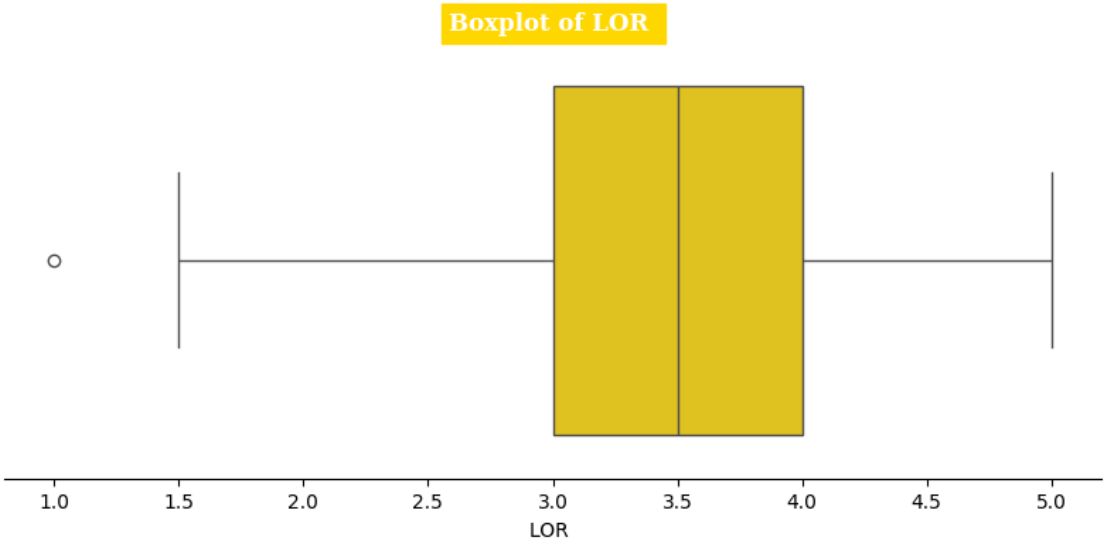
[25]:
```
# Create box plots for each continuous variable
# Loop through the variables and create the box plots
for i, col in enumerate(numerical_columns):
    plt.figure(figsize=(10, 4))
    sns.boxplot(x=copy_Jamboree_data[col], color=colors[i])
    sns.despine(left=True)
    plt.yticks([])
    plt.title(f'Boxplot of {col}', fontfamily='serif', fontweight='bold',␣
↪fontsize=12, backgroundcolor=colors[i], color='w')

    plt.show()
```

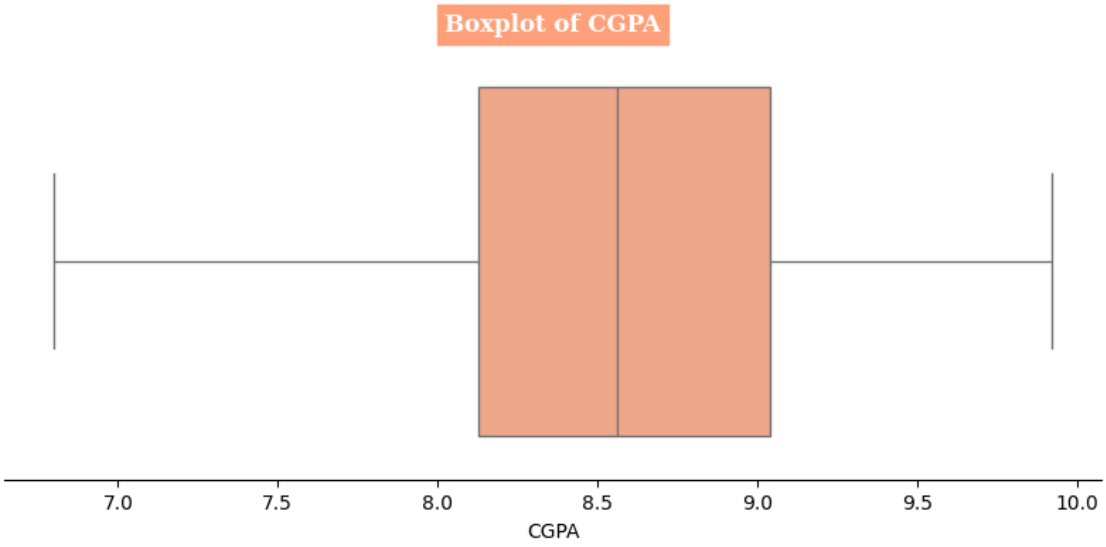


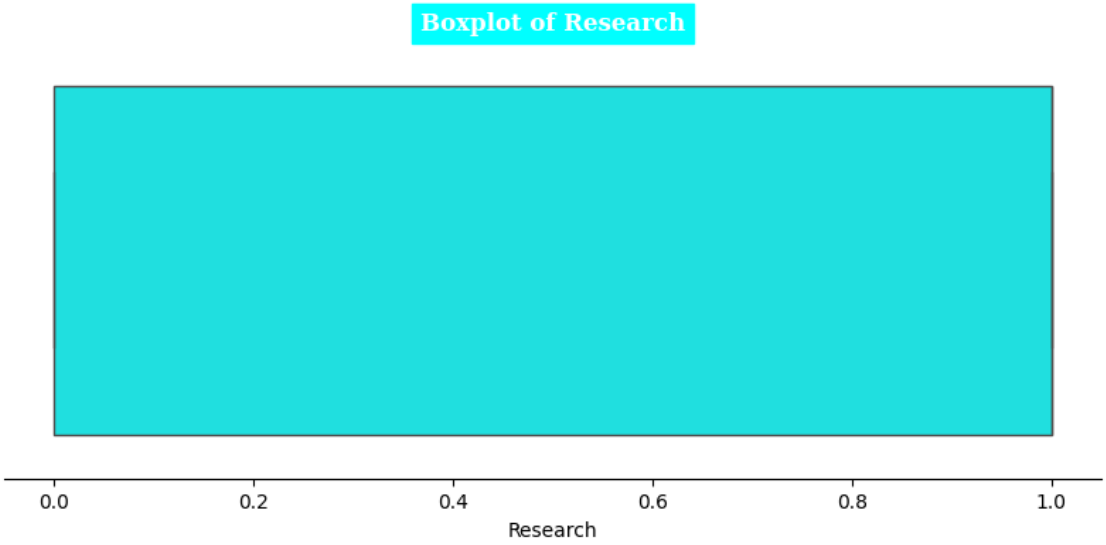

Boxplot of TOEFL Score
95
100
105
110
115
120
TOEFL Score

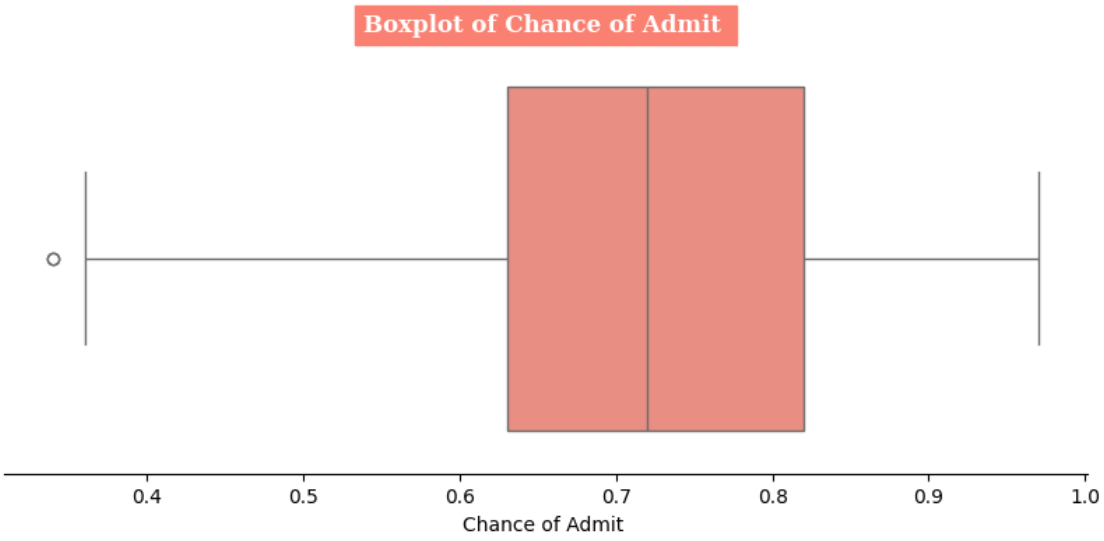

Boxplot of University Rating
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
University Rating


Boxplot of SOP
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
SOP


Boxplot of LOR
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
LOR


Boxplot of CGPA
7.0
7.5
8.0
8.5
9.0
9.5
10.0
CGPA


Boxplot of Research
0.0
0.2
0.4
0.6
0.8
1.0
Research



1.5.2 Insights

The dataset does not contain any duplicates.

No null values are present in dataset.

Total number of unique values in GRE score, TOEFL score, University Rating, SOP, LOR, CGPA, Research, Chance of Admit are 49, 29, 5, 9, 9, 184, 2, 61 respectively.

1.6 Graphical Analysis:

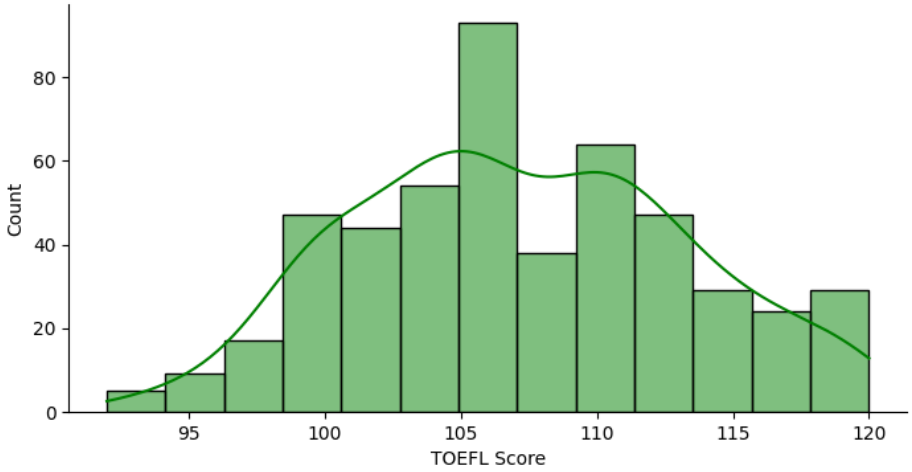
Univariate Analysis

[26]:
```
cp = 'Greens'
```

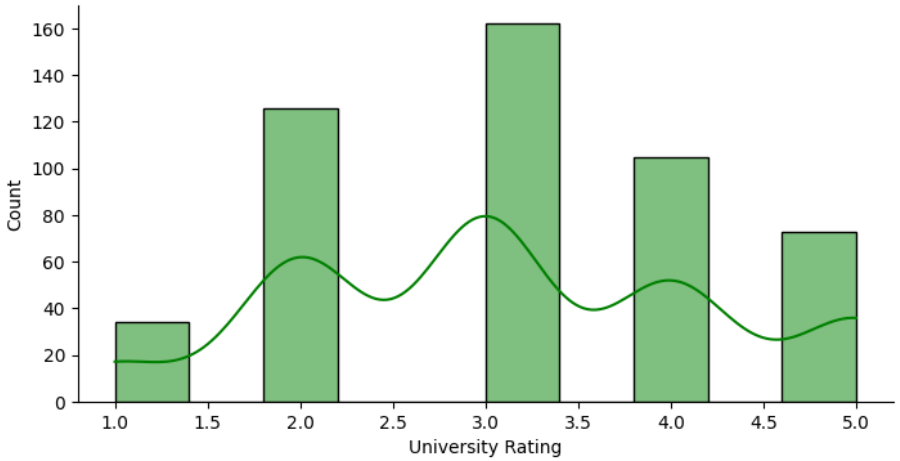
[27]:
```
for _ in copy_Jamboree_data.columns:
    plt.style.use('default')
    plt.figure(figsize = (18,4))
    plt.subplot(122)
    sns.histplot(copy_Jamboree_data[_],kde=True,color='g')
    plt.suptitle(f'Plot of␣
 ↪{_}',fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolor='g',color='w')
    sns.despine()
    plt.show()
```

Plot of GRE Score

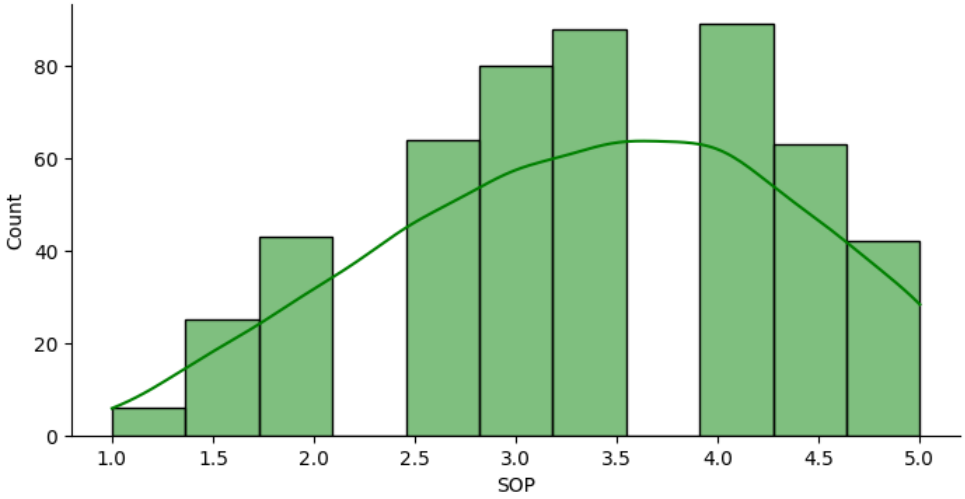
Plot of TOEFL Score



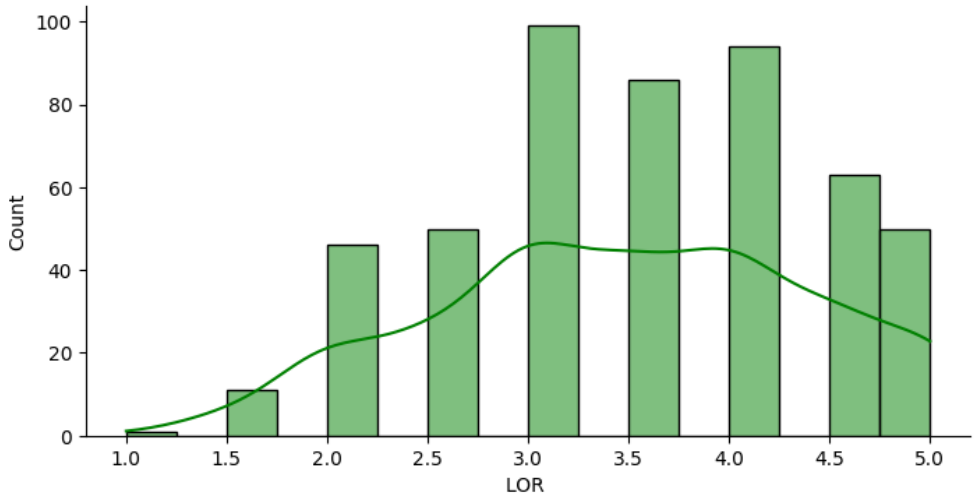
Plot of University Rating



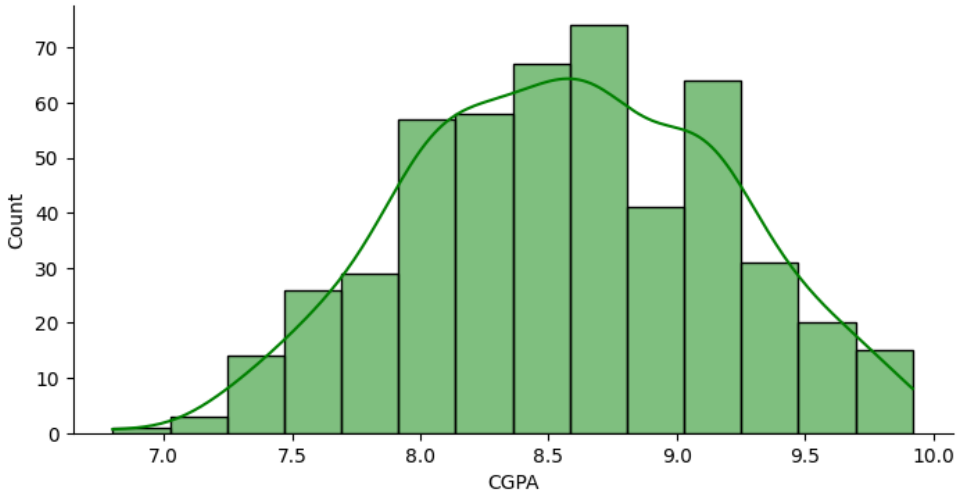
Plot of SOP



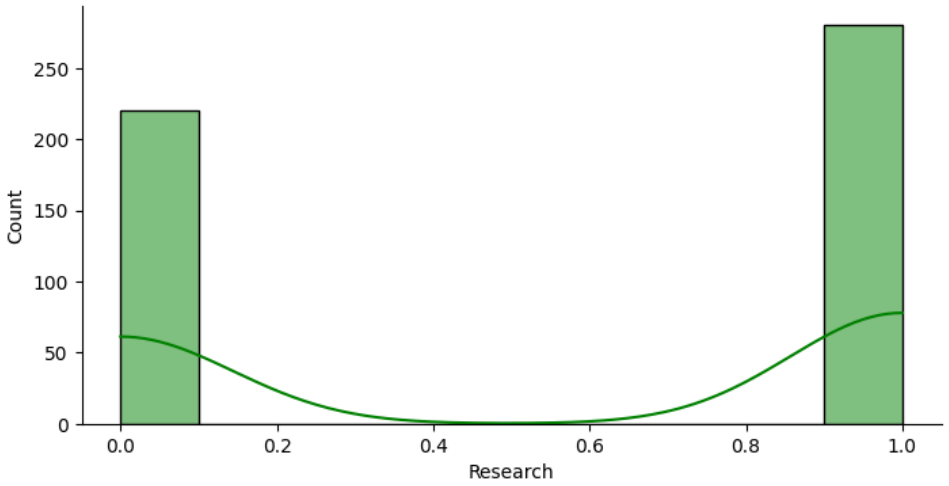
Plot of LOR



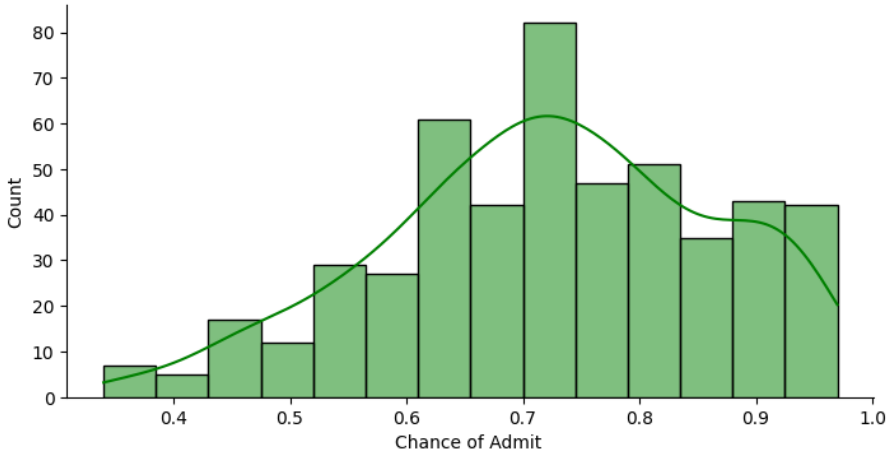
Plot of CGPA



Plot of Research



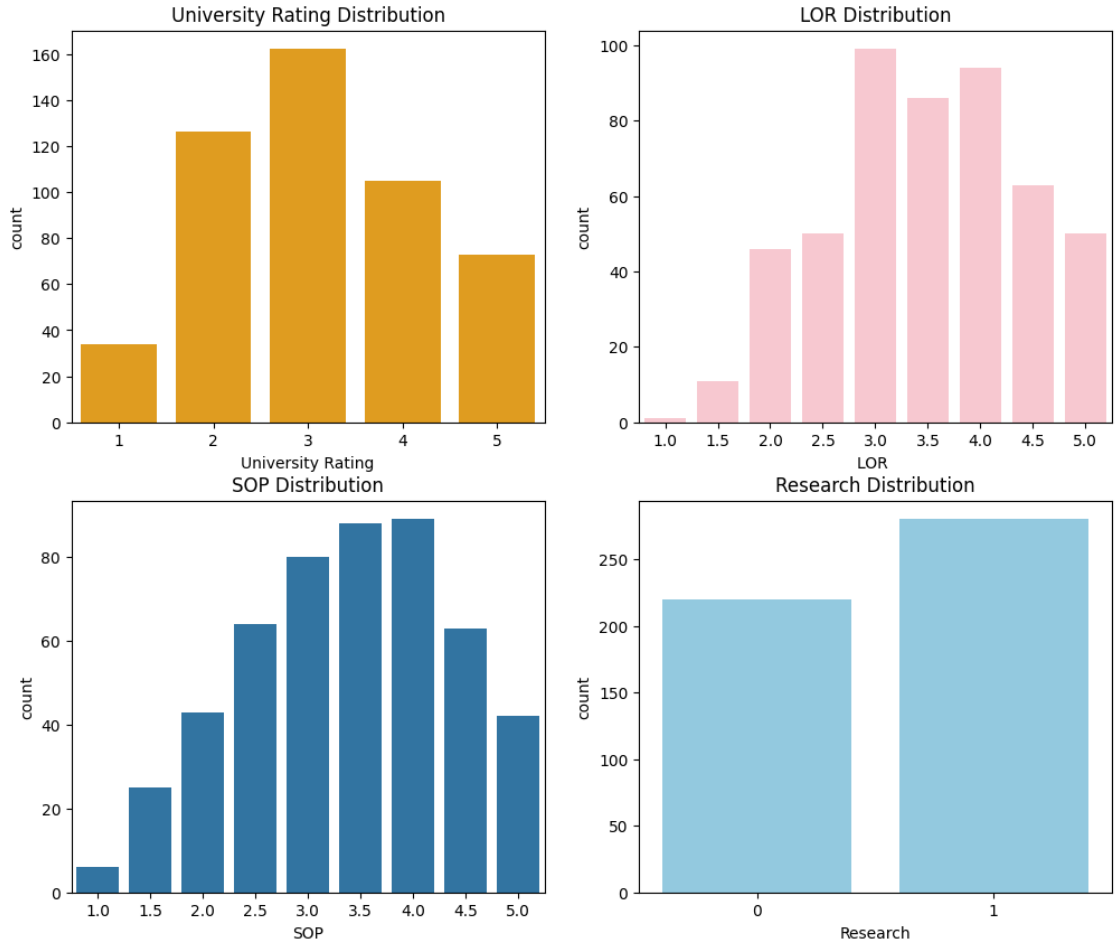
Plot of Chance of Admit



Barplots/countplots of all the categorical variables: Distribution of all other categorical features :

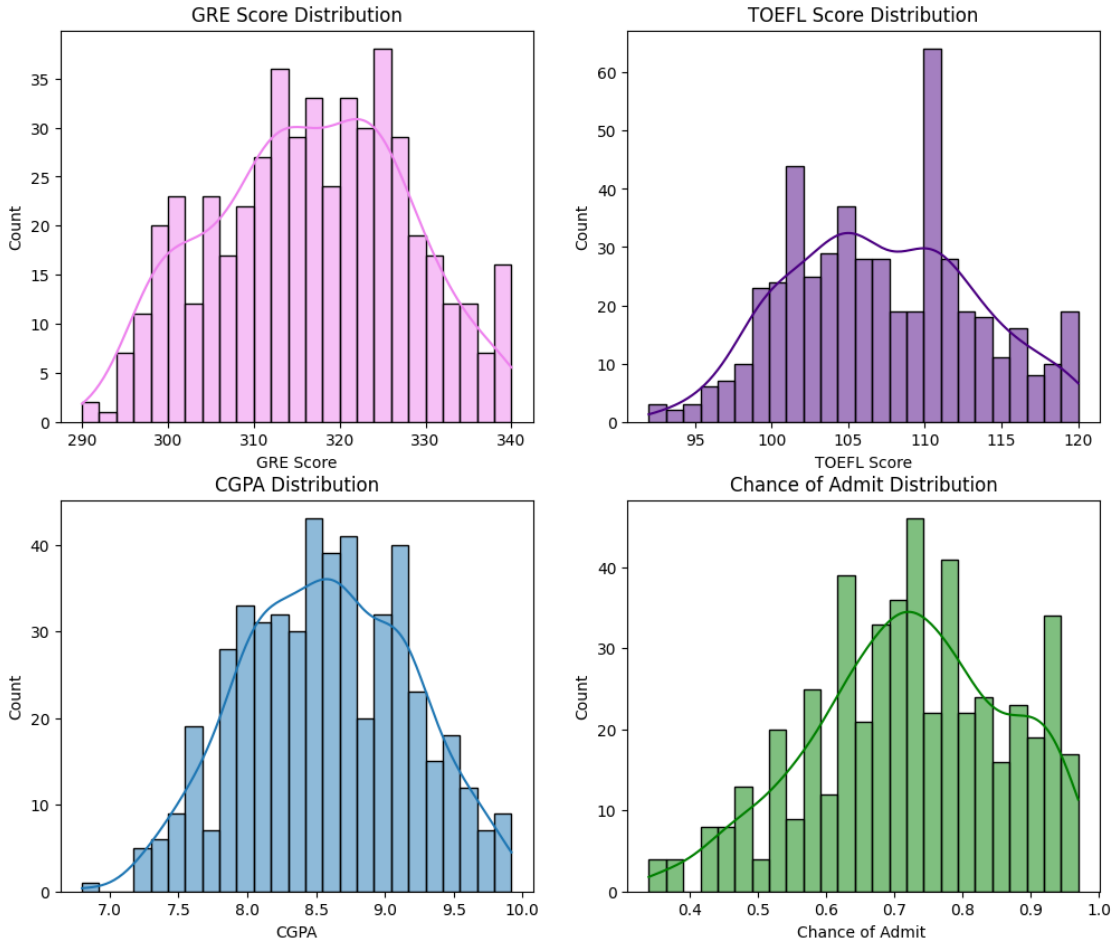
[28]:
```
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(12, 10))
sns.countplot(data=copy_Jamboree_data, x='University Rating', ax=ax[0, 0],␣
 ↪color ='orange').set_title('University Rating Distribution')
sns.countplot(data=copy_Jamboree_data, x='LOR ', ax=ax[0, 1], color ='pink').
 ↪set_title('LOR Distribution')
sns.countplot(data=copy_Jamboree_data, x='SOP', ax=ax[1, 0]).set_title('SOP␣
 ↪Distribution')
```

```
sns.countplot(data=copy_Jamboree_data, x='Research', ax=ax[1, 1], color␣
↪='skyblue').set_title('Research Distribution')
plt.show()
```



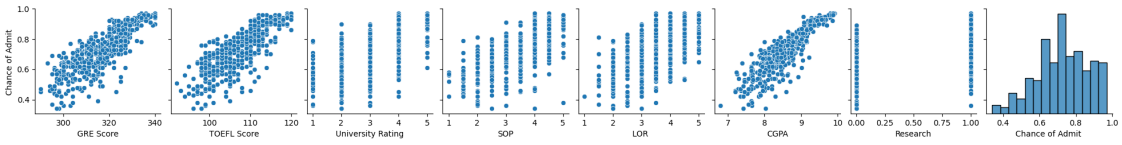
Distribution plots of all the continuous variable(s):

[29]:
```
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(12, 10))
sns.histplot(copy_Jamboree_data['GRE Score'],kde = True ,ax=ax[0, 0],bins = 25␣
↪,color='violet').set_title('GRE Score Distribution')
sns.histplot(copy_Jamboree_data['TOEFL Score'],kde = True , ax=ax[0, 1],bins =␣
↪25 , color='indigo').set_title('TOEFL Score Distribution')
sns.histplot(copy_Jamboree_data['CGPA'],kde = True ,bins = 25, ax=ax[1, 0]).
↪set_title('CGPA Distribution')
sns.histplot(copy_Jamboree_data['Chance of Admit '],kde = True ,bins = 25 ,␣
↪ax=ax[1, 1], color='green').set_title('Chance of Admit Distribution')
plt.show()
```



