Jamboree Education - Linear Regression

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.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
df=pd.read_csv("Jamboree_Admission.csv")
df.head(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

Next steps: Generate code with df | View recommended plots

```
df=df.drop('Serial No.',axis=1)
```

We have removed the "Serial Number" column since its data does not contribute to the analysis.

Observations on shape of data, data types of all the attributes, conversion of categorical attributes to 'category' (If required) , missing value detection, statistical summary.

```
df.head()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65

Next steps: Generate code with `df` View recommended plots

```
df.columns
```

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

```
df.shape
```

```
(500, 8)
```

```
df.isnull().sum()
```

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

There are no null values

```
df.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 data type

```
df.describe()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Researc
count	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.00000
mean	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.56000
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.49688
min	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.00000
25%	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.00000
50%	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.00000
75%	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.00000

Univariate Analysis (distribution plots of all the continuous variable(s) barplots/countplots of all the categorical variables)

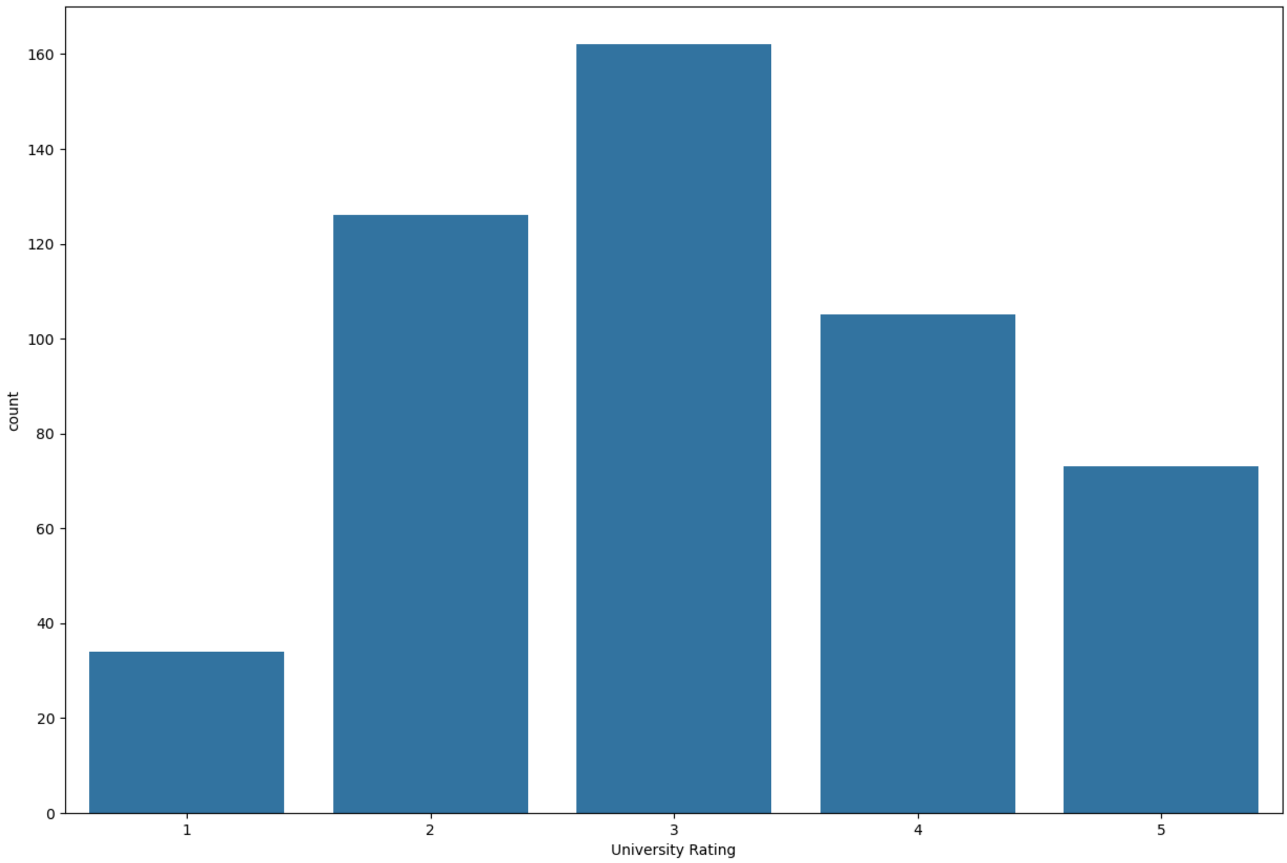
```
df["University Rating"].unique()
```

```
array([4, 3, 2, 5, 1])
```

```
df["University Rating"].value_counts()
```

```
3    162
2    126
4    105
5     73
1     34
Name: University Rating, dtype: int64
```

```
sns.countplot(data=df,x="University Rating")
plt.show()
```



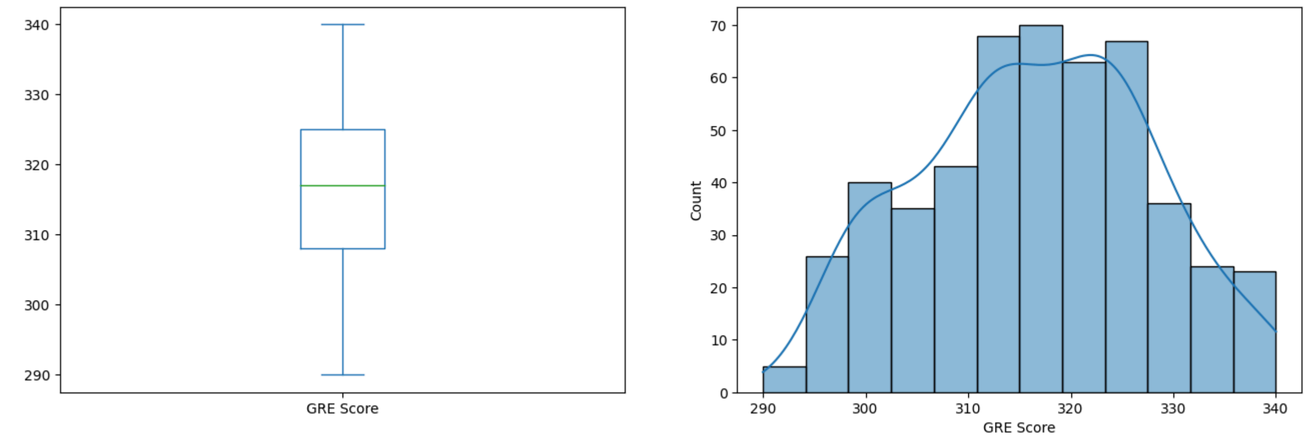

```
df["GRE Score"].unique()
```

```
array([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])
```

```
df["GRE Score"].value_counts(bins=5)
```

```
(310.0, 320.0]      154
(320.0, 330.0]      141
(300.0, 310.0]       96
(330.0, 340.0]       56
(289.949, 300.0]     53
Name: GRE Score, dtype: int64
```

```
plt.subplot(121)
df["GRE Score"].plot.box(figsize=(16,5))
plt.subplot(122)
sns.histplot(df["GRE Score"], kde=True)
plt.show()
```



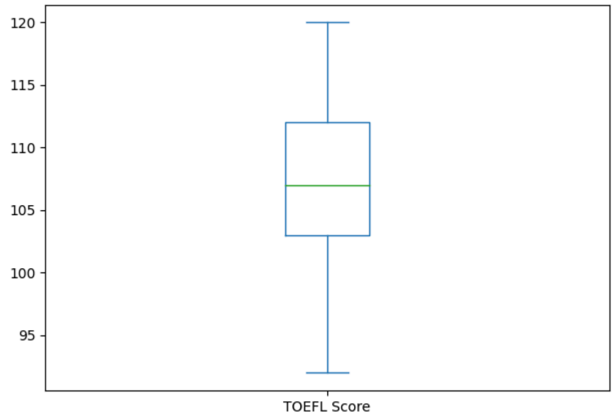

```
df["TOEFL Score"].unique()
```

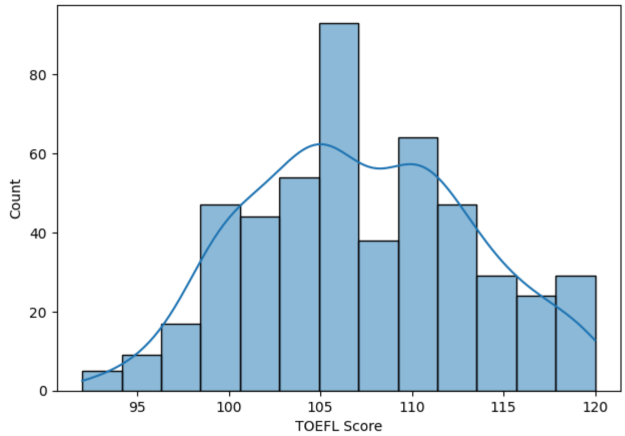
```
array([118, 107, 104, 110, 103, 115, 109, 101, 102, 108, 106, 111, 112,
       105, 114, 116, 119, 120,  98,  93,  99,  97, 117, 113, 100,  95,
        96,  94,  92])
```

```
df["TOEFL Score"].value_counts(bins=5)
```

```
(108.8, 114.4]                148
(103.2, 108.8]                141
(97.6, 103.2]                 126
(114.4, 120.0]                 64
(91.97099999999999, 97.6]      21
Name: TOEFL Score, dtype: int64
```

```
plt.subplot(121)
df["TOEFL Score"].plot.box(figsize=(16,5))
plt.subplot(122)
sns.histplot(df["TOEFL Score"], kde=True)
plt.show()
```





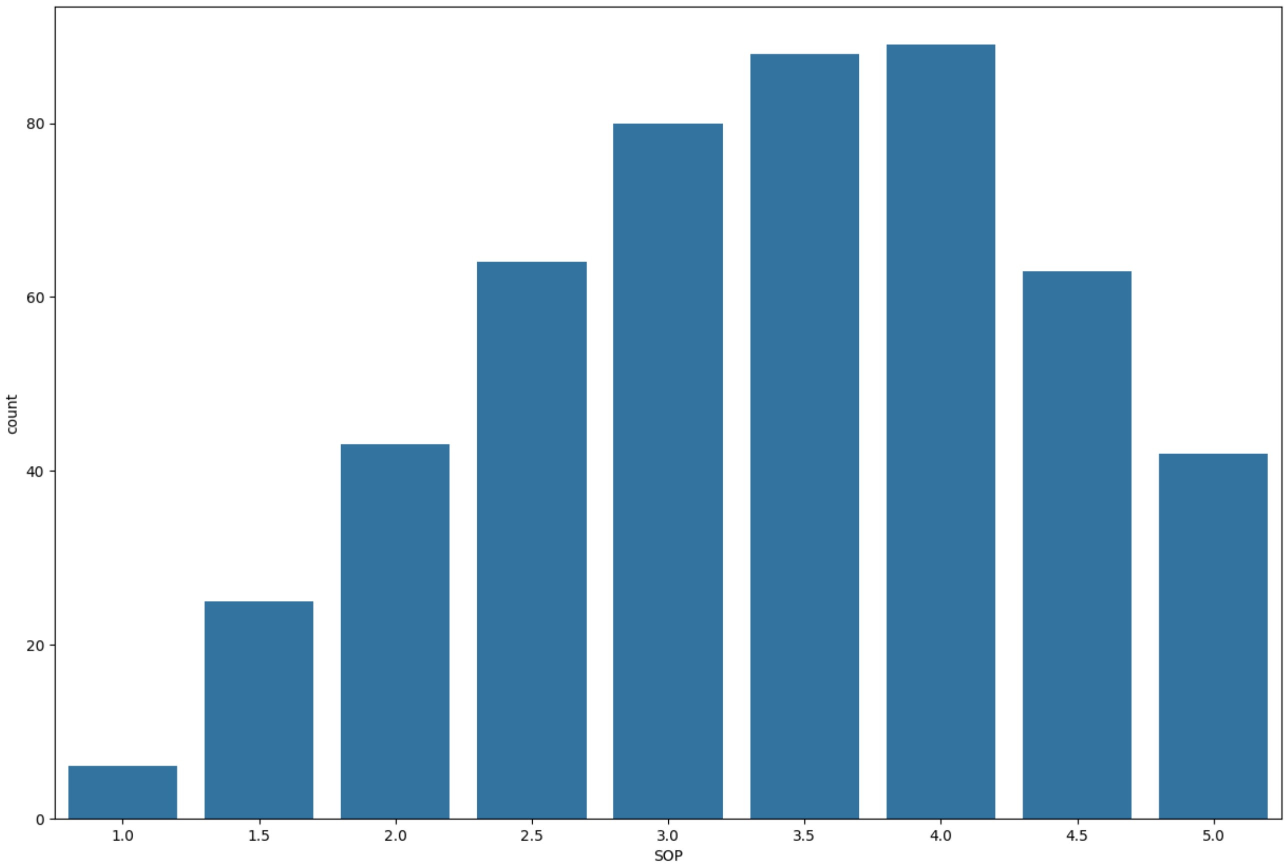

