Jamboree Education - Linear Regression

By: PARTH PATEL

Problem Statement

Jamboree Education seeks to develop a linear regression model to predict the probability of students getting admitted to Ivy League colleges. The primary objectives are to identify significant predictors, evaluate the model's performance, and derive actionable insights for business growth and targeted marketing.

```
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(10)
```

	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
5	6	330	115	5	4.5	3.0	9.34	1	0.90
6	7	321	109	3	3.0	4.0	8.20	1	0.75
7	8	308	101	2	3.0	4.0	7.90	0	0.68
8	9	302	102	1	2.0	1.5	8.00	0	0.50

1. Exploratory Data Analysis

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

```
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

```
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() # No missing values good to go
```

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

```
df.info() # all datatypes are correctly identified good to go
```

```
<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
```

```
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	0.5600
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.4968
min	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.0000
25%	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.0000
50%	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.0000
75%	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.0000

Univariate Analysis

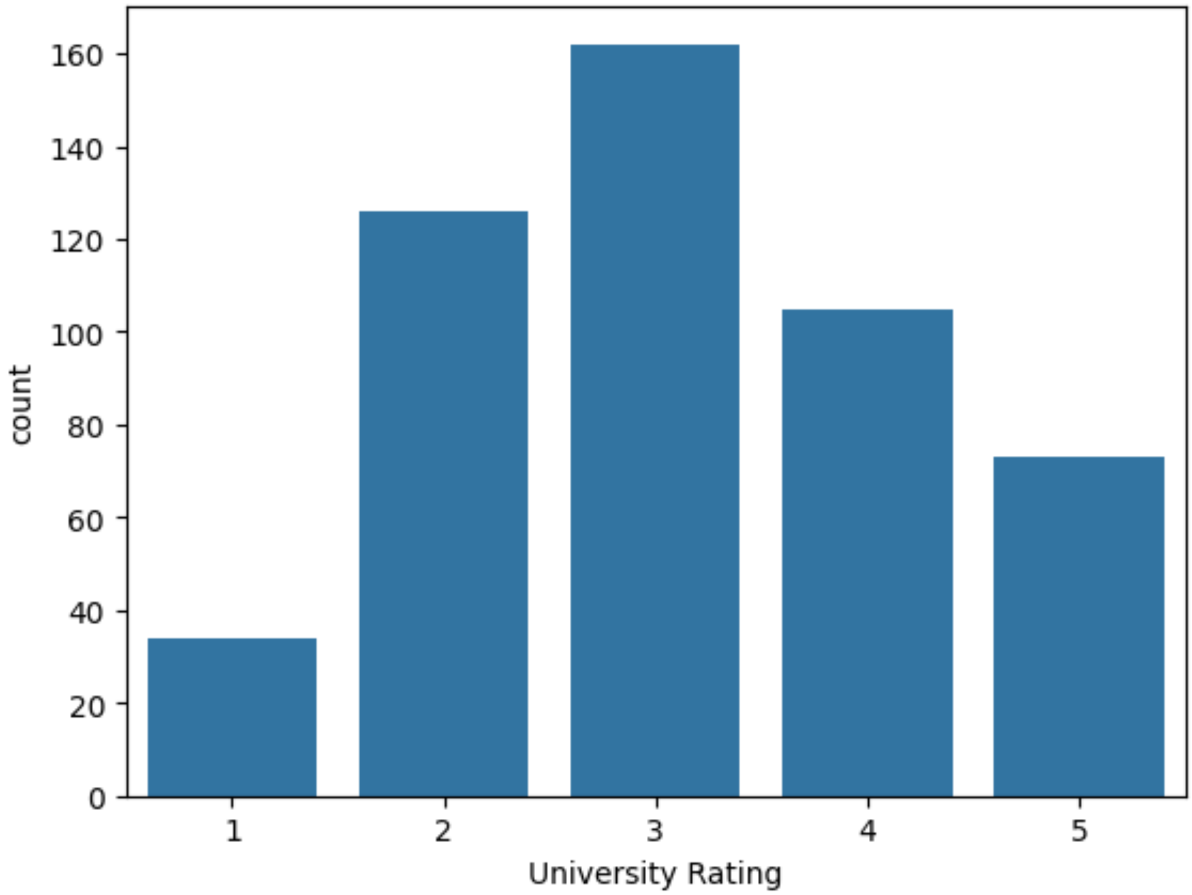
```
df["University Rating"].unique()
```

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

```
df["University Rating"].value_counts() # maximum applicant are from university ra
```

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

```
sns.countplot(data=df,x="University Rating") # Visual representation of the above
plt.show()
```



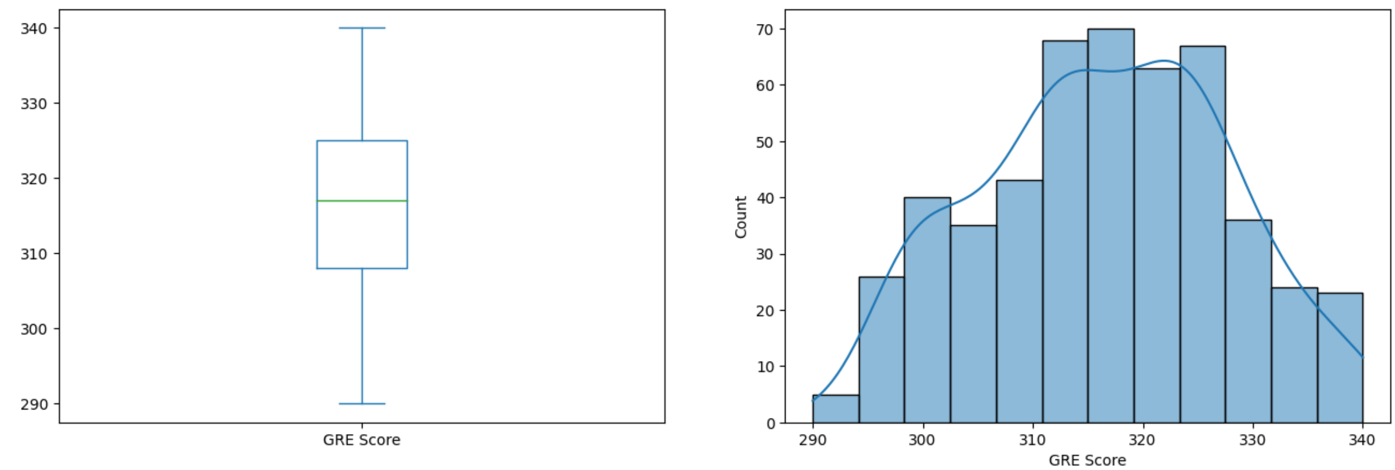

```
df["GRE Score"].unique()
```

```
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) # maximum applicant score in GRE Score is li
```

```
GRE Score
(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: count, dtype: int64
```

```
plt.subplot(121)
df["GRE Score"].plot.box(figsize=(16,5))    # Median is at 317
plt.subplot(122)                            # GRE Score data is  normaly distribu
sns.histplot(df["GRE Score"], kde=True)     # no outliers present
plt.show()
```



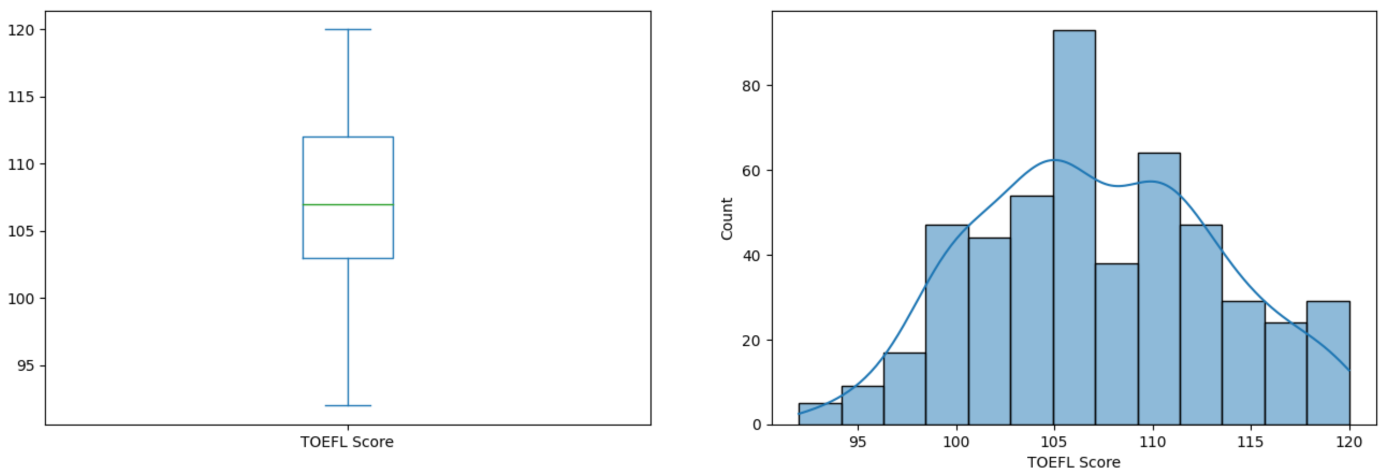

```
df["TOEFL Score"].unique()
```

```
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) # maximum applicant score in TOEFL Score i
```

```
TOEFL Score
(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: count, dtype: int64
```

```
plt.subplot(121)
df["TOEFL Score"].plot.box(figsize=(16,5))    # Median is at 107
plt.subplot(122)                             # TOEFL Score data is  normaly distri
sns.histplot(df["TOEFL Score"], kde=True)     # no outliers present
plt.show()
```