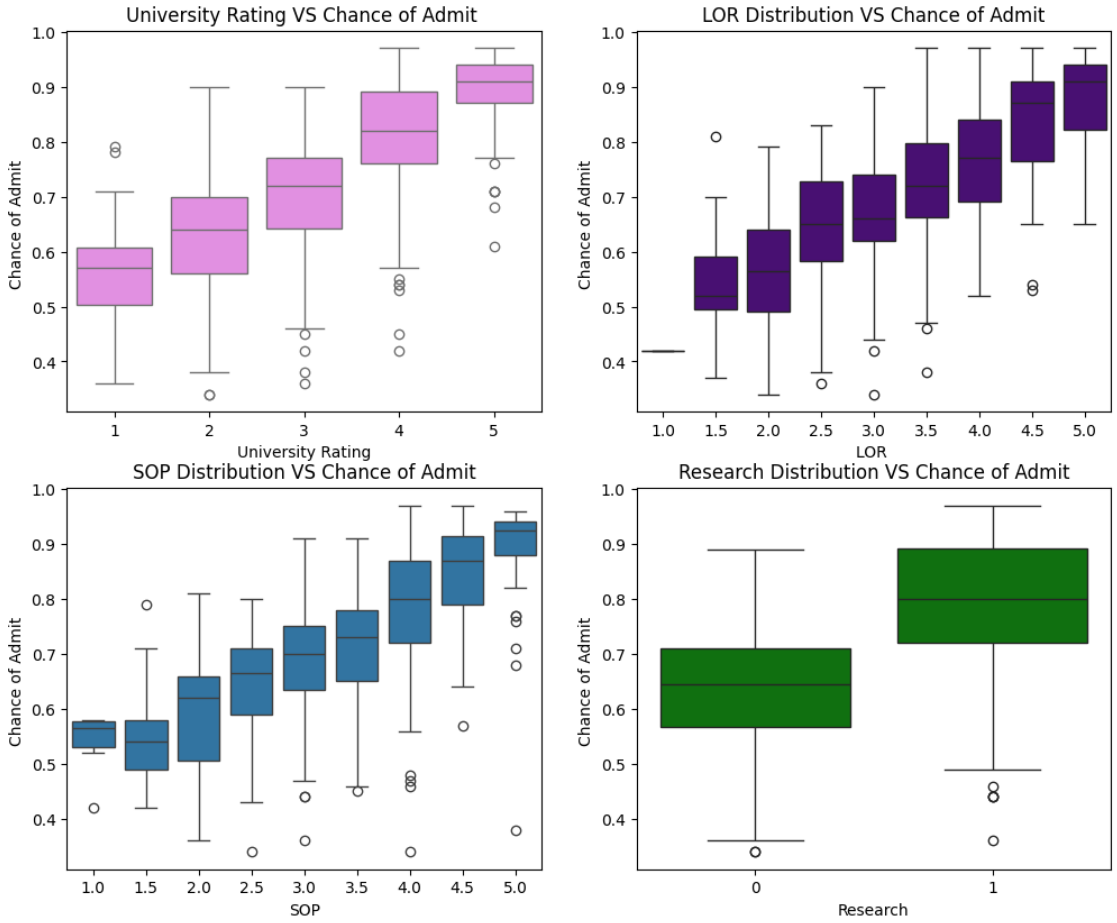
Bivariate Analysis

[30]:
```
sns.pairplot(data=copy_Jamboree_data, y_vars='Chance of Admit ')
plt.show()
```

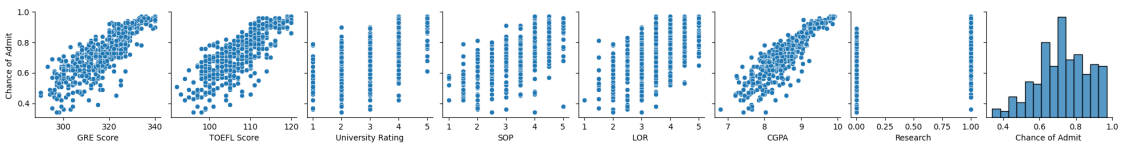


[31]:
```
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(12, 9.7))
sns.boxplot(data=copy_Jamboree_data, x='University Rating',y = 'Chance of Admit␣
↪', ax=ax[0, 0], color ='violet').set_title('University Rating VS Chance of␣
↪Admit ')
sns.boxplot(data=copy_Jamboree_data, x='LOR ',y = 'Chance of Admit ', ax=ax[0,␣
↪1], color ='indigo').set_title('LOR Distribution VS Chance of Admit')
```

```
sns.boxplot(data=copy_Jamboree_data, x='SOP',y = 'Chance of Admit ', ax=ax[1,␣
↪0]).set_title('SOP Distribution VS Chance of Admit')
sns.boxplot(data=copy_Jamboree_data, x='Research',y = 'Chance of Admit ',␣
↪ax=ax[1, 1], color ='green').set_title('Research Distribution VS Chance of␣
↪Admit')
plt.show()
```

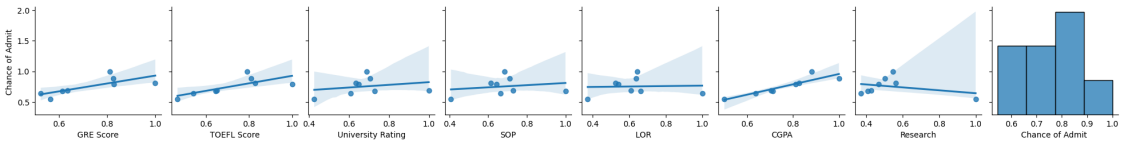


[32]:
```
sns.pairplot(data=copy_Jamboree_data, y_vars='Chance of Admit ')
plt.show()
```



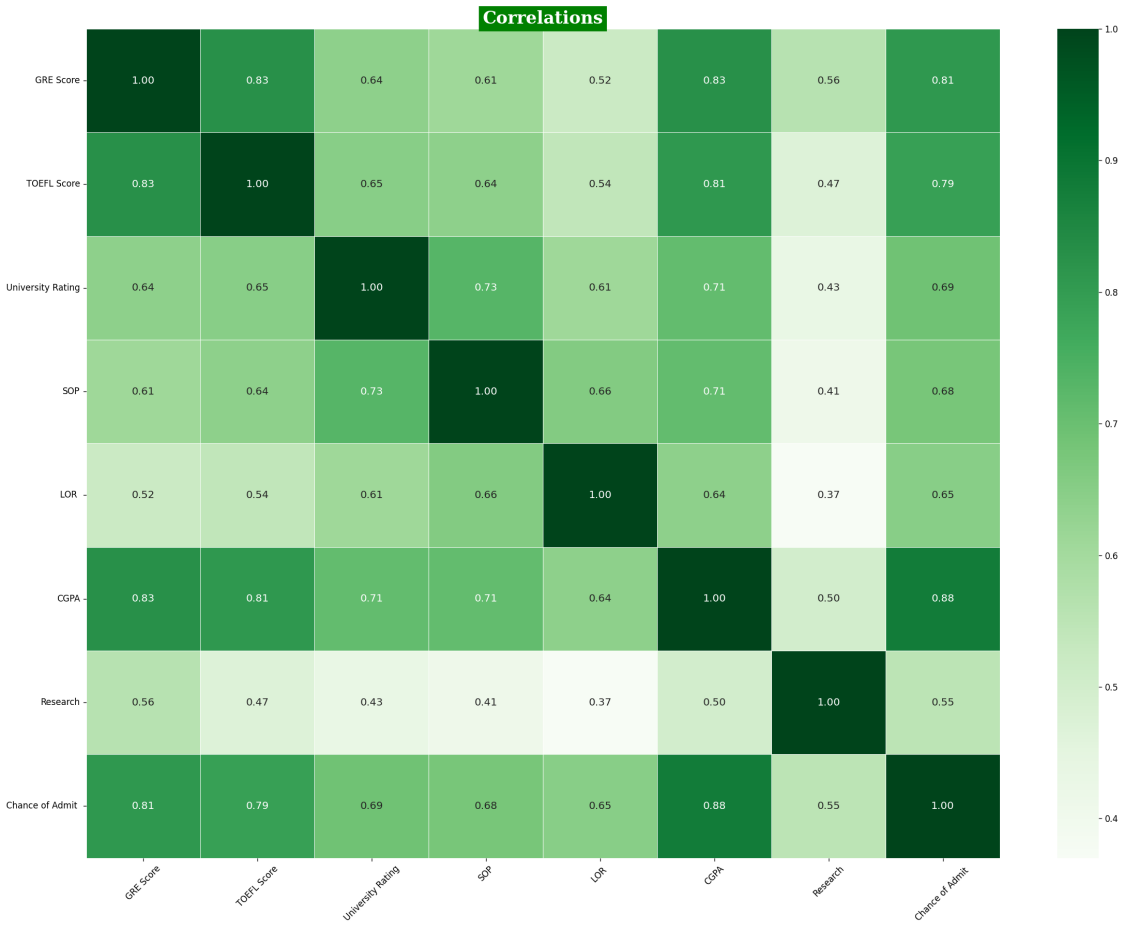
[33]:
```
sns.pairplot(copy_Jamboree_data.corr(),y_vars='Chance of Admit ',kind= 'reg')
```

[33]: <seaborn.axisgrid.PairGrid at 0x7c007e4d7af0>



[34]:
```
corr_matrix = copy_Jamboree_data.corr().round(2)
```

[35]:
```
# Calculate and round the correlation matrix

# Create the heatmap
plt.figure(figsize=(20, 15), dpi=120)  # Increase the figure size and DPI
heatmap = sns.heatmap(corr_matrix, annot=True, cmap='Greens', linewidths=0.
 ↪5,fmt=".2f", annot_kws={"size": 12})
plt.title('Correlations', fontsize=20, fontfamily='serif', fontweight='bold',␣
 ↪backgroundcolor='g', color='w')
plt.yticks(rotation=0)
plt.xticks(rotation=45)  # Rotate x-ticks for better readability
plt.tight_layout()  # Adjust layout to ensure no clipping

plt.show()
```



1.6.1 Insights

From the above all observation are: - By the correlation heatmap , We can observe GRE_score ,TOEFL_score and CGPA have very high correlation with Change of admission.

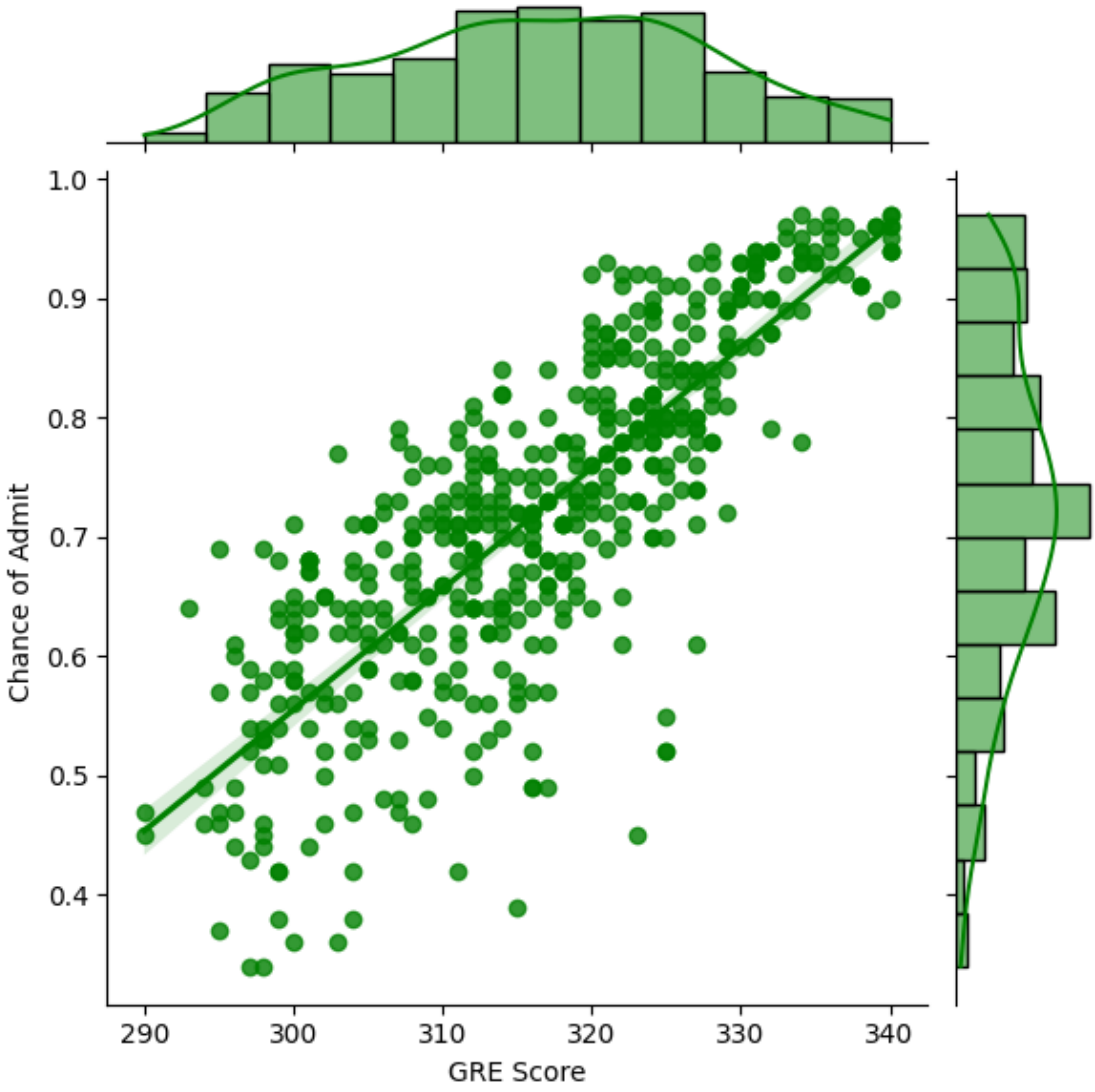
- University rating, SOP ,LOR and Research are comparatively slightly less correlated than other features.
- Independent Variables (Input datas): GRE Score, TOEFL Score, University Rating, SOP, LOR, CGPA,Research
- Target/Dependent Variable : Chance of Admit (the value we want to predict)

By observing the joint plots, we can gain insights into the relationship between each independent variable and the target variable. Here are the observations that we can make from the joint plots: - GRE , TOEFl score, CGPA VS CHANCE OF ADMISSION: It shows the positive relatioinship between the them, relationship is linear and there are some outlier also present.

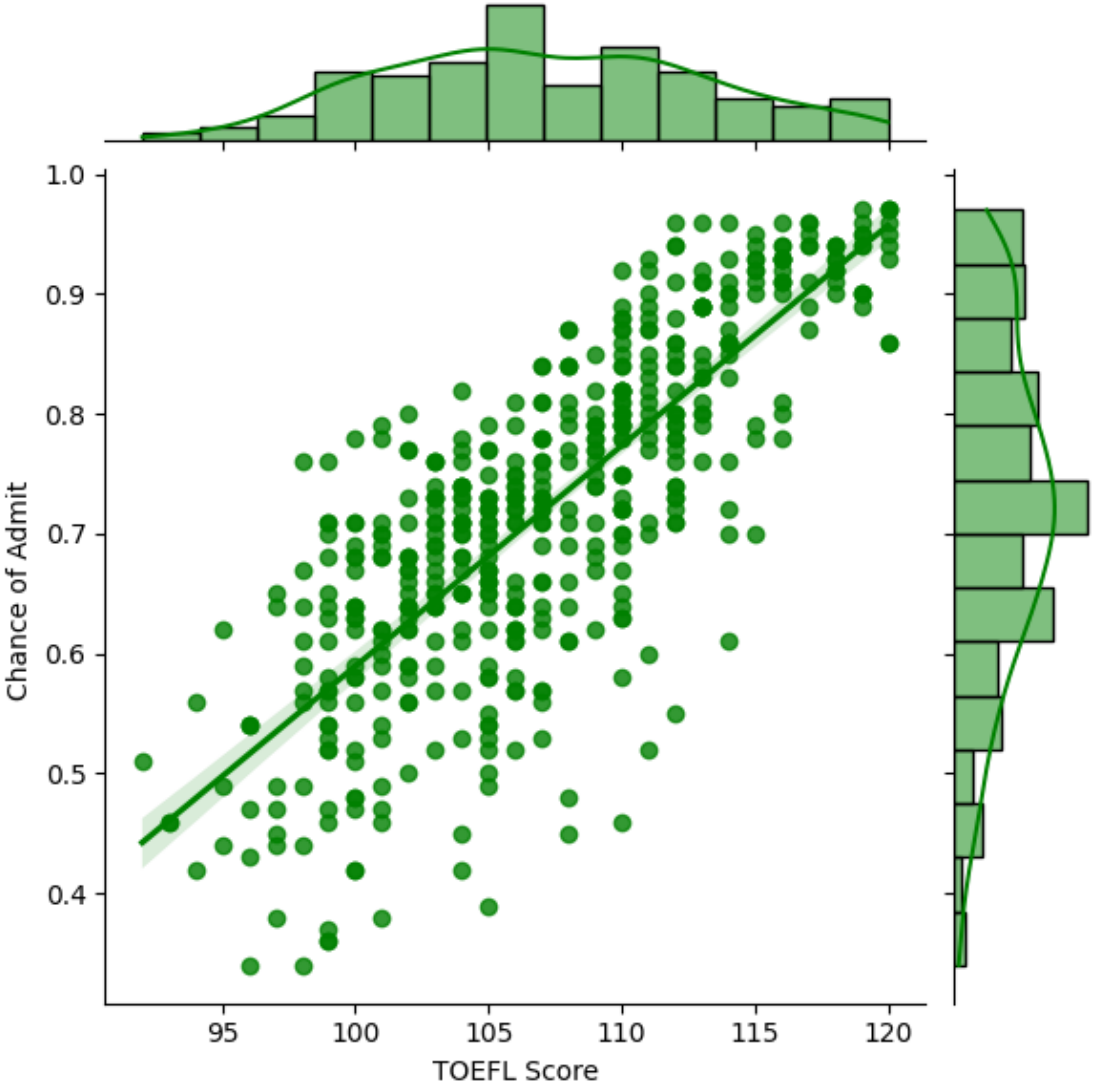
1.6.2 Checking for Linearity : How features are correlated with Target variable - chance of admit :

[36]:
```
for col in copy_Jamboree_data.columns[:-1]:
    print(col)
    sns.
 ↪jointplot(data=copy_Jamboree_data,x=copy_Jamboree_data[col],y=copy_Jamboree_data["Chance␣
 ↪of Admit "],kind="reg",color='g')
    plt.show()
```

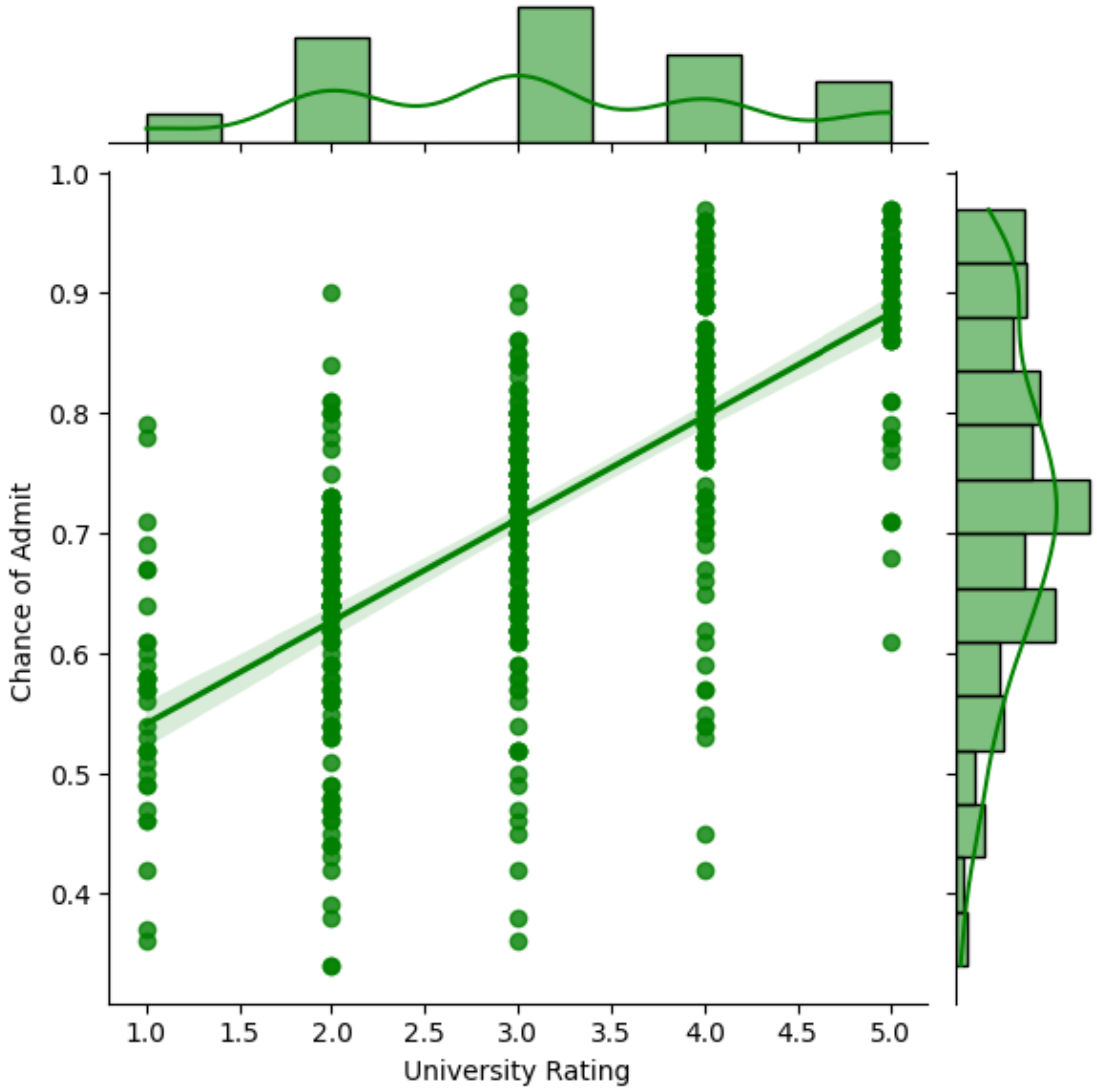
GRE Score



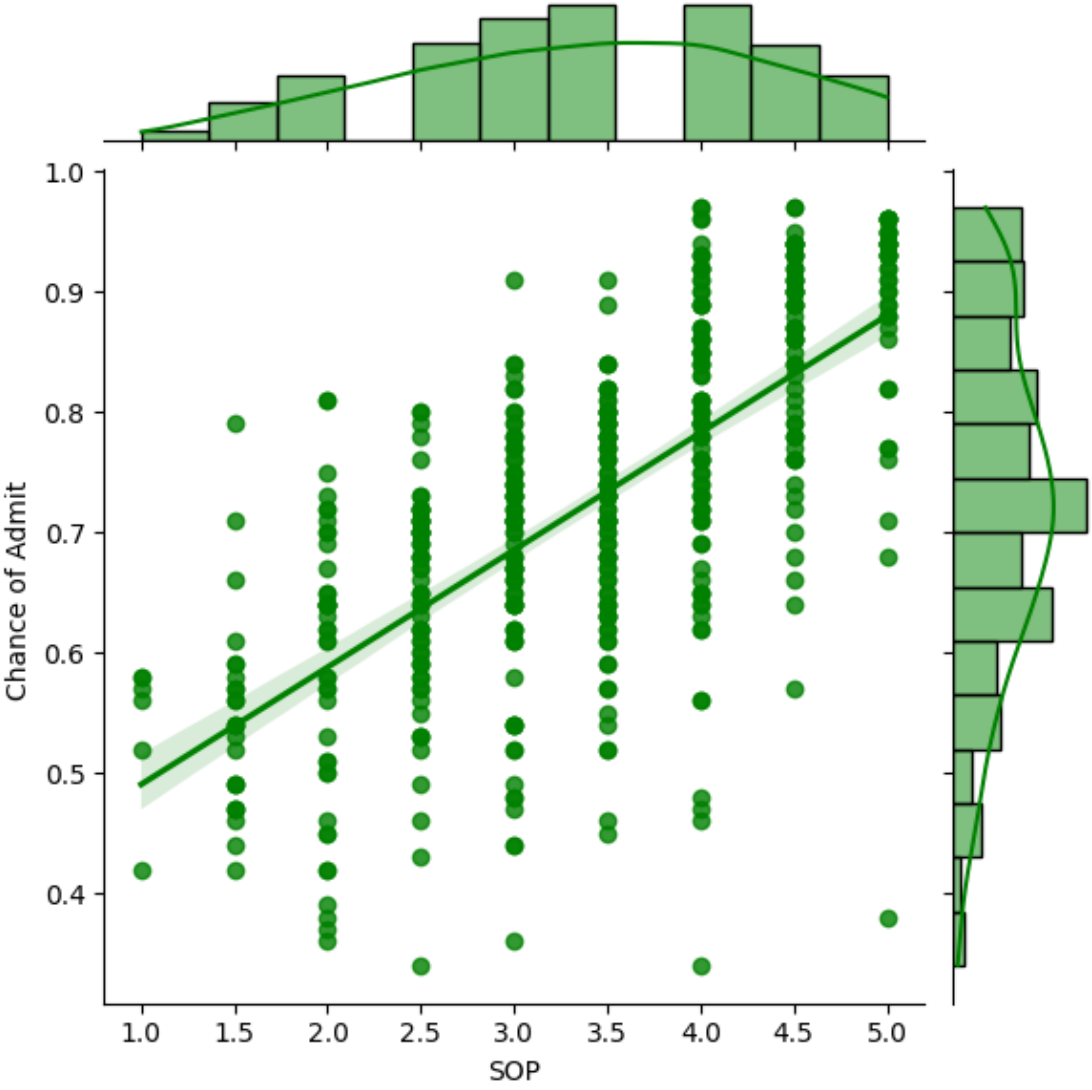
TOEFL Score



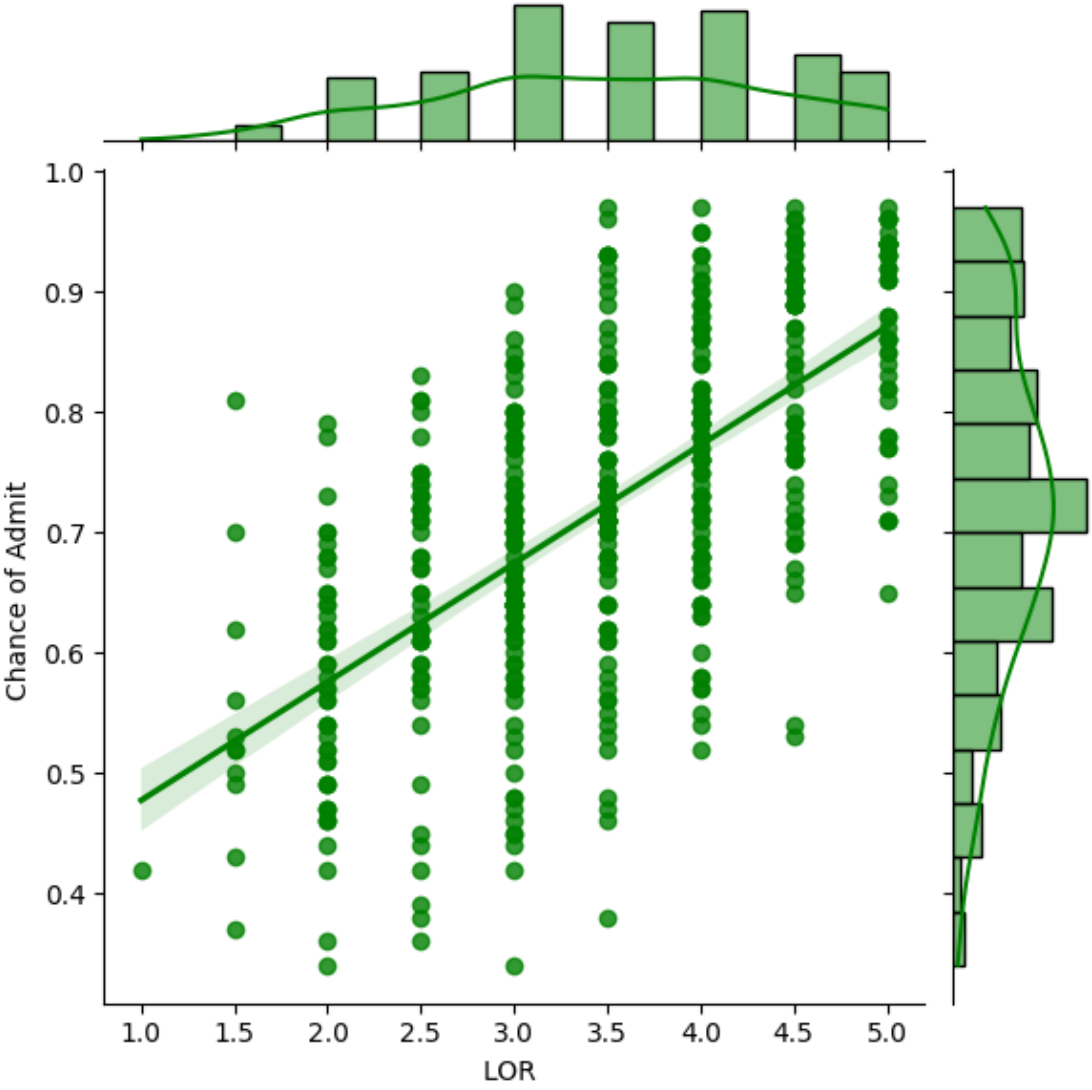
University Rating


Chance of Admit
1.0
0.9
0.8
0.7
0.6
0.5
0.4
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
University Rating