```
df["SOP"].unique()
```

```
array([4.5, 4. , 3. , 3.5, 2. , 5. , 1.5, 1. , 2.5])
```

```
df["SOP"].value_counts(bins=2)
```

```
(3.0, 5.0]      282
(0.995, 3.0]    218
Name: SOP, dtype: int64
```

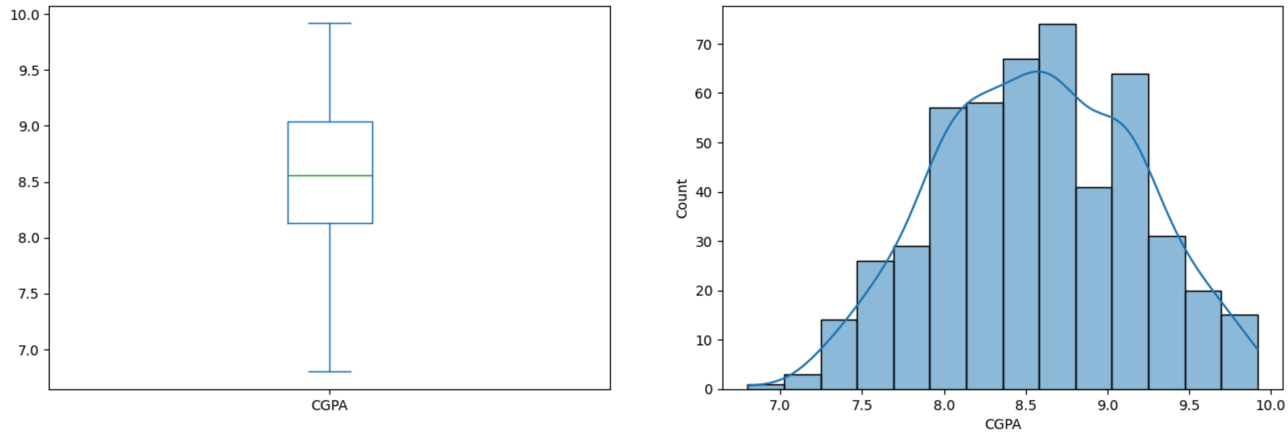
```
sns.countplot(data=df,x="SOP")
plt.show()
```




```
df["CGPA"].nunique()

     184


df["CGPA"].value_counts(bins=5)

     (8.048, 8.672]               175
     (8.672, 9.296]               156
     (7.424, 8.048]                96
```

```
(9.296, 9.92]                  61
(6.795999999999999, 7.424]     12
Name: CGPA, dtype: int64
```

```
plt.subplot(121)
df["CGPA"].plot.box(figsize=(16,5))
plt.subplot(122)
sns.histplot(df["CGPA"], kde=True)
plt.show()
```



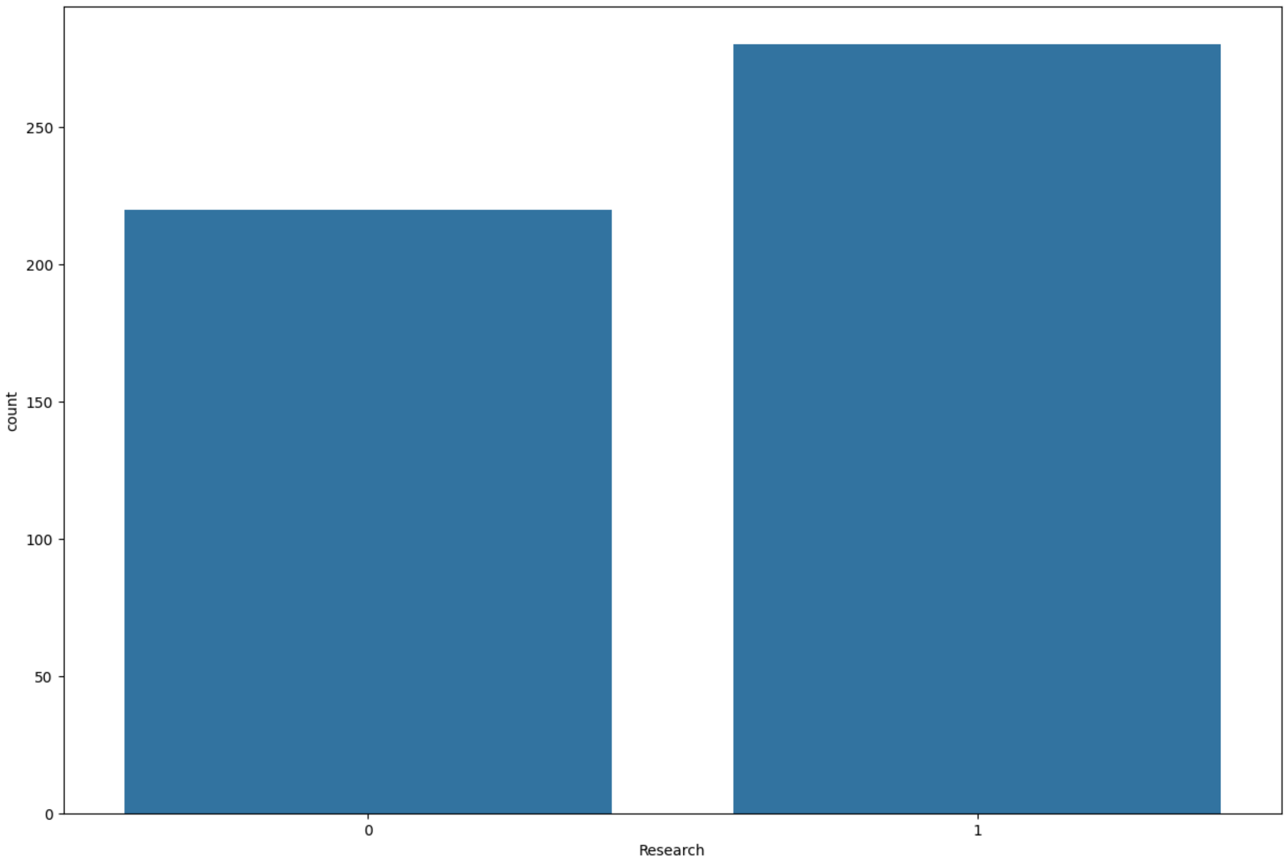

```
df["Research"].unique()
```

```
array([1, 0])
```

```
df["Research"].value_counts()
```

```
1    280
0    220
Name: Research, dtype: int64
```

```
sns.countplot(data=df,x="Research")
plt.show()
```

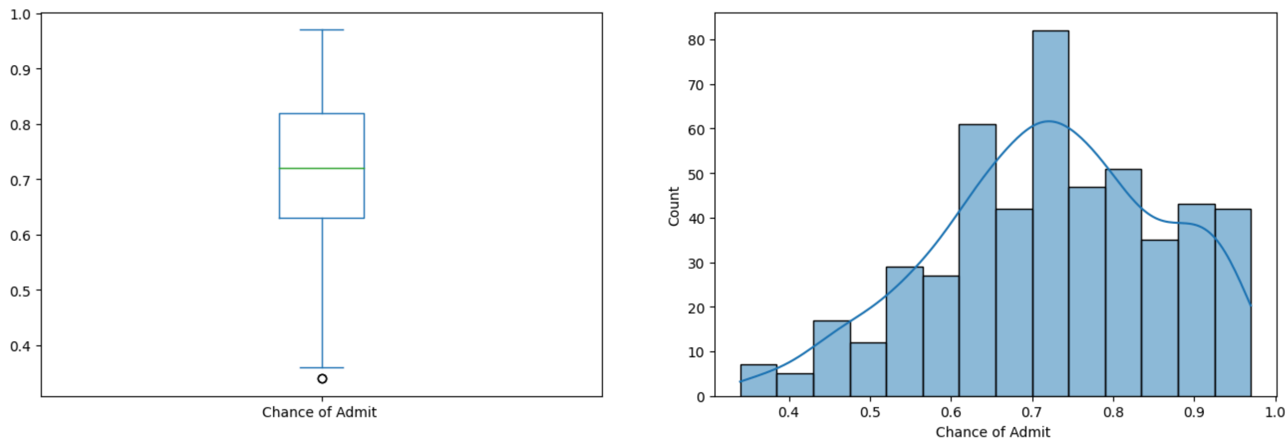



```
df["Chance of Admit "].unique()
```

```
array([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])
```

```
df["Chance of Admit "].value_counts(bins=5)

