Colab notebook link:

https://colab.research.google.com/drive/1dqX-BZA38-abO8Fp0bcTWl1ox6VXBdNw?usp=sharing

---

Insights:

Analyzing the dataset:

- There are total 500 rows and 9 columns.
- Columns are: ['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA', 'Research', 'Chance of Admit ']
- Data type of columns: 5 integer type columns, 4 float type columns.

Basic data cleaning and exploration:

Conversion of Categorical Attributes to Category:

- Further analyzing the object type column, we can find out that 4 columns ('University Rating', 'SOP', 'LOR', and 'Research') are category type columns. So, we are converting these four columns into category type columns.

Conversion of Float64 to Float32:

- Converting the types of two columns ('CGPA' and 'Chance of Admit') from float64 to float32 type columns. It reduces the size.

Removing the Unique Row Identifier:

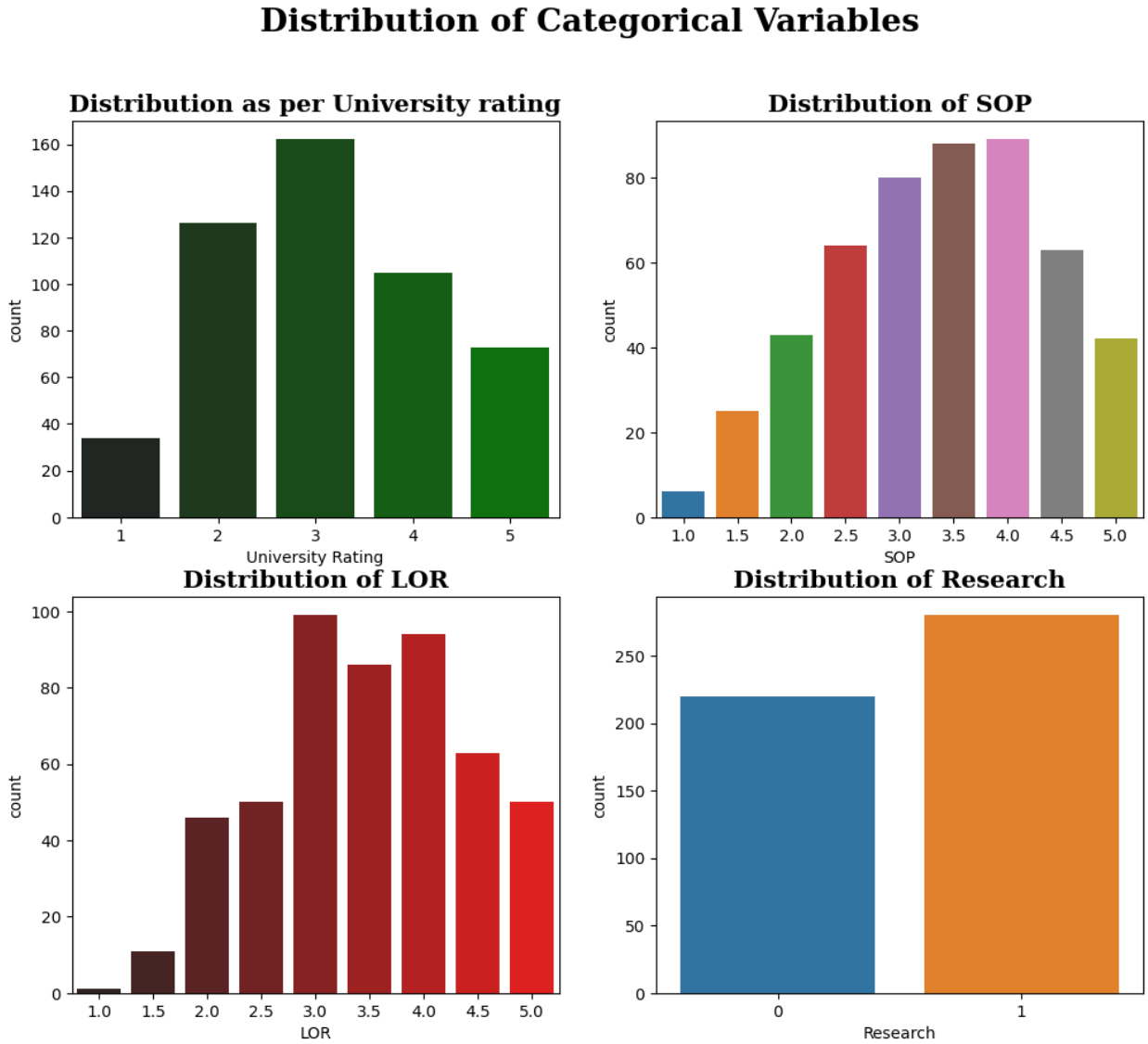
- Removing the unique row identifier means 'Serial No.' column because we don't want our model to build some understanding based on row numbers.

Missing value & Duplicates:

- There are no missing values.
- There are no duplicates in the dataset.