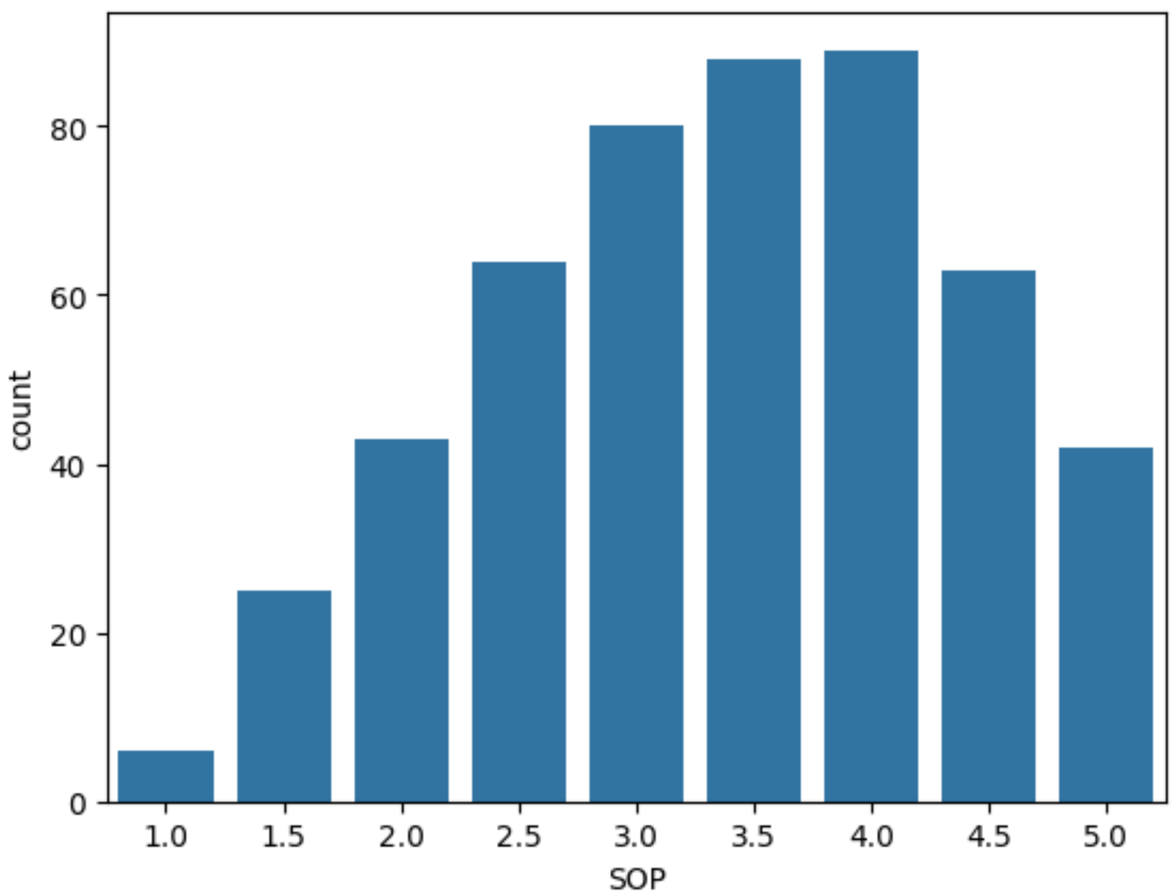
```
df["SOP"].unique()
```

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

```
df["SOP"].value_counts(bins=2) # Maximum applicants Statement of Purpose and Lett
```

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

```
sns.countplot(data=df,x="SOP") # Visual representation of the above code
plt.show()
```



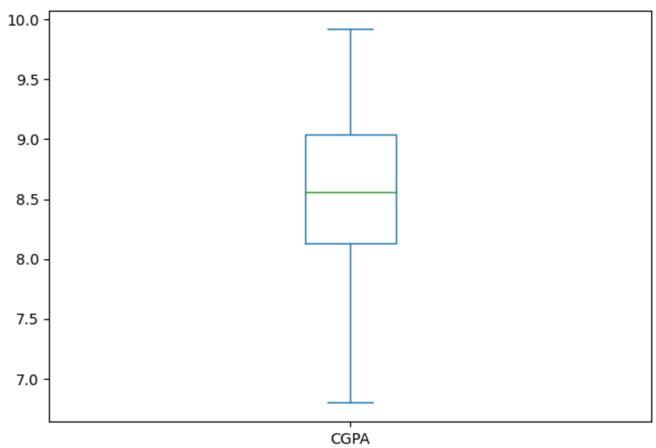

```
df["CGPA"].nunique()
```

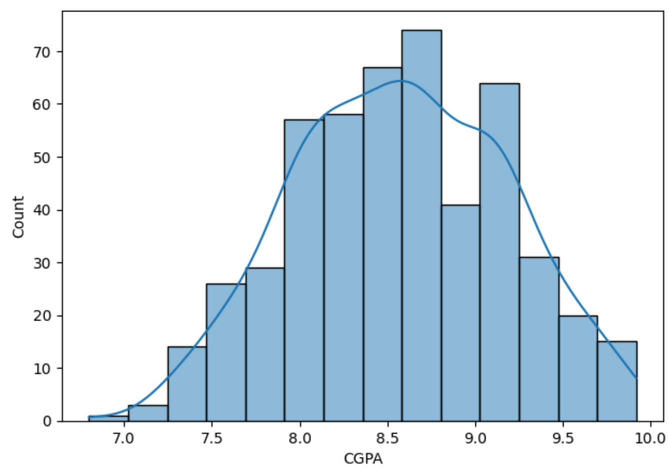
```
184
```

```
df["CGPA"].value_counts(bins=5) # Maximum applicants Undergraduate GPA score lie
```

```
CGPA
(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: count, dtype: int64
```

```
plt.subplot(121)
df["CGPA"].plot.box(figsize=(16,5))    # Median is at 8.56
plt.subplot(122)                       # CGPA Score data is normaly distributed
sns.histplot(df["CGPA"], kde=True)     # no outliers present
plt.show()
```





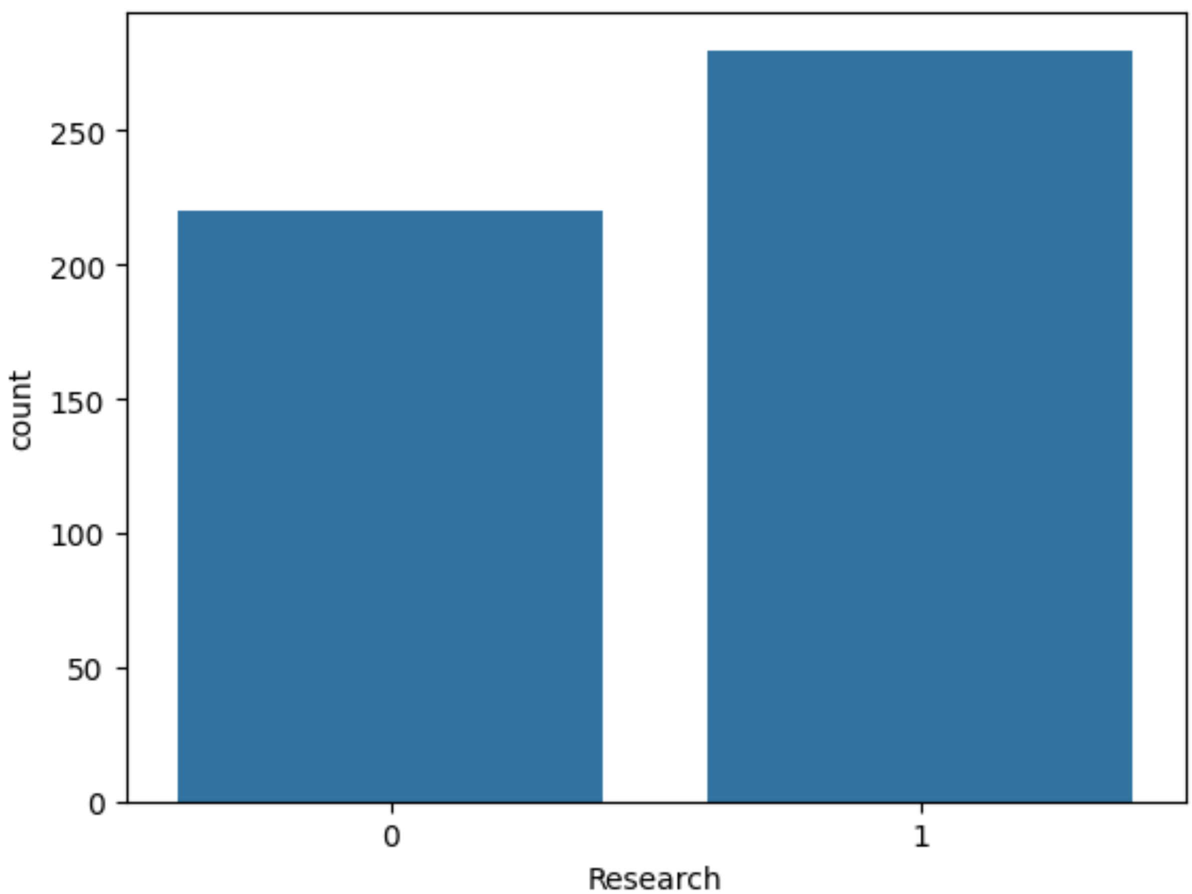
```
df["Research"].unique()
```

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

```
df["Research"].value_counts() # Maximum applicants has Research Experience score
```

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

```
sns.countplot(data=df,x="Research") # Visual representation of the above code
plt.show()                           # Research experience applicants has high cha
```



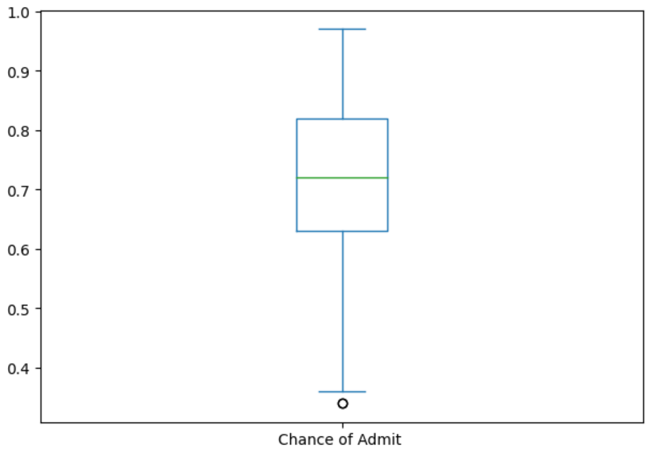
```
df["Chance of Admit "].unique()
```

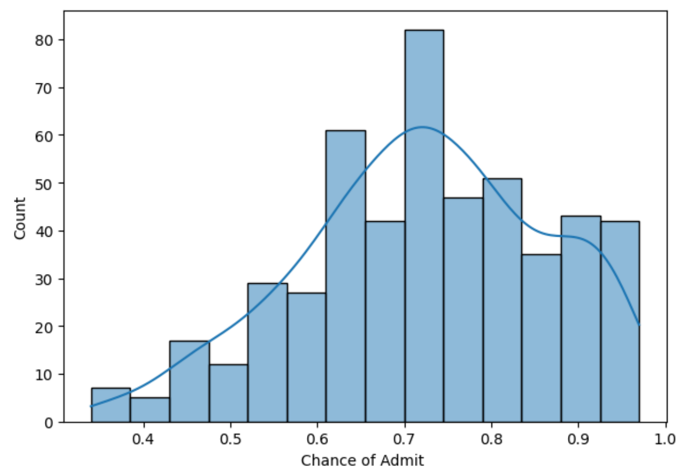
```
array([0.92, 0.76, 0.72, 0.8 , 0.65, 0.9 , 0.75, 0.68, 0.5 , 0.45, 0.52,
       0.84, 0.78, 0.62, 0.61, 0.54, 0.66, 0.63, 0.64, 0.7 , 0.94, 0.95,
       0.97, 0.44, 0.46, 0.74, 0.91, 0.88, 0.58, 0.48, 0.49, 0.53, 0.87,
       0.86, 0.89, 0.82, 0.56, 0.36, 0.42, 0.47, 0.55, 0.57, 0.96, 0.93,
       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) # Maximum applicants chances of admit
```

```
Chance of Admit
(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: count, dtype: int64
```

```
plt.subplot(121)
df["Chance of Admit "].plot.box(figsize=(16,5))    # Median is at 0.72
plt.subplot(122)                                   # Chance of admit data is left
sns.histplot(df["Chance of Admit "], kde=True)     # There are some outliers pres
plt.show()
```

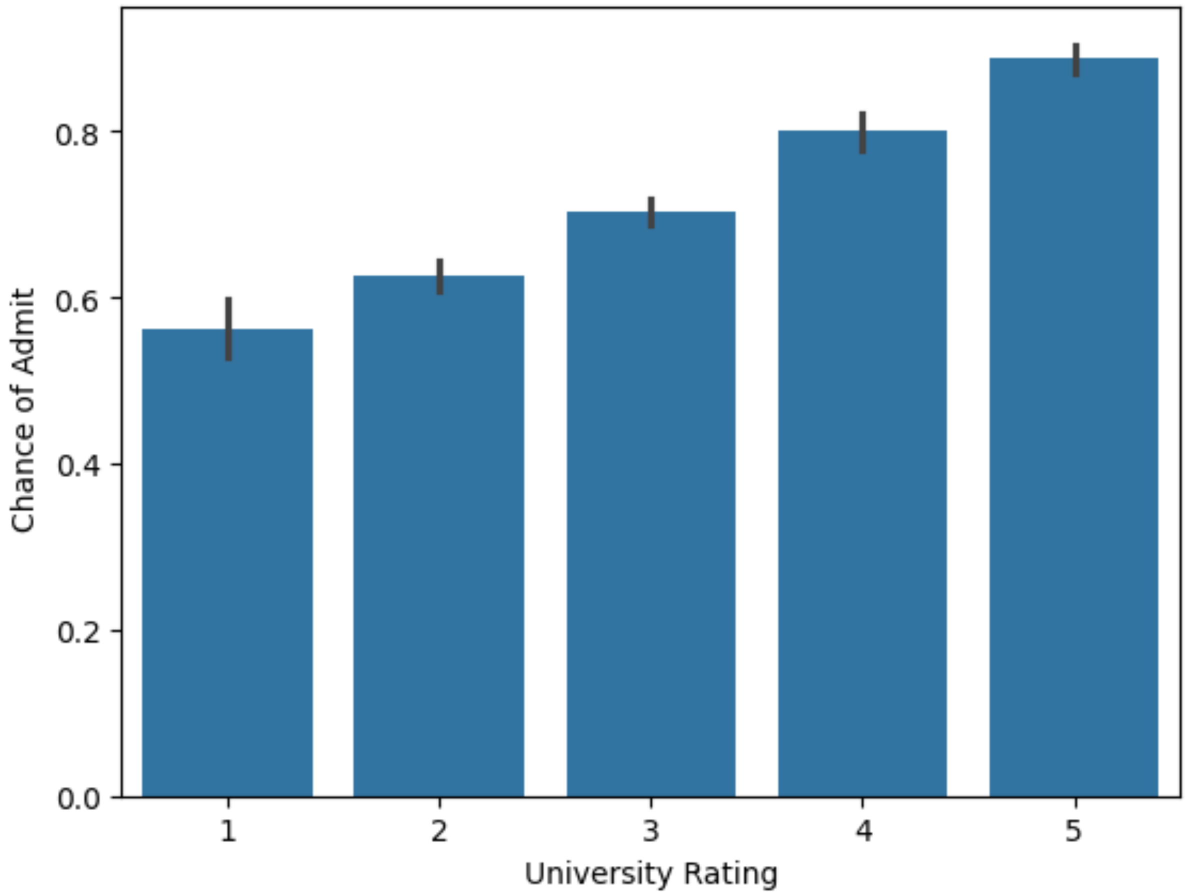




Bivariate Analysis

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

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



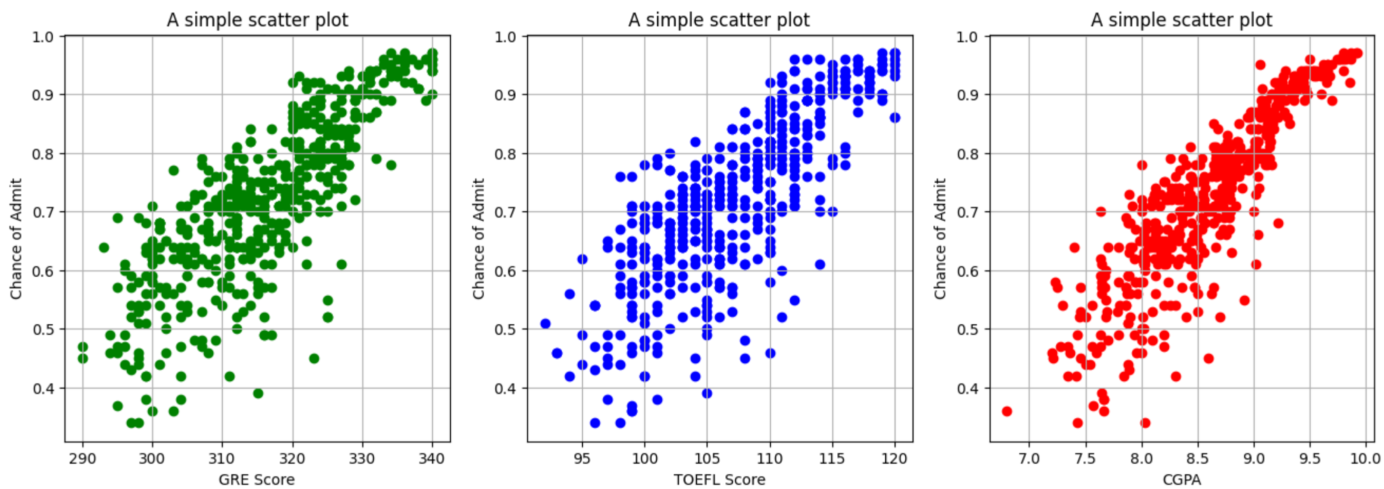

```
plt.rcParams["figure.figsize"] = (16,5)

plt.subplot(1,3,1)
plt.scatter(x="GRE Score",y="Chance of Admit ",data=df,c='g')
plt.title('A simple scatter plot')  # GRE score and chance of admit is directly p
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') # TOEFL Score and chance of admit is directly
plt.xlabel('TOEFL Score')
plt.ylabel('Chance of Admit')
plt.grid()

plt.subplot(1,3,3)
```

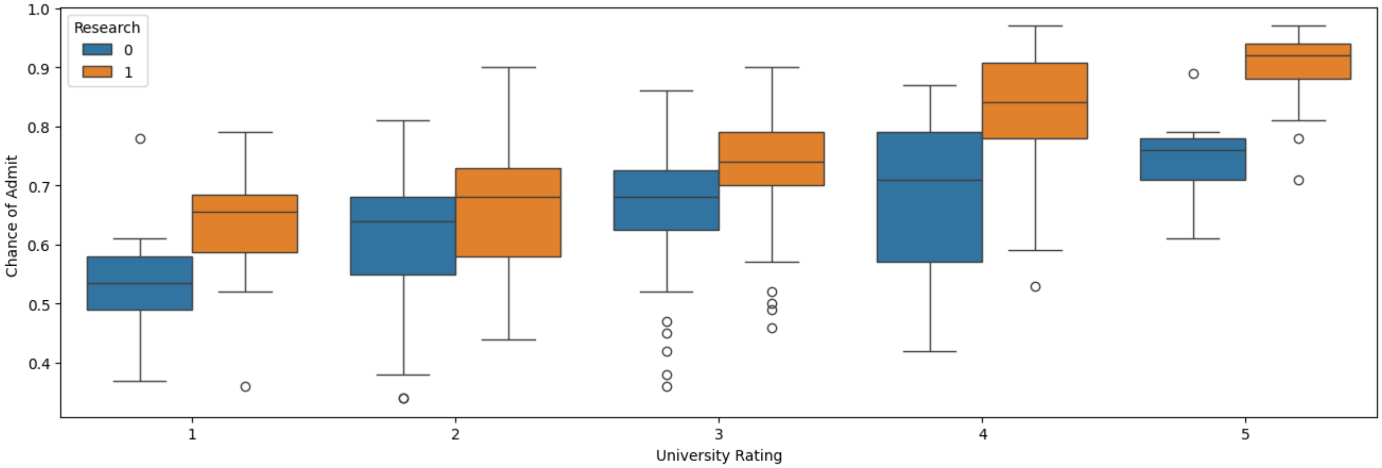
```
plt.scatter(x="CGPA",y="Chance of Admit ",data=df,c='r')
plt.title('A simple scatter plot')  # CGPA and chance of admit is directly prapor
plt.xlabel('CGPA')
plt.ylabel('Chance of Admit')
plt.grid()
```



Mulativariate Analysis

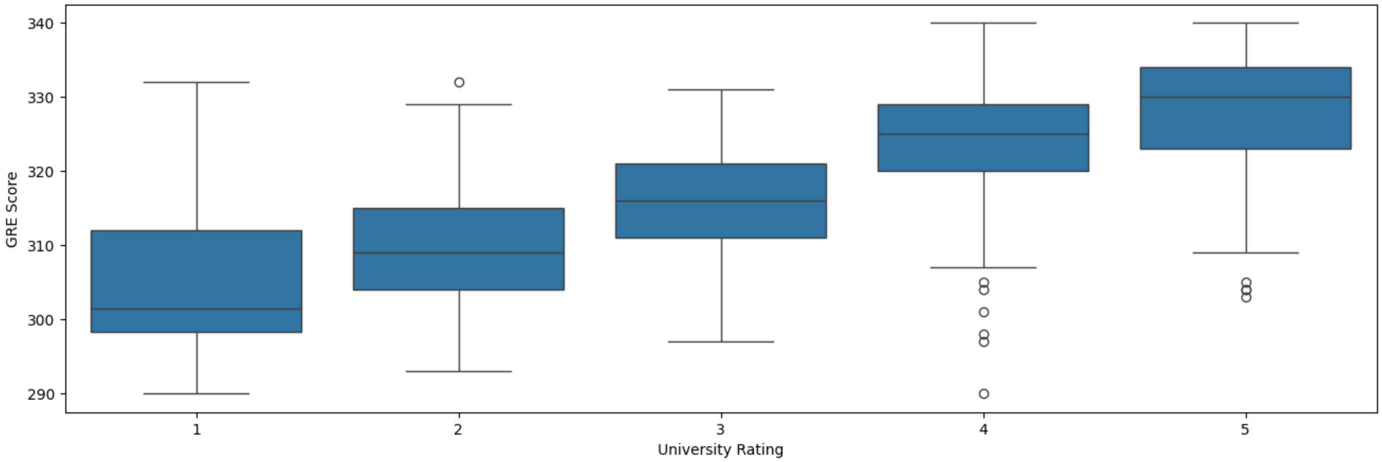
```
sns.boxplot(x="University Rating",hue="Research",data=df,y="Chance of Admit ",dod
# applicant from university rating 4 with no research experience has more chances
```

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



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

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



2. Data Preprocessing

Duplicate value check

```
bool_series = df.duplicated() # From value count we can see that there are zero d
bool_series.value_counts()
```

```
False    500
Name: count, dtype: int64
```

Missing value treatment

```
(df.isnull().sum()/len(df))*100 # No missing value present in the data
```

```
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
```

