mboree-education-linear-regression

November 10, 2024

[2]:
```
# importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

[3]:
```
# read the dataset
df = pd.read_csv('https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/
↪000/001/839/original/Jamboree_Admission.csv')
df.head()
```

[3]:
```
   Serial No.  GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  \
0           1        337          118                  4  4.5   4.5  9.65
1           2        324          107                  4  4.0   4.5  8.87
2           3        316          104                  3  3.0   3.5  8.00
3           4        322          110                  3  3.5   2.5  8.67
4           5        314          103                  2  2.0   3.0  8.21

   Research  Chance of Admit
0         1              0.92
1         1              0.76
2         1              0.72
3         1              0.80
4         0              0.65
```

[ ]:
```
# getting the no of rows and columns
df.shape
```

[ ]: (500, 9)

[ ]:
```
# getting the info of the dataset
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
```

[ ]:
```python
# checking the null values in the dataset
df.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
```

[ ]:
```python
# checking the unique values
df.nunique()
```

[ ]:
```
Serial No.           500
GRE Score             49
TOEFL Score           29
University Rating      5
SOP                    9
LOR                    9
CGPA                 184
Research               2
Chance of Admit       61
dtype: int64
```

[4]:
```python
# drop the irrelevant column
df = df.drop('Serial No.',axis = 1)
df.head()
```

[4]:

```
   GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  Research  \
0        337          118                  4  4.5   4.5  9.65         1
1        324          107                  4  4.0   4.5  8.87         1
2        316          104                  3  3.0   3.5  8.00         1
3        322          110                  3  3.5   2.5  8.67         1
4        314          103                  2  2.0   3.0  8.21         0

   Chance of Admit
0             0.92
1             0.76
2             0.72
3             0.80
4             0.65
```

Column Profiling:
GRE Scores (out of 340)
TOEFL Scores (out of 120)
University Rating (out of 5)
SOP: Statement of Purpose and
LOR: Letter of Recommendation Strength (out of 5)
Undergraduate GPA (out of 10)
Research Experience (either 0 or 1)
Chance of Admit (ranging from 0 to 1)

[ ]:

```
# getting columns name
df.columns
```

[ ]:

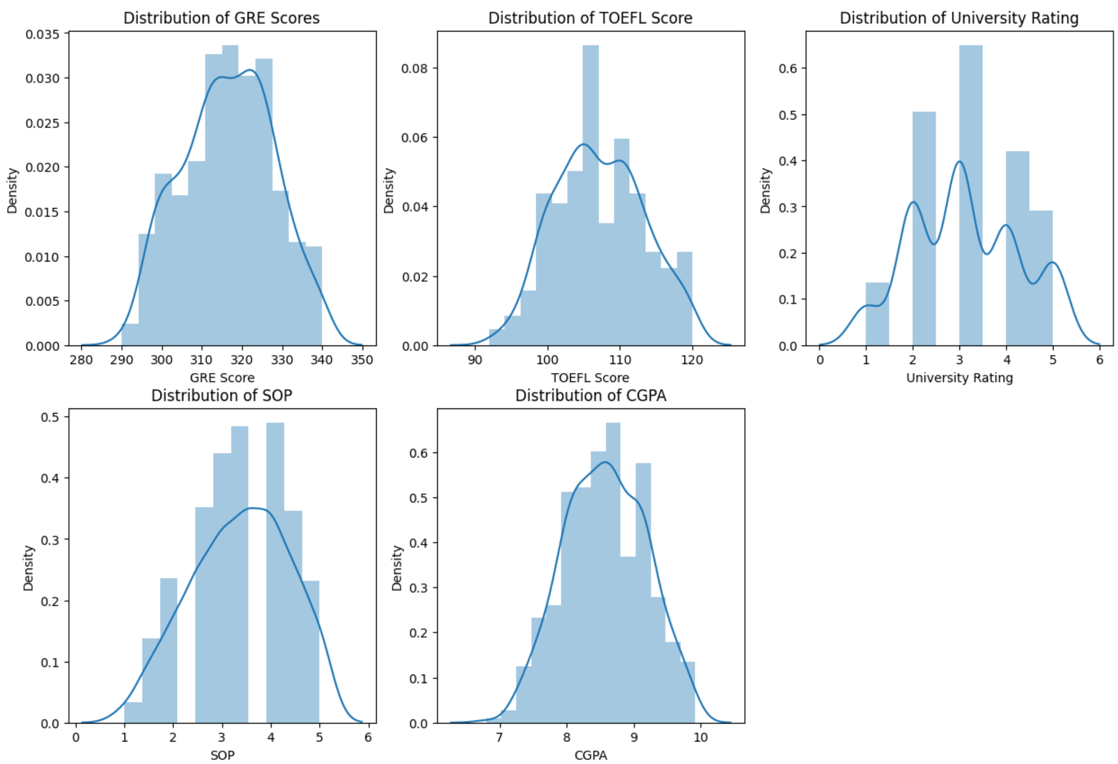
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

Distributions of the variables of graduate applicants

[ ]:

```
fig =plt.figure(figsize = (15,10))
plt.subplot(2,3,1)

sns.distplot(df['GRE Score'])
plt.title("Distribution of GRE Scores")

plt.subplot(2,3,2)
sns.distplot(df['TOEFL Score'])
plt.title("Distribution of TOEFL Score")

plt.subplot(2,3,3)
sns.distplot(df['University Rating'])
plt.title("Distribution of University Rating")

plt.subplot(2,3,4)
```

```
sns.distplot(df['SOP'])
plt.title("Distribution of SOP")

plt.subplot(2,3,5)
sns.distplot(df['CGPA'])
plt.title("Distribution of CGPA")

plt.show()
```

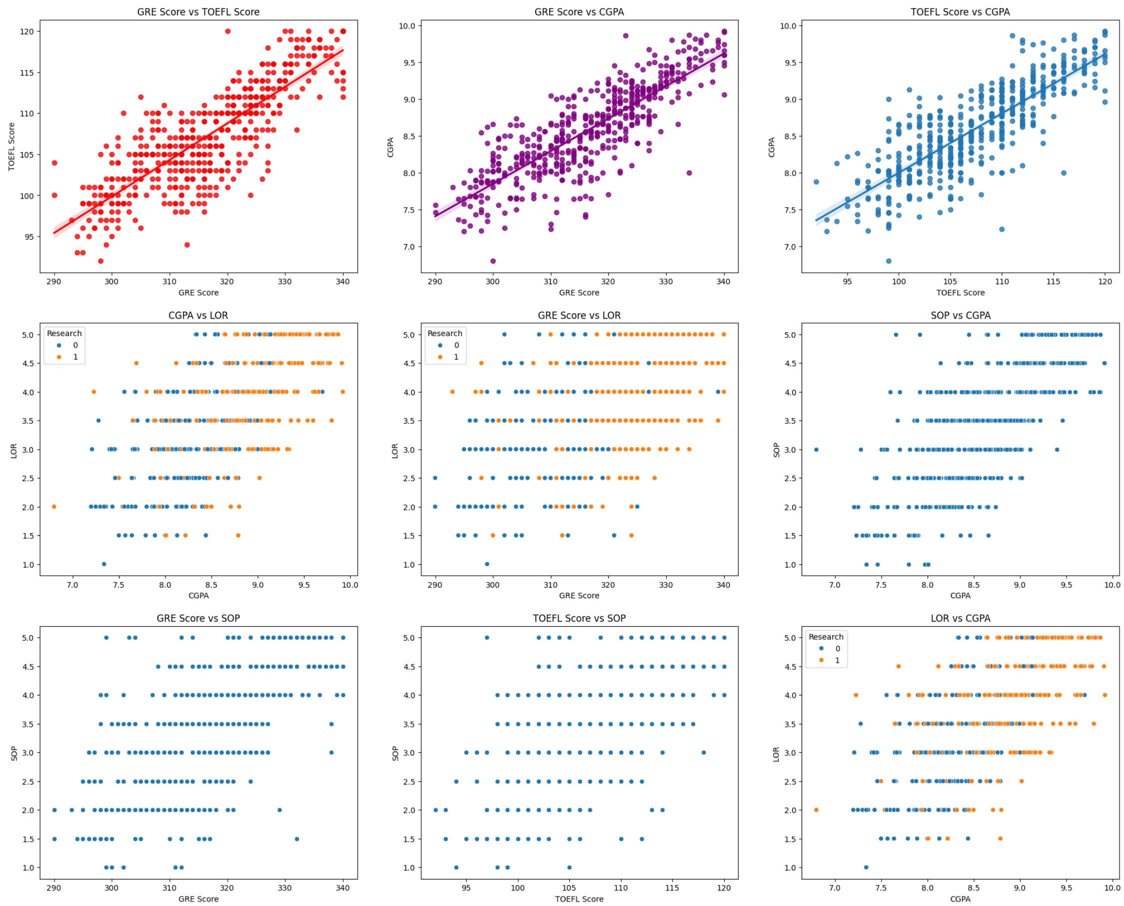


Students have varied qualifications who have applied for the university

0.0.1 Checking the relation between different features

[ ]:
```
plt.figure(figsize = (25,20))
plt.subplot(3,3,1)
sns.regplot(x = 'GRE Score', y = 'TOEFL Score', data = df, color = 'red')
plt.title("GRE Score vs TOEFL Score")

plt.subplot(3,3,2)
sns.regplot(x = 'GRE Score', y = 'CGPA', data = df, color = 'purple')
plt.title("GRE Score vs CGPA")

plt.subplot(3,3,3)
```

```
sns.regplot(x = 'TOEFL Score', y = 'CGPA', data = df)
plt.title("TOEFL Score vs CGPA")

plt.subplot(3,3,4)
sns.scatterplot(x="CGPA", y="LOR ", data=df, hue="Research")
plt.title("CGPA vs LOR")

plt.subplot(3,3,5)
sns.scatterplot(x="GRE Score", y="LOR ", data=df, hue="Research")
plt.title("GRE Score vs LOR")

plt.subplot(3,3,6)
sns.scatterplot(x="CGPA", y="SOP", data=df)
plt.title("SOP vs CGPA")

plt.subplot(3,3,7)
sns.scatterplot(x="GRE Score", y="SOP", data=df)
plt.title("GRE Score vs SOP")

plt.subplot(3,3,8)
sns.scatterplot(x="TOEFL Score", y="SOP", data=df)
plt.title("TOEFL Score vs SOP")

plt.subplot(3,3,9)
sns.scatterplot(x="CGPA", y="LOR ", data=df, hue="Research")
plt.title("LOR vs CGPA")


plt.show()
```



1. People with higher GRE Scores also have higher TOEFL Scores, both TOEFL and GRE have a verbal

2. people with higher CGPA usually have higher GRE scores

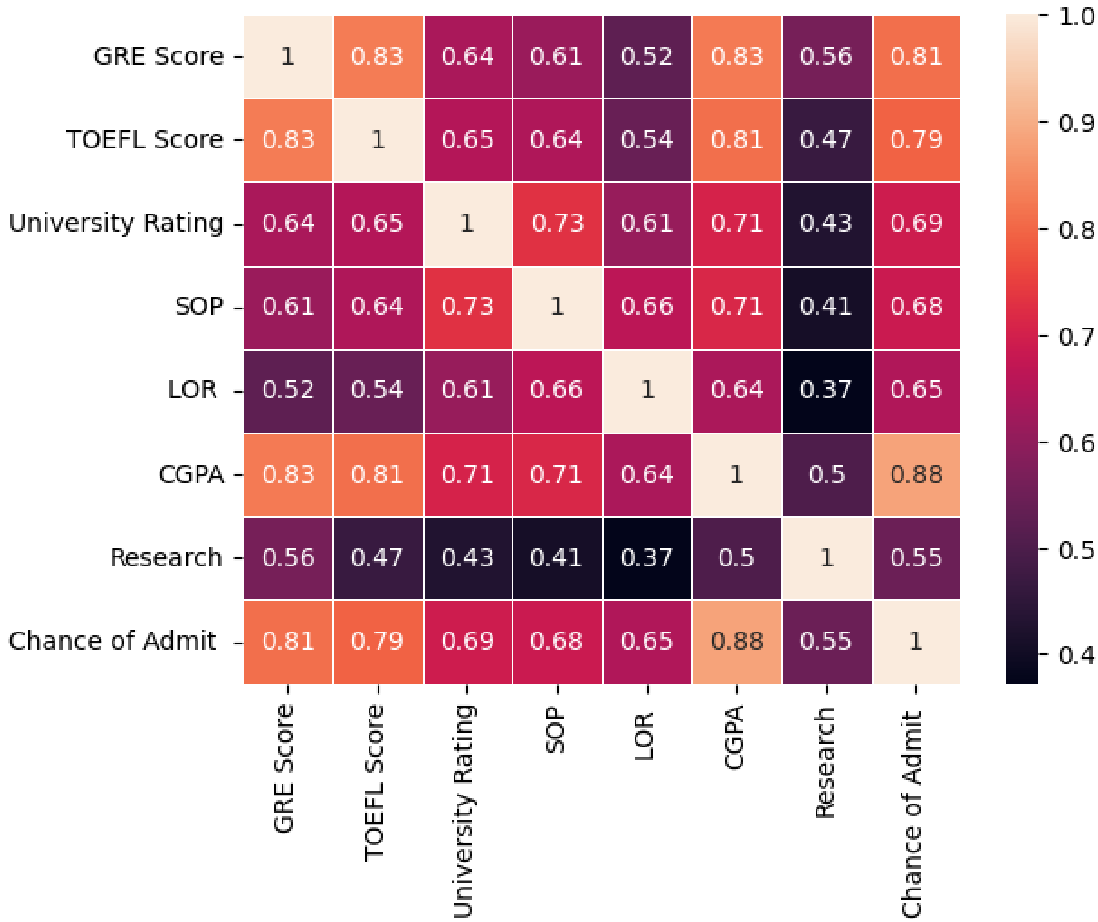
3. people with higher CGPA usually have higher TOEFL scores
4. LORs are not that related with CGPA so it is clear that a persons LOR is not dependent on that persons academic excellence, Having research experience is usually related with a good LOR
5. GRE scores and LORs are also not that related. People with different kinds of LORs have all kinds of GRE scores

6. CGPA and SOP are not that related because Statement of Purpose is related to academic performance

7. people with different SOP have different TOEFL Score. So the quality of SOP is not always related to the applicants English skills.

Correlation between features

[ ]:
```
corr = df.corr()
sns.heatmap(corr, linewidths= .5, annot = True)
plt.show()
```



0.0.2 splitting the dataset with training and testing set

[5]:
```
# importing train_test_split utility from sklearn.model_selection to split the␣
↪data
from sklearn.model_selection import train_test_split
```

[6]:
```
X = df.drop(['Chance of Admit '], axis=1)
y = df['Chance of Admit ']
```

[7]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.20,␣
↪shuffle=True)
```

[8]:
```
X_train.head()
```

[8]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
460	319	105	4	4.0	4.5	8.66	1
473	316	102	2	4.0	3.5	8.15	0
270	306	105	2	2.5	3.0	8.22	1
399	333	117	4	5.0	4.0	9.66	1
47	339	119	5	4.5	4.0	9.70	0

[ ]:
```
y_train.head()
```

[ ]:
```
346    0.47
174    0.87
488    0.76
86     0.72
199    0.72
Name: Chance of Admit , dtype: float64
```

[9]:
```
# getting the columns
X_train_columns=X_train.columns
X_train_columns
```

[9]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research'],
      dtype='object')
```

[ ]:
```
X_train.shape
```

[ ]:
```
(400, 7)
```

[ ]:
```
df.shape
```

[ ]:
```
(500, 8)
```

as we can see that X_train do not have the target column

0.0.3 scalling the dataset

[10]:
```
# importing standardscaler utility from sklearn.preprocessing to scale the data

from sklearn.preprocessing import StandardScaler

# initiate the model
scaler = StandardScaler()

# fitting the data to the model
scaler.fit(X_train)
```

[10]:
```
StandardScaler()
```

[11]:
```python
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

[ ]:
```python
X_train
```

[ ]:
```
array([[-1.41401964, -0.55093665, -0.0891475 , …,  0.55791792,
        -0.78280312,  0.90453403],
       [-0.51394348, -0.55093665, -0.9588792 , …,  0.01493211,
        -0.18138579, -1.1055416 ],
       [-0.42393587, -0.22053986, -0.0891475 , …,  0.01493211,
         0.33650135,  0.90453403],
       …,
       [ 0.83617075, -0.88133343,  0.7805842 , …,  1.64388955,
         0.28638324,  0.90453403],
       [-1.68404248, -1.37692862, -0.9588792 , …, -1.61402533,
        -1.65151703, -1.1055416 ],
       [-1.41401964, -1.54212701, -1.8286109 , …, -0.5280537 ,
        -0.93315745,  0.90453403]])
```

[12]:
```python
X_train=pd.DataFrame(X_train, columns=X_train_columns)
X_train.head()
```

[12]:
```
   GRE Score  TOEFL Score  University Rating       SOP       LOR      CGPA  \
0   0.203517    -0.376464           0.754346  0.623695  1.068485  0.113570
1  -0.063975    -0.875642          -1.005061  0.623695 -0.004022 -0.736017
2  -0.955615    -0.376464          -1.005061 -0.882209 -0.540275 -0.619407
3   1.451813     1.620250           0.754346  1.627632  0.532231  1.779427
4   1.986797     1.953036           1.634049  1.125664  0.532231  1.846061

   Research
0  0.904534
1 -1.105542
2  0.904534
3  0.904534
4 -1.105542
```

[13]:
```python
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)],
          ['Ridge Regression :', Ridge(alpha=1.0)]]

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

```python
for name,model in models:
    model.fit(X_train, y_train.values)
    predictions = model.predict(X_test)
    print(name, (np.sqrt(mean_squared_error(y_test, predictions))))
```

```
Results without removing features with multicollinearity.
Linear Regression : 0.058617244777788166
Lasso Regression : 0.12443271150653722
Ridge Regression : 0.05858592809344395
```

Linear Regression and Ridge Regression are almost identical
This indicates that adding regularization using Ridge Regression did not significantly change the model's performance, suggesting that multicollinearity might not be a severe issue in your data. The metric for Lasso Regression is noticeably worse (0.1148). This might indicate that the Lasso model is over-penalizing certain coefficients, leading to a loss in predictive power. Lasso performs feature selection by shrinking some coefficients to zero, which might be detrimental if all features are important.

Linear Regression using Statsmodel library

[14]:
```python
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:                       0.817
Model:                            OLS   Adj. R-squared:                  0.814
Method:                 Least Squares   F-statistic:                     250.3
Date:                Sun, 10 Nov 2024   Prob (F-statistic):          2.27e-140
Time:                        15:53:21   Log-Likelihood:                 558.90
No. Observations:                 400   AIC:                            -1102.
Df Residuals:                     392   BIC:                            -1070.
Df Model:                           7
Covariance Type:            nonrobust
================================================================================
=====
                        coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-----
const                 0.7250      0.003    239.893      0.000       0.719
0.731
GRE Score             0.0190      0.007      2.913      0.004       0.006
0.032
TOEFL Score           0.0192      0.006      3.320      0.001       0.008
0.031
University Rating     0.0069      0.005      1.368      0.172      -0.003
```

```
0.017
SOP                 -0.0024      0.005     -0.471      0.638      -0.013
0.008
LOR                  0.0152      0.004      3.455      0.001       0.007
0.024
CGPA                 0.0728      0.007     10.333      0.000       0.059
0.087
Research             0.0119      0.004      3.185      0.002       0.005
0.019
==============================================================================
Omnibus:                       92.406   Durbin-Watson:                   1.985
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              217.659
Skew:                          -1.154   Prob(JB):                     5.44e-48
Kurtosis:                       5.780   Cond. No.                         6.06
==============================================================================

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

[15]:
```python
X_train_new=X_train.drop(columns='SOP')
```

[16]:
```python
model1 = sm.OLS(y_train.values, X_train_new).fit()
print(model1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.817
Model:                            OLS   Adj. R-squared:                  0.814
Method:                 Least Squares   F-statistic:                     292.6
Date:                Sun, 10 Nov 2024   Prob (F-statistic):          1.42e-141
Time:                        15:53:43   Log-Likelihood:                 558.78
No. Observations:                 400   AIC:                            -1104.
Df Residuals:                     393   BIC:                            -1076.
Df Model:                           6
Covariance Type:            nonrobust
================================================================================
=====
                        coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-----
const                 0.7250      0.003    240.131      0.000       0.719
0.731
GRE Score             0.0192      0.007      2.952      0.003       0.006
0.032
TOEFL Score           0.0189      0.006      3.297      0.001       0.008
0.030
```

```
University Rating      0.0060      0.005      1.288      0.198      -0.003
0.015
LOR                    0.0146      0.004      3.460      0.001       0.006
0.023
CGPA                   0.0721      0.007     10.479      0.000       0.059
0.086
Research               0.0119      0.004      3.185      0.002       0.005
0.019
==============================================================================
Omnibus:                       94.016   Durbin-Watson:                   1.983
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              223.600
Skew:                          -1.170   Prob(JB):                     2.79e-49
Kurtosis:                       5.818   Cond. No.                         5.55
==============================================================================

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

0.1 Variance Inflation Factor

VIF score of an independent variable represents how well the variable is explained by other independent variables.

the closer the R^2 value to 1, the higher the value of VIF and the higher the multicollinearity with the particular independent variable.

[17]:
```python
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.shape[1])]
  return vif
```

[18]:
```python
calculate_vif(X_train_new,[])
```

[18]:

	features	VIF_Value
0	const	1.000000
1	GRE Score	4.638580
2	TOEFL Score	3.623603
3	University Rating	2.370268
4	LOR	1.962534
5	CGPA	5.187688
6	Research	1.540561

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

[19]:
```
X_test = sm.add_constant(X_test)
```

[22]:
```
X_test = pd.DataFrame(X_test, columns=X_train.columns) # Convert X_test to a␣
↪DataFrame with the same columns as X_train
X_test_del = list(set(X_test.columns).difference(set(X_train.columns)))
```

[23]:
```
print(f'Dropping {X_test_del} from test set')
```

```
Dropping [] from test set
```

[24]:
```
X_test_new=X_test.drop(columns=X_test_del)
```

[26]:
```
# Assuming X_train has 7 columns (including a constant)
# and X_test_new currently has 8 columns

# Get the columns present in the training data
X_train_cols = model1.model.exog_names

# Select only those columns from the test data
X_test_new = X_test_new[X_train_cols]

#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  0.040750477865831025
Root Mean Square Error  0.05834731586121157
```

0.2 Mean of Residuals

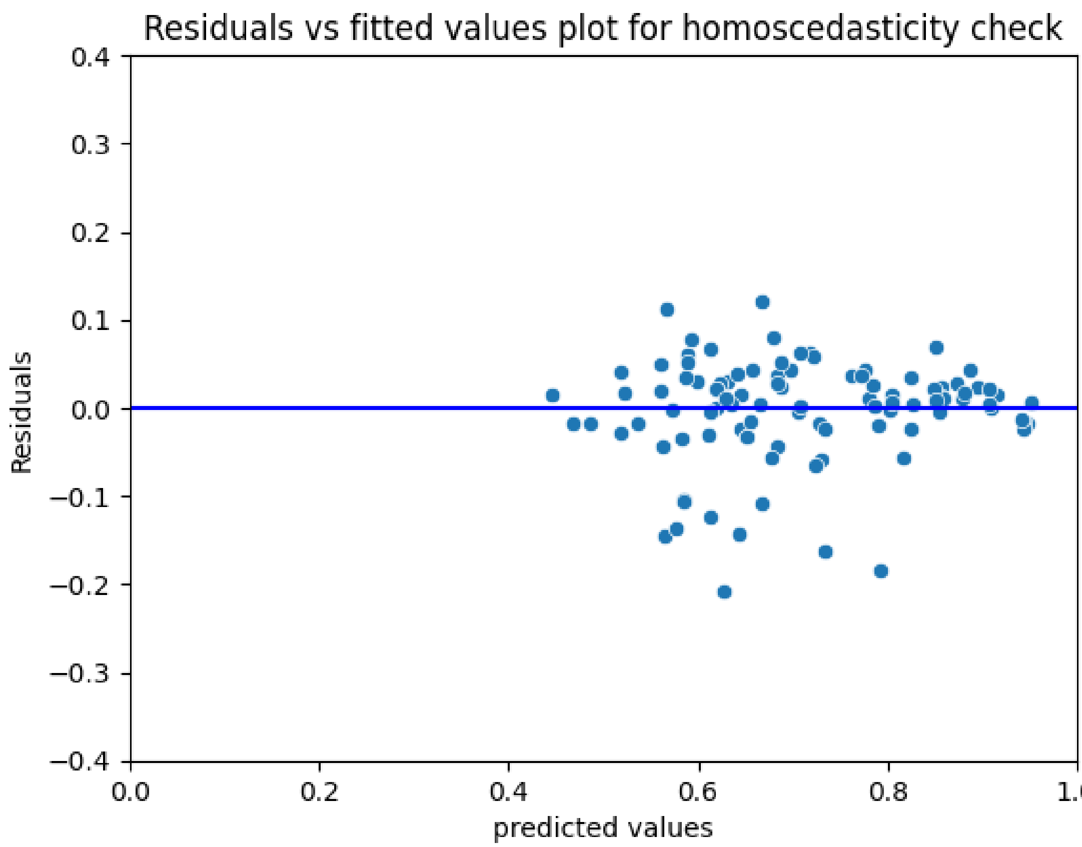
[27]:
```
residuals = y_test.values-pred
mean_residuals = np.mean(residuals)
print("Mean of Residuals {}".format(mean_residuals))
```

```
Mean of Residuals -0.00262733759593862
```

0.3 Test for Homoscedasticity

[28]:
```
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')
```



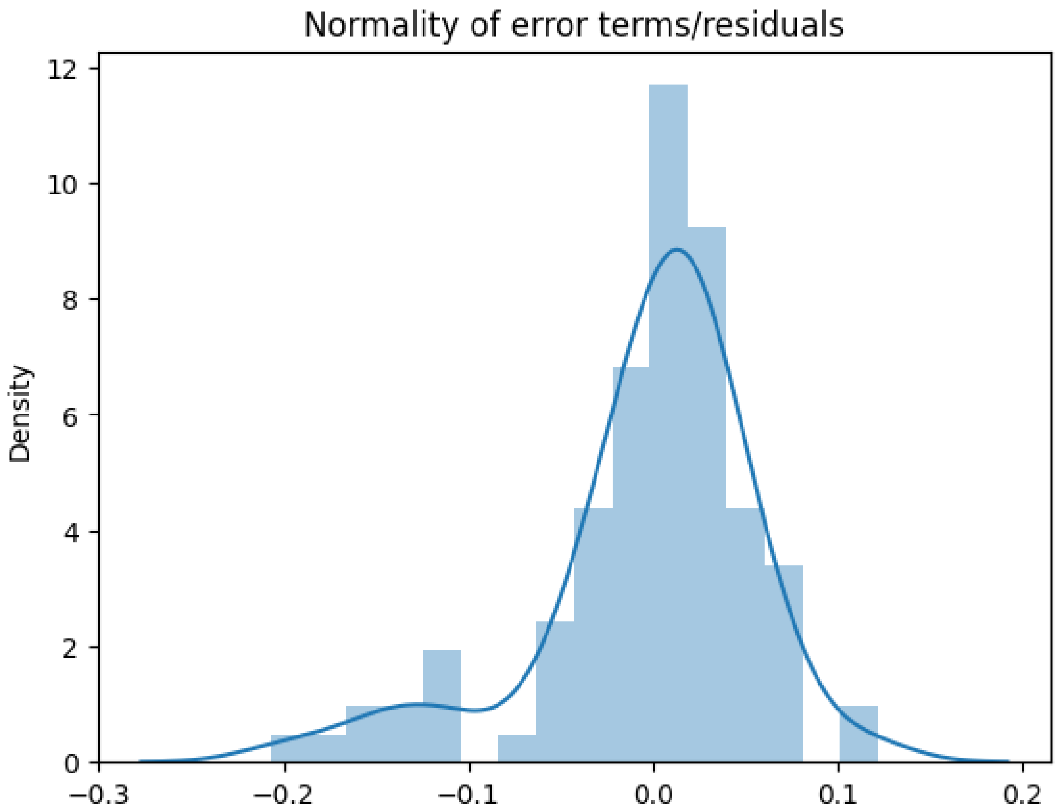
[29]:
```
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
name = ['F statistic', 'p-value']
test = sms.het_goldfeldquandt(residuals, X_test)
lzip(name, test)
```

[29]: [('F statistic', 1.1698095252835439), ('p-value', 0.306797026295602)]

Here null hypothesis is - error terms are homoscedastic and since p-values >0.05, we fail to reject the null hypothesis

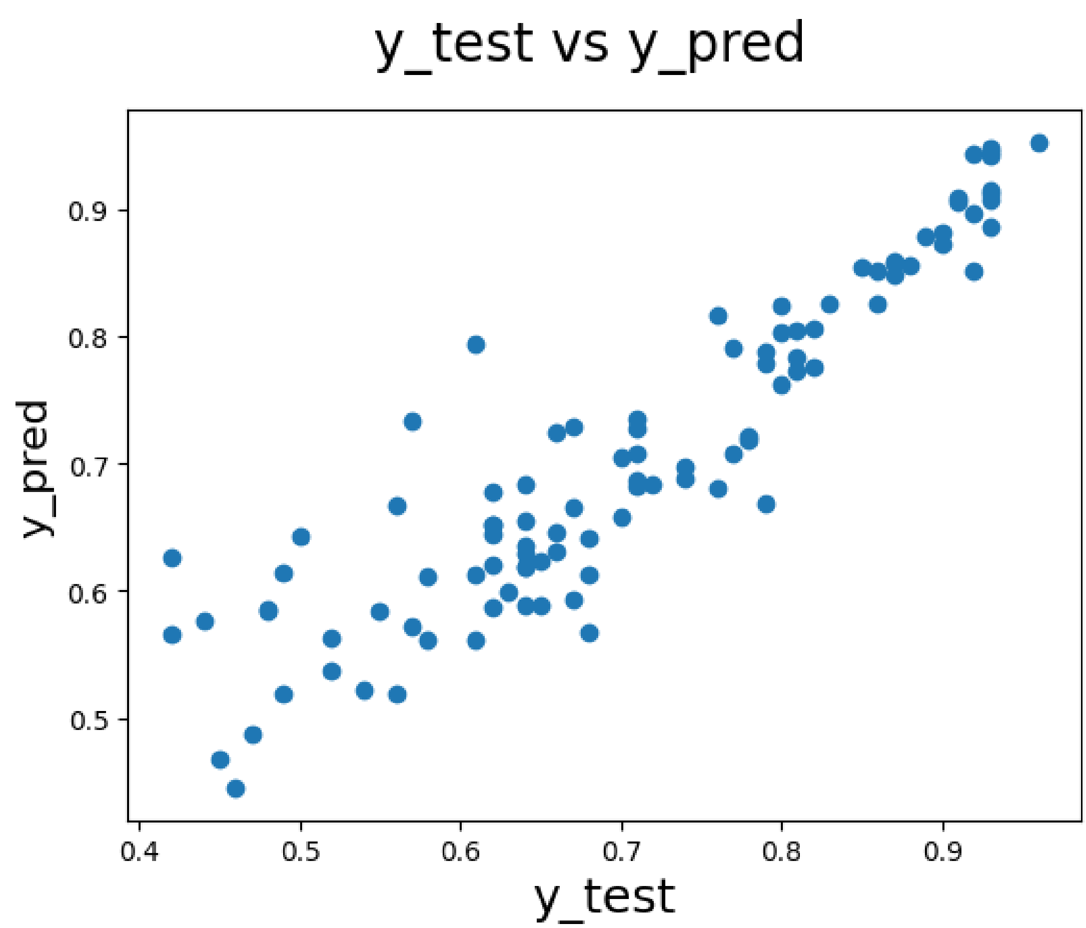
Normality of residuals

[30]:
```
p = sns.distplot(residuals,kde=True)
p = plt.title('Normality of error terms/residuals')
```



[31]:
```
# Plotting y_test and y_pred to understand the spread.
fig = plt.figure()
plt.scatter(y_test.values, pred)
fig.suptitle('y_test vs y_pred', fontsize=20)              # Plot heading
plt.xlabel('y_test', fontsize=18)                          # X-label
plt.ylabel('y_pred', fontsize=16)
```

[31]:
```
Text(0, 0.5, 'y_pred')
```



0.4 INSIGHTS

1.The scatter plot of y_test vs. y_pred shows a strong linear relationship, indicating that the model captures the underlying trend of the data well. The data points are clustered around the line pred = test y pred =y test , which is a positive sign.

2.The Mean Absolute Error (MAE) is 0.04075, and the Root Mean Square Error (RMSE) is 0.05835. Both are relatively low, indicating good predictive performance. The fact that RMSE is slightly higher than MAE suggests the presence of a few larger errors, but they don't seem to dominate the model's overall performance.

3.The mean of residuals is -0.00263, which is very close to zero. This implies that the model's predictions are unbiased on average. The slight negative value indicates a minor tendency of the model to overpredict, but the effect is minimal and may not be practically significant.

[ ]: