Problem Statement : Your analysis will help Jamboree in understanding what factors are important in graduate admissions and how these factors are interrelated among themselves. It will also help predict one's chances of admission given the rest of the variables.

```
from google.colab import files
files.upload()
```

```
Choose Files Jamboree_Admission.csv
• Jamboree_Admission.csv(text/csv) - 16176 bytes, last modified: 9/7/2024 - 100% done
Saving Jamboree_Admission.csv to Jamboree_Admission.csv
{'Jamboree_Admission.csv': b'Serial No.,GRE Score,TOEFL Score,University Rating,SOP,LOR ,CGPA,Research,Chance of Admit
\r\n1,337,118,4,4.5,4.5,9.65,1,0.92\r\n2,324,107,4,4,4.5,8.87,1,0.76\r\n3,316,104,3,3,3.5,8,1,0.72\r\n4,322,110,3,3.5,2.5,8.67,1,0.8\r\n
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import math as m
from statsmodels.stats import weightstats as stests
from scipy import stats
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.linear_model import Lasso, Ridge
```

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

	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 | New interactive sheet

Column Profiling:

Serial No. (Unique row ID)

GRE Scores (out of 340)

TOEFL Scores (out of 120)

University Rating (out of 5)

Statement of Purpose and Letter of Recommendation Strength (out of 5)

Undergraduate GPA (out of 10)

Research Experience (either 0 or 1)

Chance of Admit (ranging from 0 to 1)

Exploratory Data Analysis

```
df.shape
```

```
(500, 9)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         500 non-null    int64  
 1   GRE Score          500 non-null    int64  
 2   TOEFL Score        500 non-null    int64  
 3   University Rating  500 non-null    int64  
 4   SOP                500 non-null    float64
 5   LOR                500 non-null    float64
 6   CGPA               500 non-null    float64
 7   Research           500 non-null    int64  
 8   Chance of Admit    500 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 35.3 KB
```

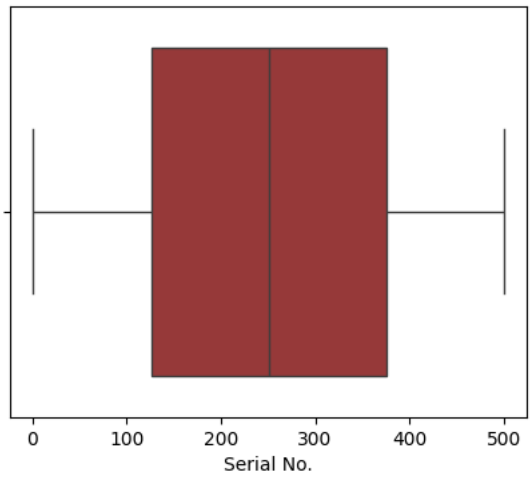
Checking null values

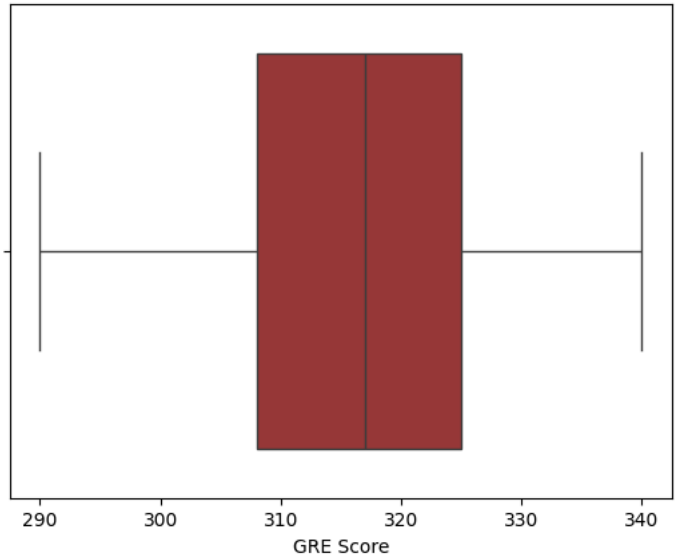
```
df.isna().sum()
```

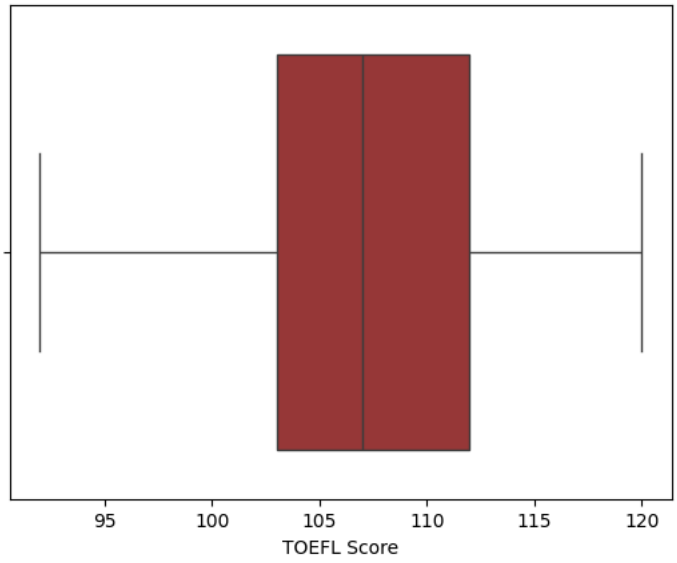
	0
Serial No.	0
GRE Score	0
TOEFL Score	0
University Rating	0
SOP	0
LOR	0
CGPA	0
Research	0
Chance of Admit	0

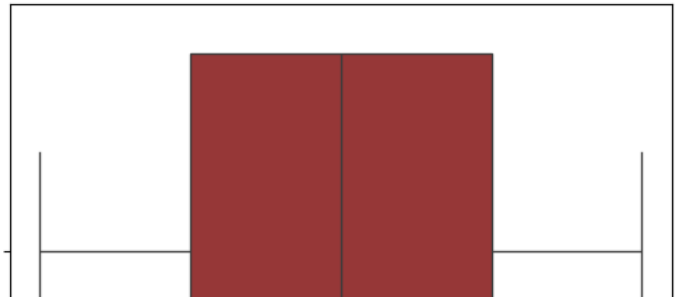
Outlier detection

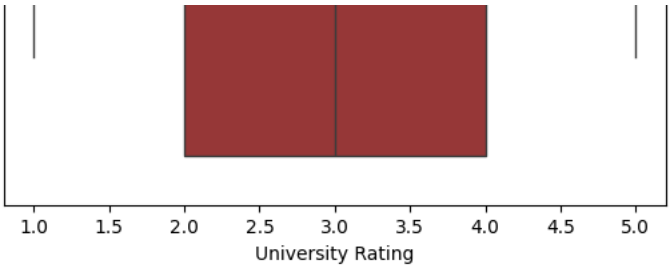
```
fig = plt.figure(figsize=(5,4))
for i in df.columns:
  sns.boxplot(x=df[i],color='brown')
  plt.show()
```