Outlier treatment

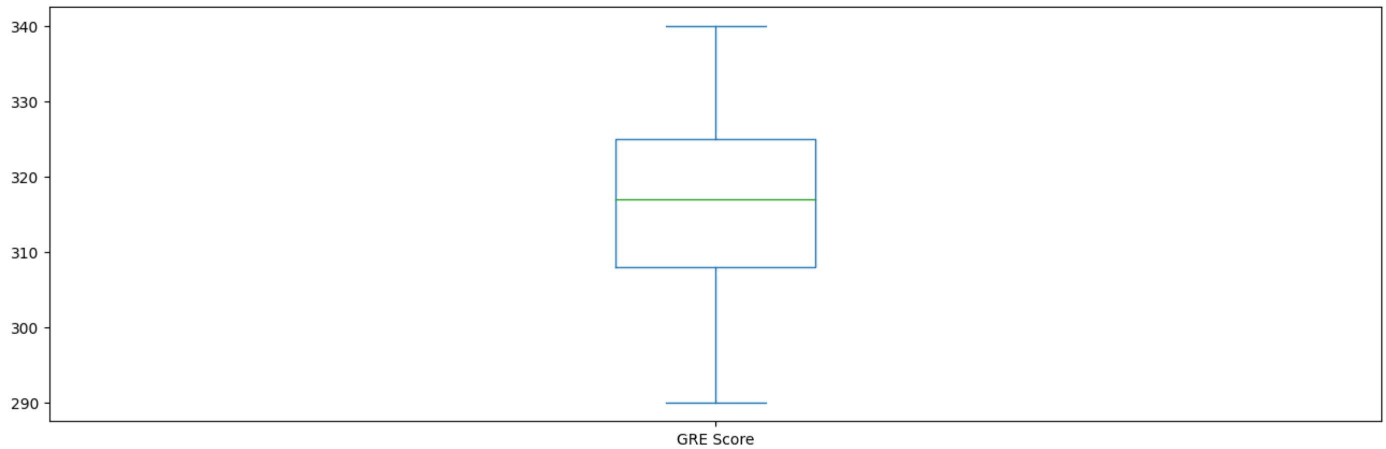
```
df.describe()
```

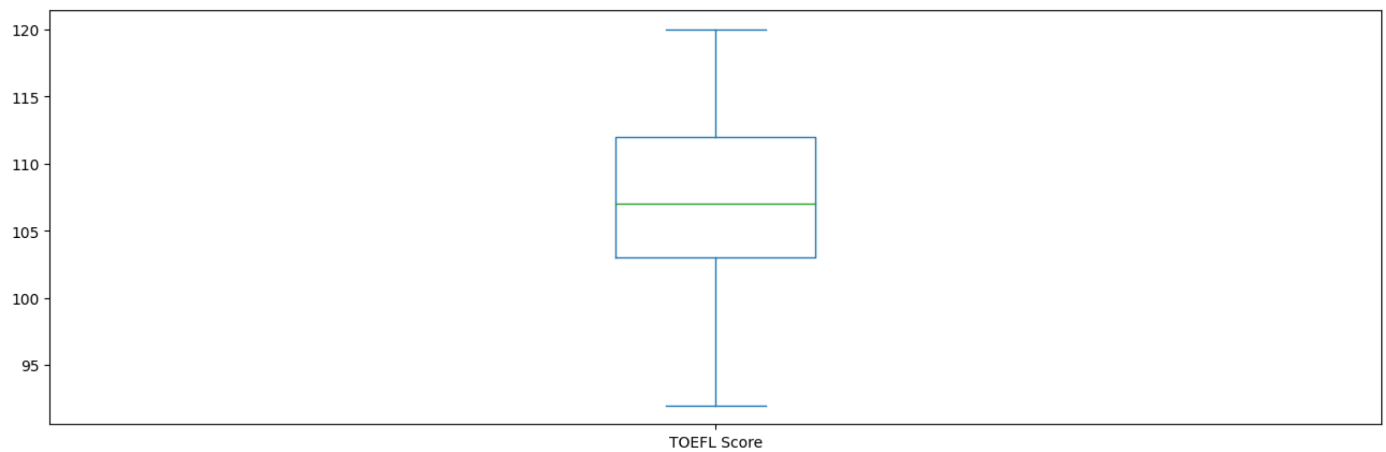
	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Researc
count	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.00000
mean	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.56000
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.49688
min	290.000000	92.000000	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

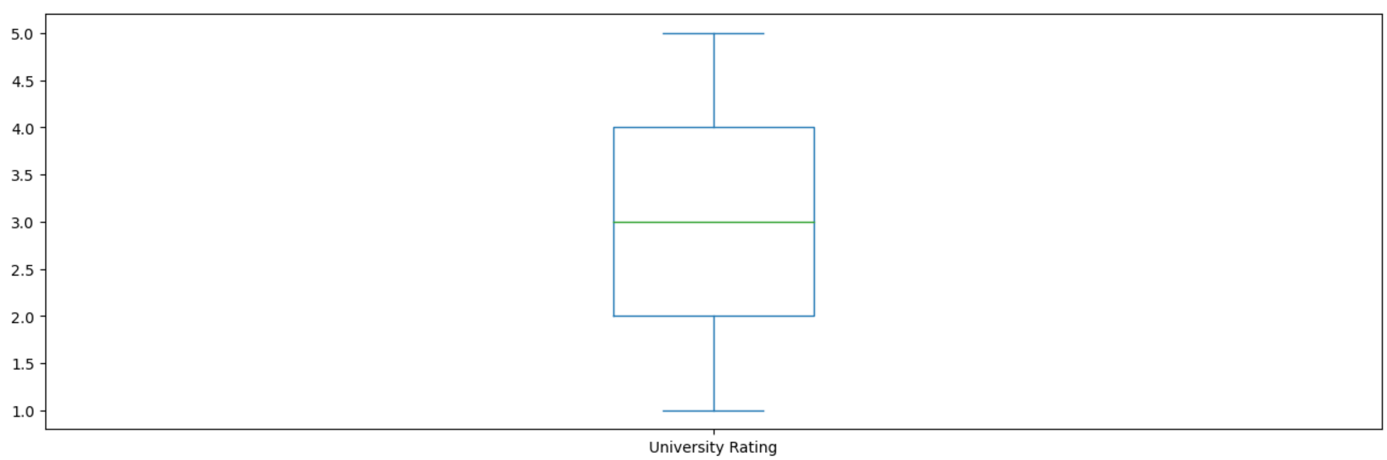
```
df.columns
```

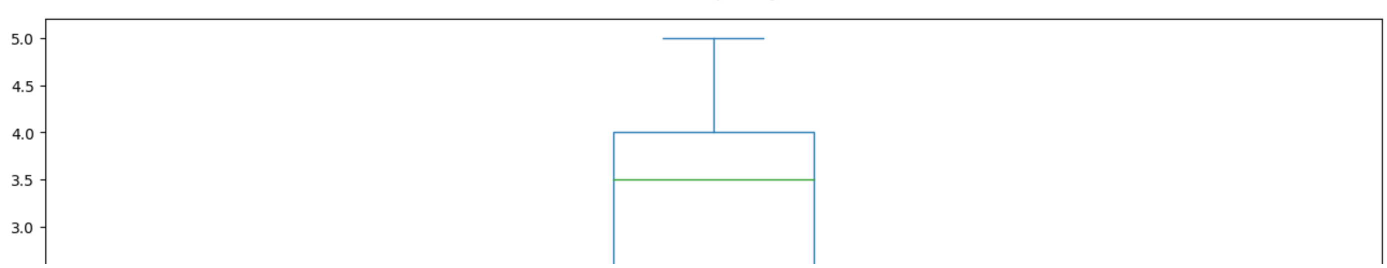
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ',
'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

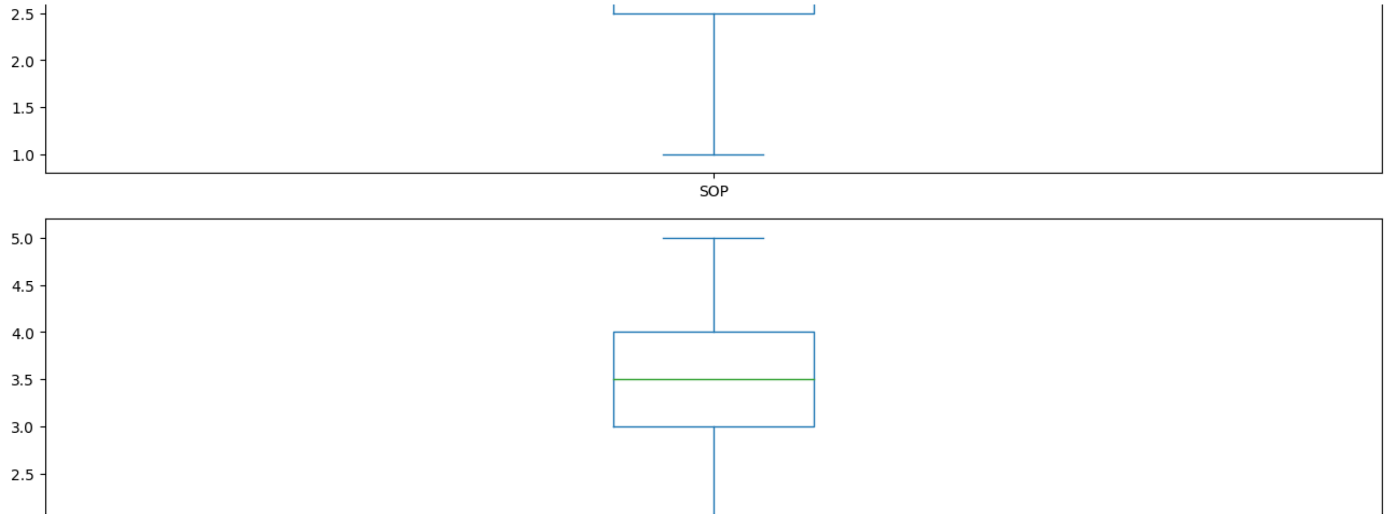
```
total_columns=['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'C
for col in total_columns:
  df[col].plot.box(figsize=(16,5))
  plt.show()
```












```
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)
```

```
df.shape # Outliers are very less in the data so we can neglect the it.
```

