In [4]:
```
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
df = pd.read_csv("Jamboree_admission.csv")
df.head()
```

Out[4]:

| | Serial No. | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research | Admission_chances |
|---|---|---|---|---|---|---|---|---|---|
| **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 |

In [5]:
```
df.shape
```

Out[5]:
```
(500, 9)
```

# check if there are any null columns in the dataset

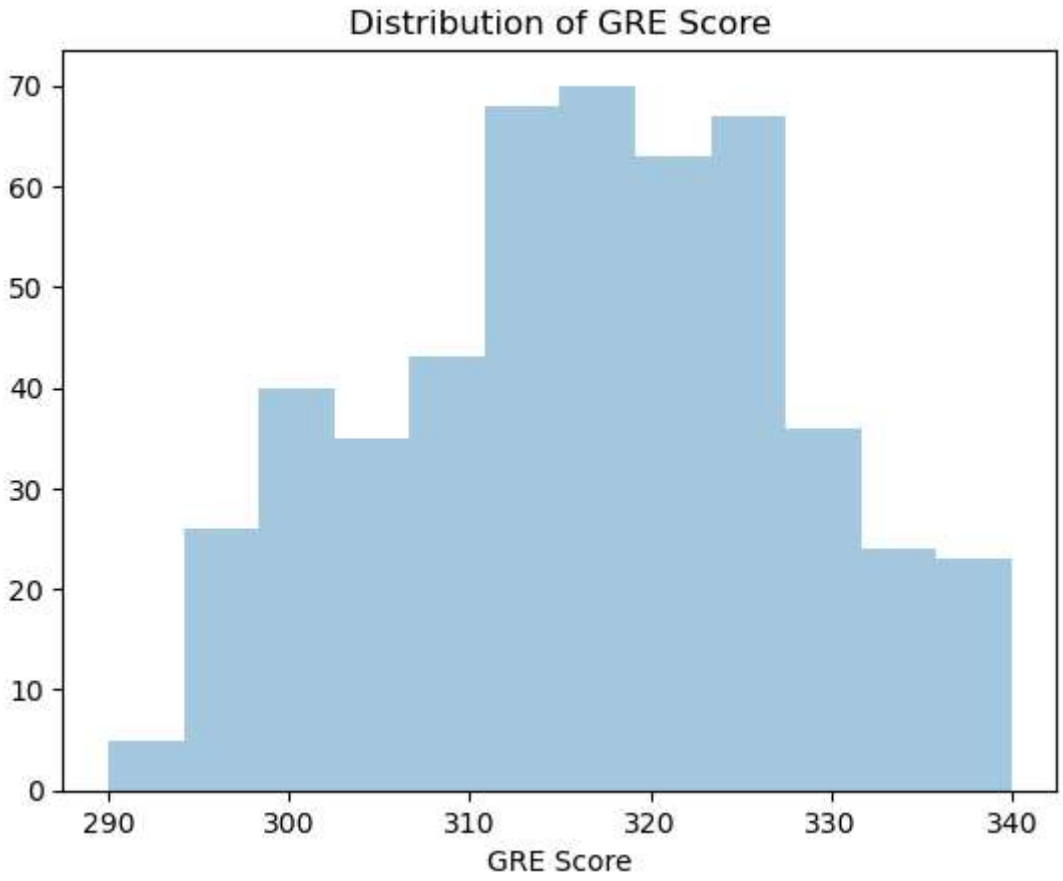
In [6]:
```
df = df.drop(['Serial No.'],axis = 1)
df.isnull().sum()
```

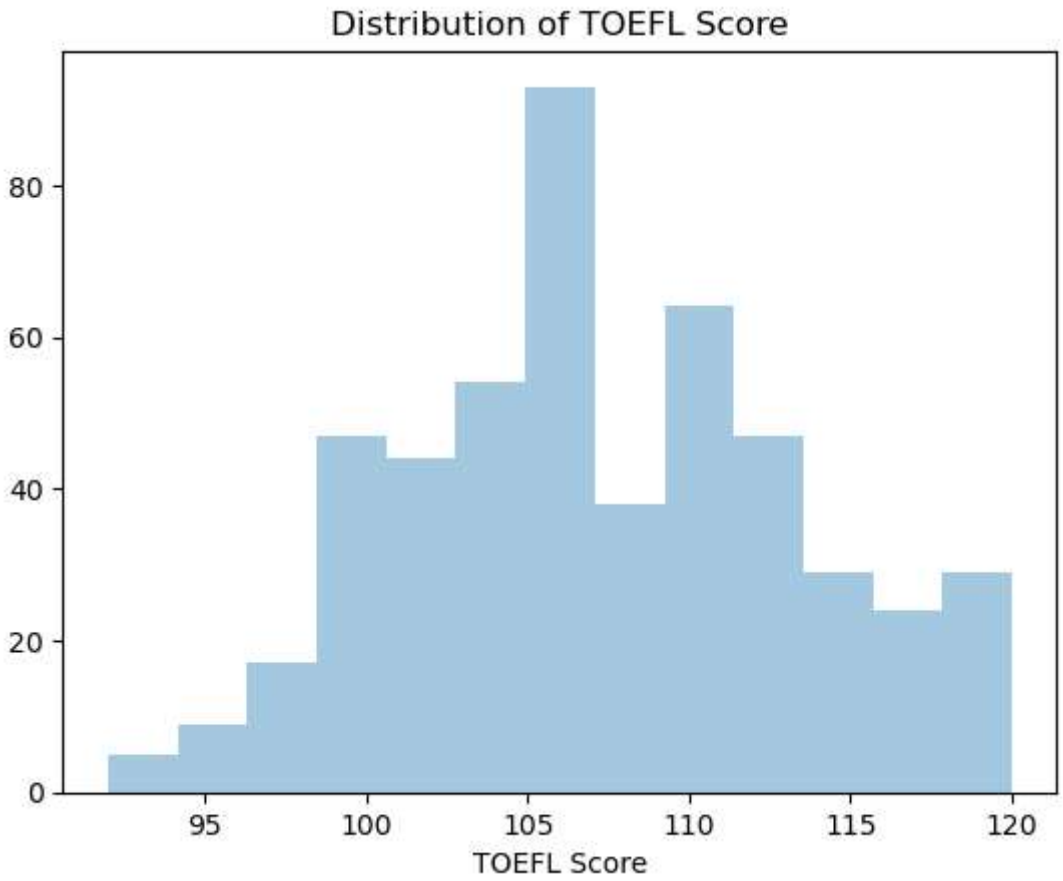
Out[6]:
```
GRE Score           0
TOEFL Score         0
University Rating   0
SOP                 0
LOR                 0
CGPA                0
Research            0
Admission_chances   0
dtype: int64
```