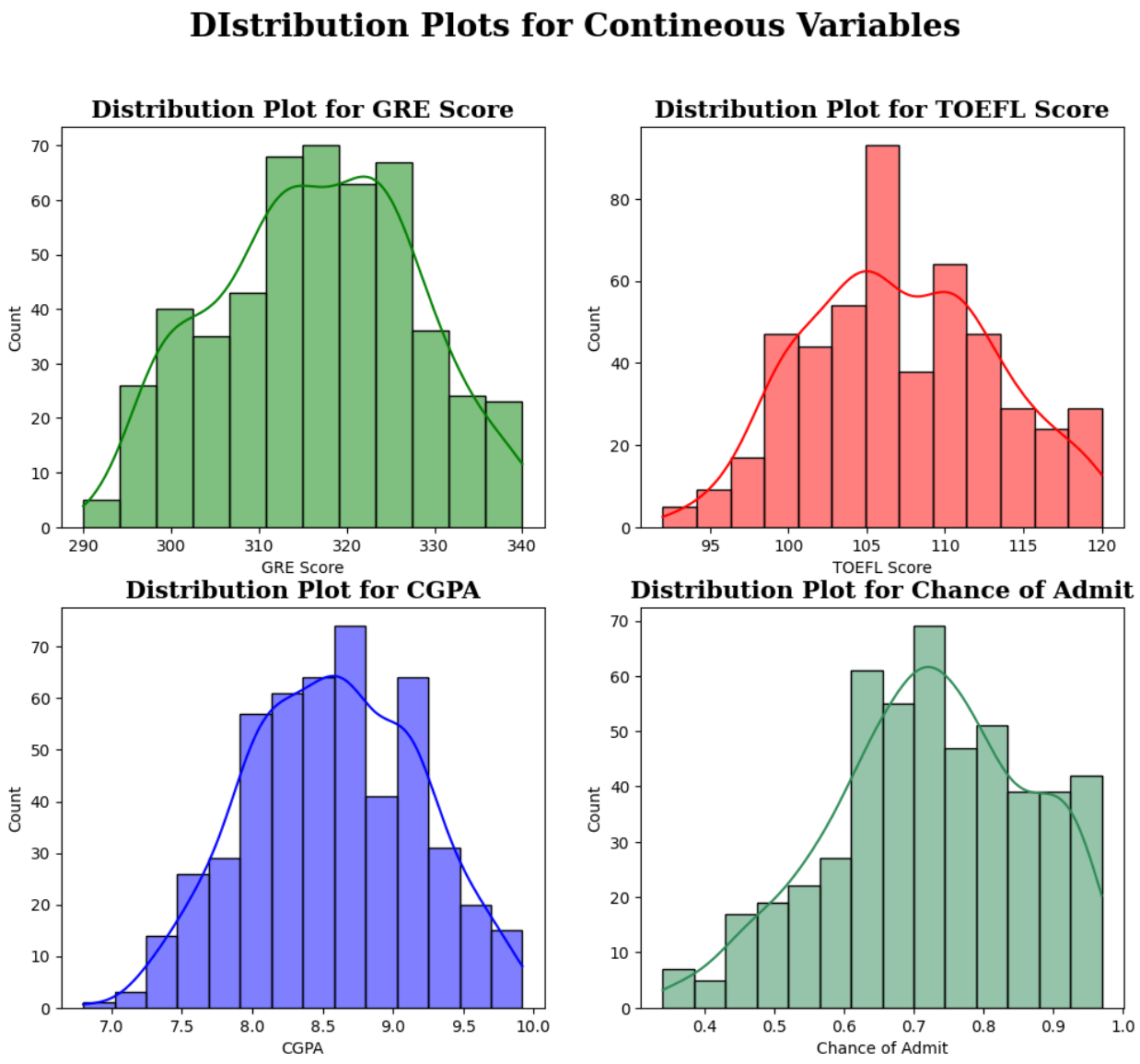
Uni-variate analysis:

Distribution Plot for Categorical Variables:



- **University rating:** Most frequent university rating is 3.
- **SOP:** Most frequent SOP is 4, followed by 3.5.
- **LOR:** Most frequent LOR is 3, followed by 4.
- **Research:** Most frequent research is 1 that means most of them have research experience.

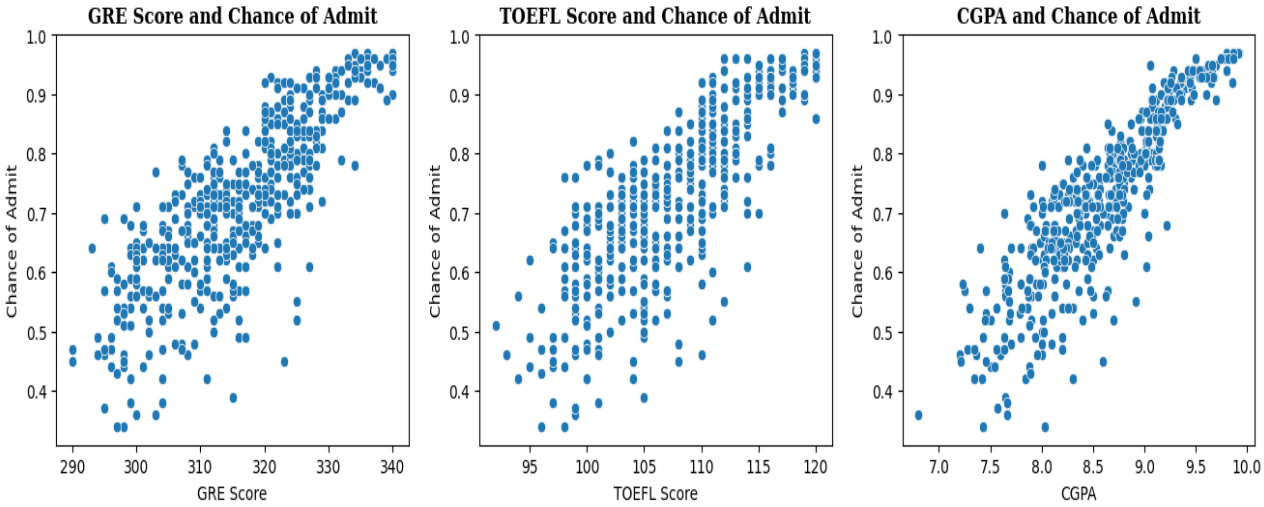
Distribution Plot for Continuous Variables:



- **GRE Score:** GRE scores are between 290 and 340, and maximum students scoring in the range 310 – 330.
- **TOEFL Score:** TOEFL scores are between 92 and 120, and maximum students scoring in the range 105-110.
- **CGPA:** CGPA ranges between 6.5 and 10, and maximum students scoring around 8.5.
- **Chance of Admit:** Chance of admit of maximum students in the range 0.7-0.75.

