Jamboree Education

Jamboree has helped thousands of students like you make it to top colleges abroad. Be it GMAT, GRE or SAT, their unique problem-solving methods ensure maximum scores with minimum effort. They recently launched a feature where students/learners can come to their website and check their probability of getting into the IVY league college. This feature estimates the chances of graduate admission from an Indian perspective.

EDA

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import pearsonr
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
```

```
df=pd.read_csv('Jamboree_Admission.csv')
```

```
df.head(5)
```

	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

Next steps: Generate code with df View recommended plots

```
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
```

Observations:

- Serial No. is Unique row ID which is not required
- University Rating,SOP,LOR & Research Experience are categorical columns. Need to change into object type.
- Shape of dataset (500,9)

```
df['University Rating']=df['University Rating'].astype('object')
df['SOP']=df['SOP'].astype('object')
```

```
df['LOR ']=df['LOR '].astype('object')
df['Research']=df['Research'].astype('object')
```

```
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    object 
 4   SOP                500 non-null    object 
 5   LOR                500 non-null    object 
 6   CGPA               500 non-null    float64
 7   Research           500 non-null    object 
 8   Chance of Admit    500 non-null    float64
```

```
dtypes: float64(2), int64(3), object(4)
memory usage: 35.3+ KB
```

```
df=df.drop(['Serial No.'],axis=1)
df.rename(columns={'LOR ':'LOR'},inplace=True)
```

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
GRE Score	500.0	316.47200	11.295148	290.00	308.0000	317.00	325.00	340.00
TOEFL Score	500.0	107.19200	6.081868	92.00	103.0000	107.00	112.00	120.00
CGPA	500.0	8.57644	0.604813	6.80	8.1275	8.56	9.04	9.92
Chance of Admit	500.0	0.72174	0.141140	0.34	0.6300	0.72	0.82	0.97

Obsevations:

- I have observed the mean GRE Score among students is 316,indicating high level of academic aptitude. Similarly, the TOEFL Score is 107, reflecting strong English language proficiency.
- In terms of university ratings, I found that the maximum rating is 5, suggesting that top-tier universities are also part of the dataset.On the other hand, the minimum rating is 1,indicating a diverse range of university rankings in the dataset.
- Interestingly, the average university rating for students who got placed is ,indicating students from a variety of university backgrounds have been successful in gaining admission.
- Furthermore, the average CGPA score of 8.57 indicates a high level of academic achievement among the students in the dataset.
- Lastly, the average Chance of Admit is 0.72,suggesting that on average students in the dataset have a relatively high likelihood of being admitted to their desired graduate programs

```
df.describe(include=object).T
```

	count	unique	top	freq
University Rating	500.0	5.0	3.0	162.0
SOP	500.0	9.0	4.0	89.0
LOR	500.0	9.0	3.0	99.0
Research	500.0	2.0	1.0	280.0

Observation:

- GRE score has range between (290,340)
- TOEFL score has range between (92,120)
- CGPA has range between (6.8,9.92)
- Chance to admit has range between (0.34,0.97)
- University Rating have 5 sub-categories
- SOP and LOR have 9 ratings sub-category
- Research have binary sub-category

```
#Numerical columns
num_cols=list(df.select_dtypes(include='number').columns)
num_cols
```

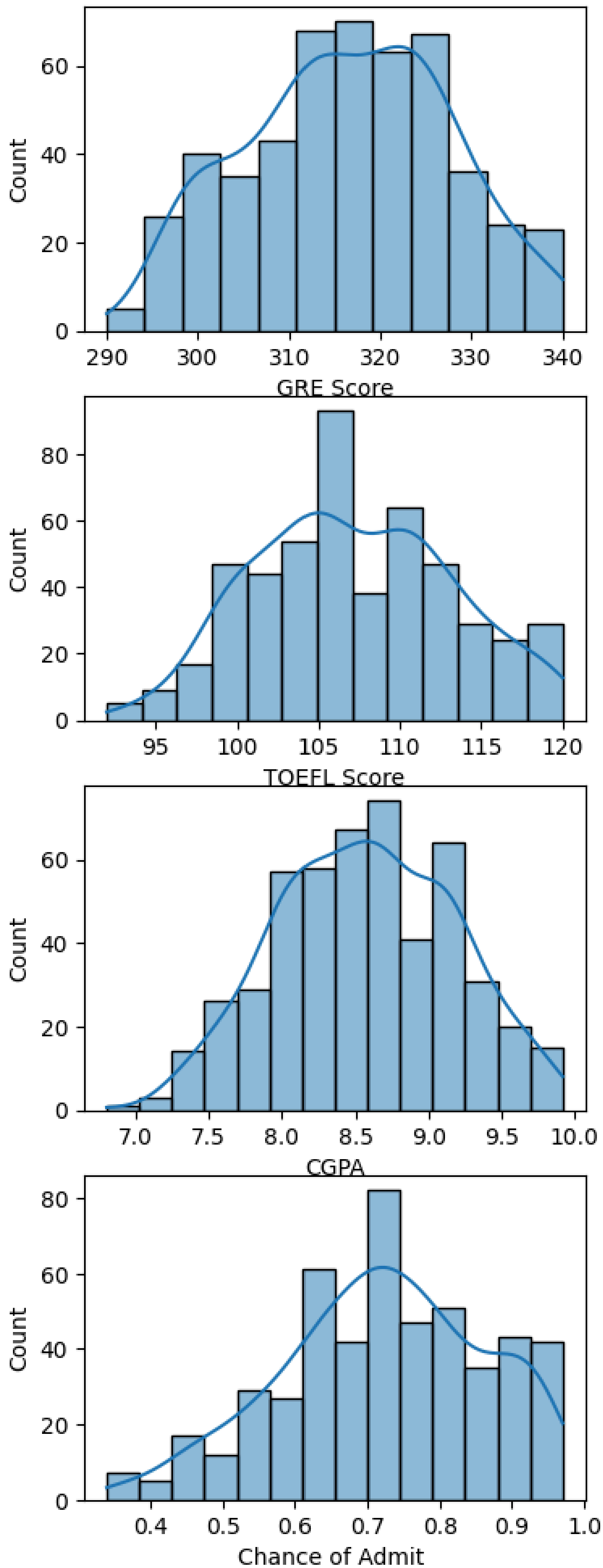
```
['GRE Score', 'TOEFL Score', 'CGPA', 'Chance of Admit ']
```

```
#Categorical columns
cat_cols=list(df.select_dtypes(include='object').columns)
cat_cols
```

```
['University Rating', 'SOP', 'LOR', 'Research']
```

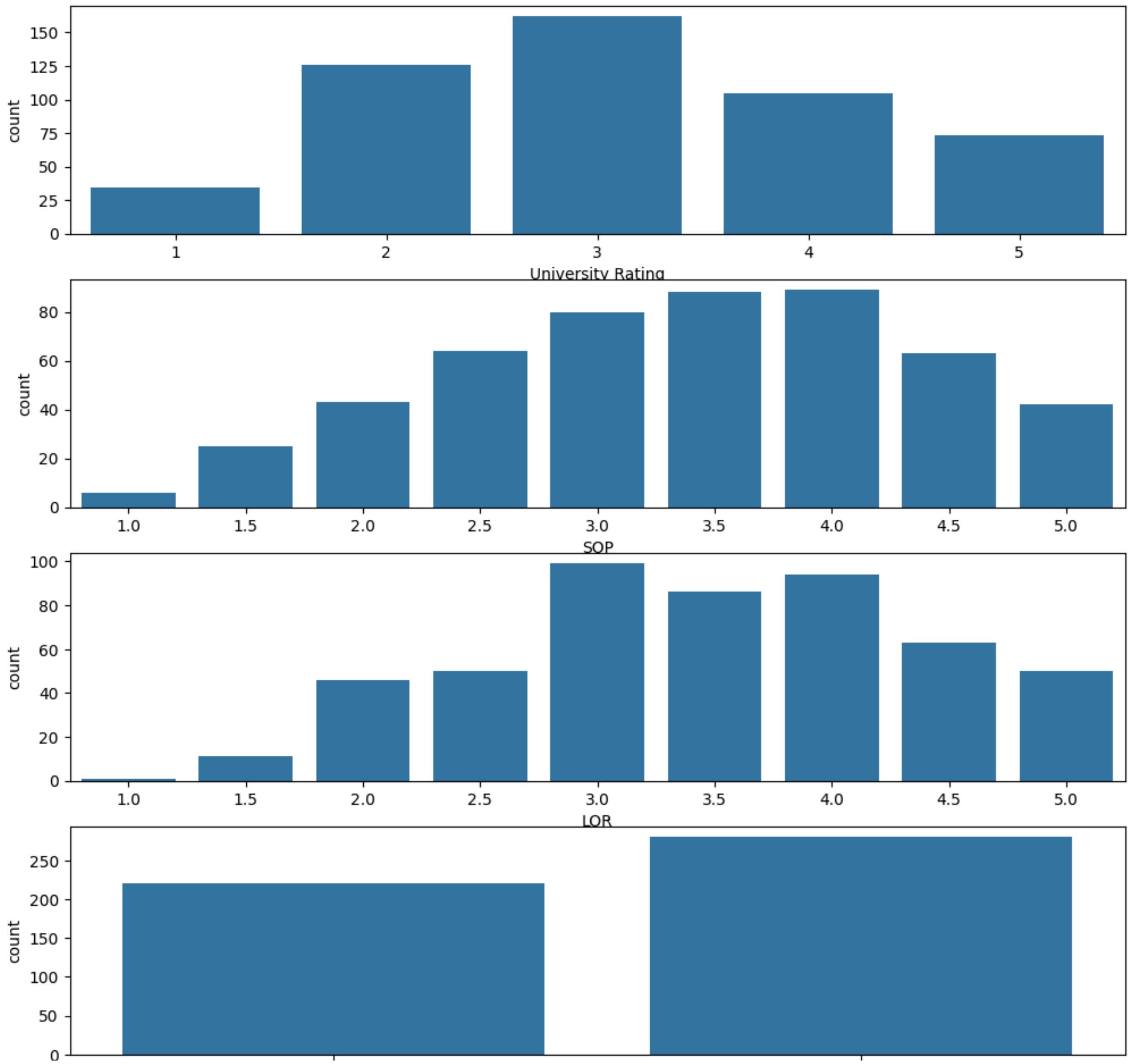
Univariate Analysis

```
fig,axs=plt.subplots(nrows=4,ncols=1,squeeze=False,figsize=(4,12))
for i in range(len(num_cols)):
  sns.histplot(df,x=df[num_cols[i]],kde=True,ax=axs[i][0])
```



Count
0
20
40
60
290
300
310
320
330
340
GRE Score
Count
0
20
40
60
80
95
100
105
110
115
120
TOEFL Score
Count
0
20
40
60
7.0
7.5
8.0
8.5
9.0
9.5
10.0
CGPA
Count
0
20
40
60
80
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Chance of Admit

```
fig,axs=plt.subplots(nrows=4,ncols=1,squeeze=False,figsize=(12,12))
for i in range(len(cat_cols)):
  sns.countplot(df,x=df[cat_cols[i]],ax=axs[i][0])
```



Observations:

- About 50% of students scored between 300 and 330 in GRE, indicating a diverse range of scores within the dataset.
- Approximately 80% of the students have scored between 100 and 115 in TOEFL highlighting a strong overall performance in English proficiency
- The graph depicts that the maximum number of students got placed in universities ranked 2 and 4 suggesting that these universities are popular choices among the students in the

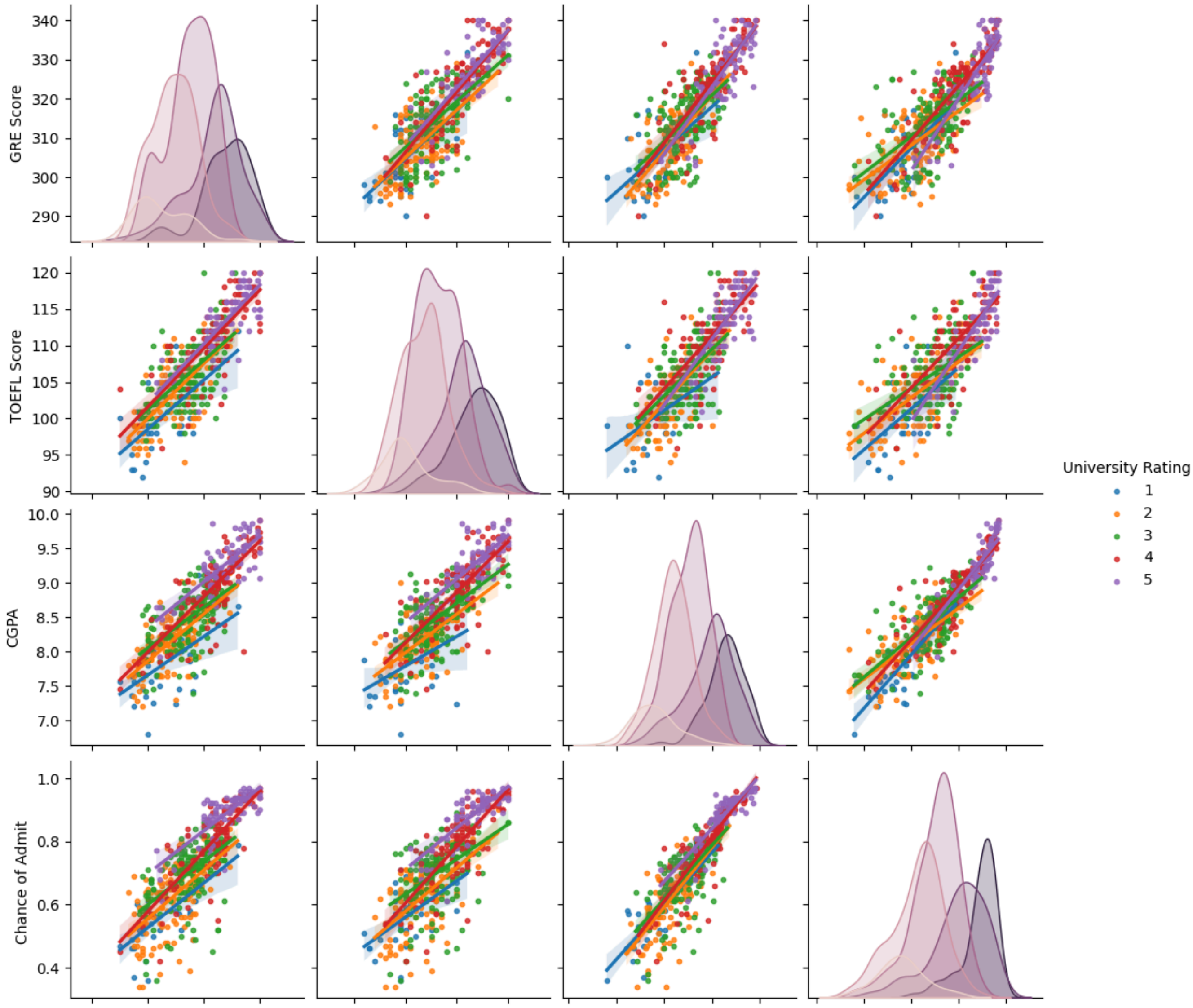
dataset.

- The Stetement of Purpose (SOP) graph shows that more than 50% of students have submitted SOPs with rating higher than 3 indicating a generally high quality of SOPs.
- Similarly more than 50% of students have received strong recommendations(rating 3 to 5) in their Letters of Recommendation (LORs).
- The CGPA graph shows a significant portion of students scoring between 7.5 and 9.0 indicating a high level of academic achievement.
- More students have conducted or published research papers compared to those who haven't suggesting a strong research-oriented background among the students.
- The chances of getting admission to IVY league universities are between 70% to 80% after meeting the qualifying requirements , indicating a promising opportunity for the students in the dataset.

Bi-variate

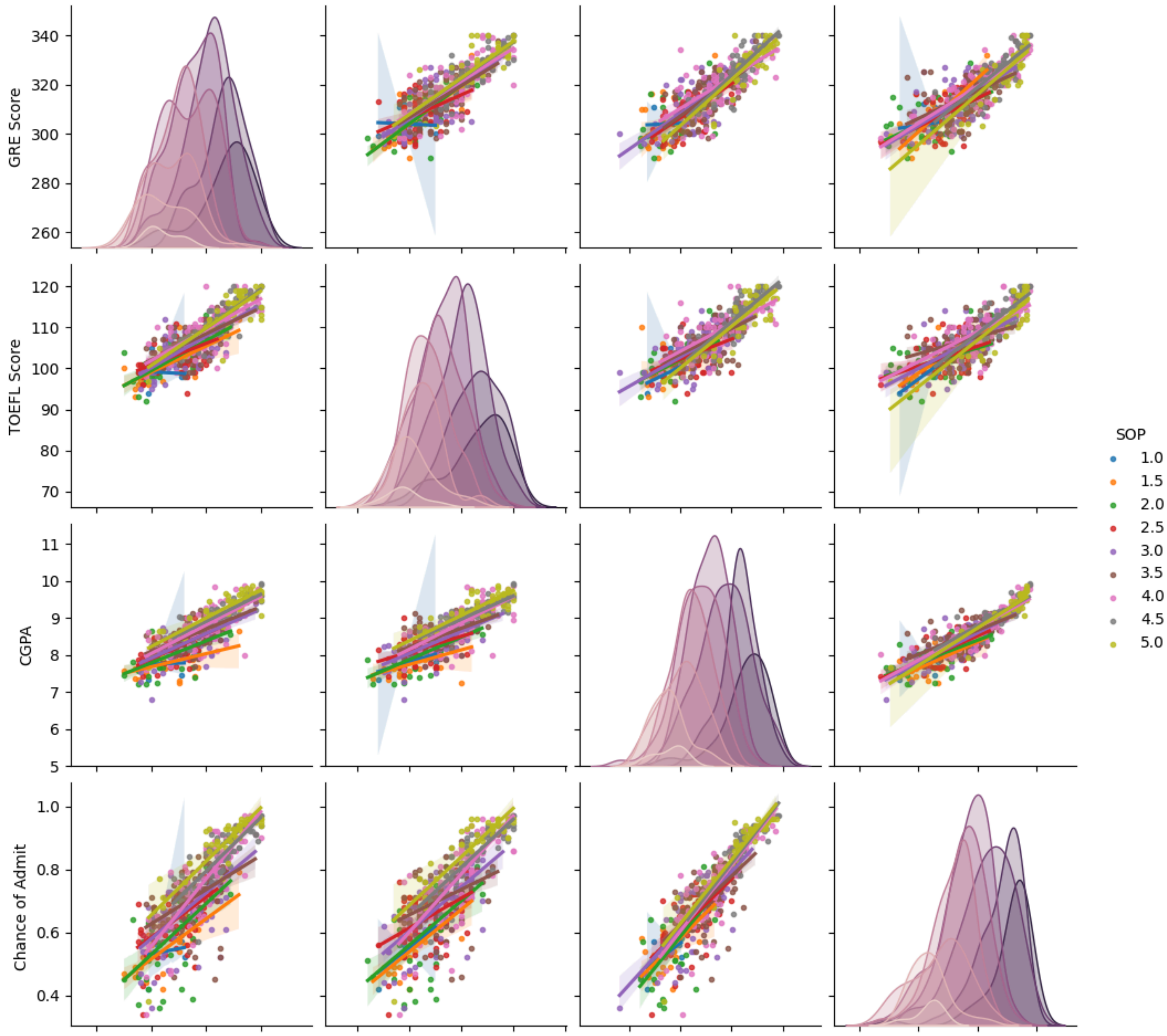
```
sns.pairplot(data = df,vars= num_cols,hue='University Rating',kind = 'reg',markers = '.')
plt.plot()
```

[]



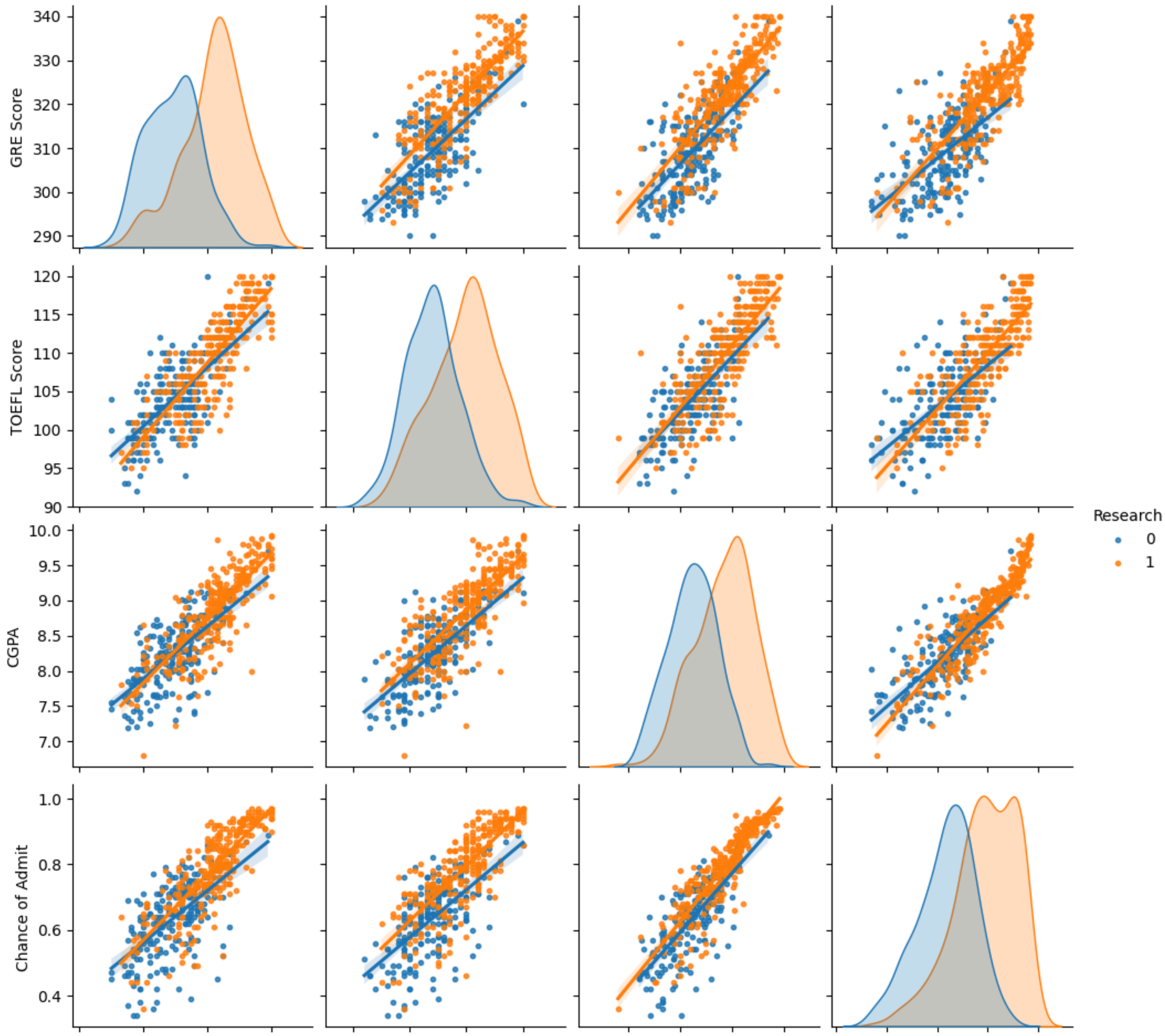

```
sns.pairplot(data = df,vars= num_cols,hue='SOP',kind = 'reg',markers = '.')
plt.plot()
```

[]



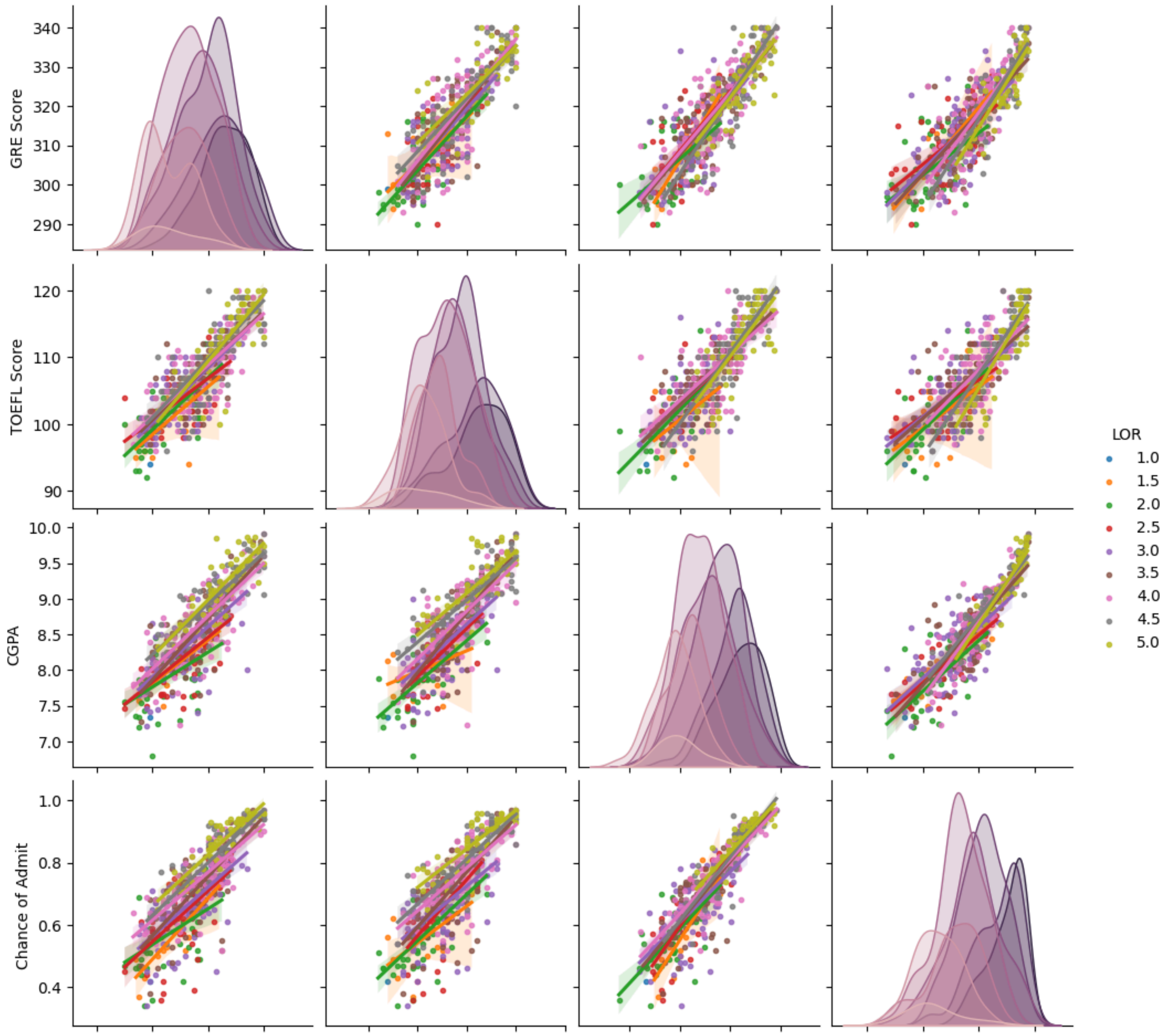

```
sns.pairplot(data = df,vars= num_cols,hue='Research',kind = 'reg',markers = '.')
plt.plot()
```

[]

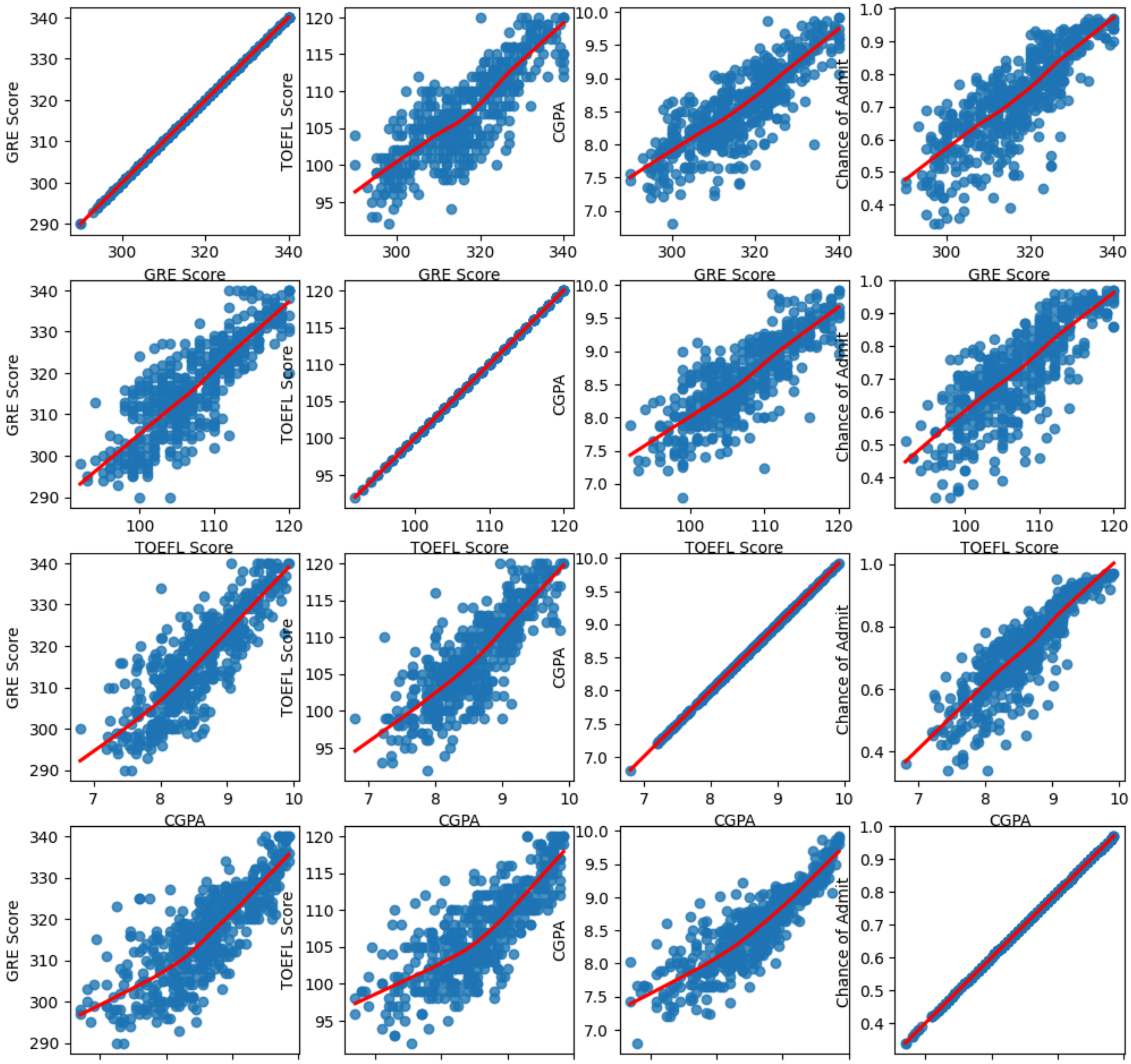



```
sns.pairplot(data = df,vars= num_cols,hue='LOR',kind = 'reg',markers = '.')
plt.plot()
```

[]




```
fig,axs=plt.subplots(nrows=4,ncols=4,squeeze=False,figsize=(12,12))
for i in range(len(num_cols)):
  for j in range(len(num_cols)):
    sns.regplot(df,x=df[num_cols[i]],y=df[num_cols[j]],lowess=True, line_kws=dict(color="r")
```



Observations:

- All the numerical_columns (CGPA,TOEFL and GRE) have somewhat linearly distributed with depended variable(Chance of Admit)

Data Preprocessing

```
df[df.duplicated()].sum()
```

```
GRE Score              0
TOEFL Score            0
University Rating      0
SOP                    0
LOR                    0
CGPA                 0.0
```

```
Research                0
Chance of Admit       0.0
dtype: object
```

Observations:

- It's been observed that there is no duplicates in the dataset

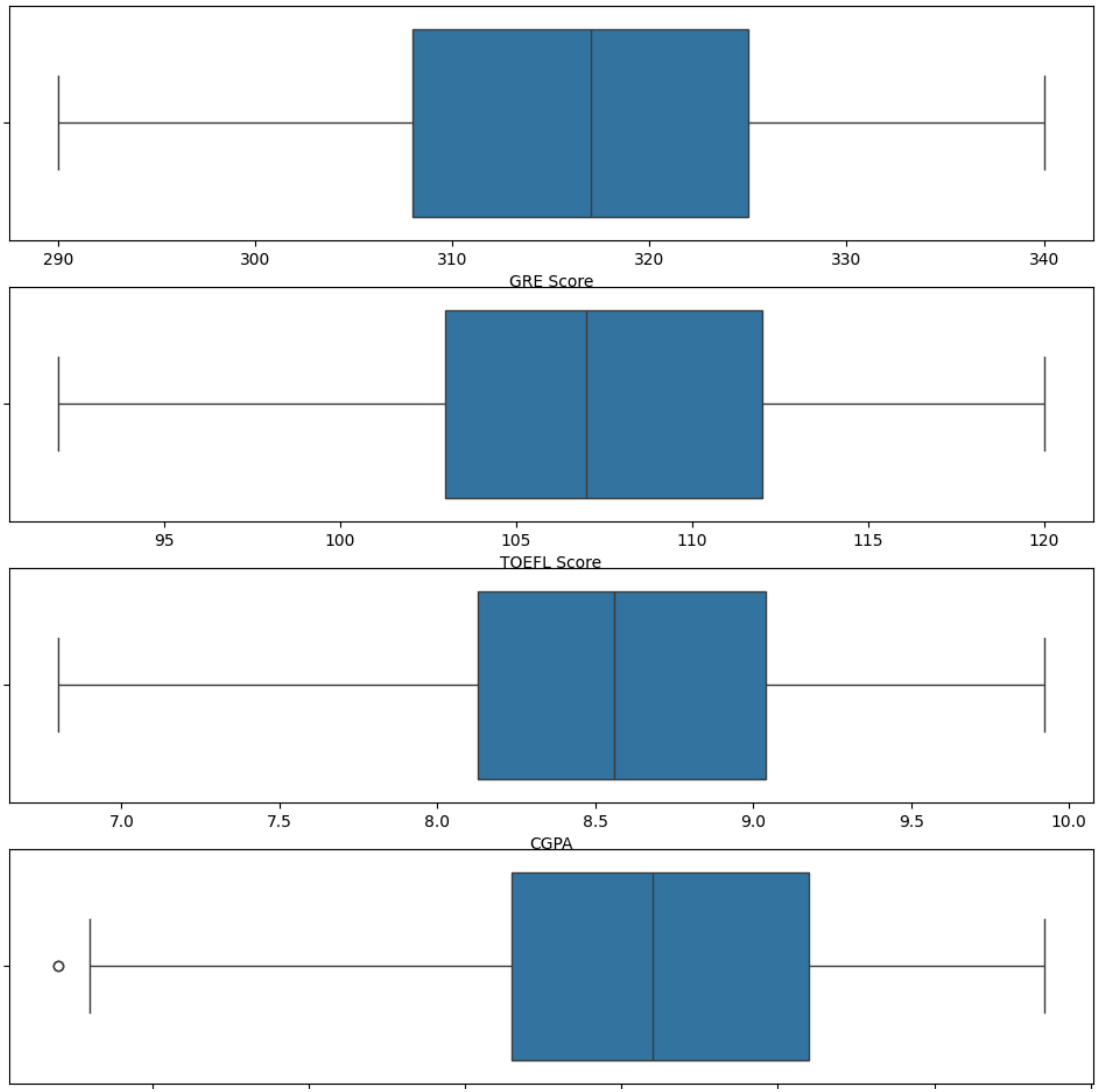
```
df.isnull().sum()
```

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

Observations:

- Dataset don't have any null values

```
fig,axs=plt.subplots(nrows=4,ncols=1,squeeze=False,figsize=(12,12))
for i in range(len(num_cols)):
  sns.boxplot(df,x=df[num_cols[i]],ax=axs[i][0])
```



Onservations:

- Visualizations are pretty tied up on both end for all numerical columns and with approximatly no outlier

Correlation

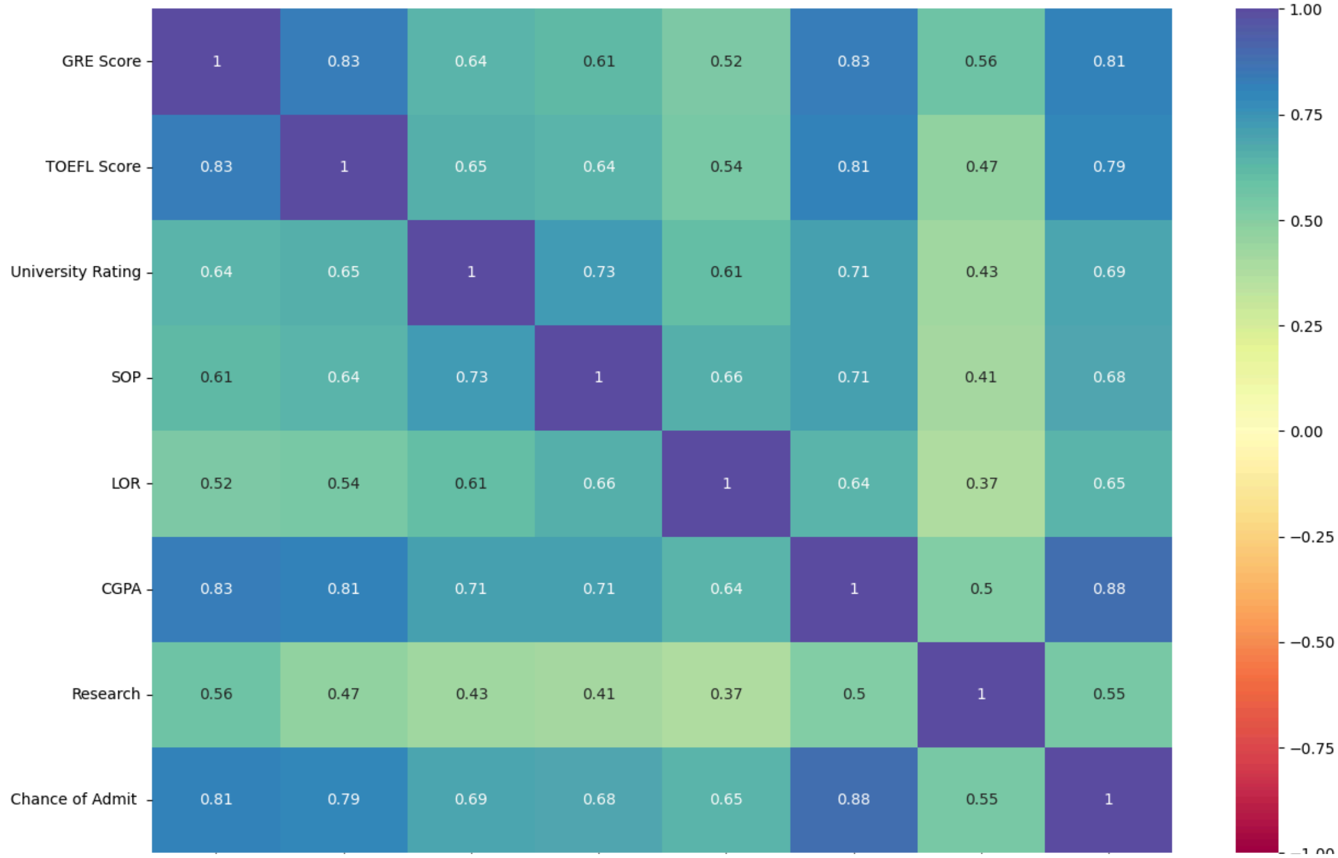
```
df_corr=df.corr()
df_corr
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
GRE Score	1.000000	0.827200	0.635376	0.613498	0.524679	0.825878	0.563398	0.810351
TOEFL Score	0.827200	1.000000	0.649799	0.644410	0.541563	0.810574	0.467012	0.792228
University Rating	0.635376	0.649799	1.000000	0.728024	0.608651	0.705254	0.427047	0.690132
SOP	0.613498	0.644410	0.728024	1.000000	0.663707	0.712154	0.408116	0.684137
LOR	0.524679	0.541563	0.608651	0.663707	1.000000	0.637469	0.372526	0.645365
CGPA	0.825878	0.810574	0.705254	0.712154	0.637469	1.000000	0.501311	0.882413
Research	0.563398	0.467012	0.427047	0.408116	0.372526	0.501311	1.000000	0.545871
Chance of Admit	0.810351	0.792228	0.690132	0.684137	0.645365	0.882413	0.545871	1.000000

Next steps: Generate code with `df_corr` View recommended plots

```
plt.figure(figsize = (15, 10))
sns.heatmap(data = df_corr, vmin = -1, vmax = 1, annot = True, cmap=sns.color_palette("Spect
plt.plot()
```

[]



Observations:

- GRE score is strong with 0.81.
- Toefl score is strong with 0.79.
- SOP,University Rating and LOR is moderate and similar rating between 0.68 to 0.69.
- CGPA is the strongest of all with 0.88.
- Research is weak at 0.55
- All correlation <90 .No need to drop any feature

Target-hot encoding

```
encoded_categorical_df=pd.DataFrame()
for i in range(len(cat_cols)):
  encoded_categorical_df[cat_cols[i]]=df.groupby(cat_cols[i])['Chance of Admit '].transform(
```

```
encoded_categorical_df
```

	University Rating	SOP	LOR	Research
0	0.801619	0.850000	0.831905	0.789964
1	0.801619	0.782809	0.831905	0.789964
2	0.702901	0.678500	0.723023	0.789964
3	0.702901	0.712045	0.640600	0.789964
4	0.626111	0.589535	0.668485	0.634909
...	...	...	...	...
495	0.888082	0.850000	0.764149	0.789964
496	0.888082	0.885000	0.872600	0.789964
497	0.888082	0.850000	0.872600	0.789964
498	0.801619	0.782809	0.872600	0.634909
499	0.801619	0.850000	0.831905	0.634909

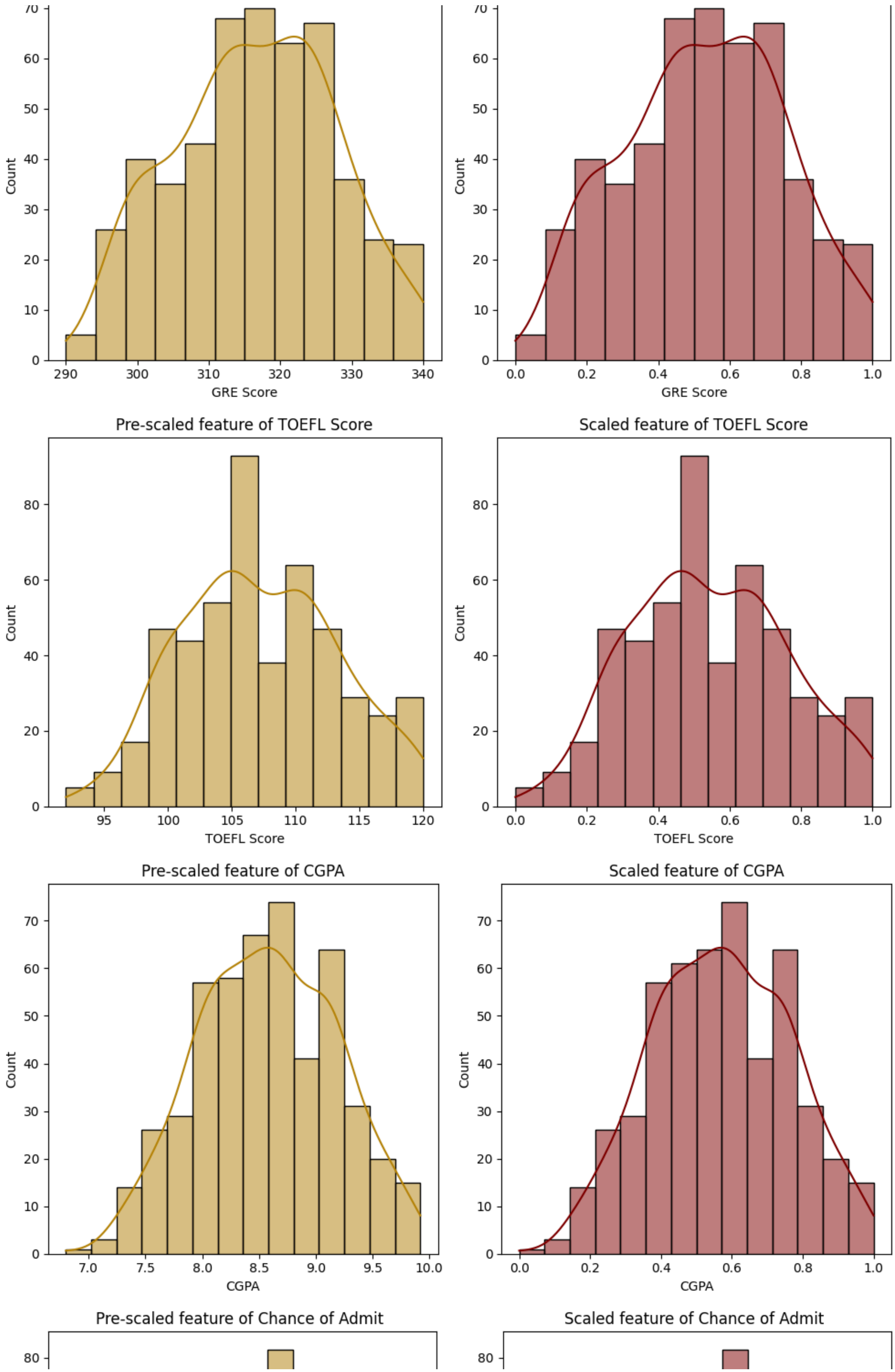
500 rows × 4 columns

Next steps: Generate code with `encoded_categorical_df` View recommended plots

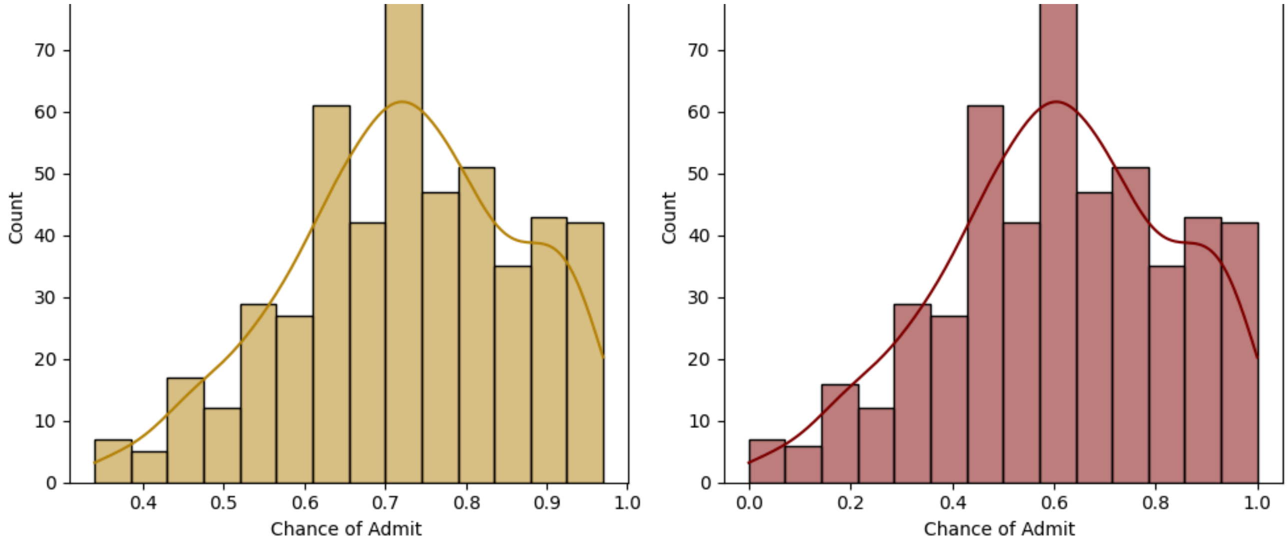
Normalize/Standardize the numerical features

```
# Normalizing/Standardizing the numerical features using MinMaxScaler
min_max_scaler = MinMaxScaler()
min_max_scaled_numerical = min_max_scaler.fit_transform(df[num_cols])
# Converting the scaled features back to a dataframe
min_max_scaled_numerical_df = pd.DataFrame(min_max_scaled_numerical, columns=num_cols)
```

```
for i in num_cols:
  fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
  axes[0].set_title(f"Pre-scaled feature of {i}")
  sns.histplot(df[i], ax = axes[0], kde=True,color='darkgoldenrod')
  axes[1].set_title(f"Scaled feature of {i}")
  sns.histplot(min_max_scaled_numerical_df[i], ax= axes[1], kde=True,color='maroon')
  plt.tight_layout()
  plt.show()
```



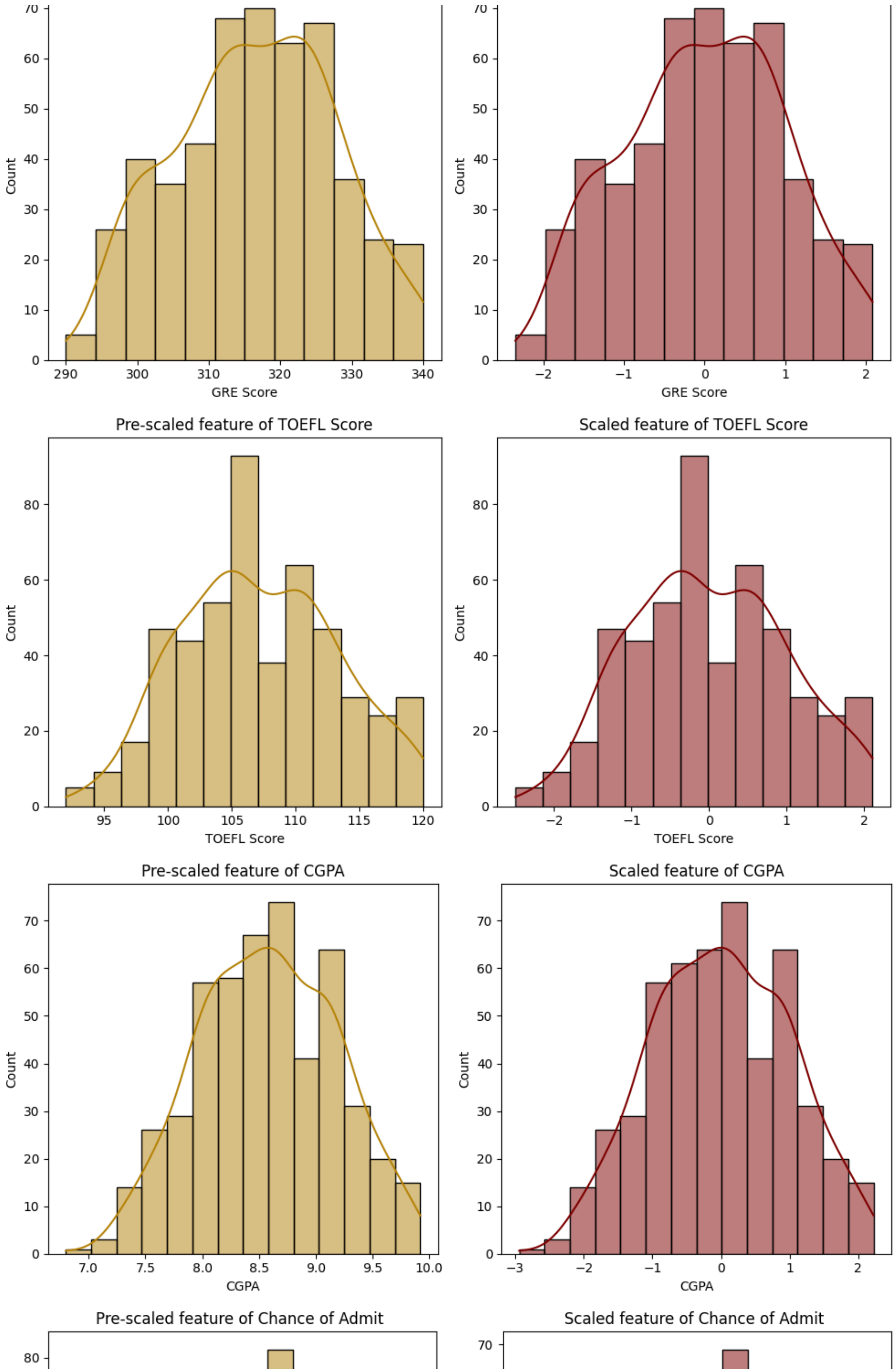
Count
GRE Score
Pre-scaled feature of TOEFL Score
Scaled feature of TOEFL Score
TOEFL Score
Pre-scaled feature of CGPA
Scaled feature of CGPA
CGPA
Pre-scaled feature of Chance of Admit
Scaled feature of Chance of Admit

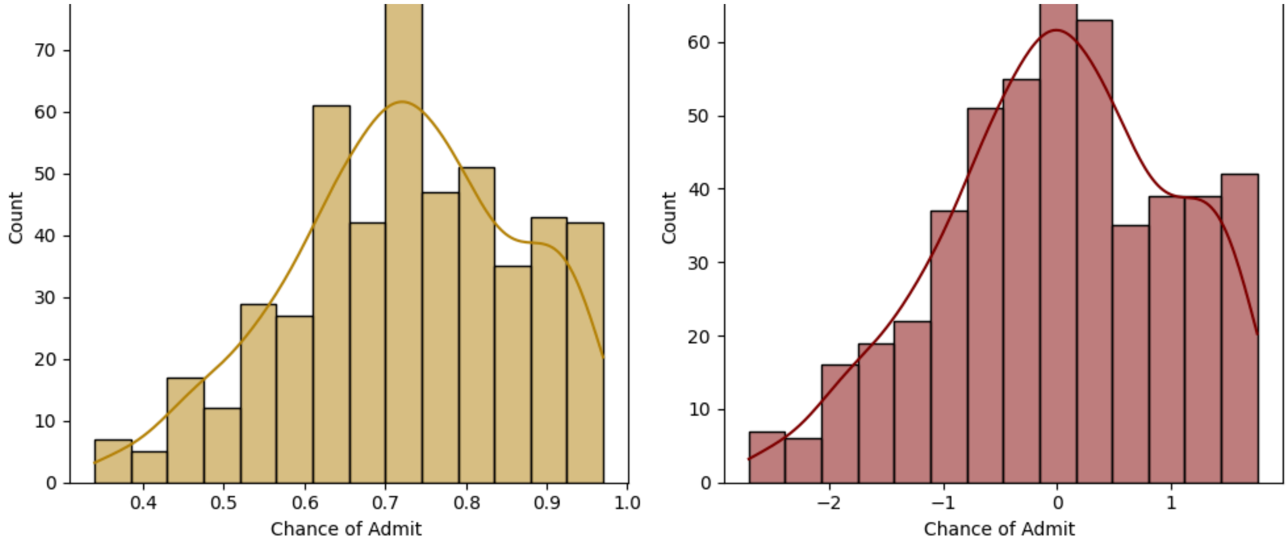



```
# Standardizing the numerical features using StandardScaler
std_scaler = StandardScaler()
std_scaled_numerical = std_scaler.fit_transform(df[num_cols])
# Converting the scaled features back to a dataframe
std_scaled_numerical_df = pd.DataFrame(std_scaled_numerical, columns=num_cols)
std_scaled_numerical_df.shape
```

```
(500, 4)
```

```
for i in num_cols:
  fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
  axes[0].set_title(f"Pre-scaled feature of {i}")
  sns.histplot(df[i], ax = axes[0], kde=True,color='darkgoldenrod')
  axes[1].set_title(f"Scaled feature of {i}")
  sns.histplot(std_scaled_numerical_df[i], ax= axes[1], kde=True,color='maroon')
  plt.tight_layout()
  plt.show()
```


GRE Score
Count
Pre-scaled feature of TOEFL Score
Scaled feature of TOEFL Score
TOEFL Score
Pre-scaled feature of CGPA
Scaled feature of CGPA
CGPA
Pre-scaled feature of Chance of Admit
Scaled feature of Chance of Admit




```
# Combining the encoded and scaled features with the rest of the dataset
processed_data = pd.concat([df.drop(cat_cols + num_cols, axis=1),encoded_categori
processed_data
```

	University Rating	SOP	LOR	Research	GRE Score	TOEFL Score	CGPA	Chance of Admit
0	0.801619	0.850000	0.831905	0.789964	0.94	0.928571	0.913462	0.920635
1	0.801619	0.782809	0.831905	0.789964	0.68	0.535714	0.663462	0.666667
2	0.702901	0.678500	0.723023	0.789964	0.52	0.428571	0.384615	0.603175
3	0.702901	0.712045	0.640600	0.789964	0.64	0.642857	0.599359	0.730159
4	0.626111	0.589535	0.668485	0.634909	0.48	0.392857	0.451923	0.492063
...	...	...	...	...	...	...	...	...
495	0.888082	0.850000	0.764149	0.789964	0.84	0.571429	0.711538	0.841270
496	0.888082	0.885000	0.872600	0.789964	0.94	0.892857	0.983974	0.984127
497	0.888082	0.850000	0.872600	0.789964	0.80	1.000000	0.884615	0.936508
498	0.801619	0.782809	0.872600	0.634909	0.44	0.392857	0.522436	0.619048
499	0.801619	0.850000	0.831905	0.634909	0.74	0.750000	0.717949	0.793651

500 rows × 8 columns

Next steps: Generate code with processed_data | View recommended plots

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

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=1)
```

```
x_test.shape ,y_test.shape
```

```
((150, 7), (150,))
```

```
model= LinearRegression()
```

```
model.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

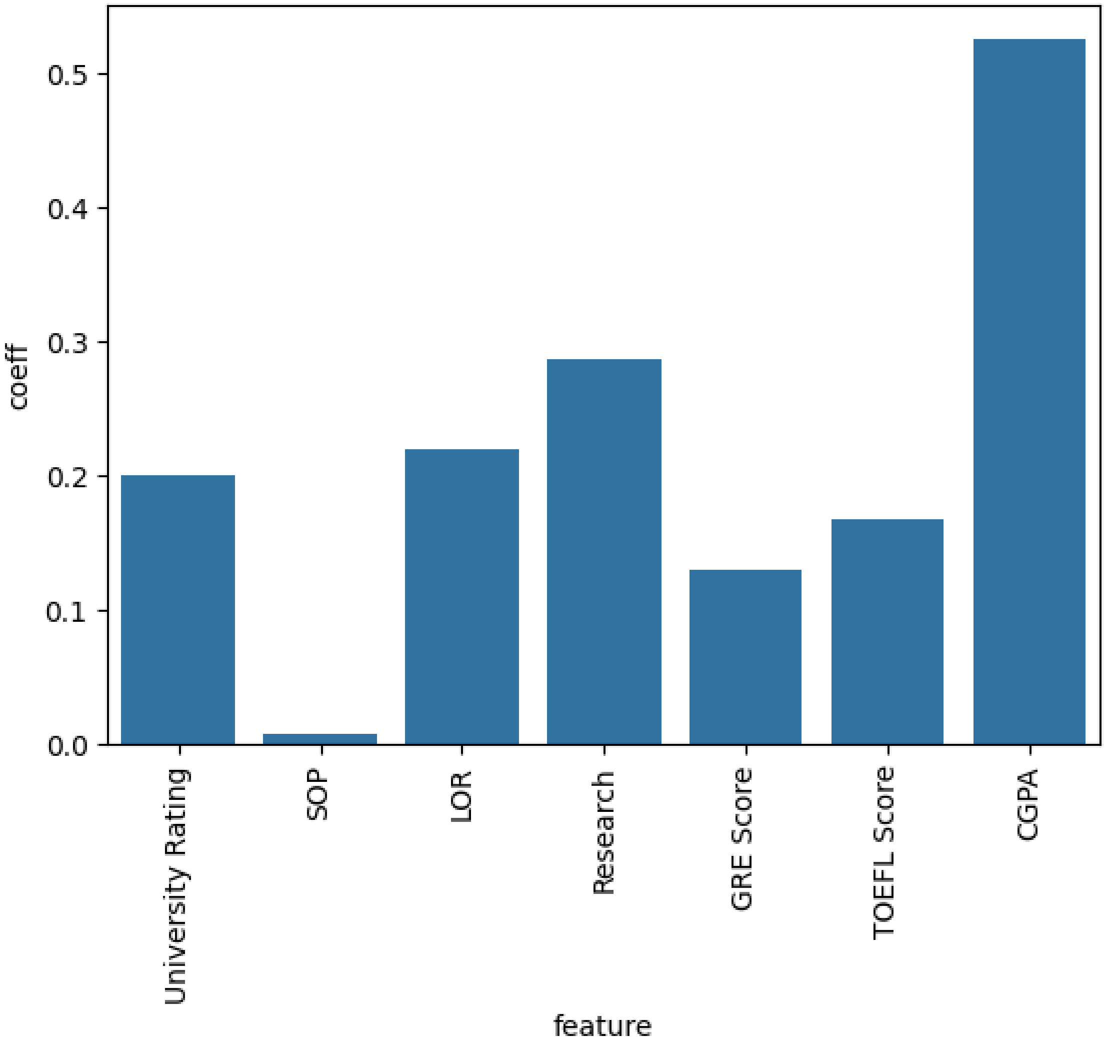
```
y_pred=model.predict(x_test)
r2=r2_score(y_test,y_pred)
print('R2 Score:',r2)
```

```
R2 Score: 0.8144122158502536
```

```
result=[]
for i in zip(x_train,model.coef_):
  result.append(list(i))
result
```

```
[['University Rating', 0.19980576777474177],
 ['SOP', -0.008424244134449421],
 ['LOR', 0.22057821272847267],
 ['Research', 0.2869800179728044],
 ['GRE Score', 0.13029944559494716],
 ['TOEFL Score', 0.16699190427274085],
 ['CGPA', 0.5253977458359129]]
```

```
imp=pd.DataFrame(list(zip(x_train.columns,np.abs(model.coef_))),columns=['feature','coeff'])
sns.barplot(x='feature',y='coeff',data=imp)
plt.xticks(rotation=90)
plt.show()
```



Observations:

- SOP have very low coefficient

```
y_train1=np.array(y_train)
x_sm=sm.add_constant(x_train)
model1 = sm.OLS(y_train1,x_sm)
results = model1.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.822
Model:                            OLS   Adj. R-squared:                  0.818
Method:                 Least Squares   F-statistic:                     225.6
Date:                Fri, 10 May 2024   Prob (F-statistic):          4.96e-124
Time:                        13:34:37   Log-Likelihood:                 339.45
No. Observations:                 350   AIC:                            -662.9
Df Residuals:                     342   BIC:                            -632.0
Df Model:                           7
Covariance Type:            nonrobust
```

```
==============================================================================
                         coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const                 -0.3522      0.068     -5.172      0.000      -0.486      -0.218
University Rating      0.1998      0.089      2.240      0.026       0.024       0.375
SOP                   -0.0084      0.087     -0.097      0.923      -0.179       0.162
LOR                    0.2206      0.076      2.891      0.004       0.070       0.371
Research               0.2870      0.080      3.607      0.000       0.130       0.443
GRE Score              0.1303      0.047      2.788      0.006       0.038       0.222
TOEFL Score            0.1670      0.046      3.609      0.000       0.076       0.258
CGPA                   0.5254      0.057      9.294      0.000       0.414       0.637
==============================================================================
Omnibus:                       76.444   Durbin-Watson:                   1.973
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              174.792
Skew:                          -1.087   Prob(JB):                     1.11e-38
Kurtosis:                       5.694   Cond. No.                         43.1
==============================================================================

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

- p>0.05 for SOP

```
x_train=x_train.drop('SOP',axis=1)
```

```
model.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Assumption of LR

Multicollinearity by VIF

```
def calculate_vif(dataset):
  vif_data = pd.DataFrame()
  vif_data['feature'] = dataset.columns
  vif_data['VIF'] = [variance_inflation_factor(dataset.values, i) for i in range(len(dataset
  return(vif_data)
```

```
calculate_vif(x_train)
```

	feature	VIF
0	University Rating	124.138653
1	LOR	95.494240
2	Research	72.815134
3	GRE Score	27.734039
4	TOEFL Score	29.345973
5	CGPA	42.677984

- University Rating is having high VIF value So we will remove University Rating

```
x_train=x_train.drop('University Rating',axis=1)
calculate_vif(x_train)
```

	feature	VIF
0	LOR	64.547915
1	Research	56.048493
2	GRE Score	27.734008
3	TOEFL Score	28.817066
4	CGPA	42.041572

```
processed_data.corr()
```

	University Rating	SOP	LOR	Research	GRE Score	TOEFL Score	CGPA	Chance of Admit
University Rating	1.000000	0.732390	0.612635	0.430072	0.638141	0.651456	0.705206	0.692469
SOP	0.732390	1.000000	0.661703	0.403590	0.620228	0.648646	0.716361	0.690643
LOR	0.612635	0.661703	1.000000	0.371336	0.528283	0.544646	0.641918	0.648522
Research	0.430072	0.403590	0.371336	1.000000	0.563398	0.467012	0.501311	0.545871
GRE Score	0.638141	0.620228	0.528283	0.563398	1.000000	0.827200	0.825878	0.810351
TOEFL Score	0.651456	0.648646	0.544646	0.467012	0.827200	1.000000	0.810574	0.792228
CGPA	0.705206	0.716361	0.641918	0.501311	0.825878	0.810574	1.000000	0.882413
Chance of Admit	0.692469	0.690643	0.648522	0.545871	0.810351	0.792228	0.882413	1.000000

- Research and LOR have high VIF's thus lets try to create new variable as a product of both

```
x_train['Research and LOR']=x_train['Research']*x_train['LOR']
```

```
x_train=x_train.drop('Research',axis=1)
calculate_vif(x_train)
```

	feature	VIF
0	LOR	102.072701
1	GRE Score	28.795447
2	TOEFL Score	28.740302
3	CGPA	43.090778
4	Research and LOR	132.448739

```
x_train=x_train.drop('LOR',axis=1)
calculate_vif(x_train)
```

	feature	VIF
0	GRE Score	26.418181
1	TOEFL Score	28.576679
2	CGPA	42.808489
3	Research and LOR	19.844840

```
x_train=x_train.drop('CGPA',axis=1)
calculate_vif(x_train)
```

	feature	VIF
0	GRE Score	22.984004
1	TOEFL Score	24.570307
2	Research and LOR	12.557738

```
x_train['TOEFL and GRE']=x_train['TOEFL Score']*x_train['GRE Score']
```

```
x_train=x_train.drop('TOEFL Score',axis=1)
calculate_vif(x_train)
```

	feature	VIF
0	GRE Score	47.339454
1	Research and LOR	14.955630
2	TOEFL and GRE	18.047546

```
x_train=x_train.drop('GRE Score',axis=1)
calculate_vif(x_train)
```

	feature	VIF
0	Research and LOR	4.052644
1	TOEFL and GRE	4.052644

- both the VIF's are below 5 now

```
x_train_sm=sm.add_constant(x_train)
```

```
model.fit(x_train_sm,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

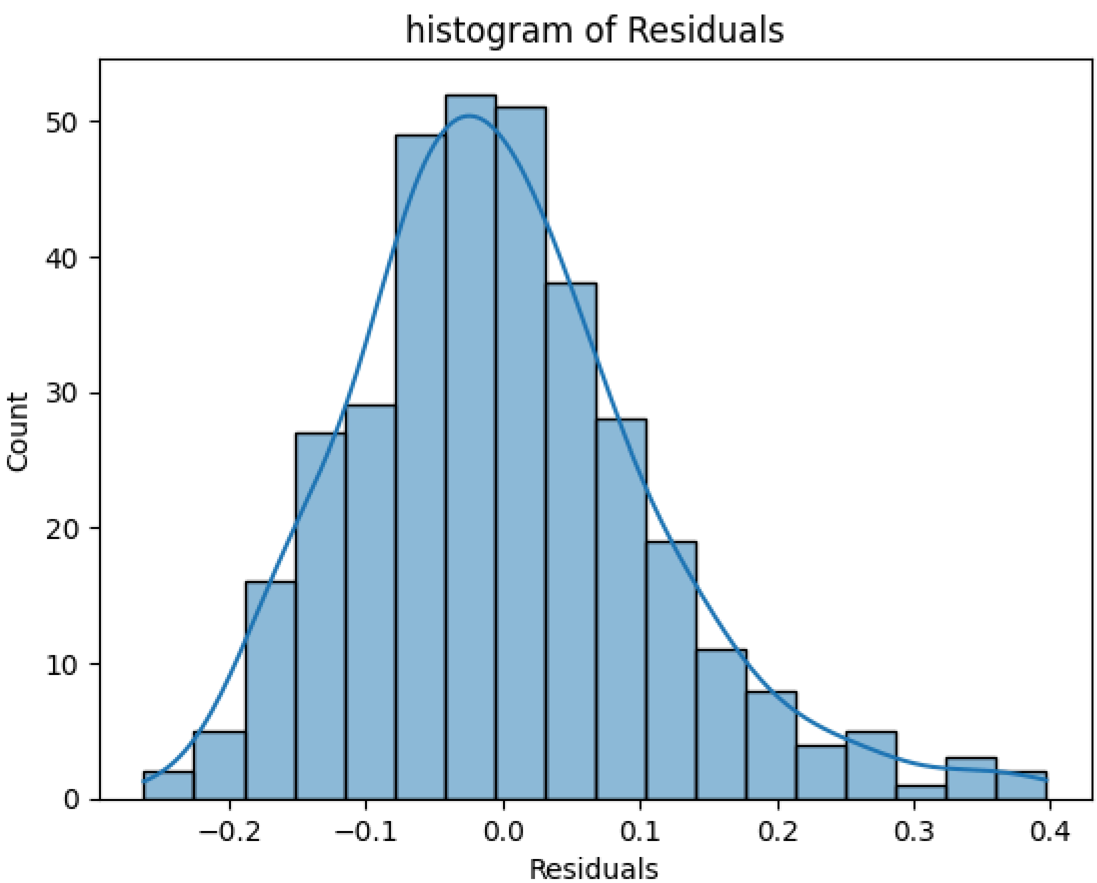
Mean of residuals

```
y_hat_residuals=model.predict(x_train_sm)
```

```
errors=y_hat_residuals-y_train
```

```
sns.histplot(errors,kde=True)
plt.xlabel("Residuals")
plt.title("histogram of Residuals")
```

```
Text(0.5, 1.0, 'histogram of Residuals')
```



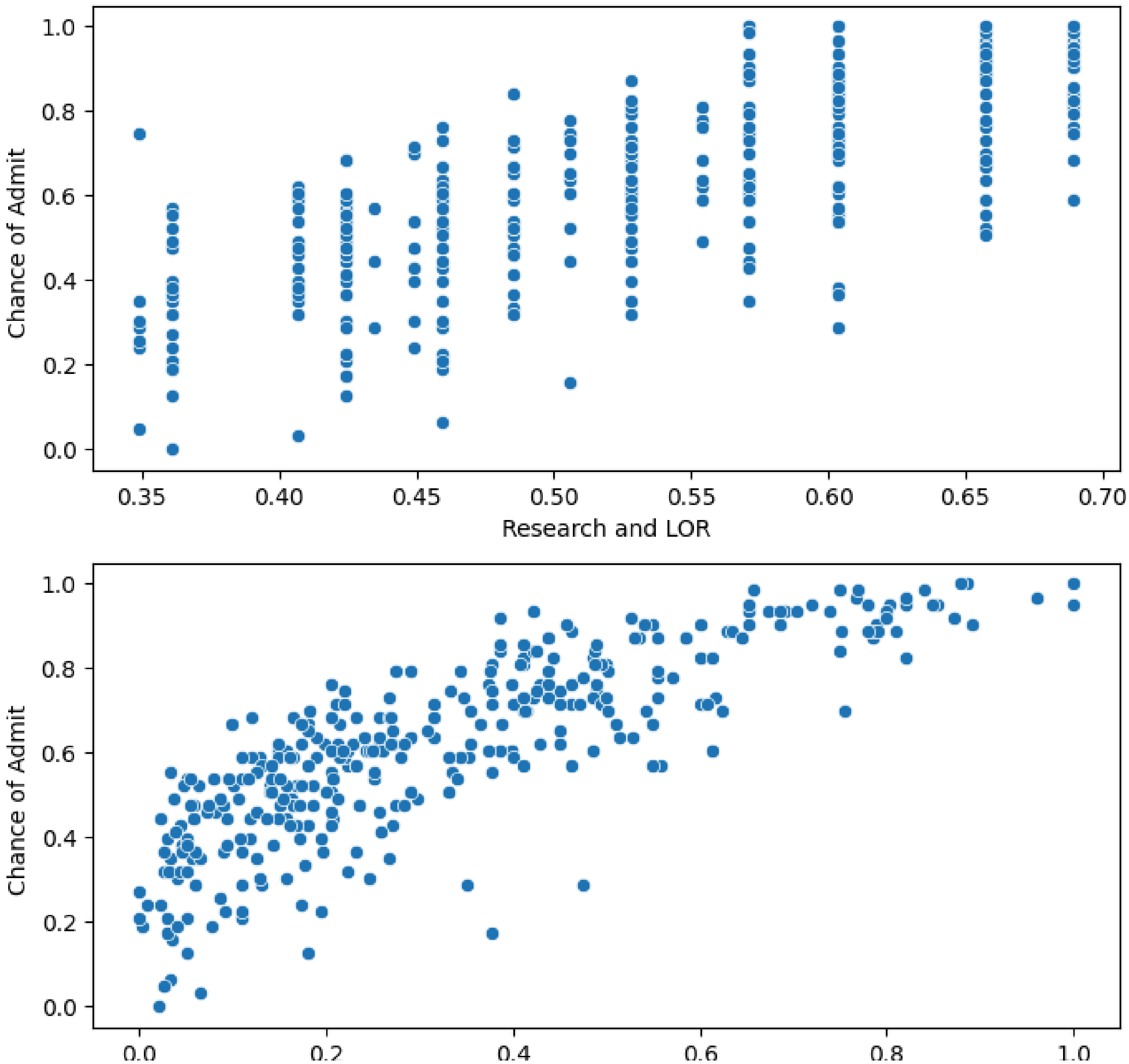

```
round(np.mean(errors),2)
```

```
-0.0
```

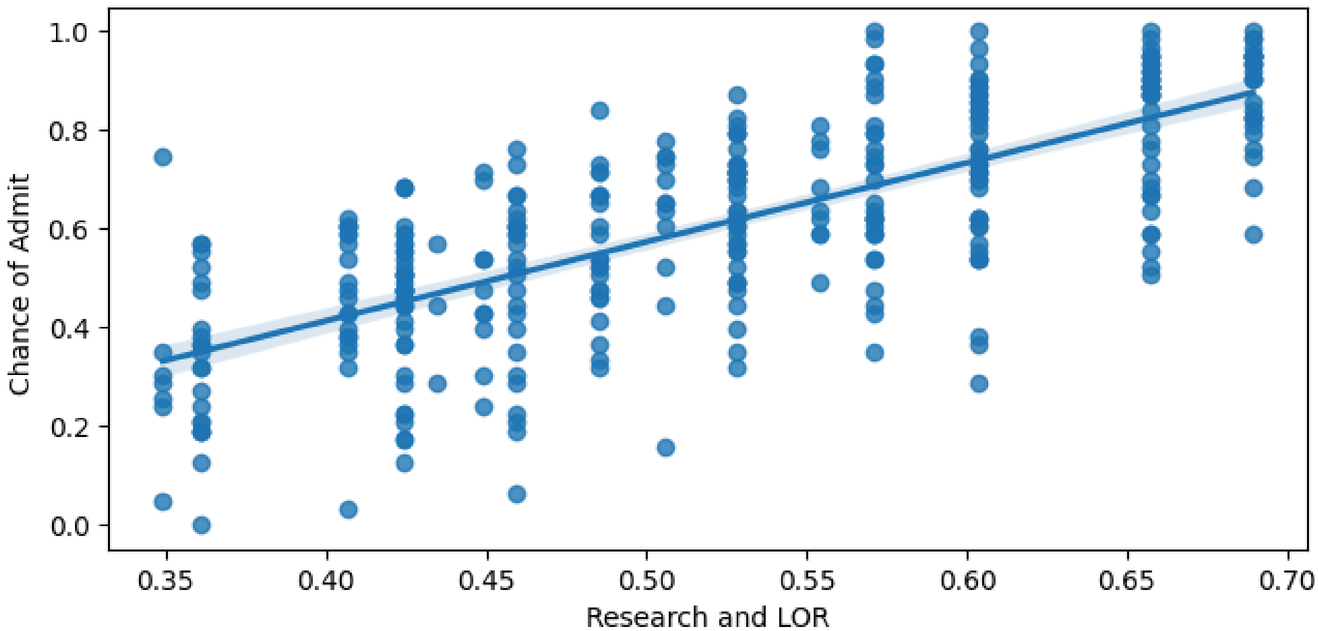
- Mean of residuals are approximately 0

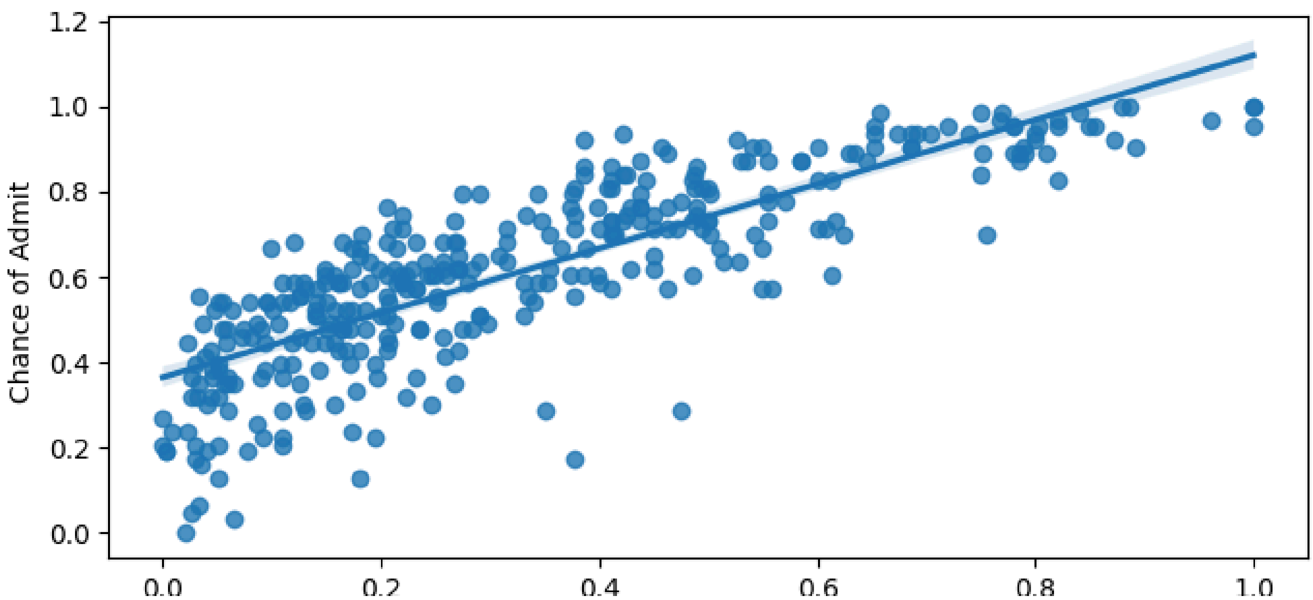
Linearity between independent and dependent variables

```
fig,axs=plt.subplots(nrows=2,ncols=1,squeeze=False,figsize=(8,8))
for i in range(2):
  sns.scatterplot(x=x_train[x_train.columns[i]],y=y_train,ax=axs[i][0])
```




```
fig,axs=plt.subplots(nrows=2,ncols=1,squeeze=False,figsize=(8,8))
for i in range(2):
  sns.regplot(x=x_train[x_train.columns[i]],y=y_train,ax=axs[i][0])
```





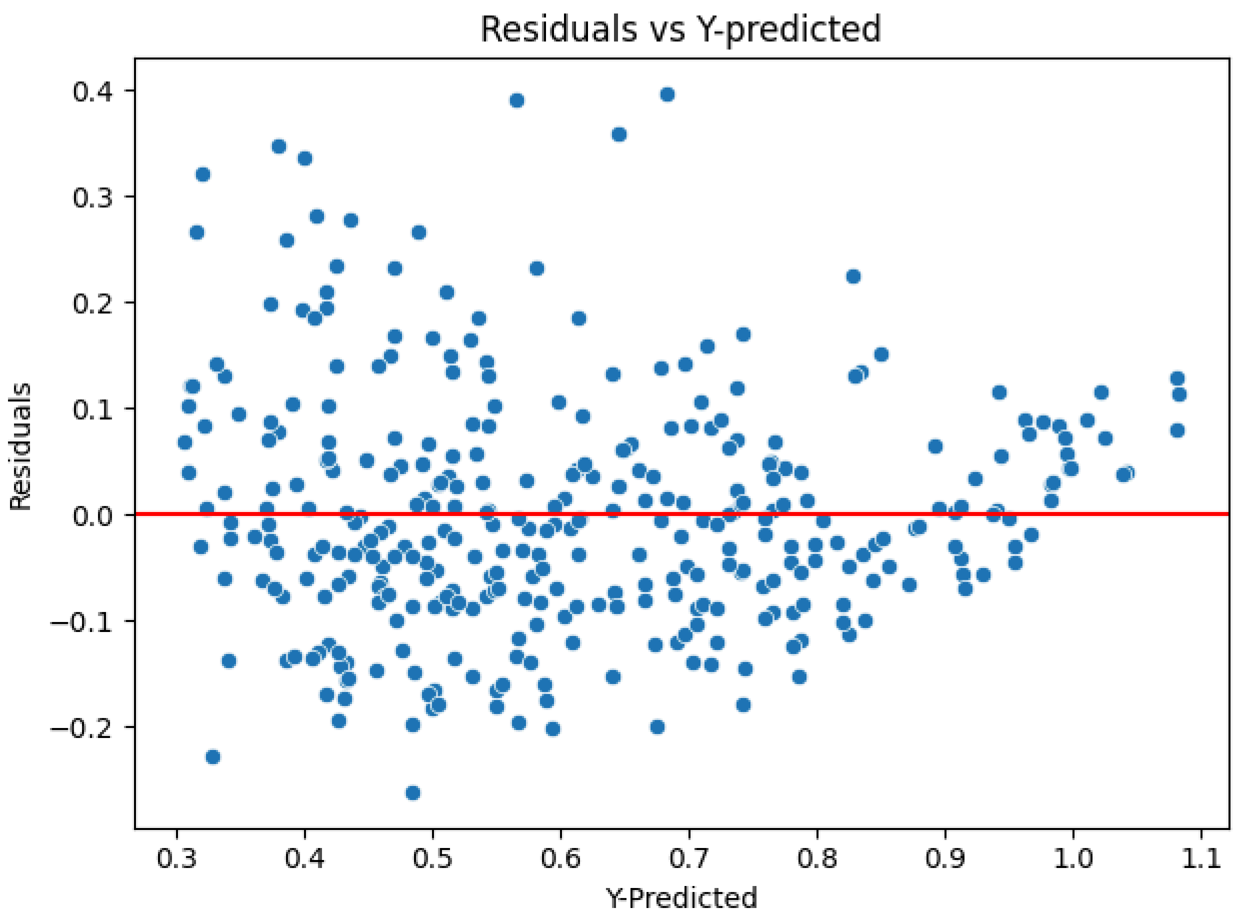

```
for i in range(2):
  print(pearsonr(x_train[x_train.columns[i]],y_train))
```

```
PearsonRResult(statistic=0.7337366020444563, pvalue=2.246461565528052e-60)
PearsonRResult(statistic=0.8237063785941752, pvalue=9.242805571070754e-88)
```

- we can see the relationship is Linear and pearsons correlation tells that they are highly correlated to target variable with very low p-value

Test for Homoscedasticity

```
sns.scatterplot(x=y_hat_residuals,y=errors)
plt.axhline(y=0, color='r', linestyle='-')
plt.ylabel('Residuals')
plt.xlabel('Y-Predicted')
plt.title('Residuals vs Y-predicted')
plt.tight_layout()
plt.show()
```




```
from statsmodels.stats.diagnostic import het_goldfeldquandt

# Perform the Goldfeld-Quandt test
gq_test = het_goldfeldquandt(y_train,x_train)

print(f"F statistic: {gq_test[0]}")
print(f"p-value: {gq_test[1]}")
```

```
F statistic: 1.1571094882255317
p-value: 0.1690660370569174
```

Observations:

- f-statistic comes out to 1.16 implying minimal differnce in varience between groups p-value of 0.16 indicates that this difference is statistically at coventional levels of siginificance(e.g.