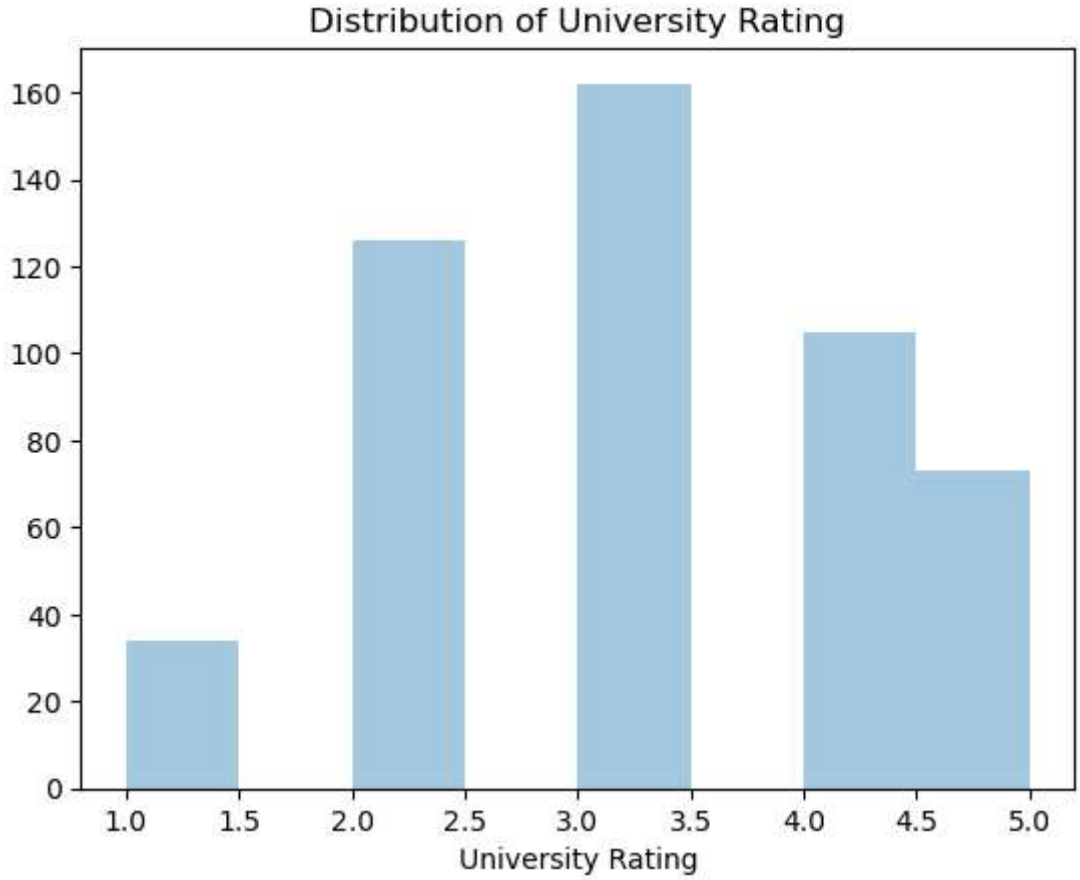
Distribution of variables of graduate applicants