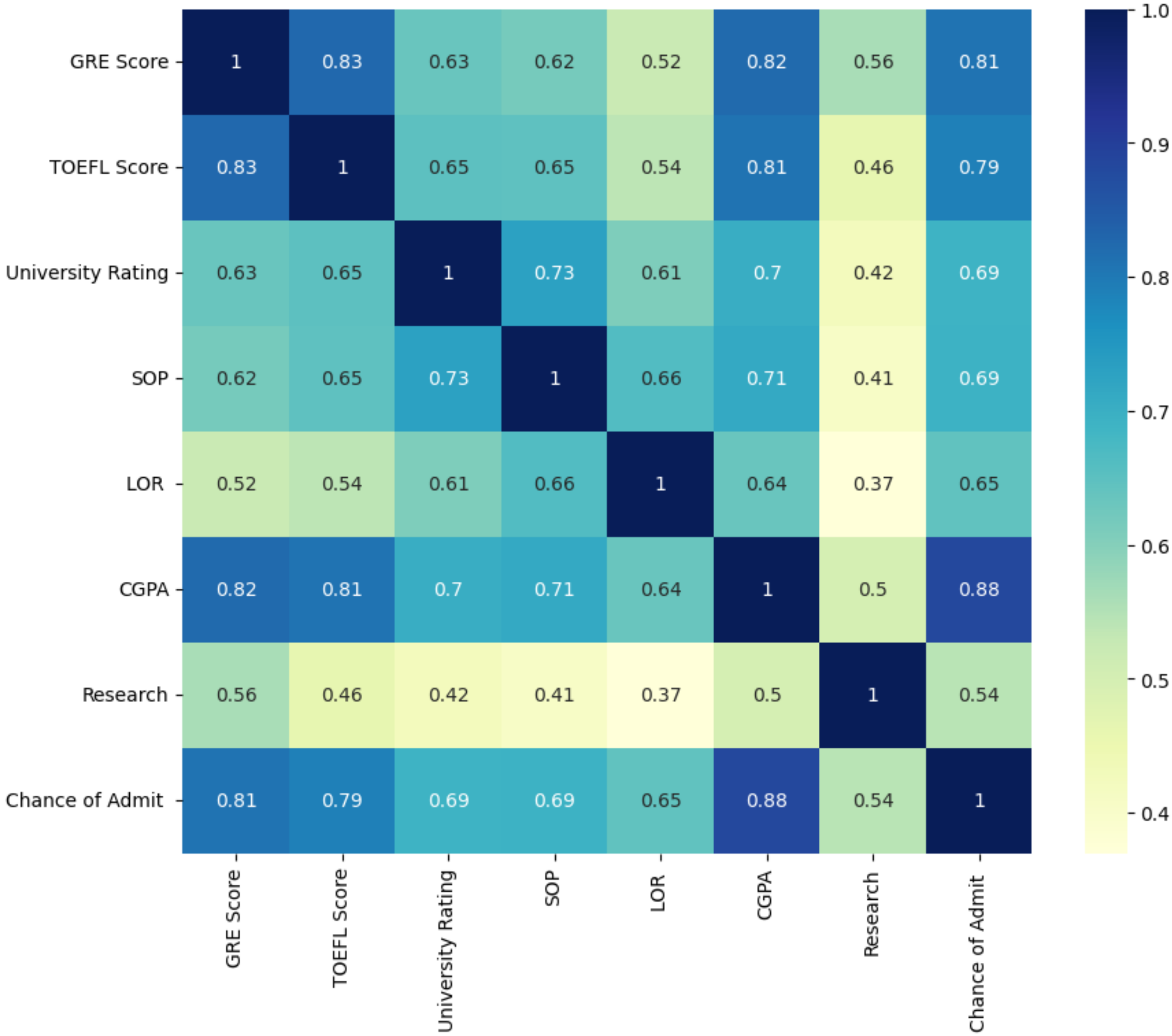
```
(498, 8)
```

Correlations

```
df.shape
```

```
(498, 8)
```

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


GRE Score
TOEFL Score
University Rating
SOP
LOR
CGPA
Research
Chance of Admit
1 0.83 0.63 0.62 0.52 0.82 0.56 0.81
0.83 1 0.65 0.65 0.54 0.81 0.46 0.79
0.63 0.65 1 0.73 0.61 0.7 0.42 0.69
0.62 0.65 0.73 1 0.66 0.71 0.41 0.69
0.52 0.54 0.61 0.66 1 0.64 0.37 0.65
0.82 0.81 0.7 0.71 0.64 1 0.5 0.88
0.56 0.46 0.42 0.41 0.37 0.5 1 0.54
0.81 0.79 0.69 0.69 0.65 0.88 0.54 1
1.0
0.9
0.8
0.7
0.6
0.5
0.4

Some insights on correlation :

1. GRE score is highly correlated with chance of admit
2. TOEFL score is highlty correlated with chanse of admit.
3. CGPA is also highly correlated with chanse of admit.
4. University rating, SOP and LOR are almost samely correlated with taget variable which is chanse to admit.
5. Some independent variables are highly correlated with the independent variables, meaning multicollinearity is present in the data. for example GRE score is highly correlated with TOEFL score with 0.83

Feature engineering

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

```
# Feature engineering adding extra parameter
ratio_CGPA_GRE=(df["CGPA"]/df["GRE Score"])*100
df["ratio_CGPA_GRE"]=ratio_CGPA_GRE
```

```
# let's combine SOP and LOR columns with name SOP_LOR_total
ratio_CGPA_TOEFL=(df["CGPA"]/df["TOEFL Score"])*100
df["ratio_CGPA_TOEFL"]=ratio_CGPA_TOEFL
```

```
df.head()
```

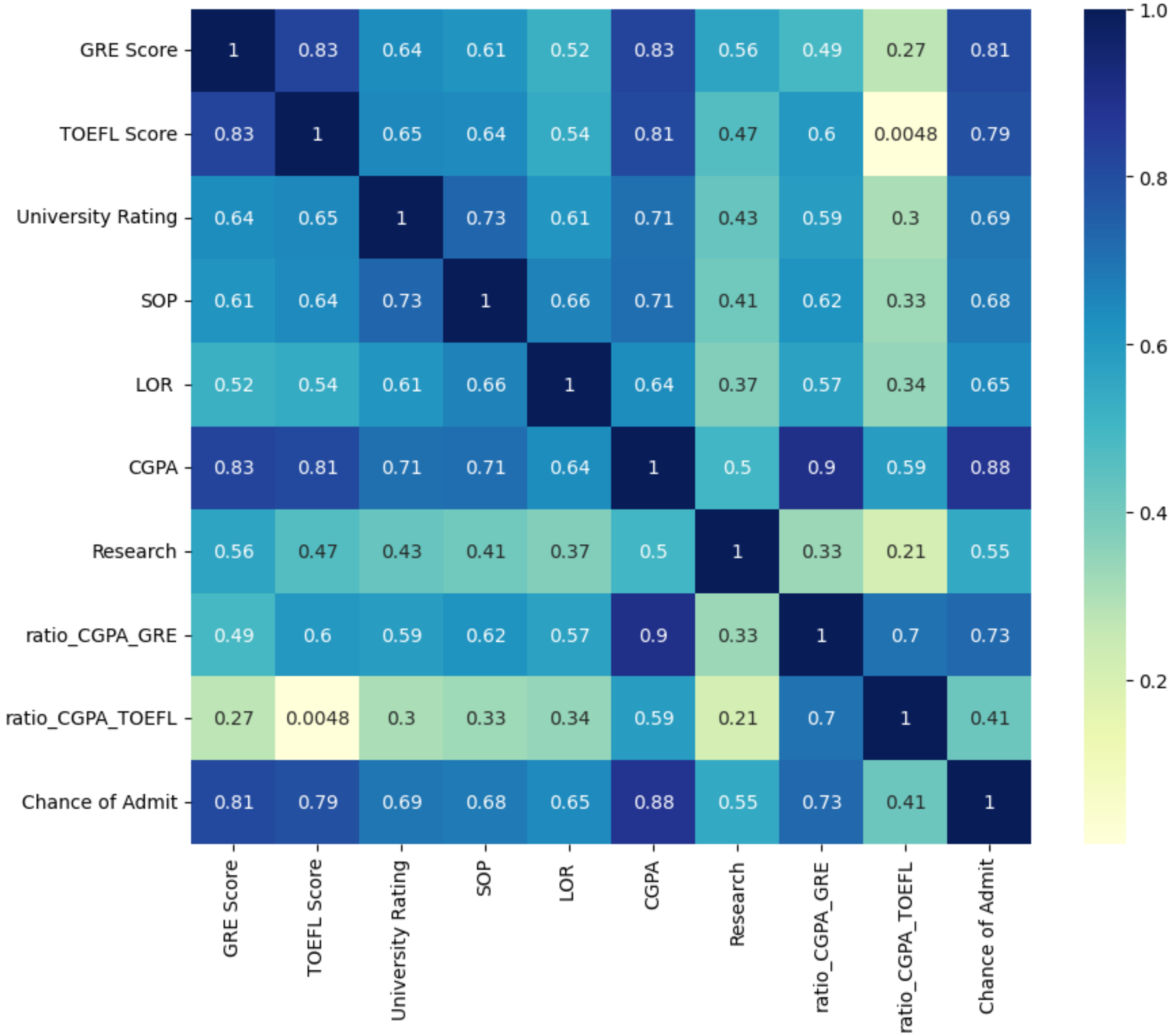
	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit	ratio_CGP
0	1	337	118	4	4.5	4.5	9.65	1	0.92	2.8
1	2	324	107	4	4.0	4.5	8.87	1	0.76	2.7
2	3	316	104	3	3.0	3.5	8.00	1	0.72	2.5
3	4	322	110	3	3.5	2.5	8.67	1	0.80	2.6

```
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_CGP
0	337	118	4	4.5	4.5	9.65	1	2.863501	
1	324	107	4	4.0	4.5	8.87	1	2.737654	
2	316	104	3	3.0	3.5	8.00	1	2.531646	
3	322	110	3	3.5	2.5	8.67	1	2.692547	

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


GRE Score
TOEFL Score
University Rating
SOP
LOR
CGPA
Research
ratio_CGPA_GRE
ratio_CGPA_TOEFL
Chance of Admit
1.0
0.8
0.6
0.4
0.2
1 0.83 0.64 0.61 0.52 0.83 0.56 0.49 0.27 0.81
0.83 1 0.65 0.64 0.54 0.81 0.47 0.6 0.0048 0.79
0.64 0.65 1 0.73 0.61 0.71 0.43 0.59 0.3 0.69
0.61 0.64 0.73 1 0.66 0.71 0.41 0.62 0.33 0.68
0.52 0.54 0.61 0.66 1 0.64 0.37 0.57 0.34 0.65
0.83 0.81 0.71 0.71 0.64 1 0.5 0.9 0.59 0.88
0.56 0.47 0.43 0.41 0.37 0.5 1 0.33 0.21 0.55
0.49 0.6 0.59 0.62 0.57 0.9 0.33 1 0.7 0.73
0.27 0.0048 0.3 0.33 0.34 0.59 0.21 0.7 1 0.41
0.81 0.79 0.69 0.68 0.65 0.88 0.55 0.73 0.41 1

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

Standardization

```
## scaling
## Lets scale the data, standardization
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 ', 'C
```

```
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_C
0	1.819238	1.778865	0.775582	1.137360	1.098944	1.776806	0.886405	
1	0.667148	-0.031601	0.775582	0.632315	1.098944	0.485859	0.886405	
2	-0.041830	-0.525364	-0.099793	-0.377773	0.017306	-0.954043	0.886405	-
3	0.489904	0.462163	-0.099793	0.127271	-1.064332	0.154847	0.886405	-

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

```
df_new1.head() # dataframe ready for the modeling
```

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

```
df_new1.shape
```

```
(500, 10)
```

Model building

Simple linear regression

```
x = df_new1["CGPA"].values # CGPA is 0.88 correlated with taget variable i.e. cha
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]) ## randomly predicted value
```

```
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
```