    (0.718, 0.844]    155
    (0.592, 0.718]    141
    (0.844, 0.97]     109
    (0.466, 0.592]     71
    (0.338, 0.466]     24
    Name: Chance of Admit , dtype: int64
```

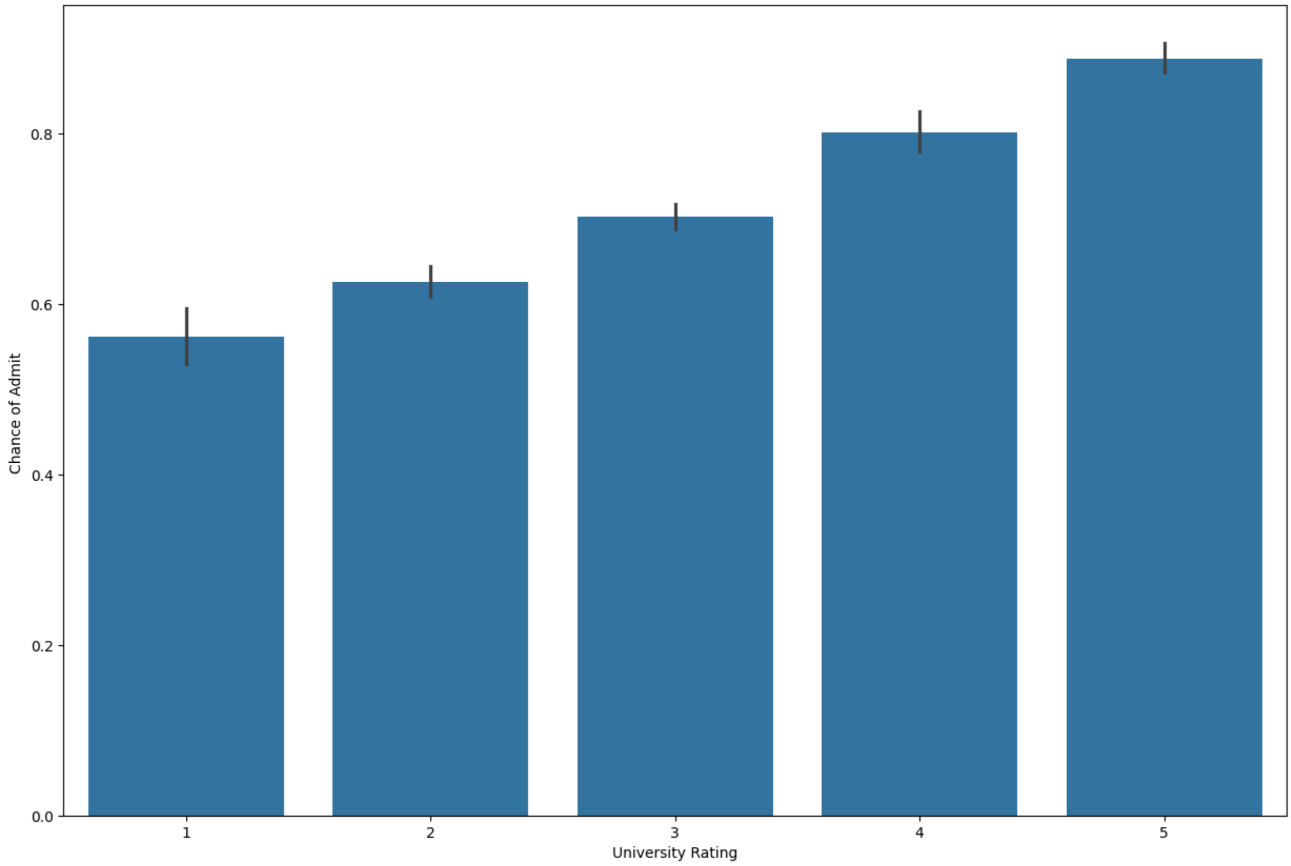
```
plt.subplot(121)
df["Chance of Admit "].plot.box(figsize=(16,5))
plt.subplot(122)
sns.histplot(df["Chance of Admit "], kde=True)
plt.show()
```



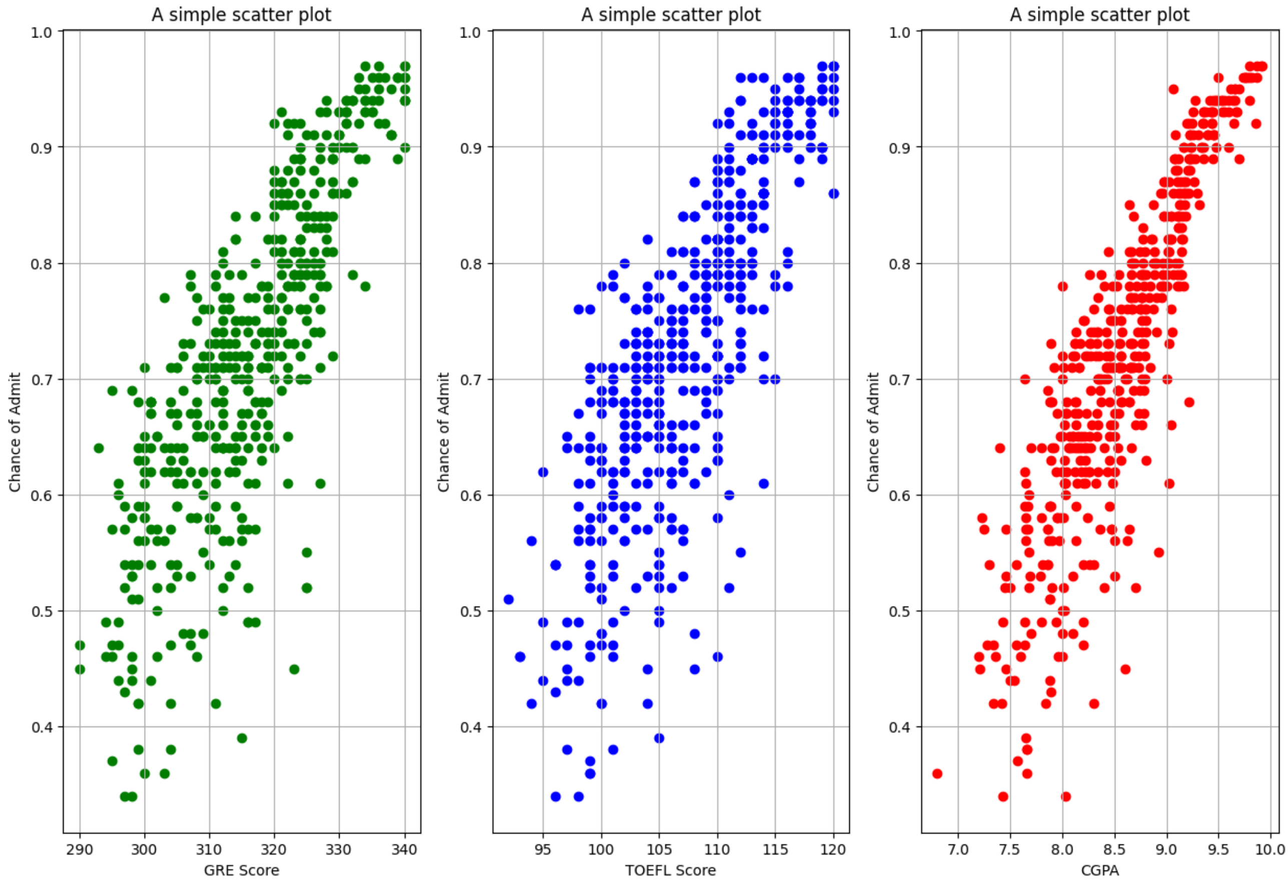
Bivariate Analysis (Relationships between important variables such as workday and count, season and count, weather and count.

```
sns.barplot(x="University Rating",y="Chance of Admit ",data=df,estimator=np.mean)
```

<Axes: xlabel='University Rating', ylabel='Chance of Admit '>



```
plt.subplot(1,3,1)
plt.scatter(x="GRE Score",y="Chance of Admit ",data=df,c='g')
plt.title('A simple scatter plot')
plt.xlabel('GRE Score')
plt.ylabel('Chance of Admit')
plt.grid()

plt.subplot(1,3,2)
plt.scatter(x="TOEFL Score",y="Chance of Admit ",data=df,c='b')
plt.title('A simple scatter plot')
plt.xlabel('TOEFL Score')
plt.ylabel('Chance of Admit')
plt.grid()

plt.subplot(1,3,3)
plt.scatter(x="CGPA",y="Chance of Admit ",data=df,c='r')
plt.title('A simple scatter plot')
plt.xlabel('CGPA')
plt.ylabel('Chance of Admit')
plt.grid()
```



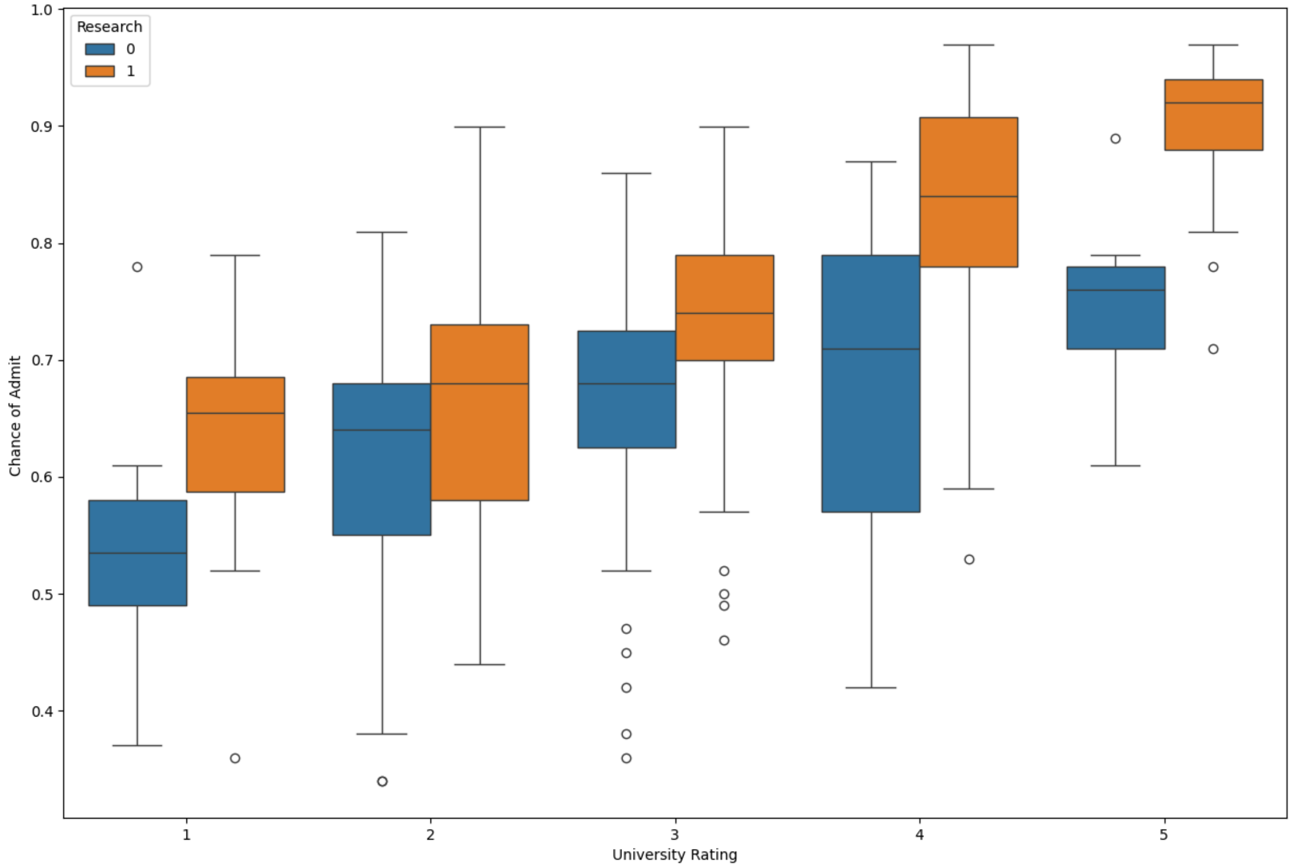
1 - GRE score and chance of admit is directly praportional with each other.

2 - TOEFL Score and chance of admit is directly praportional with each other.

3 - CGPA and chance of admit is directly praportional with each other.

```
sns.boxplot(x="University Rating",hue="Research",data=df,y="Chance of Admit ",dodge=True
```

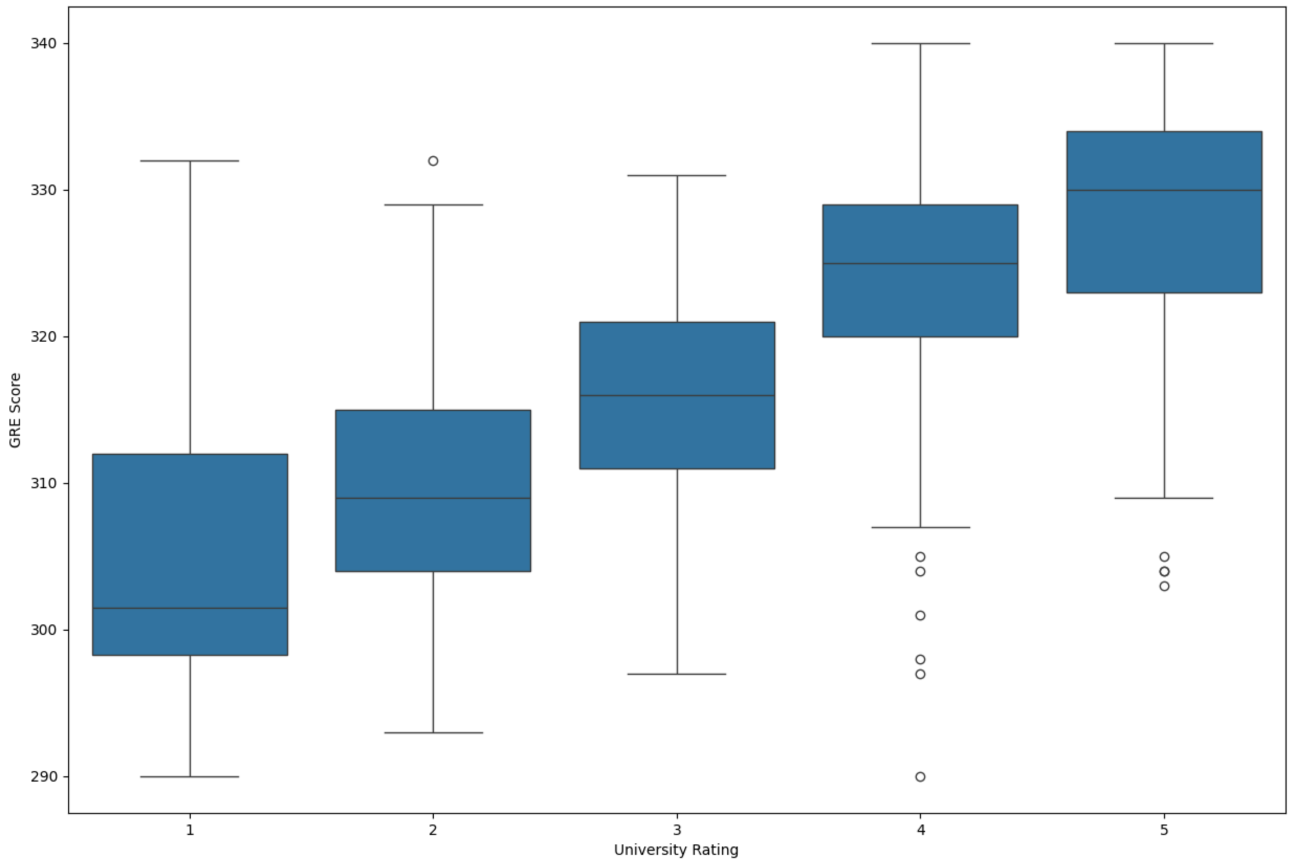
```
<Axes: xlabel='University Rating', ylabel='Chance of Admit '>
```



Applicant from university rating 4 with no research experience has more chances of admision

