```
!gdown 1UCnSk_NN02jlzj0bbSZ_j-gdGUDDJxy4
```

```
Downloading...
From: https://drive.google.com/uc?id=1UCnSk_NN02jlzj0bbSZ_j-gdGUDDJxy4
To: /content/Jamboree.csv
100% 16.2k/16.2k [00:00<00:00, 18.1MB/s]
```

```
import numpy as np
import pandas as pd
df = pd.read_csv('Jamboree.csv')
```

```
df
```

	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
...	...	...	...	...	...	...	...	...	...
495	496	332	108	5	4.5	4.0	9.02	1	0.87
496	497	337	117	5	5.0	5.0	9.87	1	0.96
497	498	330	120	5	4.5	5.0	9.56	1	0.93
498	499	312	103	4	4.0	5.0	8.43	0	0.73
499	500	327	113	4	4.5	4.5	9.04	0	0.84

500 rows × 9 columns

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

```
# Serial no. column is useless so we drop it
df = df.drop(['Serial No.'], axis = 1)
df
```

	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
...	...	...	...	...	...	...	...	...
495	332	108	5	4.5	4.0	9.02	1	0.87
496	337	117	5	5.0	5.0	9.87	1	0.96
497	330	120	5	4.5	5.0	9.56	1	0.93
498	312	103	4	4.0	5.0	8.43	0	0.73
499	327	113	4	4.5	4.5	9.04	0	0.84

500 rows × 8 columns

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

```
# check for null values
df.isnull().sum()
```

	0
GRE Score	0
TOEFL Score	0
University Rating	0
SOP	0
LOR	0
CGPA	0
Research	0
Chance of Admit	0

dtype: int64

```
df.describe()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Researcl
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(
max	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.00000(

```
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    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
```

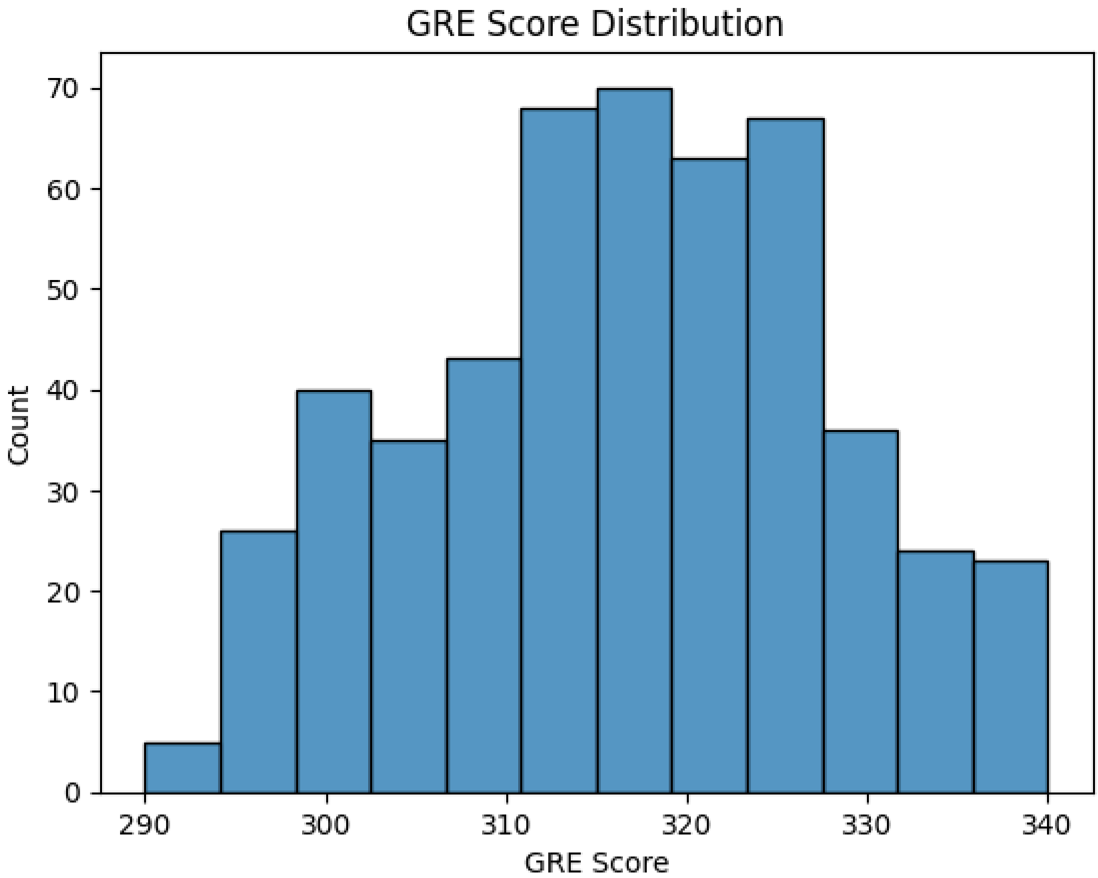
Our data is free from any object or string types, which means we don't need any categorical encoding for further machine learning analysis.

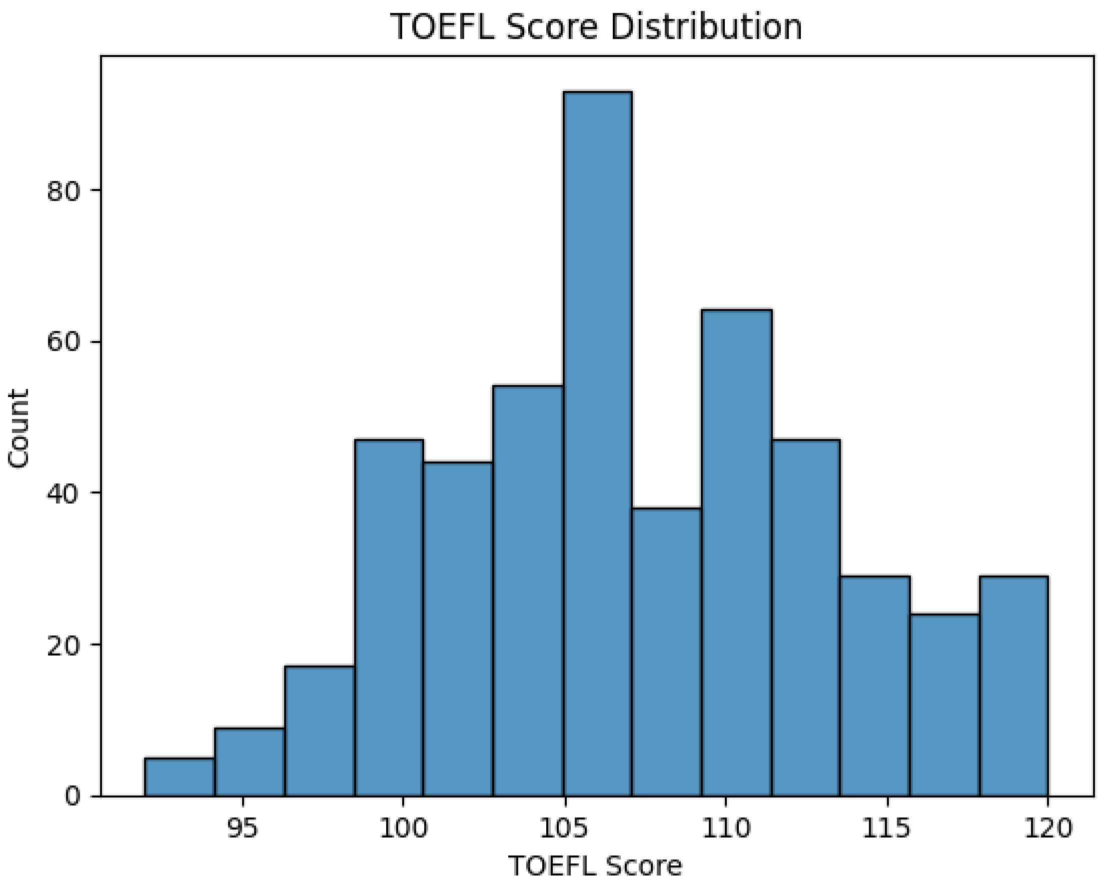
```
# it seems that the name of two the columns have redundent space in it, so we remove it.
print(df.columns)
df.rename(columns={'LOR ' : 'LOR'}, inplace = True)
df.rename(columns={'Chance of Admit ' : 'Chance of Admit'}, inplace = True)
```

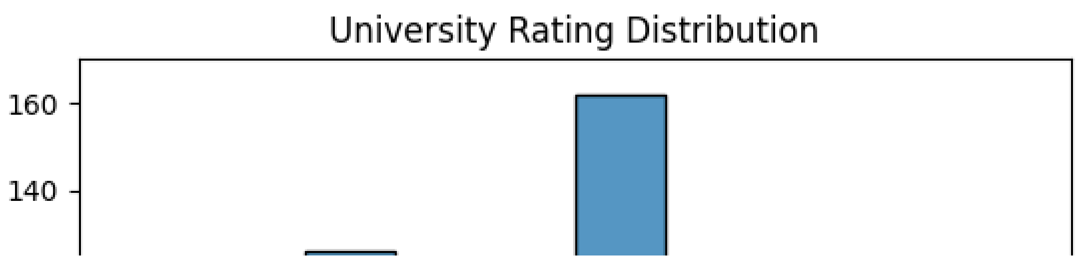
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

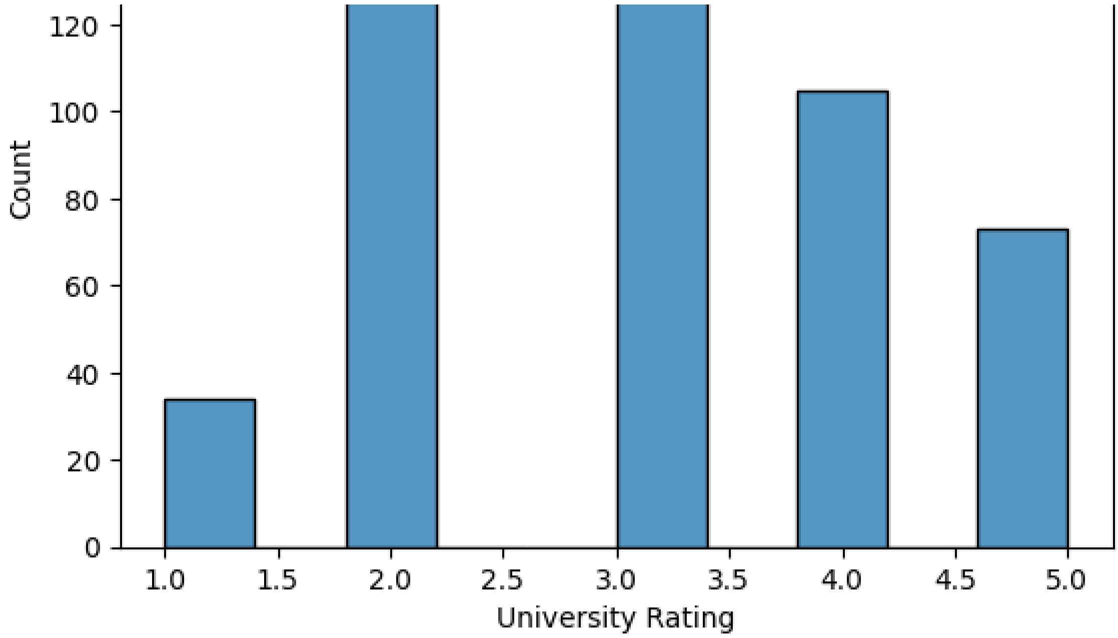
```
import seaborn as sns
import matplotlib.pyplot as plt

fig = sns.histplot(df['GRE Score'])
plt.title("GRE Score Distribution")
```

```
    plt.show()

    fig = sns.histplot(df['TOEFL Score'], kde=False)
    plt.title("TOEFL Score Distribution")
    plt.show()

    fig = sns.histplot(df['University Rating'], kde=False,)
    plt.title("University Rating Distribution")
    plt.show()

    fig = sns.histplot(df['SOP'], kde=False)
    plt.title("SOP Score Distribution")
    plt.show()

    fig = sns.histplot(df['LOR'], kde=False)
    plt.title("LOR Score Distribution")
    plt.show()

    fig = sns.histplot(df['CGPA'], kde=False)
    plt.title("CGPA Score Distribution")
    plt.show()
```

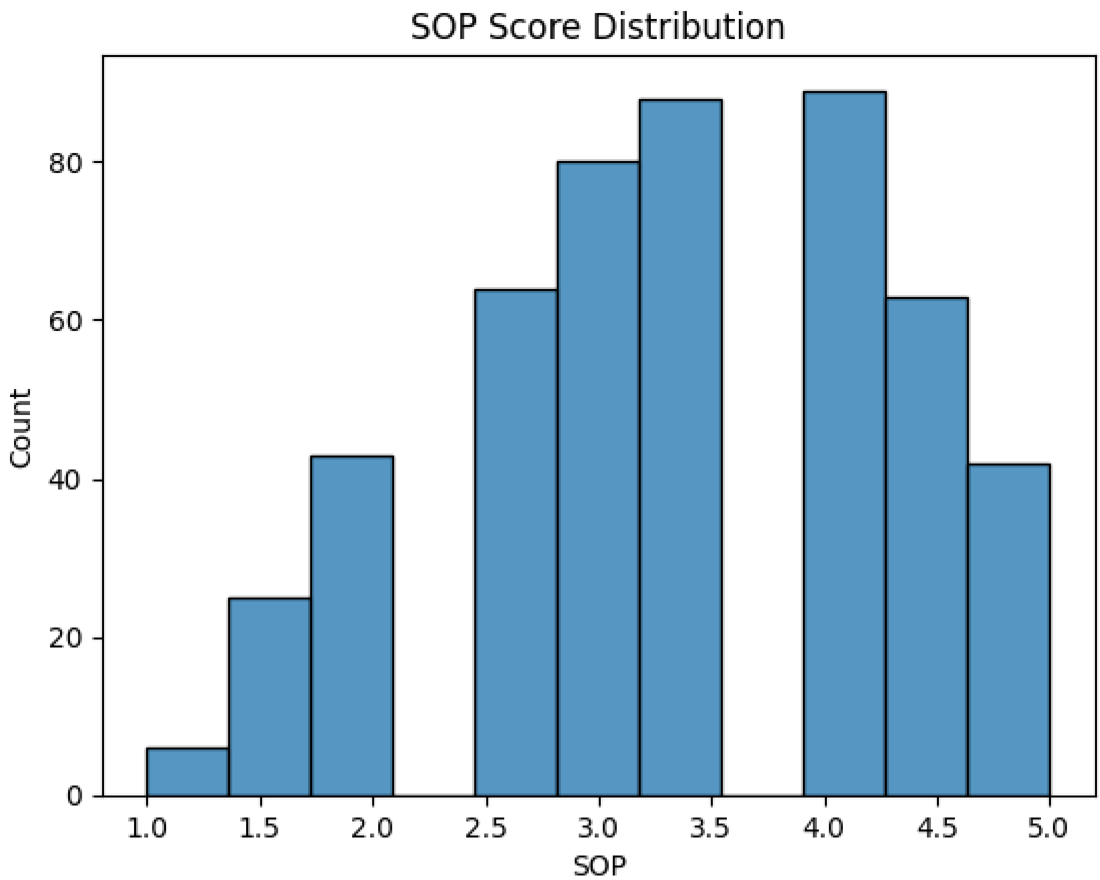

GRE Score Distribution
Count
70
60
50
40
30
20
10
0
290
300
310
320
330
340
GRE Score


TOEFL Score Distribution
Count
80
60
40
20
0
95
100
105
110
115
120
TOEFL Score

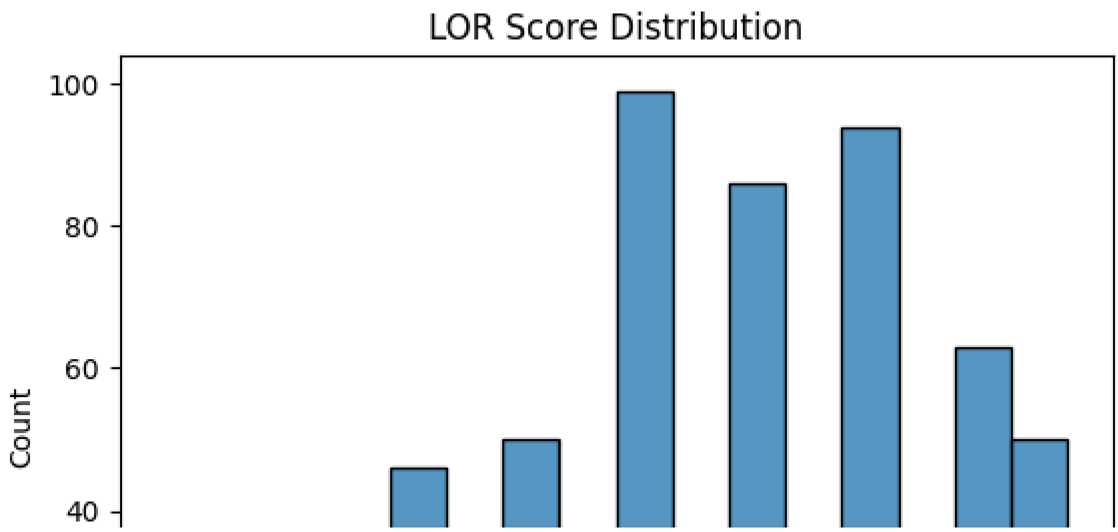

University Rating Distribution
160
140



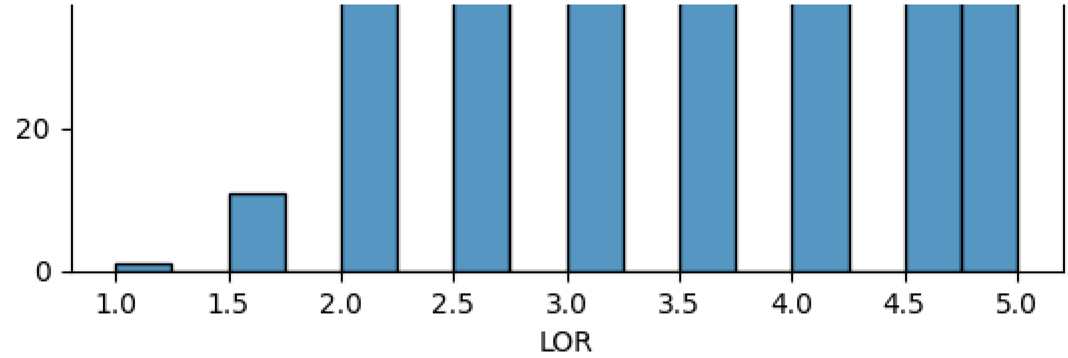
Count
University Rating
