Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from scipy.stats import probplot, shapiro
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

```
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

+ Code + Text

Loading Dataset

```
!wget "https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/839/original/Jamboree_Admission.csv"
```

```
--2024-02-14 11:26:10--  https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/839/original/Jamboree_Admission.csv
Resolving d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)... 99.84.178.132, 99.84.178.93, 99.84.178.172, ...
Connecting to d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)|99.84.178.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 16176 (16K) [text/plain]
Saving to: 'Jamboree_Admission.csv'

Jamboree_Admission. 100%[===================>]  15.80K  --.-KB/s    in 0s

2024-02-14 11:26:11 (144 MB/s) - 'Jamboree_Admission.csv' saved [16176/16176]
```

Analysis of Dataset

```
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

```
df.shape
```

```
(500, 9)
```

```
df.columns
```

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

```
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
```

Basic Data Cleaning and Exploration

Rename Columns

```
df.rename(columns = {'LOR ' : 'LOR', 'Chance of Admit ' : 'Chance of Admit'}, inplace = True)
```

Column-wise Unique Entries

```
#find out unique entries
for i in df.columns:
    print(f"Unique entries for column {i:18} = {df[i].nunique()}")
```

```
Unique entries for column Serial No.         = 500
Unique entries for column GRE Score          = 49
Unique entries for column TOEFL Score        = 29
Unique entries for column University Rating  = 5
Unique entries for column SOP                = 9
Unique entries for column LOR                = 9
Unique entries for column CGPA               = 184
Unique entries for column Research           = 2
Unique entries for column Chance of Admit    = 61
```

Conversion of Categorical Attributes to Category

```
categorical_columns = ['University Rating', 'SOP', 'LOR', 'Research']
for data in categorical_columns:
  df[data] = df[data].astype('category')
```

Updating float64 to float32 columns

```
floating_columns = ['CGPA', 'Chance of Admit']
for i in floating_columns:
    df[i] = df[i].astype('float32')
```

Removing the Unique Row Identifier

```
df = df.drop(columns='Serial No.')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype   
---  ------             --------------  -----   
 0   GRE Score          500 non-null    int64   
 1   TOEFL Score        500 non-null    int64   
 2   University Rating  500 non-null    category
 3   SOP                500 non-null    category
 4   LOR                500 non-null    category
 5   CGPA               500 non-null    float32 
 6   Research           500 non-null    category
```

```
 7   Chance of Admit    500 non-null    float32
dtypes: category(4), float32(2), int64(2)
memory usage: 14.9 KB
```

Missing Values in Dataset

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

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

Duplicate Rows

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

```
0
```

```
df.head()
```

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

Next steps: Generate code with df   View recommended plots

Statistical Summary

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
GRE Score	500.0	316.47200	11.295148	290.00	308.0000	317.00	325.00	340.00
TOEFL Score	500.0	107.19200	6.081868	92.00	103.0000	107.00	112.00	120.00
CGPA	500.0	8.57644	0.604813	6.80	8.1275	8.56	9.04	9.92
Chance of Admit	500.0	0.72174	0.141140	0.34	0.6300	0.72	0.82	0.97

```
df.describe(include='category').T
```

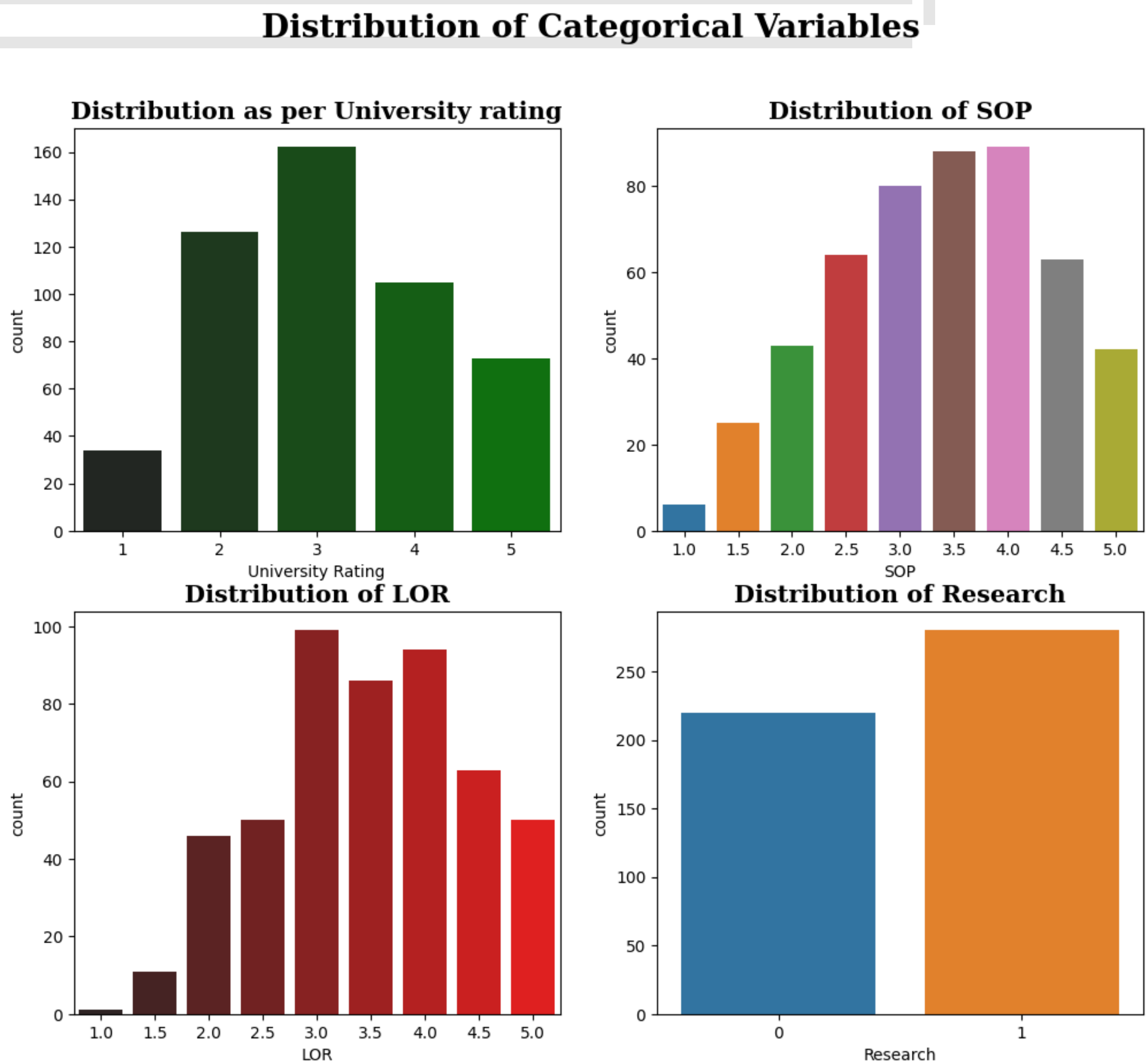
	count	unique	top	freq
University Rating	500.0	5.0	3.0	162.0
SOP	500.0	9.0	4.0	89.0
LOR	500.0	9.0	3.0	99.0
Research	500.0	2.0	1.0	280.0

Univariate Analysis

DIstribution Plot for Categorical Variables

```
plt.figure(figsize = (12, 10))
```