```
sns.boxplot(x="University Rating",data=df,y="GRE Score",dodge=True)
```

```
<Axes: xlabel='University Rating', ylabel='GRE Score'>
```



Data Preprocessing

```
bool_series = df.duplicated()
bool_series.value_counts()
```

```
False    500
dtype: int64
```

```
(df.isnull().sum()/len(df))*100
```

```
GRE Score            0.0
TOEFL Score          0.0
University Rating    0.0
SOP                  0.0
LOR                  0.0
CGPA                 0.0
Research             0.0
Chance of Admit      0.0
dtype: float64
```

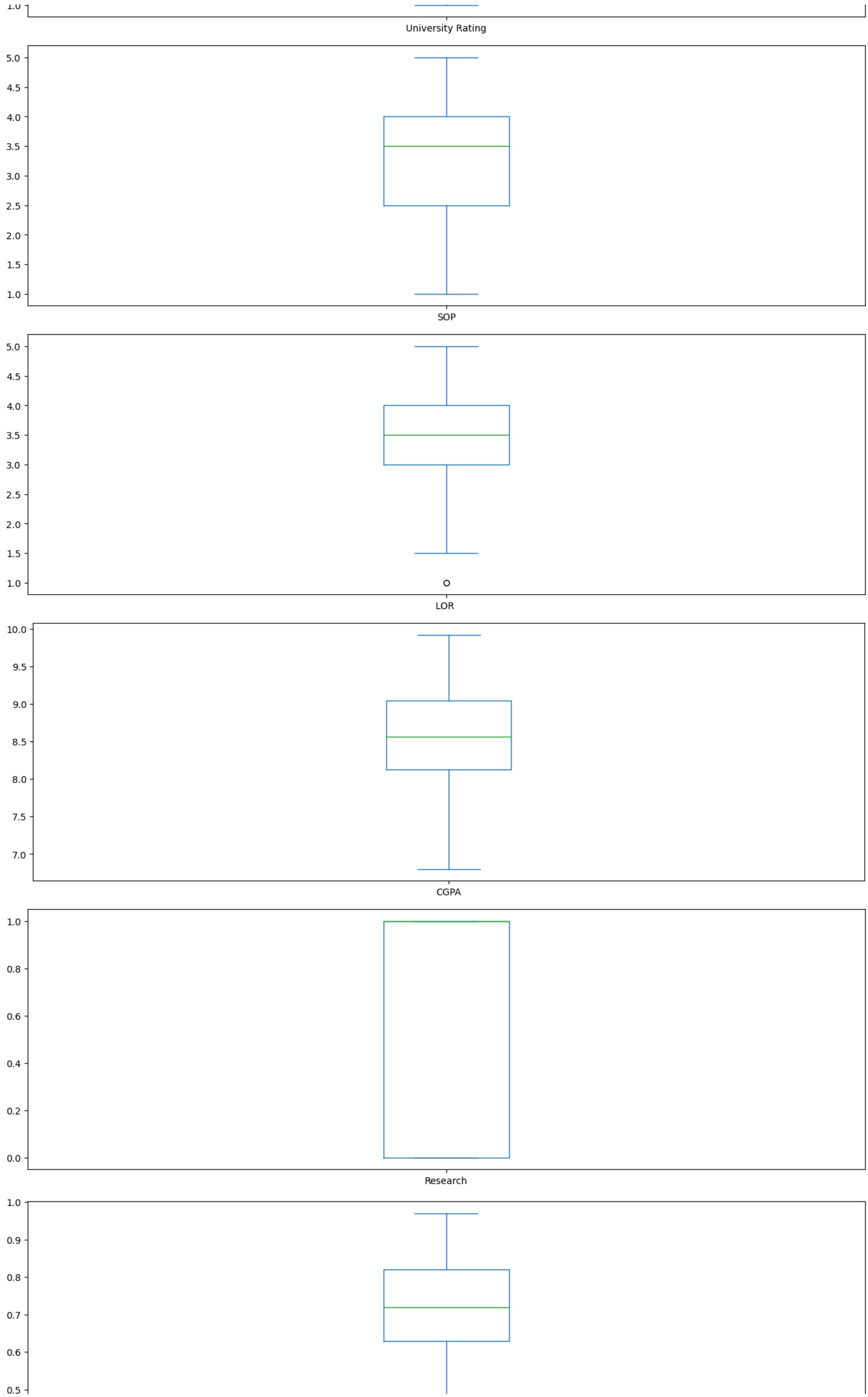
```
df.describe()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Researc
count	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.00000
mean	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.56000
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.49688
min	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.00000
25%	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.00000
50%	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.00000
75%	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.00000

```
df.columns
```

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

```
total_columns=['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA', '
for col in total_columns:
  df[col].plot.box(figsize=(16,5))
  plt.show()
```



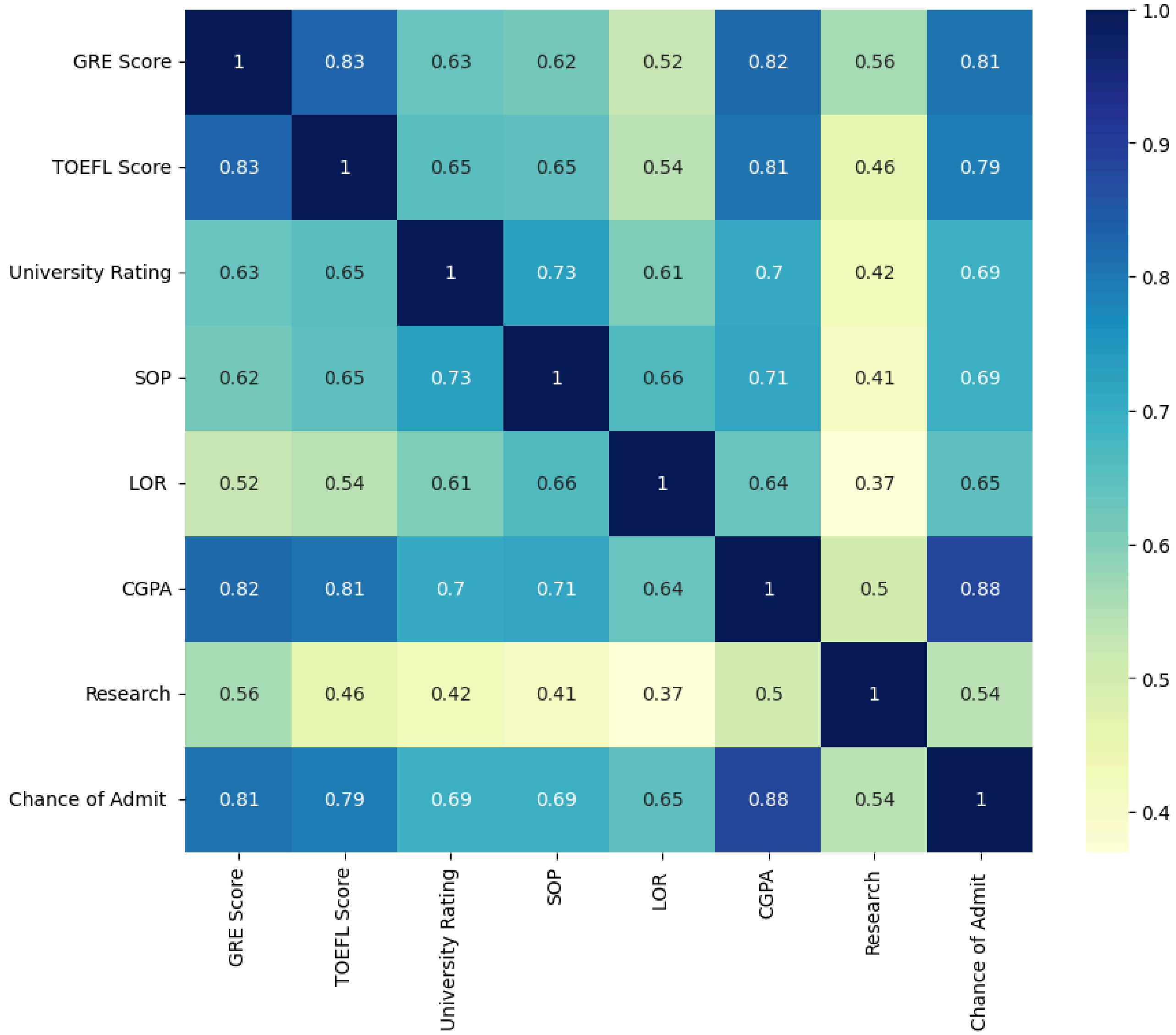
1.0
University Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
SOP
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
LOR
10.0
9.5
9.0
8.5
8.0
7.5
7.0
CGPA
1.0
0.8
0.6
0.4
0.2
0.0
Research
1.0
0.9
0.8
0.7
0.6
0.5

```
Q1=df['Chance of Admit '].quantile(0.25)
Q3=df['Chance of Admit '].quantile(0.75)
IQR=Q3-Q1
print(IQR)
lower_limit=Q1 - 1.5*IQR
Upper_limit=Q3 + 1.5*IQR
print(lower_limit,Upper_limit)
```

```
0.18999999999999995
0.3450000000000001 1.105
```

```
df=df[(df['Chance of Admit ']>lower_limit) & (df['Chance of Admit ']<Upper_limit)]
```

```
plt.figure(figsize=(10,8))
ax = sns.heatmap(df.corr(), cmap="YlGnBu", annot=True)
```




```
df=pd.read_csv("Jamboree_Admission.csv")

ratio_CGPA_GRE=(df["CGPA"]/df["GRE Score"])*100
df["ratio_CGPA_GRE"]=ratio_CGPA_GRE

ratio_CGPA_TOEFL=(df["CGPA"]/df["TOEFL Score"])*100
df["ratio_CGPA_TOEFL"]=ratio_CGPA_TOEFL
```