120
100
80
60
40
20
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

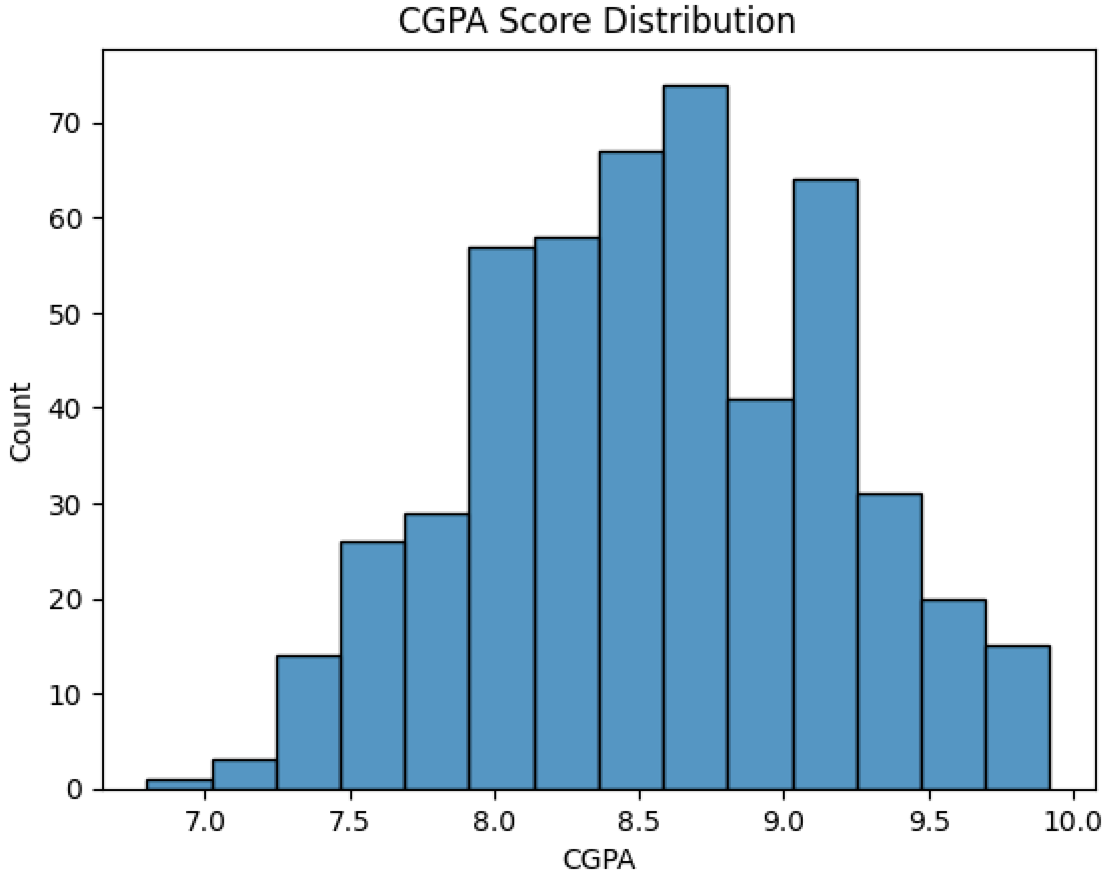


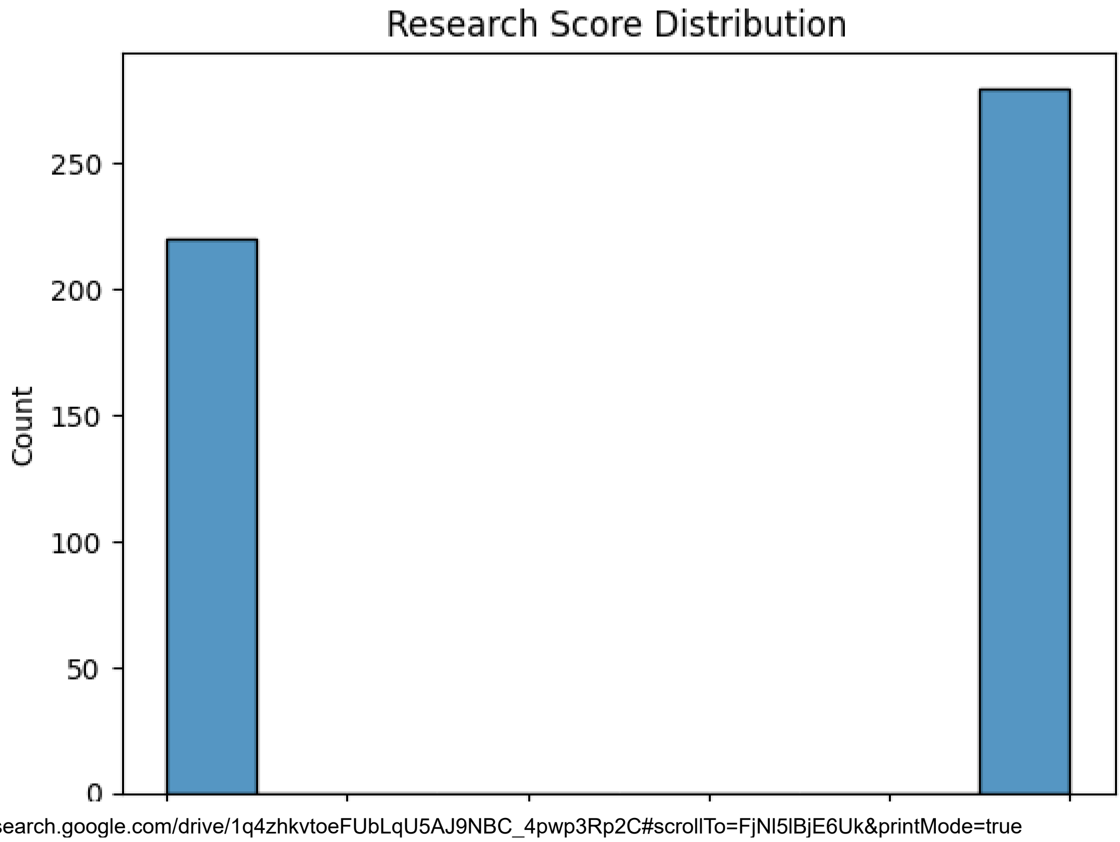
SOP Score Distribution
Count
SOP
80
60
40
20
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0



LOR Score Distribution
Count
100
80
60
40


20
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
LOR


CGPA Score Distribution
Count
0
10
20
30
40
50
60
70
7.0
7.5
8.0
8.5
9.0
9.5
10.0
CGPA