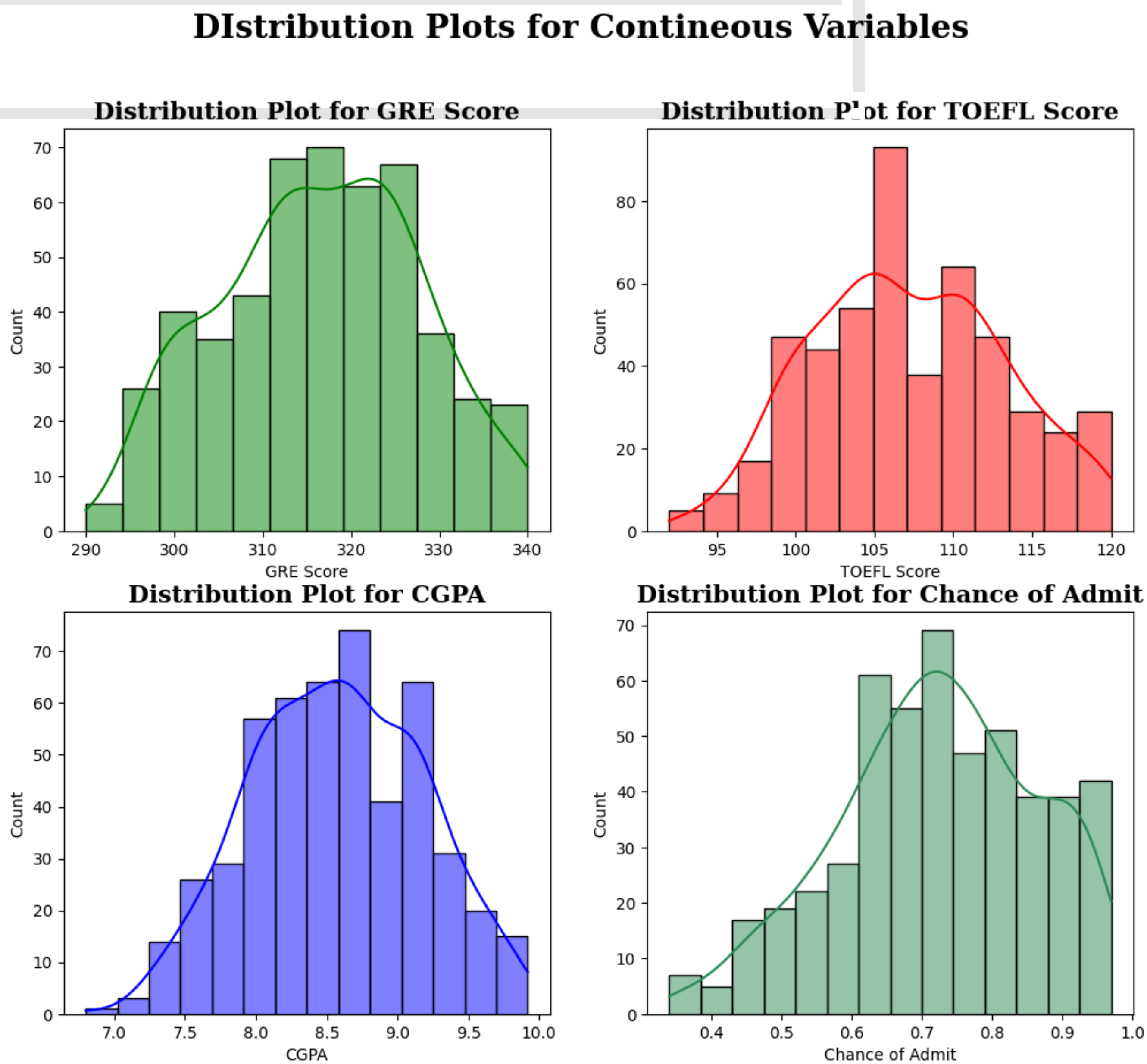
```
plt.subplot(2,2,1)
sns.countplot(data=df, x='University Rating', hue = 'University Rating', legend= False, palette = 'dark:green'
plt.title('Distribution as per University rating', {'font':'serif', 'size':15,'weight':'bold'})

plt.subplot(2,2,2)
sns.countplot(data=df, x='SOP', hue = 'SOP', legend= False)
plt.title('Distribution of SOP', {'font':'serif', 'size':15,'weight':'bold'})

plt.subplot(2,2,3)
sns.countplot(data=df, x= 'LOR', hue = 'LOR', legend= False, palette = 'dark:red')
plt.title('Distribution of LOR', {'font':'serif', 'size':15,'weight':'bold'})

plt.subplot(2,2,4)
sns.countplot(data=df, x='Research', hue = 'Research', legend= False)
plt.title('Distribution of Research', {'font':'serif', 'size':15,'weight':'bold'})

plt.suptitle("Distribution of Categorical Variables", font='serif', size=20,weight='bold')
```



DIstribution Plot for Continuous Variables

```
plt.figure(figsize=(12,10))

plt.subplot(2,2,1)
sns.histplot(df['GRE Score'], kde = True, color = 'green', label = 'GRE Score')
plt.title("Distribution Plot for GRE Score", {'font':'serif', 'size':15, 'weight':'bold'})
```

```
plt.subplot(2,2,2)
sns.histplot(df['TOEFL Score'], kde = True, color = 'red', label = 'TOEFL Score')
plt.title("Distribution Plot for TOEFL Score", {'font':'serif', 'size':15, 'weight':'bold'})

plt.subplot(2,2,3)
sns.histplot(df['CGPA'], kde = True, color = 'blue', label = 'CGPA')
plt.title("Distribution Plot for CGPA", {'font':'serif', 'size':15, 'weight':'bold'})

plt.subplot(2,2,4)
sns.histplot(df['Chance of Admit'], kde = True, color = 'seagreen', label = 'Chance of Admit')
plt.title("Distribution Plot for Chance of Admit", {'font':'serif', 'size':15, 'weight':'bold'})

plt.suptitle("DIstribution Plots for Contineous Variables", font = 'serif', size = 20, weight = 'bold'
```

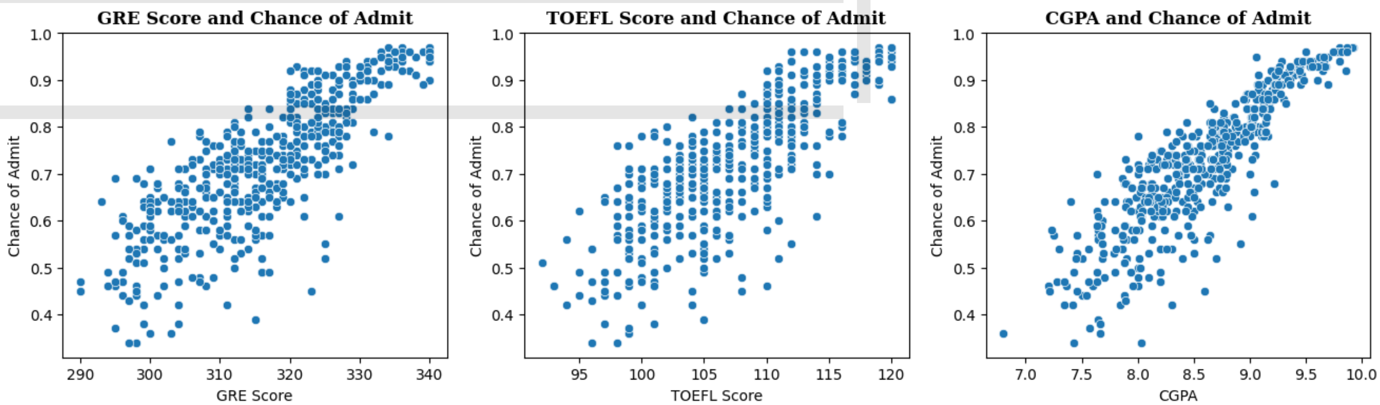


Bi-Variate Analysis

Relationship Between GRE Score, TOEFL Score and CGPA with Chance of Admit

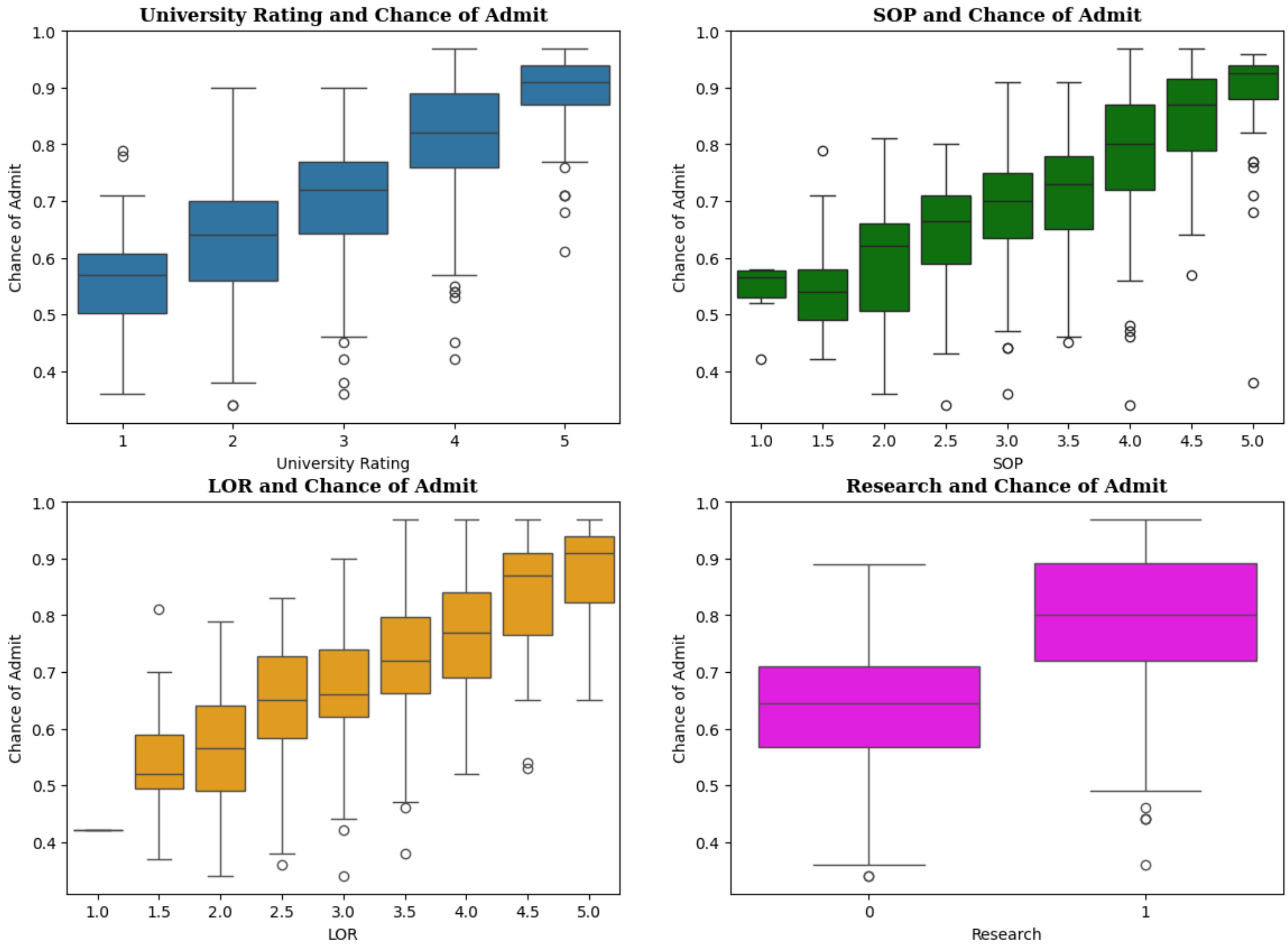
```
plt.figure(figsize=(16,4))

plt.subplot(1,3,1)
sns.scatterplot(x=df['GRE Score'], y=df['Chance of Admit'])
```

```
plt.title("GRE Score and Chance of Admit", {'font':'serif', 'size':12, 'weight':'bold'})

plt.subplot(1,3,2)
sns.scatterplot(x=df['TOEFL Score'], y=df['Chance of Admit'])
plt.title("TOEFL Score and Chance of Admit", {'font':'serif', 'size':12, 'weight':'bold'}

plt.subplot(1,3,3)
sns.scatterplot(x=df['CGPA'], y=df['Chance of Admit'])
plt.title("CGPA and Chance of Admit", {'font':'serif', 'size':12, 'weight':'bold'})


plt.show()
```



Relationship between Categorical columns and Chance of Admit

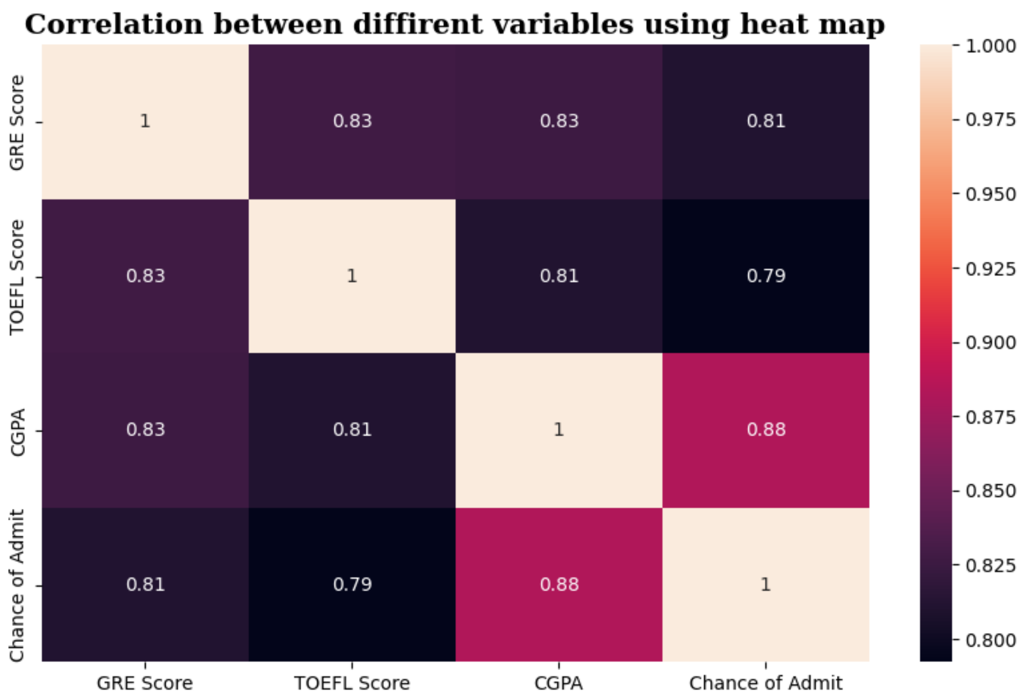
```
plt.figure(figsize=(14,10))

plt.subplot(2,2,1)
sns.boxplot(x = df['University Rating'],y= df['Chance of Admit'])
plt.title("University Rating and Chance of Admit", {'font':'serif', 'size':12, 'weight':'bold'})

plt.subplot(2,2,2)
sns.boxplot(x= df['SOP'],y= df['Chance of Admit'], color='green')
plt.title("SOP and Chance of Admit", {'font':'serif', 'size':12, 'weight':'bold'})

plt.subplot(2,2,3)
sns.boxplot(x= df['LOR'],y= df['Chance of Admit'], color='orange')
plt.title("LOR and Chance of Admit", {'font':'serif', 'size':12, 'weight':'bold'})

plt.subplot(2,2,4)
sns.boxplot(x= df['Research'],y= df['Chance of Admit'], color='magenta')
plt.title("Research and Chance of Admit", {'font':'serif', 'size':12, 'weight':'bold'})

plt.show()
```



Multi-variate Analysis

```
plt.figure(figsize = (10, 6))

sns.heatmap(data = df.corr(), annot = True)

plt.title('Correlation between diffirent variables using heat map', font='serif', size=15, weight='bold')
#plt.xticks(rotation=45)

plt.show()
```

```
<ipython-input-25-26bf52663cf8>:3: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future versior
  sns.heatmap(data = df.corr(), annot = True)
```



Outlier Detection

Outlier detection 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('---------------------------------')
```

```
#detect outliers by IQR for contineous variables
numerical_columns = ['GRE Score', 'TOEFL Score', 'CGPA', 'Chance of Admit']
for i in numerical_columns:
  outlier_by_IQR(i)
```

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

Model Building

Data preparetion 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,)
```

```
#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)
```

```
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,)
```

Transfroming 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])
```

Normalizing data by MinMaxScaler

```
scaler = MinMaxScaler()

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

```
x_train.head()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	0.62	0.678571	0.50	0.625	0.714286	0.650641	1.0
1	0.52	0.678571	0.75	0.750	1.000000	0.557692	0.0
2	0.26	0.357143	0.50	0.625	0.428571	0.544872	0.0
3	0.48	0.535714	0.25	0.375	0.714286	0.471154	0.0
4	0.36	0.500000	0.50	0.625	0.285714	0.451923	1.0

Next steps: Generate code with x_train View recommended plots

```
x_test.head()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	0.88	0.851852	0.75	0.750	0.625	0.894309	1.0
1	0.48	0.555556	0.75	0.875	0.750	0.691057	1.0
2	0.50	0.444444	0.25	0.250	0.375	0.126016	0.0
3	0.44	0.592593	0.50	0.500	0.500	0.548780	0.0
4	0.72	0.703704	0.50	0.625	0.500	0.695122	1.0

Next steps: Generate code with x_test View recommended plots

Linear Regression

`Model Evaluation`

```
model_LR = LinearRegression()
```

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

```
▾ LinearRegression
LinearRegression()
```

Model Coefficients with Column Names

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

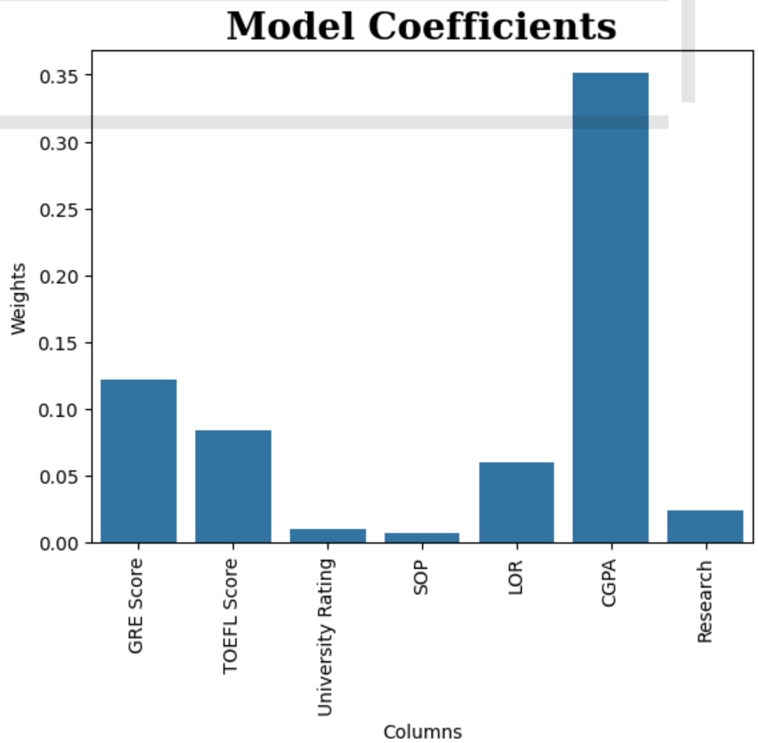
```
Intercept: 0.35558406163802325
```

```
#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
```

```
#Visualization of model coefficients
sns.barplot(x = x_train.columns, y = model_LR.coef_)
plt.title("Model Coefficients", font = 'serif', size= 20, weight= 'bold'
plt.xlabel("Columns")
plt.ylabel("Weights")
```

```
plt.xticks(rotation=90)
```



Model Performance Evaluation

```
#reshaping of target train value
y_train = y_train.values.reshape(-1, 1)

#reshaping of target test value
y_test = y_test.values.reshape(-1, 1)

y_train.shape, y_test.shape
```

```
((400, 1), (100, 1))
```

```
#prediction value for train data
y_train_LR = model_LR.predict(x_train)

#prediction value for test data
y_test_LR = model_LR.predict(x_test)
```

```
def model_evaluation(y_actual, y_prediction, model, n, d):
  MAE = mean_absolute_error(y_actual, y_prediction)
  MSE = mean_squared_error(y_actual, y_prediction)
  RMSE = np.sqrt(MSE)
  R2 = r2_score(y_actual, y_prediction)
  Adjusted_R2 = 1 - ((1-R2)*(n-1)/(n - d - 1))

  return print(f"MAE: {round(MAE,2)}\nMSE: {round(MSE,3)}\nRMSE: {round(RMSE,2)}\nR2 score: {round(R2,2)}\nAdjusted R2: {round(Adjusted_R2,2
```

```
#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
```

Ridge and Lasso regression

```
model_R = Ridge()
model_L = Lasso()
```

```
#fitting the model to training data
model_R.fit(x_train, y_train)
model_L.fit(x_train, y_train)
```

```
▾ Lasso
Lasso()
```

```
#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


Ridge Regression Test Model

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


Lasso Regression Training Model

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


Lasso Regression Test Model

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

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:                Wed, 14 Feb 2024   Prob (F-statistic):          3.41e-142
Time:                        11:26:16   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.
```

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
```

```
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

```
#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

```
#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

```
#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

```
#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

```
#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

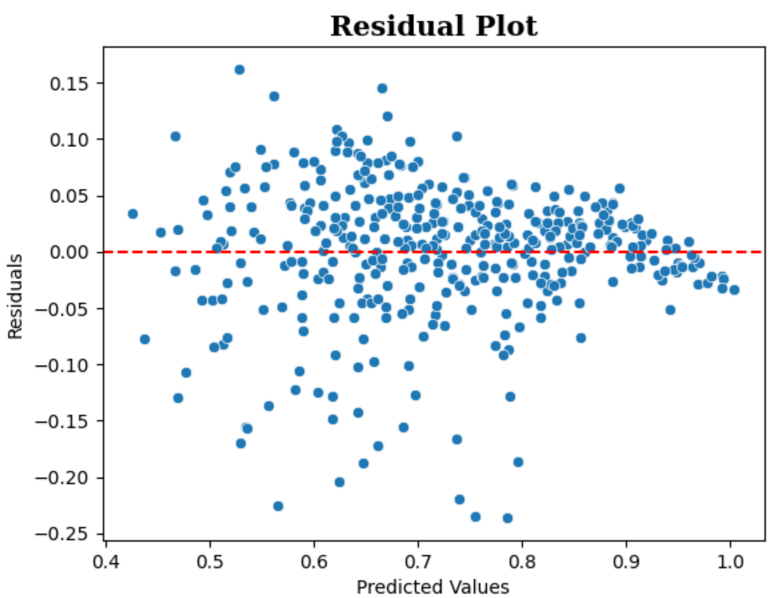
2. Mean of Residuals

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

mean_of_residuals
```

```
-3.1780134079895106e-17
```

3. Linearity of Variables

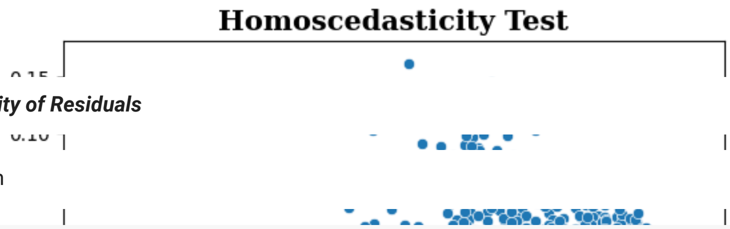
```
sns.scatterplot(x = y_train_LR.reshape(-1), y= residuals)
plt.title('Residual Plot', font = 'serif', size= 15, weight='bold')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.show();
```



4. Test for Homoscedasticity

```
errors = y_train.reshape(-1) - y_train_LR.reshape(-1)
```

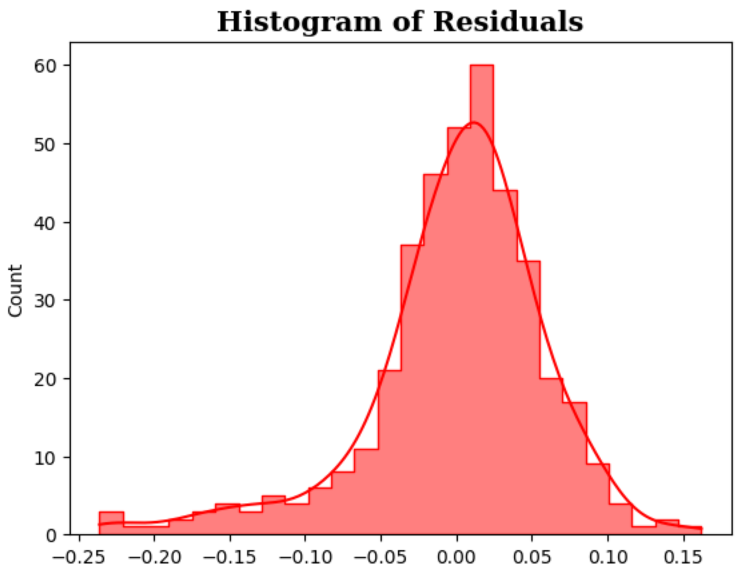
```
sns.scatterplot(x = y_train_LR.reshape(-1,), y= errors.reshape(-1, ))
sns.lineplot([0,0], color='red')
plt.title("Homoscedasticity Test", font='serif', size=15, weight='bold')
plt.xlabel("Predicted")
plt.ylabel("Errors")
plt.show()
```



5. Normality of Residuals

Histogram

```
sns.histplot(residuals, element = 'step', kde = True, color = 'red')
plt.title("Histogram of Residuals", font = 'serif', size = 15, weight = 'bold')

plt.show()
```



Q-Q plot

```
#Normality check
probplot(residuals, plot = plt, dist = 'norm')
plt.title('Q-Q plot of Residuals', weight = 'bold')
plt.show()
```