Combine SOP and LOR columns with name SOP_LOR_total

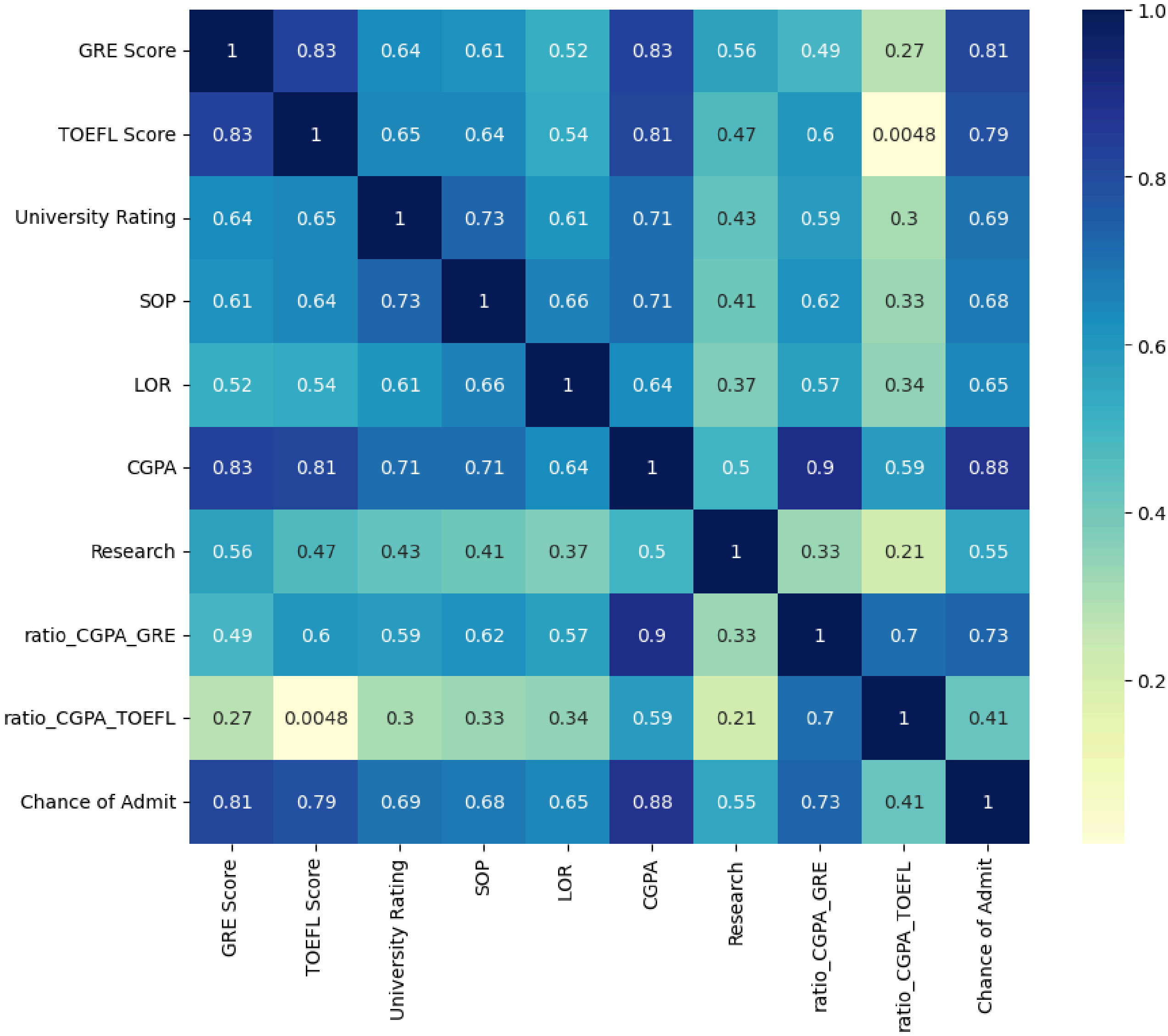
```
df["Chance of Admit"]=df["Chance of Admit "]
```

```
df_new=df.drop(columns=['Chance of Admit ',"Serial No."],axis=1)
df_new.head()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	ratio_CGPA_GRE	ratio_CGPA_TOE
0	337	118	4	4.5	4.5	9.65	1	2.863501	8.1779
1	324	107	4	4.0	4.5	8.87	1	2.737654	8.2897
2	316	104	3	3.0	3.5	8.00	1	2.531646	7.6923
3	322	110	3	3.5	2.5	8.67	1	2.692547	7.8818

Next steps: Generate code with df_new View recommended plots

```
plt.figure(figsize=(10,8))
ax = sns.heatmap(df_new.corr(), cmap="YlGnBu", annot=True)
```



GRE Score, TOEFL Score and CGPA are hightest correlated with chance of admit in same order.

New encoded features are strong predictor.

Still multicollinearity present in the data.

Data preparation for modeling

```
from sklearn.preprocessing import StandardScaler
```

```
df_new.columns
```

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

```
df_num=df_new[['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA','R
```

```
scaler = StandardScaler()
df_sc=scaler.fit_transform(df_num)
```

```
df_new_sc=pd.DataFrame(df_sc, columns=df_num.columns, index=df_num.index)
df_new_sc.head()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	ratio_CGF
0	1.819238	1.778865	0.775582	1.137360	1.098944	1.776806	0.886405	1.2
1	0.667148	-0.031601	0.775582	0.632315	1.098944	0.485859	0.886405	0.2
2	-0.041830	-0.525364	-0.099793	-0.377773	0.017306	-0.954043	0.886405	-1.4
3	0.489904	0.462163	-0.099793	0.127271	-1.064332	0.154847	0.886405	-0.1

Next steps: Generate code with df_new_sc   View recommended plots

```
df_new1=pd.concat([df_new_sc,df_new["Chance of Admit"]],axis=1)
```

```
df_new1.head()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	ratio_CGF
0	1.819238	1.778865	0.775582	1.137360	1.098944	1.776806	0.886405	1.2
1	0.667148	-0.031601	0.775582	0.632315	1.098944	0.485859	0.886405	0.2
2	-0.041830	-0.525364	-0.099793	-0.377773	0.017306	-0.954043	0.886405	-1.4
3	0.489904	0.462163	-0.099793	0.127271	-1.064332	0.154847	0.886405	-0.1

Next steps: Generate code with df_new1   View recommended plots

```
df_new1.shape
```

```
(500, 10)
```

Model building

```
x = df_new1["CGPA"].values
y = df_new1["Chance of Admit"].values
```

```
def hypothesis(x,weights):
  y_hat=weights[0]+ weights[1]*x
  return y_hat
```

```
hypothesis(2.3,[5,0.8])
```

```
6.84
```

```
def error(x,y,weights):
  n= len(x)
  err=0
  for i in range(n):
    y_hat_i=hypothesis(x[i],weights)
    err=err+(y[i] - y_hat_i)**2
  return err/n
```

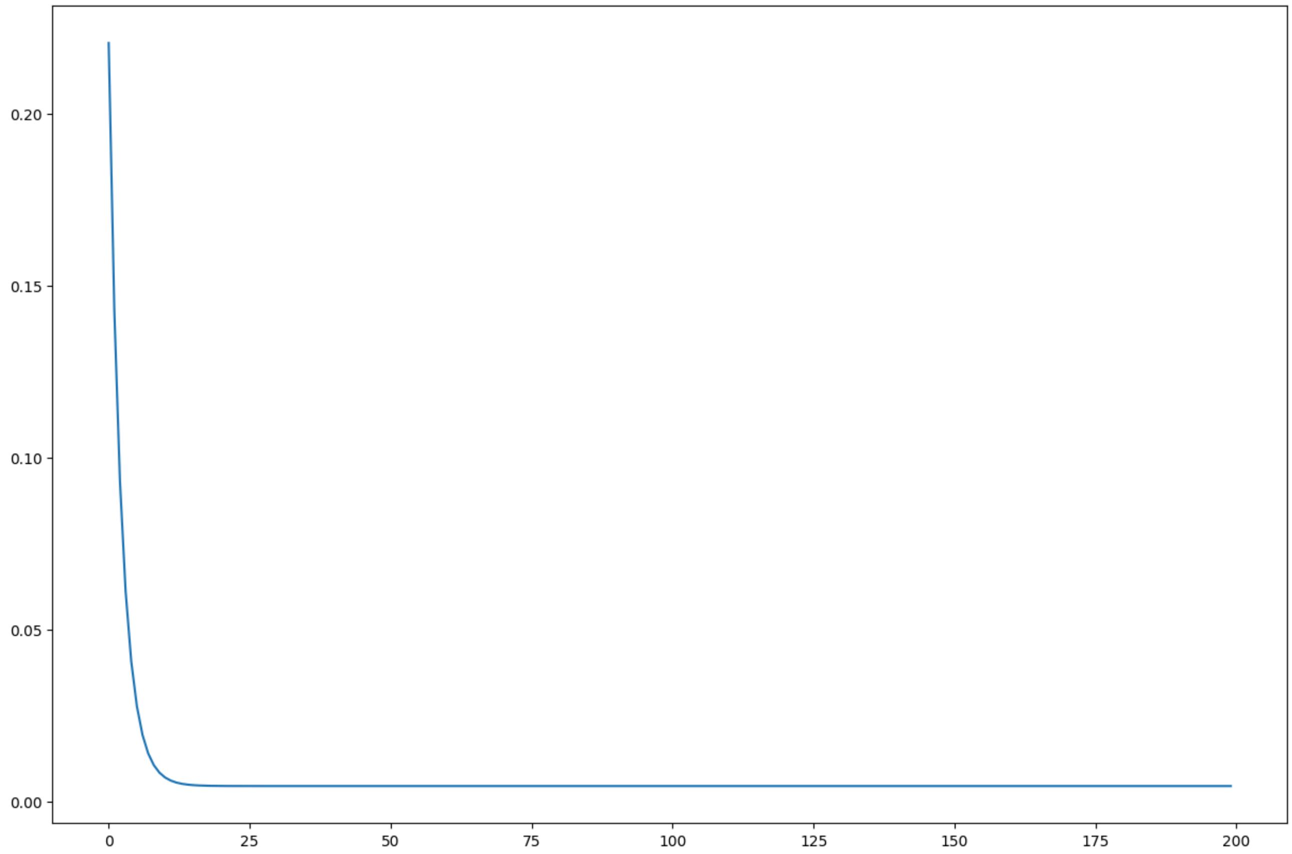
```
def gradient(x,y,weights):
    n=len(x)
    grade= np.zeros((2, ))
    for i in range(n):
        y_hat_i=hypothesis(x[i],weights)
        grade[0] += (y_hat_i - y[i])
        grade[1] += (y_hat_i - y[i])*x[i]
    return (2*grade)/n
```

```
def gradient_descent(x,y,ran_itr=200,learning_rate=0.1):
    '''step1: initialise the variable '''
    weights=np.random.rand(2)
    ''' step2: rpeate for 100 times'''
    error_list=[]
    for i in range(ran_itr):
        e=error(x,y,weights)
        error_list.append(e)
        grade = gradient(x,y,weights)
        weights[0]=weights[0]-learning_rate*grade[0]
        weights[1]=weights[1]-learning_rate*grade[1]

    return weights.round(3),error_list
```

```
opt_weights, error_list=gradient_descent(x,y)
```

```
plt.plot(error_list)
```

```
[<matplotlib.lines.Line2D at 0x7995eab5aa10>]
```




```
Y_hat=hypothesis(x,opt_weights)
```

```
def r2_score(Y, Y_hat):
    num = np.sum((Y - Y_hat)**2)
    denom = np.sum((Y - Y.mean())**2)

    r2 = 1 - num/denom

    return r2.round(3)


r2_score(y,Y_hat)
```

```
0.779
```

Performance of the simple linear regression model using CGPA veriable is 78%

Only CGPA is not important to check the chanse od admit hence let's check multivarient linear regression

Building the Linear Regression model and commenting on the model statistics and model coefficients with column names

```
df_new1.head()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	ratio_CGF
0	1.819238	1.778865	0.775582	1.137360	1.098944	1.776806	0.886405	1.2
1	0.667148	-0.031601	0.775582	0.632315	1.098944	0.485859	0.886405	0.2
2	-0.041830	-0.525364	-0.099793	-0.377773	0.017306	-0.954043	0.886405	-1.4
3	0.489904	0.462163	-0.099793	0.127271	-1.064332	0.154847	0.886405	-0.1

Next steps: Generate code with df_new1 View recommended plots

```
import statsmodels.api as sm


X = df_new1[df_new1.columns.drop('Chance of Admit')]
Y = df_new1["Chance of Admit"]


X_sm = sm.add_constant(X)  #Statmodels default is without intercept, to add intercept we

sm_model = sm.OLS(Y, X_sm).fit()

print(sm_model.summary())
```

```
                            OLS Regression Results
==============================================================================
```

```
Dep. Variable:       Chance of Admit   R-squared:                       0.823
Model:                           OLS   Adj. R-squared:                  0.819
Method:                Least Squares   F-statistic:                     252.5
Date:               Sun, 17 Mar 2024   Prob (F-statistic):          1.02e-177
Time:                       11:43:18   Log-Likelihood:                 702.37
No. Observations:                500   AIC:                            -1385.
Df Residuals:                    490   BIC:                            -1343.
Df Model:                          9
Covariance Type:           nonrobust
====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975
------------------------------------------------------------------------------------
const                 0.7217      0.003    269.021      0.000       0.716       0.72
GRE Score             0.1270      0.079      1.607      0.109      -0.028       0.28
TOEFL Score          -0.0343      0.089     -0.386      0.700      -0.209       0.14
University Rating     0.0067      0.004      1.538      0.125      -0.002       0.01
SOP                   0.0014      0.005      0.316      0.752      -0.007       0.01
LOR                   0.0156      0.004      4.066      0.000       0.008       0.02
CGPA                 -0.0739      0.121     -0.611      0.542      -0.312       0.16
Research              0.0123      0.003      3.736      0.000       0.006       0.01
ratio_CGPA_GRE        0.1357      0.101      1.345      0.179      -0.062       0.33
ratio_CGPA_TOEFL     -0.0377      0.065     -0.583      0.560      -0.165       0.08
==============================================================================
Omnibus:                     118.043   Durbin-Watson:                   0.804
Prob(Omnibus):                 0.000   Jarque-Bera (JB):              283.371
Skew:                         -1.198   Prob(JB):                     2.93e-62
Kurtosis:                      5.803   Cond. No.                         148.
==============================================================================

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

Linear Regression model

```
X = df_new1[df_new1.columns.drop('Chance of Admit')]
Y = df_new1["Chance of Admit"]

#Train and test data split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=10

from sklearn.linear_model import LinearRegression
lr = LinearRegression()

# train the model
lr.fit(X_train, y_train)
Pred = lr.predict(X_test)
from sklearn.metrics import r2_score,mean_squared_error, mean_absolute_error
print("Linear Regression R2_score :",r2_score(y_test, Pred))
```

```
Linear Regression R2_score : 0.8313554590045338
```

```
lr.coef_
```

```
array([ 0.07362709,  0.03097081,  0.00572688, -0.00115692,  0.01779132,
       -0.05144605,  0.01351941,  0.07205956,  0.00848471])
```

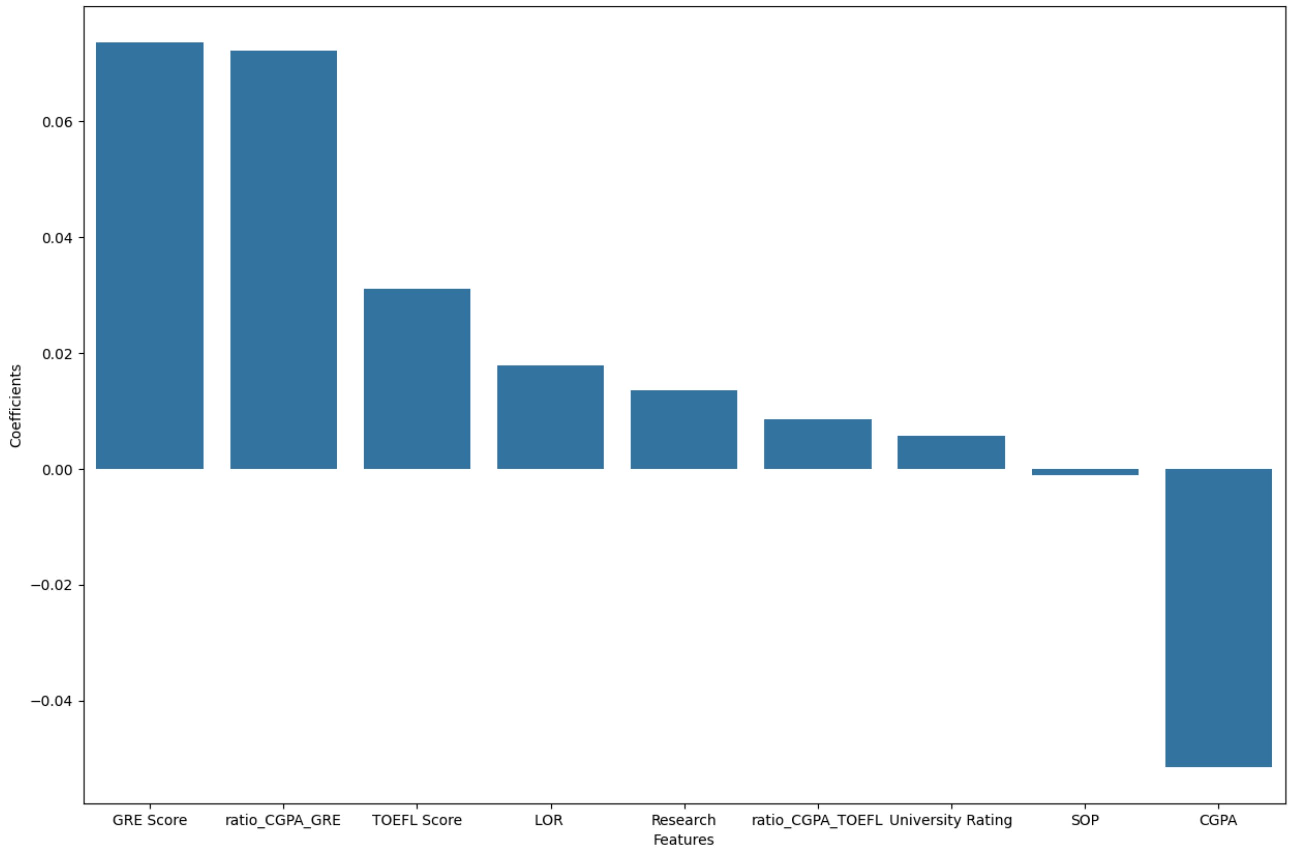
```
coeff=pd.DataFrame()                                        # GRE score has highest weight tl
X_c=X                                                       # 3rd highest weight is on CGPA :
coeff["Features"]=X_c.columns
coeff["Coefficients"]=lr.coef_
coeff["Coefficients"] = round(coeff["Coefficients"], 5)
coeff = coeff.sort_values(by = "Coefficients", ascending = False)
coeff
```

	Features	Coefficients
0	GRE Score	0.07363
7	ratio_CGPA_GRE	0.07206
1	TOEFL Score	0.03097
4	LOR	0.01779
6	Research	0.01352
8	ratio_CGPA_TOEFL	0.00848
2	University Rating	0.00573
3	SOP	-0.00116
5	CGPA	-0.05145

Next steps: Generate code with coeff | View recommended plots

```
sns.barplot(x="Features",y="Coefficients",data=coeff)
```

```
<Axes: xlabel='Features', ylabel='Coefficients'>
```



Lasso regression using sklearn

```
from sklearn.linear_model import Lasso

X = df_new1[df_new1.columns.drop('Chance of Admit')]
Y = df_new1["Chance of Admit"]

#Train and test data split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=10


# initialize Lasso regression and set the value of alpha equal to 1
ls = Lasso(alpha= 1)

# fit the model
ls.fit(X_train,y_train)

#predict
ls_pred=ls.predict(X_test)
#r2_score
lasso_r2_score=r2_score(y_test, ls_pred)

#print intercepts and coefficients rounded off upto 2 decimal digit
print("Coefficients:",list(zip(X.columns, ls.coef_)))
print("Intercepts:",ls.intercept_.round(2))
print("LASSO R2_score:",lasso_r2_score)
```

```
Coefficients: [('GRE Score', 0.0), ('TOEFL Score', 0.0), ('University Rating', 0.0),
Intercepts: 0.72
LASSO R2_score: -0.0424956830527512
```

In this data set all feature are important there is no as such less important feature hence we can not make all the features equal to zero as it has some multicolinearity but we can not remove it by lasso regression. Hence we can canclude that lasso regression is not suitable for this dataset.

Ridge regression using sklearn

```
from sklearn.linear_model import Ridge

rd=Ridge()
rd.fit(X_train, y_train)

#predict
rd_pred=ls.predict(X_test)
#r2_score
ridge_r2_score=r2_score(y_test, rd_pred)

#print intercepts and coefficients rounded off upto 2 decimal digit
print("Coefficients:",list(zip(X.columns, rd.coef_)))
print("Intercepts:",rd.intercept_.round(2))
print("Ridge R2_score:",ridge_r2_score.round(5))
```

```
Coefficients: [('GRE Score', 0.03614361074336035), ('TOEFL Score', 0.032593241490360
Intercepts: 0.72
Ridge R2_score: -0.0425
```

Same with the ridge regression there is no need to regularise the model as each feature has it's own importance and without making it zero or moving it toward zero we can build the linear regression model with zero mean_square_error value and r2 score upto 0.8+

Testing the assumptions of the linear regression model

Multicollinearity check by VIF score

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

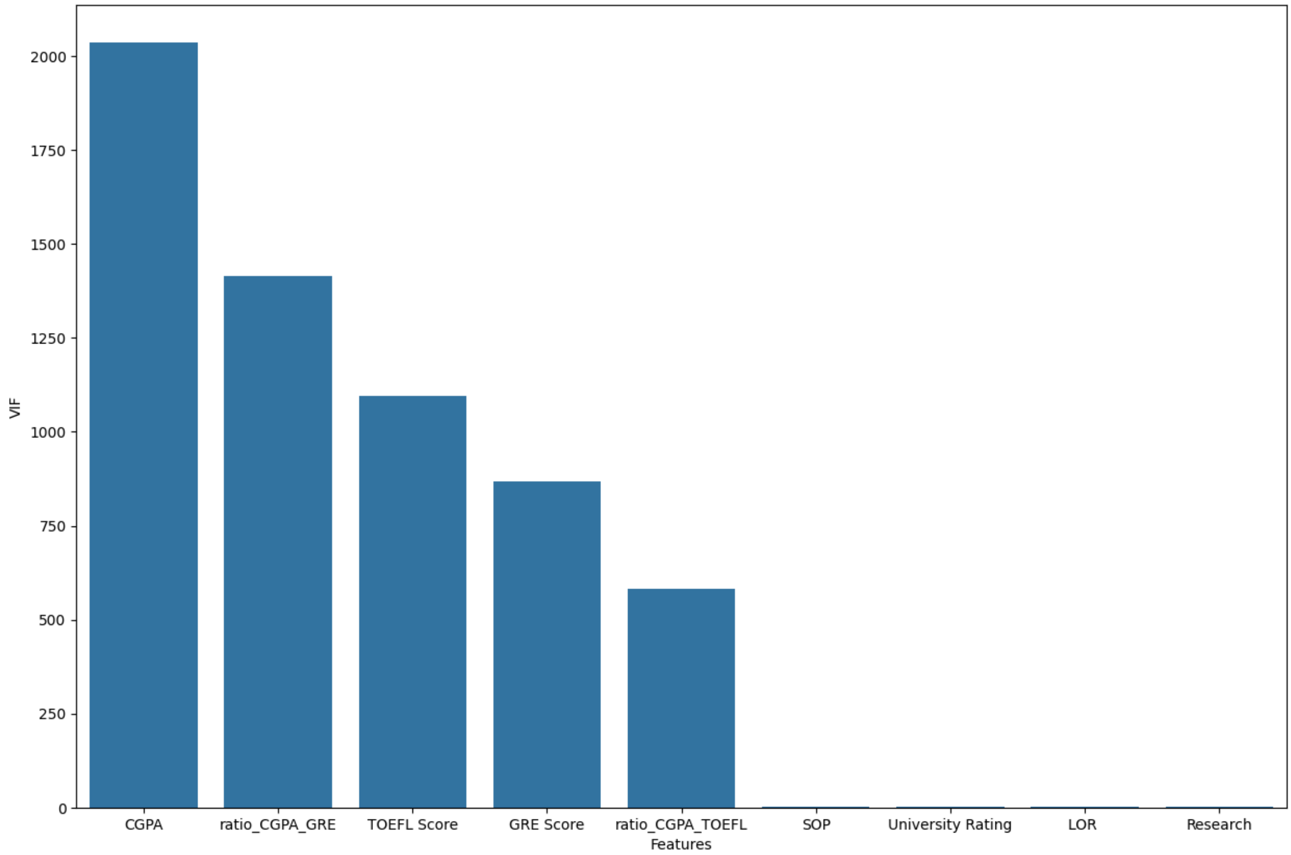
```
vif = pd.DataFrame()
X_t = X
vif['Features'] = X_t.columns
vif['VIF'] = [variance_inflation_factor(X_t.values, i) for i in range(X_t.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