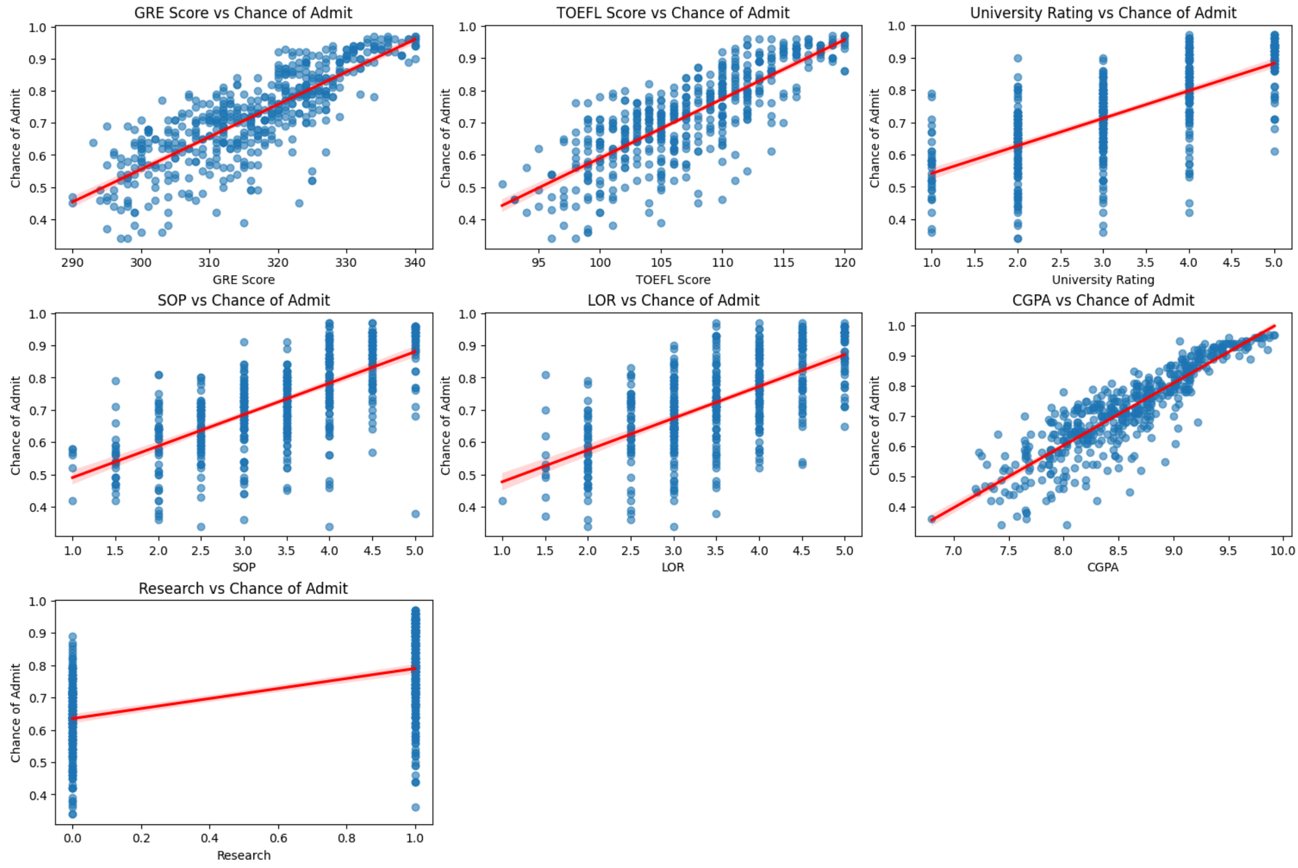

Research Score Distribution
Count
0
50
100
150
200
250


0.0
0.2
0.4
0.6
0.8
1.0
Research

All the histogram shows a bell-shaped which indicates it a normal distribution.

Understanding the relation between different factors reponsible for graduate admissions

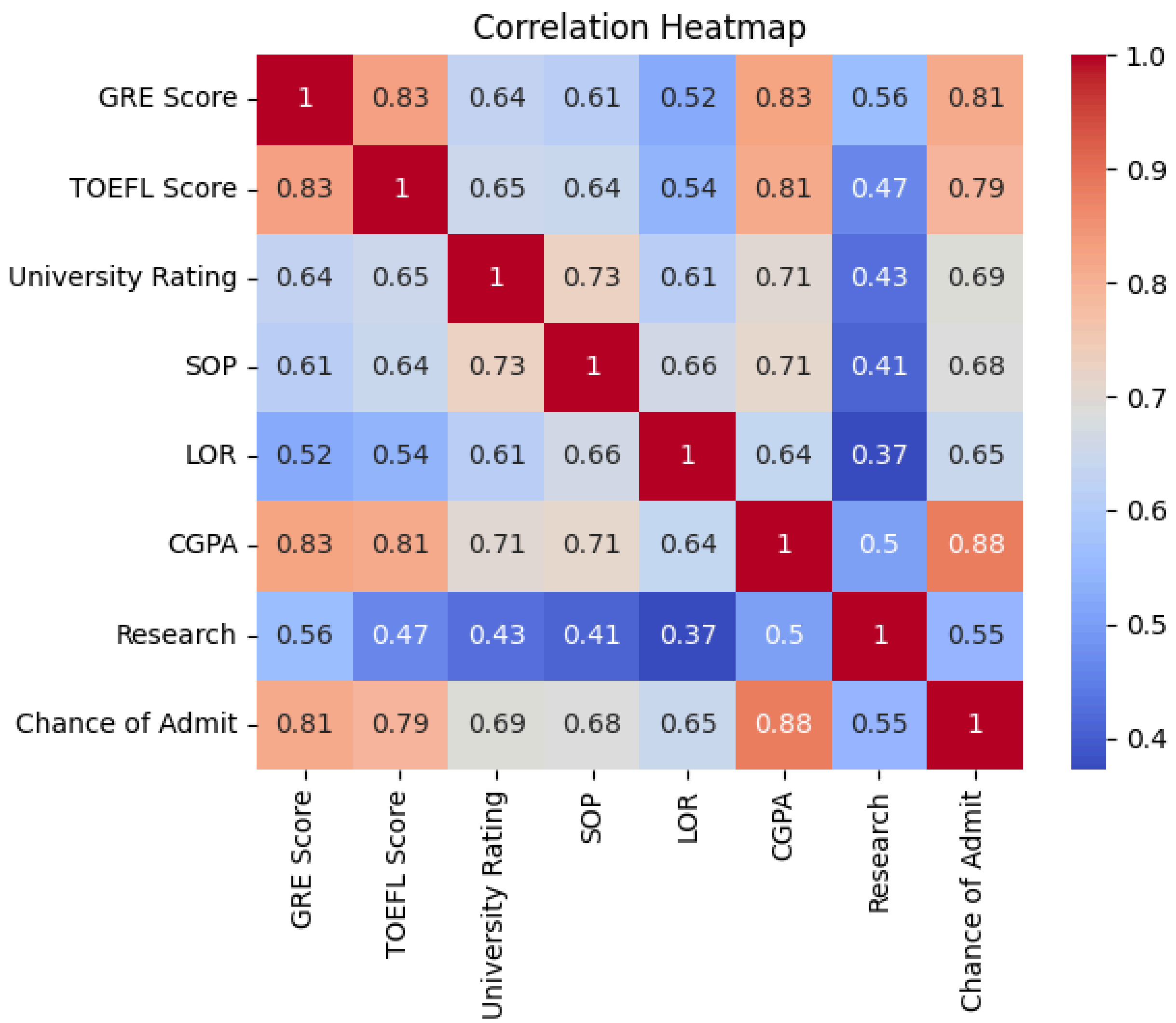
```
# fig = sns.regplot(x="GRE Score", y="TOEFL Score", data=df)
# plt.title("GRE Score vs TOEFL Score")
# plt.show()

columns = ['GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
           'LOR', 'CGPA', 'Research']
n_cols = 3
n_rows = (len(columns) + n_cols - 1) // n_cols

# creating subplots for grid view
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 10), constrained_layout=True)

# flatten the axes array for easier indexing
axes = axes.flatten()

#  generate regression plots in each subplot
for i, col in enumerate(columns):
    sns.regplot(x=col, y='Chance of Admit', data=df, scatter_kws={"alpha": 0.6}, line_k

    axes[i].set_title(f'{col} vs Chance of Admit')
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Chance of Admit')

# turn off any unused subplots
for j in range(len(columns), len(axes)):
    axes[j].axis('off')

plt.show()
```



- Red line(linear regression line) helps to visualize the trend or correlation between the variables
- Strongest Factors : GRE, TOEFL, and CGPA have the most significant positive correlations with the Chance of Admit.
- Moderate Factors : University Rating, SOP, and LOR moderately influence admissions.
- Weak Factors : Research has relatively minor impact when compared to any other academic score.

From the above analysis it is recommended to study hard to score more in GRE, TOEFL and CGPA which will ensure a high chance of admit

```
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```



1. Here CGPA, GRE score and TOEFL score are the variables with the highest positive correlation(most infuential factors) with the chance of admit.
2. Test score matters and Research experience adds value and provides an edge, especially when combined with strong academics

Split the dataset into training and testing sets and prepare the inputs and outputs.

```
# spliting data into training and testing subsets.
from sklearn.model_selection import train_test_split

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

```
# X_train, y_train -> 80% of the datapoints
# X_test, y_test -> 20% of the datapoints
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True)
```

```
X_train
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
4	314	103	2	2.0	3.0	8.21	0
246	316	105	3	3.0	3.5	8.73	0
120	335	117	5	5.0	5.0	9.56	1
469	326	114	4	4.0	3.5	9.16	1
429	340	115	5	5.0	4.5	9.06	1
...	...	...	...	...	...	...	...
10	325	106	3	3.5	4.0	8.40	1
186	317	107	3	3.5	3.0	8.68	1
285	331	116	5	4.0	4.0	9.26	1
119	327	104	5	3.0	3.5	8.84	1
52	334	116	4	4.0	3.0	8.00	1

400 rows × 7 columns

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

```
y_train
```

	Chance of Admit
4	0.65
246	0.72
120	0.94
469	0.86
429	0.95
...	...
10	0.52
186	0.84
285	0.93
119	0.71
52	0.78

400 rows × 1 columns

dtype: float64

```
#Standardization
from sklearn.preprocessing import StandardScaler
```

```
std = StandardScaler()
X_train_columns = X_train.columns
X_train_std = std.fit_transform(X_train)
# X_train = scaler.fit_transform(X_train)
# X_test = scaler.transform(X_test)
```

- By standardizing features, we ensure that they contribute equally to the model's training process, preveting features with larger ranges from dominating the learning algorithm.

What's the difference between fit_transform and simply tranform ?

- We should use fit_transform on our training data to learn the parameters and transform/standardize the data
- For the test set or any new data, we use only transform. This ensures that the scaling of the test data is based solely on the parameters learned from the training data.

When we scale the test set using parameters from the training set, we prevent the test set from "learning" anything about the training data. This is critical to ensure the test set remains unbiased and truly represents unseen data. If the test set were scaled independently of the training set (like using fit_transform), it would no longer act as a fair measure of the model's performance, as it could indirectly "know" how the model was trained.

```
X_train_std
```

```
array([[-0.26678057, -0.73261374, -1.05354393, ..., -0.55183018,
        -0.65134848, -1.13389342],
       [-0.08728145, -0.40500558, -0.14921438, ..., -0.00546367,
         0.21257502, -1.13389342],
       [ 1.61796021,  1.56064335,  1.65944473, ...,  1.63363589,
         1.59152985,  0.8819171 ],
       ...,
       [ 1.25896197,  1.39683928,  1.65944473, ...,  0.54090285,
         1.09311245,  0.8819171 ],
       [ 0.89996372, -0.56880966,  1.65944473, ..., -0.00546367,
         0.39532807,  0.8819171 ],
       [ 1.52821065,  1.39683928,  0.75511518, ..., -0.55183018,
        -1.00024067,  0.8819171 ]])
```

X_train_std has lost its dataset properties.

```
# bringing back dataframe
X_train = pd.DataFrame(X_train_std, columns=X_train_columns)
X_train
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	-0.266781	-0.732614	-1.053544	-1.441076	-0.551830	-0.651348	-1.133893
1	-0.087281	-0.405006	-0.149214	-0.417221	-0.005464	0.212575	-1.133893
2	1.617960	1.560643	1.659445	1.630490	1.633636	1.591530	0.881917
3	0.810214	1.069231	0.755115	0.606634	-0.005464	0.926973	0.881917
4	2.066708	1.233035	1.659445	1.630490	1.087269	0.760834	0.881917
...	...	...	...	...	...	...	...
395	0.720465	-0.241202	-0.149214	0.094707	0.540903	-0.335684	0.881917
396	0.002468	-0.077397	-0.149214	0.094707	-0.551830	0.129505	0.881917
397	1.258962	1.396839	1.659445	0.606634	0.540903	1.093112	0.881917
398	0.899964	-0.568810	1.659445	-0.417221	-0.005464	0.395328	0.881917
399	1.528211	1.396839	0.755115	0.606634	-0.551830	-1.000241	0.881917

400 rows × 7 columns

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

```
y_train
```

	Chance of Admit
4	0.65
246	0.72
120	0.94
469	0.86
429	0.95
...	...
10	0.52
186	0.84
285	0.93
119	0.71
52	0.78

400 rows × 1 columns

dtype: float64

Now we can start with the modeling.

sm.OLS

```
y_train.shape
```

```
(400,)
```

```
import statsmodels.api as sm
X_train = sm.add_constant(X_train)
model = sm.OLS(y_train.values, X_train).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.836
Model:                            OLS   Adj. R-squared:                  0.833
Method:                 Least Squares   F-statistic:                     285.3
Date:                Thu, 21 Nov 2024   Prob (F-statistic):          1.50e-149
Time:                        07:48:20   Log-Likelihood:                 575.53
No. Observations:                 400   AIC:                            -1135.
Df Residuals:                     392   BIC:                            -1103.
Df Model:                           7
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const                 0.7271      0.003    250.802      0.000       0.721       0.733
GRE Score             0.0236      0.006      3.913      0.000       0.012       0.035
TOEFL Score           0.0170      0.006      2.982      0.003       0.006       0.028
University Rating     0.0078      0.005      1.715      0.087      -0.001       0.017
SOP                   0.0013      0.005      0.262      0.793      -0.008       0.011
LOR                   0.0136      0.004      3.303      0.001       0.006       0.022
CGPA                  0.0720      0.006     11.529      0.000       0.060       0.084
Research              0.0115      0.004      3.215      0.001       0.004       0.019
==============================================================================
Omnibus:                       90.534   Durbin-Watson:                   2.020
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              221.621
Skew:                          -1.115   Prob(JB):                     7.51e-49
Kurtosis:                       5.885   Cond. No.                         5.59
==============================================================================

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

- R-squared is 0.836 which indicates that appoximately 83.6% of the variance in the dependent variable which is chance of admit is explained by the independent variable.
- Adjusted R-squared is 0.833, which is close to the R-squared value, indicating that the model fits the data well and is not overly complex.
- SOP's p-value is 0.793 which is greater than the threshold of 0.05 which makes this insignificant influencing the admissions chances.

