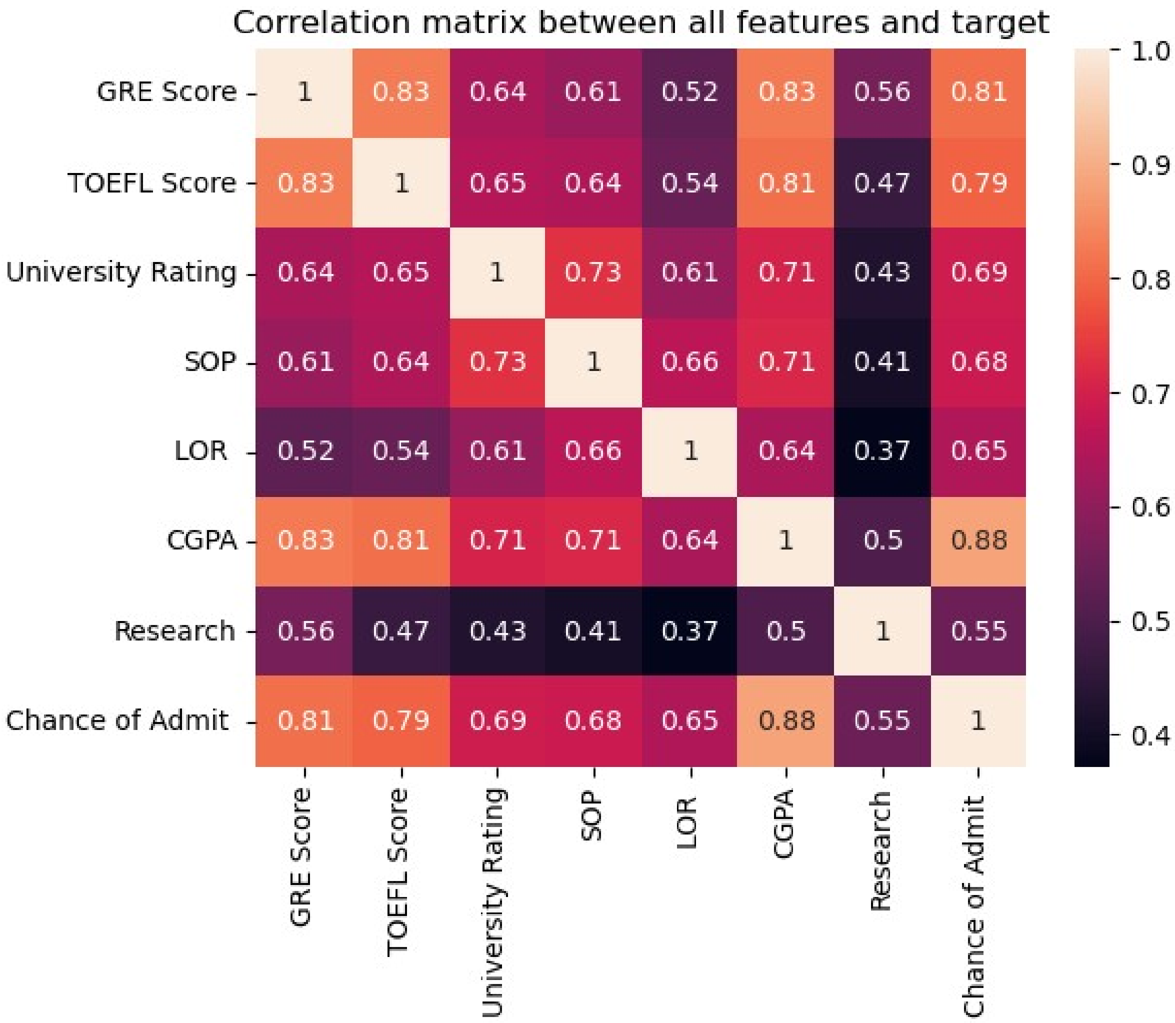
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data=pd.read_csv("/Users/sridhar/Downloads/jamboree_case_scaler.csv")

data

     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
..          ...        ...          ...                ...  ...   ...
...
495         496        332          108                  5  4.5   4.0
9.02
496         497        337          117                  5  5.0   5.0
9.87
497         498        330          120                  5  4.5   5.0
9.56
498         499        312          103                  4  4.0   5.0
8.43
499         500        327          113                  4  4.5   4.5
9.04

     Research  Chance of Admit
0           1              0.92
1           1              0.76
2           1              0.72
3           1              0.80
4           0              0.65
..        ...               ...
495         1              0.87
496         1              0.96
497         1              0.93
498         0              0.73
499         0              0.84

[500 rows x 9 columns]

data.shape
```

```
(500, 9)

data.drop('Serial No.', axis=1, inplace=True)

data.isnull().sum()

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

data.dtypes

GRE Score              int64
TOEFL Score            int64
University Rating      int64
SOP                  float64
LOR                  float64
CGPA                 float64
Research               int64
Chance of Admit      float64
dtype: object

#data['Research']=data['Research'].astype('object')



sns.heatmap(data.corr(), annot=True)
plt.title('Correlation matrix between all features and target')
plt.show()
```




```
#So the feature importances of the chance of admission in descending order are:
#1. CGPA
#2. GRE score
#3. TOEFL Score

data.columns

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

data.rename(columns={'LOR ':'LOR','Chance of Admit ':'Chance of admit'}, inplace=True)

data.columns
```

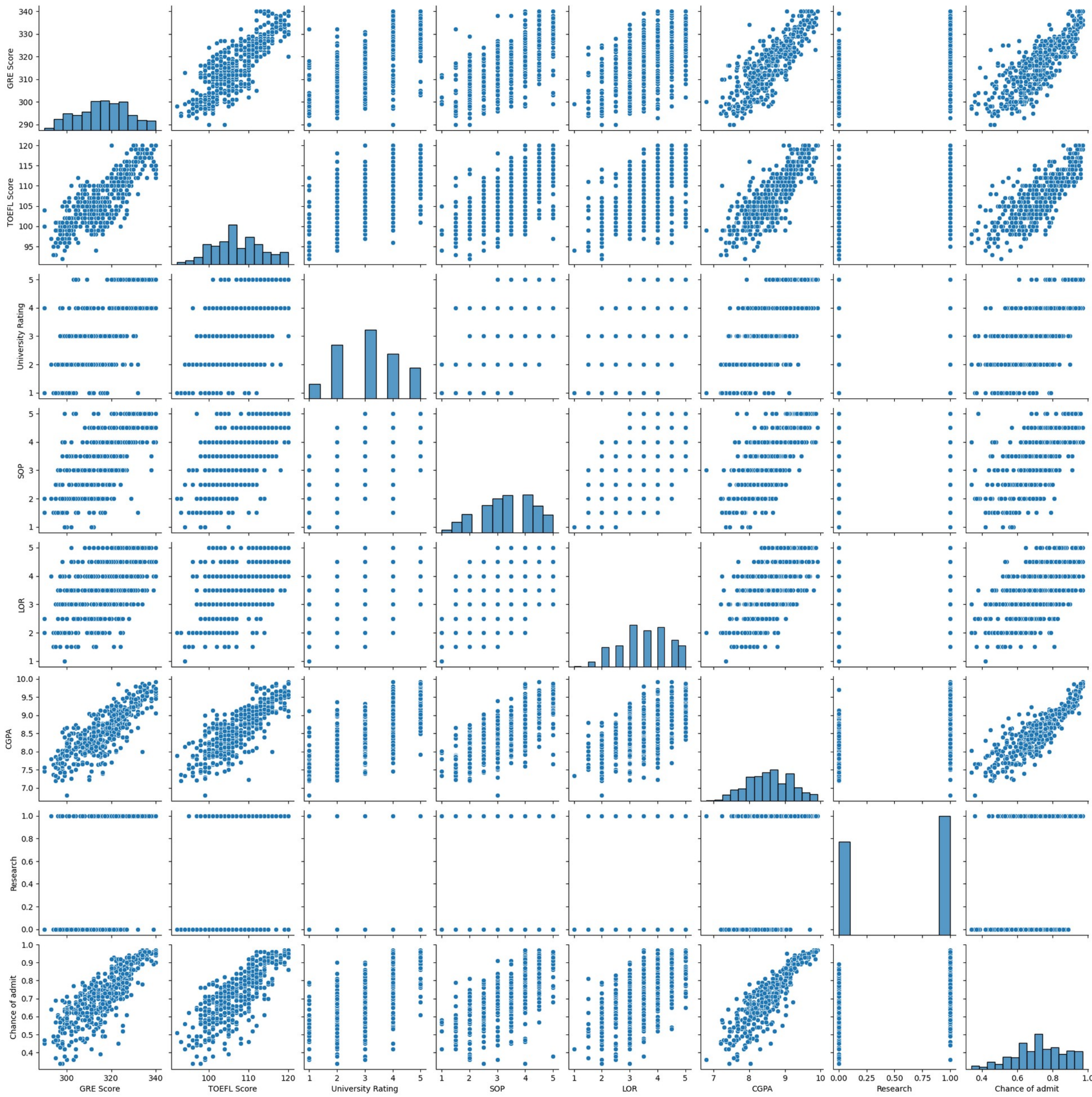
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR',
'CGPA',
       'Research', 'Chance of admit'],
      dtype='object')
```

```
data.describe()
```

```
        GRE Score  TOEFL Score  University Rating         SOP
LOR  \
count  500.000000   500.000000         500.000000  500.000000
500.00000
mean   316.472000   107.192000           3.114000    3.374000
3.48400
std     11.295148     6.081868           1.143512    0.991004
0.92545
min    290.000000    92.000000           1.000000    1.000000
1.00000
25%    308.000000   103.000000           2.000000    2.500000
3.00000
50%    317.000000   107.000000           3.000000    3.500000
3.50000
75%    325.000000   112.000000           4.000000    4.000000
4.00000
max    340.000000   120.000000           5.000000    5.000000
5.00000

             CGPA    Research  Chance of admit
count  500.000000  500.000000        500.00000
mean     8.576440    0.560000          0.72174
std      0.604813    0.496884          0.14114
min      6.800000    0.000000          0.34000
25%      8.127500    0.000000          0.63000
50%      8.560000    1.000000          0.72000
75%      9.040000    1.000000          0.82000
max      9.920000    1.000000          0.97000
```

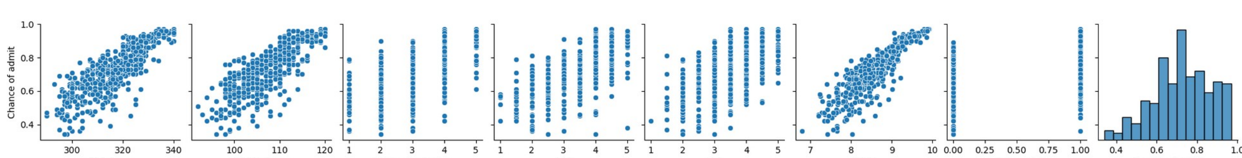
```
sns.pairplot(data)
```

```
<seaborn.axisgrid.PairGrid at 0x12ba030e0>
```

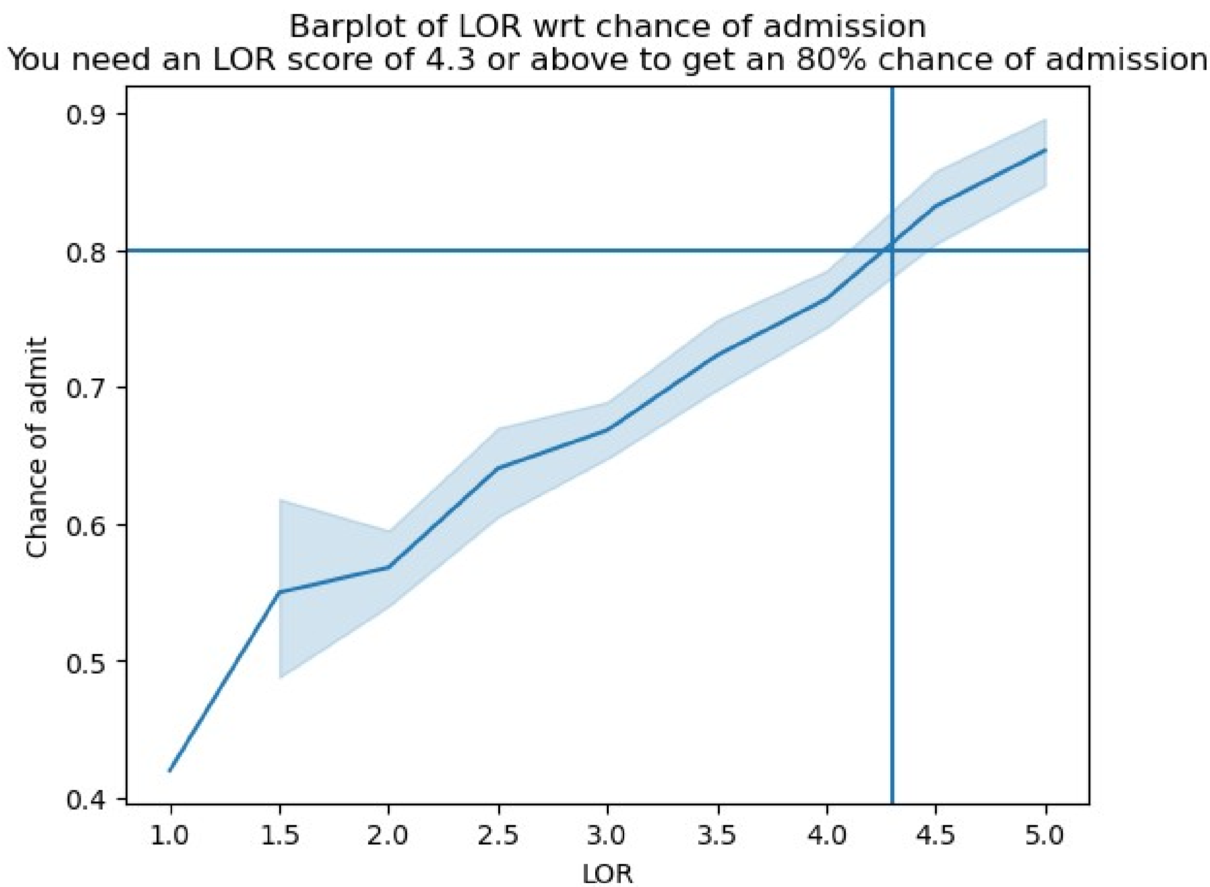



```python
#Checking for the assumptions of linear regression
#1. Linearity assumption
sns.pairplot(data, y_vars='Chance of admit', kind='scatter')
```

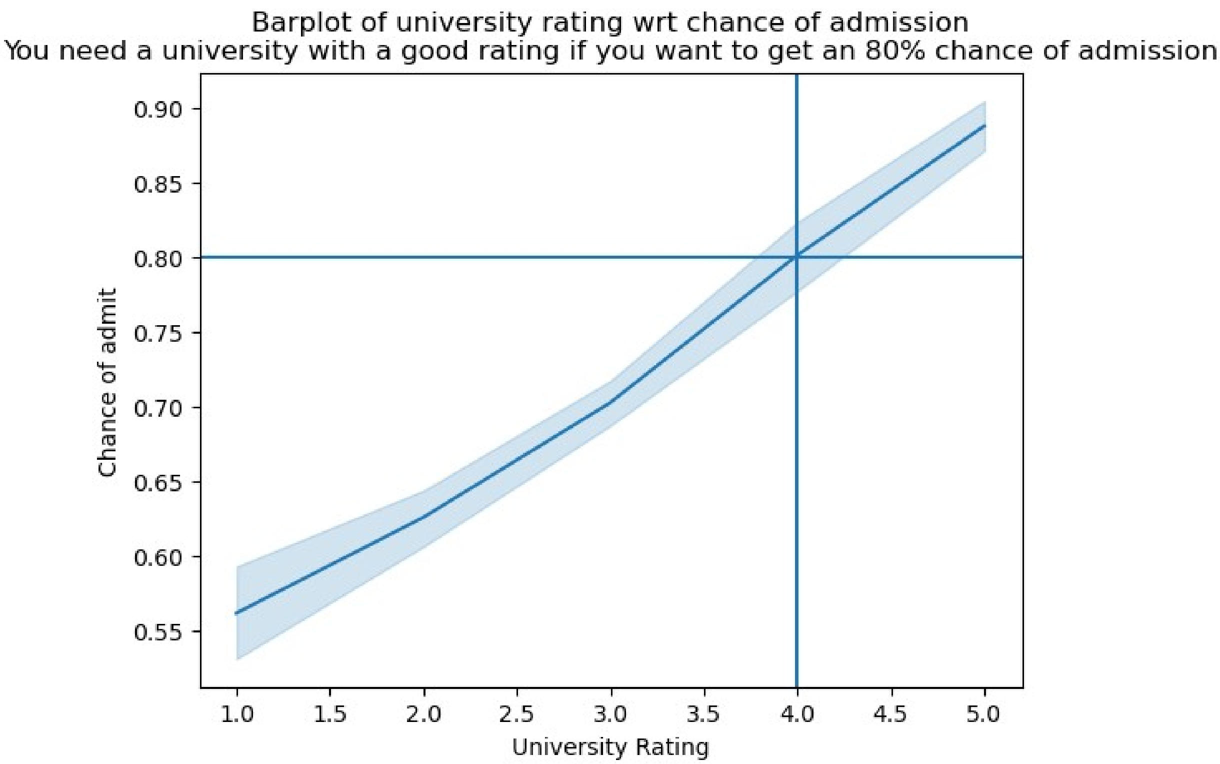
```
<seaborn.axisgrid.PairGrid at 0x12d14ab40>
```



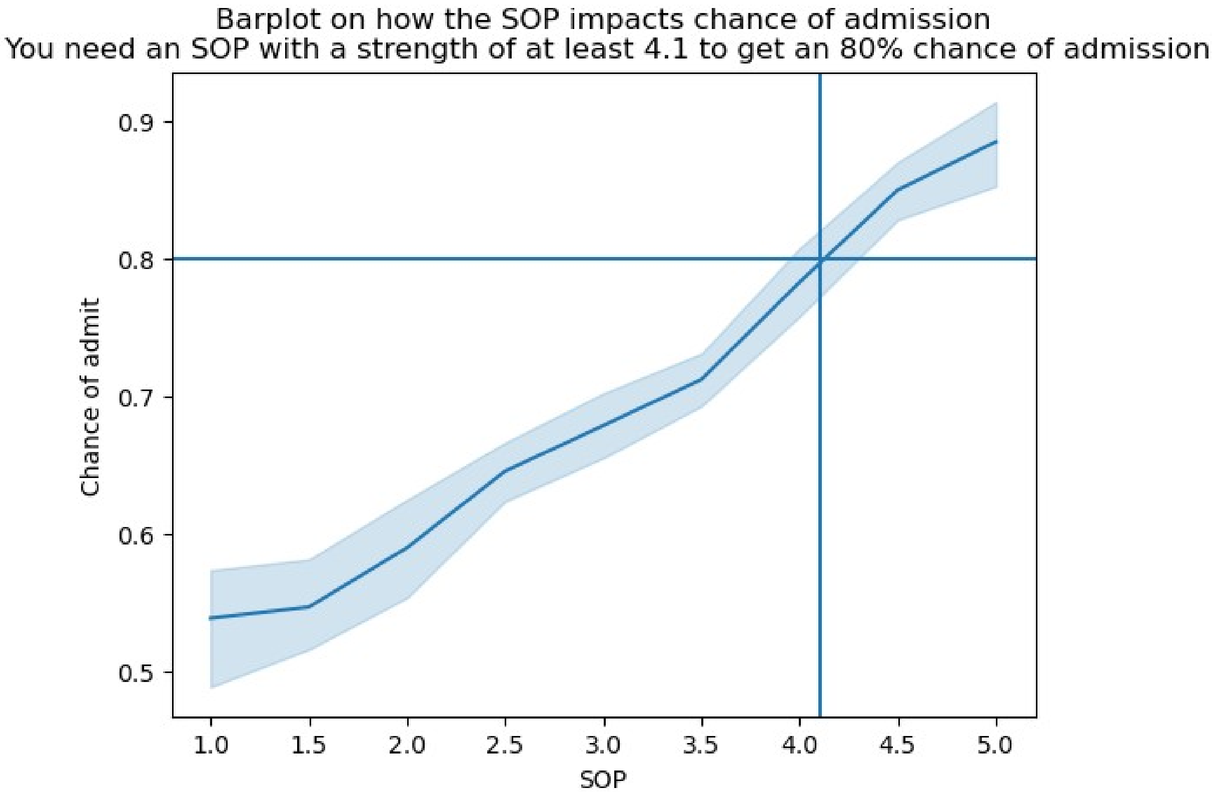
```
#almost every input appears to vary linearly with the output

sns.lineplot(data=data, x='LOR',y='Chance of admit')
plt.title('Barplot of LOR wrt chance of admission\nYou need an LOR
score of 4.3 or above to get an 80% chance of admission')
plt.axhline(.8)
plt.axvline(4.3)
plt.show()
```



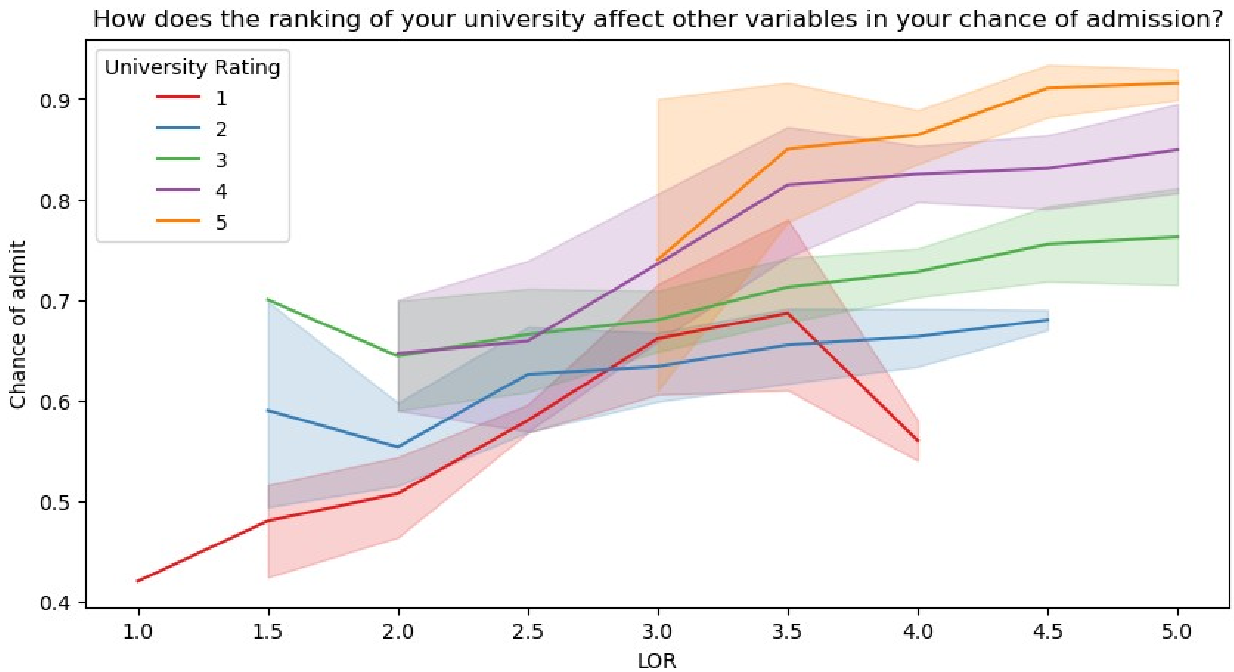

```
sns.lineplot(data=data, x='University Rating', y='Chance of admit')
plt.title('Barplot of university rating wrt chance of admission\nYou
need a university with a good rating if you want to get an 80% chance
of admission')
plt.axhline(.8)
plt.axvline(4)
plt.show()
```



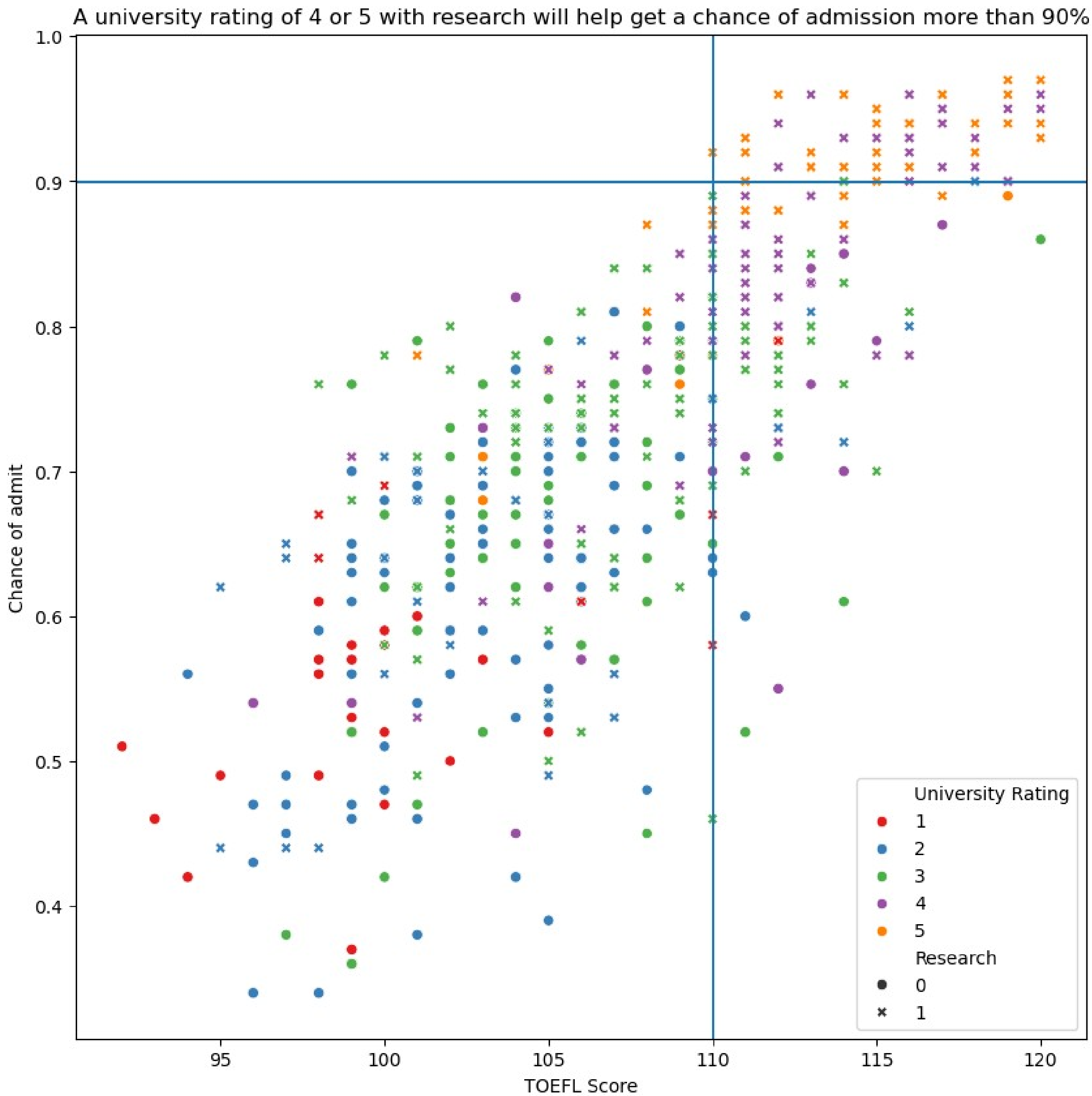

```
sns.lineplot(data=data, x='SOP', y='Chance of admit')
plt.title('Barplot on how the SOP impacts chance of admission\n You
need an SOP with a strength of at least 4.1 to get an 80% chance of
admission')
plt.axhline(.8)
plt.axvline(4.1)
plt.show()
```



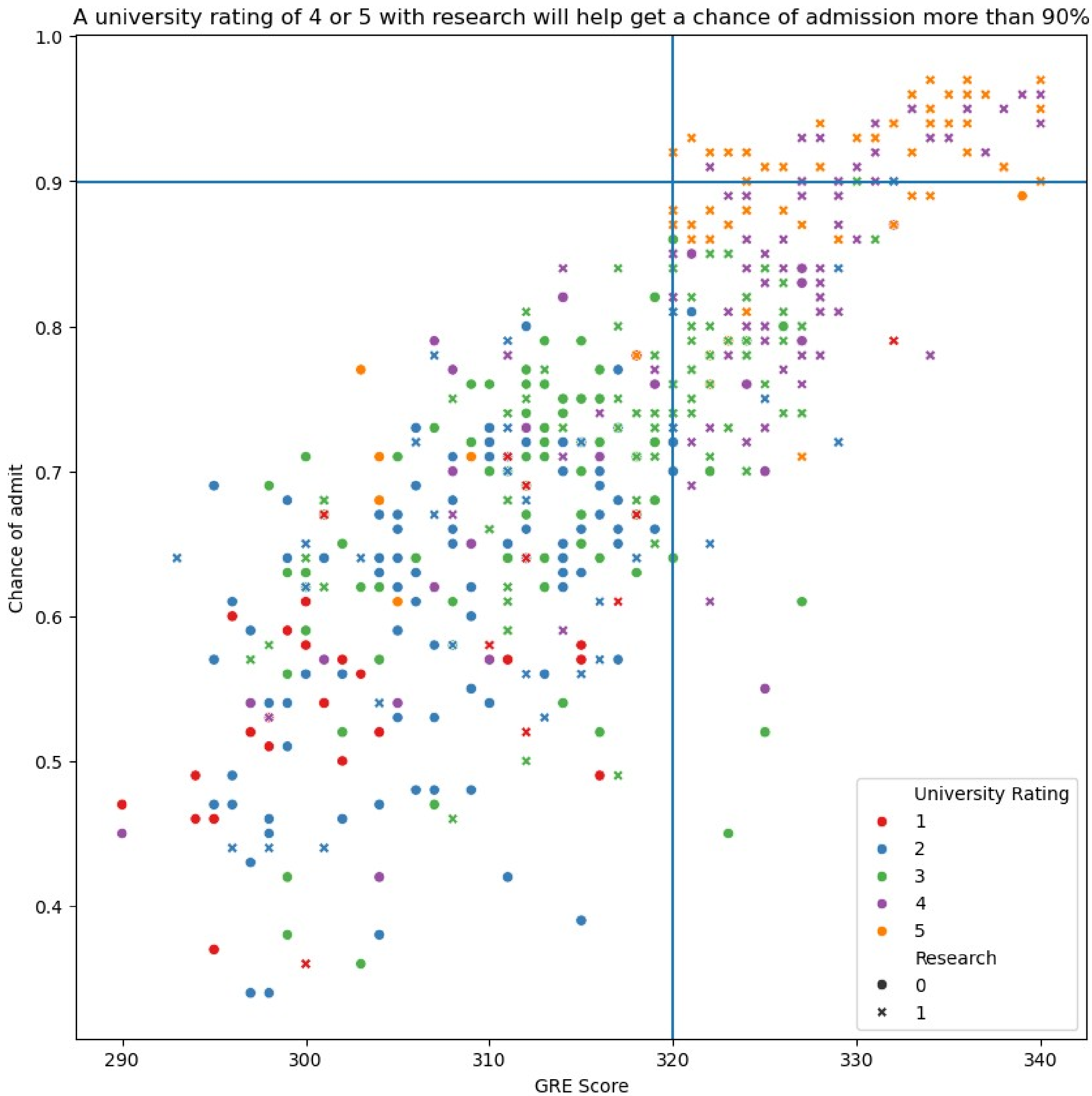

```
#assumption 2: Multicollinearity assumption

plt.figure(figsize=(10,5))
sns.lineplot(data=data, x='LOR', y='Chance of admit', hue='University
Rating', palette='Set1')
plt.title('How does the ranking of your university affect other
variables in your chance of admission?')
plt.show()
```




```
plt.figure(figsize=(10,10))
sns.scatterplot(data=data, x='TOEFL Score', y='Chance of admit',
hue='University Rating', style='Research', palette='Set1')
plt.title('A university rating of 4 or 5 with research will help get a
chance of admission more than 90%')
plt.axhline(.9)
plt.axvline(110)
plt.show()
```

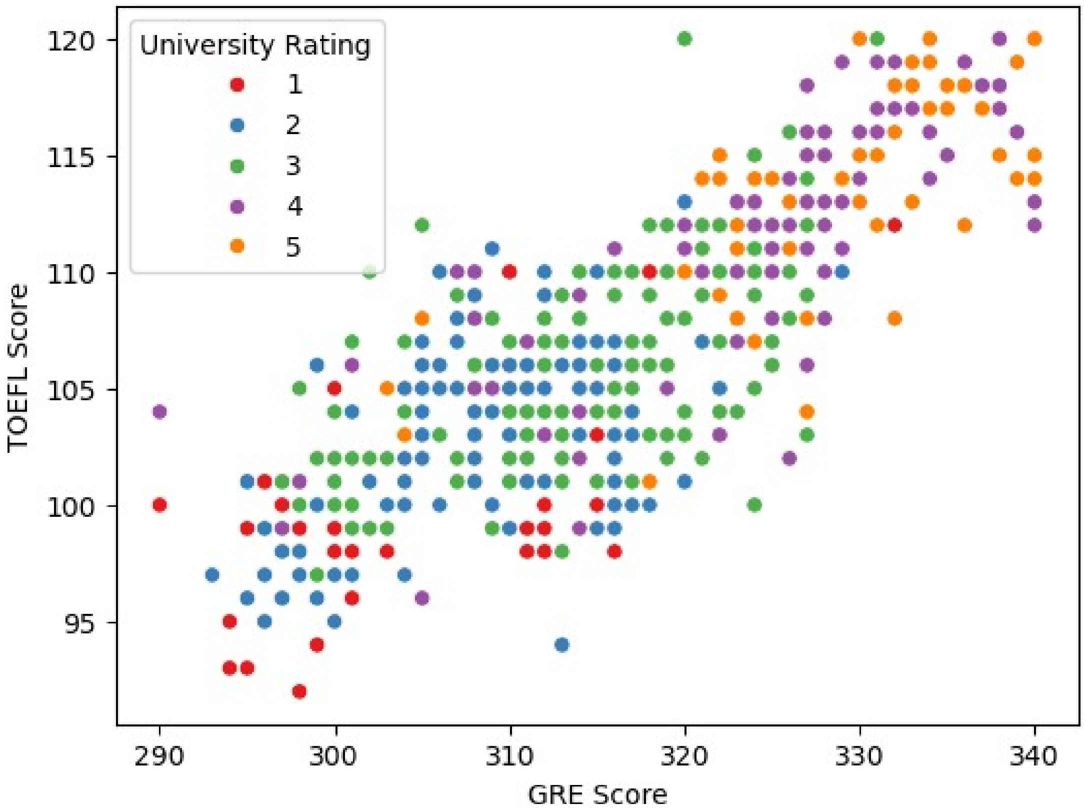



```
plt.figure(figsize=(10,10))
sns.scatterplot(data=data, x='GRE Score', y='Chance of admit',
hue='University Rating', style='Research', palette='Set1')
plt.title('A university rating of 4 or 5 with research will help get a
chance of admission more than 90%')
plt.axhline(.9)
plt.axvline(320)
plt.show()
```

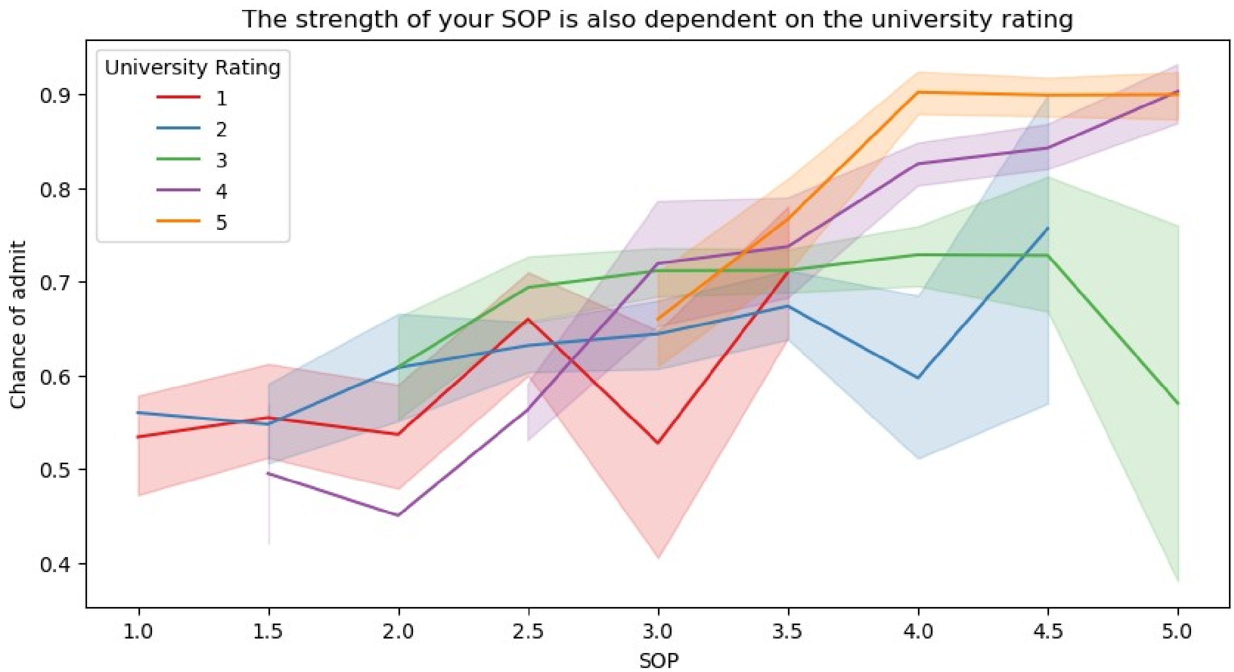



```
#This indicates that the strength of the letter of recommendation is
dependent on the university rating.
#For a university rating of 1, the strenght of the LOR is always 4 or
less
#For a university rating of 2, it is 4.5 or below.
#It also shows that if we have a university rating of 5, even then an
LOR strength of 3 has a wide variance.
#Finally, even if we have a strong LOR, the unoversity rating decides
the chance of admission.

sns.scatterplot(x=data['GRE Score'],y=data[ 'TOEFL Score'],
hue=data['University Rating'],palette="Set1")
```

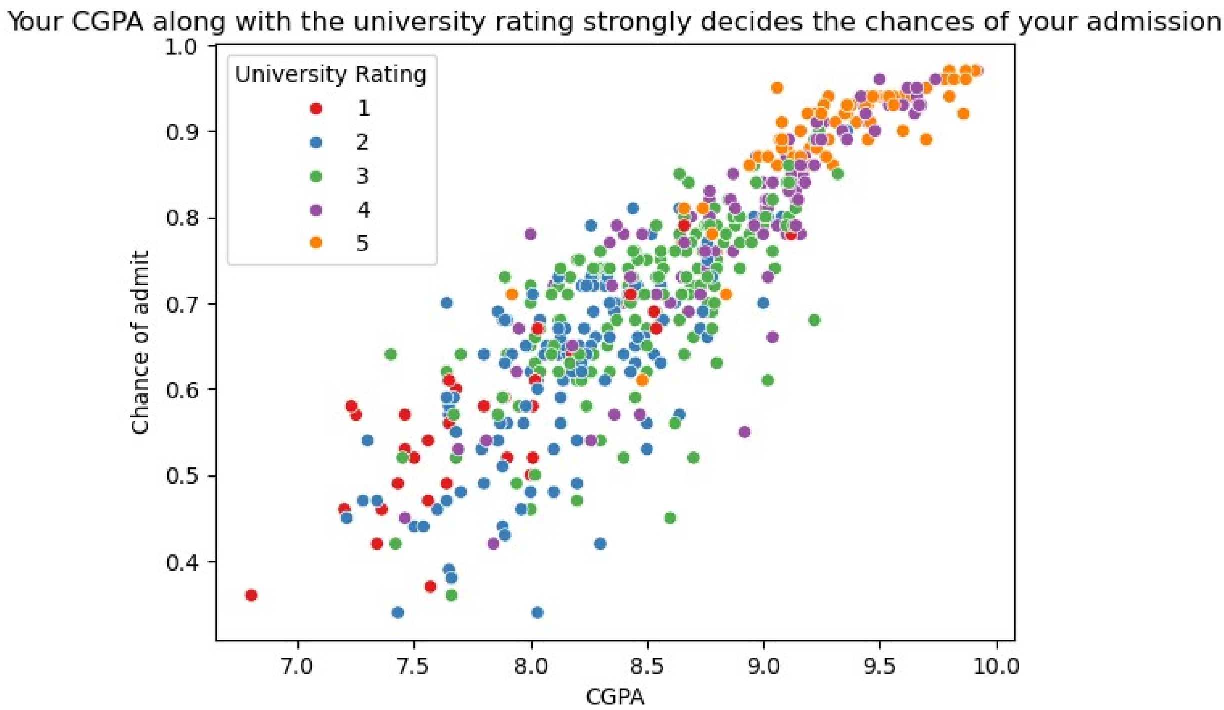
```
<Axes: xlabel='GRE Score', ylabel='TOEFL Score'>
```



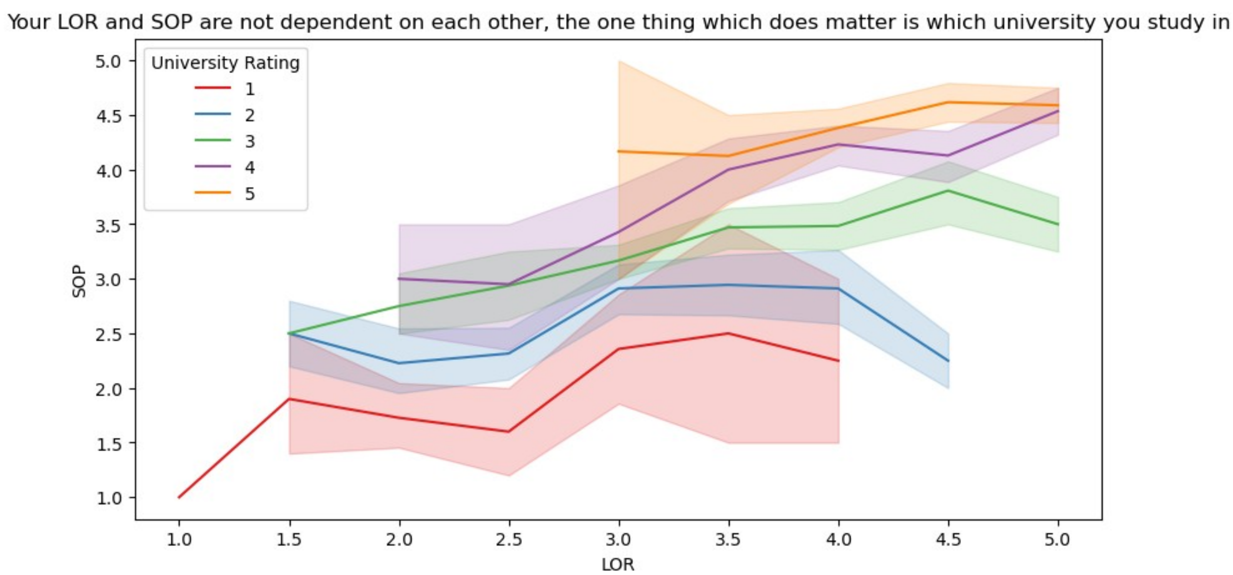

```
#the graph above shows that as the GRE score increases the TOEFL score
also increases, so very high multicollinearity

plt.figure(figsize=(10,5))
sns.lineplot(data=data, x='SOP', y='Chance of admit', hue='University
Rating',palette='Set1')
plt.title('The strength of your SOP is also dependent on the
university rating')
plt.show()
```




```
#Again, there is a relation between university rating and SOP
#For university ratings 1 and 2, we do not have an SOP strength of 5
#Similarly, for a university rating of 4 or 5, a good CGPA score will
lead to more or less a good chance of admission.

sns.scatterplot(data=data, x='CGPA', y='Chance of admit',
hue='University Rating', palette="Set1")
plt.title('Your CGPA along with the university rating strongly decides
the chances of your admission')
plt.show()
```

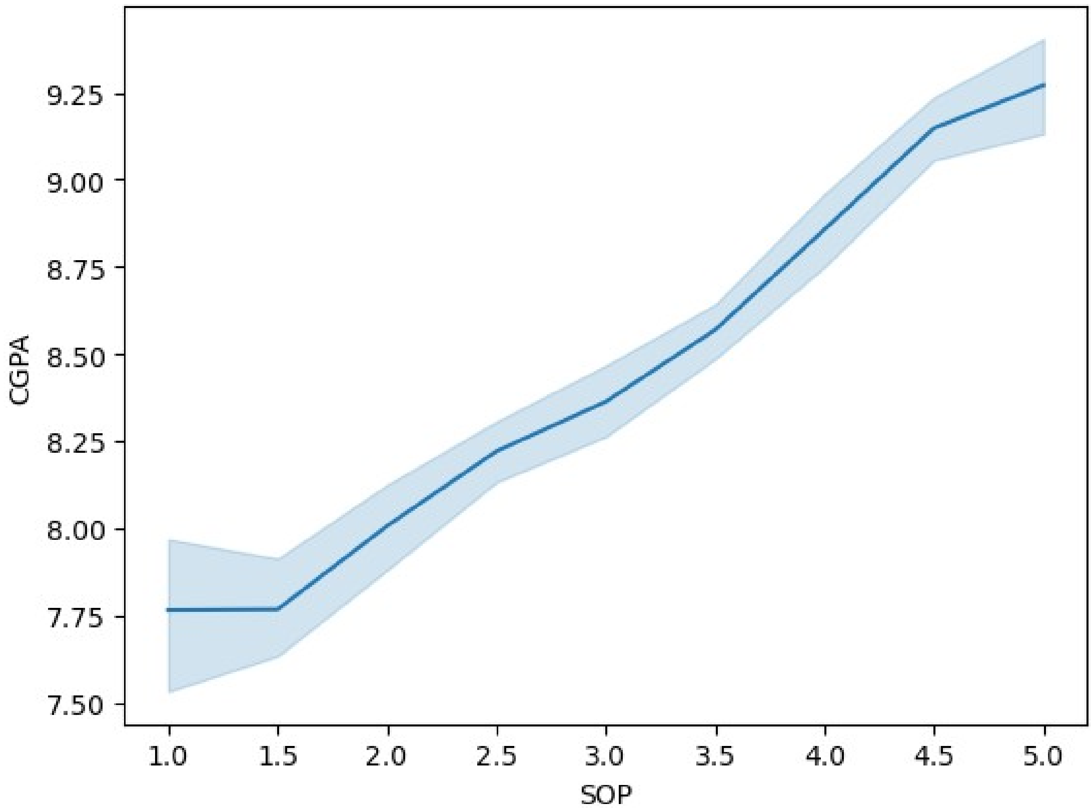



```
plt.figure(figsize=(10,5))
sns.lineplot(data=data, x='LOR', y='SOP', hue='University Rating',palette='Set1')
plt.title('Your LOR and SOP are not dependent on each other, the one thing which does matter is which university you study in')
plt.show()
```

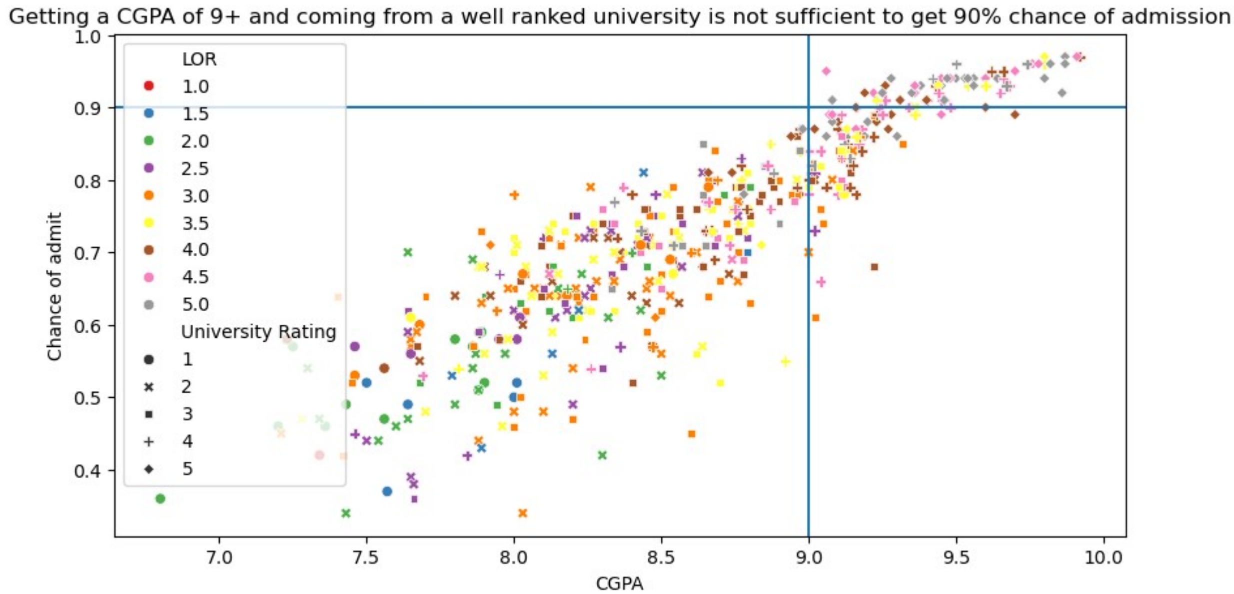



```
sns.lineplot(data=data, x='SOP', y='CGPA') #Again, this shows strong positive correlation or multicollinearity between CGPA and SOP
```

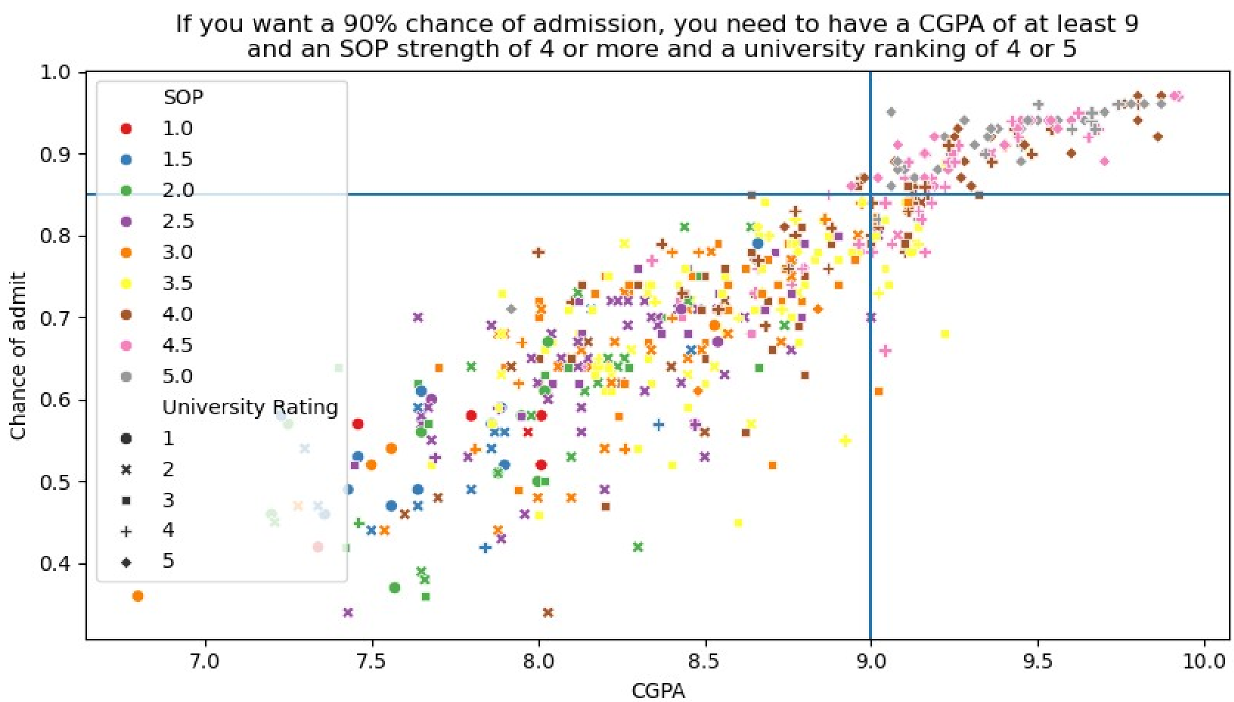
```
<Axes: xlabel='SOP', ylabel='CGPA'>
```




```
plt.figure(figsize=(10,5))
sns.scatterplot(data=data, x='CGPA', y='Chance of admit', hue='LOR',
style='University Rating',palette='Set1')
plt.axhline(.9)
plt.axvline(9)
plt.title('Getting a CGPA of 9+ and coming from a well ranked
university is not sufficient to get 90% chance of admission')
plt.show()
```

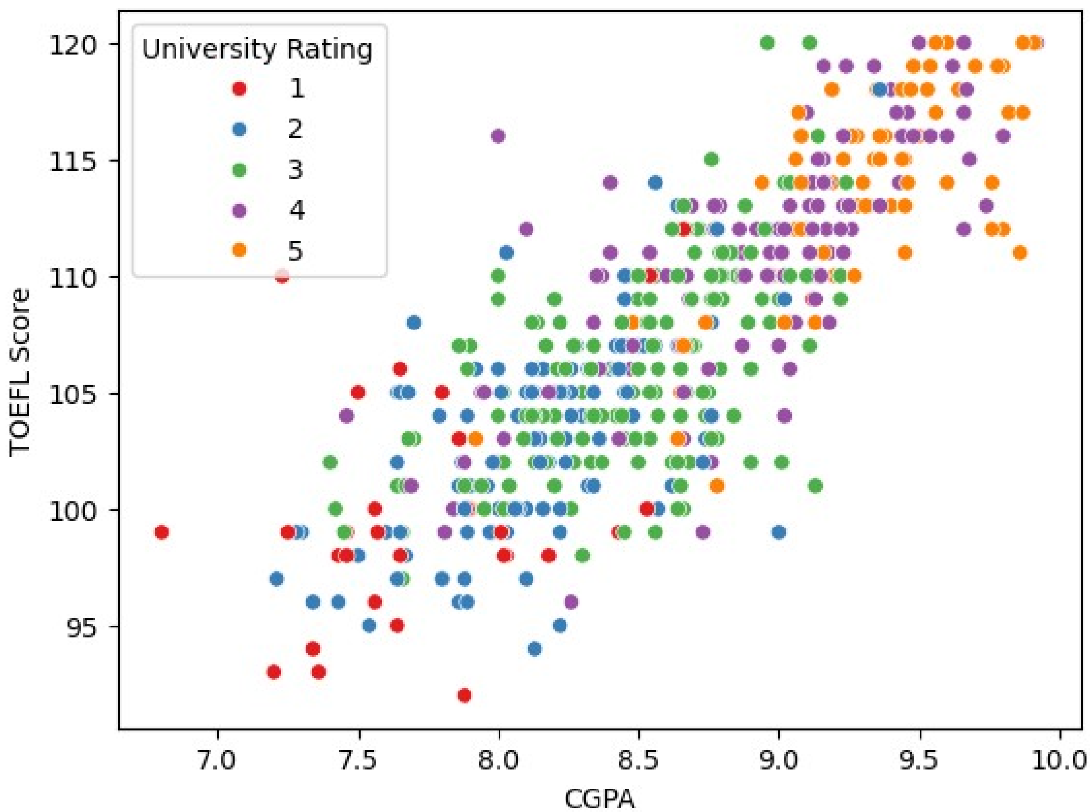



```
plt.figure(figsize=(10,5))
sns.scatterplot(data=data, x='CGPA', y='Chance of admit', hue='SOP',
style='University Rating',palette="Set1")
plt.axhline(.85)
plt.axvline(9)
plt.title("If you want a 90% chance of admission, you need to have a
CGPA of at least 9\n and an SOP strength of 4 or more and a university
ranking of 4 or 5")
plt.show()
```



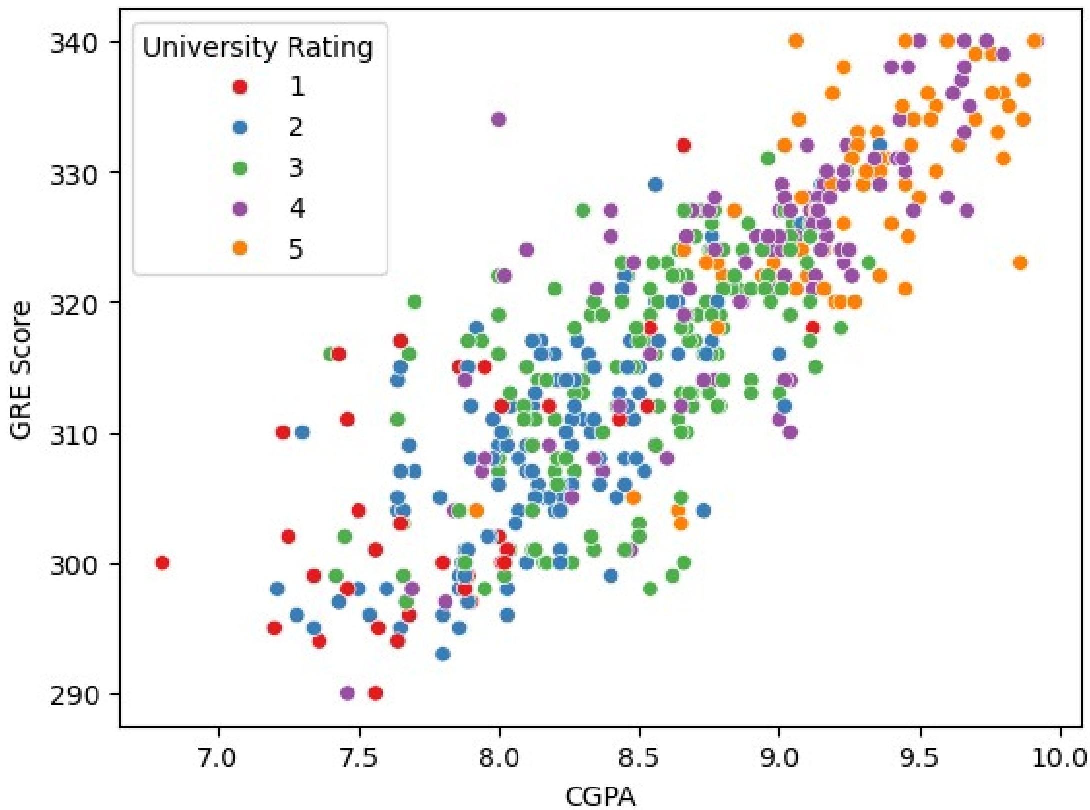
```
#here we see that the SOP needs to be of strength 4.0 and above to get more than 85% chance of acceptance, while the LOR score can still be 3.5 and we can get the acceptance of more than 85% chance of acceptance

sns.scatterplot(data=data, x='CGPA', y='TOEFL Score', hue='University Rating', palette='Set1')
```

```
<Axes: xlabel='CGPA', ylabel='TOEFL Score'>
```

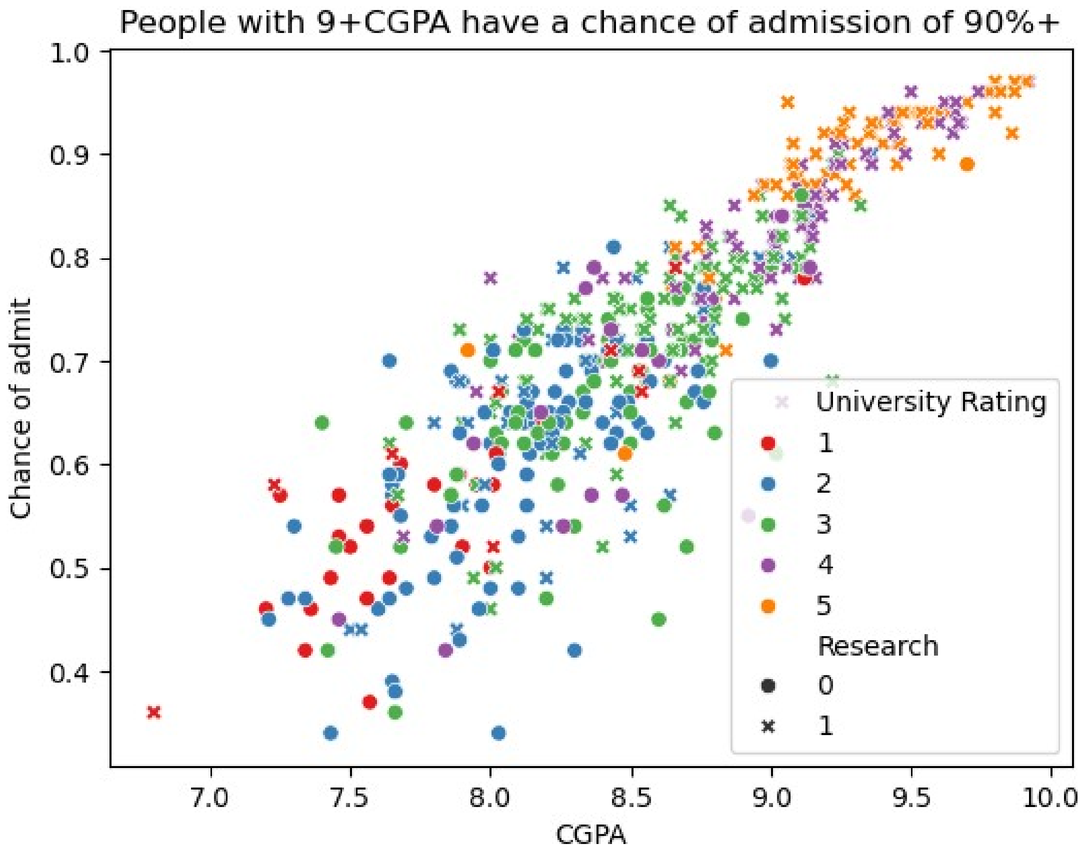



```
sns.scatterplot(data=data, x='CGPA', y='GRE Score', hue='University Rating', palette='Set1')
```

```
<Axes: xlabel='CGPA', ylabel='GRE Score'>
```

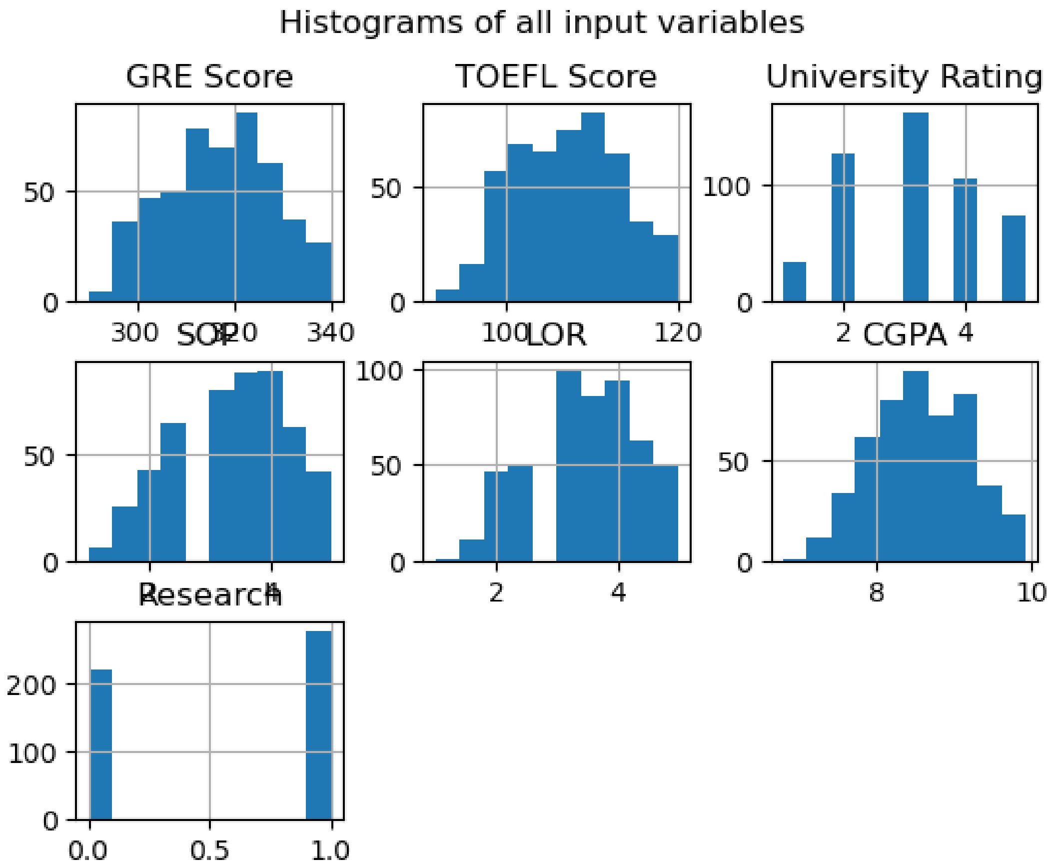



```
sns.scatterplot(data=data, x='CGPA',y='Chance of admit',
hue='University Rating', style='Research', palette='Set1')
plt.title('People with 9+CGPA have a chance of admission of 90%+')
plt.show()
```



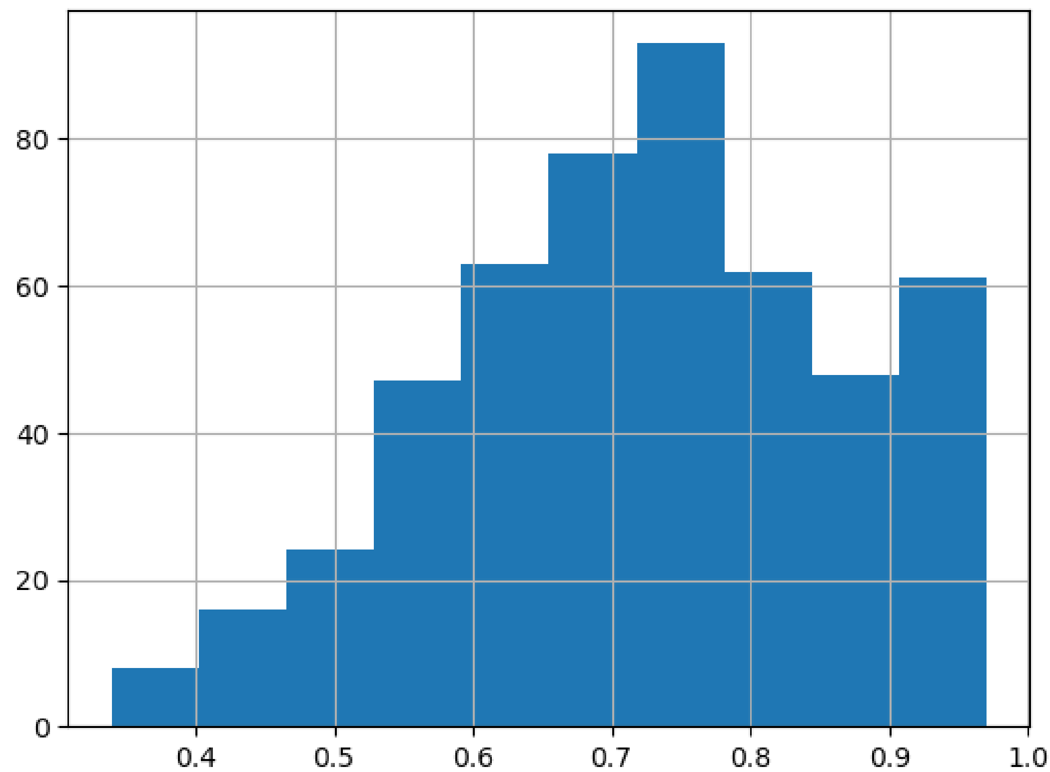

```
plt.figure(figsize=(10,5))
data.drop('Chance of admit', axis=1).hist()
plt.suptitle('Histograms of all input variables')
plt.show()
```

```
<Figure size 1000x500 with 0 Axes>
```




```
data['Chance of admit'].hist() #The output variable is definitely not normal
```

```
<Axes: >
```




```
import statsmodels
from statsmodels.stats.outliers_influence import
variance_inflation_factor as vif

x=data.drop('Chance of admit', axis=1)

x

     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
..         ...          ...                ...  ...  ...   ...        .
..
495        332          108                  5  4.5  4.0  9.02
1
496        337          117                  5  5.0  5.0  9.87
1
```

```
497         330          120                      5  4.5  5.0  9.56
1
498         312          103                      4  4.0  5.0  8.43
0
499         327          113                      4  4.5  4.5  9.04
0

[500 rows x 7 columns]
```

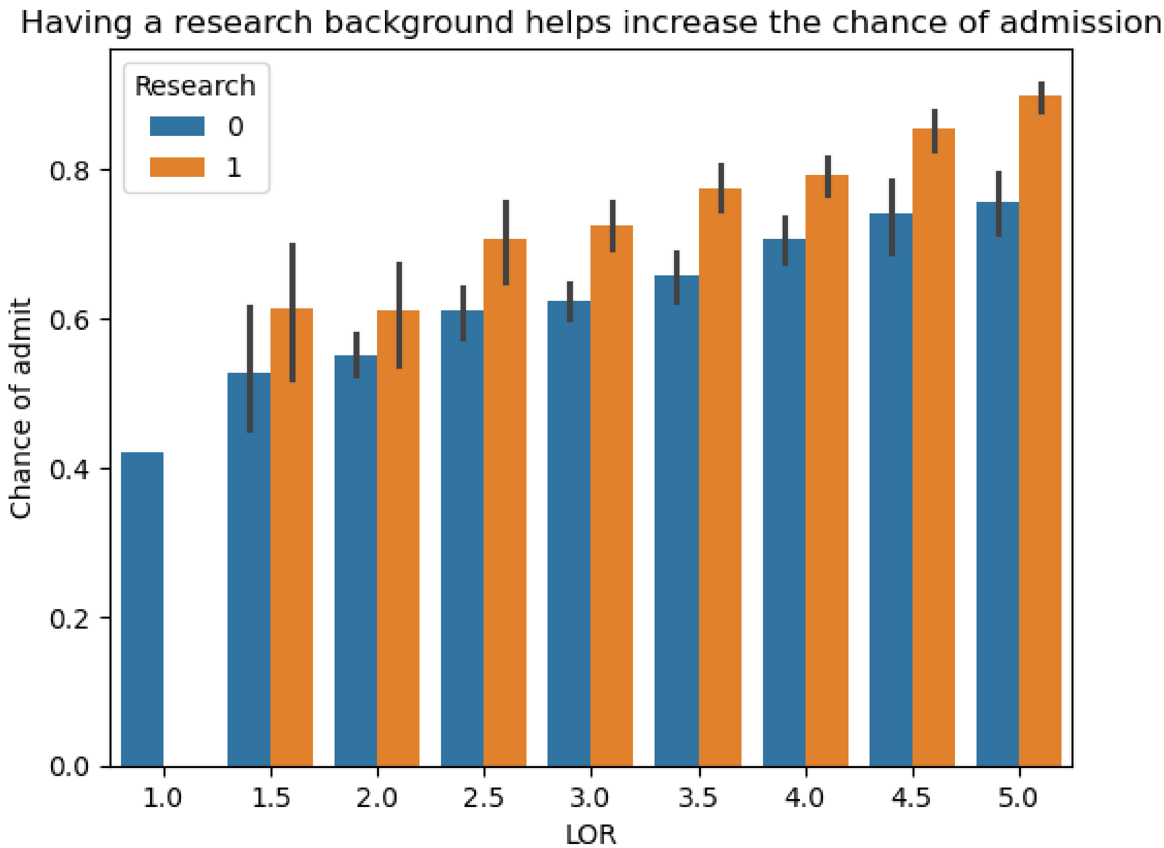
```
x.dtypes
```

```
GRE Score              int64
TOEFL Score            int64
University Rating      int64
SOP                  float64
LOR                  float64
CGPA                 float64
Research               int64
dtype: object
```

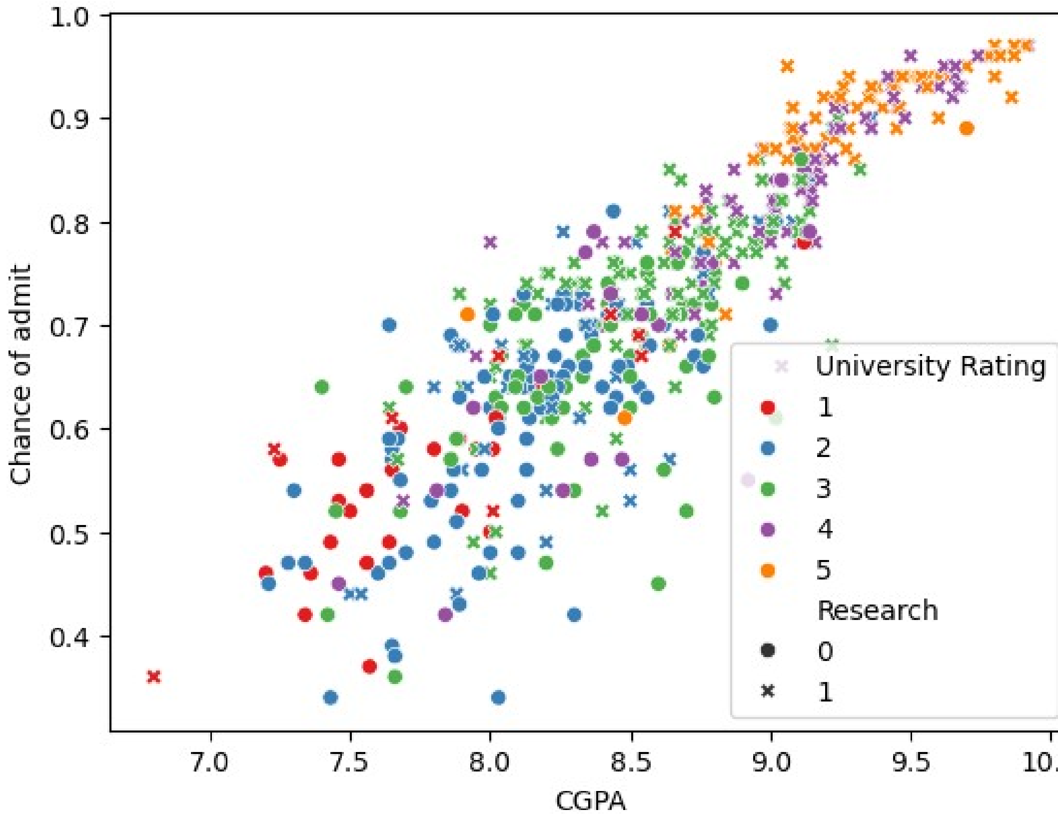
```
x.values
```

```
array([[337.  , 118.  ,   4.  , ...,   4.5 ,   9.65,   1.  ],
       [324.  , 107.  ,   4.  , ...,   4.5 ,   8.87,   1.  ],
       [316.  , 104.  ,   3.  , ...,   3.5 ,   8.  ,   1.  ],
       ...,
       [330.  , 120.  ,   5.  , ...,   5.  ,   9.56,   1.  ],
       [312.  , 103.  ,   4.  , ...,   5.  ,   8.43,   0.  ],
       [327.  , 113.  ,   4.  , ...,   4.5 ,   9.04,   0.  ]])
```

```
sns.barplot(data=data, x='LOR', y='Chance of admit', hue='Research')
plt.title('Having a research background helps increase the chance of admission')
plt.show()
```

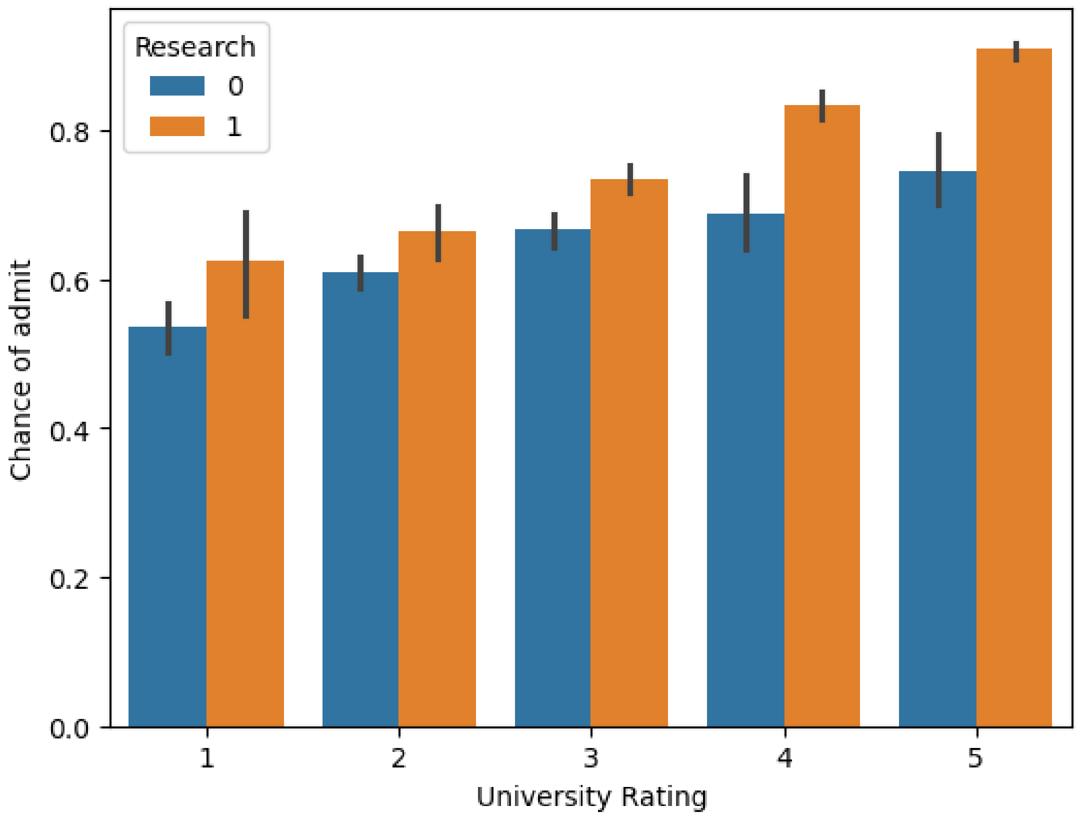



```
sns.scatterplot(data=data, y='Chance of admit', x='CGPA',
hue='University Rating',style='Research', palette='Set1')

<Axes: xlabel='CGPA', ylabel='Chance of admit'>
```

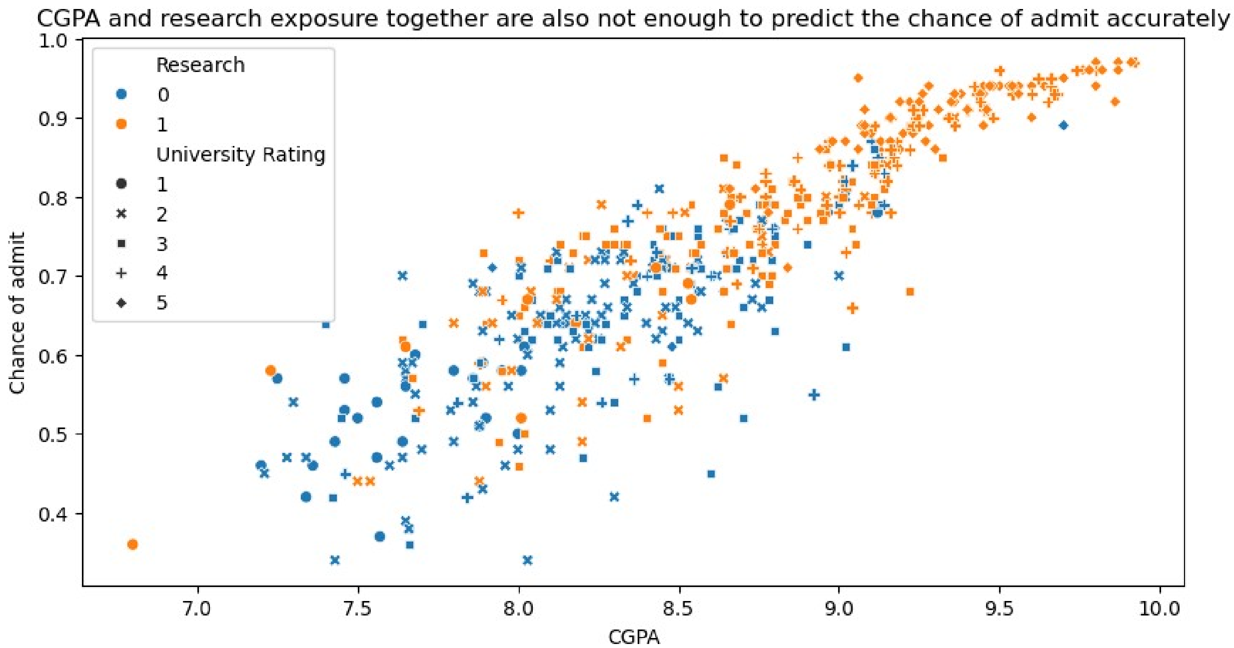



```
sns.barplot(data=data, x='University Rating', y='Chance of admit',
hue='Research')
```

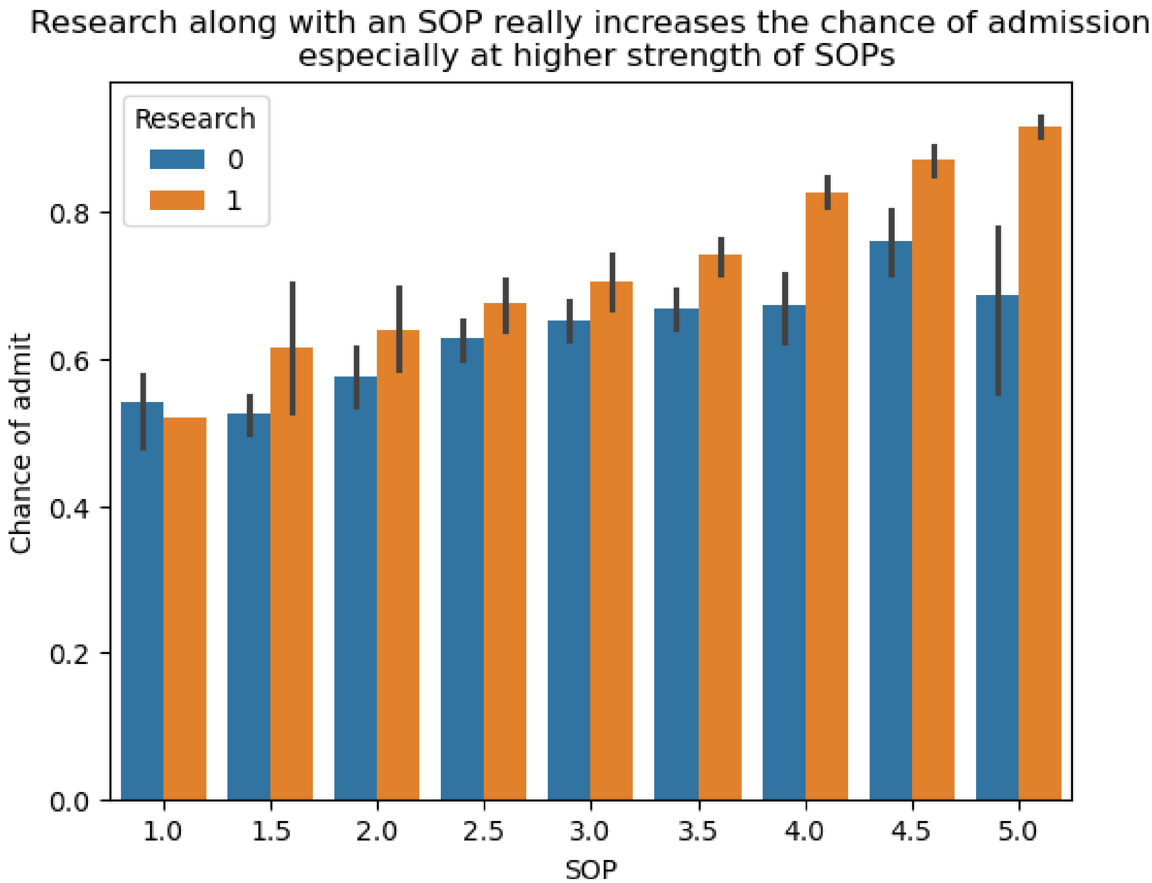
```
<Axes: xlabel='University Rating', ylabel='Chance of admit'>
```




```
plt.figure(figsize=(10,5))
sns.scatterplot(data=data, x='CGPA',y='Chance of admit',
hue='Research', style='University Rating')
plt.title('CGPA and research exposure together are also not enough to
predict the chance of admit accurately')
plt.show()
```




```
sns.barplot(data=data, x='SOP', y='Chance of admit', hue='Research')
plt.title('Research along with an SOP really increases the chance of admission\n especially at higher strength of SOPs')
plt.show()
```




```
vif_df=pd.DataFrame()
vif_df['features']=x.columns
vif_df['vif']=[vif(x.values, i) for i in range(x.shape[1])]

vif_df.sort_values('vif', ascending=False)
```

	features	vif
0	GRE Score	1308.061089
1	TOEFL Score	1215.951898
5	CGPA	950.817985
3	SOP	35.265006
4	LOR	30.911476
2	University Rating	20.933361
6	Research	2.869493

```
#Dropping one by one
x_new=x.drop(['GRE Score'], axis=1)

vif_df_1=pd.DataFrame()
vif_df_1['Features']=x_new.columns
vif_df_1['VIF']=[vif(x_new,i) for i in range(x_new.shape[1])]
```

```
vif_df_1.sort_values('VIF', ascending=False)
```

```
              Features         VIF
4                 CGPA  728.778312
0          TOEFL Score  639.741892
2                  SOP   33.733613
3                  LOR   30.631503
1    University Rating   19.884298
5             Research    2.863301
```

```
x_new_2=x_new.drop('CGPA', axis=1)
vif_df_2=pd.DataFrame()
vif_df_2['Features']=x_new_2.columns
vif_df_2['VIF']=[vif(x_new_2.values,i) for i in
range(x_new_2.shape[1])]
```

```
vif_df_2.sort_values('VIF', ascending=False)
```

```
              Features        VIF
2                  SOP  33.273087
3                  LOR  29.531351
0          TOEFL Score  22.035055
1    University Rating  19.747053
4             Research   2.849489
```

```
x_new_3=x_new_2.drop('SOP', axis=1)
vif_df_3=pd.DataFrame()

vif_df_3['Features']=x_new_3.columns

vif_df_3['vif']=[vif(x_new_3.values, i) for i in
range(x_new_3.shape[1])]
```

```
vif_df_3.sort_values('vif', ascending=False)
```

```
              Features        vif
2                  LOR  25.700130
0          TOEFL Score  19.844499
1    University Rating  14.952839
3             Research   2.824467
```

```
x_new_4=x_new_3.drop('LOR', axis=1)
vif_df_4=pd.DataFrame()
vif_df_4['Features']=x_new_4.columns

vif_df_4['vif']=[vif(x_new_4,i) for i in range(x_new_4.shape[1])]
```

```
vif_df_4.sort_values('vif', ascending=False)
```

```
              Features        vif
1    University Rating  11.840110
```

```
0        TOEFL Score  10.258756
2           Research   2.780788
```

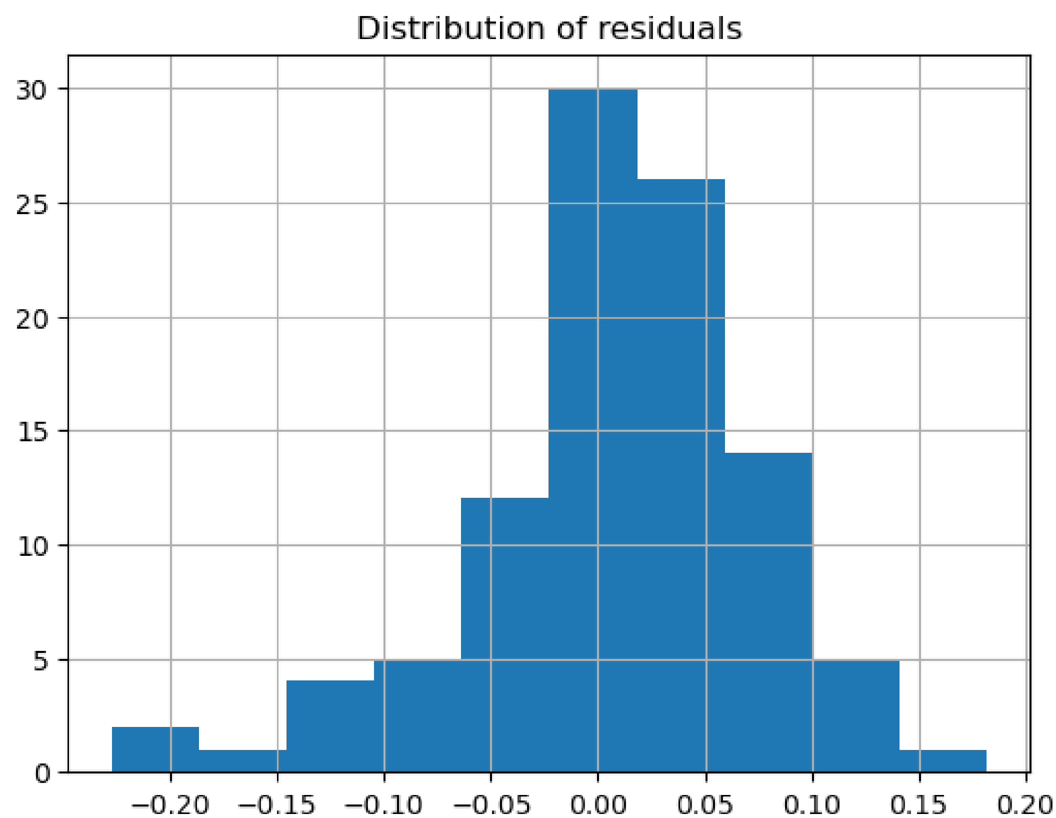
```python
y=data['Chance of admit']

import statsmodels.api as sm
import statsmodels.formula.api as smf

import sklearn
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest=train_test_split(x[['University Rating','TOEFL Score','Research']], y, test_size=0.2, random_state=10)

sc=RobustScaler() #I use robust scaler because it can take care of outliers better than standard scaler

xtrain_sc=sc.fit_transform(xtrain)
xtest_sc=sc.transform(xtest)

#sm.add_constant(xtrain)
model=sm.OLS(ytrain, sm.add_constant(xtrain_sc)).fit()



model.summary()
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results

======================================================================
========
Dep. Variable:       Chance of admit   R-squared:
0.702
Model:                           OLS   Adj. R-squared:
0.700
Method:                Least Squares   F-statistic:
310.9
Date:               Sun, 06 Apr 2025   Prob (F-statistic):
1.09e-103
Time:                       08:16:38   Log-Likelihood:
452.23
No. Observations:                400   AIC:
-896.5
Df Residuals:                    396   BIC:
-880.5
Df Model:                          3

Covariance Type:           nonrobust
```

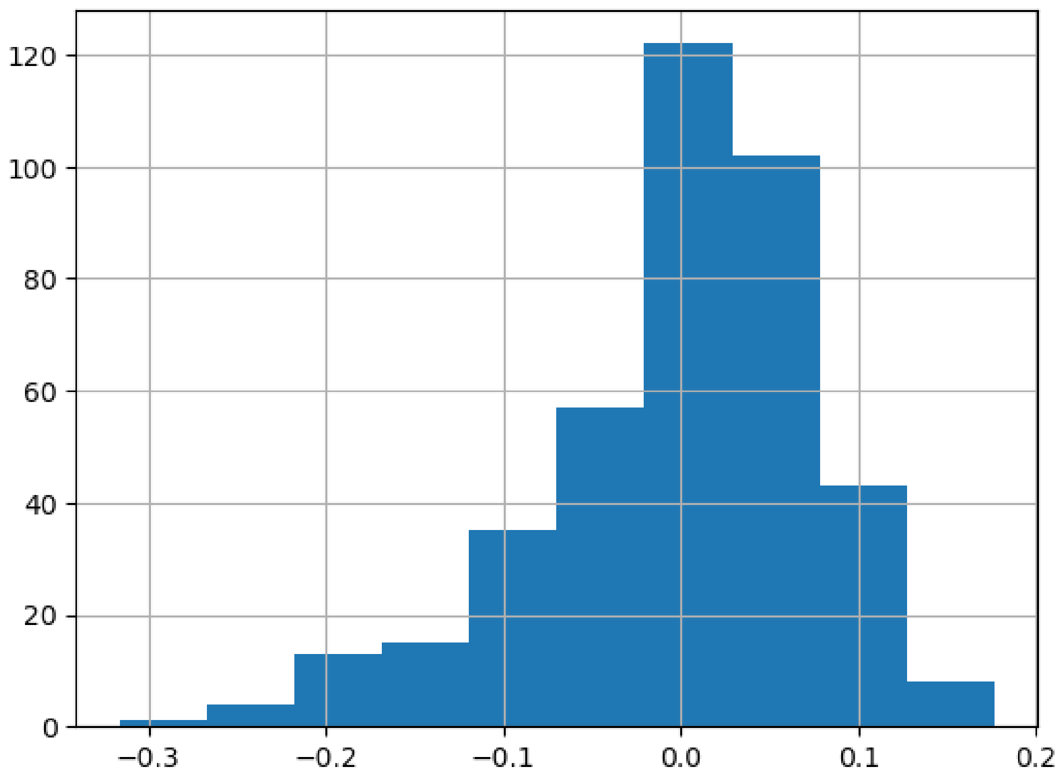
```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.7391      0.006    131.152      0.000       0.728       0.750
x1             0.0661      0.009      7.132      0.000       0.048       0.084
x2             0.1112      0.008     13.678      0.000       0.095       0.127
x3             0.0568      0.009      6.206      0.000       0.039       0.075
==============================================================================
Omnibus:                       50.690   Durbin-Watson:                   1.918
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               69.515
Skew:                          -0.881   Prob(JB):                     8.04e-16
Kurtosis:                       4.032   Cond. No.                         3.15
==============================================================================

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

#This shows that all the features have a p value of 0 which makes them significant
#This also shows that my R square and adjusted R squared are around 70%

ypred=model.predict(sm.add_constant(xtest_sc))

from sklearn.metrics import r2_score, mean_squared_error

print(r2_score(ytest, ypred))

0.7202136807576062

print(mean_squared_error(ytest, ypred))

0.004835544157603101

model.params #This shows that the maximum weightage is given to TOEFL score followed by a university ranking and then the research
```

```
const    0.739085
x1       0.066082
x2       0.111150
x3       0.056768
dtype: float64
```

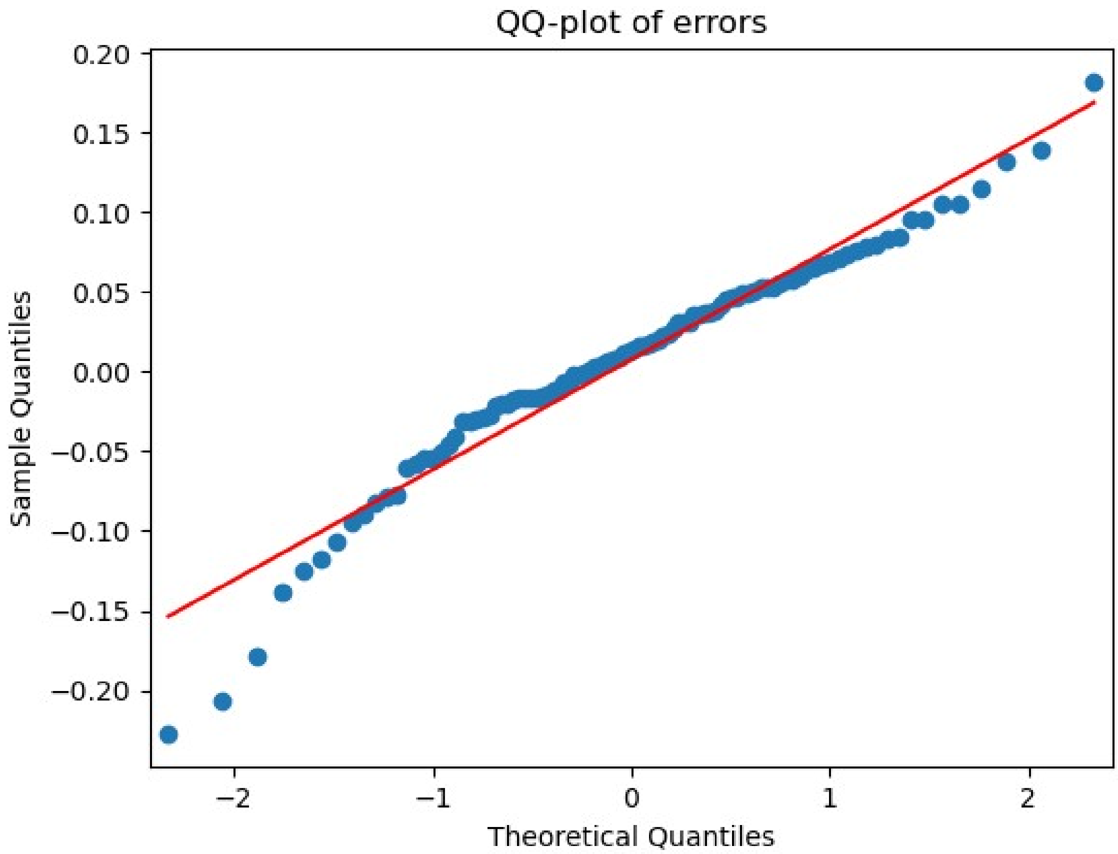
```
#Assumption 3. Testing for normal distribution of residuals or errors

errors=ytest-ypred

errors.hist()
plt.title('Distribution of residuals')
plt.show()
```



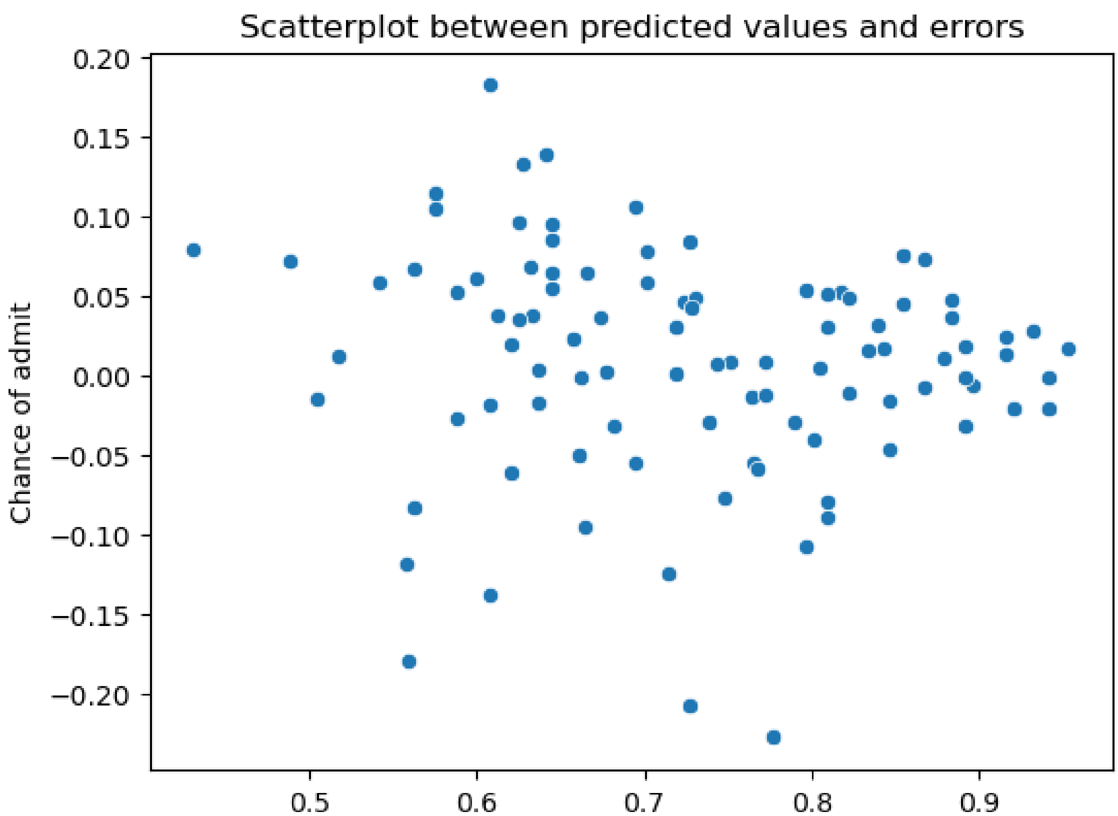

```
model.resid.hist()

<Axes: >
```




```
sm.qqplot(errors, line='s')
plt.title('QQ-plot of errors')
plt.show() #This shows the errors are not varying normally, especially
in the bottom part of the chart
```

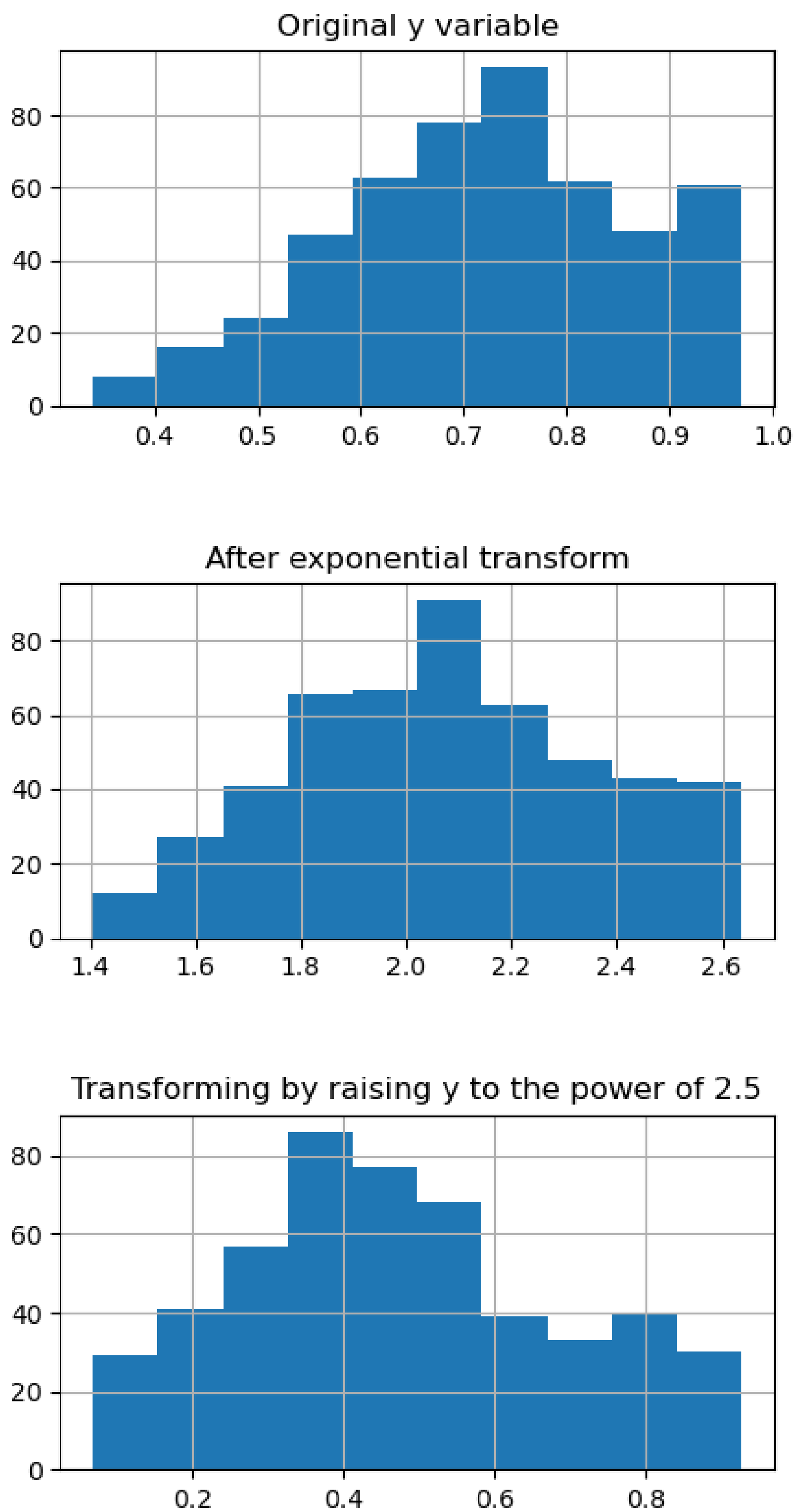



```
import scipy

from scipy.stats import shapiro

shapiro(errors)

ShapiroResult(statistic=0.9607114267806554,
pvalue=0.004520814834426949)

shapiro(model.resid)

ShapiroResult(statistic=0.9529907110853131, pvalue=5.559417892019577e-
10)

#This p value is much less than 0.05 so the data is not normal, hence
we can reject the null hypothesis
#So ideally, linear regression should not be applicable on this data

#Assumption 4. Test for homoskedasticity by scatter plot

sns.scatterplot(x=ypred, y=errors) #The shape is not very clear so
doing the statistical test below as well
plt.title('Scatterplot between predicted values and errors')
plt.show()
```




```
#tests for autocorrelation:
statsmodels.stats.stattools.durbin_watson(model.resid)
#since the value is close to 2, it appears there is no autocorrelation

1.9182515771561721

#The errors are not normally distributed

#tests for homoskedasticity by using the het_white test

from statsmodels.stats.diagnostic import het_white

#model.model.exog

het_white(model.resid, model.model.exog) #The p value of .00032 is
less than 0.05, hence we can reject the null hypothesis
#Therefore the data is not homoskedastic
#So linear regression is not applicable

(28.931331823321038,
 0.00032601744171109067,
 3.8106662300885783,
 0.00025086280545936004)

#However, I read while researching for this project that it is
possible to transform the variables and recheck and then apply linear
```

```
regression and take an inverse transform.
#I also read that we can use WLS instead of OLS to run linear
regression when the homoskedasticity assumption fails, and since we do
not know the weights beforehand, I ran WLS from scratch.
#I am sharing my results below.
#(I am pasting the links where I read this below, please do correct
and share feedback if I am
mistaken.#https://www.statsmodels.org/dev/examples/notebooks/generated
/wls.html
#https://www.itl.nist.gov/div898/handbook/pmd/section4/pmd453.htm)

#to fo fix this issue, I tried to take a transform of the target
variable to make it more normal and then recalculate things

#I am pasting below the y variable with different transforms that I
tried.
# I finally selected the one with y**2.5 because it satisfied the
assumptions albeit very narrowly

plt.figure(figsize=(5,10))
plt.subplot(3,1,1)
data['Chance of admit'].hist()
plt.title('Original y variable')
plt.subplot(3,1,2)
np.exp(data['Chance of admit']).hist()
plt.title('After exponential transform')
plt.subplot(3,1,3)
(data['Chance of admit']**2.5).hist()
plt.title('Transforming by raising y to the power of 2.5')
plt.subplots_adjust(hspace=0.5)
```

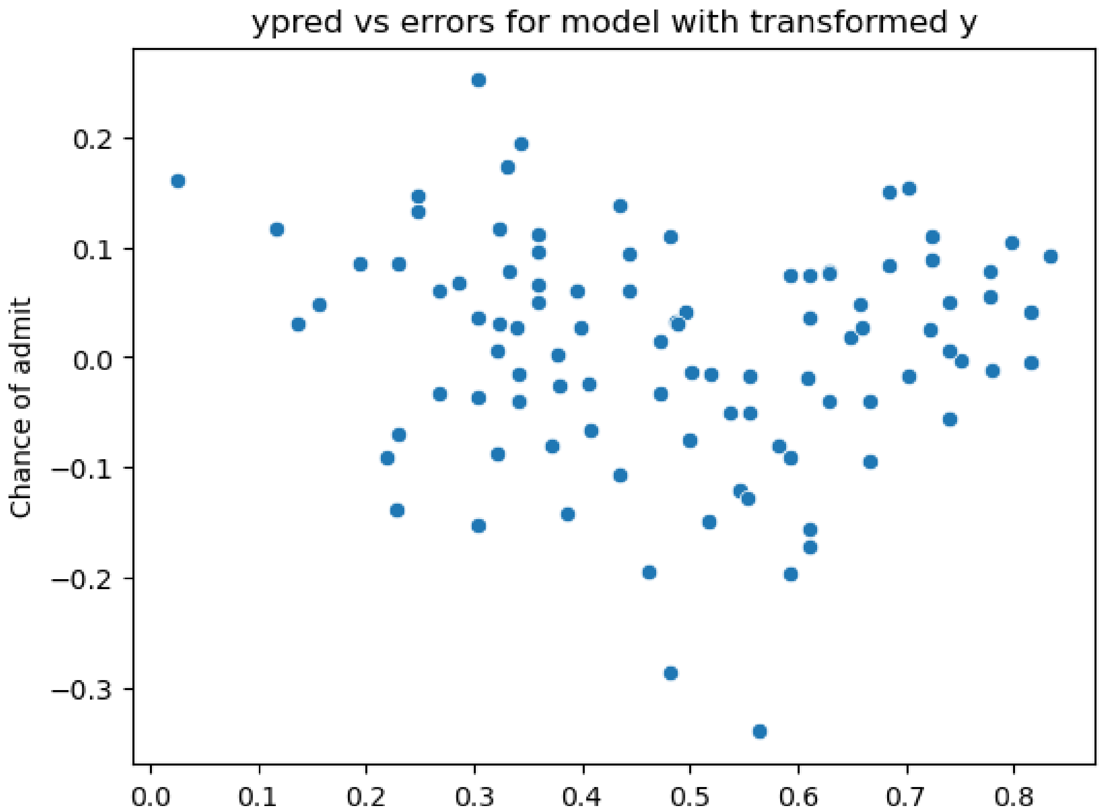

Original y variable
0
20
40
60
80
0.4
0.5
0.6
0.7
0.8
0.9
1.0
After exponential transform
0
20
40
60
80
1.4
1.6
1.8
2.0
2.2
2.4
2.6
Transforming by raising y to the power of 2.5
0
20
40
60
80
0.2
0.4
0.6
0.8

```
#y_new=np.exp(y)
y_new=y**2.5

xtrain, xtest, ytrain_new, ytest_new=train_test_split(x[['University
Rating','TOEFL Score','Research']], y_new, test_size=0.2,
random_state=10)

xtrain_sc=sc.fit_transform(xtrain)
xtest_sc=sc.transform(xtest)

model_new=sm.OLS(ytrain_new, sm.add_constant(xtrain_sc)).fit()

model_new.summary()
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results

======================================================================
========
Dep. Variable:         Chance of admit   R-squared:
0.747
Model:                             OLS   Adj. R-squared:
0.745
Method:                  Least Squares   F-statistic:
390.2
Date:                 Sun, 06 Apr 2025   Prob (F-statistic):
7.50e-118
Time:                         08:17:06   Log-Likelihood:
322.61
No. Observations:                  400   AIC:
-637.2
Df Residuals:                      396   BIC:
-621.3
Df Model:                            3

Covariance Type:             nonrobust

======================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
--------
const          0.4998      0.008     64.146      0.000       0.485
0.515
x1             0.1095      0.013      8.550      0.000       0.084
0.135
x2             0.1689      0.011     15.032      0.000       0.147
0.191
x3             0.0842      0.013      6.657      0.000       0.059
```

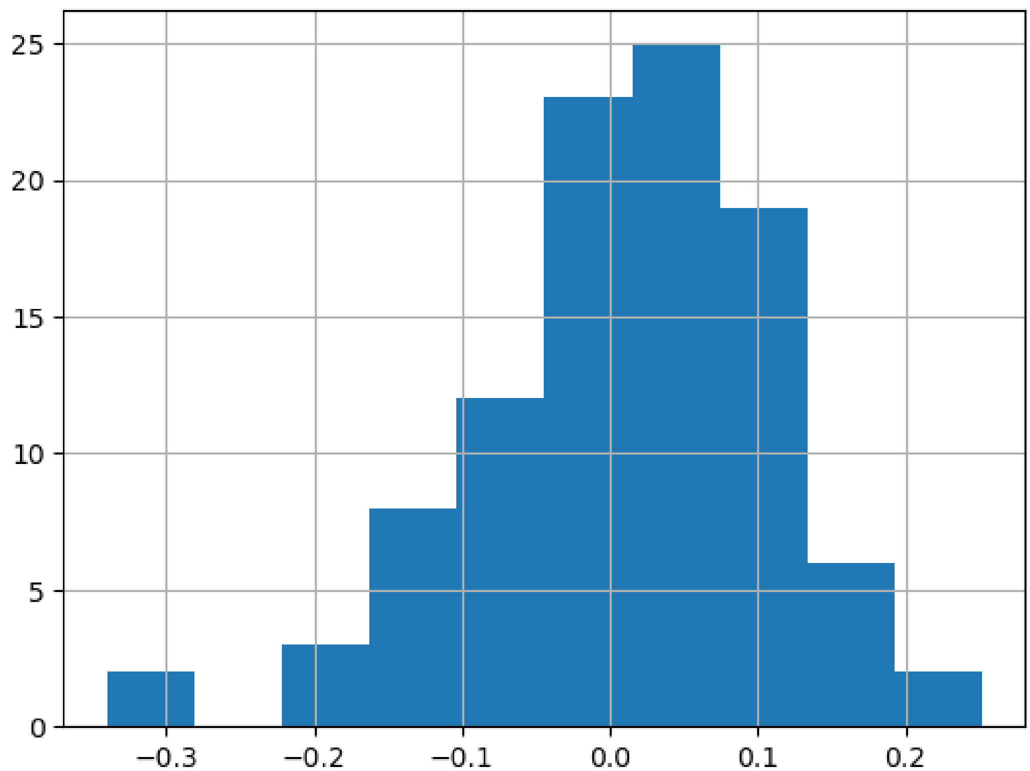
```
0.109
==============================================================================
========
Omnibus:                       20.129   Durbin-Watson:
1.896
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
21.796
Skew:                          -0.562   Prob(JB):
1.85e-05
Kurtosis:                       3.216   Cond. No.
3.15
==============================================================================
========

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

#The r squared and adjusted r squared have increased, meaning the
model captures the variance better

ypred_new=model_new.predict(sm.add_constant(xtest_sc))

errors_new=ytest_new-ypred_new

errors_new

151    0.078415
424    0.049223
154    0.138030
190    0.083999
131    0.032625
         ...
50     0.172610
264   -0.014220
34     0.153481
78    -0.091441
223    0.026362
Name: Chance of admit, Length: 100, dtype: float64

shapiro(model_new.resid)

ShapiroResult(statistic=0.9780720818003544, pvalue=9.43232976750163e-
06)

shapiro(errors_new) #here my Shapiro result gives me a p value > 0.05
when I test for normality of errors

ShapiroResult(statistic=0.9769044314301868,
pvalue=0.07609045640886782)
```

```
#test for heteroskedasticity
sns.scatterplot(x=ypred_new, y=errors_new)
plt.title('ypred vs errors for model with transformed y')
plt.show() #This plot looks a lot better than the earlier one
```



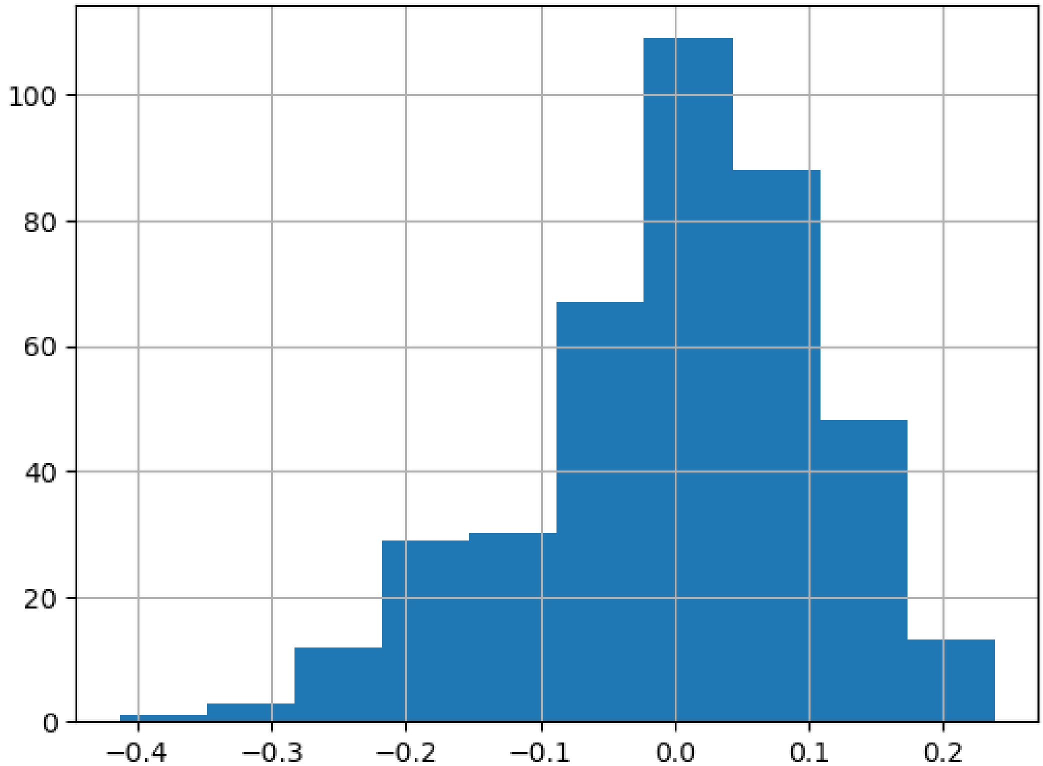

```
errors_new.hist()
```

```
<Axes: >
```

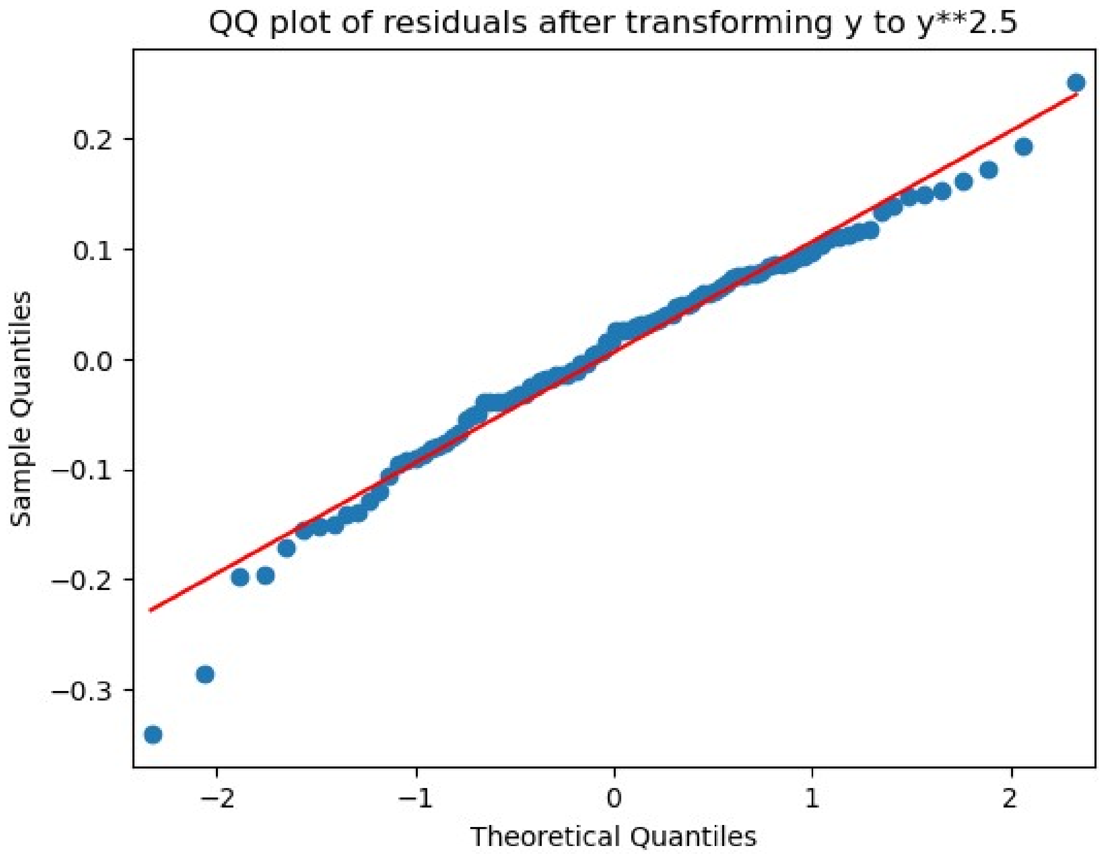



```
model_new.resid.hist()
```

```
<Axes: >
```




```
sm.qqplot(errors_new, line='s')
plt.title('QQ plot of residuals after transforming y to y**2.5')
plt.show()
```




```
het_white(model_new.resid, model_new.model.exog)
```

```
(14.221365138978825,
 0.07617497421614607,
 1.801730729380574,
 0.0751732004793797)
```

```
# the p value for the het while test is also 0.07 so now all the
assumptions are satisfied.

model_new.params
```

```
const    0.499829
x1       0.109539
x2       0.168899
x3       0.084192
dtype: float64
```

```
print(r2_score(ytest_new, ypred_new))
```

```
0.7573952575750972
```

```
print(mean_squared_error(ytest_new, ypred_new))
```

```
0.010109047558311008
```

```
#Therefore I believe we should take a prediction and take the 2.5th
root of it if we want to get good results from the linear regression



#The alternative is to use WLS and dividing by the variance of the
weights
#I am trying it manually below because we do not know the initial
weights so I am taking them randomly

xtrain_sc.shape

(400, 3)

w=np.arange(xtrain_sc.shape[1])
#I am assigning different weights because if I assign all weights as 1
then the variance will be zero
#This will cause an error in the WLS function

#w.shape

#xtrain.shape

#ytrain.shape

#I followed the concept of WLS and tried manually, however I am
getting a very different result
#Can you please give some feedback on where I am going wrong here?
#Is this formula correct?

def wls(xtrain, ytrain, lr, weights=w):
    bias=np.zeros(xtrain.shape[1])
    #e=(ytrain-np.mean(ytrain))**2
    for i in range(50):
        ypred_cust=np.dot(xtrain, weights)+sum(bias)

        e=ytrain-ypred_cust
        mse=np.mean(sum(e**2))
        if(mse<.01):
            break
        else:
            weights=weights-(lr*np.dot(xtrain.T,e**2)/len(xtrain))-
(((2*lr)*np.dot(xtrain.T, e))/len(xtrain))
            #This is the weight update formula for WLS, we take MSE as
(w*(y-ypred)**2)/N
            #This I understood from the site
https://www.stat.cmu.edu/~cshalizi/mreg/15/lectures/24/lecture-24--
25.pdf

            bias=bias-((2*lr*e).sum()/len(xtrain))
    return(weights, bias)
```

```
weights_wls, bias_wls=wls(xtrain_sc, ytrain,.001)

weights_wls, bias_wls

(array([0.12560387, 1.22341097, 2.42169278]),
 array([-0.19089759, -0.19089759, -0.19089759]))

xtest_sc.shape

(100, 3)

ypred_wls=np.dot(xtest_sc,weights_wls)+sum(bias_wls)

ytest.shape

(100,)

ypred_wls

array([ 0.77632206,  0.50445296, -2.85845101,  0.0966493 , -
3.1406508 ,
       -0.98049644, -3.32905661, -0.31115436, -3.46499116, -
0.70862733,
        0.71352013,  0.23258385,  0.03384736, -3.40218922, -
0.10208719,
       -2.38784542, -1.5722381 , -1.23756706, -1.26269622, -
3.19312205,
       -3.53812377, -0.8342312 , -0.78175995,  0.16978192, -
5.15900772,
        1.04819117, -3.40218922, -4.34340041, -3.26625467, -
2.58658191,
        1.04819117,  0.30571647, -1.3883001 , -1.45110203, -
0.57269278,
        1.12132378, -0.10208719, -3.67405833, -3.60092571, -
1.25236554,
        1.12132378, -3.73686026, -0.84456189,  0.3685184 ,
1.18412572,
       -4.35373109, -4.00872937, -2.25191087, -0.85489257, -
3.47532184,
       -0.44708891, -2.92125295, -4.82433668, -3.19312205, -
0.90736382,
       -0.23802174,  0.50445296, -3.87279481, -2.84812033,
0.77632206,
       -3.19312205,  0.0966493 , -0.10208719, -3.87279481,
0.03384736,
       -3.73686026, -0.50989085, -2.79564908, -3.93559675, -
2.11597631,
```

```
        0.71352013, -3.80999288, -4.00872937, -2.58658191, -
0.30082368,
       -3.40218922, -3.19312205, -1.22723638, -0.23802174, -
0.43675823,
       -3.32905661, -3.40218922, -2.85845101, -3.80999288,
0.30571647,
       -0.50989085,  0.50445296, -3.80999288, -0.70862733,
1.25725834,
       -0.10208719, -3.67405833, -4.20746586,  0.10697998, -
0.98049644,
       -1.79610375, -0.22769106,  0.23258385, -2.26670935, -
2.7853184 ])
```

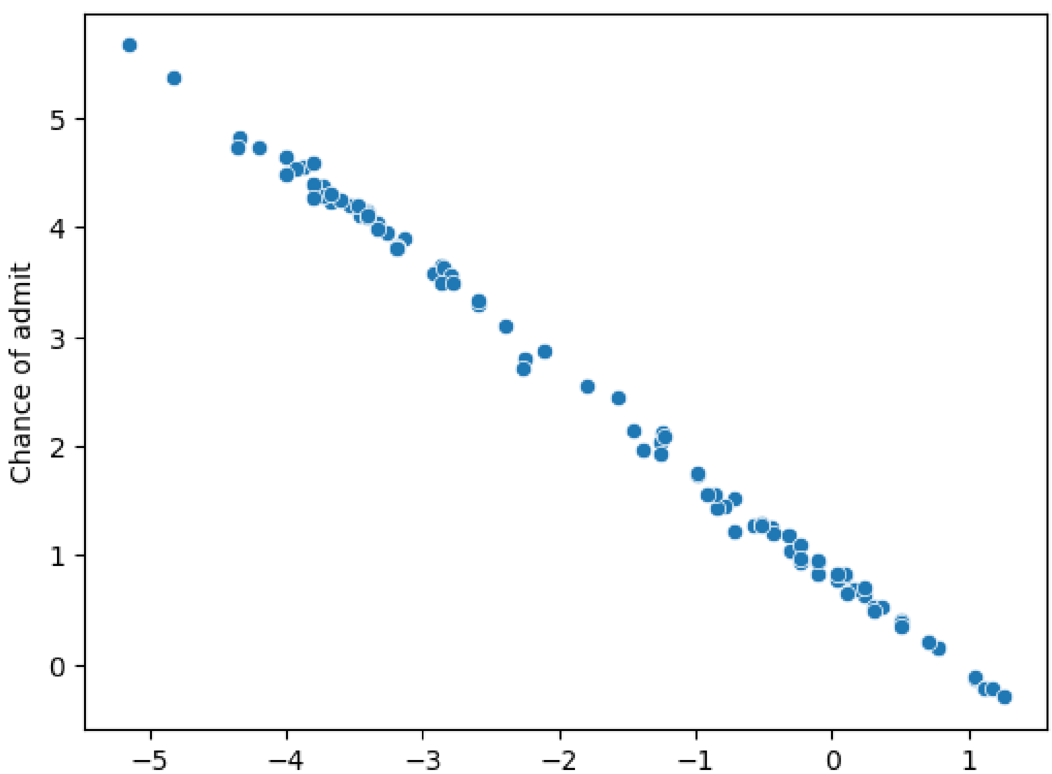
```
errors_wls=ytest-ypred_wls
```

```
shapiro(errors_wls) #This one fails when I try it out manually, it
works when I use the statsmodels
#I need to check why
```

```
ShapiroResult(statistic=0.9183541472339452,
pvalue=1.1523530294281153e-05)
```

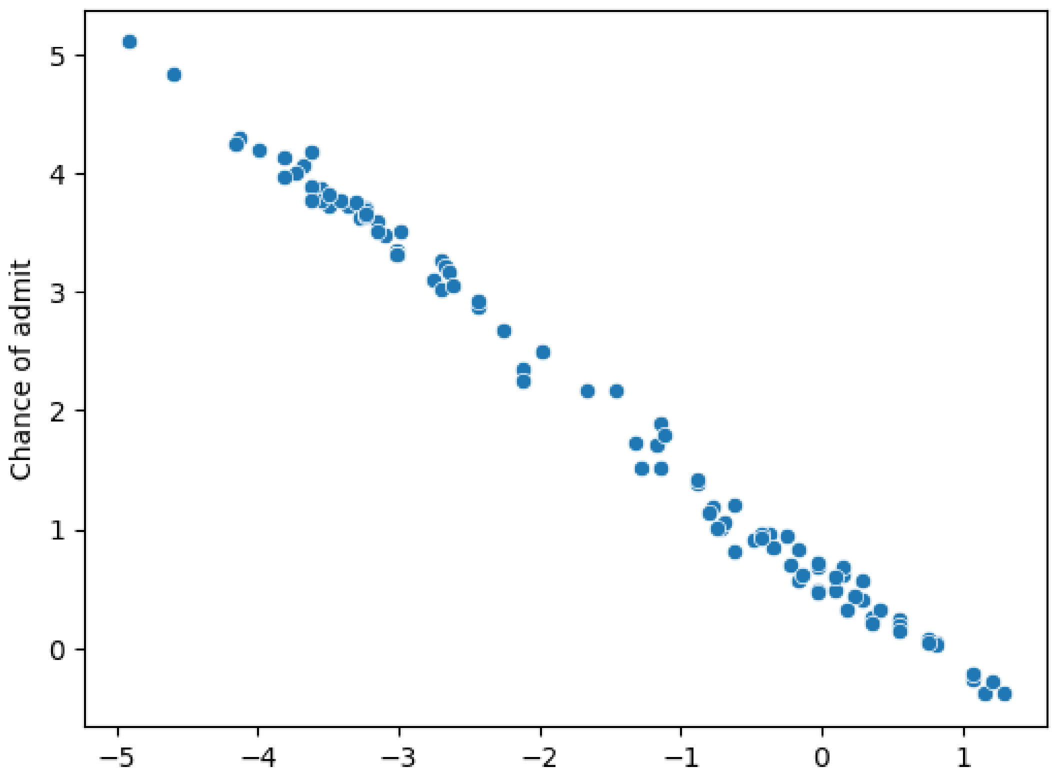
```
sns.scatterplot(x=ypred_wls, y=errors_wls)
```

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



```
# I also tried fitting the same WLS model on the transformed y
variable and there was no impact at all

weights_wls_2, bias_wls_2=wls(xtrain_sc, ytrain_new,.001)

ypred_wls_2=np.dot(xtest_sc,weights_wls_2)+sum(bias_wls_2)

errors_2_wls=ytest_new-ypred_wls_2

sns.scatterplot(x=ypred_wls_2, y=errors_2_wls) #This is not matching
the result from the statsmodels library,
#I tried to understand the formula for WLS and apply it, and I will
recheck again
```

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

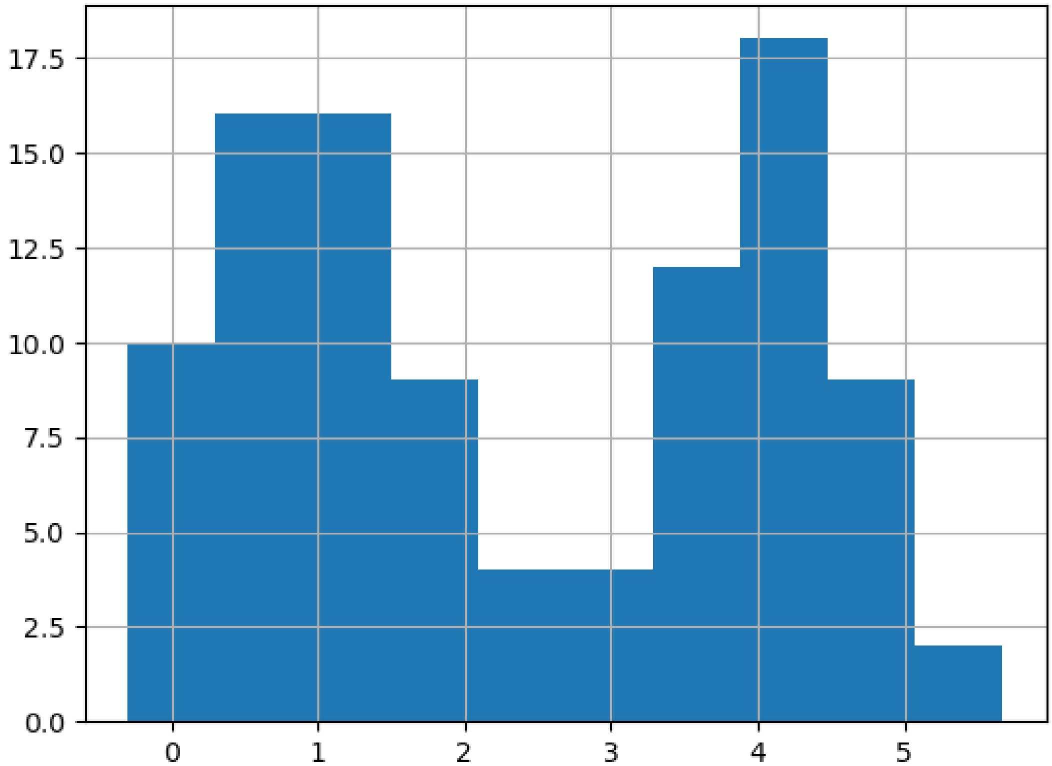



```
shapiro(errors_wls)
```

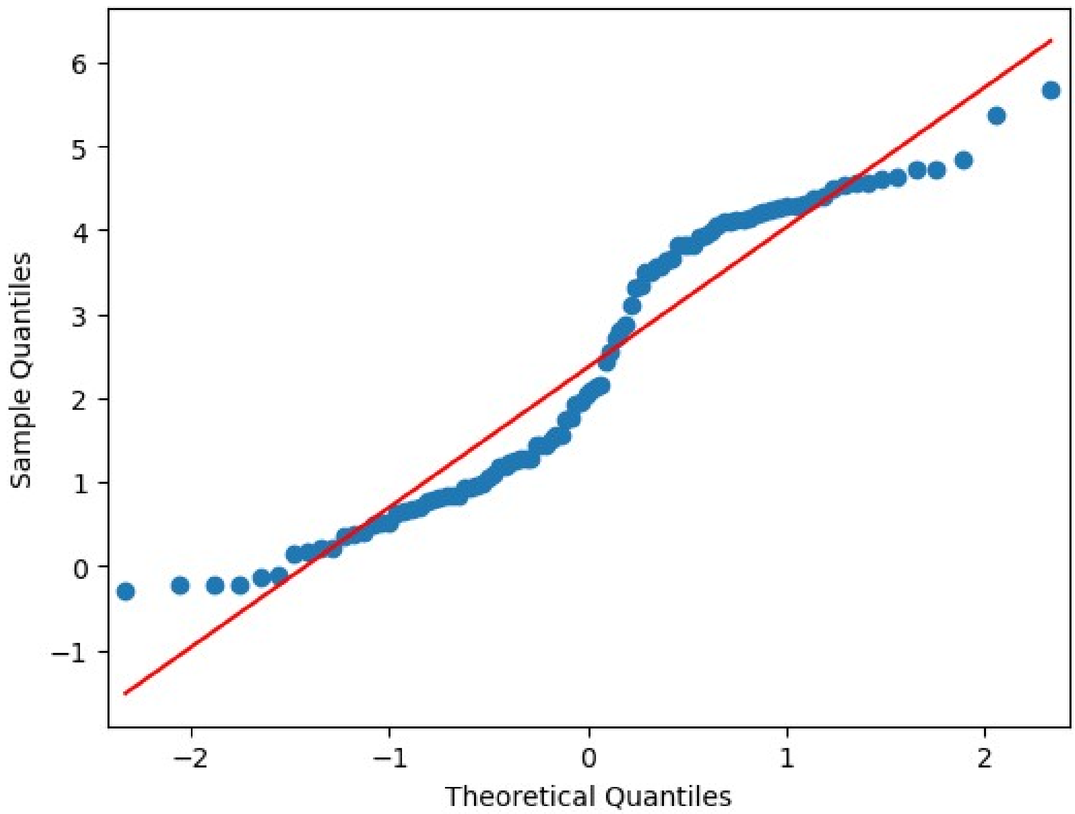
```
ShapiroResult(statistic=0.9183541472339452,
pvalue=1.1523530294281153e-05)
```

```
errors_wls.hist() #This fails when I try manually, however it works
when I try it from the statsmodels library
```

```
<Axes: >
```




```
sm.qqplot(errors_wls, line='s') #This fails when I try manually, however it works when I try it from the statsmodels library
plt.show()
```




```
#trying WLS directly from statsmodels by taking variance as deviation
from the mean

we=(ytrain-ytrain.mean())**2

model_wls=sm.WLS(ytrain, sm.add_constant(xtrain_sc),
weights=1/we).fit()

model_wls.summary()

<class 'statsmodels.iolib.summary.Summary'>
"""
                            WLS Regression Results

======================================================================
========
Dep. Variable:        Chance of admit   R-squared:
0.018
Model:                            WLS   Adj. R-squared:
0.010
Method:                 Least Squares   F-statistic:
2.394
Date:                Sun, 06 Apr 2025   Prob (F-statistic):
0.0680
Time:                        09:55:25   Log-Likelihood:
```

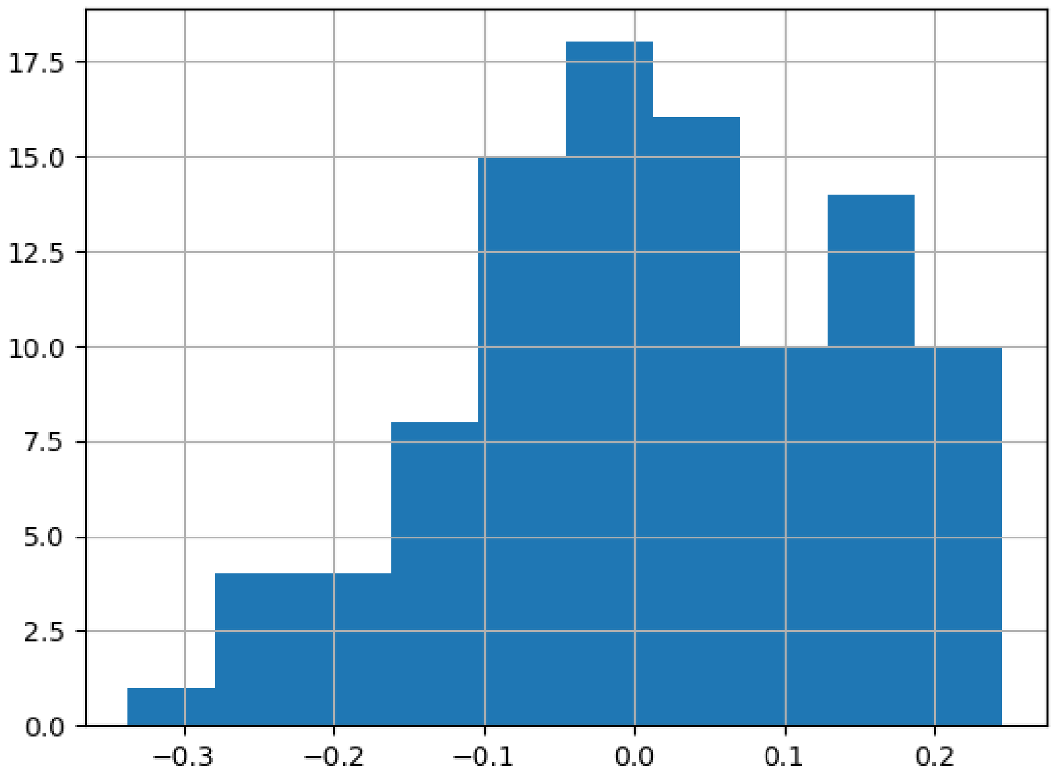
```
482.36
No. Observations:                 400   AIC:
-956.7
Df Residuals:                     396   BIC:
-940.8
Df Model:                           3

Covariance Type:            nonrobust

======================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
--------
const          0.7201      0.001    975.296      0.000       0.719
0.722
x1             0.0010      0.001      0.818      0.414      -0.001
0.004
x2             0.0027      0.001      2.172      0.030       0.000
0.005
x3             0.0004      0.001      0.482      0.630      -0.001
0.002
======================================================================
========
Omnibus:                     1768.898   Durbin-Watson:
2.012
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
64.787
Skew:                          -0.037   Prob(JB):
8.54e-15
Kurtosis:                       1.030   Cond. No.
4.00
======================================================================
========

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

# We see the features in WLS are not significant, the p values are
more than 0.05
#Also we see that the R squared and adjusted R squared are very low,
showing that WLS does not capture the variance properly

ypred_new_wls=model_wls.predict(sm.add_constant(xtest_sc))

print(mean_squared_error(ytest, ypred_new_wls))
print(r2_score(ytest, ypred_new_wls))
```

```
0.016989027957984456
0.017008749181452343
```

```
errors_new_wls=ytest-ypred_new_wls
```

```
errors_new_wls.hist()
```

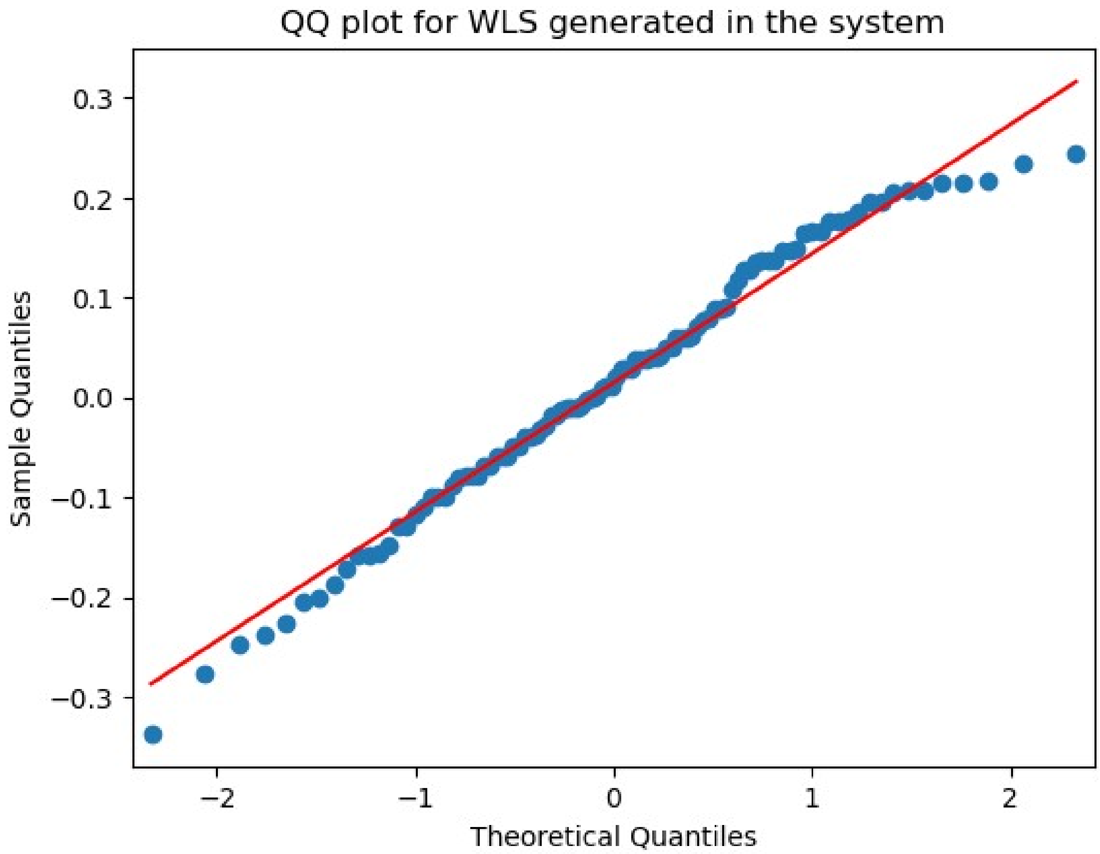
```
<Axes: >
```



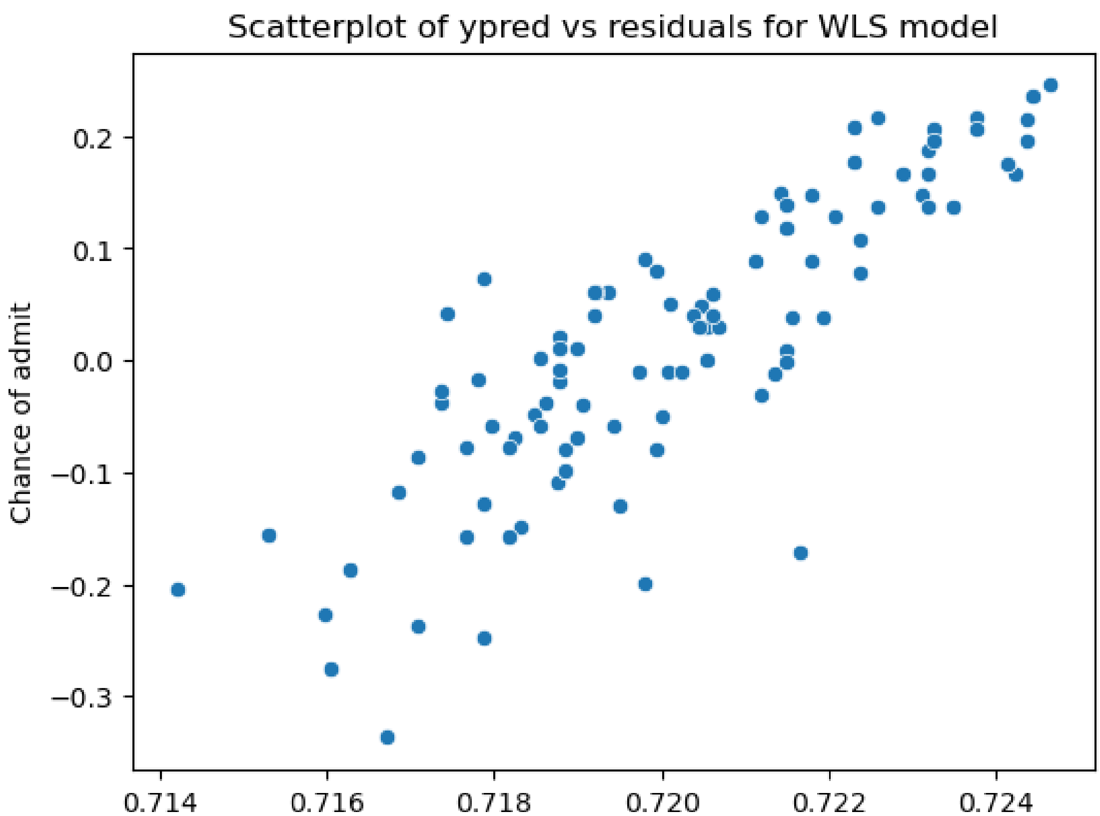

```
shapiro(errors_new_wls)
```

```
ShapiroResult(statistic=0.9802109281527774,
pvalue=0.13805320351181843)
```

```
sm.qqplot(errors_new_wls, line='s')
plt.title('QQ plot for WLS generated in the system')
plt.show() #This shows that WLS from statsmodels corrects the
normality of errors
```




```
sns.scatterplot(x=ypred_new_wls, y=errors_new_wls)
plt.title('Scatterplot of ypred vs residuals for WLS model')
plt.show()
```




```
het_white(model_wls.resid, model_wls.model.exog) #The p value here is
much less than 0.05
#so WLS still fails when the data is not homoskedastic, it does not
correct the problem
```

```
(130.4905311084176,
 2.2384853669395407e-24,
 23.66419530324378,
 1.371997684365566e-29)
```

```
#Conclusions:
# 1. Only 3 factors are super important and significant in getting a
good chance of admission in the weighted order below
#a. TOEFL score
#b. University rating
#c. Research
#The rest of the features are correlated to these and can be derived

# If the aim is to get a 90% chance of admission, then based on the
EDA and model I would suggest the following factors:
# A TOEFL score of 110 out of 120 (or a GRE score of 320 out of 340),
a university rating of 4 or 5, and research
```

```
#2. Strength of SOP or LOR will only be applicable when the university
rating is good, and when the TOEFL/GRE score is good
#3. Having research can significantly boost the chances of admission,
especially given a good SOP
#4. The linear regression model cannot be used directly here, even
though there is a linear relationship between each feature and the
target.
# However, by taking the target variable and transforming raising it
to the power of 2.5, it is possible to use linear regression
# Once we predict, then again we need to take the 2.5th root of the
predcted chance of admission.

# 5. People with good TOEFL/GRE scores will also have a good CGPA,
provided they are from a good university so it becomes more important
to focus on the exam scores,
#the same knowledge will lead to a good cgpa as well
# 6. The strength of the SOP is marginally more important than the
strength of the LOR in getting a good chance of admission

#7. Changing the y value from the original output to y**2.5 helps us
satisfy all the assumptions for linear regression and apply it.
#However the WLS model still does not satisfy the homoskedasticity
assumption even though it can be used for correctijng the normality of
errors.
```