	Features	VIF
5	CGPA	2036.53
7	ratio_CGPA_GRE	1413.88
1	TOEFL Score	1095.53
0	GRE Score	867.55
8	ratio_CGPA_TOEFL	580.79
3	SOP	2.84
2	University Rating	2.67
4	LOR	2.04
6	Research	1.50

Next steps: Generate code with `vif` View recommended plots

```
sns.barplot(x="Features",y="VIF",data=vif)
```

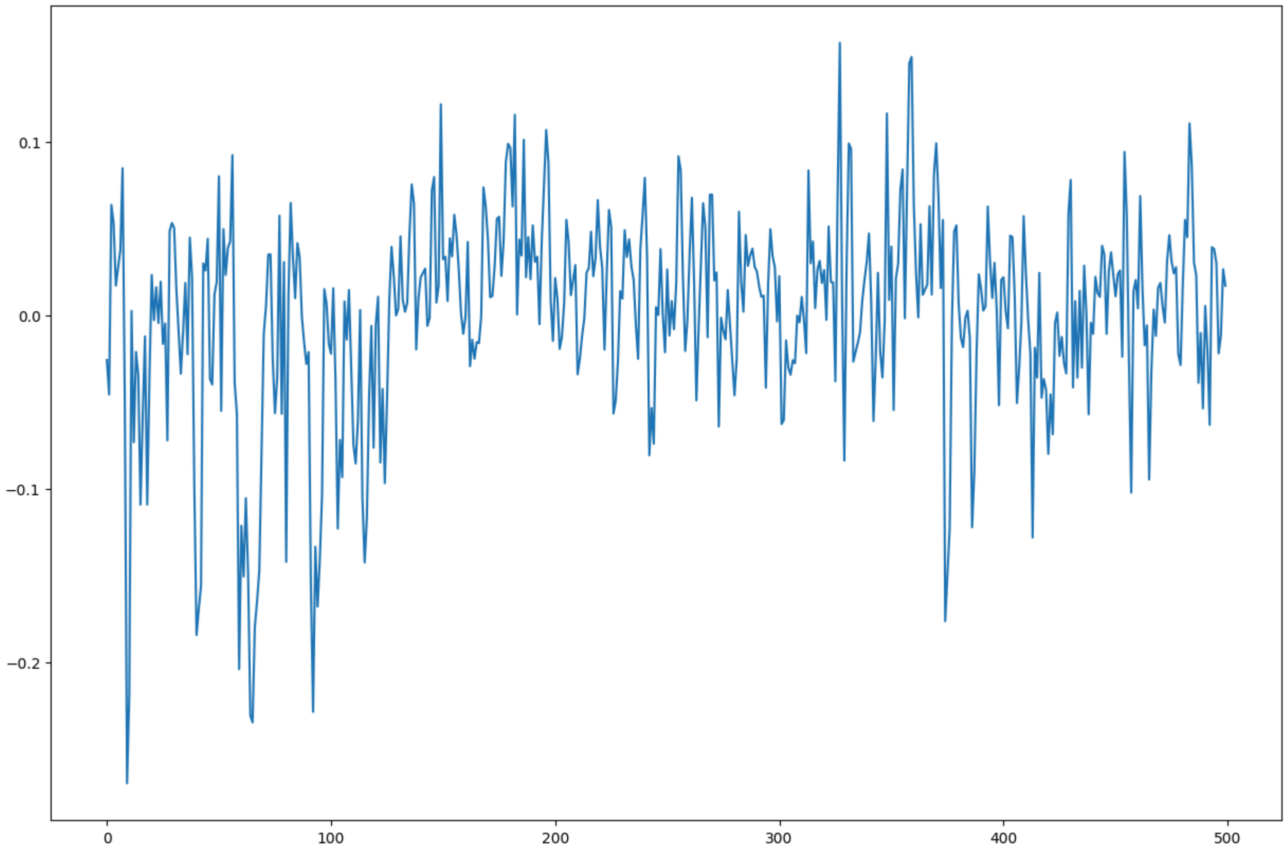
<Axes: xlabel='Features', ylabel='VIF'>



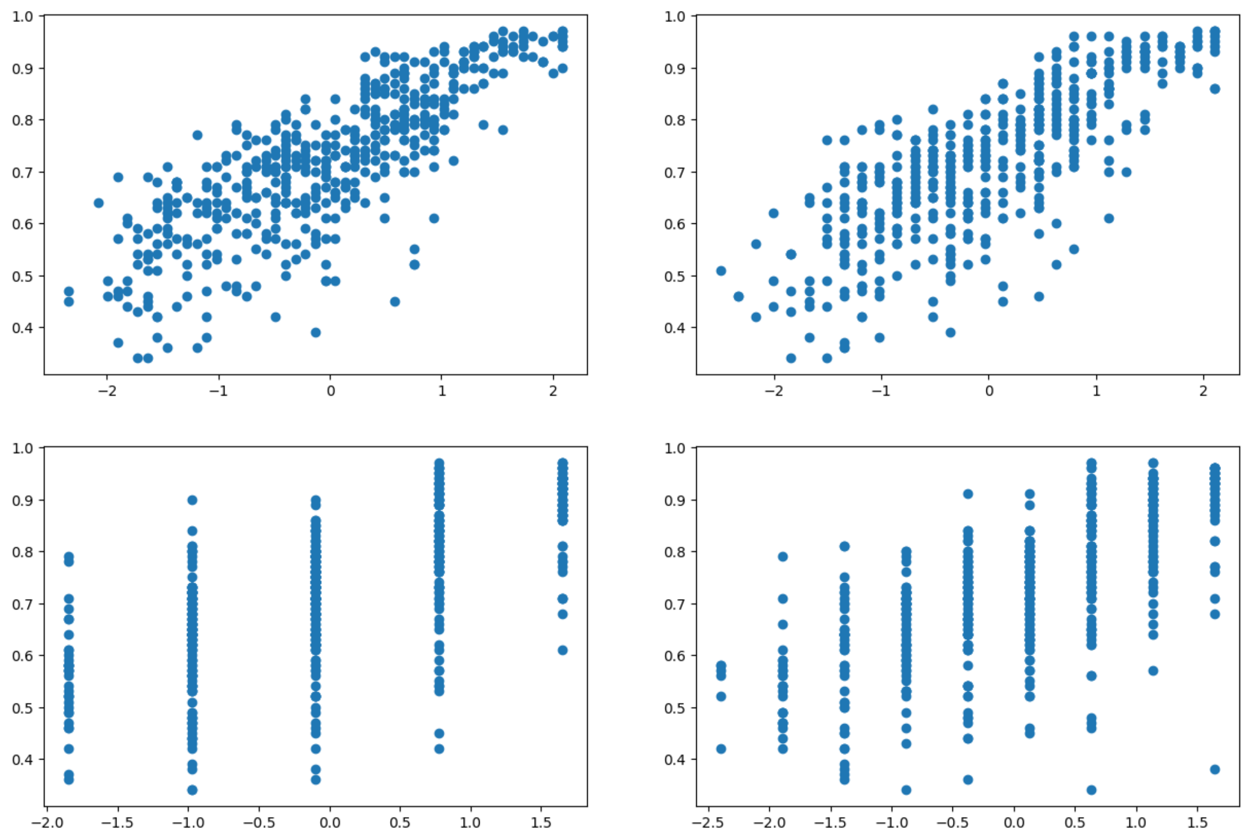
The mean of residuals is nearly zero

```
residuals=sm_model.resid
plt.plot(residuals.index,residuals)
```

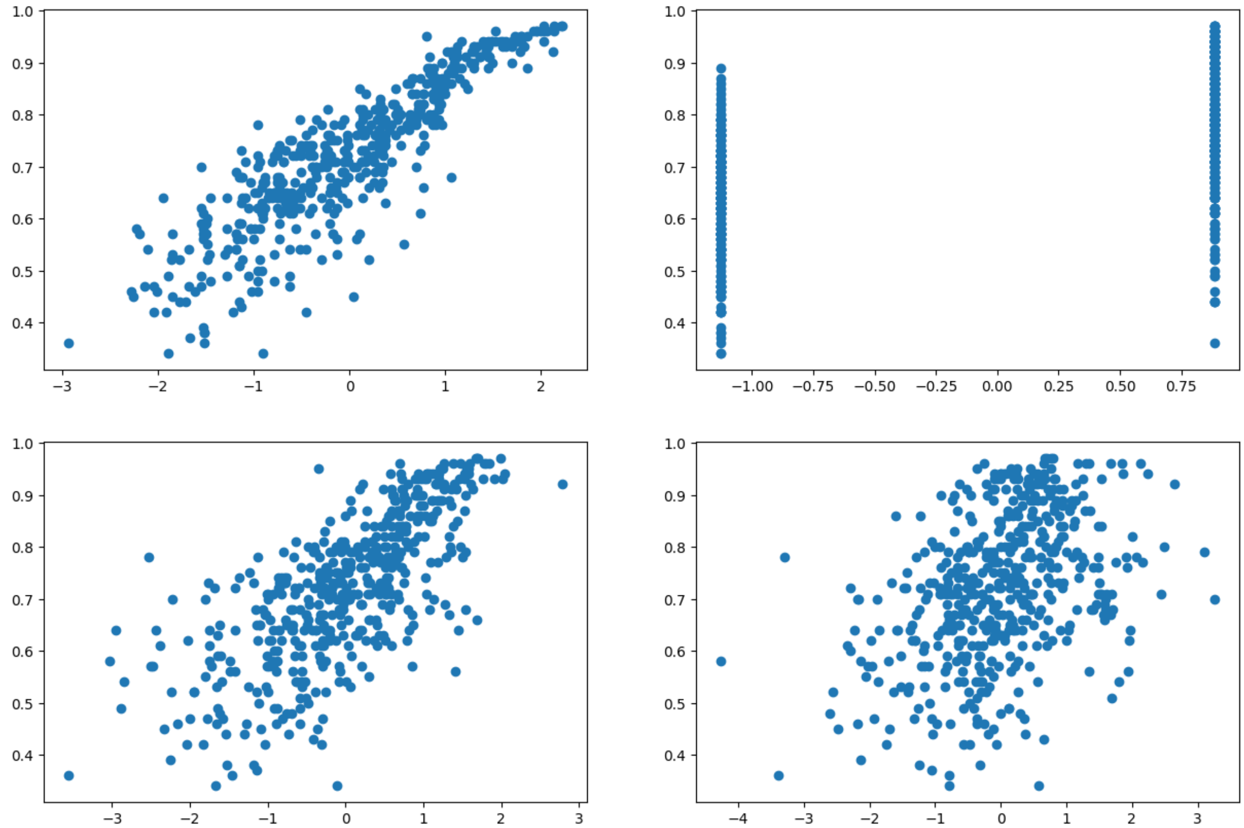
```
[<matplotlib.lines.Line2D at 0x7995ea92d2a0>]
```



Linearity of variables



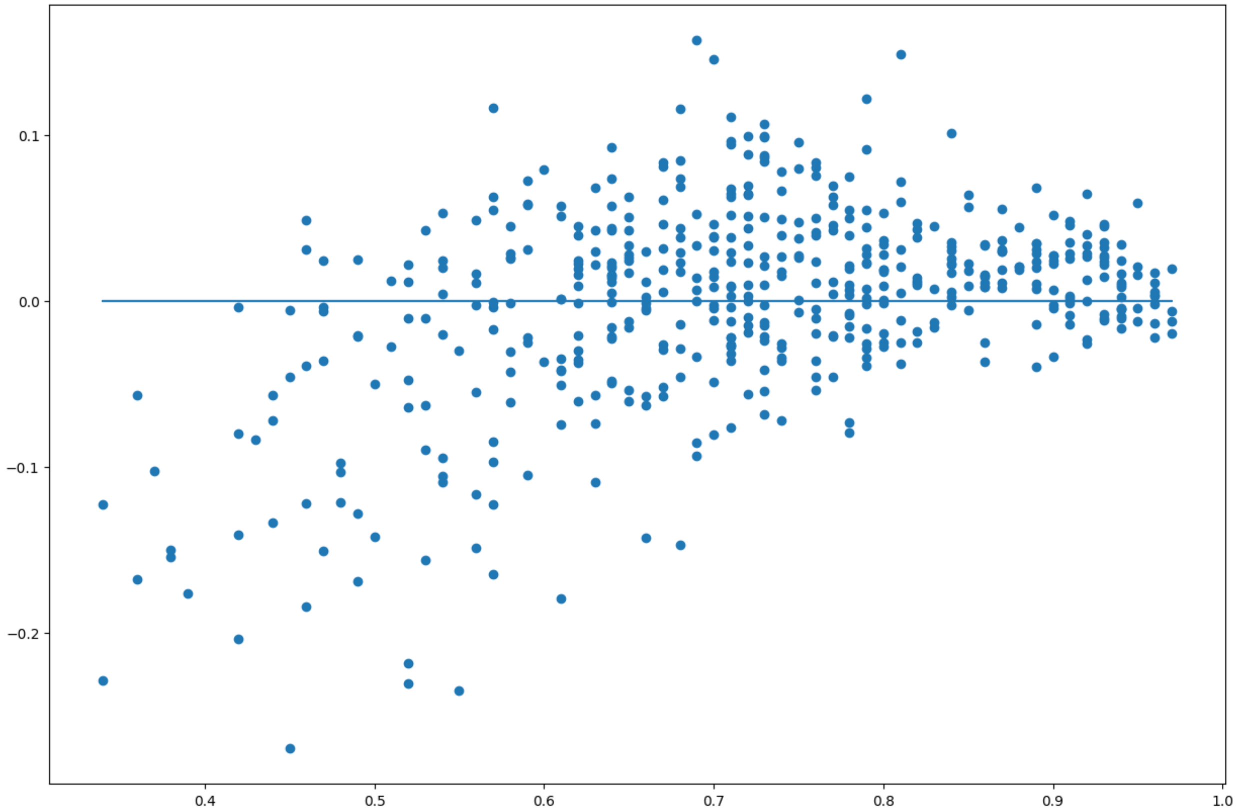

```
from mpl_toolkits.mplot3d import axes3d
plt.rcParams["figure.figsize"]=(15,10)
fig,((ax1,ax2),(ax3,ax4))=plt.subplots(2,2)
ax1.scatter(X["CGPA"],Y)
ax2.scatter(X["Research"],Y)
ax3.scatter(X["ratio_CGPA_GRE"],Y)
ax4.scatter(X["ratio_CGPA_TOEFL"],Y)
plt.show()
```



Test for Homoscedasticity

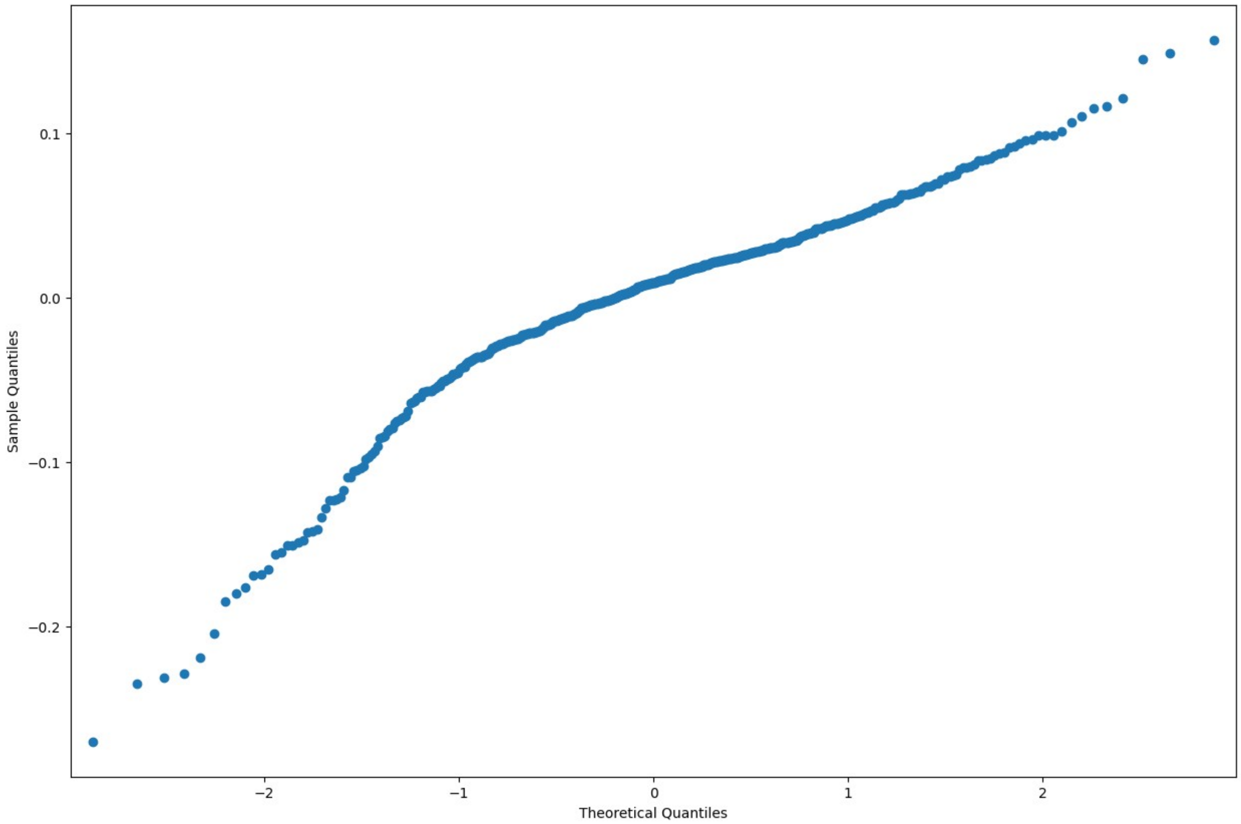
```
residuals=sm_model.resid
plt.scatter(Y,residuals)
plt.plot(Y,[0]*len(Y))
```

```
[<matplotlib.lines.Line2D at 0x7995ec91b910>]
```



Normality of residuals

```
residuals=sm_model.resid
sm.qqplot(residuals)
plt.show()
```

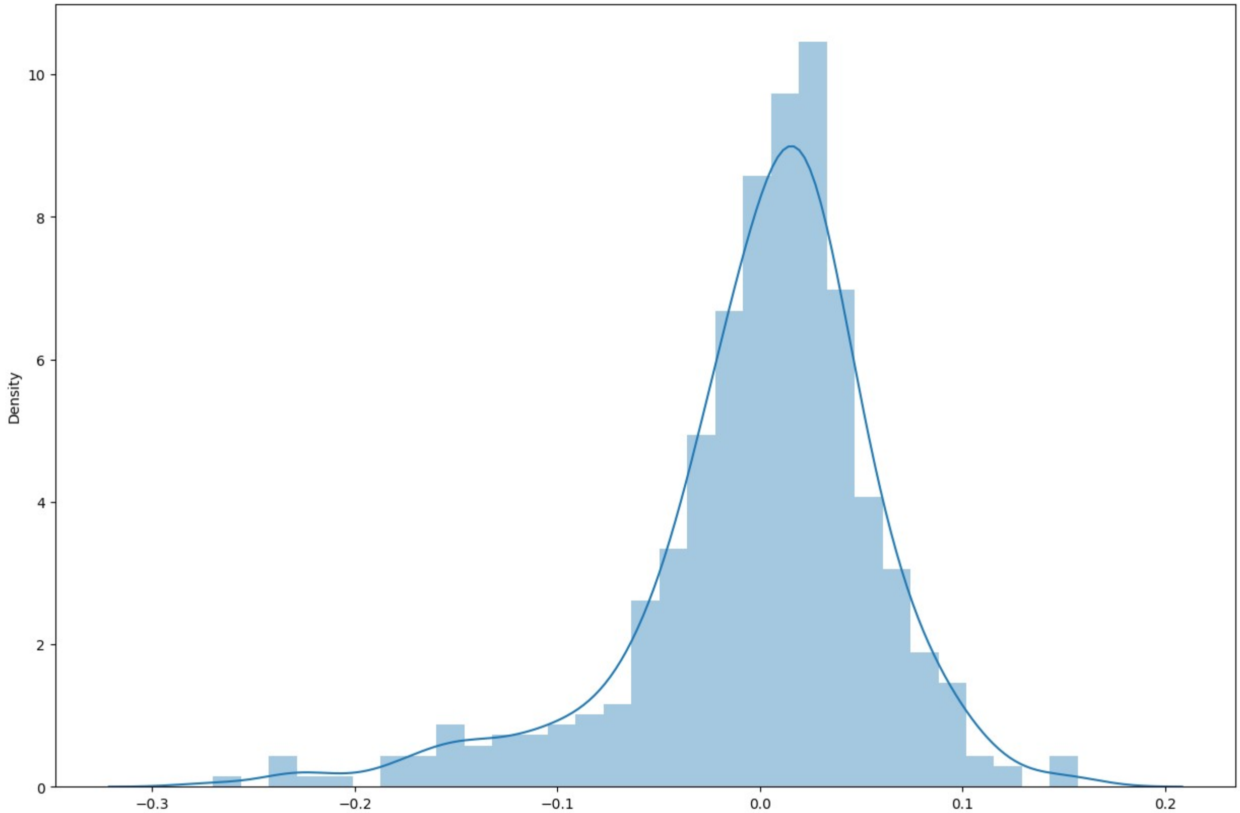



```
np.mean(residuals)
```

```
1.0746958878371515e-16
```

```
sns.distplot(residuals)
```

```
<Axes: ylabel='Density'>
```



Model performance evaluation

Metrics checked - MAE, RMSE, R2, Adj R2

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

predict=lr.predict(X_test)

MSE=mean_squared_error(y_test, predict)
print("Mean_absolute_error=",mean_absolute_error(y_test,
predict).round(3))
print("Root_mean_squared_error=",np.sqrt(MSE).round(3))
print("R2_score=",r2_score(y_test, predict).round(3))

Mean_absolute_error= 0.044
Root_mean_squared_error= 0.057
R2_score= 0.831

Adj_r2_score=1 - (1-lr.score(X, Y))*(len(Y)-1)/(len(Y)-X.shape[1]-1)
print("Adjusted R2 score=",np.round(Adj_r2_score,3))

Adjusted R2 score= 0.818
```

- Mean_absolute_error(MAE) is 0.044
- Root_mean_squared_error(RMSE) is 0.057

- R2_score(R2) is 0.831
- Adjusted R2 score(Adj R2) is 0.818

Train and test performance

```
predict_train=lr.predict(X_train)
predict_test=lr.predict(X_test)

print("r2_score of train data=",r2_score(y_train,
predict_train).round(3))
print("r2_score of test data=",r2_score(y_test,
predict_test).round(3))
print()
print("mean_squared_error of train data=",mean_squared_error(y_train,
predict_train).round(3))
print("mean_squared_error of test data=",mean_squared_error(y_test,
predict_test).round(3))
print()
print("mean_absolute_error of train
data=",mean_absolute_error(y_train, predict_train).round(3))
print("mean_absolute_error of test data=",mean_absolute_error(y_test,
predict_test).round(3))
```

```
r2_score of train data= 0.818
r2_score of test data= 0.831

mean_squared_error of train data= 0.004
mean_squared_error of test data= 0.003

mean_absolute_error of train data= 0.043
mean_absolute_error of test data= 0.044
```

R2 score of train data and test data is almost same there is only the difference of 0.013

A value of 0.8 for R-square score sounds good. It means linear regression model is performing pretty good.

Mean square error and mean absolute error is almost zero it means that model is pefectly build.

linear regression model is performing very well on the unseen data which is test data.

Actionable Insights & Recommendations :

1. The website has implemented a linear regression model or a feature allowing students/learners to assess their likelihood of gaining admission to an Ivy League college. This model yields an 81% accuracy rate in predicting admission probabilities.

2. This functionality serves to attract a broader audience of students/learners, providing Jamboree with valuable demographic insights for marketing purposes.

3. Utilizing this model, Jamboree can identify students/learners with lower admission probabilities and offer tailored coaching services to assist them in gaining admission to their desired universities. This aspect holds significant value from a business standpoint.

4. A suggested enhancement for data collection involves incorporating an additional column for city or region names. This enhancement would facilitate targeting specific audience segments from those regions, allowing for tailored marketing strategies.