```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df_jamboree = pd.read_csv("https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/839/original/Jamboree_Admission.
```

```
df_jamboree.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

```
# Size of dataset

df_jamboree.shape
```

```
(500, 9)
```

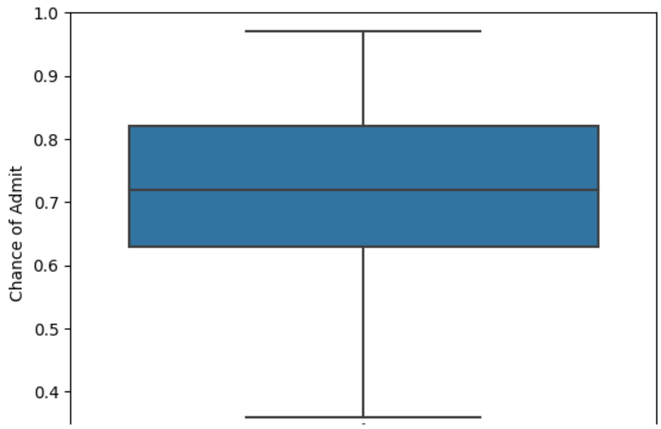
```
#Count of null values

df_jamboree.isnull().sum()
```

```
Serial No.           0
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

```
# Feature selection

# Since Serial No. do not influence the target variable Chance of Admit, Serial No. is removed

df_jamboree.drop("Serial No.", axis = 1, inplace = True)
```

```
# outlier analysis

sns.boxplot(data = df_jamboree, y = "Chance of Admit ")
```

```
<Axes: ylabel='Chance of Admit '>
```



```
# Removal of outliers

for col in df_jamboree.columns:
  upper_limit = (np.percentile(df_jamboree[col],75)) + 1.5 * ((np.percentile(df_jamboree[col],75)) - (np.percentile(df_jamboree
  lower_limit = (np.percentile(df_jamboree[col],25)) - 1.5 * ((np.percentile(df_jamboree[col],75)) - (np.percentile(df_jamboree

  df_jamboree = df_jamboree[(df_jamboree[col] >= lower_limit) & (df_jamboree[col]<= upper_limit)]
```

```
df_jamboree.shape
```

```
(497, 8)
```

```
# Taking predictors as x and label as y

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

```
# Data preprocessing
# Scaling all the predictors so that values of all the preditors lie in the same range.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit_transform(x)

# scaler results in Numpy array of scaled value, hence to get back dataframe, data frame of array of scaled values is created

x = pd.DataFrame(scaler.fit_transform(x), columns = x.columns)
```

```
# data set is split into train data and test data

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.30, random_state = 1)
```

```
# Building Linear Regression model using Sklearn libraries

from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(x_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
# coefficients/estimators of model

model.coef_
```

```
array([0.09146609, 0.10026502, 0.02201733, 0.01048318, 0.0568357 ,
       0.33575974, 0.02631611])
```

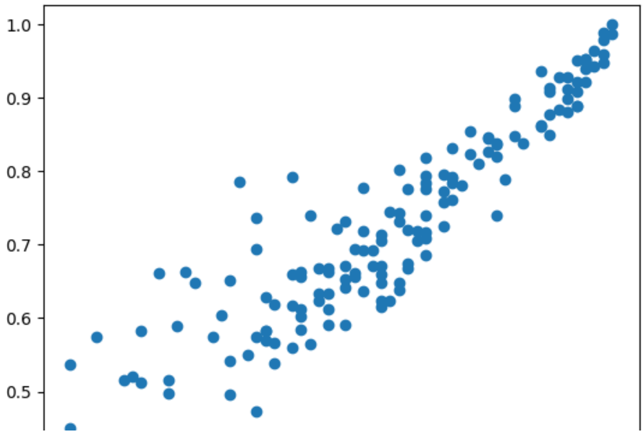
```
# Intercept of model

model.intercept_
```

```
0.36633275701626655
```

```
# Model has been built, now will validate our model using scatter plot, R_square and  Adj_R_square

y_test_pred = model.predict(x_test)
```

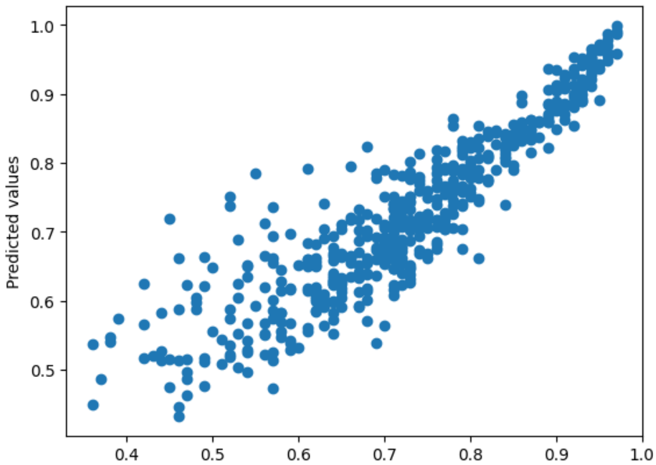
```
# Scatter plot ----> Actual label value Vs predicted label value

plt.scatter(y_test, y_test_pred)
```

```
<matplotlib.collections.PathCollection at 0x7b296f459ab0>
```




```
# Scatter -------> Actual values of label and predicted values of label

y_predicted = model.predict(x)

plt.scatter(y, y_predicted)
plt.xlabel("Actual values")
plt.ylabel("Predicted values")
```

```
Text(0, 0.5, 'Predicted values')
```




```
# R_squre and Adjusted_R_sqaure

num = np.sum((y_test - y_test_pred)**2)
den = np.sum((y_test - y_test_pred.mean())**2)

R_square = 1 - (num/den)

Adj_R_square = 1 - ((1 - R_square)*(len(y_test)-1)/(len(y_test)- x.shape[1]- 1))

print(f"R_square: {R_square}")
print(f"Adjusted R_square: {Adj_R_square}")
```

```
R_square: 0.8273146556450441
Adjusted R_square: 0.8188019978247294
```

```
# checking the overfitting----> In order to check overfitting, R_square and Adjusted R_square calulated on train data and comp

y_train_pred = model.predict(x_train)

num = np.sum((y_train - y_train_pred)**2)
den = np.sum((y_train - y_train_pred.mean())**2)

R_square = 1 - (num/den)

Adj_R_square = 1 - ((1 - R_square)*(len(y_train)-1)/(len(y_train)- x.shape[1]- 1))

print(f"R_square: {R_square}")
print(f"Adjusted R_square: {Adj_R_square}")
```

```
R_square: 0.8179463692092138
Adjusted R_square: 0.8141871496943598
```

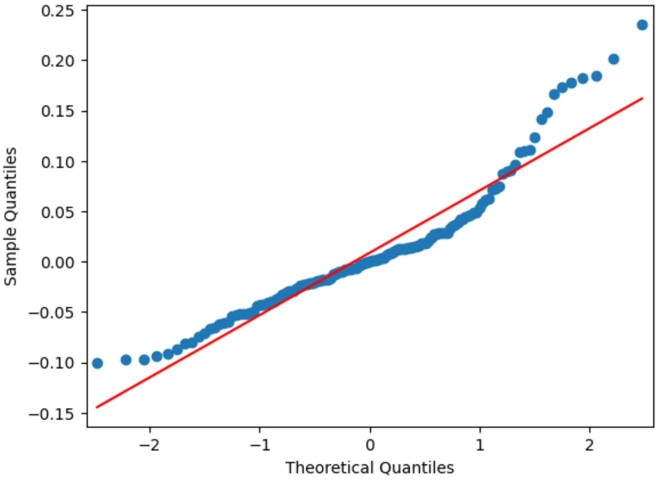
```
# Assumption of Linear Regression

# Assumption 01:Each independent variable has linear relationship with dependent variable

df_jamboree.corr()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	(
GRE Score	1.000000	0.824360	0.631514	0.614286	0.518457	0.82
TOEFL Score	0.824360	1.000000	0.645349	0.643806	0.533263	0.80
University Rating	0.631514	0.645349	1.000000	0.727569	0.603831	0.70
SOP	0.614286	0.643806	0.727569	1.000000	0.659858	0.71

```
# Assumption 02: No multicollinearity ----> Presence of Multicollinearity will not have any impact on prediction.... But weigh
# To find the existence of multicollinearity, VIF associated to each independent variable is found
# VIF = 1/(1- (R_square_i)**2)

from statsmodels.stats.outliers_influence import variance_inflation_factor

VIF = pd.DataFrame()

VIF["Features"] = x.columns

VIF["VIF"] = [variance_inflation_factor(x_train.values,i) for i in range(x_train.shape[1])]

VIF
```

	Features	VIF
0	GRE Score	26.851618
1	TOEFL Score	27.375418
2	University Rating	10.890710
3	SOP	18.383690
4	LOR	10.687881
5	CGPA	37.993106

```
# Assumption 03: Errors are normally ditributed

# Q-Q plot

from statsmodels.graphics.gofplots import qqplot

qqplot((y_test_pred - y_test), line = "s")
```




```
# Assumption 04: There should not be any oulier ----> Already outliers has been removed in data cleaning stage
# Assumption 05: No hetiroscadacity

plt.scatter(y, (y_predicted - y))
plt.xlabel("Predicted values")
plt.ylabel("Residuals")
```

```
Text(0, 0.5, 'Residuals')
```