In [ ]:

```
import pandas as pd
import warnings
#warnings.filterwarnings("ignore")
df = pd.read_csv('/Users/suraaj/Desktop/Datasets/Admission_Predict_Ver1.1.csv')
df.head()
```

Out[1]:

	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

In [ ]:

```
df.shape
```

Out[2]:

```
(500, 9)
```

Now, let us drop the irrelevant column and check if there are any null values in the dataset

In [ ]:

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

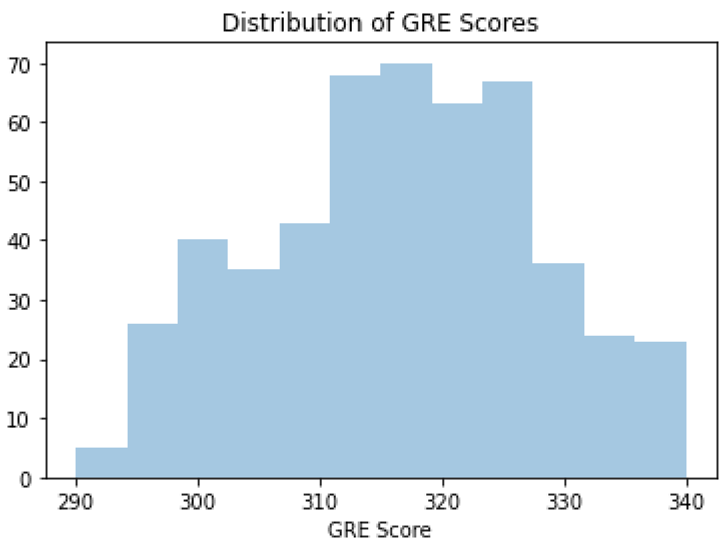
Out[3]:

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

Lets see the distribution of the variables of graduate applicants.

In [ ]:

```
imporx plt
import seaborn as sns


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

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

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

plt.show()
```

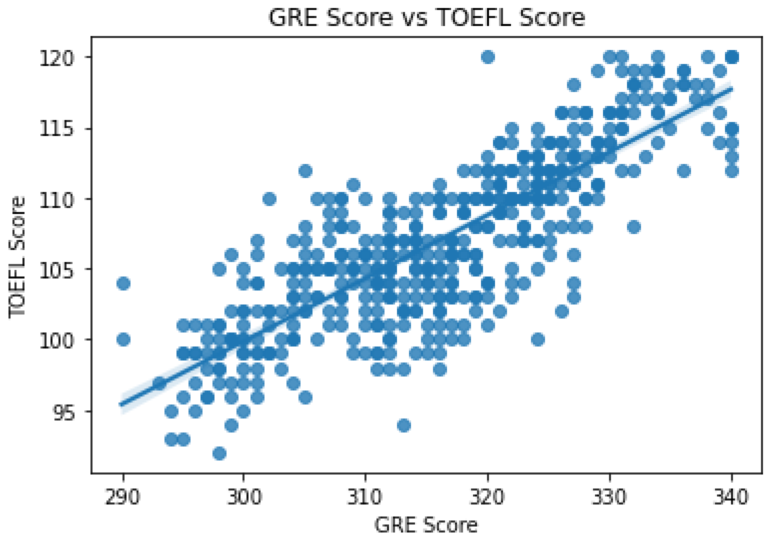


It is clear from the distributions, students with varied merit apply for the university.

Understanding the relation between different factors responsible for graduate admissions

In [ ]:

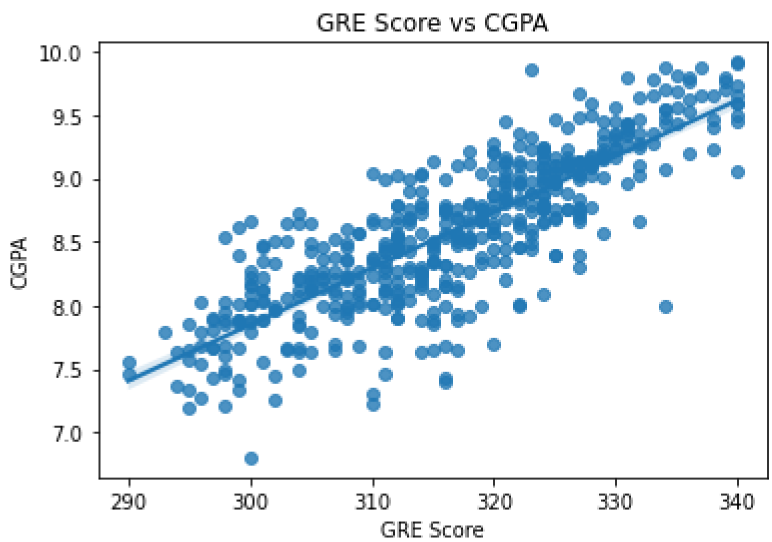
```
fig = sns.regplot(x="GRE Score", y="TOEFL Score", data=df)
plt.title("GRE Score vs TOEFL Score")
plt.show()
```