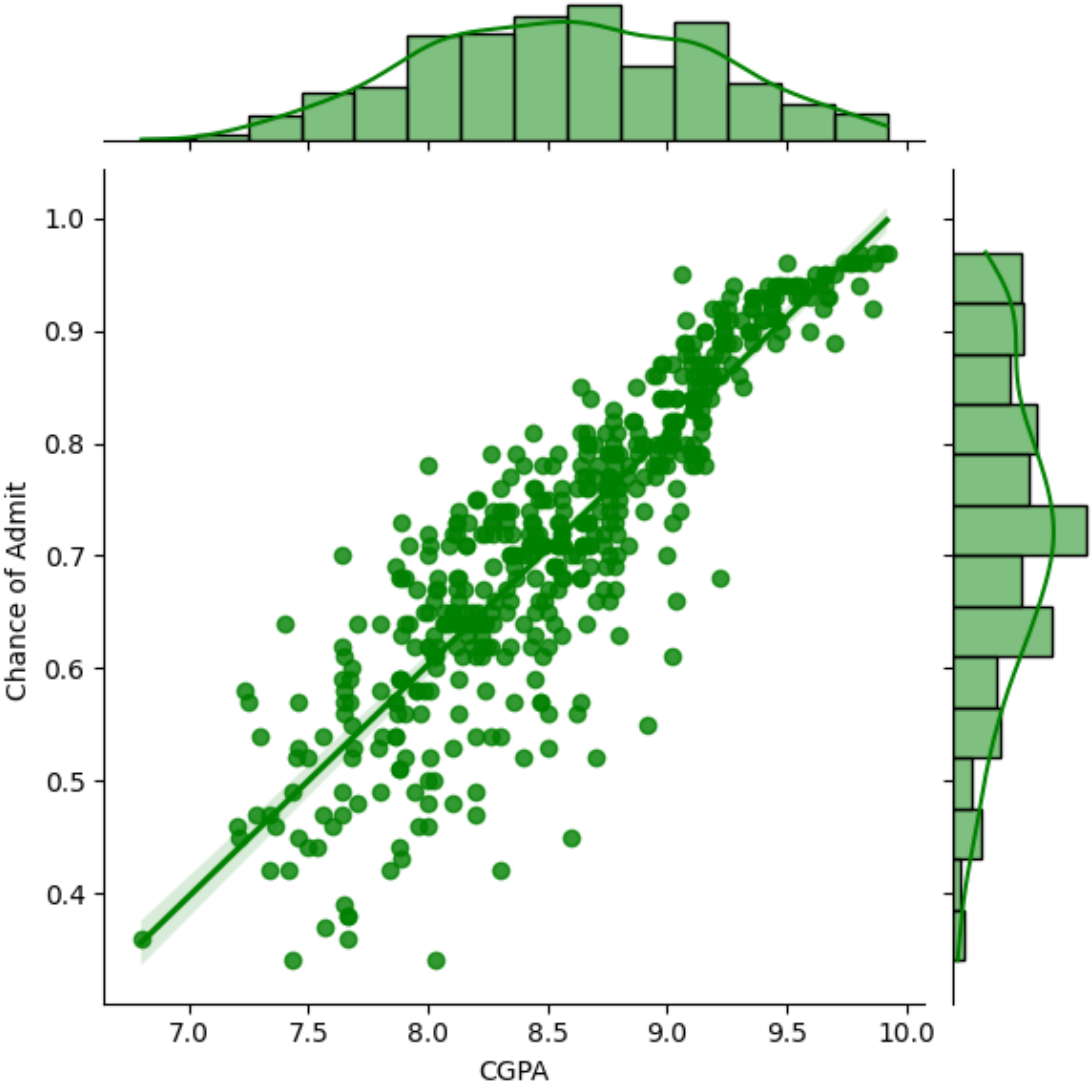
SOP



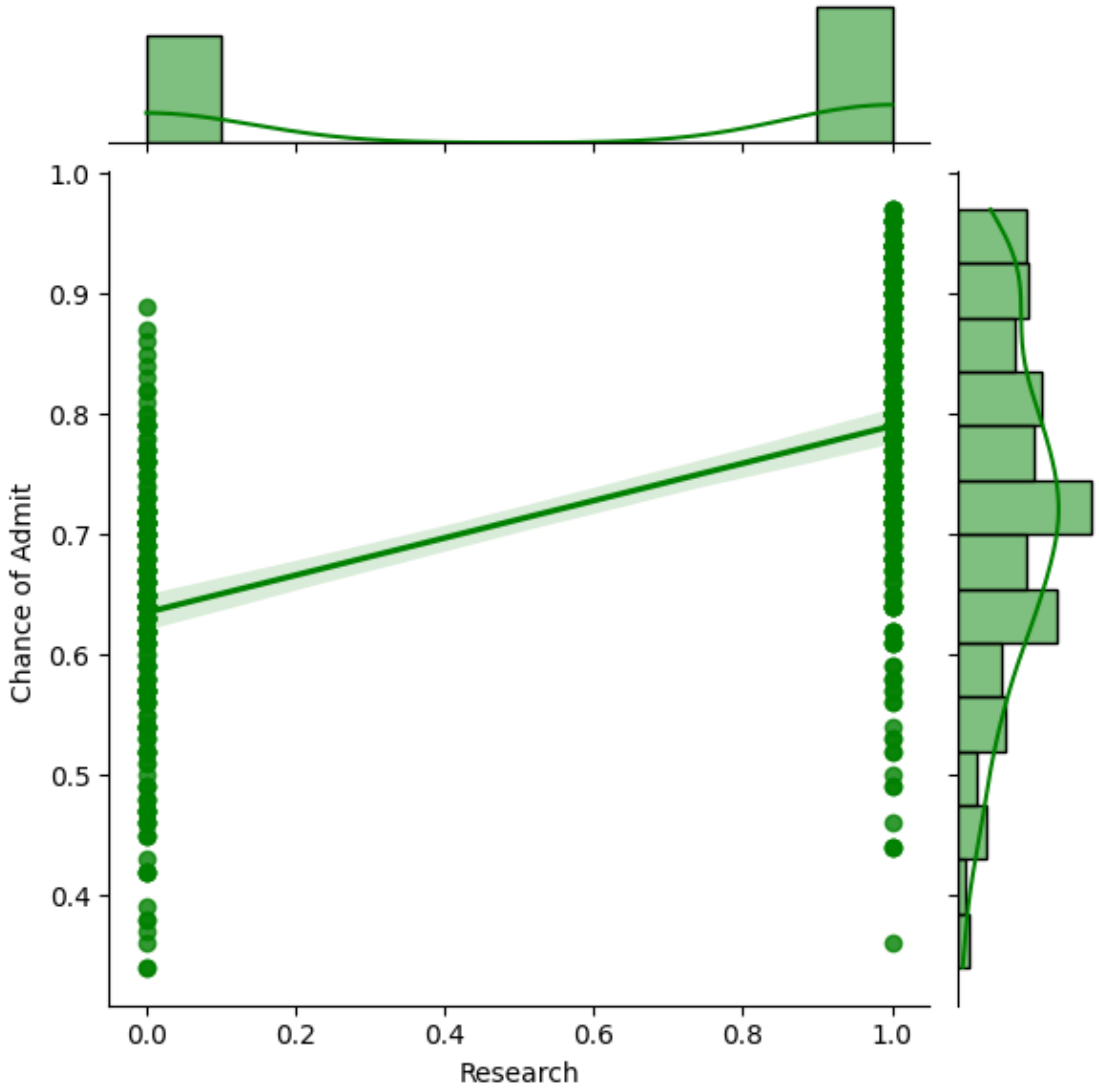
LOR



CGPA



Research



1.6.3 Insights

With higher GRE score , there is high probability of getting an admition. Students having high toefl score , has higher probability of getting admition .

1.7 Data Modelling

Standardization

[37]:
```
scaler = StandardScaler()
scaled_data = pd.DataFrame(scaler.fit_transform(copy_Jamboree_data), columns =␣
↪copy_Jamboree_data.columns)
```

[38]:
```
scaled_data
```

[38]:
```
     GRE Score  TOEFL Score  University Rating       SOP      LOR       CGPA  \
0     1.819238     1.778865           0.775582  1.137360  1.098944  1.776806
1     0.667148    -0.031601           0.775582  0.632315  1.098944  0.485859
2    -0.041830    -0.525364          -0.099793 -0.377773  0.017306 -0.954043
3     0.489904     0.462163          -0.099793  0.127271 -1.064332  0.154847
4    -0.219074    -0.689952          -0.975168 -1.387862 -0.523513 -0.606480
..         …            …                  …         …        …         …
495   1.376126     0.132987           1.650957  1.137360  0.558125  0.734118
496   1.819238     1.614278           1.650957  1.642404  1.639763  2.140919
497   1.198882     2.108041           1.650957  1.137360  1.639763  1.627851
498  -0.396319    -0.689952           0.775582  0.632315  1.639763 -0.242367
499   0.933015     0.955926           0.775582  1.137360  1.098944  0.767220

     Research  Chance of Admit
0    0.886405          1.406107
1    0.886405          0.271349
2    0.886405         -0.012340
3    0.886405          0.555039
4   -1.128152         -0.508797
..        …                 …
495  0.886405          1.051495
496  0.886405          1.689797
497  0.886405          1.477030
498 -1.128152          0.058582
499 -1.128152          0.838728

[500 rows x 8 columns]
```

Train-Test data split

[39]:
```
x = scaled_data.iloc[:,:-1]
y = scaled_data.iloc[:,-1]
print(x.shape , y.shape)
```

```
(500, 7) (500,)
```

[40]:
```
# Split the data into training and test data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.
 ↪2,random_state=42)
print(f'Shape of x_train: {x_train.shape}')
print(f'Shape of x_test: {x_test.shape}')
print(f'Shape of y_train: {y_train.shape}')
print(f'Shape of y_test: {y_test.shape}')
```

```
Shape of x_train: (400, 7)
Shape of x_test: (100, 7)
Shape of y_train: (400,)
Shape of y_test: (100,)
```

1.8 Linear Regression

[41]:
```
lr_model = LinearRegression()
lr_model.fit(x_train,y_train)
```

[41]:
```
LinearRegression()
```

[42]:
```
# Predicting values for the training and test data
y_pred_train = lr_model.predict(x_train)
y_pred_test = lr_model.predict(x_test)
```

[43]:
```
# Check the r2 score on trian data:
print('R2-Score On Train data:', r2_score(y_train, lr_model.predict(x_train)))
# Check the r2 score on test data:
print('R2-Score on Test data:', r2_score(y_test, lr_model.predict(x_test)))
```

```
R2-Score On Train data: 0.8210671369321554
R2-Score on Test data: 0.8188432567829628
```

All the feature's coefficients and Intercept

[44]:
```
lr_model_weights = pd.DataFrame(lr_model.coef_.
 ↪reshape(1,-1),columns=copy_Jamboree_data.columns[:-1])
lr_model_weights["Intercept"] = lr_model.intercept_
lr_model_weights
```

[44]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Intercept
0	0.194823	0.129095	0.020812	0.012735	0.113028	0.482199	0.084586	0.007736

[45]:
```
def model_evaluation(y_actual, y_forecast, model):
    n = len(y_actual)

    # Determine the number of predictors (p)
    if len(model.coef_.shape) == 1:
        p = len(model.coef_)
    else:
        p = len(model.coef_[0])

    # Calculate evaluation metrics
    MSE = np.round(mean_squared_error(y_true= y_actual, y_pred= y_forecast,␣
 ↪squared=True), 2)
    MAE = np.round(mean_absolute_error(y_true=y_actual, y_pred=y_forecast), 2)
    RMSE = np.round(mean_squared_error(y_true=y_actual, y_pred=y_forecast,␣
 ↪squared=False), 2)
    r2 = np.round(r2_score(y_true=y_actual, y_pred=y_forecast), 2)
```

```
    adj_r2 = np.round(1 - ((1-r2)*(n-1)/(n-p-1)), 2)

    # Return or print the results
    print(f"MSE: {MSE}\nMAE: {MAE}\nRMSE: {RMSE}\nR2 Score: {r2}\nAdjusted R2: {adj_r2}")
```

[46]:
```
model_evaluation(y_train.values, y_pred_train, lr_model)
```

```
MSE: 0.18
MAE: 0.3
RMSE: 0.42
R2 Score: 0.82
Adjusted R2: 0.82
```

[47]:
```
model_evaluation(y_test.values, y_pred_test, lr_model)
```

```
MSE: 0.19
MAE: 0.3
RMSE: 0.43
R2 Score: 0.82
Adjusted R2: 0.81
```

Linear Regression using OLS

[48]:
```
new_x_train = sm.add_constant(x_train)
model = sm.OLS(y_train, new_x_train)
results = model.fit()
# statstical summary of the model
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       Chance of Admit   R-squared:                       0.821
Model:                           OLS   Adj. R-squared:                  0.818
Method:                Least Squares   F-statistic:                     257.0
Date:               Sun, 04 Aug 2024   Prob (F-statistic):          3.41e-142
Time:                       15:18:08   Log-Likelihood:                -221.69
No. Observations:                400   AIC:                             459.4
Df Residuals:                    392   BIC:                             491.3
Df Model:                          7
Covariance Type:           nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0077      0.021      0.363      0.717      -0.034
```

```
0.050
GRE Score              0.1948      0.046      4.196      0.000       0.104
0.286
TOEFL Score            0.1291      0.041      3.174      0.002       0.049
0.209
University Rating      0.0208      0.034      0.611      0.541      -0.046
0.088
SOP                    0.0127      0.036      0.357      0.721      -0.057
0.083
LOR                    0.1130      0.030      3.761      0.000       0.054
0.172
CGPA                   0.4822      0.046     10.444      0.000       0.391
0.573
Research               0.0846      0.026      3.231      0.001       0.033
0.136
==============================================================================
Omnibus:                       86.232   Durbin-Watson:                   2.050
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              190.099
Skew:                          -1.107   Prob(JB):                     5.25e-42
Kurtosis:                       5.551   Cond. No.                         5.72
==============================================================================

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

Conidering the very low p_valued Features and highly weighted coef features as the major contributers of Model Prediction, CGPA,GRE,TOEFL,LOR are the features contributing to model building…

1.9 Testing Assumptions of Linear Regression Model

1. **No multicolinearity:** > Multicollinearity check by VIF(Variance Inflation Factor) score. > Variables are dropped one-by-one till none has a VIF>5.
2. **Mean of Residuals** should be close to zero.
3. Linear relationship between independent & dependent variables.
    - This can be checked using the following methods:
        - Scatter plots
        - Regression plots
        - Pearson Correlation
4. Test for **Homoscedasticity**
    - Create a scatterplot of residuals against predicted values.
    - Perform a Goldfeld-Quandt test to check the presence of

- **Heteroscedasticity** in the data.
- If the obtained `p-value > 0.05`, there is no strong evidence of heteroscedasticity.

5. Normality of Residuals
   - Almost bell-shaped curve in residuals distribution.
6. Impact of **Outliers**

1.9.1 Multicolinearity check:

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.

The formula for VIF is as follows:

`VIF(j) = 1 / (1 - R(j)^2)`

Where: - j represents the jth predictor variable. - R(j)^2 is the coefficient of determination (R-squared) obtained from regressing the jth predictor variable on all the other predictorvriables.

" - Calculate the VIF for each variable. - Identify variables with VIF greater than 5. - Drop the variable with the highest VIF. - Repeat steps 1-3 until no variable has a VIF eater than 5.
"

[49]:
```
# Import library to check for vif scores;
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

[50]:
```
vif = pd.DataFrame()
vif['Variable'] = x_train.columns
vif['VIF'] = [variance_inflation_factor(x_train.values, i) for i in␣
 ↪range(x_train.shape[1])]
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

[50]:

	Variable	VIF
5	CGPA	4.653698
0	GRE Score	4.489201
1	TOEFL Score	3.665067
3	SOP	2.785753
2	University Rating	2.571847
4	LOR	1.977668
6	Research	1.517206

Insights: As the Variance Inflation Factor(VIF) score is less than 5 for all the features we can say that there is no much multicolinearity between the features.

1.9.2 Mean of Residuals:

- 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.
- **The mean of residuals being close to zero indicates that, on average, the predictions made by the linear regression model are accurate, with an equal balance of overestimations and underestimations. This is a desirable characteristic of a well-fitted regression model.**

[51]:
```
residuals_test = (y_test - y_pred_test)
residuals_test.mean()
```

[51]:
```
-0.03867840379282768
```

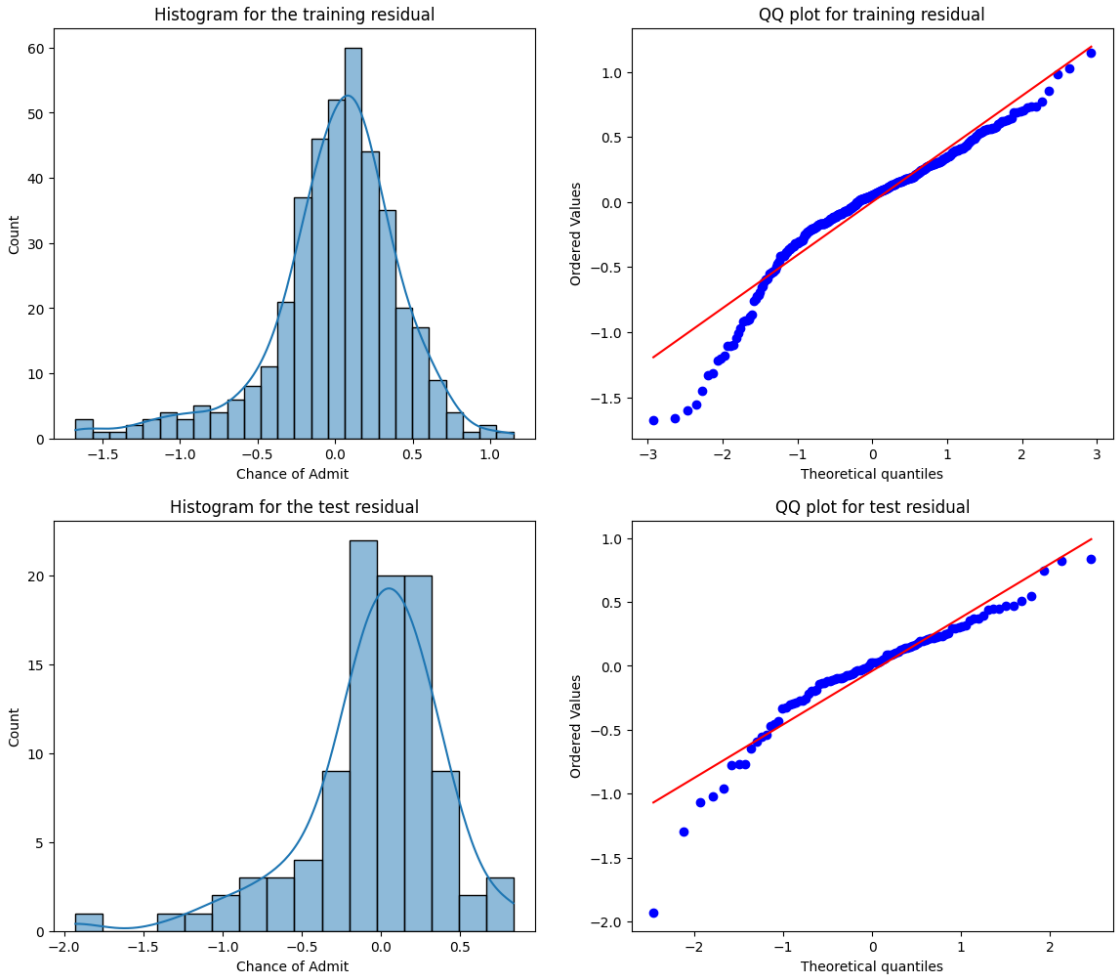
[52]:
```
residuals_train = (y_train - y_pred_train)
residuals_train.mean()
```

[52]:
```
9.436895709313831e-18
```

1.9.3 Insights:

Since the mean of residuals is very close to 0, we can say that the model is UnBiased.

[53]:
```
plt.figure(figsize=(14,12))
plt.subplot(2,2,1)
sns.histplot(residuals_train,kde = True)
plt.title('Histogram for the training residual')
plt.subplot(2,2,2)
stats.probplot(residuals_train, plot = plt)
plt.title('QQ plot for training residual')
plt.subplot(2,2,3)
sns.histplot(residuals_test,kde = True)
plt.title('Histogram for the test residual')
plt.subplot(2,2,4)
stats.probplot(residuals_test, plot = plt)
plt.title('QQ plot for test residual')
plt.show()
```



From the Histplot & kdeplot for the test data , we can see that the Residuals are left skewed and not perfectly normally distributed. The QQ plot shows that residuals are slightly deviating from the straight diagonal , thus not Gaussian.

1.9.4 The Mean of Residuals is Nearly Zero?

[54]:
```
# The Mean of residulas
residuals_test.mean(), residuals_train.mean()
```

[54]:
```
(-0.03867840379282768, 9.436895709313831e-18)
```

Conclusion:

By examing the difference between the y_actual and y_predicted value, it is observed that the mean of the Errors are approximately been vary close to Zero. Which suggests that, on average, the model is not systematically underestimating or overestimating the target variable. A mean residual close to zero indicates that the model is making predictions that are, on average, accurate. #Linearity of varibales: 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.

1.9.5 Linear Relationships:

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.

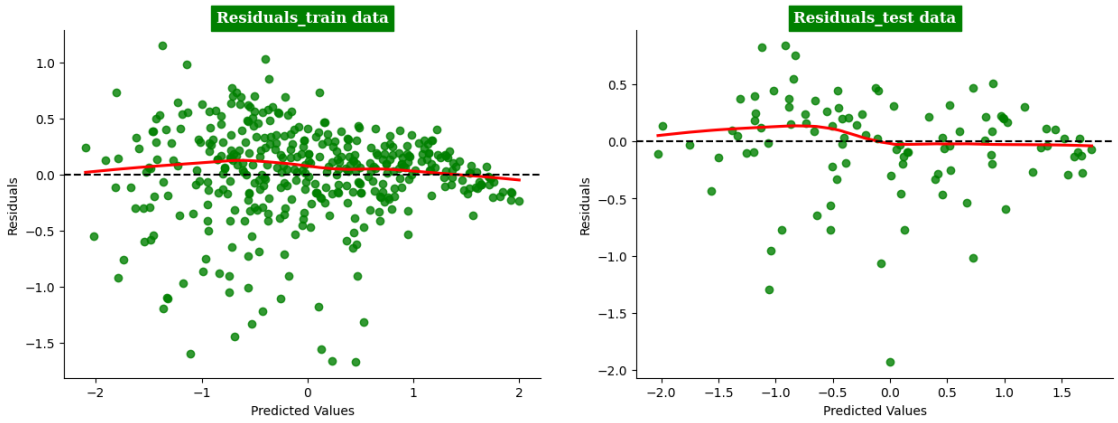
If there is a visible pattern in the residual plot, it suggests a violation of the linearity assumption. Common patterns that indicate non-linearity include:

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

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.

[55]:
```
plt.figure(figsize=(15,5))
plt.subplot(121)
plt.title('Residuals_train␣
 ↪data',fontsize=12,fontfamily='serif',fontweight='bold',backgroundcolor='g',color='w')
sns.regplot(x=y_pred_train, y=residuals_train, lowess=True,␣
 ↪color='g',line_kws={'color': 'red'})
plt.axhline(y=0, color='k', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.subplot(122)
plt.title('Residuals_test␣
 ↪data',fontsize=12,fontfamily='serif',fontweight='bold',backgroundcolor='g',color='w')
sns.regplot(x=y_pred_test, y=residuals_test, lowess=True,color='g'␣
 ↪,line_kws={'color': 'red'})
```

```
plt.axhline(y=0, color='k', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
sns.despine()
plt.show()
```



1.9.6 Insights:

From the Joint plot & pairplot in the graphical analysis, we can say that there is linear relationship between dependent variable and independent variables.

As we can observe, GRE Score, TOEFL Score and CGPA have a linear relationship with the Chance of Admit. Although GRE score and TOEFL score are more scattered, CGPA has a much more more linear relationship with the Chance of Admit. In a linear regression model, the residuals are 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.

1.10 Homoscedasticity

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.

To test for homoscedasticity, there are several graphical and statistical methods that you can use:

1. 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.

2. 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.

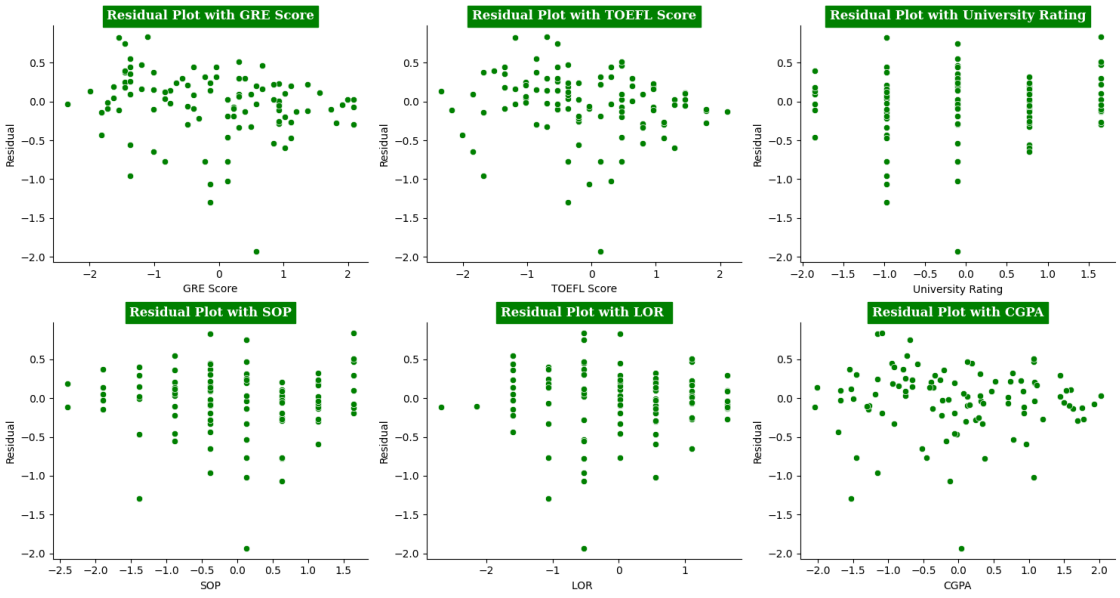
3. Breusch-Pagan Test: This is a statistical test for homoscedasticity. It involves regressing the squared residuals on the independent variables and checking the significance of the resulting model. If the p-value is greater than a chosen significance level (e.g., 0.05), it suggests homoscedasticity. However, this test assumes that the errors follow a normal distribution.

4. 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.

It's important to note that the visual inspection of plots is often the first step to identify potential violations of homoscedasticity. Statistical tests can provide additional evidence, but they may have assumptions or limitations that need to be considered.

Scatterplot of residuals with each independent variable to check for Homoscedasticity

[56]:
```
# Scatterplot of residuals with each independent variable to check for␣
↪Homoscedasticity
plt.figure(figsize=(15,8))

for i, col in enumerate(x_test.columns[:-1], 1):
    plt.subplot(2, 3, i)
    sns.scatterplot(x=x_test[col], y=residuals_test, color='g')
    plt.title(f'Residual Plot with {col}', fontsize=12, fontfamily='serif',␣
↪fontweight='bold', backgroundcolor='g', color='w')
    plt.xlabel(col)
    plt.ylabel('Residual')

plt.tight_layout()
sns.despine()
plt.show()
```



[57]:
```
ols_model = results
predicted = ols_model.predict()
residuals = ols_model.resid
```

[58]:
```
# Assuming results is your fitted OLS model
ols_model = results

# Obtain residuals and predicted values
residuals = ols_model.resid
predicted = ols_model.predict()

# Create a DataFrame for easier manipulation
df = pd.DataFrame({
    'predicted': predicted,
    'residuals': residuals
})

# Sort data by predicted values
df_sorted = df.sort_values(by='predicted').reset_index(drop=True)

# Split data into two groups
n = len(df_sorted)
split_point = n // 2
group1 = df_sorted.iloc[:split_point]
group2 = df_sorted.iloc[split_point:]

# Extract the independent variables from the original model
X = ols_model.model.exog  # Get the original design matrix

# Split the design matrix according to the groups
X_group1 = X[:split_point]
X_group2 = X[split_point:]

# Fit models to each group
model_group1 = sm.OLS(group1['residuals'], X_group1).fit()
model_group2 = sm.OLS(group2['residuals'], X_group2).fit()

# Calculate residual variances
var_group1 = np.var(model_group1.resid, ddof=1)
var_group2 = np.var(model_group2.resid, ddof=1)

# Compute Goldfeld-Quandt test statistic
F_statistic = var_group1 / var_group2

# Compute p-value for the F-statistic
```

```
df1 = X_group1.shape[1] - 1  # Degrees of freedom for group1
df2 = len(group2) - X_group2.shape[1]  # Degrees of freedom for group2
p_value = 1 - stats.f.cdf(F_statistic, df1, df2)