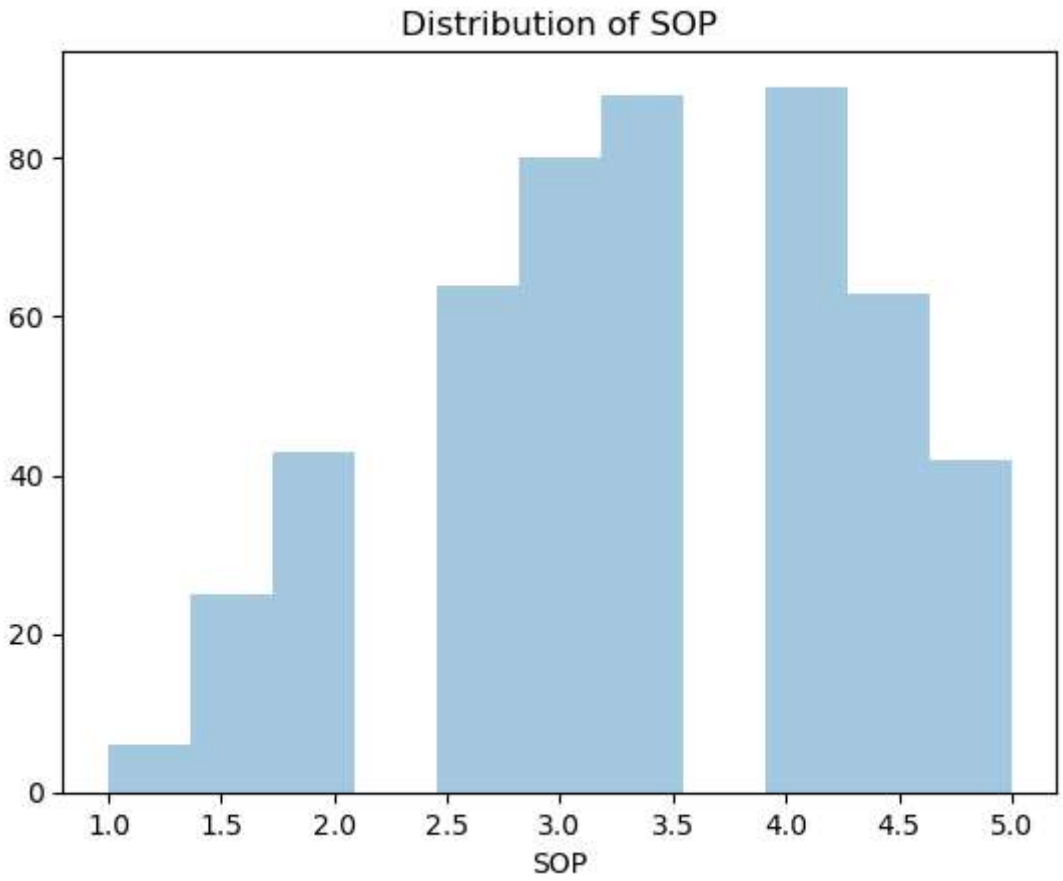
In [7]:
```
import matplotlib.pyplot as plt
import seaborn as sns

fig = sns.distplot(df['GRE Score'], kde = False)
plt.title("Distribution of GRE Score")
plt.show()

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

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

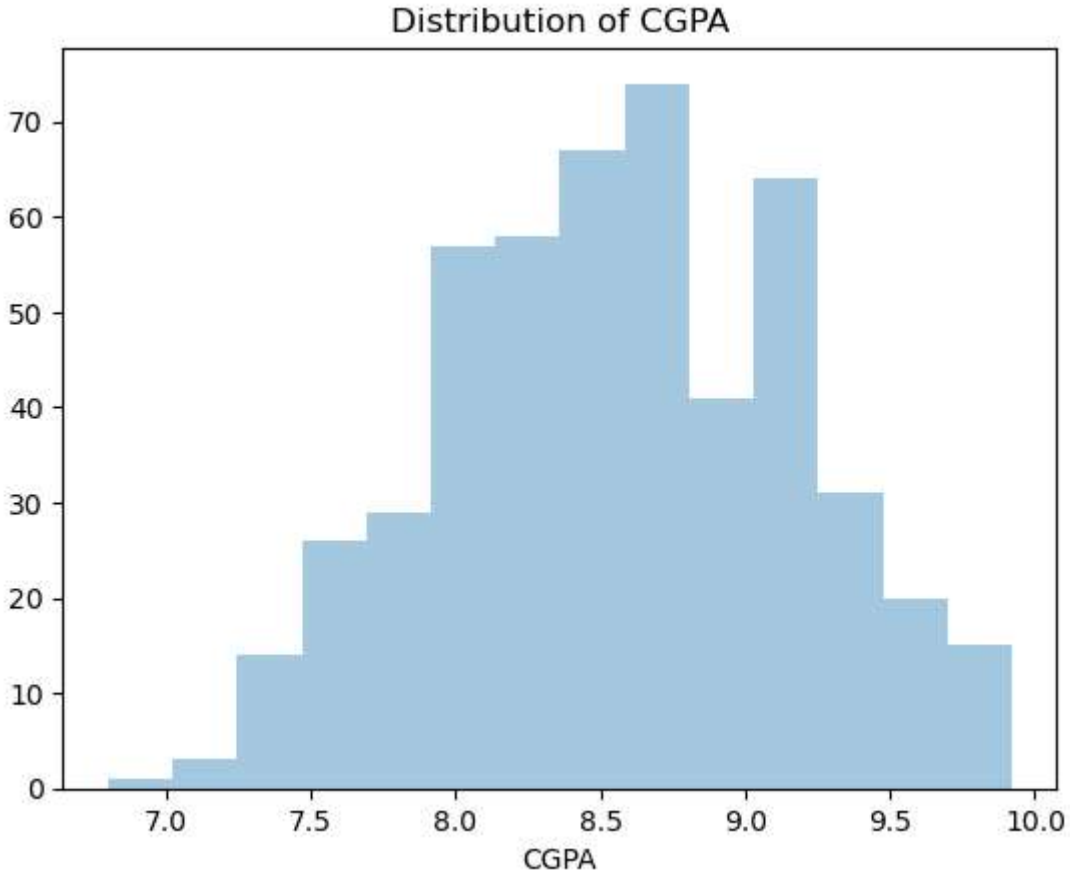
```
fig = sns.distplot(df['SOP'], kde = False)
plt.title("Distribution of SOP")
plt.show()