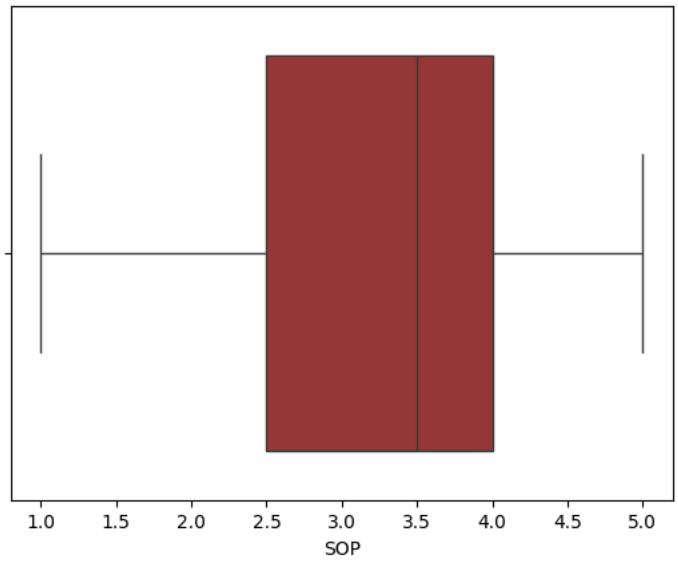

0
100
200
300
400
500
Serial No.

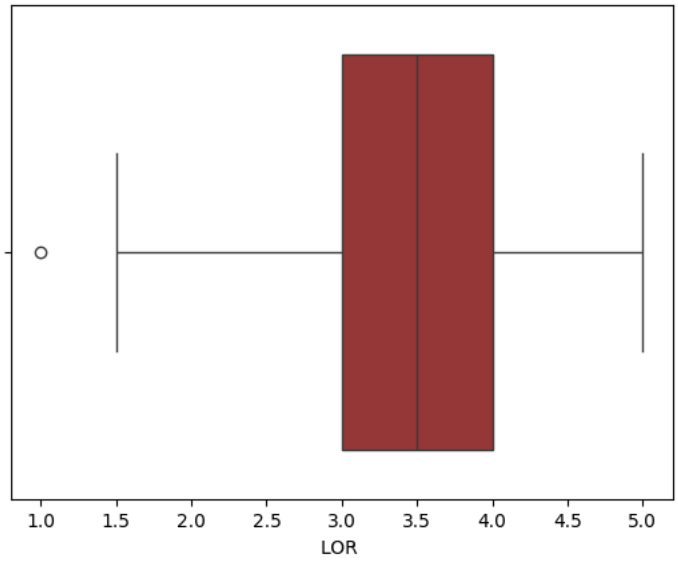

290
300
310
320
330
340
GRE Score

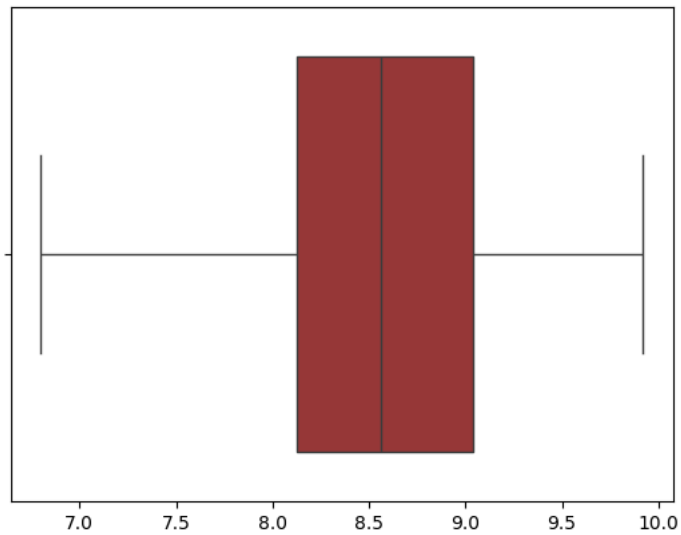

95
100
105
110
115
120
TOEFL Score

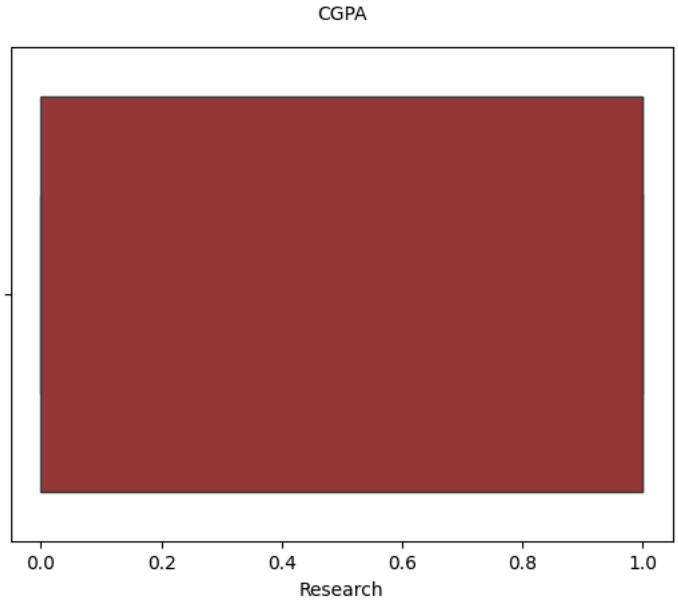


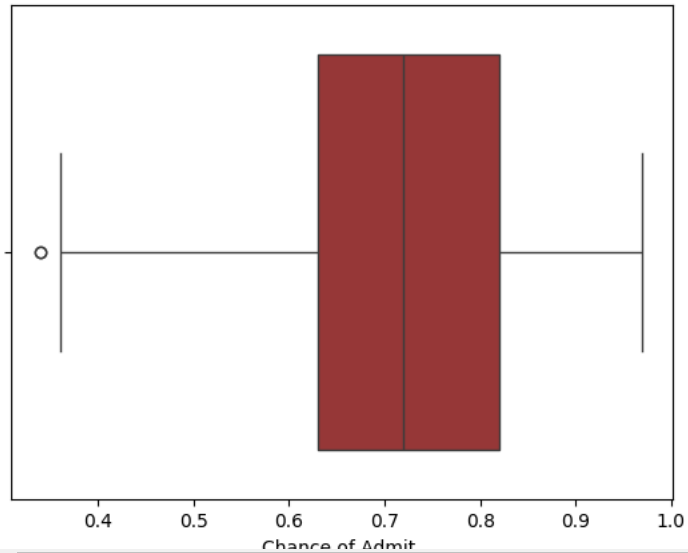

1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
University Rating


1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
SOP


1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
LOR


7.0
7.5
8.0
8.5
9.0
9.5
10.0


CGPA
0.0
0.2
0.4
0.6
0.8
1.0
Research


0
0.4
0.5
0.6
0.7
0.8
0.9
1.0

```
df.describe()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	499.000000	499.000000	499.000000	499.000000	499.000000	499.000000	499.000000	499.000000	499.000000
mean	250.304609	316.507014	107.218437	3.118236	3.378758	3.488978	8.578918	0.561122	0.722345
std	144.560681	11.279288	6.059145	1.140725	0.986266	0.919653	0.602874	0.496748	0.140632
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.500000	6.800000	0.000000	0.340000
25%	125.500000	308.000000	103.000000	2.000000	2.500000	3.000000	8.130000	0.000000	0.630000
50%	250.000000	317.000000	107.000000	3.000000	3.500000	3.500000	8.560000	1.000000	0.720000
75%	375.500000	325.000000	112.000000	4.000000	4.000000	4.000000	9.040000	1.000000	0.820000
max	500.000000	340.000000	120.000000	5.000000	5.000000	5.000000	9.920000	1.000000	0.970000

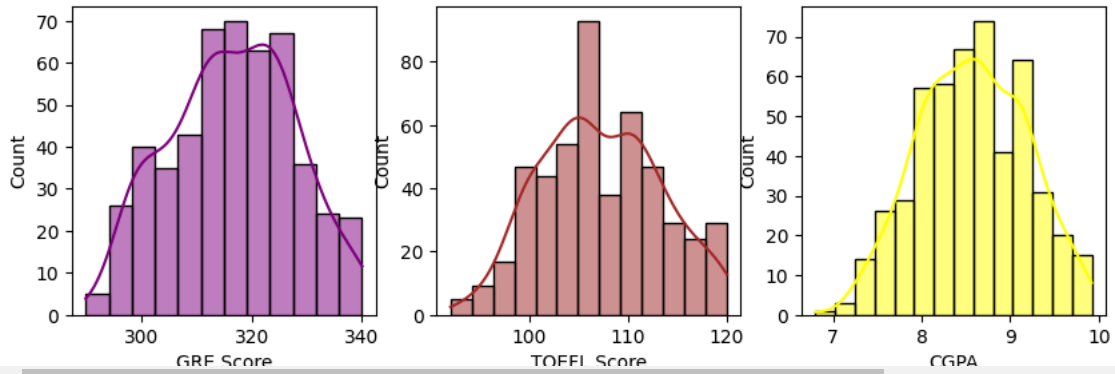
```
df.shape
```

```
(499, 9)
```

Univariate analysis of all continous values

```
fig = plt.figure(figsize=(10,3))
plt.subplot(1,3,1)
plt.xlabel('GRE Score')
sns.histplot(df['GRE Score'], kde=True,color='purple')

plt.subplot(1,3,2)
plt.xlabel('TOEFL Score')
sns.histplot(df['TOEFL Score'], kde=True,color='brown')

plt.subplot(1,3,3)
plt.xlabel('CGPA')
sns.histplot(df['CGPA'], kde=True,color='yellow')
```

```
<Axes: xlabel='CGPA', ylabel='Count'>
```



From the above graph we can see,the average GRE Score is 320.

Average TOFEL Score is 103 to 105

Average CGPA mark is 8.5

Univariate analysis of all Categorical values

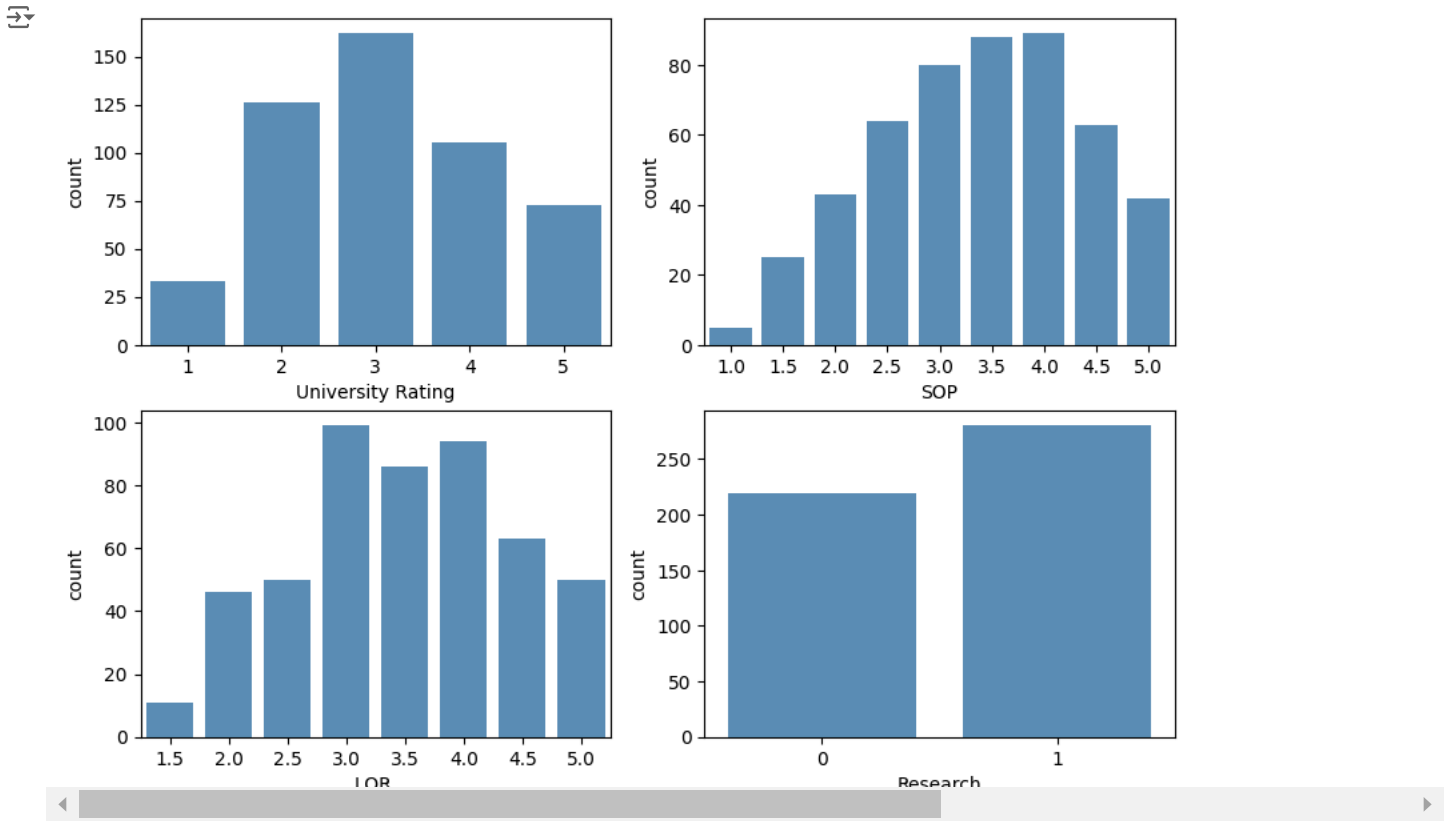
```
df.columns
```

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

```
df_cat=['University Rating', 'SOP','LOR ','Research']
```

```
# countplots for categorical variables
cols, rows = 2, 2
```

```
fig, axs = plt.subplots(rows, cols, figsize=(10, 7))

index = 0
for row in range(rows):
    for col in range(cols):
        sns.countplot(x=df_cat[index], data=df, ax=axs[row, col], alpha=0.8)
        index += 1

plt.show()
```



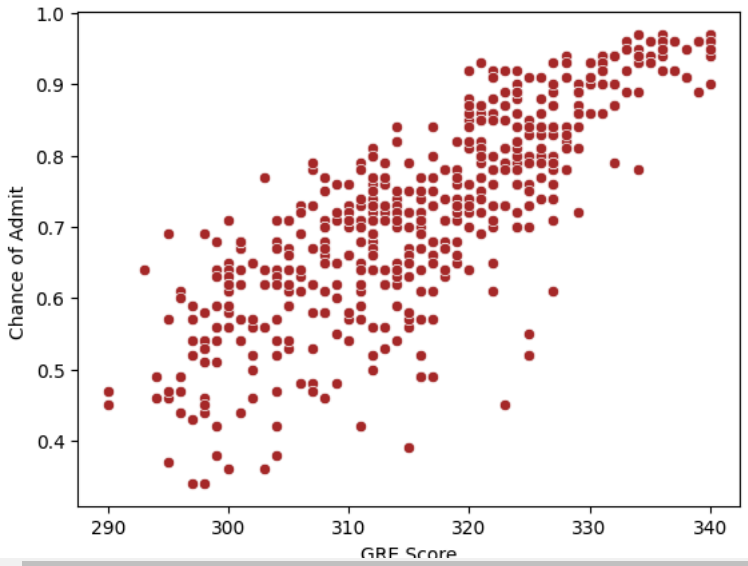
From the above graph we can see the most students are research student.

And have the unversity rating of 5

Bivariate Analysis

```
sns.scatterplot(x='GRE Score', y='Chance of Admit ', data=df,color='brown')
```

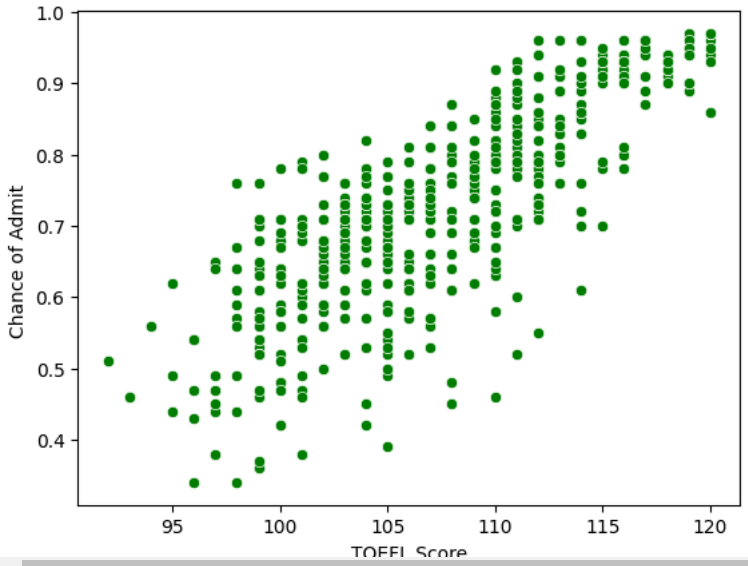
```
<Axes: xlabel='GRE Score', ylabel='Chance of Admit '>
```



From the above scatterplot,we can see as the **GRE score** increase the **change of admit** also increase.

```
sns.scatterplot(x='TOEFL Score', y='Chance of Admit ', data=df,color='green')
```

```
<Axes: xlabel='TOEFL Score', ylabel='Chance of Admit '>
```



From the above scatterplot,we can see as the **TOEFL Score** increase the **change of admit** also increase.

Correlation among independent variables

```
df_corr=df.select_dtypes(include=np.number)
df_corr.head()
```

	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

```
df_corr.drop(['Serial No.'], axis = 1, inplace = True)
df_corr.head()
```

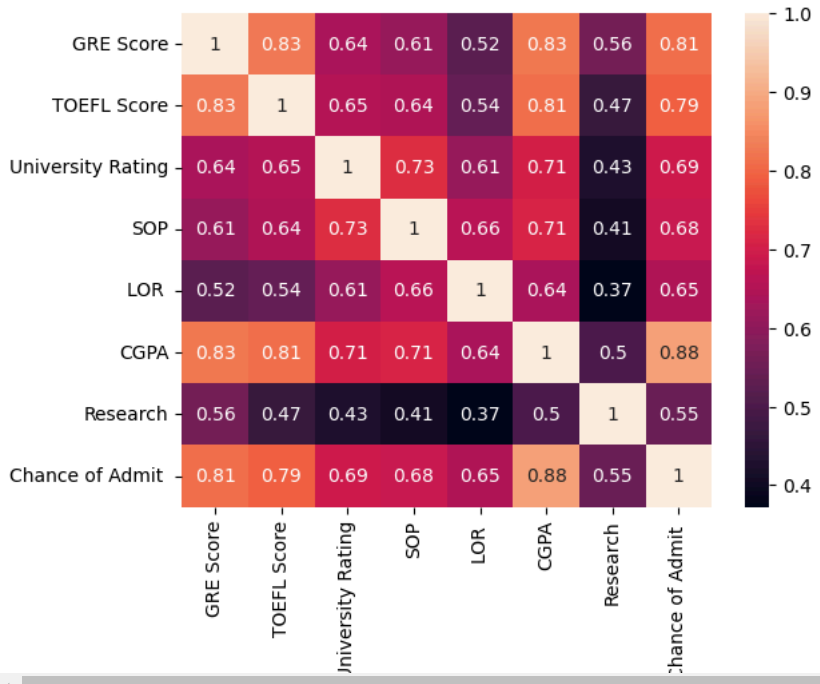
	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65

```
df_corr.corr()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
GRE Score	1.000000	0.827200	0.635376	0.613498	0.524679	0.825878	0.563398	0.810351
TOEFL Score	0.827200	1.000000	0.649799	0.644410	0.541563	0.810574	0.467012	0.792228
University Rating	0.635376	0.649799	1.000000	0.728024	0.608651	0.705254	0.427047	0.690132
SOP	0.613498	0.644410	0.728024	1.000000	0.663707	0.712154	0.408116	0.684137
LOR	0.524679	0.541563	0.608651	0.663707	1.000000	0.637469	0.372526	0.645365
CGPA	0.825878	0.810574	0.705254	0.712154	0.637469	1.000000	0.501311	0.882413
Research	0.563398	0.467012	0.427047	0.408116	0.372526	0.501311	1.000000	0.545871
Chance of Admit	0.810351	0.792228	0.690132	0.684137	0.645365	0.882413	0.545871	1.000000

```
sns.heatmap(df_corr.corr(),annot=True)
```

```
<Axes: >
```



Diagonal values are all 1 because they represent correlation between the same variable

The stronger the colour, the stronger the correlation between the variables Positive corelation:

Positive Correlation: Research students have postive correlation with change of Admit.

CGPA also impacting the change of Admit,but not entirely

Checking Duplicate values in dataframe

```
df.duplicated().sum()
```

```
0
```

Prepare the data for modeling

Perform the train-test split

```
df.head()
```

	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

```
#Since our problem statement is what are the factor influening the admission chance and we have predict the Real values which is giving the change of a


#And we have the independent or the feature varibles and one dependent or Target variable,hence we can used Supervised Algorithm


#We are using Linear Regression Algorithm.


#Getting the feature and target variables
y=df['Chance of Admit ']    # Target variable
x=df.drop(['Chance of Admit ','Serial No.'],axis=1) # Feature Variables


x.shape, y.shape
```

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

Perform Standardisation

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x = pd.DataFrame(scaler.fit_transform(x), columns = x.columns)
x.head()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	0.94	0.928571	0.75	0.875	0.875	0.913462	1.0
1	0.68	0.535714	0.75	0.750	0.875	0.663462	1.0
2	0.52	0.428571	0.50	0.500	0.625	0.384615	1.0
3	0.64	0.642857	0.50	0.625	0.375	0.599359	1.0
4	0.48	0.392857	0.25	0.250	0.500	0.451923	0.0

Next steps: Generate code with x | View recommended plots | New interactive sheet

```
# 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,)
```

```
X_sm = sm.add_constant(x_train)  # Statmodels default is without intercept, to add intercept we need to add constant.

model = sm.OLS(y_train, X_sm)
results = model.fit()

# Print the summary statistics 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:              Wed, 11 Sep 2024   Prob (F-statistic):          3.41e-142
Time:                      10:43:01   Log-Likelihood:                561.91
No. Observations:               400   AIC:                            -1108.
Df Residuals:                   392   BIC:                            -1076.
Df Model:                         7
Covariance Type:            nonrobust
====================================================================================
                       coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------
const                0.3470      0.010     33.788      0.000       0.327       0.367
GRE Score            0.1217      0.029      4.196      0.000       0.065       0.179
TOEFL Score          0.0839      0.026      3.174      0.002       0.032       0.136
University Rating    0.0103      0.017      0.611      0.541      -0.023       0.043
SOP                  0.0073      0.020      0.357      0.721      -0.033       0.047
LOR                  0.0690      0.018      3.761      0.000       0.033       0.105
CGPA                 0.3511      0.034     10.444      0.000       0.285       0.417
Research             0.0240      0.007      3.231      0.001       0.009       0.039
==============================================================================
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.                         23.6
==============================================================================

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

Base Model: Linear Regression

```
from sklearn.linear_model import LinearRegression
model_lr = LinearRegression()
model_lr.fit(x_train, y_train)
```

```
LinearRegression
LinearRegression()
```

```
# Bias Term of the Model
model_lr.intercept_
```

```
0.3469650583945579
```

```
# Predicting values for the training and test data
y_pred_train = model_lr.predict(x_train)
y_pred_test = model_lr.predict(x_test)
```

```
# Evaluating the model using multiple loss functions
def model_evaluation(y_actual, y_forecast, model):
  n = len(y_actual) #number of datapoints
  if len(model.coef_.shape)==1:
    p = len(model.coef_) #dimension of the vector
  else:
    p = len(model.coef_[0]) #dimension of the vector

  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 print(f"MAE: {MAE}\nRMSE: {RMSE}\nR2 Score: {r2}\nAdjusted R2: {adj_r2}")


# Metrics for training data
model_evaluation(y_train.values, y_pred_train, model_lr)
```

```
MAE: 0.04
RMSE: 0.06
R2 Score: 0.82
Adjusted R2: 0.82
```

```
#Metrics for test data
model_evaluation(y_test.values, y_pred_test, model_lr)
```

```
MAE: 0.04
RMSE: 0.06
R2 Score: 0.82
Adjusted R2: 0.81
```

Since there is no difference in the loss scores of training and test data, we can conclude that there is no overfitting of the model

Mean Absolute Error of 0.04 shows that on an average, the absolute difference between the actual and predicted values of chance of admit is 4%

Root Mean Square Error of 0.06 means that on an average, the root of squared difference between the actual and predicted values is 6%

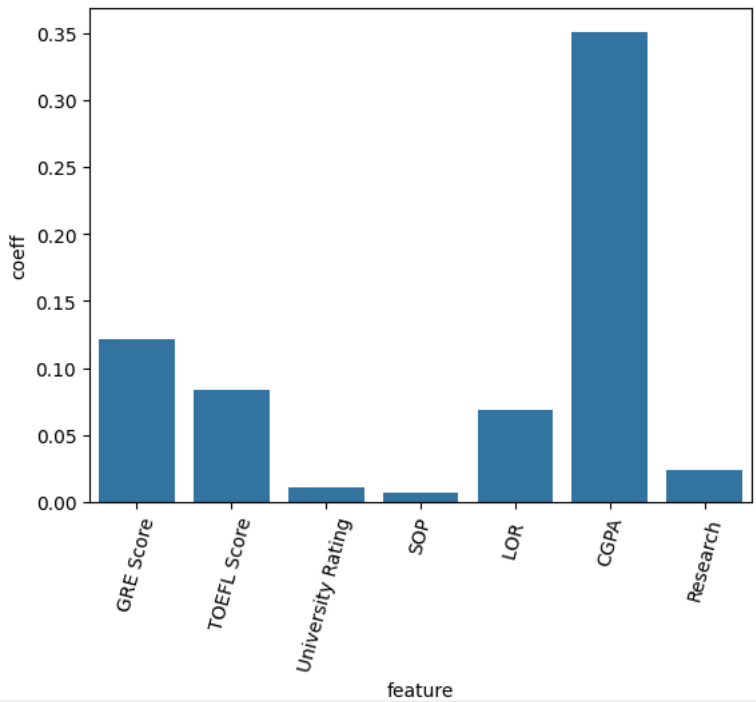
R2 Score of 0.82 means that our model captures 82% variance in the data

Adjusted R2 is an extension of R2 which shows how the number of features used changes the accuracy of the prediction

Model Coefficients

```
import seaborn as sns

imp = pd.DataFrame(list(zip(x_train.columns,np.abs(model_lr.coef_))),
                   columns=['feature', 'coeff'])
sns.barplot(x='feature', y='coeff', data=imp)
plt.xticks(rotation = 75)
```

```
([0, 1, 2, 3, 4, 5, 6],
 [Text(0, 0, 'GRE Score'),
  Text(1, 0, 'TOEFL Score'),
  Text(2, 0, 'University Rating'),
  Text(3, 0, 'SOP'),
  Text(4, 0, 'LOR '),
  Text(5, 0, 'CGPA'),
  Text(6, 0, 'Research')])
```



From the above graph we can see CGPA & GRE scores have the highest weight.

SOP, University rating, and research have the lowest weights

We see that almost all the variables (excluding research) have a very high level of colinearity. This was also observed from the correlation heatmap which showed strong positive correlation between GRE score, TOEFL score and CGPA.

```
x_test.columns[np.argmax(np.abs(model_lr.coef_))]
```

Testing Assumptions of Linear Regression Model

Regression is a **parametric** approach, which means that it makes **assumptions** about the data for analysis. For successful regression analysis, it's essential to validate the following assumptions.

1. Multicollinearity check by VIF score

As multicollinearity makes it difficult to find out which **variable** is **contributing** towards the **prediction** of the response variable, it leads one to conclude incorrectly, the effects of a variable on the target variable. Though it does not affect the precision of the model predictions, it is essential to properly detect and deal with the multicollinearity present in the model, as random removal of any of these correlated variables from the model causes the coefficient values to swing wildly and even change signs.

Multicollinearity can be detected using the following methods.

Variance Inflation Factor (VIF)

VIF explains the relationship of one independent variable with all the other independent variables.

The common heuristic followed for the VIF values is if VIF > 10 then the value is high and it should be dropped.

And if the VIF=5 then it may be valid but should be inspected first.

If VIF < 5, then it is considered a good VIF value

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 predictor variables.

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


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

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


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

	feature	VIF
0	GRE Score	1308.061089
1	TOEFL Score	1215.951898
2	University Rating	20.933361
3	SOP	35.265006
4	LOR	30.911476
5	CGPA	950.817985
6	Research	2.860403

Next steps: Generate code with res | View recommended plots | New interactive sheet

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

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

Next steps: Generate code with res | View recommended plots | New interactive sheet

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

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

Next steps: Generate code with res | View recommended plots | New interactive sheet

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

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

Next steps: Generate code with res | View recommended plots | New interactive sheet

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

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

Next steps: Generate code with res | View recommended plots | New interactive sheet

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

	feature	VIF
0	CGPA	2.455008
1	Research	2.455008

Next steps: Generate code with res | View recommended plots | New interactive sheet

Run the linear reression model with 2 features alone in test data

```
#Getting the feature and target variables
y=df['Chance of Admit ']    # Target variable
x_V=df[['CGPA', 'Research']] # Feature Variables


# now again train the model with these only two features
sc = StandardScaler()
x_scaled = sc.fit_transform(x_V)

X_train_v, X_test_v, y_train_v, y_test_v = train_test_split(x_scaled, y, test_size=0.3, random_state=1)


x_scaled.shape
```

```
(500, 2)
```

```
from sklearn.linear_model import LinearRegression
model_lrv = LinearRegression()
model_lrv.fit(X_train_v, y_train_v)
```

```
LinearRegression
LinearRegression()
```

```
# Predicting values for the training and test data
y_pred_train_v = model_lrv.predict(X_train_v)
y_pred_test_v = model_lrv.predict(X_test_v)


# Evaluating the model using multiple loss functions
def model_evaluation(y_actual, y_forecast, model):
  n = len(y_actual) #number of datapoints
  if len(model.coef_.shape)==1:
    p = len(model.coef_) #dimension of the vector
  else:
    p = len(model.coef_[0]) #dimension of the vector

  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 print(f"MAE: {MAE}\nRMSE: {RMSE}\nR2 Score: {r2}\nAdjusted R2: {adj_r2}")
```

```
#Metrics for test data after VIF and keepinf 2 independent variables,
model_evaluation(y_test_v.values, y_pred_test_v, model_lrv)
```

```
MAE: 0.05
RMSE: 0.07
R2 Score: 0.81
Adjusted R2: 0.81
```

We can see there is no prediction change even after removing all irrelvant dimentions

lasso And Ridge Regression Model

```
lasso_model = Lasso(alpha=0.001)  # Alpha is the regularization strength
ridge_model = Ridge(alpha=0.001)    # Alpha is the regularization strength

# Fit the models to the training data
lasso_model.fit(X_train_v, y_train_v)
ridge_model.fit(X_train_v, y_train_v)

# Predicting values for train and test data

y_train_ridge = ridge_model.predict(X_train_v)
y_test_ridge = ridge_model.predict(X_test_v)

y_train_lasso = lasso_model.predict(X_train_v)
y_test_lasso = lasso_model.predict(X_test_v)


print('\n\nLasso Regression Test Accuracy\n')
model_evaluation(y_test_v.values, y_test_lasso, lasso_model)
```

```
Lasso Regression Test Accuracy

MAE: 0.05
RMSE: 0.07
R2 Score: 0.81
Adjusted R2: 0.81
```

```
print('\n\nRidge Regression Test Accuracy\n')
model_evaluation(y_test_v.values, y_test_ridge, ridge_model)
```

```
Ridge Regression Test Accuracy

MAE: 0.05
RMSE: 0.07
R2 Score: 0.81
Adjusted R2: 0.81
```

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

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.

```
residuals = y_test.values - y_pred_test
residuals.reshape((-1,))
print('Mean of Residuals: ', residuals.mean())
```

```
Mean of Residuals:  -0.005453623717661311
```

```
residuals.reshape((-1,))
```

```
array([ 0.01542527,  0.04481873, -0.18265986,  0.06263032, -0.07588282,
        0.02793439, -0.00459746, -0.07850923, -0.14378728,  0.01258502,
       -0.27193204,  0.04410882, -0.04632227,  0.00322832,  0.0658482 ,
       -0.01979177,  0.03068058,  0.05272705,  0.00660424, -0.00155958,
        0.00451832, -0.01305833, -0.04232308, -0.00506004, -0.04027648,
        0.07751346, -0.01840363, -0.02741471,  0.01222989, -0.01343096,
        0.03466965, -0.01004034, -0.01474225,  0.02883712,  0.06286984,
       -0.0271816 ,  0.05576161, -0.01718977,  0.03635733, -0.15057279,
       -0.01036982, -0.13495143,  0.02924609,  0.03169578, -0.04112903,
        0.06206238, -0.0383926 ,  0.00097235, -0.03591651,  0.01429352,
        0.04194031,  0.11633762,  0.06640289,  0.01341071, -0.0136427 ,
        0.04253186,  0.00399604, -0.10916386,  0.02743018, -0.0612903 ,
        0.05049241,  0.01968812,  0.02103279,  0.11796194,  0.01914132,
       -0.10866369,  0.0232453 , -0.03783006,  0.04168193,  0.00332608,
        0.0223182 ,  0.00412279,  0.00847434,  0.01728981,  0.0257047 ,
       -0.03084583, -0.01627224, -0.08373777,  0.05270426, -0.09142139,
        0.03041964,  0.07187084,  0.03306828,  0.0128929 , -0.00411583,
       -0.0268525 ,  0.10555545, -0.01875671, -0.00899999, -0.00320147,
       -0.0128225 , -0.00802002, -0.00576583,  0.01943511, -0.04694407,
       -0.01563138,  0.03335741, -0.06598826, -0.06469446, -0.10865736])
```

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

3.Linearity of Variables

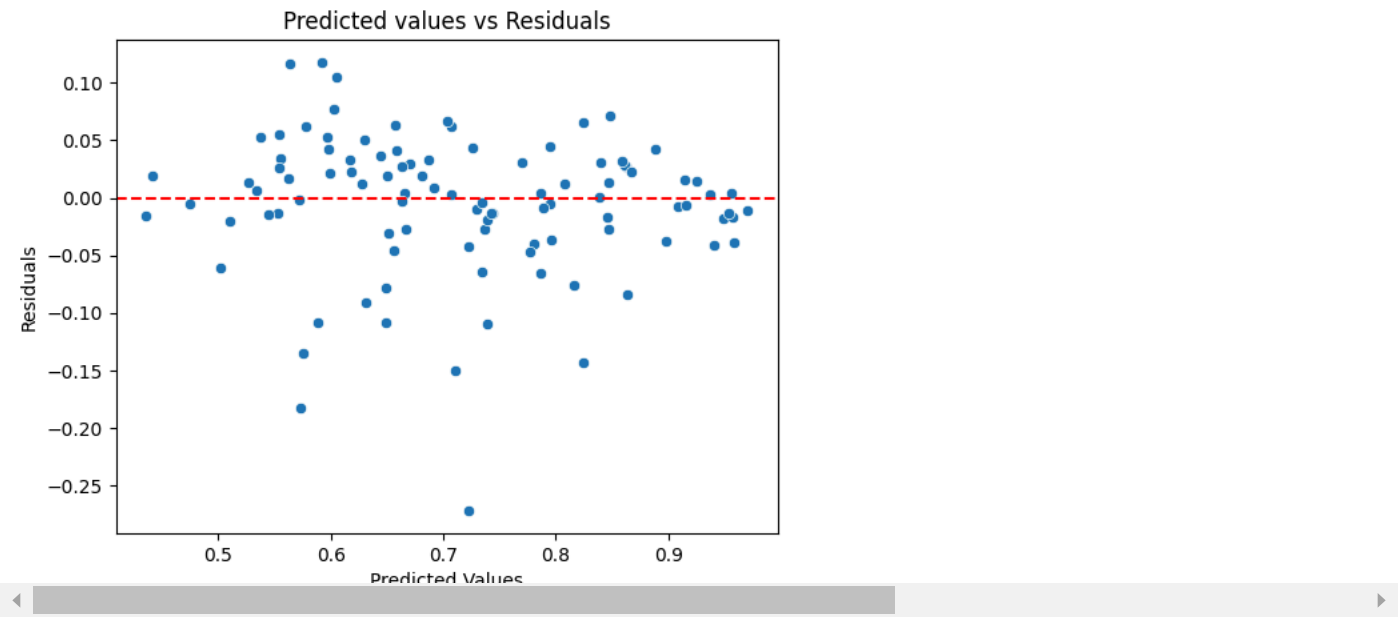
There needs to be a linear relationship between the dependent variable and independent variables.

It means that the effect of the independent variables on the dependent variable is constant across different levels of 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.

```
errors = y_test.values - y_pred_test
```

```
sns.scatterplot(x=y_pred_test,y=errors)
plt.title('Residual Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title("Predicted values vs Residuals")
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```

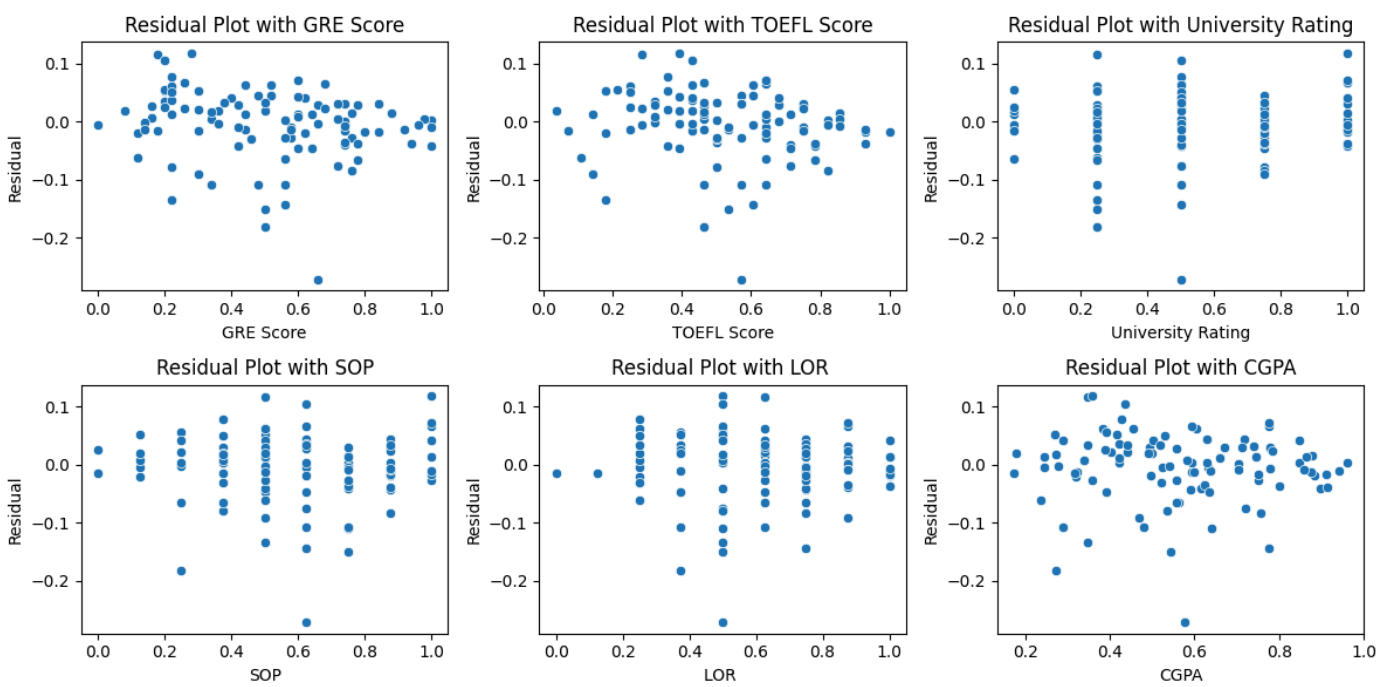


Since the residual plot shows no clear pattern or trend in residuals, we can conclude that linearity of variables exists

4.Homoscedasticity

scatterplot of residuals against predicted values

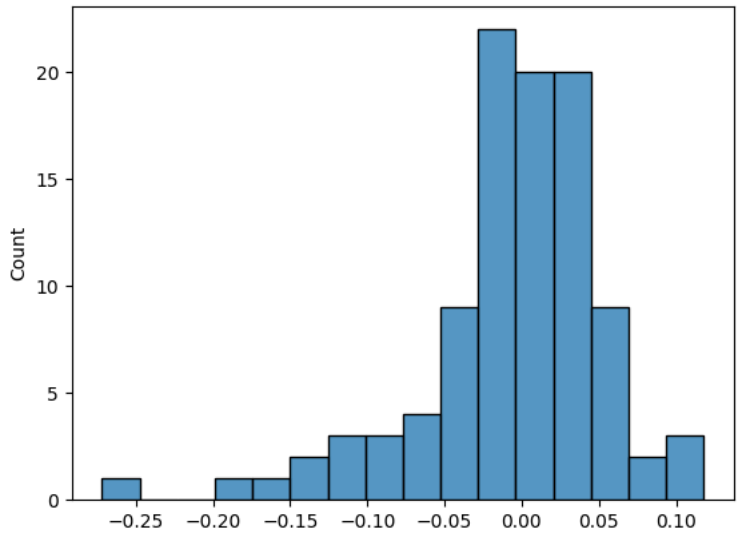
```
plt.figure(figsize=(12,6))
i=1
for col in x_test.columns[:-1]:
  ax = plt.subplot(2,3,i)
  sns.scatterplot(x=x_test[col].values.reshape((-1,)), y=residuals.reshape((-1,)))
  plt.title(f'Residual Plot with {col}')
  plt.xlabel(col)
  plt.ylabel('Residual')
  i+=1

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



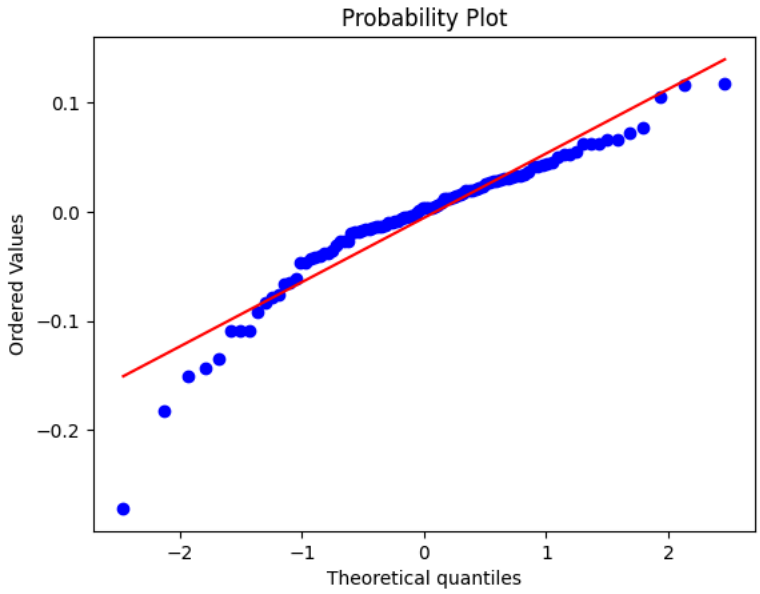
Since we do not see any significant change in the spread of residuals with respect to change in independent variables, we can conclude that homoscedasticity is met.

5.Normality of residuals

```
sns.histplot(residuals)
plt.show()
```




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



Insights

Multicollinearity is present in the data.

After removing collinear features there are only two variables which are important in making predictions for the target variables.

Indepedent variables are linearly correlated with dependent variables.

Recommendations

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

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

Following are the final model results on the test data:

RMSE: 0.07

MAE: 0.05

R2_score: 0.81

Adjusted_R2: 0.81