# Create a DataFrame for the test results similar to Breusch-Pagan
goldfeld_quandt_test = pd.DataFrame({
    'value': [F_statistic, p_value],
}, index=['F-statistic', 'p-value'])

# Display the results
print(goldfeld_quandt_test)
```

```
                value
F-statistic  2.338222
p-value      0.025956
```

Insights:

Since we do not see any significant change in the spread of residuals with respect to change in independent variables, we can conclude that Homoscedasticity is met . Since the p-value is much lower than the alpha value, we can Reject the null hypothesis and conclude that Heteroscedasticity is present Since the p-value is significantly less than the conventional significance level (e.g., 0.05), we reject the null hypothesis of homoscedasticity. This suggests that there is evidence of heteroscedasticity in the residuals, indicating that the variance of the residuals is not constant across all levels of the independent variables. This violation of the homoscedasticity assumption may affect the validity of the linear regression model's results. >

1.10.1 Normality of Residuals:

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.

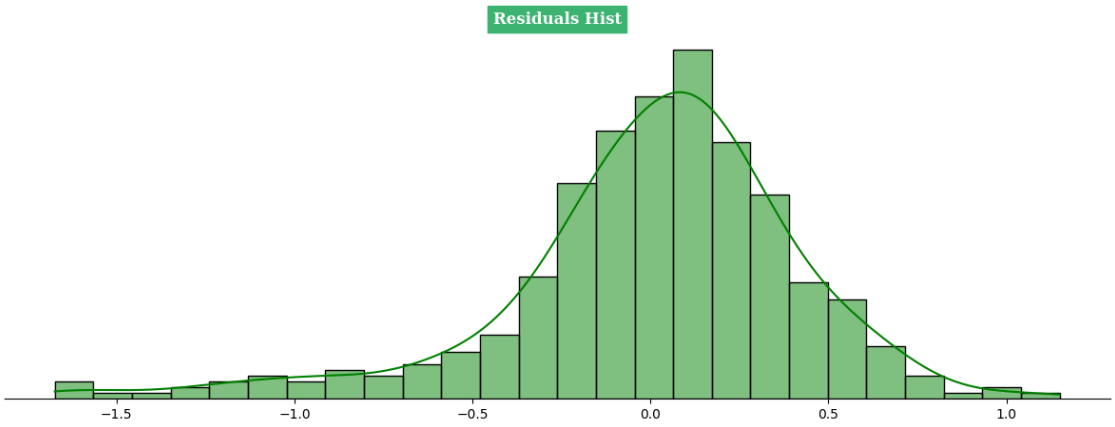
`Shapiro-Wilk Test`: This is a statistical test that checks the null hypothesis that the residuals are normally distributed. The Shapiro-Wilk test calculates a test statistic and

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.

>> **`Anderson-Darling or Jarque_Bera`** can also be done as data size increases.

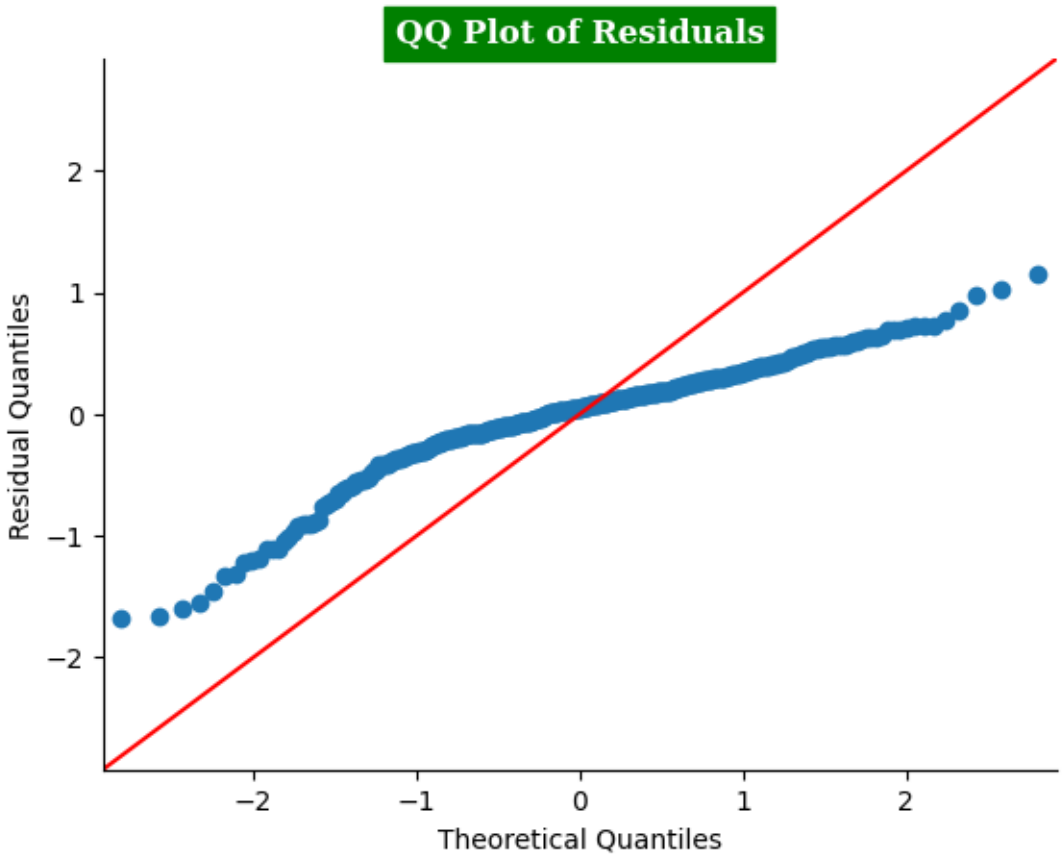
Skewness and Kurtosis: Calculate the skewness and kurtosis of the residuals. Skewness measures the asymmetry of the distribution, and a value close to zero suggests normality. Kurtosis measures the heaviness of the tails of the distribution compared to a normal distribution, and a value close to zero suggests similar tail behavior.

[59]:
```
plt.figure(figsize=(15,5))
sns.histplot(residuals, kde= True,color='g')
plt.title('Residuals␣
 ↪Hist',fontsize=12,fontfamily='serif',fontweight='bold',backgroundcolor='mediumseagreen',col
sns.despine(left=True)
plt.ylabel("")
plt.yticks([])
plt.show()
```



[60]:
```
# QQ-Plot of residuals
plt.figure(figsize=(15,5))
sm.qqplot(residuals,line='45')
plt.title('QQ Plot of␣
 ↪Residuals',fontsize=12,fontfamily='serif',fontweight='bold',backgroundcolor='g',color='w')
plt.ylabel('Residual Quantiles')
sns.despine()
plt.show();
```

```
<Figure size 1500x500 with 0 Axes>
```



1.10.2 JARQUE BERA test:

[61]:
```
jb_stat, jb_p_value = stats.jarque_bera(residuals)


print("Jarque-Bera Test Statistic:", jb_stat)
print("p-value:", jb_p_value)

if jb_p_value < 0.05:
    print("Reject the null hypothesis: Residuals are not normally distributed.")
else:
    print("Fail to reject the null hypothesis: Residuals are normally␣
 ↪distributed.")
```

```
Jarque-Bera Test Statistic: 190.09887364276915
p-value: 5.25477446043155e-42
Reject the null hypothesis: Residuals are not normally distributed.
```

Insights: From the Histplot & kdeplot , we can see that the Residuals are `left skewed` and not perfectly normally distributed.

The QQ plot shows that residuals are slightly deviating from the straight diagonal , thus not `Gaussian`.

From Jarque Bera test , we conclude that the Residuals are `Not Normally distributed`.

Hence this assumption is not met.

1.11 Lasso and Ridge Regression - L1 & L2 Regularization

Ridge and Lasso regression are both regularization techniques used to prevent overfitting in linear regression models. They work by adding a penalty term to the cost function, which helps to control the complexity of the model by shrinking the coefficient values.

Lasso Regression: Lasso regression uses L1 regularization, where the penalty term is the sum of the absolute values of the coefficients multiplied by a regularization parameter (lambda or alpha). Lasso regression has the ability to shrink some coefficients to exactly zero, effectively performing feature selection. This makes Lasso regression useful when dealing with high-dimensional data where only a few variables are relevant.

Ridge Regression: Ridge regression uses L2 regularization, where the penalty term is the squared sum of the coefficients multiplied by a regularization parameter (lambda or alpha). The regularization term helps to reduce the impact of less important features on the model and prevents them from dominating the model. Ridge regression can help in reducing the variance of the model and is particularly useful when dealing with multicollinearity (high correlation between independent variables).

The main differences between Ridge and Lasso regression are:

- Ridge regression tends to shrink all coefficient values towards zero, but it rarely makes them exactly zero. On the other hand, Lasso regression can make coefficient values exactly zero, performing variable selection.
- Ridge regression is suitable when dealing with multicollinearity, as it will shrink correlated variables together. Lasso regression, however, can select one variable from a set of highly correlated variables and make the others zero.

[62]:
```
model_lasso = Lasso(alpha=0.45)
model_lasso.fit(x_train, y_train)
```

[62]:
```
Lasso(alpha=0.45)
```

reason for (alpha) ... try giving just lasso()... Weights are all zero.. inorder to manipulate that we introduce an alpha... In Lasso regression, the alpha parameter ( ) controls the strength of regularization. It's typically written as a positive value greater than 0. The choice of alpha affects the amount of shrinkage applied to the coefficients.

You would specify the alpha value when initializing the Lasso regression model. Typically, you would experiment with different alpha values to find the one that best balances model complexity and performance on your dataset through techniques like cross-validation.

For example, if you're using scikit-learn in Python, you would set the alpha parameter when creating the Lasso regression model object, like this:

```
from sklearn.linear_model import Lasso

lasso_model = Lasso(alpha= )
```

Here, `alpha` is set to 0.1, but you can adjust it based on your experimentation and validation results. Lower values of alpha result in less regularization, potentially leading to overfitting, while higher values increase regularization, potentially improving generalization to unseen data.

[63]:
```
model_ridge = Ridge()
model_ridge.fit(x_train, y_train)
```

[63]:
```
Ridge()
```

[64]:
```
y_pred_train_ridge = model_ridge.predict(x_train)
y_pred_test_ridge = model_ridge.predict(x_test)

y_pred_train_lasso = model_lasso.predict(x_train)
y_pred_test_lasso = model_lasso.predict(x_test)
```

[65]:
```
lasso_model_weights = pd.DataFrame(model_lasso.coef_.
 ↪reshape(1,-1),columns=copy_Jamboree_data.columns[:-1])
lasso_model_weights["Intercept"] = model_lasso.intercept_
lasso_model_weights
```

[65]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Intercept
0	0.019231	0.0	0.0	0.0	0.0	0.408647	0.0	0.013919

[66]:
```
ridge_model_weights = pd.DataFrame(model_ridge.coef_.
 ↪reshape(1,-1),columns=copy_Jamboree_data.columns[:-1])
ridge_model_weights["Intercept"] = model_ridge.intercept_
ridge_model_weights
```

[66]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Intercept
0	0.195584	0.130073	0.021575	0.013802	0.113221	0.478123	0.084673	0.007726

[67]:
```
print('Linear Regression Training Accuracy\n')
model_evaluation(y_train.values, y_pred_train, lr_model)
print('*'*25)
print('\nLinear Regression Test Accuracy\n')
```

```
model_evaluation(y_test.values, y_pred_test, lr_model)
print('---'*25)
print('\nRidge Regression Training Accuracy\n')
model_evaluation(y_train.values, y_pred_train_ridge, model_ridge)
print('*'*25)
print('\n\nRidge Regression Test Accuracy\n')
model_evaluation(y_test.values, y_pred_test_ridge, model_ridge)
print('---'*25)
print('\n\nLasso Regression Training Accuracy\n')
model_evaluation(y_train.values, y_pred_train_lasso, model_lasso)
print('*'*25)
print('\n\nLasso Regression Test Accuracy\n')
model_evaluation(y_test.values, y_pred_test_lasso, model_lasso)
print('---'*25)
```

```
Linear Regression Training Accuracy

MSE: 0.18
MAE: 0.3
RMSE: 0.42
R2 Score: 0.82
Adjusted R2: 0.82
*************************

Linear Regression Test Accuracy

MSE: 0.19
MAE: 0.3
RMSE: 0.43
R2 Score: 0.82
Adjusted R2: 0.81
---------------------------------------------------------------------------

Ridge Regression Training Accuracy

MSE: 0.18
MAE: 0.3
RMSE: 0.42
R2 Score: 0.82
Adjusted R2: 0.82
*************************


Ridge Regression Test Accuracy

MSE: 0.19
MAE: 0.3
```

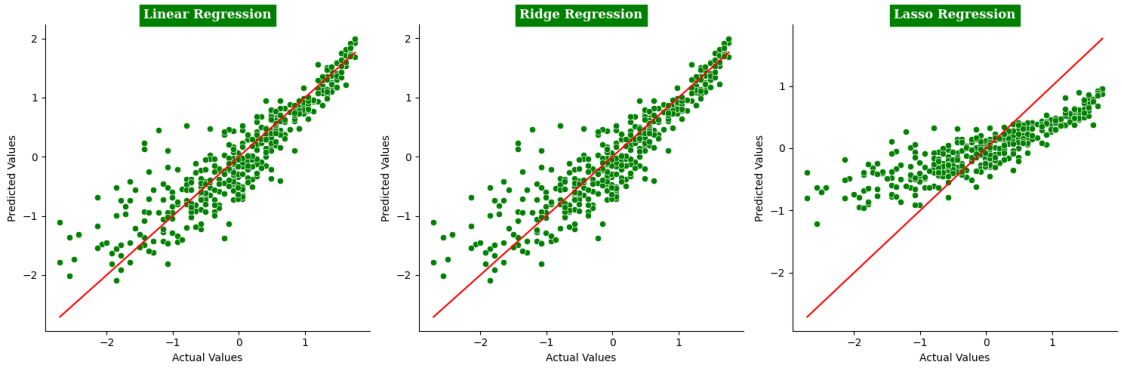
```
RMSE: 0.43
R2 Score: 0.82
Adjusted R2: 0.81
--------------------------------------------------------------------------------


Lasso Regression Training Accuracy

MSE: 0.43
MAE: 0.52
RMSE: 0.65
R2 Score: 0.57
Adjusted R2: 0.56
**********


Lasso Regression Test Accuracy

MSE: 0.43
MAE: 0.51
RMSE: 0.65
R2 Score: 0.58
Adjusted R2: 0.55
--------------------------------------------------------------------------------
```

1.11.1 Observation:

- While Linear Regression and Ridge regression have similar scores, Lasso regression has not performed well on both training and test data

[68]:
```
actual_values = y_train.values.reshape((-1,))
predicted_values = [y_pred_train.reshape((-1,)), y_pred_train_ridge.
 ↪reshape((-1,)), y_pred_train_lasso.reshape((-1,))]
model = ['Linear Regression', 'Ridge Regression', 'Lasso Regression']
plt.figure(figsize=(15,5))
i=1
for preds in predicted_values:
    plt.subplot(1,3,i)
    sns.scatterplot(x=actual_values, y=preds,color='g')
    plt.plot([np.min(actual_values), np.max(actual_values)], [np.
 ↪min(actual_values), np.max(actual_values)], 'r-')
    plt.xlabel('Actual Values')
    plt.ylabel('Predicted Values')
    plt.
 ↪title(model[i-1],fontsize=12,fontfamily='serif',fontweight='bold',backgroundcolor='g',color
    i+=1
plt.tight_layout()
sns.despine()
```

```
plt.show();
```



1.11.2 Insights:

We can observe that both `Linear Regression` and `Ridge Regression` have similar accuracy while Lasso regression has oversimplified the model.

This is the reason that the r2 score of Lasso regression is 0. It doesn't capture any variance in the target variable. It has predicted the same value across all instances.

1.11.3 ElasticNet regression:

Elastic Net regression is a type of regression analysis that combines penalties from both Lasso (L1 regularization) and Ridge (L2 regularization) methods. It is particularly useful when dealing with datasets that have high dimensionality and multicollinearity (correlation between predictors).

Here's how Elastic Net works:

1. **Objective Function**: In Elastic Net regression, the objective function includes two components: the residual sum of squares (RSS), which measures the difference between the observed and predicted values, and two penalty terms: one for L1 regularization and one for L2 regularization.

2. **L1 Regularization (Lasso)**: The L1 penalty encourages sparsity in the coefficient estimates by adding the absolute values of the coefficients to the objective function. This tends to force some coefficients to be exactly zero, effectively performing feature selection.

3. **L2 Regularization (Ridge)**: The L2 penalty adds the squared values of the coefficients to the objective function. This tends to shrink the coefficients towards zero, but it does not enforce sparsity as strongly as L1 regularization.

4. **Mixing Parameter ( )**: Elastic Net introduces a mixing parameter, (alpha), which controls the balance between L1 and L2 regularization. When = 0, Elastic Net reduces to Ridge regression, and when = 1, it reduces to Lasso regression. Intermediate values of allow for a combination of both penalties.

5. **Regularization Strength ( )**: Additionally, Elastic Net includes a regularization strength parameter, (lambda), which controls the overall strength of regularization. Higher values of

result in stronger regularization, leading to more shrinkage of coefficients.

Benefits of Elastic Net regression include:

- **Feature Selection**: Like Lasso regression, Elastic Net can perform automatic feature selection by setting some coefficients to zero.
- **Handling Multicollinearity**: Like Ridge regression, Elastic Net can handle multicollinearity by shrinking the coefficients of correlated predictors.
- **Flexibility**: The mixing parameter allows for a flexible trade-off between L1 and L2 regularization, offering more control over the type of regularization applied.

However, Elastic Net also has some drawbacks, such as the need to tune hyperparameters like and , and it may be computationally more expensive compared to simpler regression methods.

Overall, Elastic Net regression is a powerful technique for regression analysis, especially in situations where there are many predictors with potentially correlated features. It strikes a balance between the strengths of Lasso and Ridge regression, offering flexibility and improved performance in certain scenarios.

we have to find alpha... find the value such that loss is minimum Finding the optimal alpha value for ElasticNet regression typically involves techniques like cross-validation. Here's a general approach to finding the optimal alpha value:

1. **Define a Range of Alpha Values:** Start by defining a range of alpha values to search over. This range should cover a broad spectrum of possibilities, from very small values (close to 0) to larger values.
2. **Cross-Validation:** Use k-fold cross-validation to evaluate the performance of the ElasticNet regression model for each alpha value in the defined range. For each fold, train the model on the training data and evaluate its performance on the validation data.
3. **Select the Best Alpha:** Choose the alpha value that results in the best performance metric on the validation set. This metric could be mean squared error (MSE), mean absolute error (MAE), R-squared, or another appropriate metric depending on your specific problem.
4. **Final Model:** Once you have selected the best alpha value using cross-validation, retrain the ElasticNet regression model using the entire training dataset and the chosen alpha value. This will be your final model.

Here's an example of how you can perform cross-validated grid search for alpha value in scikit-learn:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import ElasticNet

# Define a range of alpha values to search over
alpha_range = [0.001, 0.01, 0.1, 1.0, 10.0]

# Create ElasticNet regression model
elastic_net = ElasticNet()

# Define grid search parameters
param_grid = {'alpha': alpha_range}
```

```
# Perform grid search with cross-validation
grid_search = GridSearchCV(estimator=elastic_net, param_grid=param_grid, cv=5, scoring='neg_mea
grid_search.fit(X_train, y_train)

# Get the best alpha value
best_alpha = grid_search.best_params_['alpha']
```

In this example, `alpha_range` defines the range of alpha values to search over. Grid search is performed with 5-fold cross-validation, and the best alpha value is selected based on the mean squared error (MSE) metric. Adjust the `alpha_range` according to your problem domain and dataset characteristics.

– same for lasso and ridge as well

Here i did manual calculations for alpha

[69]:
```
ElasticNet_model = ElasticNet(alpha=0.1)
ElasticNet_model.fit(x_train , y_train)
```

[69]:
```
ElasticNet(alpha=0.1)
```

[70]:
```
y_pred_train_en = ElasticNet_model.predict(x_train)
y_pred_test_en = ElasticNet_model.predict(x_test)
```

[71]:
```
train_R2 = ElasticNet_model.score(x_train,y_train)
test_R2 = ElasticNet_model.score(x_test,y_test)
train_R2 , test_R2
```

[71]:
```
(0.815275981798617, 0.8160571185396226)
```

[72]:
```
en_model_weights = pd.DataFrame(ElasticNet_model.coef_.
 ↪reshape(1,-1),columns=copy_Jamboree_data.columns[:-1])
en_model_weights["Intercept"] = ElasticNet_model.intercept_
en_model_weights
```

[72]:

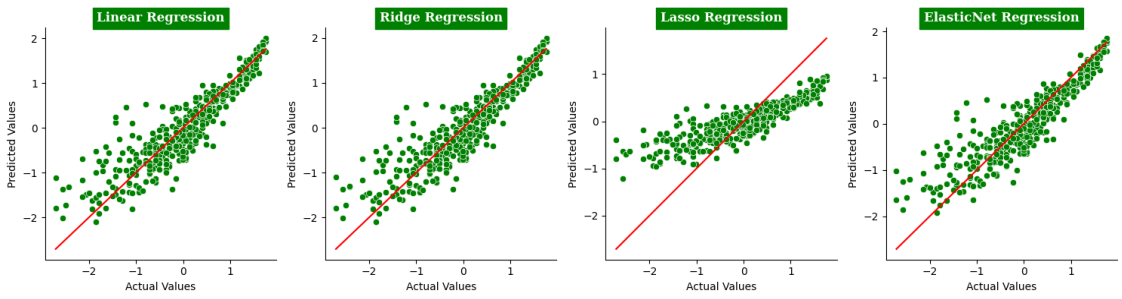
	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Intercept
0	0.195256	0.133398	0.025306	0.020393	0.093235	0.422272	0.060278	0.00772

[73]:
```
print('ElasticNet Regression Training Accuracy\n')
model_evaluation(y_train.values, y_pred_train_en, ElasticNet_model)
print('*'*25)
print('\nElasticNet Regression Test Accuracy\n')
model_evaluation(y_test.values, y_pred_test_en, ElasticNet_model)
print('---'*25)
```

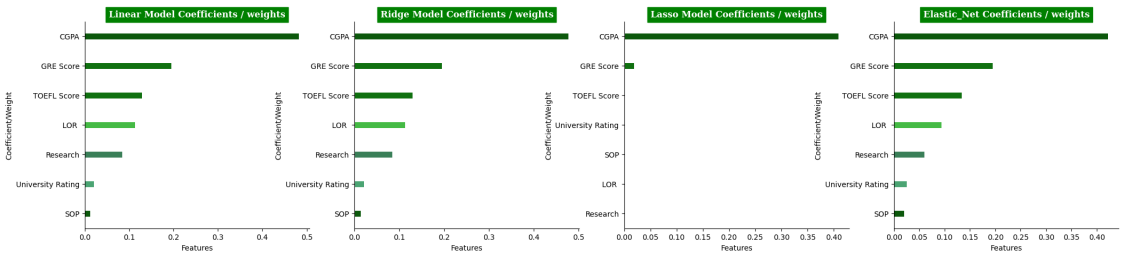
```
ElasticNet Regression Training Accuracy
```

```
MSE: 0.18
MAE: 0.31
RMSE: 0.43
R2 Score: 0.82
Adjusted R2: 0.82
**************************

ElasticNet Regression Test Accuracy

MSE: 0.19
MAE: 0.3
RMSE: 0.43
R2 Score: 0.82
Adjusted R2: 0.81
--------------------------------------------------------------------------------
```

[74]:
```
actual_values = y_train.values.reshape((-1,))
predicted_values = [y_pred_train.reshape((-1,)), y_pred_train_ridge.
 ↪reshape((-1,)),
                    y_pred_train_lasso.reshape((-1,)),y_pred_train_en.
 ↪reshape((-1,))]
model = ['Linear Regression', 'Ridge Regression', 'Lasso␣
 ↪Regression','ElasticNet Regression']
plt.figure(figsize=(15,4))
i=1
for preds in predicted_values:
    plt.subplot(1,4,i)
    sns.scatterplot(x=actual_values, y=preds,color='g')
    plt.plot([np.min(actual_values), np.max(actual_values)], [np.
 ↪min(actual_values), np.max(actual_values)], 'r-')
    plt.xlabel('Actual Values')
    plt.ylabel('Predicted Values')
    plt.
 ↪title(model[i-1],fontsize=12,fontfamily='serif',fontweight='bold',backgroundcolor='g',color
    i+=1
plt.tight_layout()
sns.despine()
plt.show();
```



[75]:
```
model_major_weights = {"Linear Model":lr_model_weights,
                       "Ridge Model":ridge_model_weights,
                       "Lasso Model":lasso_model_weights,
                       "Elastic_Net":en_model_weights}
# excluding w0-intercept
plt.figure(figsize=(25,5))
i=1
for model,data in model_major_weights.items():
    model_weights_data = data.melt()

    plt.subplot(1,4,i)
    sns.barplot(data=model_weights_data[:-1].
 ↪sort_values(by='value',ascending=False),
                y='variable', x='value',width=0.
 ↪2,palette=['darkgreen','g','green','limegreen','seagreen','mediumseagreen'])
    plt.xlabel('Features')
    plt.ylabel('Coefficient/Weight')
    plt.title(f'{model} Coefficients /␣
 ↪weights',fontsize=12,fontfamily='serif',fontweight='bold',backgroundcolor='g',color='w')
    i+=1
sns.despine()
plt.show()
```



1.12 Regression Analysis Summary:

- Upon conducting regression analysis, it's evident that CGPA emerges as the most influential feature in predicting admission chances.
- Additionally, GRE and TOEFL scores also exhibit significant importance in the predictive model.
- Following the initial regression model, a thorough check for multicollinearity was performed, revealing VIF scores consistently below 5, indicative of low multicollinearity among predictors.
- Despite the absence of high multicollinearity, it's noteworthy that the residuals do not conform perfectly to a normal distribution. Furthermore, the residual plots indicate some level of heteroscedasticity.
- Subsequent exploration involving regularized models such as Ridge and Lasso regression showcased comparable results to the Linear Regression Model.
- Moreover, employing ElasticNet (L1+L2) regression yielded results consistent with the other regression models, further reinforcing the predictive capabilities of the features under consideration.

1.13 Business Insights & Recommendations

1.13.1 Insights:

- First column was observed as unique row identifier which was dropped and was not required for model building.
- University Rating , SOP and LOR strength and research are seems to be discrete random Variables , but also ordinal numeric data.
- All the other features are numeric, ordinal and continuous.
- No null values were present in data.
- No Significant amount of outliers were found in data.sChance of admission(target variable) and GRE score(an independent feature) are nearly normally distribted.
- Independent Variables (Input data): GRE Score, TOEFL Score, University Rating, SOP, LOR, CGPA, Research
- Target/Dependent Variable : Chance of Admit (the value we want to predict)
- From correlation heatmap , we can observe GRE score, TOEFL score and CGPA have very high correlation with Change of admission.
- University rating, SOP ,LOR and Research have comparatively slightly less correlated than other features.
- Chances of admit is a probability measure , which is within 0 to 1 which is good (no outliers or missleading data in column).
- Range of GRE score looks like between 290 to 340.
- Range of TOEFL score is between 92 to 120.

- University rating , SOP and LOR are distributed between range of 1 to 5.
- CGPA range is between 6.8 to 9.92.
- From boxplots (distribution of chance of admition (probability of getting admition) as per GRE score) : with higher GRE score , there is high probability of getting an admission.
- Students having high toefl score , has higher probability of getting admission.
- From count plots, we can observe , statement of purpose SOP strength is positively correlated with Chance of Admission.
- We can also similar pattern in Letter of Recommendation Stength and University rating , have positive correlation with Chaces of Admission.
- Student having research has higher chances of Admission , but also we can observe some outliers within that category.

  Model Predictors:
  Our analysis identified several key predictors strongly correlated with admission chances. Notably, **GRE score, TOEFL score, and CGPA** emerged as significant factors influencing admission probabilities.

  Multicollinearity Check:
  Assessing multicollinearity revealed no significant issues, indicating the robustness of our model despite high correlations among predictors.

  Model Performance: Both Linear Regression and Ridge Regression models exhibited promising performance, capturing up to approximentally 82% of the variance in admission probabilities.

  Data Distribution: Exploratory data analysis uncovered left-skewed distributions in admission probabilities and strong positive correlations between exam scores and admission chances.

1.13.2 Recommendations:

Feature Enhancement: Encourage students to focus on improving GRE scores, CGPA, and the quality of Letters of Recommendation (LOR), as these factors significantly influence admission chances.

Data Augmentation: Collect a wider range of data beyond academic metrics to capture applicants' holistic profiles, including extracurricular achievements, personal statements, and diversity factors.

Additional Features: Given the strong correlation among CGPA, we can enrich the predictive model with additional diverse features such as Research, work experience, internships, or extracurricular activities.

By implementing these recommendations, we can further enhance our admissions process, providing valuable insights and support to both applicants and educational institutions.