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




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

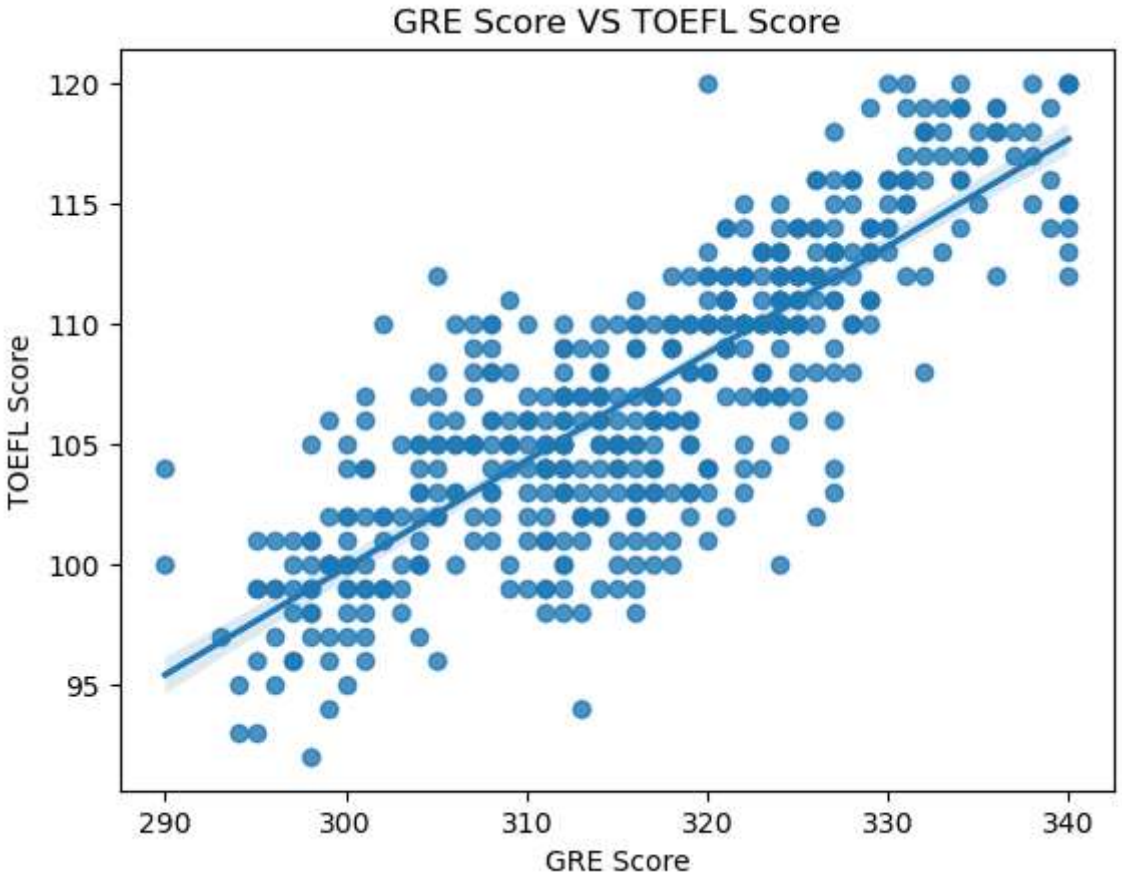

Distribution of University Rating
0
20
40
60
80
100
120
140
160
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
University Rating