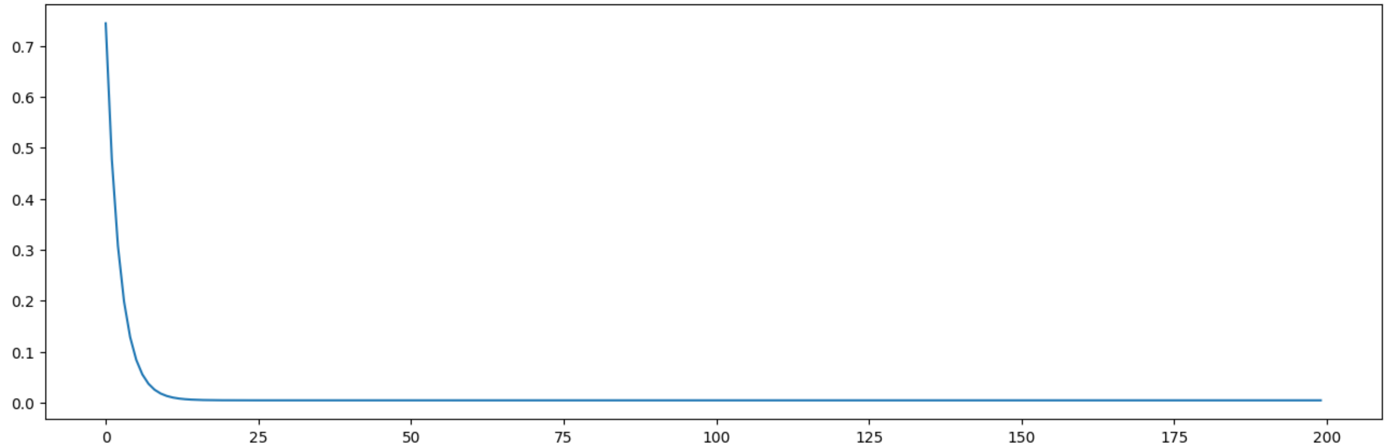
```
def gradient_descent(x,y,ran_itr=200,learning_rate=0.1):
    '''step1: initialise the variable '''
    weights=np.random.rand(2)
    ''' step2: rpeate for 100 times'''
    error_list=[]
    for i in range(ran_itr):
        e=error(x,y,weights)
        error_list.append(e)
        grade = gradient(x,y,weights)
        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 0x794f2ccf95a0>]
```




```
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) # performance of the simple linear regression model using CGPA
                  # Only CGPA is not important to check the chanse od admit hence
```

```
0.779
```

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_C
0	1.819238	1.778865	0.775582	1.137360	1.098944	1.776806	0.886405	
1	0.667148	-0.031601	0.775582	0.632315	1.098944	0.485859	0.886405	
2	-0.041830	-0.525364	-0.099793	-0.377773	0.017306	-0.954043	0.886405	-
3	0.489904	0.462163	-0.099793	0.127271	-1.064332	0.154847	0.886405	-

```
# Statmodels implementation of Linear regression
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 inter

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

print(sm_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:      Chance of Admit   R-squared:                       0.82
Model:                          OLS   Adj. R-squared:                  0.81
Method:               Least Squares   F-statistic:                     252.
Date:              Tue, 16 Jul 2024   Prob (F-statistic):          1.02e-17
Time:                      02:45:57   Log-Likelihood:                  702.3
No. Observations:               500   AIC:                            -1385.
Df Residuals:                   490   BIC:                            -1343.
Df Model:                         9
Covariance Type:          nonrobust
==============================================================================
                        coef    std err          t      P>|t|      [0.025
------------------------------------------------------------------------------
const                 0.7217      0.003    269.021      0.000       0.716
GRE Score             0.1270      0.079      1.607      0.109      -0.028
TOEFL Score          -0.0343      0.089     -0.386      0.700      -0.209
University Rating     0.0067      0.004      1.538      0.125      -0.002
SOP                   0.0014      0.005      0.316      0.752      -0.007
LOR                   0.0156      0.004      4.066      0.000       0.008
CGPA                 -0.0739      0.121     -0.611      0.542      -0.312
Research              0.0123      0.003      3.736      0.000       0.006
ratio_CGPA_GRE        0.1357      0.101      1.345      0.179      -0.062
ratio_CGPA_TOEFL     -0.0377      0.065     -0.583      0.560      -0.165
==============================================================================
Omnibus:                    118.043   Durbin-Watson:                   0.80
Prob(Omnibus):                0.000   Jarque-Bera (JB):              283.37
Skew:                        -1.198   Prob(JB):                    2.93e-6
Kurtosis:                     5.803   Cond. No.                        148.
==============================================================================

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

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_s

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_
```

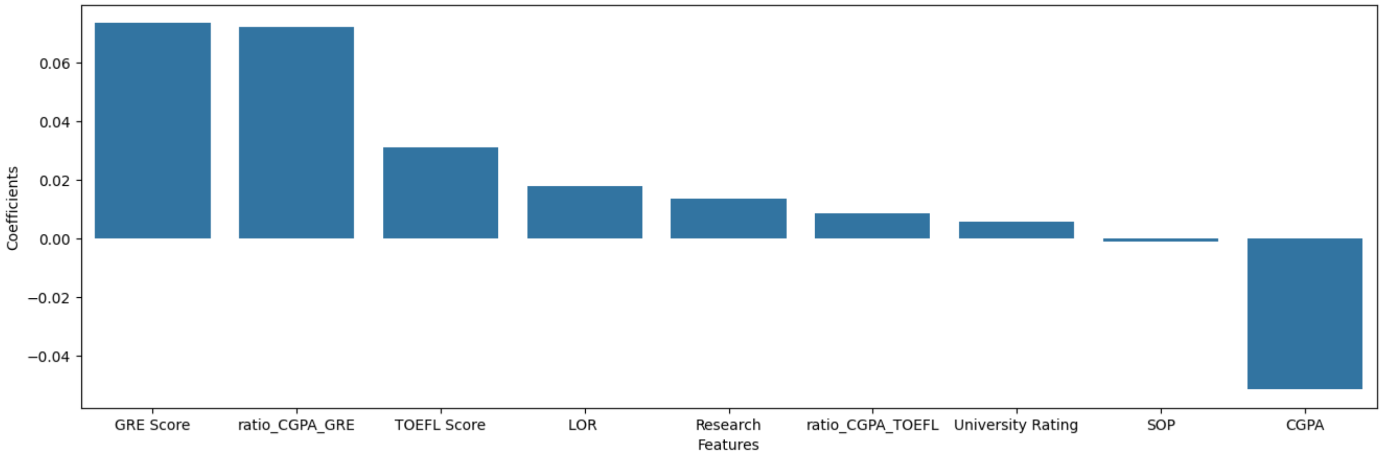
```
array([ 0.07362709,  0.03097081,  0.00572688, -0.00115692,  0.01779132,
       -0.05144605,  0.01351941,  0.07205956,  0.00848471])
```

```
coeff=pd.DataFrame()                                        # GRE score has highest w
X_c=X                                                       # 3rd highest weight is o
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

```
sns.barplot(x="Features",y="Coefficients",data=coeff) # visual represntation of t
```

```
<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_s


# 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',
Intercepts: 0.72
LASSO R2_score: -0.0424956830527512
```

Note:- Here, 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', 0.03259324
Intercepts: 0.72
Ridge R2_score: -0.0425
```

Note:- 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

1.Multicollinearity check by VIF score (variables are dropped one-by-one till none has VIF>5)

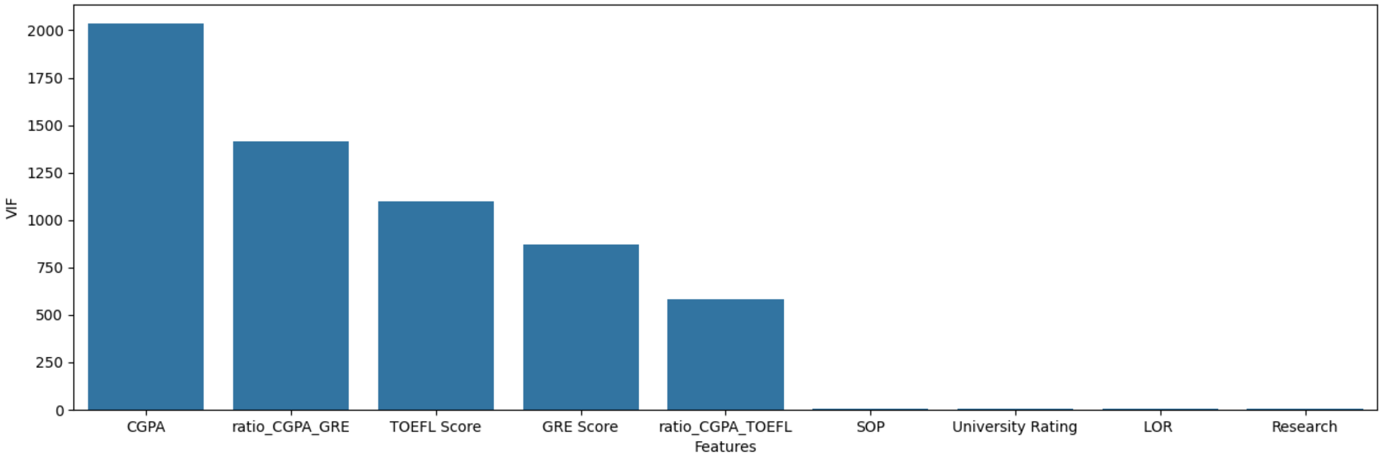
```
# VIF (Variance Inflation Factor)
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
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

```
sns.barplot(x="Features",y="VIF",data=vif) # visual rpresntation of VIF for each
                                           # Any variable with a VIF of 10 or abo
                                           # CGPA, TOEFL score, GRE score these a
                                           # SOP, University rating, LOR and Rese
                                           # new encoded feature ratio_CGPA_GRE a
```

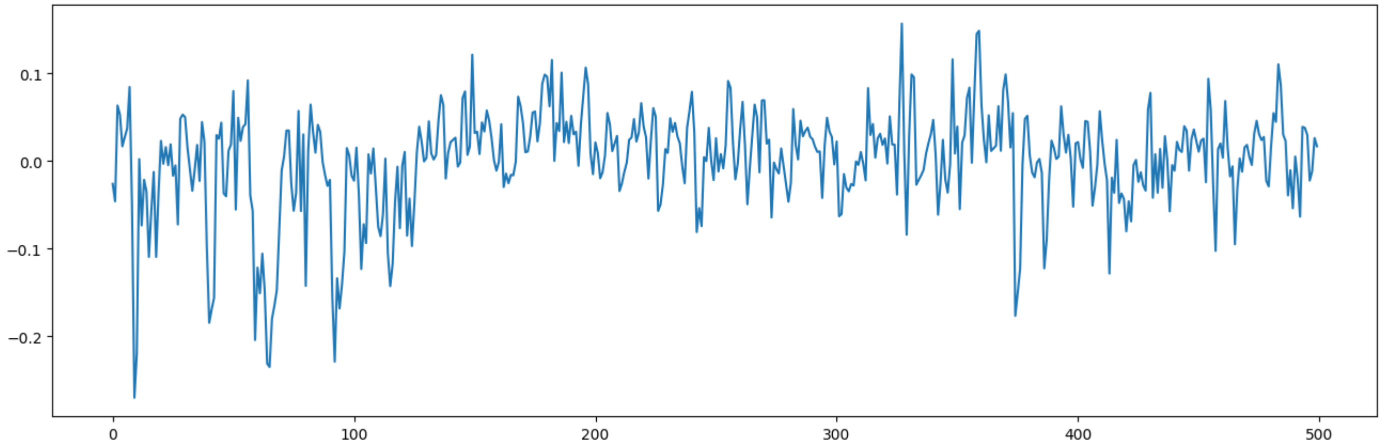
```
<Axes: xlabel='Features', ylabel='VIF'>
```



2. The mean of residuals is nearly zero

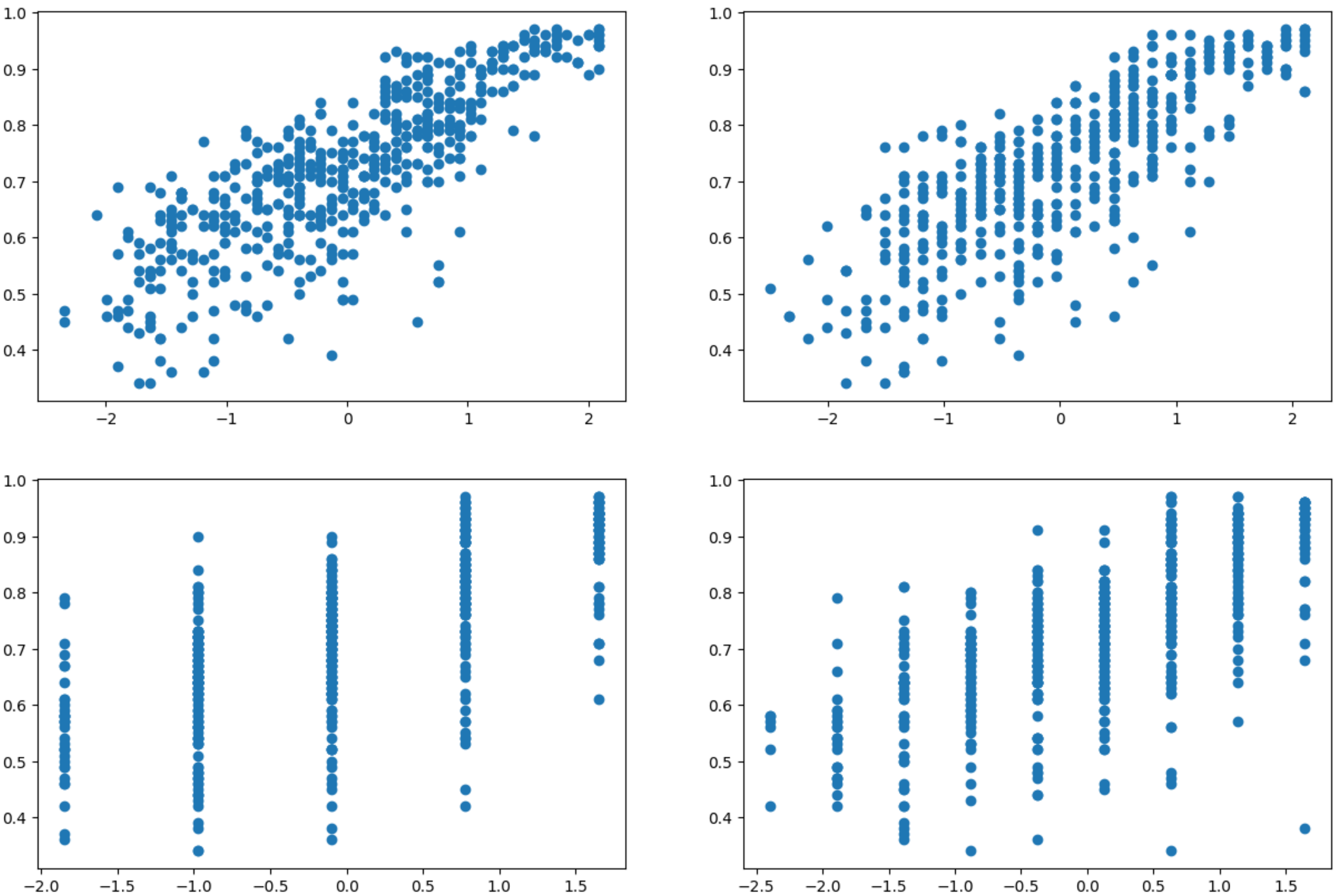
```
residuals=sm_model.resid
plt.plot(residuals.index,residuals)
```

[<matplotlib.lines.Line2D at 0x794edc9915a0>]



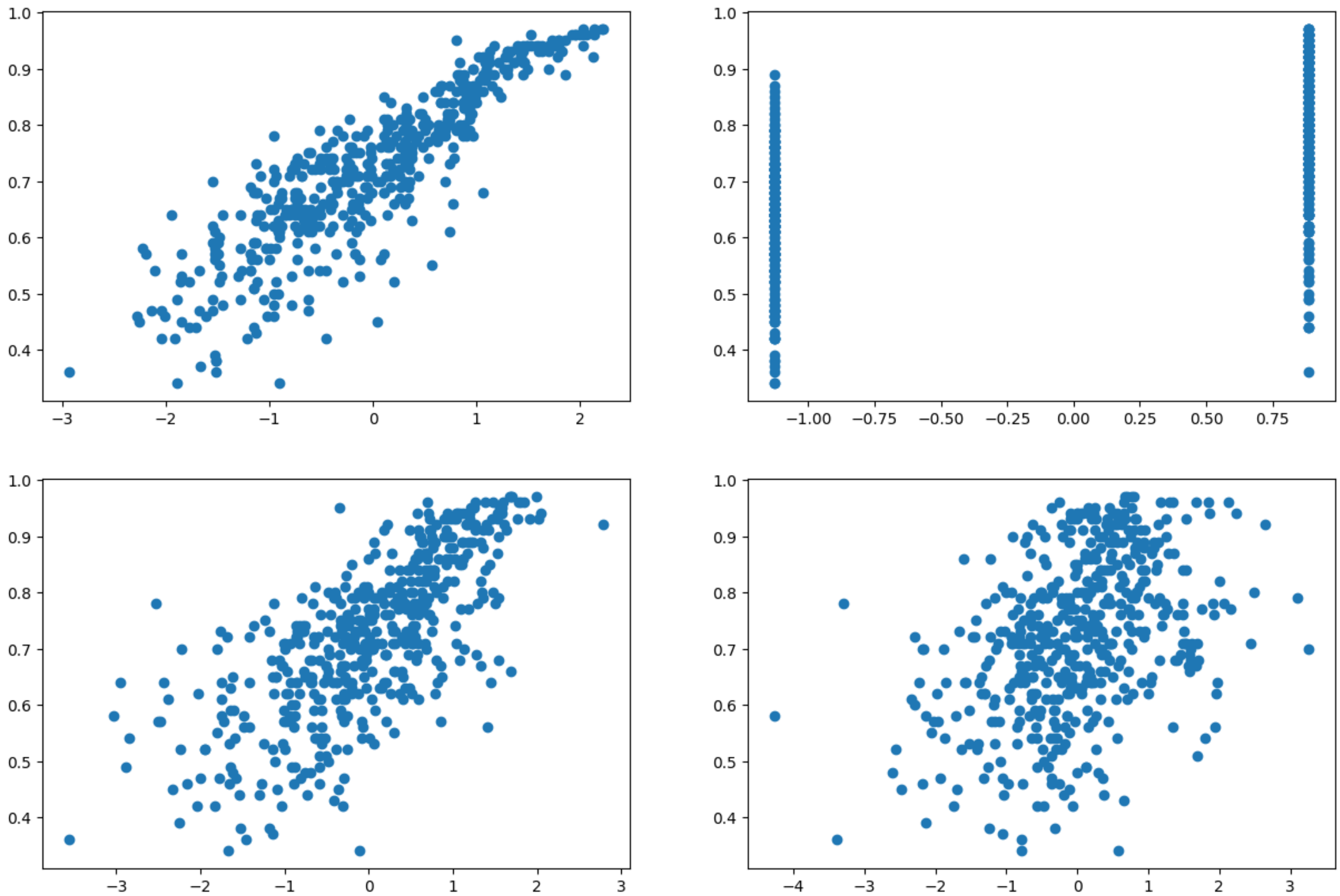
3. Linearity of variables (no pattern in the residual plot)

```
from mpl_toolkits.mplot3d import axes3d                  # In this dataset almos
plt.rcParams["figure.figsize"]=(15,10)                   # for example GRE scoir
fig,((ax1,ax2),(ax3,ax4))=plt.subplots(2,2)
ax1.scatter(X["GRE Score"],Y)
ax2.scatter(X["TOEFL Score"],Y)
ax3.scatter(X["University Rating"],Y)
ax4.scatter(X["SOP"],Y)
plt.show()
```


1.0
0.9
0.8
0.7
0.6
0.5
0.4
−2
−1
0
1
2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
−2.5

Start coding or generate with AI.

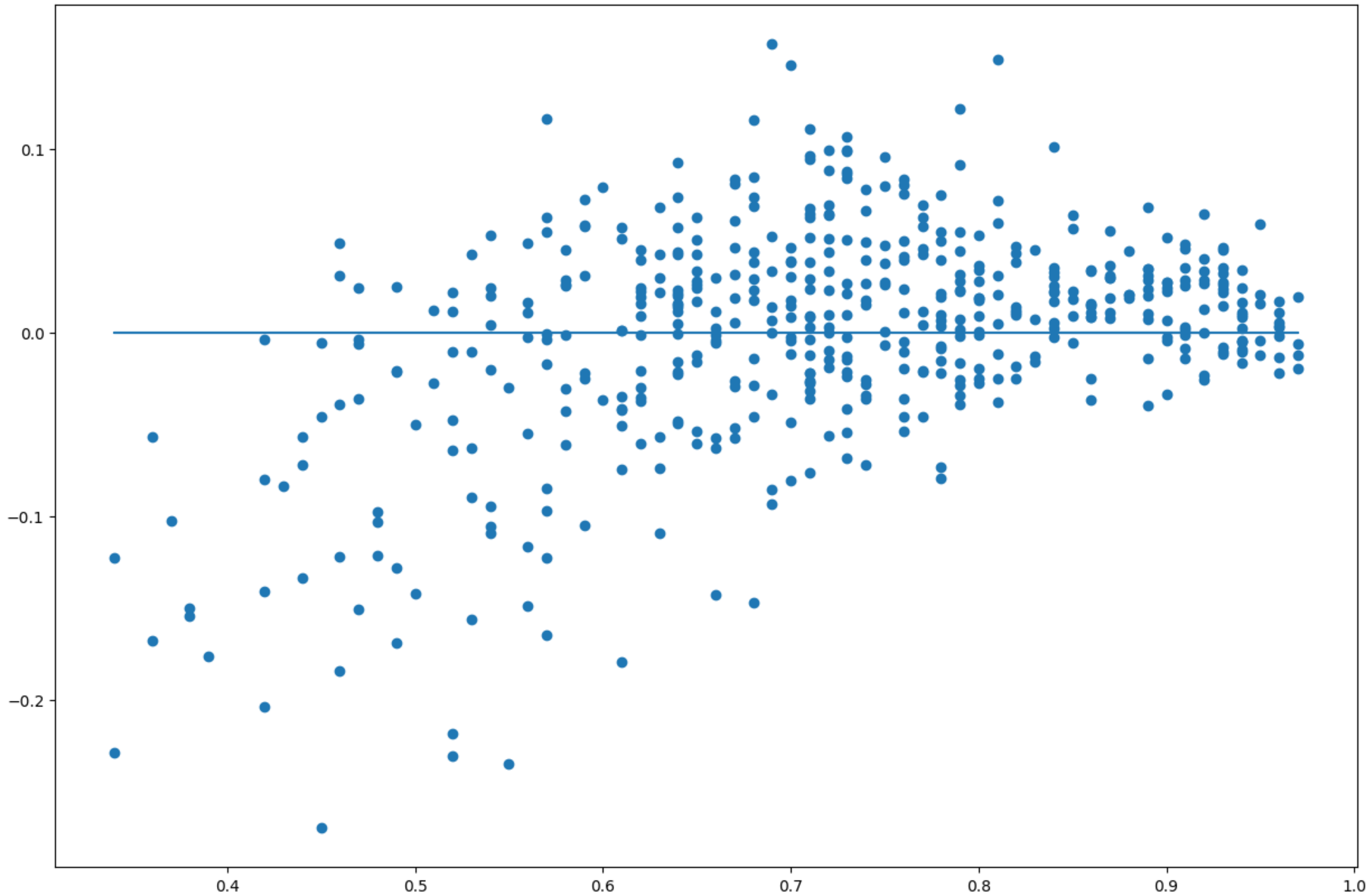
```
from mpl_toolkits.mplot3d import axes3d           # CGPA is linearly related wi
plt.rcParams["figure.figsize"]=(15,10)
fig,((ax1,ax2),(ax3,ax4))=plt.subplots(2,2)
ax1.scatter(X["CGPA"],Y)
ax2.scatter(X["Research"],Y)
ax3.scatter(X["ratio_CGPA_GRE"],Y)
ax4.scatter(X["ratio_CGPA_TOEFL"],Y)
plt.show()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



4. Test for Homoscedasticity

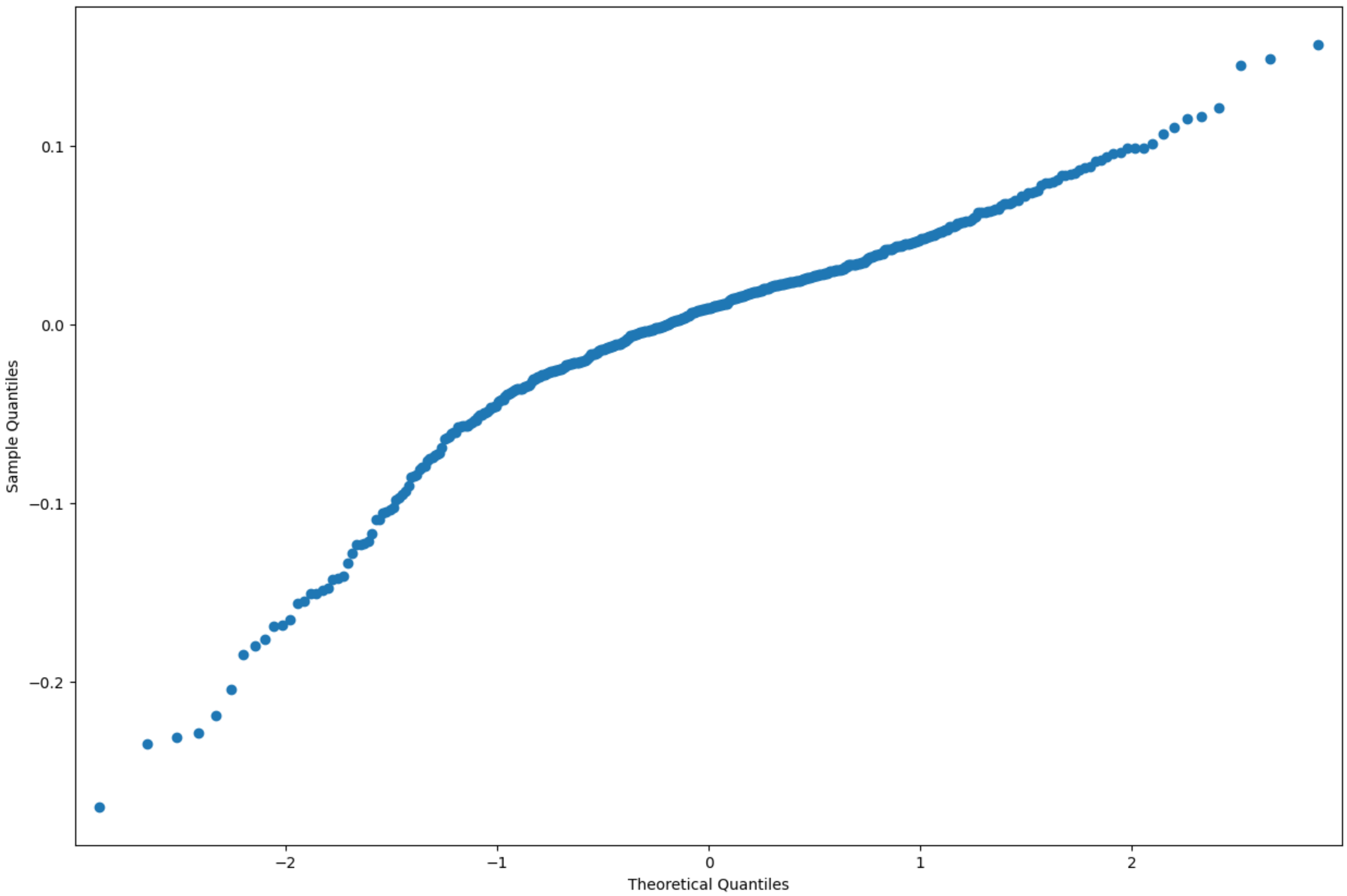
```
residuals=sm_model.resid   # In prob/stats proof of linear regression, we assume 
plt.scatter(Y,residuals)   # Homoscedasticity exists in our data.
plt.plot(Y,[0]*len(Y))     # There is no outliers present in the dataset.
```

[<matplotlib.lines.Line2D at 0x794edc6d42e0>]



5. Normality of residuals (almost bell-shaped curve in residuals distribution, points in QQ plot are almost all on the line)

```
residuals=sm_model.resid    # from qqplot we can say that normality of residuals
sm.qqplot(residuals)
plt.show()
```

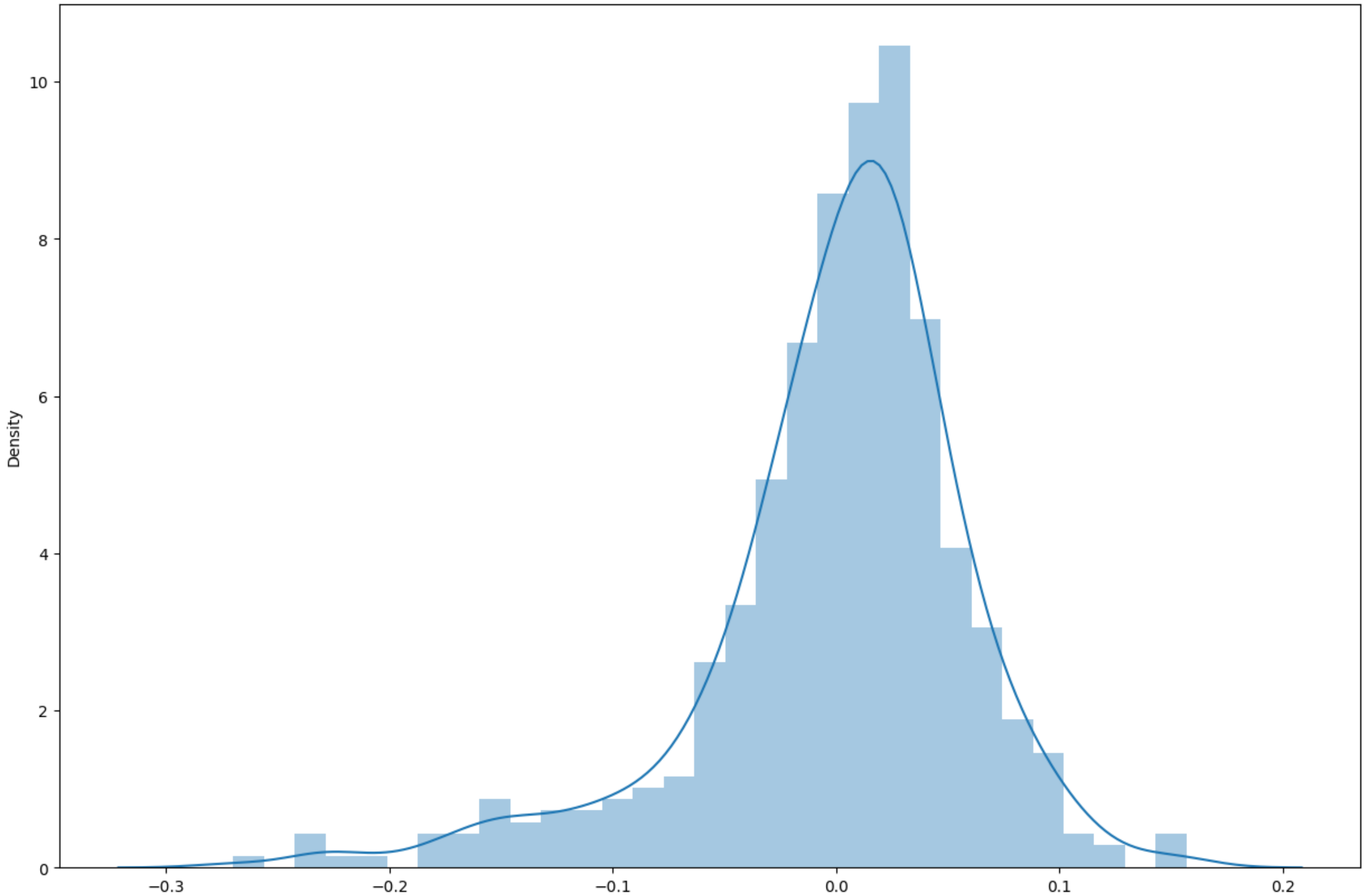

0.1
0.0
−0.1
−0.2
Sample Quantiles
−2
−1
0
1
2
Theoretical Quantiles

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

```
1.0746958878371515e-16
```

```
sns.distplot(residuals) # Perfectly normaly distributed cure we found here.
                        # we get almost bell-shaped curve in residuals distributi
                        # There is very less outliers present in the dataset
```

<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
```

```
#Adjusted R2 score
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

Start coding or generate with AI.

Train and test performances are checked

```
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_tra
print("mean_squared_error of test data=",mean_squared_error(y_test, predict_test)
print()
print("mean_absolute_error of train data=",mean_absolute_error(y_train, predict_t
print("mean_absolute_error of test data=",mean_absolute_error(y_test, predict_tes
```

```
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
```

Comments on the performance measures

- 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. Probability of Admission:

The linear regression model developed can be used as a feature where students/learners can input their details on Jamboree's website to check their probability of getting into an Ivy League college. The model has shown an accuracy of 81%, providing a reliable estimate of a student's chances of admission.

2. Attracting and Engaging Audience:

By offering this predictive tool, Jamboree can attract a larger audience of students/learners interested in their probability of admission. This feature not only engages users but also helps Jamboree gather basic information about potential students for targeted marketing efforts.

3. Identifying Students for Coaching:

The model can identify students/learners with a lower probability of admission. Jamboree can proactively reach out to these students and offer tailored coaching services to improve their chances of getting into their dream universities. This approach can enhance Jamboree's business by converting these students into paying customers.

4.Region-Specific Marketing: To further refine the data collection process, it is recommended to add a column for city or region names in the dataset. This additional information will allow Jamboree to identify target audiences from specific regions and tailor their marketing strategies accordingly. Region-specific insights can lead to more effective and focused marketing campaigns.