- Likewise the university rating is also not significant for the influence of addmission chances.

```
# let try to remove SOP and anayalse again
X_train_improved = X_train.drop(columns='SOP')
model1 = sm.OLS(y_train.values, X_train_improved).fit()
print(model1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.836
Model:                            OLS   Adj. R-squared:                  0.833
Method:                 Least Squares   F-statistic:                     333.6
Date:                Thu, 21 Nov 2024   Prob (F-statistic):          8.15e-151
Time:                        07:53:00   Log-Likelihood:                 575.49
No. Observations:                 400   AIC:                            -1137.
Df Residuals:                     393   BIC:                            -1109.
Df Model:                           6
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const                 0.7271      0.003    251.100      0.000       0.721       0.733
GRE Score             0.0236      0.006      3.912      0.000       0.012       0.035
TOEFL Score           0.0172      0.006      3.045      0.002       0.006       0.028
University Rating     0.0082      0.004      1.922      0.055      -0.000       0.017
LOR                   0.0140      0.004      3.533      0.000       0.006       0.022
CGPA                  0.0723      0.006     11.758      0.000       0.060       0.084
Research              0.0115      0.004      3.222      0.001       0.004       0.019
==============================================================================
Omnibus:                       89.660   Durbin-Watson:                   2.018
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              218.061
Skew:                          -1.107   Prob(JB):                     4.45e-48
Kurtosis:                       5.860   Cond. No.                         5.15
==============================================================================

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

From the improved model's summary we can confirm that SOP was redundant as deleting it did not affect the R-squared and adj-R-squared by any means.

P-value of University Rating become 0.055 from 0.087, which means we have to remove this also and check the model's summary again to confirm this is redundant or not.

```
X_train_improved_v2 = X_train.drop(columns='University Rating')
model2 = sm.OLS(y_train.values, X_train_improved_v2).fit()
print(model2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.835
```

```
Model:                            OLS   Adj. R-squared:                  0.832
Method:                 Least Squares   F-statistic:                     330.7
Date:                Thu, 21 Nov 2024   Prob (F-statistic):          3.41e-150
Time:                        07:58:03   Log-Likelihood:                 574.03
No. Observations:                 400   AIC:                            -1134.
Df Residuals:                     393   BIC:                            -1106.
Df Model:                           6
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const           0.7271      0.003    250.185      0.000       0.721       0.733
GRE Score       0.0241      0.006      3.984      0.000       0.012       0.036
TOEFL Score     0.0180      0.006      3.169      0.002       0.007       0.029
SOP             0.0041      0.005      0.899      0.369      -0.005       0.013
LOR             0.0146      0.004      3.567      0.000       0.007       0.023
CGPA            0.0733      0.006     11.777      0.000       0.061       0.086
Research        0.0122      0.004      3.408      0.001       0.005       0.019
==============================================================================
Omnibus:                       91.078   Durbin-Watson:                   2.018
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              224.228
Skew:                          -1.119   Prob(JB):                     2.04e-49
Kurtosis:                       5.906   Cond. No.                         5.20
==============================================================================

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

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calculate_vif(dataset, col):
  dataset = dataset.drop(columns = col, axis = 1 )
  vif = pd.DataFrame()
  vif['features'] = dataset.columns
  vif['VIF_Value'] = [variance_inflation_factor(dataset.values, i) for i in range(dataset
  return vif
```

```
calculate_vif(X_train_improved_v2,[])
```

	features	VIF_Value
0	const	1.000000
1	GRE Score	4.317809
2	TOEFL Score	3.813095
3	SOP	2.415445
4	LOR	1.991263
5	CGPA	4.584347
6	Research	1.512508

VIF(Variance Inflation Factor)

- VIF score of an independent variable represents how well the variable is explained by other independent variables.
- So, the closer the R^2 value to 1, the higher the value of VIF and higher the multicollinearity with the particular independent variables.
- high vif -> high multicollinearity -> independent variable is high explained by other independent variables
- low vif -> low multicollinearity -> independent variable has little correlation with the other variables.

VIF is calculated as 1 / (1 - R^2)

VIF looks fine and hence, we can go ahead with the next predictions

```
# transforming the tests
X_test_std = std.transform(X_test)
X_test = pd.DataFrame(X_test_std, columns=X_train_columns)
X_test = sm.add_constant(X_test)
```

```
X_test_del = list(set(X_test.columns) - set(X_train_improved_v2.columns))
# print(f'Dropping {X_test_del} from test set')
```

```
X_test_new = X_test.drop(columns=X_test_del)
```

```
# prediction from the clean model

pred = model2.predict(X_test_new)

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

print('Mean Absolute Error', mean_absolute_error(y_test.values, pred))
print('Mean Squared Error', np.sqrt(mean_squared_error(y_test.values, pred)))
```

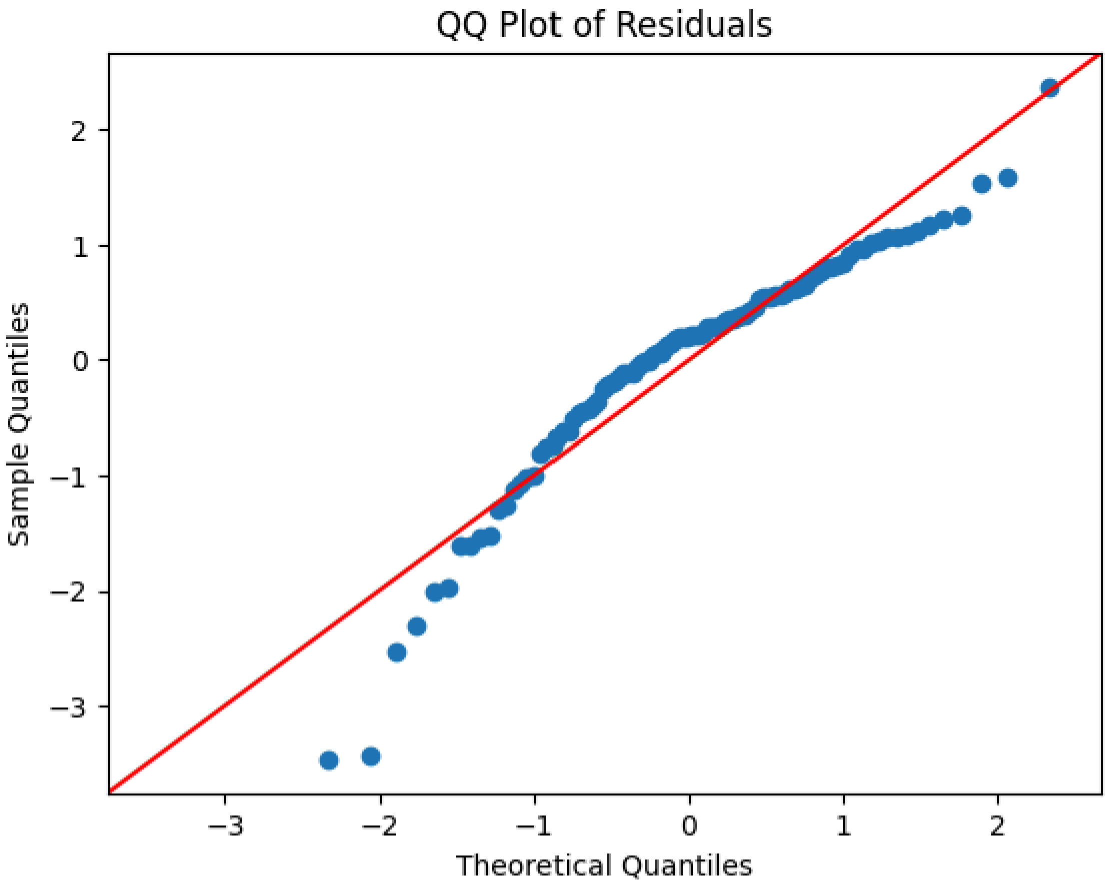
```
Mean Absolute Error 0.05025797479861531
Mean Squared Error 0.06755244450283296
```

Mean of Residuals

```
residuals = y_test.values - pred
print('Mean of Residuals', residuals.mean())
```

```
Mean of Residuals 0.0011312301983650514
```

```
## Quantile-Quantile plot to assess if a set of residuals follows a normal distribution

sm.qqplot(residuals, line='45', fit=True)

plt.title("QQ Plot of Residuals")
plt.show()
```

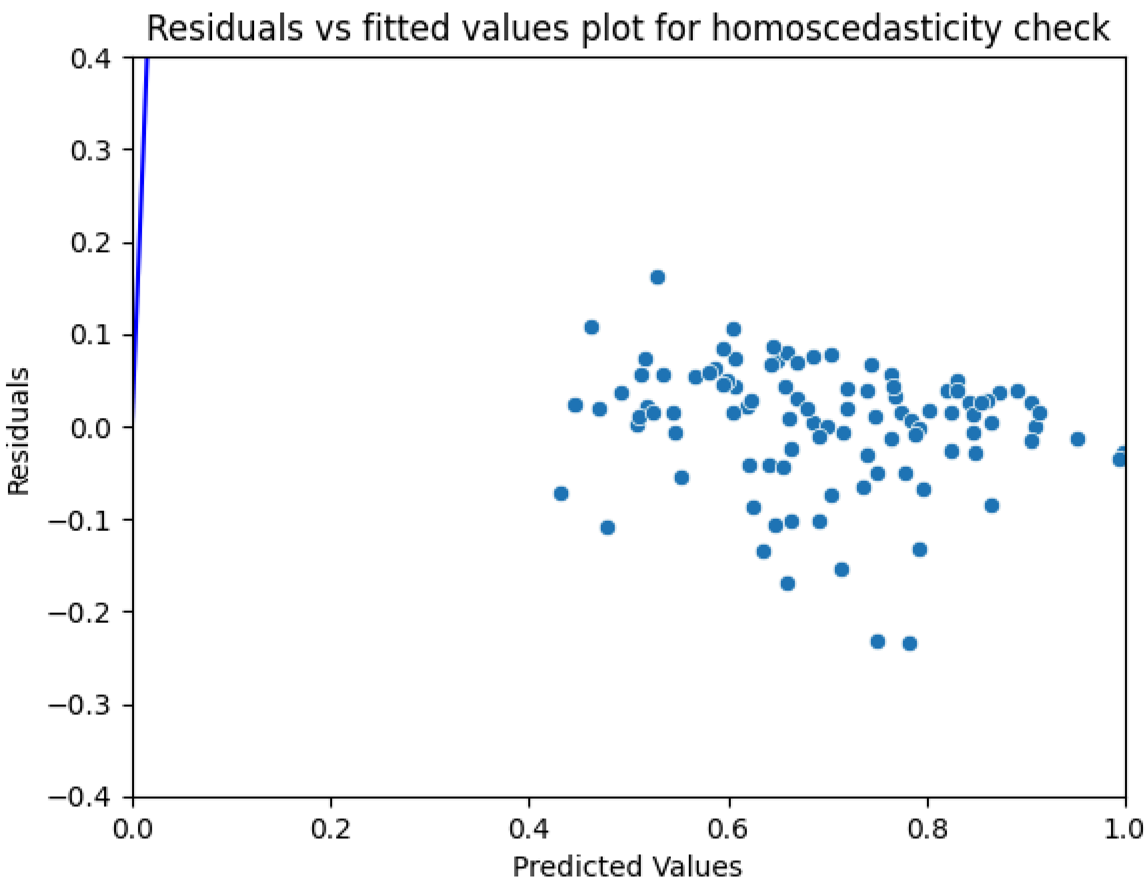


Majority of the points follows the 45 deg red line indicating the residuals are normally distributred in the middle range.

But there are visible deviation in both of the ends which indicates outliers.

Test for Homoscedasticity

```
p = sns.scatterplot(x = pred, y = residuals)
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.ylim(-0.4, 0.4)
plt.xlim(0,1)
p = sns.lineplot([0,26], color = 'blue')
p = plt.title('Residuals vs fitted values plot for homoscedasticity check')
```




```
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
name = ['F statistic', 'p-value']
test = sms.het_goldfeldquandt(residuals, X_test_new)
lzip(name, test)
```

```
[('F statistic', 0.8753150885645835), ('p-value', 0.667834445226687)]
```

```
p = sns.distplot(residuals, kde = True)
p = plt.title('Residuals Distribution')
```