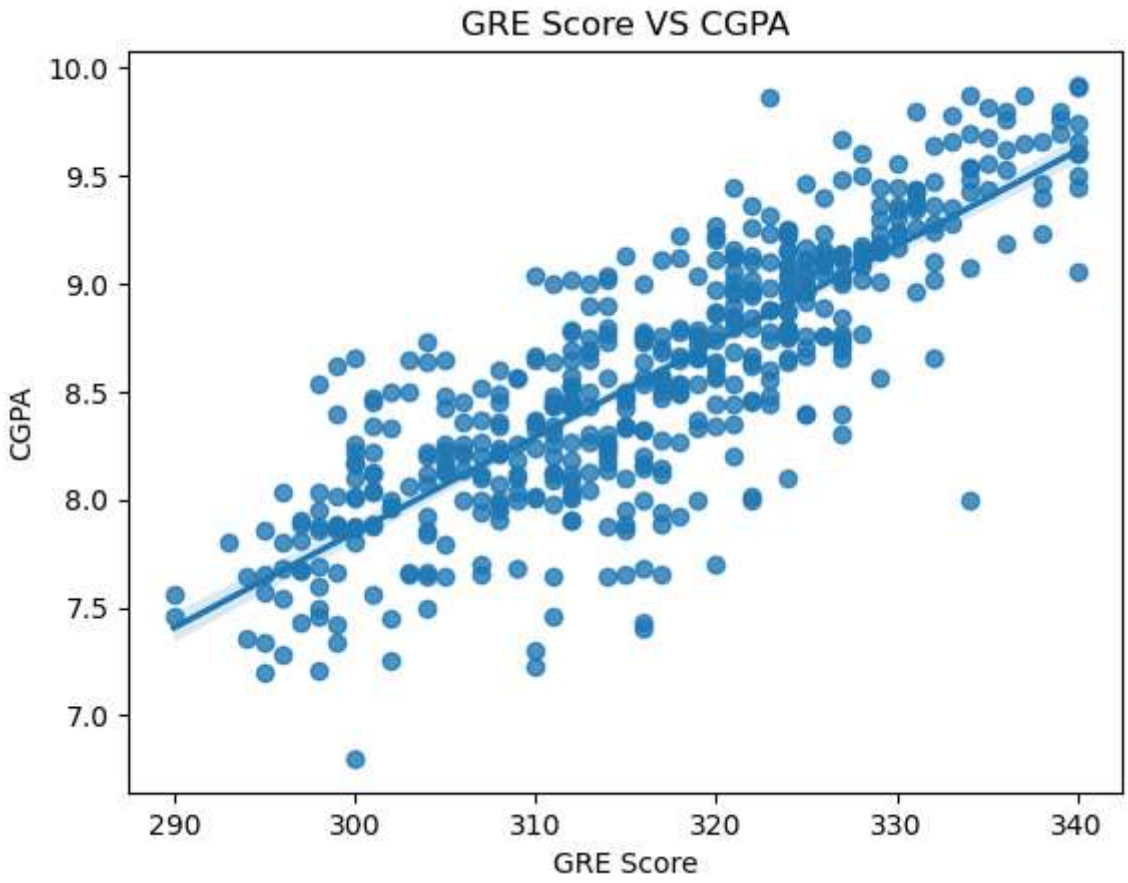




check corelations among variables

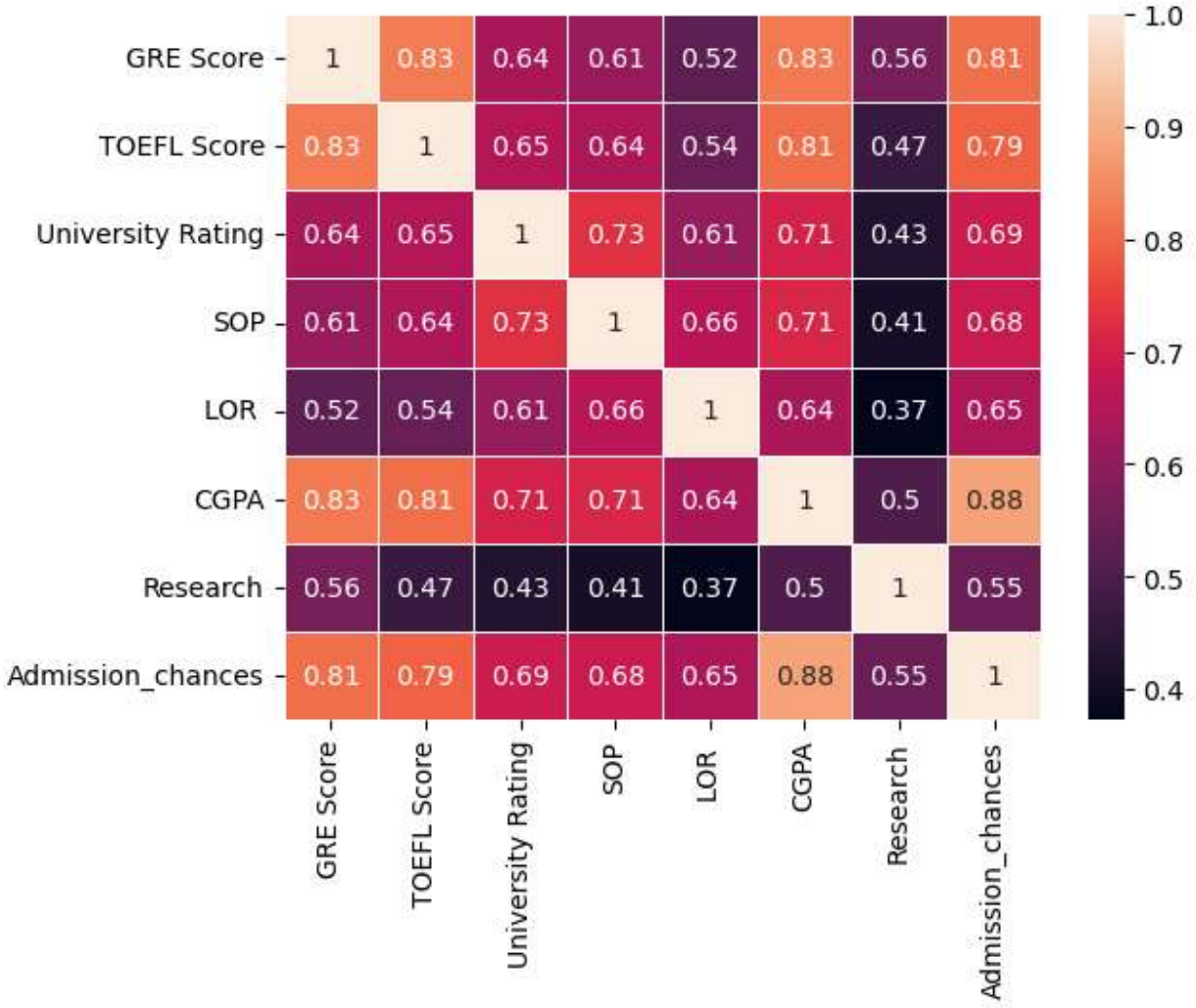
In [8]:
```
fig = sns.regplot(x= "GRE Score", y = "TOEFL Score", data = df)
plt.title("GRE Score VS TOEFL Score")
plt.show()
```




In [9]:
```
fig = sns.regplot(x= "GRE Score", y = "CGPA", data = df)
plt.title("GRE Score VS CGPA")
plt.show()
```




In [10]:
```
import numpy as np
corr = df.corr()
sns.heatmap(corr,linewidths= 0.5,annot = True)
plt.show()
```




In [11]:
```
from sklearn.model_selection import train_test_split

x= df.drop(['Admission_chances'], axis = 1)
y = df['Admission_chances']
```

In [12]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.20, shuffle = True)
```

In [13]:
```
x_train
```

Out[13]:

| | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research |
|---|---|---|---|---|---|---|---|
| **428** | 316 | 103 | 2 | 2.0 | 4.5 | 8.74 | 0 |
| **90** | 318 | 106 | 2 | 4.0 | 4.0 | 7.92 | 1 |
| **125** | 300 | 100 | 3 | 2.0 | 3.0 | 8.66 | 1 |
| **244** | 314 | 107 | 2 | 2.5 | 4.0 | 8.56 | 0 |
| **285** | 331 | 116 | 5 | 4.0 | 4.0 | 9.26 | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **482** | 328 | 113 | 4 | 4.0 | 2.5 | 8.77 | 1 |
| **116** | 299 | 102 | 3 | 4.0 | 3.5 | 8.62 | 0 |
| **149** | 311 | 106 | 2 | 3.5 | 3.0 | 8.26 | 1 |
| **361** | 334 | 116 | 4 | 4.0 | 3.5 | 9.54 | 1 |
| **486** | 319 | 102 | 3 | 2.5 | 2.5 | 8.37 | 0 |

400 rows × 7 columns

In [14]:
```
y_train
```

Out[14]:
```
428    0.69
90     0.64
125    0.64
244    0.63
285    0.93
       ...
482    0.83
116    0.56
149    0.79
361    0.93
486    0.68
Name: Admission_chances, Length: 400, dtype: float64
```

In [15]:
```
from sklearn.preprocessing import StandardScaler
x_train_columns = x_train.columns
std = StandardScaler()
x_train_std = std.fit_transform(x_train)
```

In [16]:
```
x_train_std
```

Out[16]:
```
array([[-0.02430865, -0.6708421 , -0.98909971, ...,  1.07671533,
         0.28288912, -1.1055416 ],
       [ 0.14932457, -0.18706174, -0.98909971, ...,  0.54103606,
        -1.07524367,  0.90453403],
       [-1.41337444, -1.15462246, -0.11764622, ..., -0.53032248,
         0.15038836,  0.90453403],
       ...,
       [-0.45839171, -0.18706174, -0.98909971, ..., -0.53032248,
        -0.51211544,  0.90453403],
       [ 1.53839036,  1.42553947,  0.75380727, ...,  0.00535679,
         1.60789672,  0.90453403],
       [ 0.23614118, -0.83210222, -0.11764622, ..., -1.06600175,
        -0.32992689, -1.1055416 ]])
```

In [17]:
```
x_train = pd.DataFrame(x_train_std , columns = x_train_columns)
```

In [18]:
```
x_train
```

Out[18]:

| | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research |
|---|---|---|---|---|---|---|---|
| **0** | -0.024309 | -0.670842 | -0.989100 | -1.411370 | 1.076715 | 0.282889 | -1.105542 |
| **1** | 0.149325 | -0.187062 | -0.989100 | 0.632243 | 0.541036 | -1.075244 | 0.904534 |
| **2** | -1.413374 | -1.154622 | -0.117646 | -1.411370 | -0.530322 | 0.150388 | 0.904534 |
| **3** | -0.197942 | -0.025802 | -0.989100 | -0.900467 | 0.541036 | -0.015238 | -1.105542 |
| **4** | 1.277941 | 1.425539 | 1.625261 | 0.632243 | 0.541036 | 1.144144 | 0.904534 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **395** | 1.017491 | 0.941759 | 0.753807 | 0.632243 | -1.066002 | 0.332577 | 0.904534 |
| **396** | -1.500191 | -0.832102 | -0.117646 | 0.632243 | 0.005357 | 0.084138 | -1.105542 |
| **397** | -0.458392 | -0.187062 | -0.989100 | 0.121340 | -0.530322 | -0.512115 | 0.904534 |
| **398** | 1.538390 | 1.425539 | 0.753807 | 0.632243 | 0.005357 | 1.607897 | 0.904534 |
| **399** | 0.236141 | -0.832102 | -0.117646 | -0.900467 | -1.066002 | -0.329927 | -1.105542 |

400 rows × 7 columns

# Calculate MSE with different LR models

In [19]:
```
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso,Ridge,LinearRegression
from sklearn.metrics import mean_squared_error