People with higher GRE Scores also have higher TOEFL Scores which is justified because both TOEFL and GRE have a verbal section which although not similar are relatable

In [ ]:

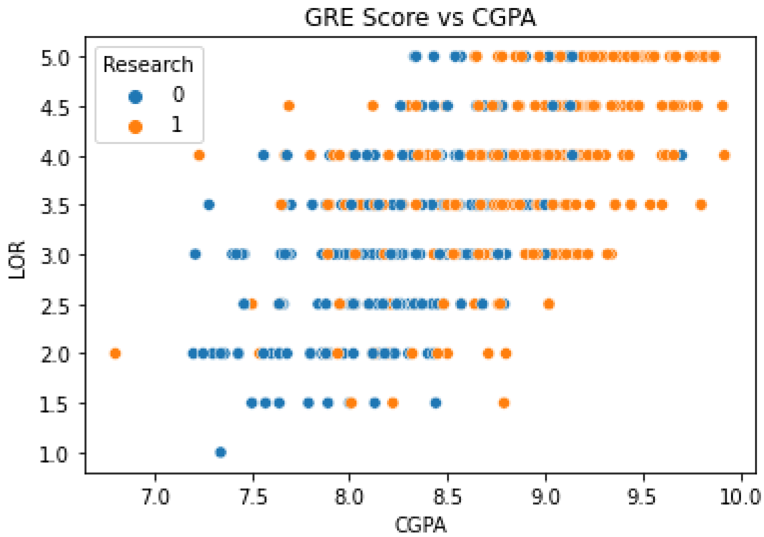
```
fig = sns.regplot(x="GRE Score", y="CGPA", data=df)
plt.title("GRE Score vs CGPA")
plt.show()
```



Although there are exceptions, people with higher CGPA usually have higher GRE scores maybe because they are smart or hard working

In [ ]:

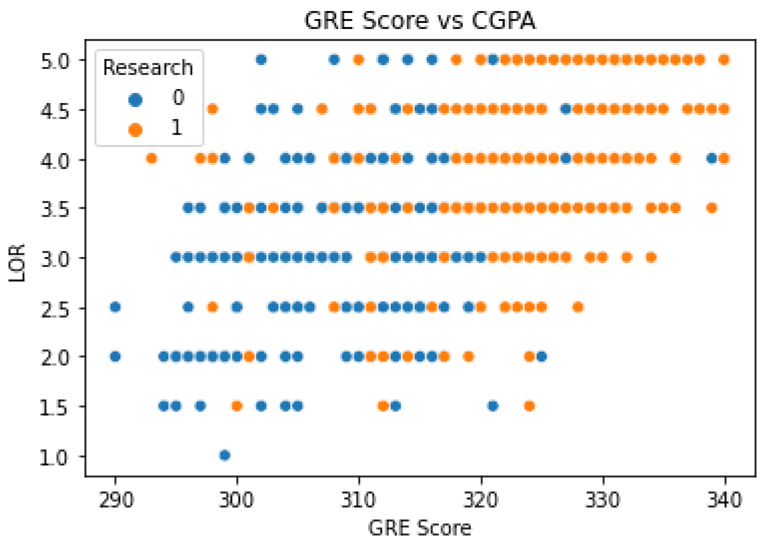
```
fig = sns.scatterplot(x="CGPA", y="LOR ", data=df, hue="Research")
plt.title("GRE Score vs CGPA")
plt.show()
```



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 which might be justified by the fact that supervisors have personal interaction with the students performing research which usually results in good LORs

In [ ]:

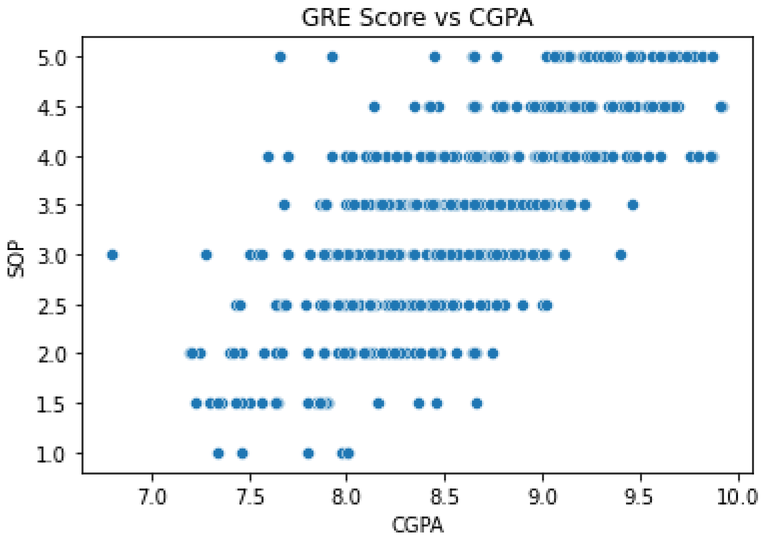
```
fig = sns.scatterplot(x="GRE Score", y="LOR ", data=df, hue="Research")
plt.title("GRE Score vs CGPA")
plt.show()
```



GRE scores and LORs are also not that related. People with different kinds of LORs have all kinds of GRE scores

In [ ]:

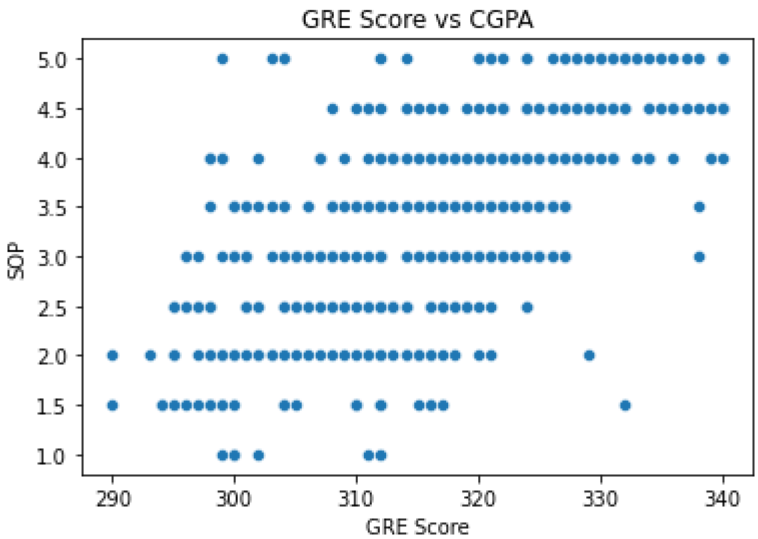
```
fig = sns.scatterplot(x="CGPA", y="SOP", data=df)
plt.title("GRE Score vs CGPA")
plt.show()
```



CGPA and SOP are not that related because Statement of Purpose is related to academic performance, but since people with good CGPA tend to be more hard working so they have good things to say in their SOP which might explain the slight move towards higher CGPA as along with good SOPs

In [ ]:

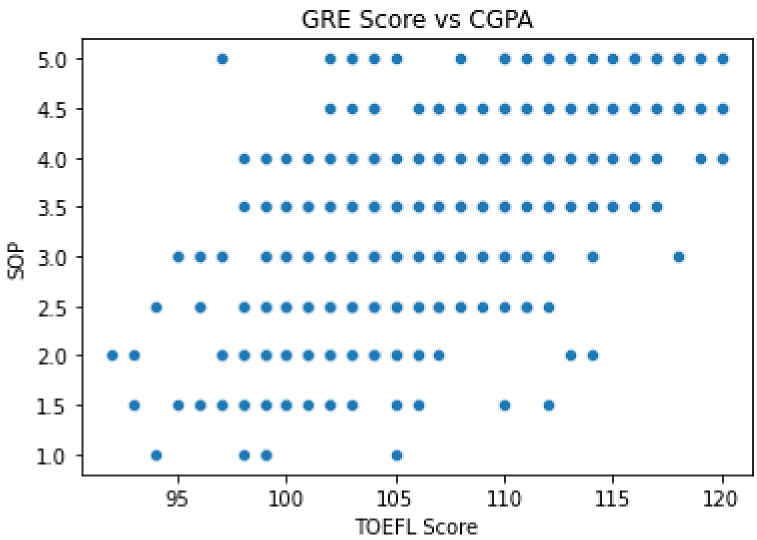
```
fig = sns.scatterplot(x="GRE Score", y="SOP", data=df)
plt.title("GRE Score vs CGPA")
plt.show()
```



Similary, GRE Score and CGPA is only slightly related

In [ ]:

```
fig = sns.scatterplot(x="TOE.. y="SOP", data=df)
plt.title("GRE Score vs CGPA")
plt.show()
```

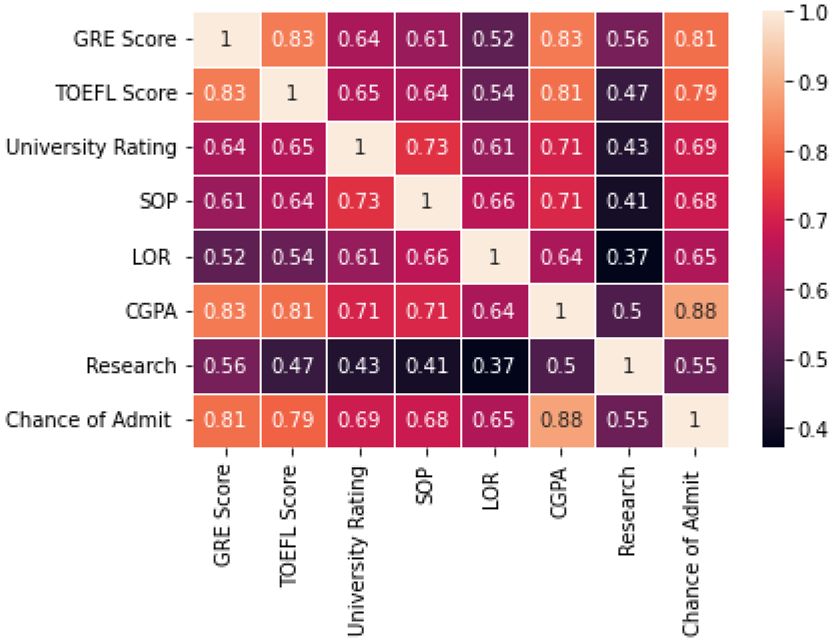


Applicants with different kinds of SOP have different kinds of TOEFL Score. So the quality of SOP is not always related to the applicants English skills.

Correlation among variables

In [ ]:

```
import numpy as np
corr = df.corr()
#fig, ax = plt.subplots(figsize=(8, 8))
#colormap = sns.diverging_palette(220, 10, as_cmap=True)
#dropSelf = np.zeros_like(corr)
#dropSelf[np.triu_indices_from(dropSelf)] = True
#colormap = sns.diverging_palette(220, 10, as_cmap=True)
sns.heatmap(corr, linewidths=.5, annot=True)
plt.show()
```



Lets split the dataset with training and testing set and prepare the inputs and outputs

In [ ]:

```
from sklearn.model_selection importS

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

In [ ]:

```

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.20, shuffle=Tr
```

In [ ]:

```
X_train
```

Out[15]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
479	325	110	4	4.5	4.0	8.96	1
493	300	95	2	3.0	1.5	8.22	1
184	316	106	2	2.5	4.0	8.32	0
397	330	116	4	5.0	4.5	9.45	1
323	305	102	2	2.0	2.5	8.18	0
...	...	...	...	...	...	...	...
173	323	113	4	4.0	4.5	9.23	1
470	320	110	5	4.0	4.0	9.27	1
132	309	105	5	3.5	3.5	8.56	0
2	316	104	3	3.0	3.5	8.00	1
105	316	110	3	4.0	4.5	8.78	1

400 rows × 7 columns

In [ ]:

```
y_train
```

Out[16]:

```
479    0.79
493    0.62
184    0.72
397    0.91
323    0.62
       ... 
173    0.89
470    0.87
132    0.71
2      0.72
105    0.69
Name: Chance of Admit , Length: 400, dtype: float64
```

In [ ]:

```
#Standardization
from sklearn.preprocessing import StandardScaler
X_train_columns=X_train.columns
std=StandardScaler()
X_train_std=std.fit_transform(X_train)
```

In [ ]:

```
X_train_std
```

Out[18]:

```
array([[ 0.72625624,  0.42425133,  0.7191839 , ...,  0.54664996,
         0.5976762 ,  0.86413245],
       [-1.46126256, -2.03043332, -0.99827019, ..., -2.17300157,
        -0.62064316,  0.86413245],
       [-0.06125053, -0.23033124, -0.99827019, ...,  0.54664996,
        -0.45600541, -1.15723001],
       ...,
       [-0.67375579, -0.39397689,  1.57791095, ...,  0.00271965,
        -0.06087481, -1.15723001],
       [-0.06125053, -0.55762253, -0.13954315, ...,  0.00271965,
        -0.98284622,  0.86413245],
       [-0.06125053,  0.42425133, -0.13954315, ...,  1.09058026,
         0.30132825,  0.86413245]])
```

In [ ]:

```
X_train=pd.DataFrame(X_train_std, columns=X_train_columns)
```

In [ ]:

```
X_train
```

Out[20]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	0.726256	0.424251	0.719184	1.125357	0.546650	0.597676	0.864132
1	-1.461263	-2.030433	-0.998270	-0.423299	-2.173002	-0.620643	0.864132
2	-0.061251	-0.230331	-0.998270	-0.939518	0.546650	-0.456005	-1.157230
3	1.163760	1.406125	0.719184	1.641576	1.090580	1.404401	0.864132
4	-1.023759	-0.884914	-0.998270	-1.455737	-1.085141	-0.686498	-1.157230
...	...	...	...	...	...	...	...
395	0.551255	0.915188	0.719184	0.609138	1.090580	1.042198	0.864132
396	0.288752	0.424251	1.577911	0.609138	0.546650	1.108053	0.864132
397	-0.673756	-0.393977	1.577911	0.092919	0.002720	-0.060875	-1.157230
398	-0.061251	-0.557623	-0.139543	-0.423299	0.002720	-0.982846	0.864132
399	-0.061251	0.424251	-0.139543	0.609138	1.090580	0.301328	0.864132

400 rows × 7 columns

Lets use a bunch of different algorithms to see which model performs better

In [ ]:

```
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 values
          ['Ridge Regression :', Ridge(alpha=1.0)] #try with different alpha values
          ]

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 : 0.05298305536300251
Lasso Regression : 0.10605223378225762
Ridge Regression : 0.052917718455970944
```

Linear Regression using Statsmodel library

- Adjusted. R-squared reflects the fit of the model. R-squared values range from 0 to 1, where a higher value generally indicates a better fit, assuming certain conditions are met.
- const coefficient is your Y-intercept. It means that if both the Interest_Rate and Unemployment_Rate coefficients are zero, then the expected output (i.e., the Y) would be equal to the const coefficient.
- Interest_Rate coefficient represents the change in the output Y due to a change of one unit in the interest rate (everything else held constant)
- Unemployment_Rate coefficient represents the change in the output Y due to a change of one unit in the unemployment rate (everything else held constant)
- std err reflects the level of accuracy of the coefficients. The lower it is, the higher is the level of accuracy
- P >|t| is your p-value. A p-value of less than 0.05 is considered to be statistically significant
- Confidence Interval represents the range in which our coefficients are likely to fall (with a likelihood of 95%)

In [ ]:

```
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:
0.819
Model:                            OLS   Adj. R-squared:
0.816
Method:                 Least Squares   F-statistic:
253.1
Date:                Sun, 03 Jul 2022   Prob (F-statistic):
3.78e-141
Time:                        05:44:36   Log-Likelihood:
549.48
No. Observations:                 400   AIC:
-1083.
Df Residuals:                     392   BIC:
-1051.
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
===============
                        coef    std err          t      P>|t|      [0.
025      0.975]
------------------------------------------------------------------------------
---------------
const                 0.7241      0.003    234.011      0.000       0.
718       0.730
GRE Score             0.0149      0.007      2.234      0.026       0.
002       0.028
TOEFL Score           0.0178      0.006      2.899      0.004       0.
006       0.030
University Rating     0.0098      0.005      1.944      0.053      -0.
000       0.020
SOP                   0.0003      0.005      0.051      0.959      -0.
010       0.010
LOR                   0.0150      0.004      3.350      0.001       0.
006       0.024
CGPA                  0.0753      0.007     10.765      0.000       0.
062       0.089
Research              0.0154      0.004      4.108      0.000       0.
008       0.023
==============================================================================
========
Omnibus:                       94.925   Durbin-Watson:
2.046
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
199.062
Skew:                          -1.242   Prob(JB):
5.95e-44
Kurtosis:                       5.402   Cond. No.
5.91
==============================================================================
========

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

In [ ]:

```
X_train_new=X_train.drop(columns='SOP')
```

In [ ]:

```
model1 = sm.OLS(y_train.values, X_train_new).fit()
print(model1.summary())
```

```
                            OLS Regression Results
==============================================================================
========
Dep. Variable:                      y   R-squared:
0.819
Model:                            OLS   Adj. R-squared:
0.816
Method:                 Least Squares   F-statistic:
296.1
Date:                Sun, 03 Jul 2022   Prob (F-statistic):
2.11e-142
Time:                        05:44:36   Log-Likelihood:
549.48
No. Observations:                 400   AIC:
-1085.
Df Residuals:                     393   BIC:
-1057.
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
===============
                        coef    std err          t      P>|t|      [0.
025      0.975]
------------------------------------------------------------------------------
---------------
const                 0.7241      0.003    234.309      0.000       0.
718       0.730
GRE Score             0.0149      0.007      2.240      0.026       0.
002       0.028
TOEFL Score           0.0178      0.006      2.918      0.004       0.
006       0.030
University Rating     0.0099      0.005      2.130      0.034       0.
001       0.019
LOR                   0.0151      0.004      3.498      0.001       0.
007       0.024
CGPA                  0.0754      0.007     11.052      0.000       0.
062       0.089
Research              0.0154      0.004      4.115      0.000       0.
008       0.023
==============================================================================
========
Omnibus:                       94.784   Durbin-Watson:
2.046
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
198.587
Skew:                          -1.241   Prob(JB):
7.54e-44
Kurtosis:                       5.399   Cond. No.
5.40
==============================================================================
========

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

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 the higher the multicollinearity with the particular independent variable.

In [ ]:

```
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(datas
  return vif
```

In [ ]:

```
calculate_vif(X_train_new,[])
```

Out[28]:

	features	VIF_Value
0	const	1.000000
1	GRE Score	4.615374
2	TOEFL Score	3.902858
3	University Rating	2.245892
4	LOR	1.953575
5	CGPA	4.871981
6	Research	1.471018

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

In [ ]:

```
X_test_std= std.transform(X_test)
```

In [ ]:

```
X_test=pd.DataFrame(X_test_std, columns=X_train_columns) # col name same as train da
```

In [ ]:

```
X_test = sm.add_constant(X_test)
```

In [ ]:

```
X_test_del=list(set(X_test.columns).difference(set(X_train_new.columns)))
```

In [ ]:

```
print(f'Dropping {X_test_del} from test set')
```

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

In [ ]:

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

In [ ]:

```
#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  0.03889667283539957
Root Mean Square Error  0.05299573386462354
```

Mean of Residuals

In [ ]:

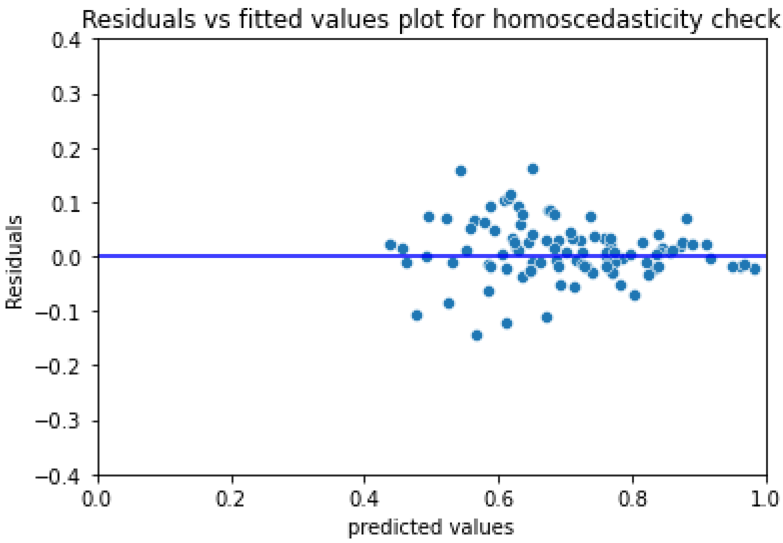
```
residuals = y_test.values-pred
mean_residuals = np.mean(residuals)
print("Mean of Residuals {}".format(mean_residuals))
```

```
Mean of Residuals 0.010929365649686953
```

Test for Homoscedasticity

In [ ]:

```
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],[0,0],color='blue')
p = plt.title('Residuals vs fitted values plot for homoscedasticity check")
```



In [ ]:

```
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[42]:

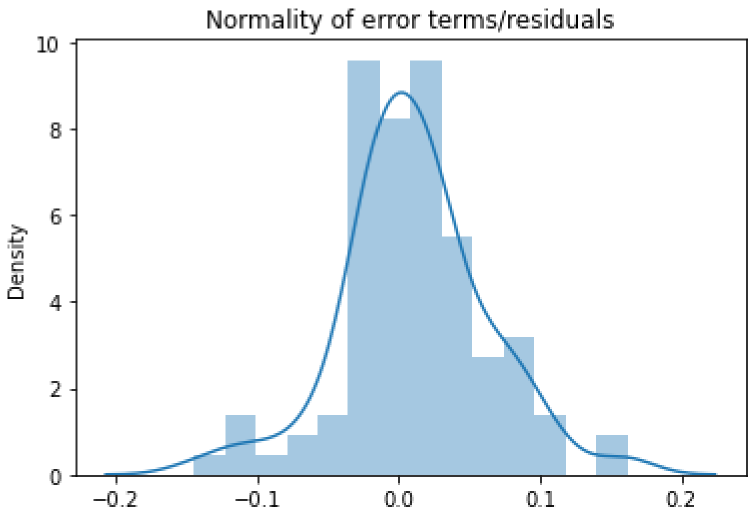
```
[('F statistic', 0.800831247191971), ('p-value', 0.7626036849906835)]
```

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

Normality of residuals

In [ ]:

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

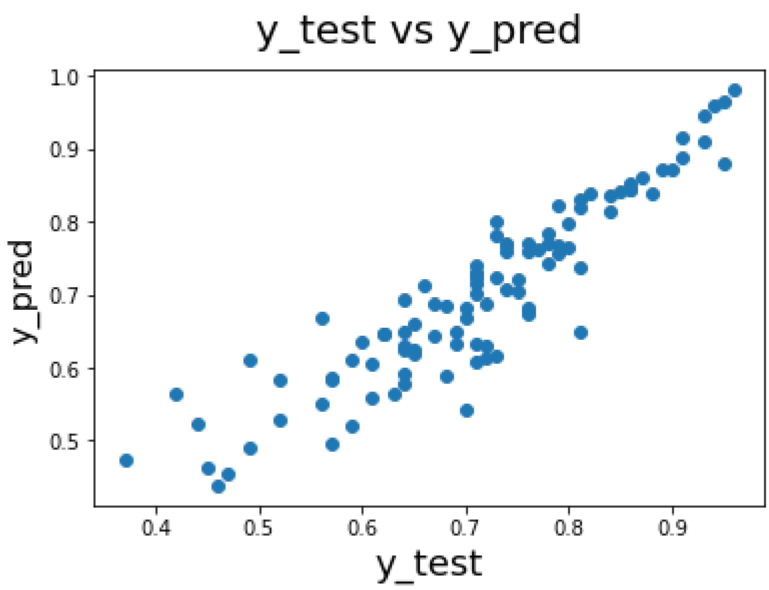


In [ ]:

```
# 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)                          # Y-label
```

Out[52]:

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



Is this good? we are seeing a pattern?

Bias-Variance Tradeoff

- Bias is as a result of over simplified model assumptions
- Variance occurs when the assumptions are too complex

- The more preferred model is one with low bias and low varinace.
- Dimensionality reduction and feature selection can decrease variance by simplifying models.
- Similarly, a larger training set tends to decrease variance.
- For reducing Bias: Change the model, Ensure the date is truly representative(Ensure that the training data is diverse and represents all possible groups or outcomes.), Parameter tuning.
- The bias–variance decomposition forms the conceptual basis for regression regularization methods such as Lasso and ridge regression.
- Regularization methods introduce bias into the regression solution that can reduce variance considerably relative to the ordinary least squares (OLS) solution.
- Although the OLS solution provides non-biased regression estimates, the lower variance solutions produced by regularization techniques provide superior MSE performance.
- Linear and Generalized linear models can be regularized to decrease their variance at the cost of increasing their bias.