Bi-variate Analysis:

Relationship between GRE Score, TOEFL Score and CGPA with Chance of Admit:



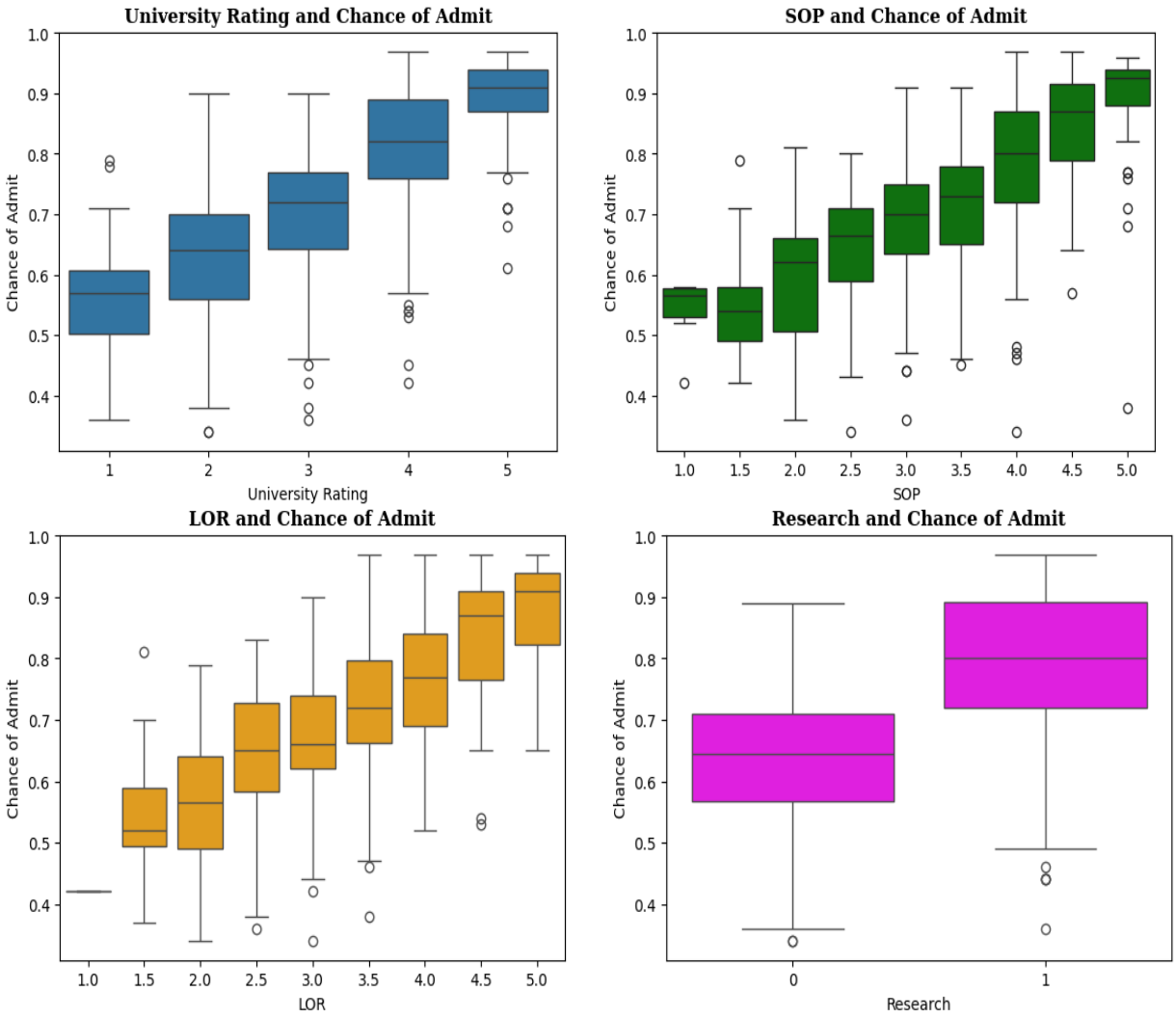
From the above Scatter plot distribution, we can analyze the following relationship:

- **GRE score and Chance of Admit:** Clearly, we can see that there is a linear correlation between GRE score and Chance of admit. If students get maximum GRE score then chances of admission is maximum.

- **TOEFL score and Chance of Admit:** From the above scatter plot diagram we also can say that there is linear correlation between TOEFL score and Chance of Admit.

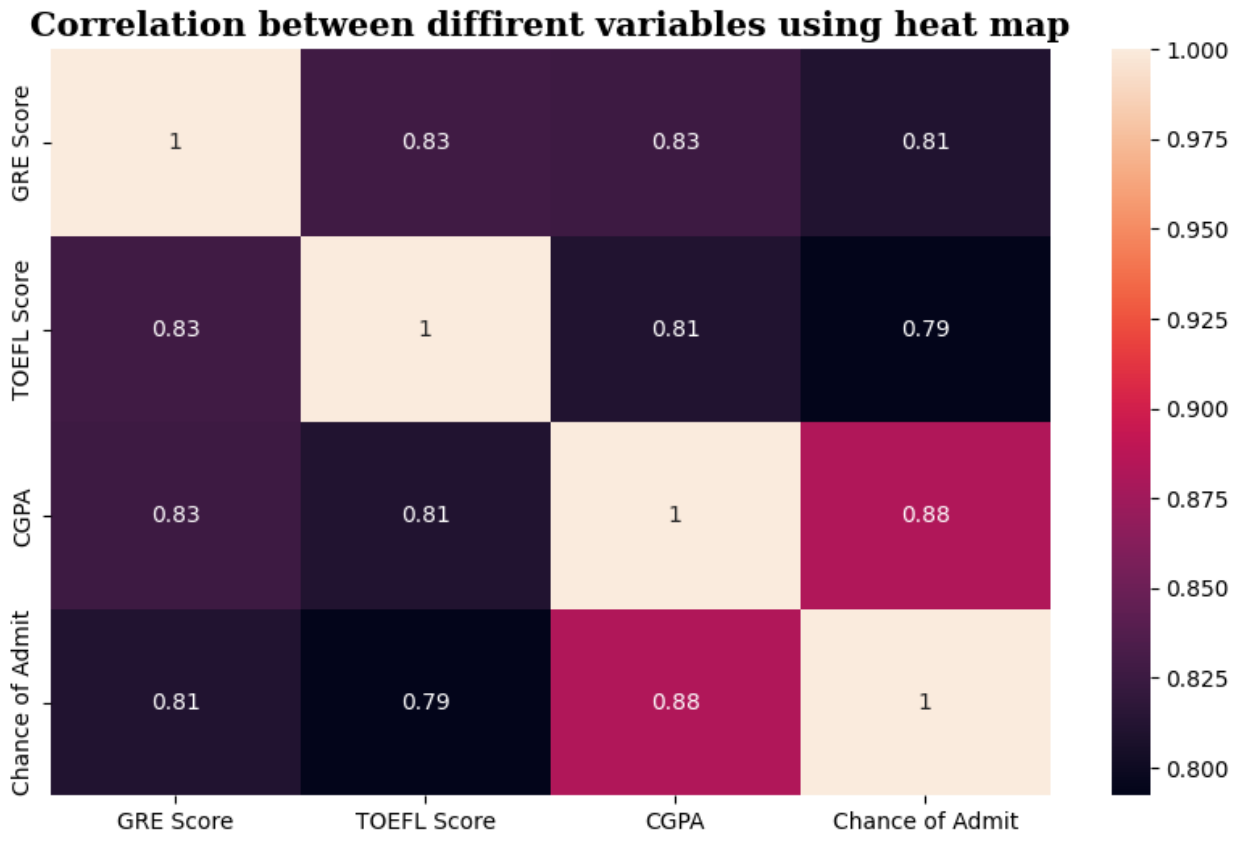
- **CGPA and Chance of Admit:** The linear correlation between CGPA and Chance of Admit is strong.

Relationship between Categorical columns and Chance of Admit:



- **University rating and Chance of Admit:** From the above diagram, we can say that with the increase of university rating the chances of admission also increase.
- **SOP and Chance of Admit:** From the above diagram, we can say that with the increase of SOP, Chance of admit also increase.
- **LOR and Chance of Admit:** From the above diagram, we can say that with the increase of LOR, Chance of admit also increase.
- **Research and Chance of Admit:** Students, who have research experience is high chance of admission as compare to other students, who don't have research experience.

Multi-variate Analysis:



- From the above heat map, we can observe that, all the exam scores (GRE score, TOEFL score and CGPA) is strongly positive correlation with chance of admit.
- Even, they are also strongly correlated amongst themselves.

Outlier Detection:

Outlier detection of Continuous variables by IQR method:

```
#Detecting IQR
def outlier_by_IQR(i):
  Q1 = np.quantile(df[i], 0.25)
  Q3 = np.quantile(df[i], 0.75)
  IQR = Q3 - Q1
  Lower_outlier = Q1 - 1.5*IQR
  Higher_outlier = Q3 + 1.5*IQR
  outliers = df.loc[(df[i] < Lower_outlier) | (df[i] > Higher_outlier)]
  print('Column :', i)
  print(f'Q1 : {Q1}')
  print(f'Q3 : {Q3}')
  print(f'IQR : {IQR}')
  print(f'Lower outlier : {Lower_outlier}')
  print(f'Upper outlier : {Higher_outlier}')
  print(f'Number of outliers : {outliers.shape[0]}')
  print('-----------------------------------')
```

```
Column : GRE Score
Q1 : 308.0
Q3 : 325.0
IQR : 17.0
Lower outlier : 282.5
Upper outlier : 350.5
Number of outliers : 0
-----------------------------------
Column : TOEFL Score
Q1 : 103.0
Q3 : 112.0
IQR : 9.0
Lower outlier : 89.5
Upper outlier : 125.5
Number of outliers : 0
-----------------------------------
```

```
Column : CGPA
Q1 : 8.127500057220459
Q3 : 9.039999961853027
IQR : 0.9124999046325684
Lower outlier : 6.7587502002716064
Upper outlier : 10.40874981880188
Number of outliers : 0
-----------------------------------
Column : Chance of Admit
Q1 : 0.6299999952316284
Q3 : 0.8199999928474426
IQR : 0.1899999976158142
Lower outlier : 0.3449999988079071
Upper outlier : 1.104999989271164
Number of outliers : 2
-----------------------------------
```

- There is no outlier in GRE score, TOEFL score and CGPA.
- 2 outliers found in Chance of Admit.

Model Building:

Data preparation for Model Building:

```
x = df.drop('Chance of Admit', axis=1)
y = df['Chance of Admit']

print("Shape of x: ", x.shape)
print("Shape of y: ", y.shape)
```

```
Shape of x:  (500, 7)
Shape of y:  (500,)
```

- We observed from our previous analysis that 'Chance of Admit' column is our target column. So, we differentiate our features columns and target column by x and y respectively.

```
#split the data in test and train data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42)
```

- We split our data into training data and test data by 80% and 20%.

```
print("Shape of x_train: ", x_train.shape)
print("Shape of y_train: ", y_train.shape)
```

```
Shape of x_train:  (400, 7)
Shape of y_train:  (400,)
```

```
print("Shape of x_test: ", x_test.shape)
print("Shape of y_test: ", y_test.shape)
```

```
Shape of x_test:  (100, 7)
Shape of y_test:  (100,)
```

- The shape of training data and test data are as above.

Transforming Categorical columns by Label Encoding:

```
categorical_columns
```

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

```
#Transforming categorical columns in the train data and test data
Label_encoder = LabelEncoder()

#encode label in column-wise for train data
for i in categorical_columns:
  x_train[i] = Label_encoder.fit_transform(x_train[i])

#encode label in column-wise for test data
for i in categorical_columns:
  x_test[i] = Label_encoder.fit_transform(x_test[i])
```

- Using Label encoding technique, converted categorical columns into numerical ones so that they can be fitted by machine learning models which only take numerical data. It is an important pre-processing step in a machine learning project.

Normalizing data by MinMaxScaler:

```
scaler = MinMaxScaler()

#Normalizing train data
x_train = pd.DataFrame(scaler.fit_transform(x_train), columns = x_train.columns)

#Normalizing test data
x_test = pd.DataFrame(scaler.fit_transform(x_test), columns = x_test.columns)
```

- Scaling or normalizing training data and testing data by MinMaxScaler technique. It scales the values to a specific value range without changing the shape of the original distribution.

Linear Regression:

```
model_LR = LinearRegression()

#fit the model in training data
model_LR.fit(x_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

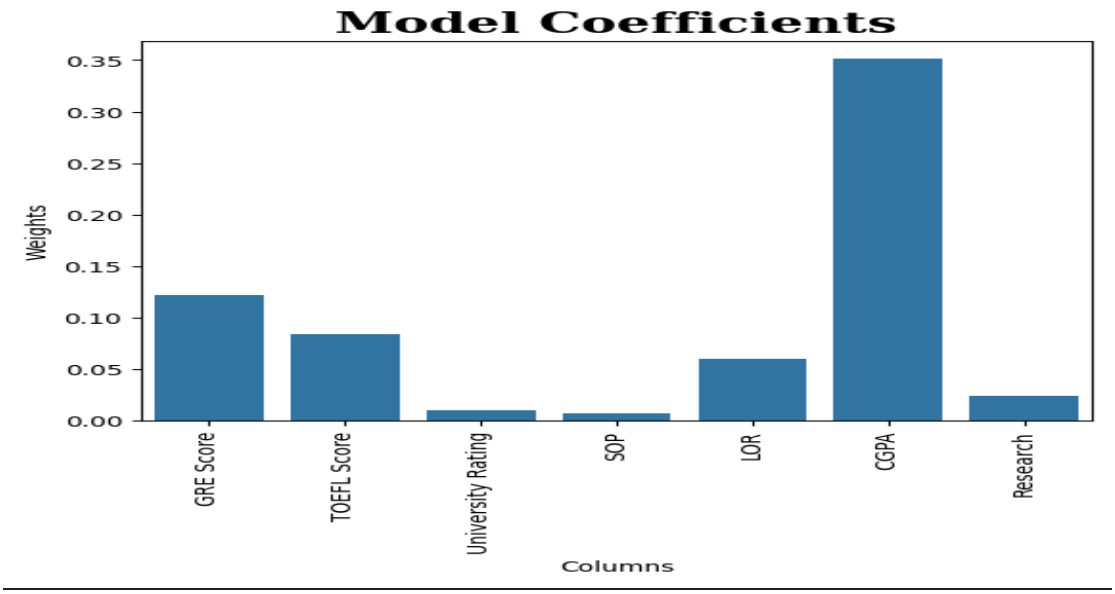
- ***Model Coefficients:*** The value of weights of features and interception are as follows:

```
#bias
W0 = model_LR.intercept_
print(f"Intercept: {W0}")
```

```
Intercept: 0.35558406163802325
```

```
#model coefficients
weights = pd.DataFrame({"Column" : x.columns, "Weight" : model_LR.coef_})
print(weights)
```

```
              Column    Weight
0          GRE Score  0.121722
1        TOEFL Score  0.083884
2  University Rating  0.010275
3                SOP  0.007255
4                LOR  0.060333
5               CGPA  0.351085
6           Research  0.024027
```



- CGPA have the highest weight, followed by GRE Score and TOEFL Score.
- SOP, University rating and Research have the lowest weight values.

Model Performance Evaluation:

```
#check the training data
print('Linear Regression Training Model\n')
model_evaluation(y_train, y_train_LR, model_LR, n = x_train.shape[0], d = x_train.shape[1])
#check the test data
print('\nLinear Regression Test Model\n')
model_evaluation(y_test, y_test_LR, model_LR, n = x_test.shape[0], d = x_test.shape[1])
```

```
Linear Regression Training Model

MAE: 0.04
MSE: 0.004
RMSE: 0.06
R2 score: 0.82
Adjusted R2: 0.82

Linear Regression Test Model

MAE: 0.05
MSE: 0.004
RMSE: 0.07
R2 score: 0.79
Adjusted R2: 0.77
```

- Although there is a small difference in the loss scores of training and test data, we can neglect it and consider that the model is not over fitting.

- Mean absolute error of 0.04 for training data shows that on an average, the difference between the actual and predicted values of chance of admit is 4%.
- Mean squared error of 0.004 represents that on an average, the squared difference between the actual and predicted values is 0.4%.
- Root mean squared error of 0.06 for training data represents that on an average, the root of squared difference between the actual and predicted values is 6%.
- R2 score of 0.82 for training data represents that the proportion of the variance in the dependent variables is 82%.
- Adjusted R2 is a modified version R2 and it is adjusted for the number of independent variables in the model.

Ridge and Lasso Regression:

```
#prediction for training and test data
y_train_R = model_R.predict(x_train)
y_test_R = model_R.predict(x_test)

y_train_L = model_L.predict(x_train)
y_test_L = model_L.predict(x_test)
```

```
# Evaluating Model Performance
print('Ridge Regression Training Model\n')
model_evaluation(y_train, y_train_R, model_R, n = x_train.shape[0], d=x_train.shape[1])
print('\n\nRidge Regression Test Model\n')
model_evaluation(y_test, y_test_R, model_R, n = x_test.shape[0], d=x_test.shape[1])
print('\n\nLasso Regression Training Model\n')
model_evaluation(y_train, y_train_L, model_L, n = x_train.shape[0], d=x_train.shape[1])
print('\n\nLasso Regression Test Model\n')
model_evaluation(y_test, y_test_L, model_L, n = x_test.shape[0], d=x_test.shape[1])
```

```
Ridge Regression Training Model

MAE: 0.04
MSE: 0.004
RMSE: 0.06
R2 score: 0.82
Adjusted R2: 0.82
```

```
Lasso Regression Training Model

MAE: 0.11
MSE: 0.02
RMSE: 0.14
R2 score: -0.0
Adjusted R2: -0.02
```

```
Ridge Regression Test Model

MAE: 0.05
MSE: 0.004
RMSE: 0.06
R2 score: 0.8
Adjusted R2: 0.79
```

```
Lasso Regression Test Model

MAE: 0.12
MSE: 0.021
RMSE: 0.14
R2 score: -0.01
Adjusted R2: -0.08
```

- Ridge and Lasso regression are both regularization techniques used to prevent overfitting in linear regression models. They work by adding a penalty term to the cost function, which helps to control the complexity of the model by shrinking the coefficient values.
- Ridge regression is suitable when dealing with multicollinearity, as it will shrink correlated variables together. Lasso regression, however, can select one variable from a set of highly correlated variables and make the others zero.
- From the above performance test, we can say that, linear regression and ridge regression have similar values but lasso regression has not performed well for both training and test data.

Linear Regression from Statsmodel library:

```
#add the constant term
x_sm = sm.add_constant(x_train)

#performing the oridinary least squares regression and fitting the model
results = sm.OLS(y_train, x_sm).fit()

# statstical summary of the model
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.821
Model:                            OLS   Adj. R-squared:                  0.818
Method:                 Least Squares   F-statistic:                     257.0
Date:                Tue, 13 Feb 2024   Prob (F-statistic):          3.41e-142
Time:                        21:39:54   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               0.3556      0.010     36.366      0.000       0.336       0.375
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                 0.0603      0.016      3.761      0.000       0.029       0.092
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.                         23.4
==============================================================================

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

The method we will use to create linear regression models in the statsmodel library is OLS.

- ***Ordinary Least Squares (OLS):***
  - It is a common method for fitting linear models to data. It minimizes the sum of squared errors between the observed and predicted values of the outcome variable
  - From the above OLS regression result summary, we can find out R2, adjusted R2, F statistics, P-value, the coefficient of different variables, and also the intersection point. The full information in a single sheet.
  - If we compare the linear regression model coefficient values, R2 and adjusted R2 with OLS summary then we can see that both the model's evaluation result is same.

Testing the Assumptions of the Linear Regression model:

1. *Multicollinearity check by VIF score:*

```
def check_VIF(X_t):
  vif = pd.DataFrame()

  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)
  return vif
```

- Variance inflation factor or VIF is a method to check the multi-collinearity for each independent variable. Higher the value of VIF means higher correlation of the variable with the other variables.
- If the VIF is greater than 5, then we dropped the variables one by one till the remaining variables VIF score comes to less than 5.

```
X_t = pd.DataFrame(x_train, columns=x_train.columns)
check_VIF(X_t)
```

	Features	VIF
5	CGPA	39.76
0	GRE Score	31.20
1	TOEFL Score	26.76
3	SOP	18.57
4	LOR	11.01
2	University Rating	10.95
6	Research	3.36

- We can see that VIF Scores of all variables (excluding Research) is too high, so we drop the CGPA column.

```
#drop CGPA and again check VIF
X_t.drop(columns = ['CGPA'], inplace = True)
check_VIF(X_t)
```

	Features	VIF
0	GRE Score	24.83
1	TOEFL Score	24.22
3	SOP	17.26
2	University Rating	10.90
4	LOR	10.15
5	Research	3.36

- After dropping the CGPA column, still we can see that the variables are too high, so we drop the GRE score column.

```
#drop GRE Score and again check VIF
X_t.drop(columns = ['GRE Score'], inplace = True)
check_VIF(X_t)
```

	Features	VIF
2	SOP	17.07
0	TOEFL Score	12.73
1	University Rating	10.79
3	LOR	10.09
4	Research	2.99

- After dropping the GRE score column, still the variables are too high, so now we drop the SOP column.

```
#drop SOP and again check VIF
X_t.drop(columns = ['SOP'], inplace = True)
check_VIF(X_t)
```

	Features	VIF
0	TOEFL Score	10.51
1	University Rating	9.33
2	LOR	8.17
3	Research	2.98

- After removing SOP column, still the variables are high VIF score, so now we drop TOEFL score column.

```
#drop TOEFL Score and again check VIF
X_t.drop(columns = ['TOEFL Score'], inplace = True)
check_VIF(X_t)
```

	Features	VIF
0	University Rating	7.19
1	LOR	6.49
2	Research	2.77

- After removing TOEFL column, still University rating and LOR columns have VIF score greater than 5. So, now drop the University rating column.

```
#drop University Rating and again check VIF
X_t.drop(columns = ['University Rating'], inplace = True)
check_VIF(X_t)
```

	Features	VIF
0	LOR	2.44
1	Research	2.44

- Finally, after removing University column, we get the VIF scores for LOR and Research column is less than 5.

2. *Mean of Residuals:*

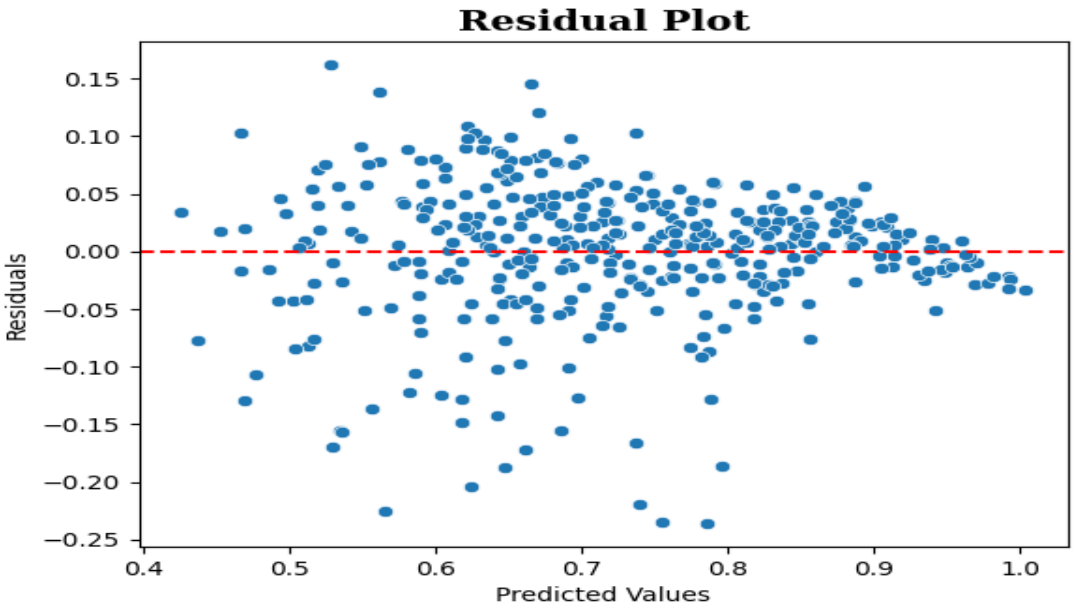
```
residuals = y_train.reshape(-1) - y_train_LR.reshape(-1)
mean_of_residuals = np.mean(residuals)

mean_of_residuals
```

```
-3.1780134079895106e-17
```

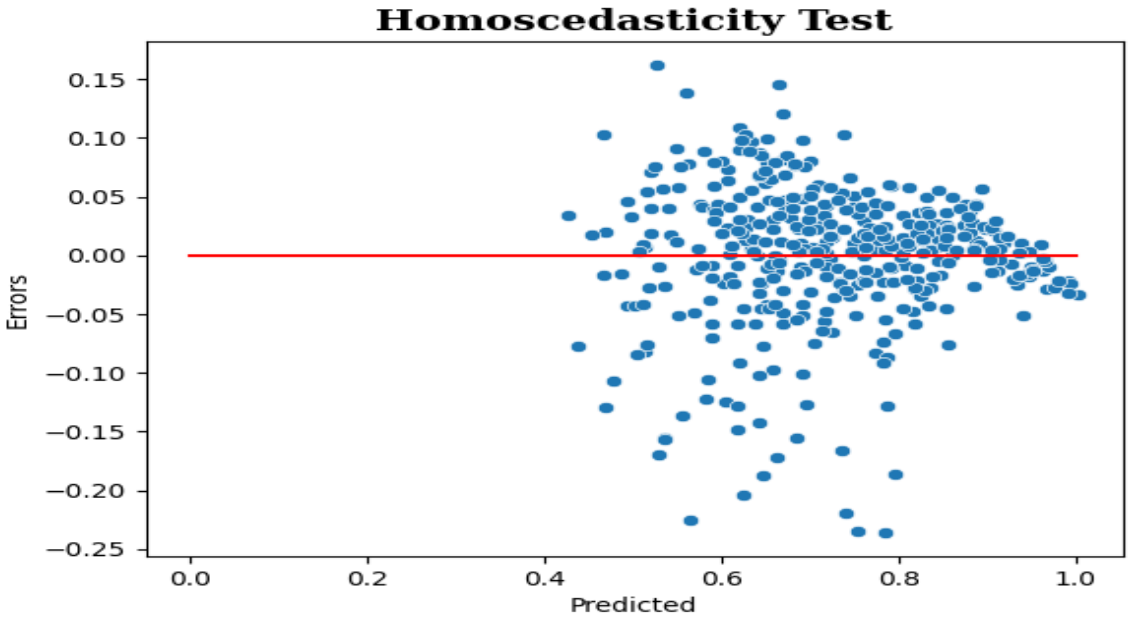
- The mean of residuals is useful to assess the overall bias in the regression model. If the mean of residuals is close to zero, it indicates that the model is unbiased on average. However, if the mean of residuals is significantly different from zero, it suggests that the model is systematically overestimating or underestimating the observed values.
- From the above results, we can say that the mean of residuals is close to zero, it means the model is unbiased.

3. Linearity of Variables:



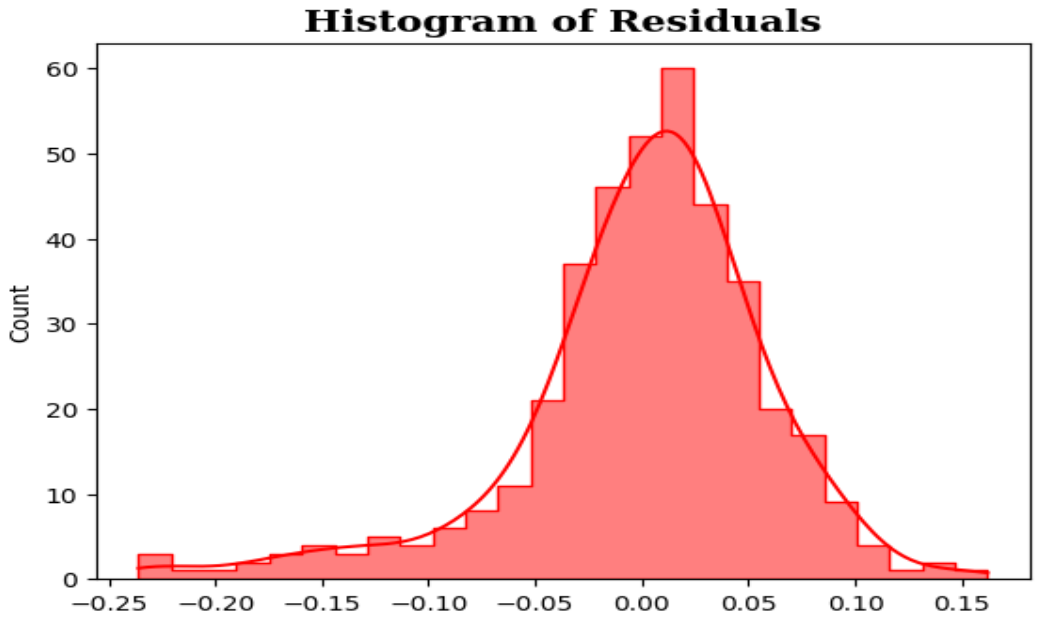
- Linearity of variables refers to the assumption that there is a linear relationship between the independent variables and the dependent variable in a regression model.
- We can check the linearity of data by residual plot. If residual vs. predicted values graph shows any pattern, it may indicate that the linear regression model is not appropriate.
- From the above residual plot, we can see that there is no pattern or trend, so it indicates that our linear regression model is a good fit.

3. ***Test for Homoscedasticity:***

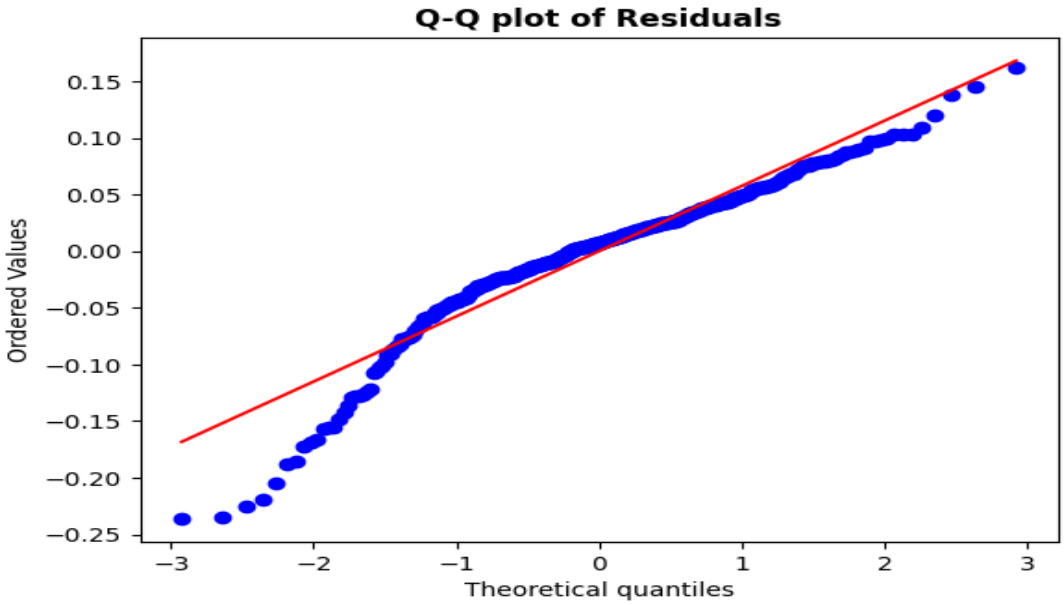


- Homoscedasticity refers that the residuals or errors have equal or almost equal variance across the regression line. By plotting the errors with predicted values we can check that there should not be any pattern in the errors.
- In graphical representation, if there is a definite pattern (like linear or quadratic or funnel shaped) obtained, it means it violates the homoscedasticity, it indicates that the variability of the errors is not consistent across the range of the predictors, which can lead to unreliable and biased regression estimates.
- From the above errors vs. predicted plot, we can see that there is no pattern, it indicates that homoscedasticity is present.

4. ***Normality of Residuals:***
   - Normality of residuals refers to the assumption that the residuals or errors in statistical model are normally distributed. Residuals are the differences between the observed values and the predicted values from the model.
   - When residuals are normally distributed, it implies that the errors are random, unbiased, and have consistent variability.
   - We can check the normality of residuals by various methods, like residual histogram, Q-Q plot and Shapiro-wilk test.



- **Histogram of Residuals:** the above histogram shows that, the distribution of residuals is left skewed but still it is close to normal distribution.



- **Q-Q plot:** From the above Q-Q plot, we can see that the residuals are slightly deviating from the straight line. It means, as per Q-Q plot the residuals are not normally distributed.

```python
#Normality Check using Shapiro-Wilk test

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(residuals)
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 7.735529881577885e-13
Reject Ho. The sample does not follow normal distribution
```

- **Shapiro-Wilk test:** After using Shapiro-wilk test, we can confirm that the residuals does not follow the normal distribution.

Colab notebook link:

https://colab.research.google.com/drive/1dqX-BZA38-abO8Fp0bcTWl1ox6VXBdNw?usp=sharing