models = [['Linear Regression :', LinearRegression()],
          ['Lasso Regression :', Lasso(alpha = 0.1)], # try with different alpha value
          ['Ridge Regression :', Ridge(alpha = 1.0)],] # try with different alpha valu


print("Results without removing features with multicollinearity...")

for name,model in models:
    model.fit(x_train,y_train.values)
    predictions = model.predict(std.transform(x_test))
    print(name, (np.sqrt(mean_squared_error(y_test,predictions))))
```

```
Results without removing features with multicollinearity...
Linear Regression : 0.06819790464566258
Lasso Regression : 0.12081600300122337
Ridge Regression : 0.06819998952754852
```

Importing statistical model

In [20]:
```
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:                       0.835
Model:                            OLS   Adj. R-squared:                  0.832
Method:                 Least Squares   F-statistic:                     282.8
Date:                Tue, 10 Oct 2023   Prob (F-statistic):          6.27e-149
Time:                        08:31:22   Log-Likelihood:                 576.58
No. Observations:                 400   AIC:                            -1137.
Df Residuals:                     392   BIC:                            -1105.
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const                0.7215      0.003    249.532      0.000       0.716       0.727
GRE Score            0.0219      0.006      3.469      0.001       0.009       0.034
TOEFL Score          0.0200      0.006      3.389      0.001       0.008       0.032
University Rating    0.0040      0.005      0.850      0.396      -0.005       0.013
SOP                  0.0036      0.005      0.753      0.452      -0.006       0.013
LOR                  0.0130      0.004      3.183      0.002       0.005       0.021
CGPA                 0.0709      0.006     11.254      0.000       0.059       0.083
Research             0.0122      0.004      3.414      0.001       0.005       0.019
==============================================================================
Omnibus:                       87.346   Durbin-Watson:                   2.045
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              199.020
Skew:                          -1.106   Prob(JB):                     6.07e-44
Kurtosis:                       5.654   Cond. No.                         5.79
==============================================================================

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

# H0 : weight = 0 (independent var is not influencing)

# HA : weight != 0 (independent var is influencing target var)

P_value of SOP & university rating are greater than alpha(0.05) so they are not impacting but among these two we reject SOP cause that is more ingignificant

In [22]:
```
x_train_new = x_train.drop(columns = 'SOP')
```

In [23]:
```
model1 = sm.OLS(y_train.values, x_train_new).fit()
print(model1.summary())
```

OLS Regression Results

| | | | |
|---|---|---|---|
| Dep. Variable: | y | R-squared: | 0.834 |
| Model: | OLS | Adj. R-squared: | 0.832 |
| Method: | Least Squares | F-statistic: | 330.2 |
| Date: | Tue, 10 Oct 2023 | Prob (F-statistic): | 4.38e-150 |
| Time: | 08:44:28 | Log-Likelihood: | 576.29 |
| No. Observations: | 400 | AIC: | -1139. |
| Df Residuals: | 393 | BIC: | -1111. |
| Df Model: | 6 | | |
| Covariance Type: | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| const | 0.7215 | 0.003 | 249.670 | 0.000 | 0.716 | 0.727 |
| GRE Score | 0.0217 | 0.006 | 3.448 | 0.001 | 0.009 | 0.034 |
| TOEFL Score | 0.0205 | 0.006 | 3.507 | 0.001 | 0.009 | 0.032 |
| University Rating | 0.0052 | 0.004 | 1.199 | 0.231 | -0.003 | 0.014 |
| LOR | 0.0139 | 0.004 | 3.581 | 0.000 | 0.006 | 0.022 |
| CGPA | 0.0717 | 0.006 | 11.525 | 0.000 | 0.059 | 0.084 |
| Research | 0.0122 | 0.004 | 3.426 | 0.001 | 0.005 | 0.019 |

| | | | |
|---|---|---|---|
| Omnibus: | 85.790 | Durbin-Watson: | 2.039 |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 194.140 |
| Skew: | -1.090 | Prob(JB): | 6.97e-43 |
| Kurtosis: | 5.626 | Cond. No. | 5.35 |

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

calculate VIF for every variables

In [26]:
```
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(dat
    return vif
```

In [27]:
```
calculate_VIF(x_train_new,[])
```

Out[27]:

| | features | VIF_Value |
|---|---|---|
| 0 | const | 1.000000 |
| 1 | GRE Score | 4.747657 |
| 2 | TOEFL Score | 4.091238 |
| 3 | University Rating | 2.261098 |
| 4 | LOR | 1.811046 |
| 5 | CGPA | 4.634133 |
| 6 | Research | 1.524826 |

In [28]:
```
x_test_std = std.transform(x_test)
x_test = pd.DataFrame(x_test_std, columns = x_train_columns)
```

In [29]:
```
x_test = sm.add_constant(x_test)
```

In [30]:
```
x_test_del = list(set(x_test.columns).difference(set(x_train_new.columns)))
```

In [31]:
```
print(f'Dropping {x_test_del} from test set')
```

```
Dropping ['SOP'] from test set
```

In [32]:
```
x_test_new = x_test.drop(columns = x_test_del)
```

In [33]:
```
#prediction from the clean model
pred = model1.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("Root Mean Square Error", np.sqrt(mean_squared_error(y_test.values,pred)))
```

```
Mean Absolute Error  0.04658398170083695
Root Mean Square Error 0.06804667020613017
```

# Mean of Residuals

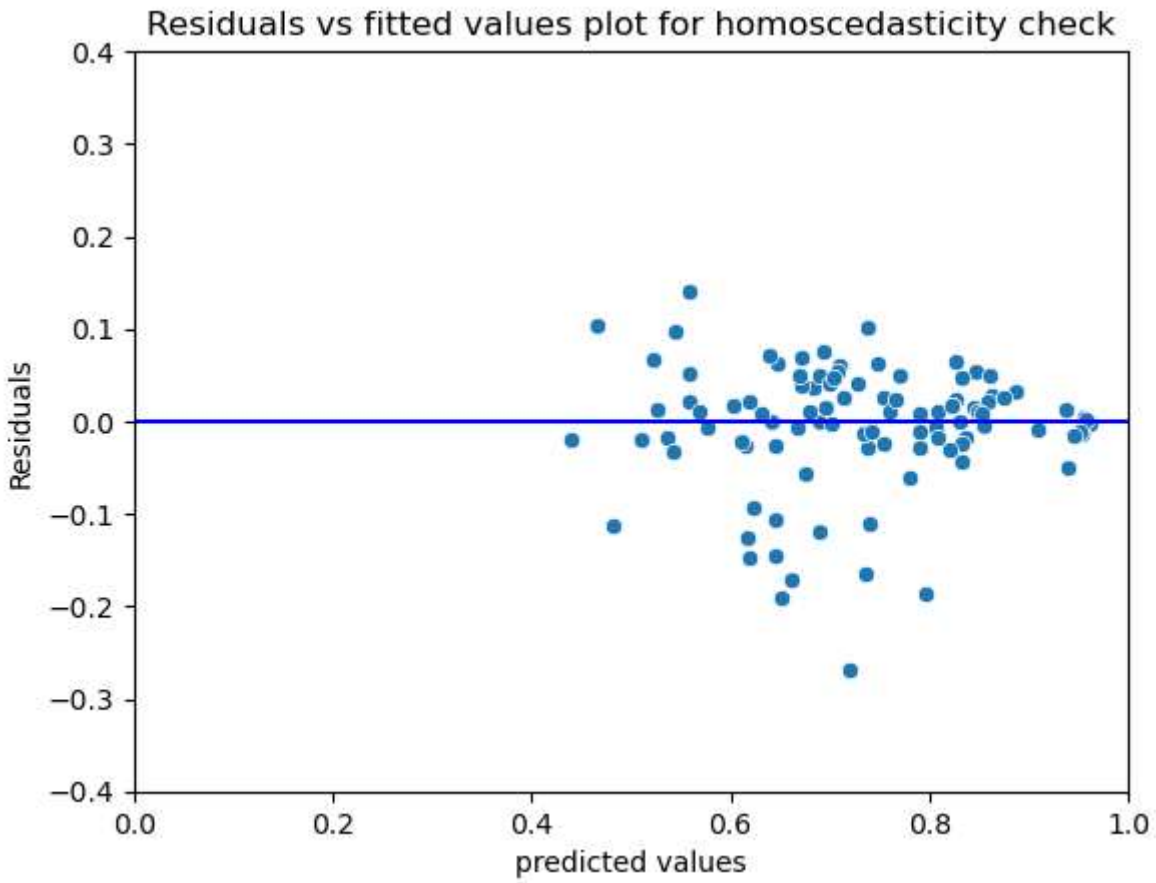
In [34]:
```
residuals = y_test.values-pred
mean_residuals = np.mean(residuals)
print("Mean Of Residuals {}".format(mean_residuals))
```

```
Mean Of Residuals -0.0053687900528389286
```

# Test of Homoscedasticity

In [35]:
```
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(x=[0,26], y = [0,0], color = 'blue')
p = plt.title('Residuals vs fitted values plot for homoscedasticity check')
```



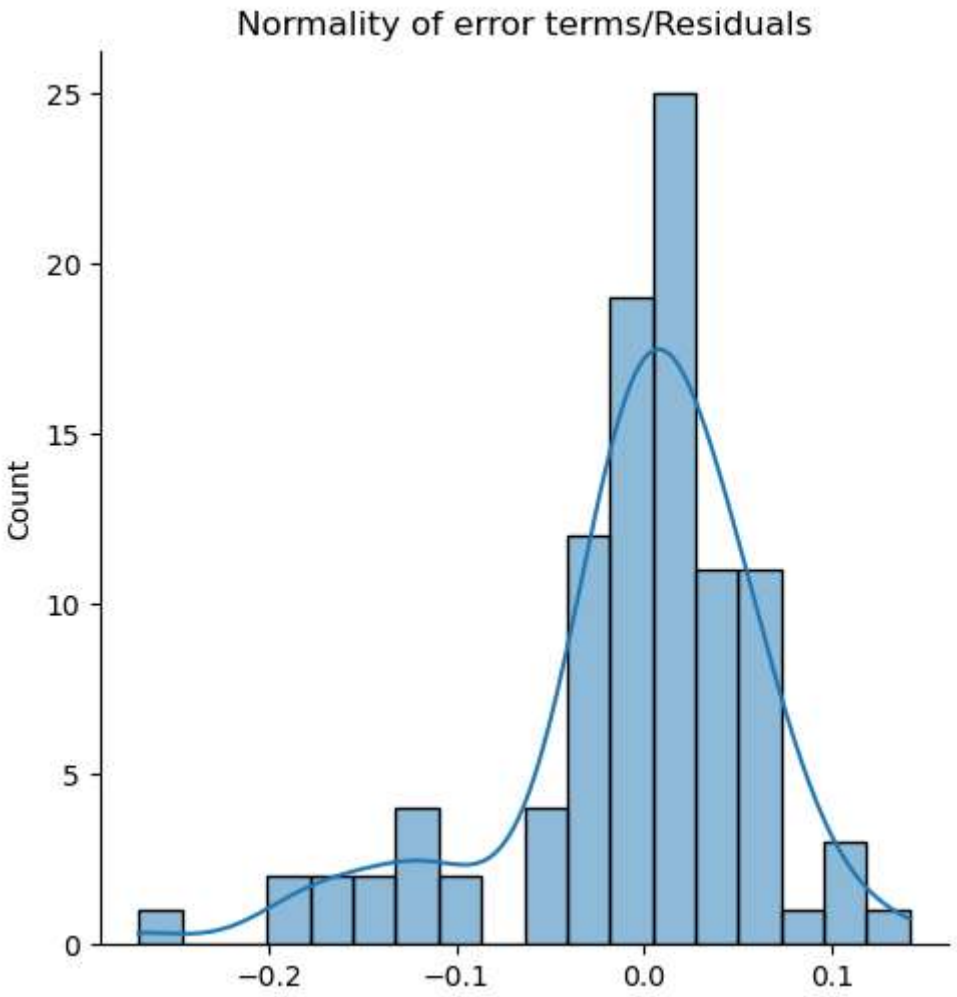

In [36]:
```
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
name = ['F statistic', 'p-value']
test = sms.het_goldfeldquandt(residuals,x_test)
lzip(name,test)
```

Out[36]:
```
[('F statistic', 0.4412687691627571), ('p-value', 0.9953389489953618)]
```

Here null hypothesis is error term are homoscedastic since p_value>0.5 we fail to reject null hypothesis

# Normality of Residuals

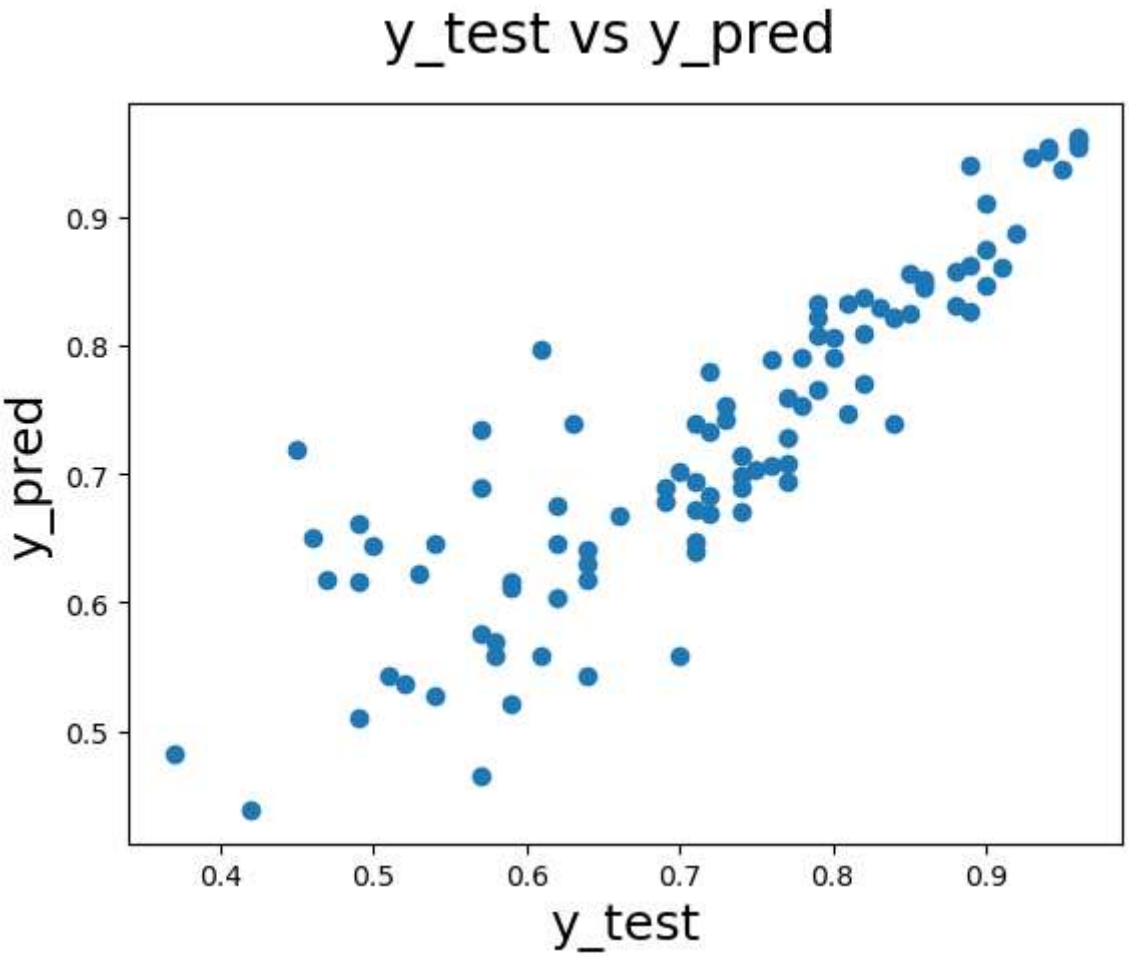
In [37]:
```
p = sns.displot(residuals,kde = True)
p = plt.title('Normality of error terms/Residuals')
```




# Q-Q Plot

In [38]:
```
#plotting y_test vs y_pred to understand the spread

fig = plt.figure()
plt.scatter(y_test.values, pred)
fig.suptitle('y_test vs y_pred', fontsize = 20)
plt.xlabel('y_test', fontsize = 18)
plt.ylabel('y_pred', fontsize = 18)
```

Out[38]:
```
Text(0, 0.5, 'y_pred')
```




In [ ]: