cxtkj7zym

February 8, 2024

Business Case:Jamboree Prepared by: Deepali Gupta

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import matplotlib.colors as mcolors
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

Problem Statement Jamboree seeks to enhance its services by offering personalized admission probability assessments for IVY league colleges. This analysis aims to identify and understand the key factors influencing graduate admissions from an Indian perspective. Through comprehensive data collection, integration, and cleaning, the study will employ statistical and machine learning techniques to determine the importance of factors such as standardized test scores, academic achievements, and extracurricular involvement. The developed predictive model will be integrated into Jamboree's website, allowing students to input their details and receive tailored admission probability assessments. Continuous improvement mechanisms will ensure the model's accuracy and relevance, optimizing students' efforts for successful admissions.

Importing Dataset

[2]:
```
jm=pd.read_csv('Jamboree_Admission.txt')
```

[3]:
```
jm
```

[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
..          …          …            …                 …    …    …     …
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]
```

[4]:
```
jm.columns
```

[4]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

[5]:
```
jm.shape
```

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

There are 9 Columns and 500 rows in the dataset.Rows contains the data of 500 students score in different exams, ratings of university, SOP,LOR and their chance of Admission.

Lets explore the columns:-

Serial No. (Unique row ID) GRE Scores (out of 340) TOEFL Scores (out of 120) University Rating (out of 5) Statement of Purpose and 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)

Lets search for Null Values and type of DATA SET

[6]:
```
jm.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
```

1) No null values are present in the any of the columns.
2) Serial No., GRE Score, TOEFL Score, University Rating and Research are present in the form of int64 while SOP,LOR,CGPA and chance of Admit is present as float64.

[7]:
```
jm.describe()
```

[7]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP
count	500.000000	500.000000	500.000000	500.000000	500.000000
mean	250.500000	316.472000	107.192000	3.114000	3.374000
std	144.481833	11.295148	6.081868	1.143512	0.991004
min	1.000000	290.000000	92.000000	1.000000	1.000000
25%	125.750000	308.000000	103.000000	2.000000	2.500000
50%	250.500000	317.000000	107.000000	3.000000	3.500000
75%	375.250000	325.000000	112.000000	4.000000	4.000000
max	500.000000	340.000000	120.000000	5.000000	5.000000

	LOR	CGPA	Research	Chance of Admit
count	500.00000	500.000000	500.000000	500.00000
mean	3.48400	8.576440	0.560000	0.72174
std	0.92545	0.604813	0.496884	0.14114
min	1.00000	6.800000	0.000000	0.34000
25%	3.00000	8.127500	0.000000	0.63000
50%	3.50000	8.560000	1.000000	0.72000
75%	4.00000	9.040000	1.000000	0.82000
max	5.00000	9.920000	1.000000	0.97000

University Rating: On a scale of 1 to 5, the university rating averages at 3.11.

Statement of Purpose (SOP) and Letter of Recommendation (LOR): SOP scores fall within the range of 1 to 5, with an average score of 3.37. LOR scores also range from 1 to 5, with an average of 3.48.

CGPA (Cumulative Grade Point Average): The CGPA varies between 6.8 and 9.92, with a mean of 8.58.

Research Experience: Approximately 56% of applicants possess research experience, given the mean value of 0.56.

Chance of Admit: The chance of admission spans from 0.34 to 0.97, with an average value of 0.72.

Seggregating Categorical and Numerical Columns

[8]:
```
num_col=['GRE Score','TOEFL Score','CGPA']
cat_col=['University Rating','SOP','LOR ','Research']
target='Chance of Admit '
```

Univariate Analysis

[9]:
```
rating=jm['University Rating'].value_counts().reset_index().
 ↪rename(columns={'University Rating':'Count','index':'University Rating'})
```

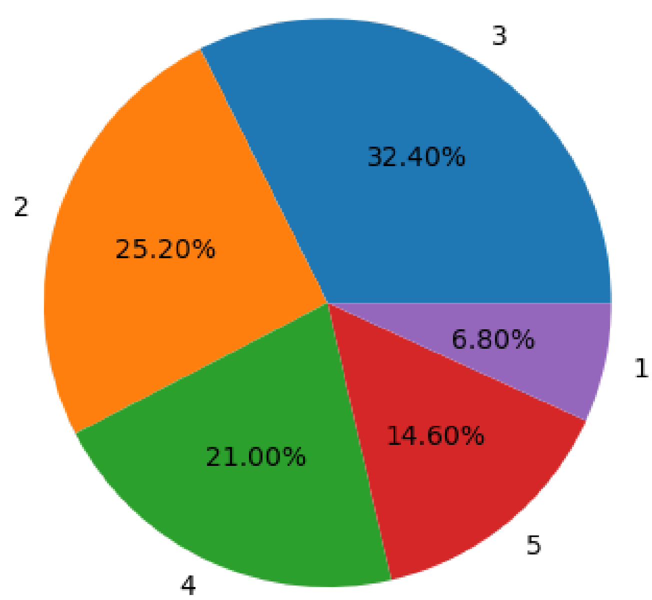
[10]:
```
rating
```

[10]:

	University Rating	Count
0	3	162
1	2	126
2	4	105
3	5	73
4	1	34

[11]:
```
plt.pie(rating['Count'],labels=rating['University Rating'],autopct = '%.2f%%')
plt.plot
```

[11]:
```
<function matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None,
**kwargs)>
```

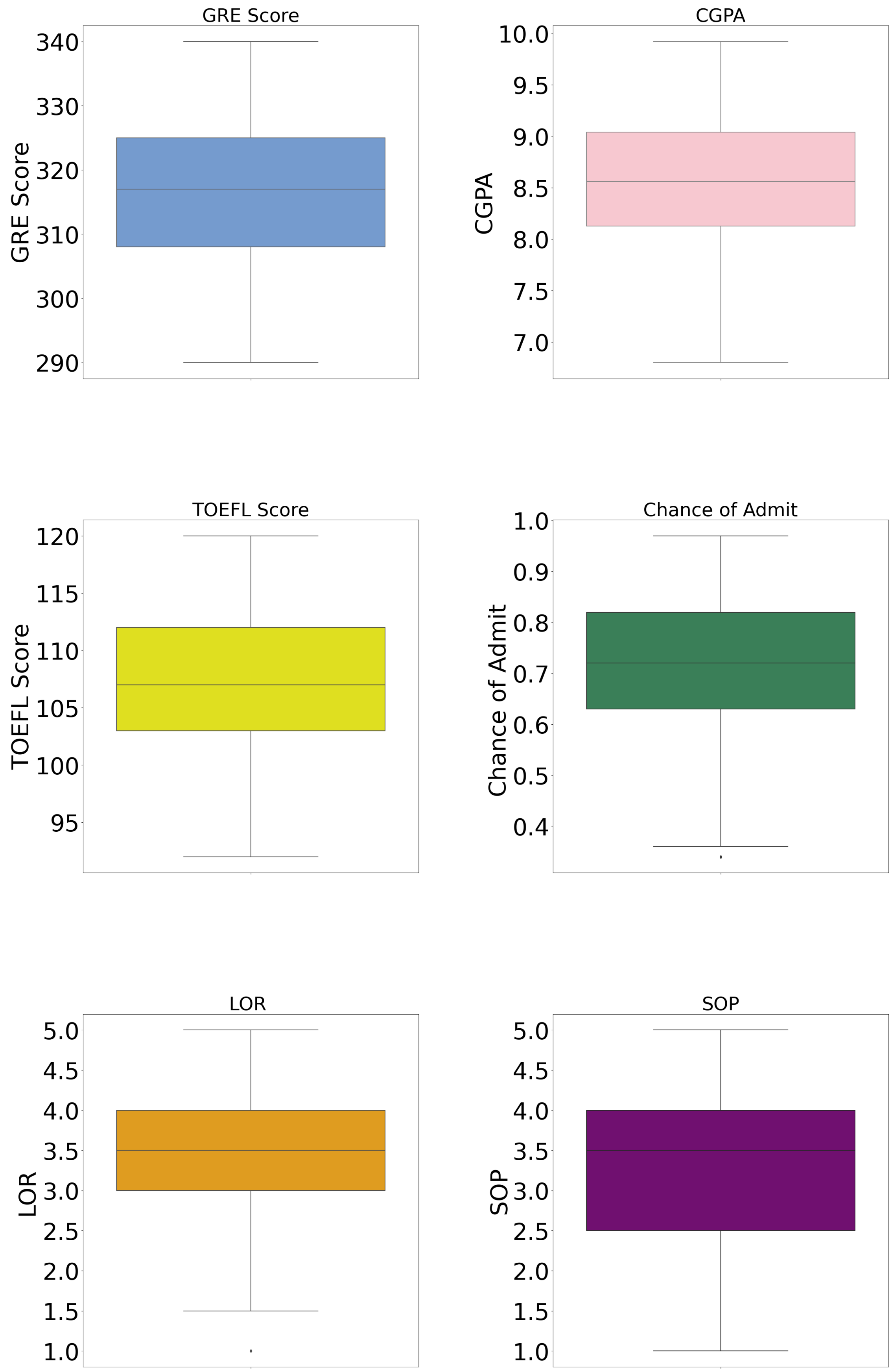


[ ]:

```

```

[12]:

```
fig=plt.figure(figsize=(30,50))
plt.subplot(3,2,1)
plt.title('GRE Score',fontsize=40)
plt.yticks(fontsize = 50)
plt.ylabel('GRE Score',fontsize = 50)
sns.boxplot(y=jm['GRE Score'],color="#69d")
plt.subplot(3,2,2)
plt.title('CGPA',fontsize=40)
plt.yticks(fontsize = 50)
plt.ylabel('CGPA ',fontsize = 50)
sns.boxplot(y=jm['CGPA'],color='pink')
plt.subplot(3,2,3)
plt.title('TOEFL Score',fontsize=40)
plt.yticks(fontsize = 50)
sns.boxplot(y=jm['TOEFL Score'],color='yellow')
plt.ylabel('TOEFL Score',fontsize = 50)
#plt.subplots_adjust(hspace=6)
plt.subplot(3,2,4)
plt.title('Chance of Admit',fontsize=40)
plt.yticks(fontsize = 50)
sns.boxplot(y=jm['Chance of Admit '],color="seagreen")
plt.ylabel('Chance of Admit ',fontsize = 50)

plt.subplot(3,2,5)
plt.title('LOR ',fontsize=40)
plt.yticks(fontsize = 50)
sns.boxplot(y=jm['LOR '],color="orange")
plt.ylabel('LOR ',fontsize = 50)

plt.subplot(3,2,6)
plt.title('SOP',fontsize=40)
plt.yticks(fontsize = 50)
sns.boxplot(y=jm['SOP'],color="purple")
plt.ylabel('SOP',fontsize = 50)


plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.4)
plt.show()
```


GRE Score
340
330
320
310
300
290
GRE Score
CGPA
10.0
9.5
9.0
8.5
8.0
7.5
7.0
CGPA
TOEFL Score
120
115
110
105
100
95
TOEFL Score
Chance of Admit
1.0
0.9
0.8
0.7
0.6
0.5
0.4
Chance of Admit
LOR
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
LOR
SOP
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
SOP

[13]:
```
jm.columns
```

[13]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

Outlier Dettection

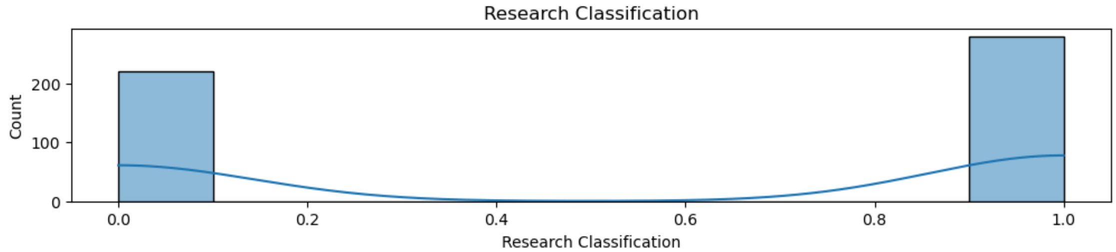
[14]:
```
def get_outliers(jm, series_name):
    q1 = np.percentile(jm[series_name], 25)
    q3 = np.percentile(jm[series_name], 75)
    iqr = q3 - q1

    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    outliers = jm.loc[(jm[series_name]<lower_bound) |
 ↪(jm[series_name]>upper_bound), series_name]
    return outliers.to_list()
```

[15]:
```
res = []
for col in jm.columns:
    t = get_outliers(jm,col)
    if len(t)>0:
        res.append([col, len(t), t])

df_t = pd.DataFrame(res,columns=["Column Name", "No. of Outliers", "Outliers"]).
 ↪sort_values("No. of Outliers", ascending=False)
df_t
```

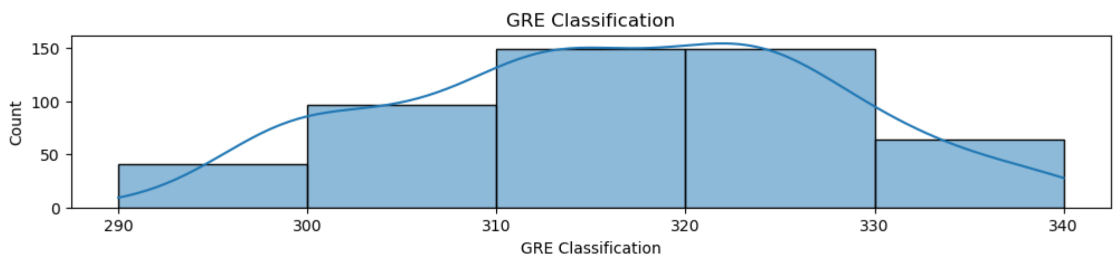
[15]:

	Column Name	No. of Outliers	Outliers
1	Chance of Admit	2	[0.34, 0.34]
0	LOR	1	[1.0]

[16]:
```
plt.figure(figsize=(12, 2))
sns.histplot(jm['Research'], bins=10, kde=True)
plt.title('Research Classification')
plt.xlabel('Research Classification')
plt.ylabel('Count')
plt.show()
```

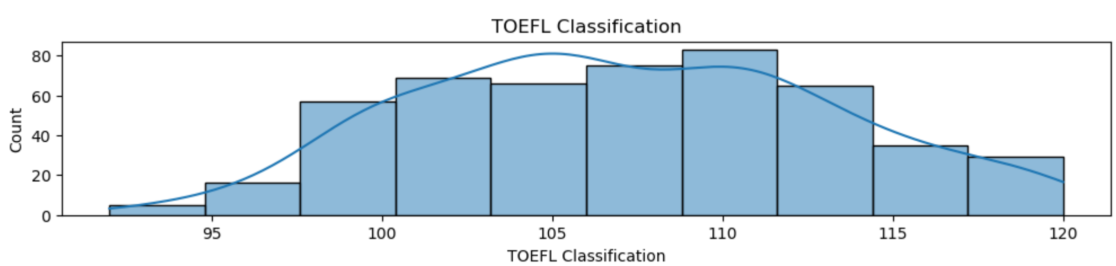



```
[17]: plt.figure(figsize=(12, 2))
sns.histplot(jm['GRE Score'], bins=5, kde=True)
plt.title('GRE Classification')
plt.xlabel('GRE Classification')
plt.ylabel('Count')
plt.show()
```

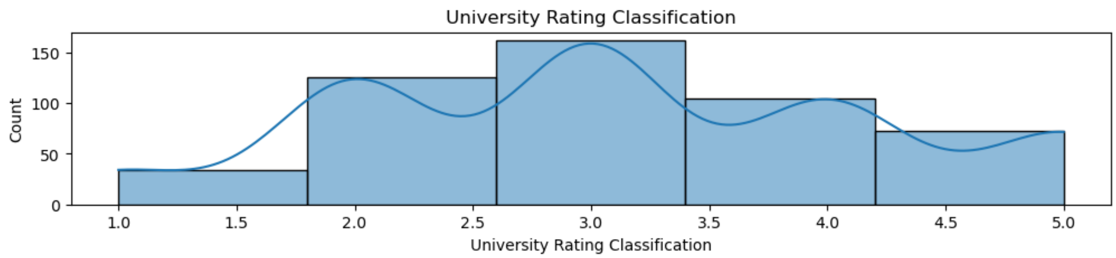


A limited number of students exhibit extremely high or low TOEFL scores, as evident from the outliers at both ends of the distribution curve.

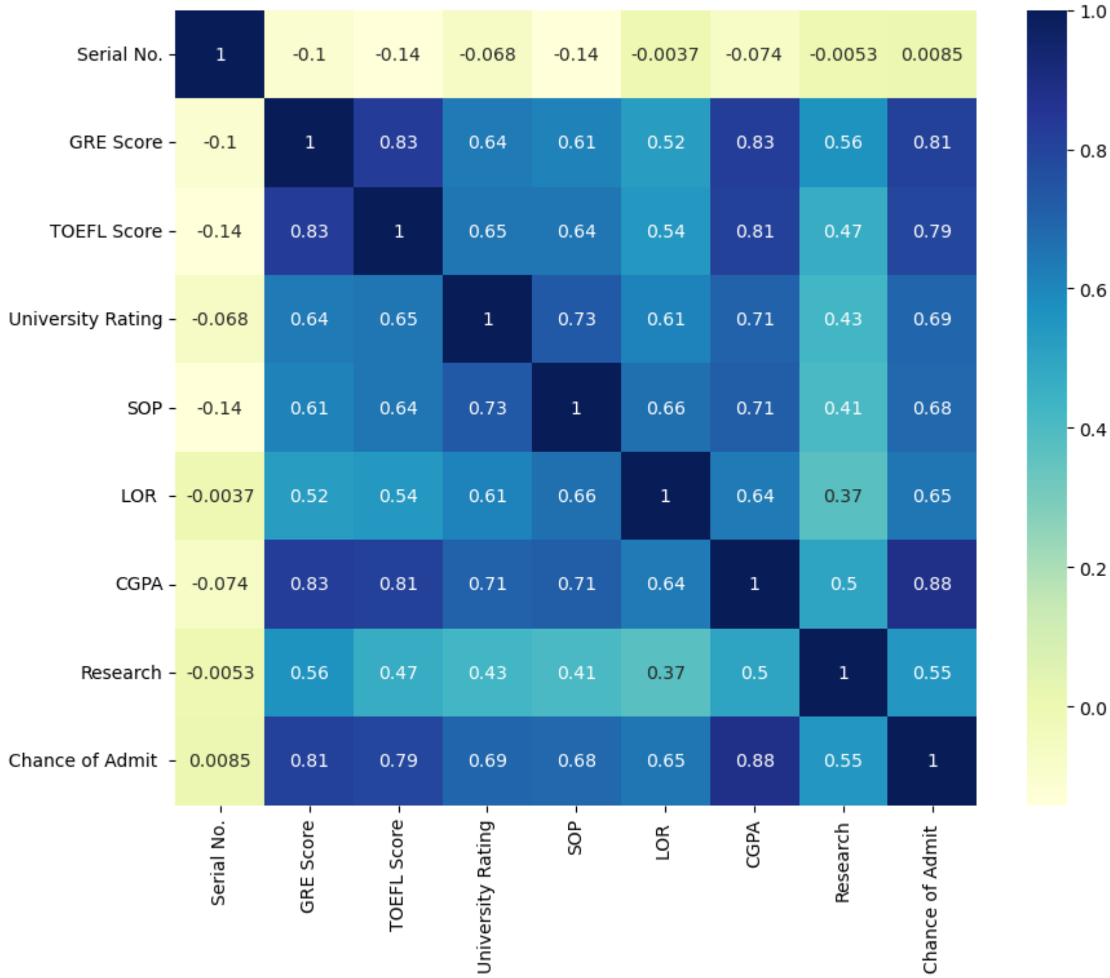
```
[18]: plt.figure(figsize=(12, 2))
sns.histplot(jm['TOEFL Score'], bins=10, kde=True)
plt.title('TOEFL Classification')
plt.xlabel('TOEFL Classification')
plt.ylabel('Count')
plt.show()
```



[19]:
```
plt.figure(figsize=(12, 2))
sns.histplot(jm['University Rating'], bins=5, kde=True)
plt.title('University Rating Classification')
plt.xlabel('University Rating Classification')
plt.ylabel('Count')
plt.show()
```

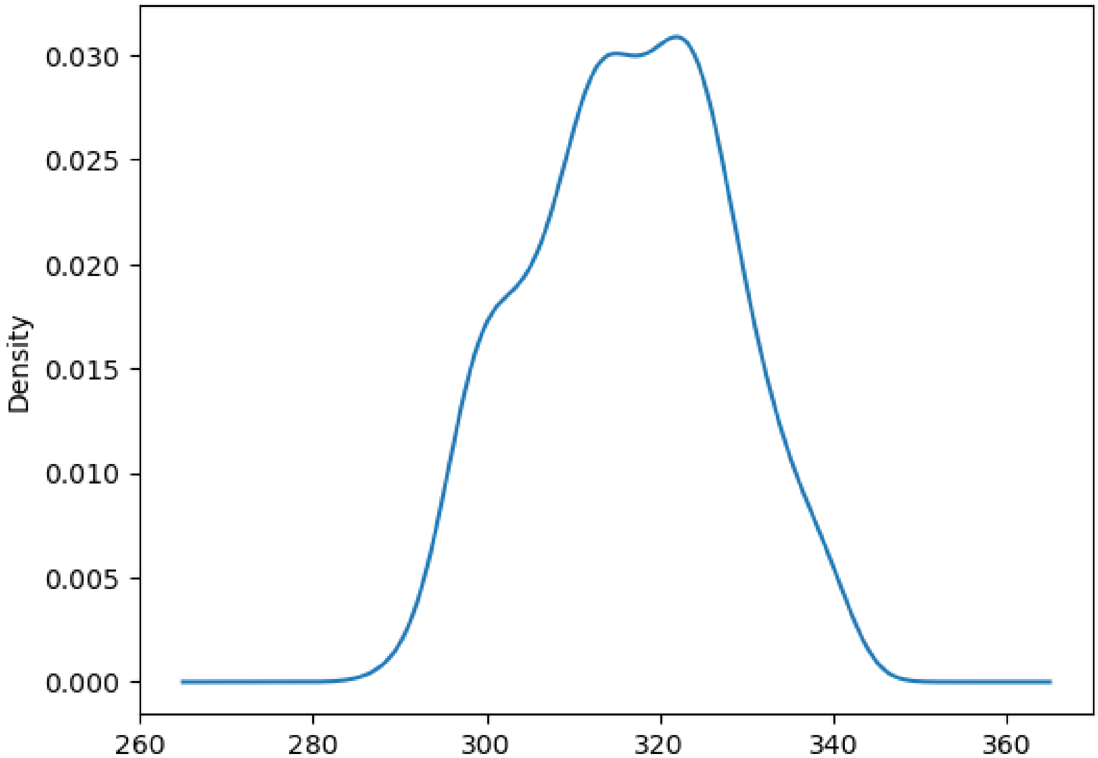


[20]:
```
plt.figure(figsize=(10,8))
ax = sns.heatmap(jm.corr(), cmap="YlGnBu", annot=True)
```



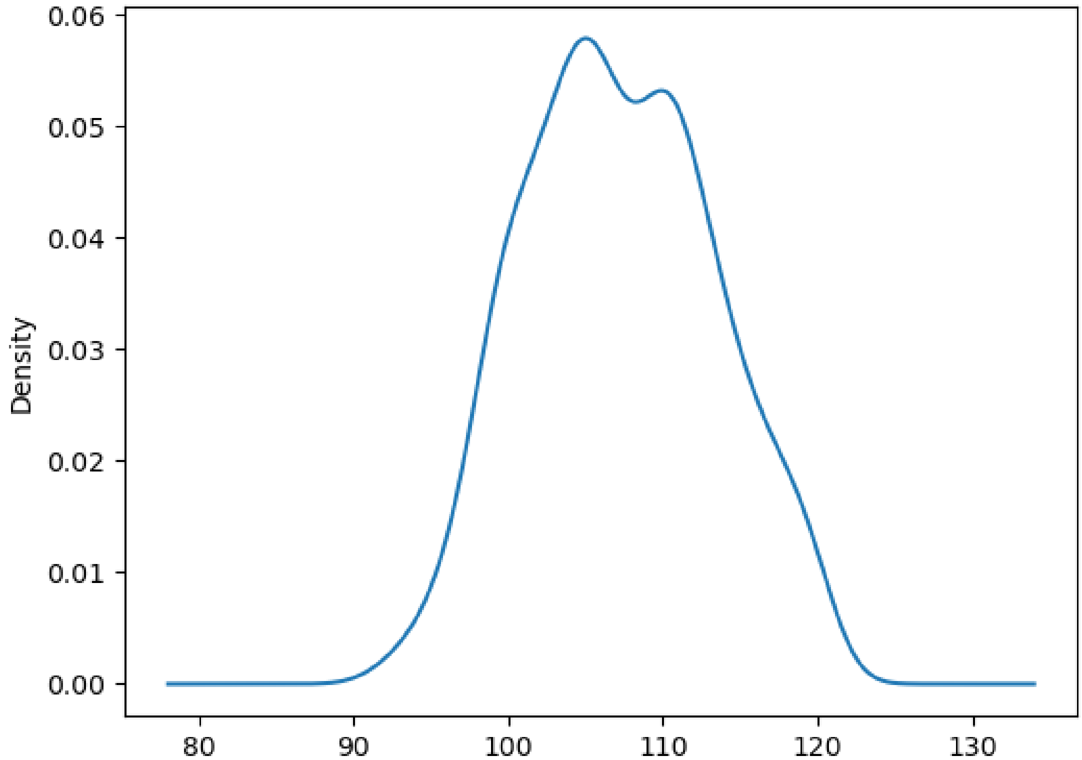
[21]:
```
jm['GRE Score'].plot.density()
```

[21]: <Axes: ylabel='Density'>



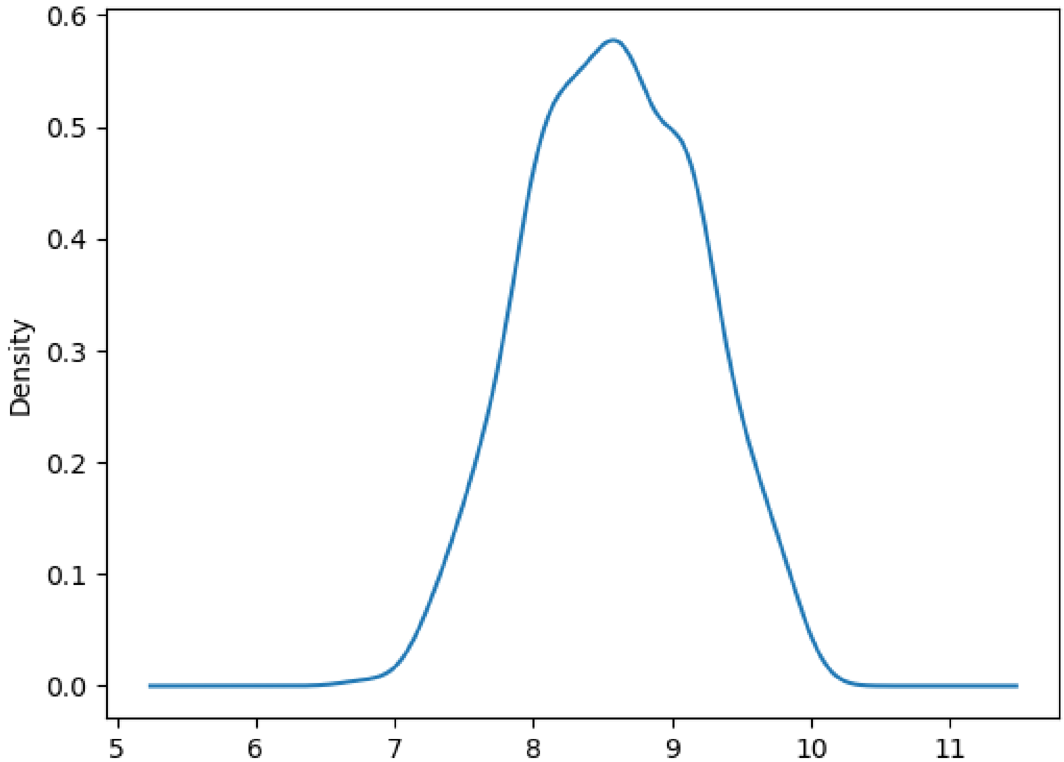
[22]:
```
jm['TOEFL Score'].plot.density()
```

[22]: <Axes: ylabel='Density'>

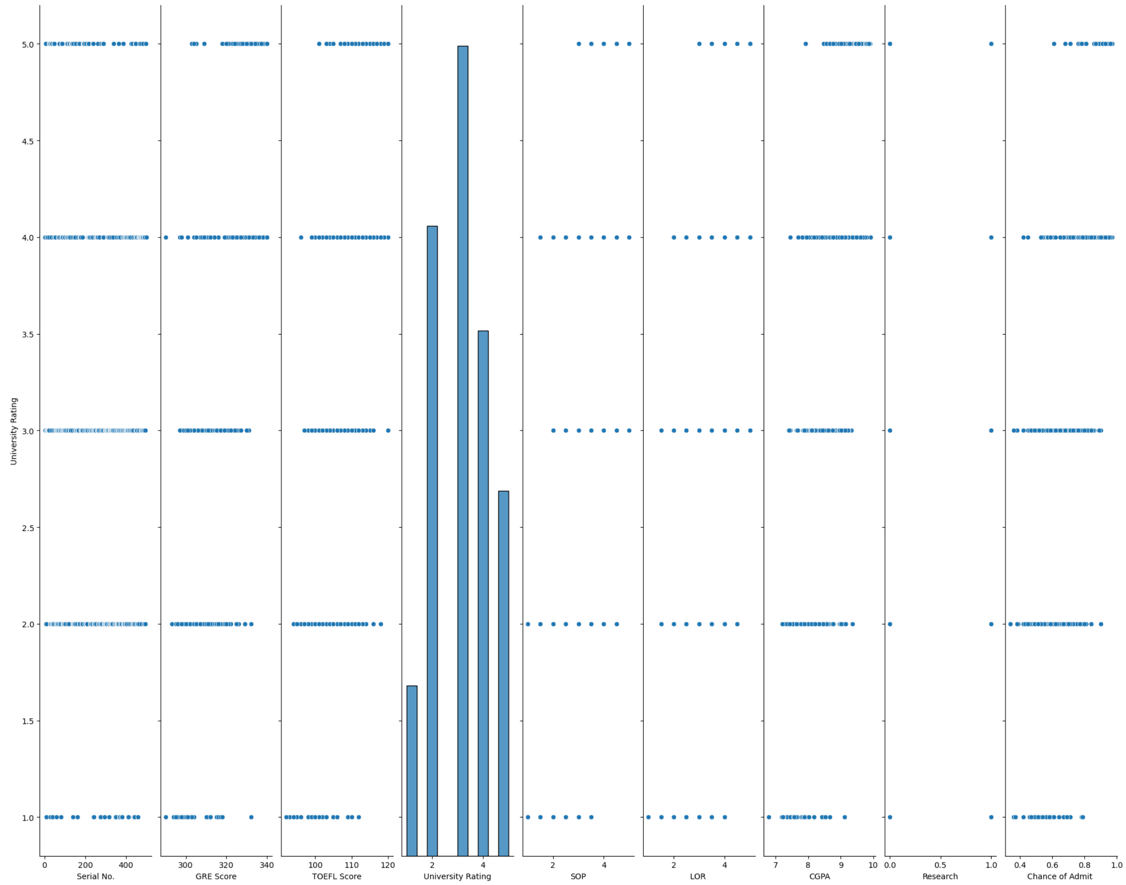



```
[23]: jm['CGPA'].plot.density()
```

```
[23]: <Axes: ylabel='Density'>
```

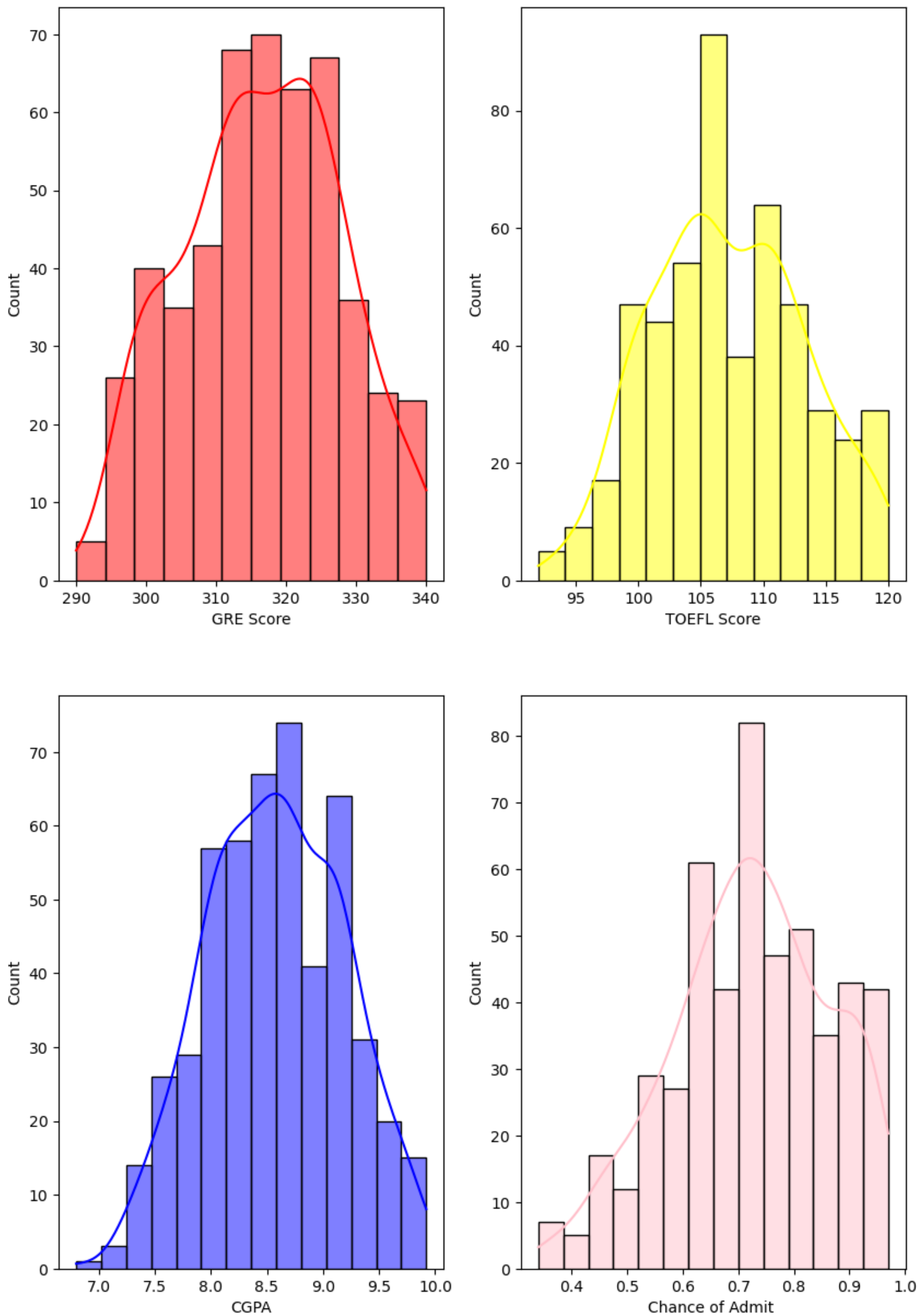


[24]:
```
pp=sns.pairplot(jm, y_vars=["University Rating"]);
pp.fig.set_size_inches(20,20)
```



[25]:
```
fig=plt.figure(figsize=(10,15))

plt.subplot(2,2,1)
sns.histplot(jm['GRE Score'],kde=True,color='Red')

plt.subplot(2,2,2)
sns.histplot(jm['TOEFL Score'],kde=True,color='yellow')

plt.subplot(2,2,3)
sns.histplot(jm['CGPA'],kde=True,color='blue')

plt.subplot(2,2,4)
sns.histplot(jm['Chance of Admit '],kde=True,color='pink')
```

[25]:
```
<Axes: xlabel='Chance of Admit ', ylabel='Count'>
```

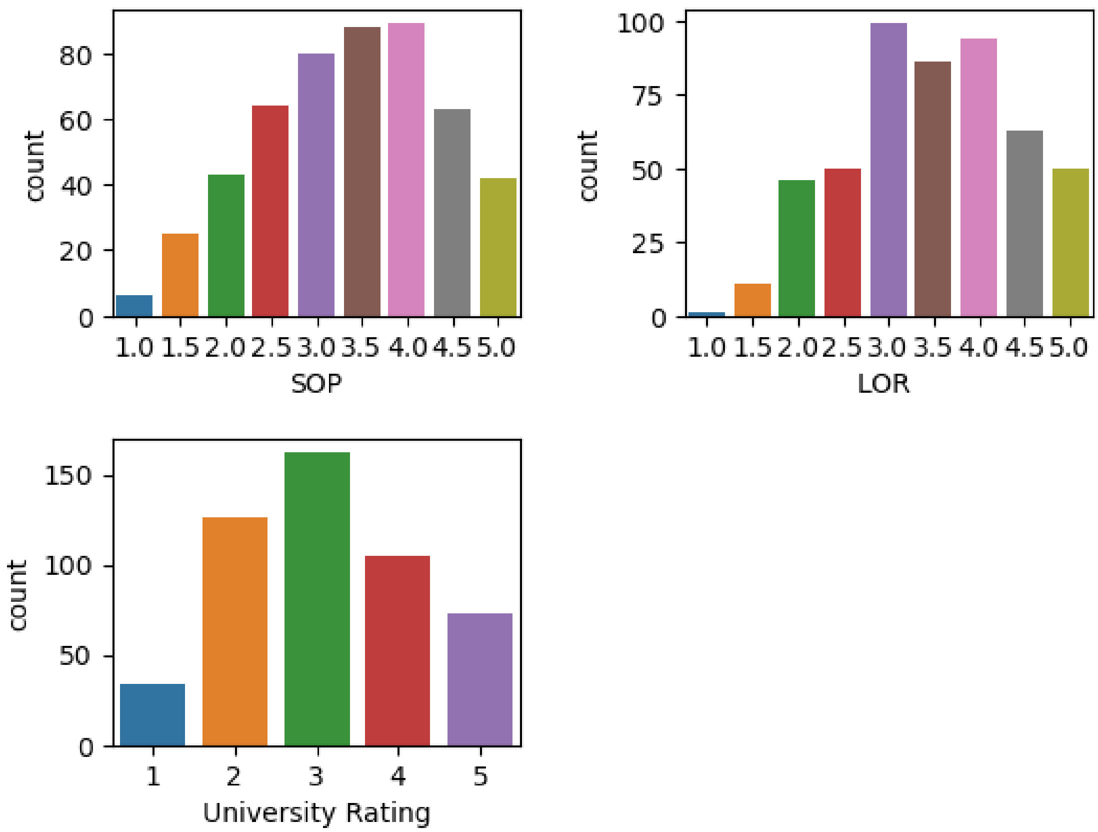

0
10
20
30
40
50
60
70
Count
290
300
310
320
330
340
GRE Score
0
20
40
60
80
Count
95
100
105
110
115
120
TOEFL Score
0
10
20
30
40
50
60
70
Count
7.0
7.5
8.0
8.5
9.0
9.5
10.0
CGPA
0
10
20
30
40
50
60
70
80
Count
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Chance of Admit

Categorical Plots

[26]:
```
jm.columns
```

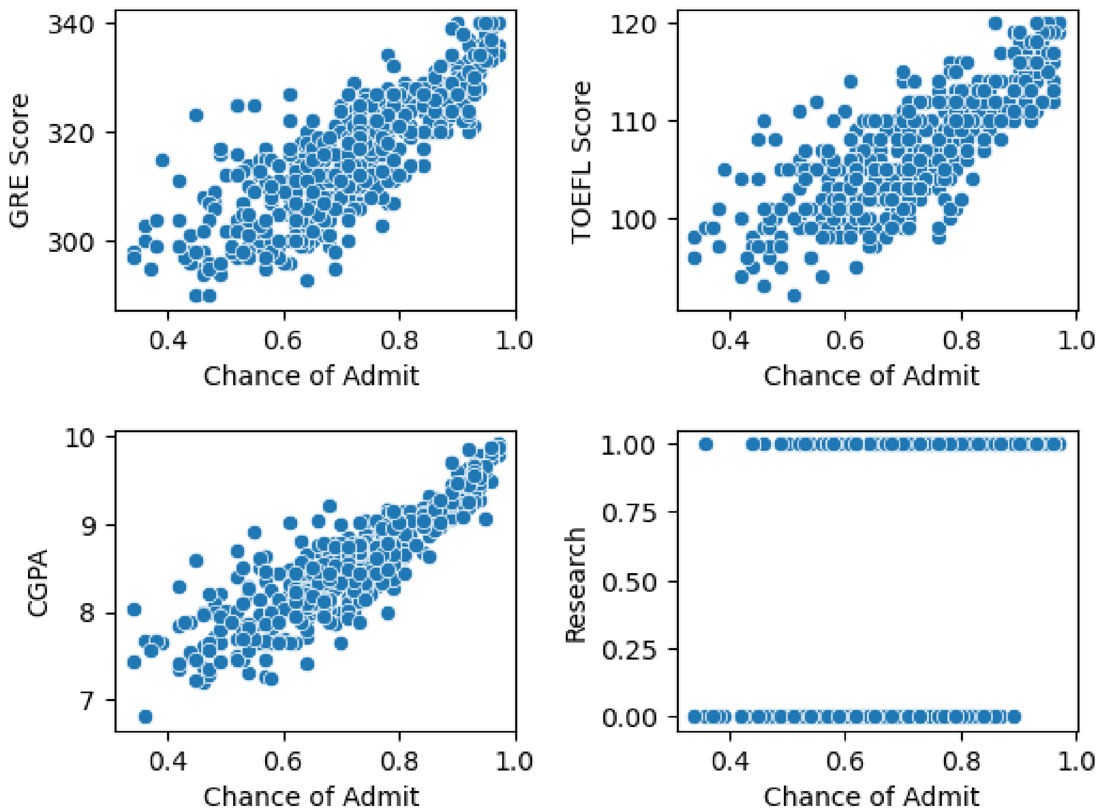
[26]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

[27]:
```
plt.subplot(2,2,1)
sns.countplot(x ='SOP', data = jm)

plt.subplot(2,2,2)
sns.countplot(x ='LOR ', data = jm)


plt.subplot(2,2,3)
sns.countplot(x ='University Rating', data = jm)


plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.4)
plt.show()
```



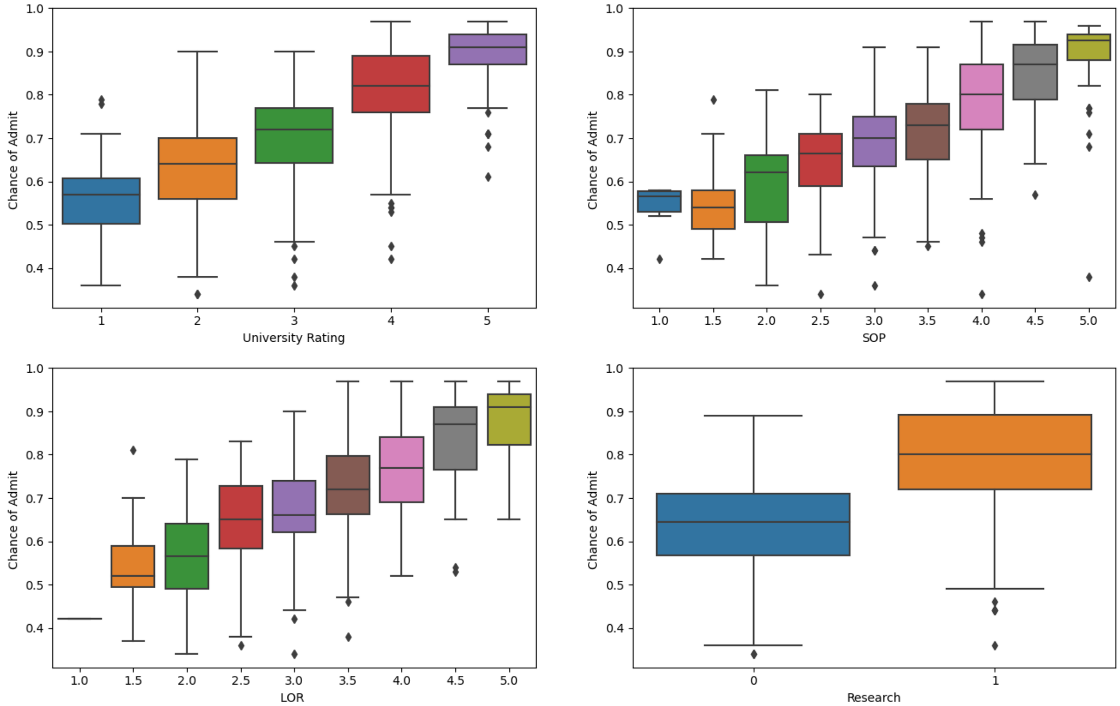
BIVARIATE ANALYSIS

[28]:
```
plt.subplot(2,2,1)
sns.scatterplot(y='GRE Score',x='Chance of Admit ',data=jm)

plt.subplot(2,2,2)
sns.scatterplot(y='TOEFL Score',x='Chance of Admit ',data=jm)

plt.subplot(2,2,3)
sns.scatterplot(y='CGPA',x='Chance of Admit ',data=jm)

plt.subplot(2,2,4)
sns.scatterplot(y='Research',x='Chance of Admit ',data=jm)

plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.4)
plt.show()
```



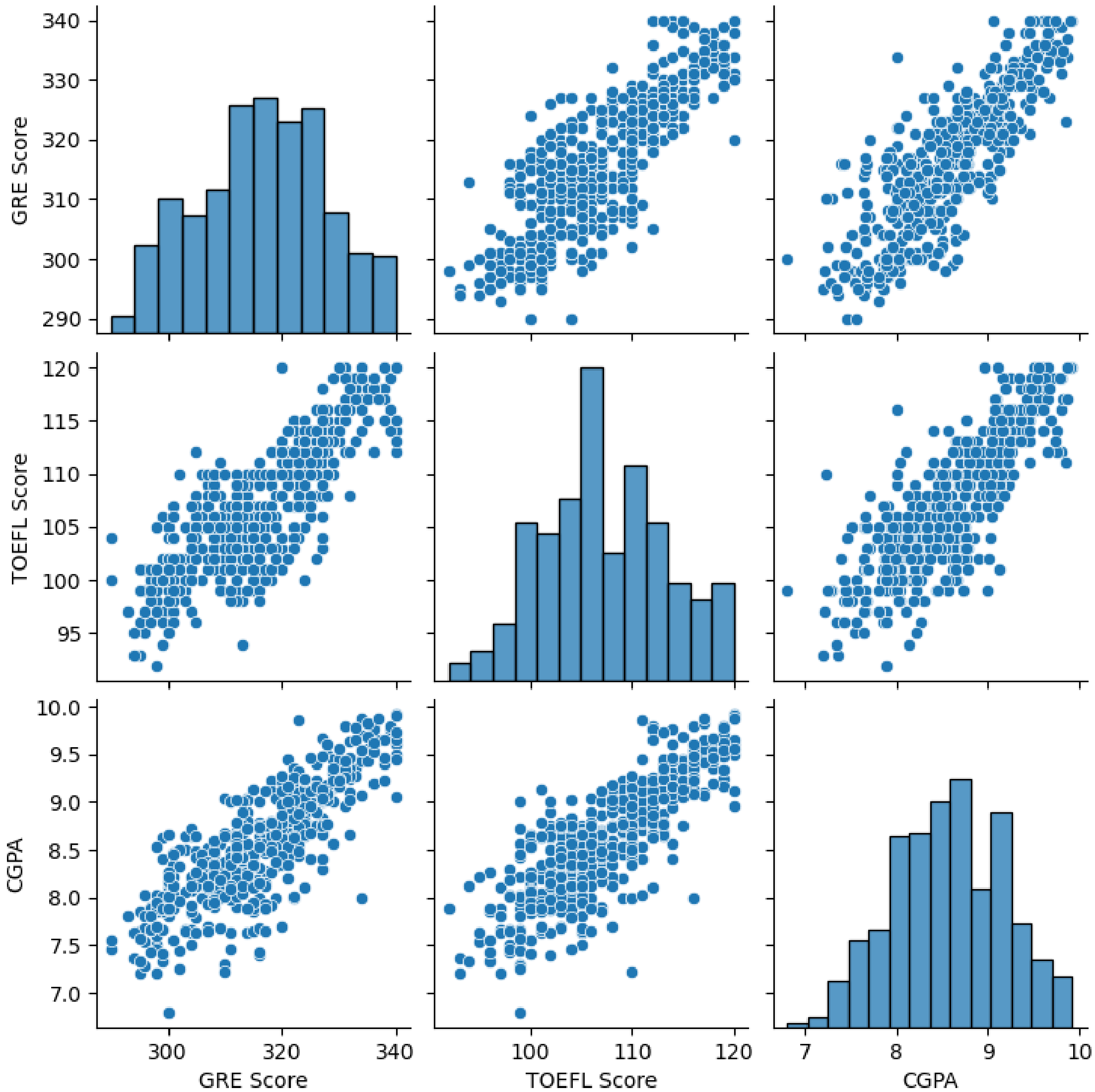
Looks like linear relationship between target and features.

[29]:
```
rows, cols = 2,2
fig, axs = plt.subplots(rows, cols, figsize=(16,10))

index = 0
for row in range(rows):
    for col in range(cols):
        sns.boxplot(x=cat_col[index], y=target, data=jm, ax=axs[row,col])
        index += 1
```



[30]:
```
sns.pairplot(jm[num_col])
```

[30]: <seaborn.axisgrid.PairGrid at 0x15c6950d0>



[31]:
```
jm=jm.drop(columns=['Serial No.'])
```

[32]:
```
jm.columns
```

[32]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

[33]:
```
scaler = MinMaxScaler()
scaler.fit(jm)
scaled_values = scaler.transform(jm)  # returns numpy.ndarray not df.
scaled_jm = pd.DataFrame(scaled_values,columns=jm.columns)
scaled_jm.head()
```

[33]:

```
   GRE Score  TOEFL Score  University Rating    SOP    LOR       CGPA  \
0       0.94     0.928571               0.75  0.875  0.875  0.913462
1       0.68     0.535714               0.75  0.750  0.875  0.663462
2       0.52     0.428571               0.50  0.500  0.625  0.384615
3       0.64     0.642857               0.50  0.625  0.375  0.599359
4       0.48     0.392857               0.25  0.250  0.500  0.451923

   Research  Chance of Admit
0       1.0          0.920635
1       1.0          0.666667
2       1.0          0.603175
3       1.0          0.730159
4       0.0          0.492063
```

[34]:

```
Y=jm['Chance of Admit ']
X=jm.drop(columns=[target])
```

[35]:

```
Y
```

[35]:

```
0      0.92
1      0.76
2      0.72
3      0.80
4      0.65
       …
495    0.87
496    0.96
497    0.93
498    0.73
499    0.84
Name: Chance of Admit , Length: 500, dtype: float64
```

[36]:

```
X
```

[36]:

```
     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]
```

[37]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import r2_score

from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy import stats
```

[38]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,␣
 ↪random_state=1)
```

[39]:
```
print(X_train.shape, Y_train.shape)
print(X_test.shape, Y_test.shape)
```

```
(350, 7) (350,)
(150, 7) (150,)
```

[42]:
```
#Linear Regression
```

[43]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, Y_train)
```

[43]:
```
LinearRegression()
```

[44]:
```
model.coef_
```

[44]:
```
array([ 0.00165342,  0.00381453,  0.01012349, -0.00100952,  0.01351732,
        0.10703419,  0.02813965])
```

[45]:
```
model.intercept_
```

[45]:
```
-1.2161131174465911
```

[53]:
```
model_scores = {}
model_scores["Score Parameter"] = ["R2","Adjusted R2"]
```

[55]:
```
train_r2 = np.round(lr_model.score(X_train, Y_train),4)
train_adj_r2 = np.round(1 - ((1-train_r2)*(X_train.shape[0]-1)) / (X_train.
 ↪shape[0] - X_train.shape[1] - 1),4)
model_scores["Training Score"] = [train_r2, train_adj_r2]
```

[56]:
```
y_pred = model.predict(X_test)
```

[58]:
```
from sklearn.metrics import r2_score
test_r2 = np.round(r2_score(Y_test, y_pred),4)
```

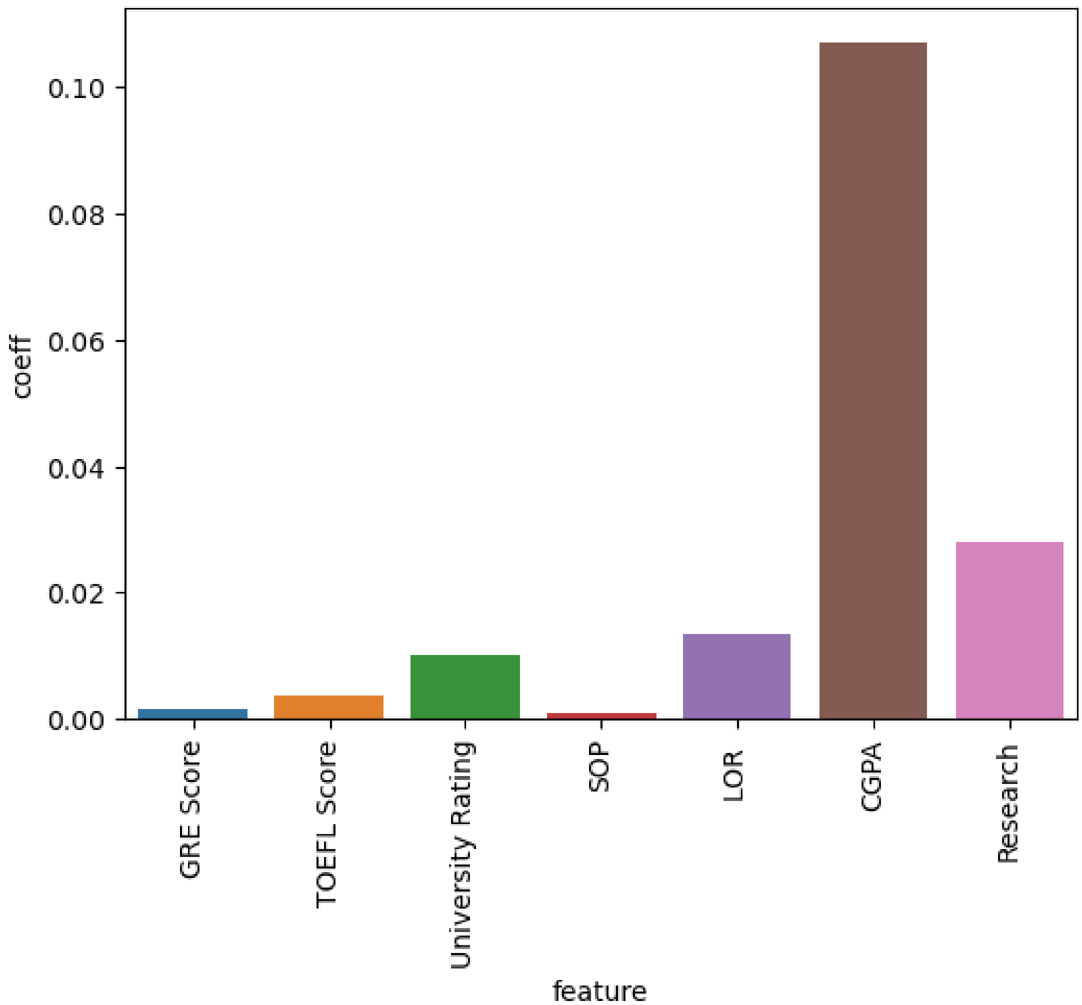
```
test_adj_r2 = np.round(1 - ((1-test_r2)*(X_test.shape[0]-1)) / (X_test.shape[0]␣
 ↪- X_test.shape[1] - 1),4)
model_scores["Test Score"] = [test_r2, test_adj_r2]
```

[62]:
```
print("Linear Regression Model Scores")
print(model_scores)
```

```
Linear Regression Model Scores
{'Score Parameter': ['R2', 'Adjusted R2'], 'Training Score': [0.821, 0.8173],
'Test Score': [0.8158, 0.8067]}
```

[63]:
```
imp = pd.DataFrame(list(zip(X_test.columns,np.abs(model.coef_))),
                   columns=['feature', 'coeff'])
sns.barplot(x='feature', y='coeff', data=imp)
plt.xticks(rotation=90)
```

[63]:
```
(array([0, 1, 2, 3, 4, 5, 6]),
 [Text(0, 0, 'GRE Score'),
  Text(1, 0, 'TOEFL Score'),
  Text(2, 0, 'University Rating'),
  Text(3, 0, 'SOP'),
  Text(4, 0, 'LOR '),
  Text(5, 0, 'CGPA'),
  Text(6, 0, 'Research')])
```



[64]:
```
X_test.columns[np.argmax(np.abs(model.coef_))]
```

[64]: 'CGPA'

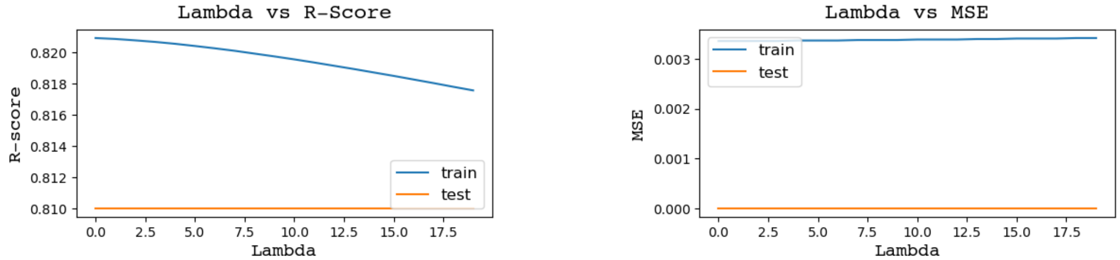
[66]:
```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(Y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.004125934236707796
```

[ ]:
```
import statsmodels.api as sm
X_sm = sm.add_constant(X_train)  # Statmodels default is without intercept, to␣
↪add intercept we need to add constant.

model = sm.OLS(Y_train, X_sm)
```

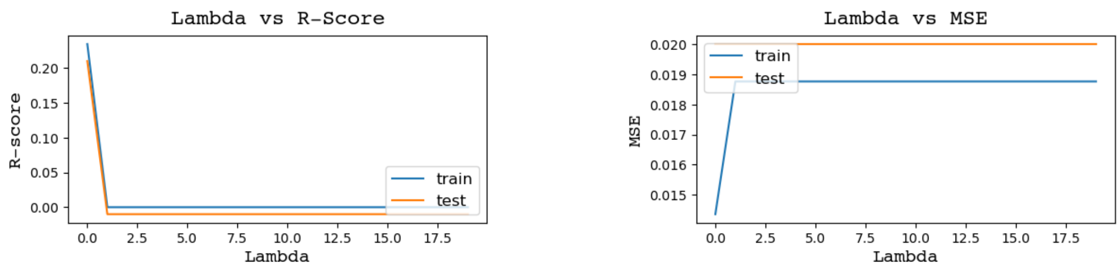
```
results = model.fit()

# Print the summary statistics of the model
print(results.summary())
```

[67]:
```
from sklearn.linear_model import Ridge

train_r2 = []
test_r2  = []

train_mse = []
test_mse = []
lambdas = np.round(np.linspace(1,10,num=20),4)
for i in lambdas:
    rdg = Ridge(alpha = i)
    rdg.fit(X_train, Y_train)

    train_r2.append(np.round(rdg.score(X_train,Y_train),5))
    train_mse.append(np.round(mean_squared_error(Y_train, rdg.
 ↪predict(X_train)),5))

    y_pred = rdg.predict(X_test)
    test_r2.append(np.round(r2_score(Y_test,y_pred),2))
    test_mse.append(np.round(mean_squared_error(Y_test,y_pred),2))

fig, axes = plt.subplots(1, 2, figsize=(12, 3))
axes[0].plot(list(range(0, lambdas.size)), train_r2, label="train")
axes[0].plot(list(range(0, lambdas.size)), test_r2, label="test")
axes[0].legend(loc='lower right', fontsize=12)
axes[0].set_xlabel("Lambda", fontsize=14, fontname='Courier')
axes[0].set_ylabel("R-score", fontsize=14, fontname='Courier')
axes[0].set_title("Lambda vs R-Score", y=1.02, fontsize=16, fontname='Courier')␣
 ↪ # Fixed title method

axes[1].plot(list(range(0, lambdas.size)), train_mse, label="train")
axes[1].plot(list(range(0, lambdas.size)), test_mse, label="test")
axes[1].legend(loc='upper left', fontsize=12)
axes[1].set_xlabel("Lambda", fontsize=14, fontname='Courier')
axes[1].set_ylabel("MSE", fontsize=14, fontname='Courier')
axes[1].set_title("Lambda vs MSE", y=1.02, fontsize=16, fontname='Courier')  #␣
 ↪Fixed title method

plt.tight_layout()
plt.subplots_adjust(hspace=0.3)
plt.subplots_adjust(wspace=0.5)
plt.show()
```



[68]:
```
from sklearn.linear_model import Lasso

train_r2 = []
test_r2  = []

train_mse = []
test_mse = []
lambdas = np.round(np.linspace(1,10,num=20),4)
for i in lambdas:
    lso = Lasso(alpha = i)
    lso.fit(X_train, Y_train)

    train_r2.append(np.round(lso.score(X_train, Y_train),5))
    train_mse.append(np.round(mean_squared_error(Y_train, lso.
 ↪predict(X_train)),5))

    y_pred = lso.predict(X_test)
    test_r2.append(np.round(r2_score(Y_test,y_pred),2))
    test_mse.append(np.round(mean_squared_error(Y_test,y_pred),2))

fig, axes = plt.subplots(1, 2, figsize=(12, 3))
axes[0].plot(list(range(0, lambdas.size)), train_r2, label="train")
axes[0].plot(list(range(0, lambdas.size)), test_r2, label="test")
axes[0].legend(loc='lower right', fontsize=12)
axes[0].set_xlabel("Lambda", fontsize=14, fontname='Courier')
axes[0].set_ylabel("R-score", fontsize=14, fontname='Courier')
axes[0].set_title("Lambda vs R-Score", y=1.02, fontsize=16, fontname='Courier')␣
 ↪ # Fixed title method

axes[1].plot(list(range(0, lambdas.size)), train_mse, label="train")
axes[1].plot(list(range(0, lambdas.size)), test_mse, label="test")
axes[1].legend(loc='upper left', fontsize=12)
axes[1].set_xlabel("Lambda", fontsize=14, fontname='Courier')
axes[1].set_ylabel("MSE", fontsize=14, fontname='Courier')
axes[1].set_title("Lambda vs MSE", y=1.02, fontsize=16, fontname='Courier')  #␣
 ↪Fixed title method
```

```
plt.tight_layout()
plt.subplots_adjust(hspace=0.3)
plt.subplots_adjust(wspace=0.5)
plt.show()
```



Testing the assumptions of the linear regression model

Assumption 1 : Multicollinearity check by VIF score (variables are dropped one-by-one till none has VIF>5)

[72]:
```
def vif(newdf):
    # VIF dataframe
    vif_data = pd.DataFrame()
    vif_data["feature"] = newdf.columns

    # calculating VIF for each feature
    vif_data["VIF"] = [variance_inflation_factor(newdf.values, i)
                              for i in range(len(newdf.columns))]
    return vif_data
```

[74]:
```
res = vif(jm.iloc[:,:-1])
res
```

[74]:

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

[75]:
```
# drop GRE Score and again calculate the VIF
res = vif(jm.iloc[:, 1:-1])
res
```

[75]:
```
             feature         VIF
0        TOEFL Score  639.741892
1  University Rating   19.884298
2                SOP   33.733613
3               LOR    30.631503
4               CGPA  728.778312
5           Research    2.863301
```

[77]:
```
# # drop TOEFL Score and again calculate the VIF
res = vif(jm.iloc[:,2:-1])
res
```

[77]:
```
             feature        VIF
0  University Rating  19.777410
1                SOP  33.625178
2               LOR   30.356252
3               CGPA  25.101796
4           Research   2.842227
```

[79]:
```
# Now lets drop the SOP and again calculate VIF
res = vif(jm.iloc[:,2:-1].drop(columns=['SOP']))
res
```

[79]:
```
             feature        VIF
0  University Rating  15.140770
1               LOR   26.918495
2               CGPA  22.369655
3           Research   2.819171
```

[80]:
```
# lets drop the LOR as well
newdf = jm.iloc[:,2:-1].drop(columns=['SOP'])
newdf = newdf.drop(columns=['LOR '], axis=1)
res = vif(newdf)
res
```

[80]:
```
             feature        VIF
0  University Rating  12.498400
1               CGPA  11.040746
2           Research   2.783179
```

[81]:
```
# drop the University Rating
newdf = newdf.drop(columns=['University Rating'])
res = vif(newdf)
res
```

[81]:
```
  feature       VIF
0    CGPA  2.455008
```

```
1  Research  2.455008
```

Assumption 2 : The mean of residuals is nearly zero

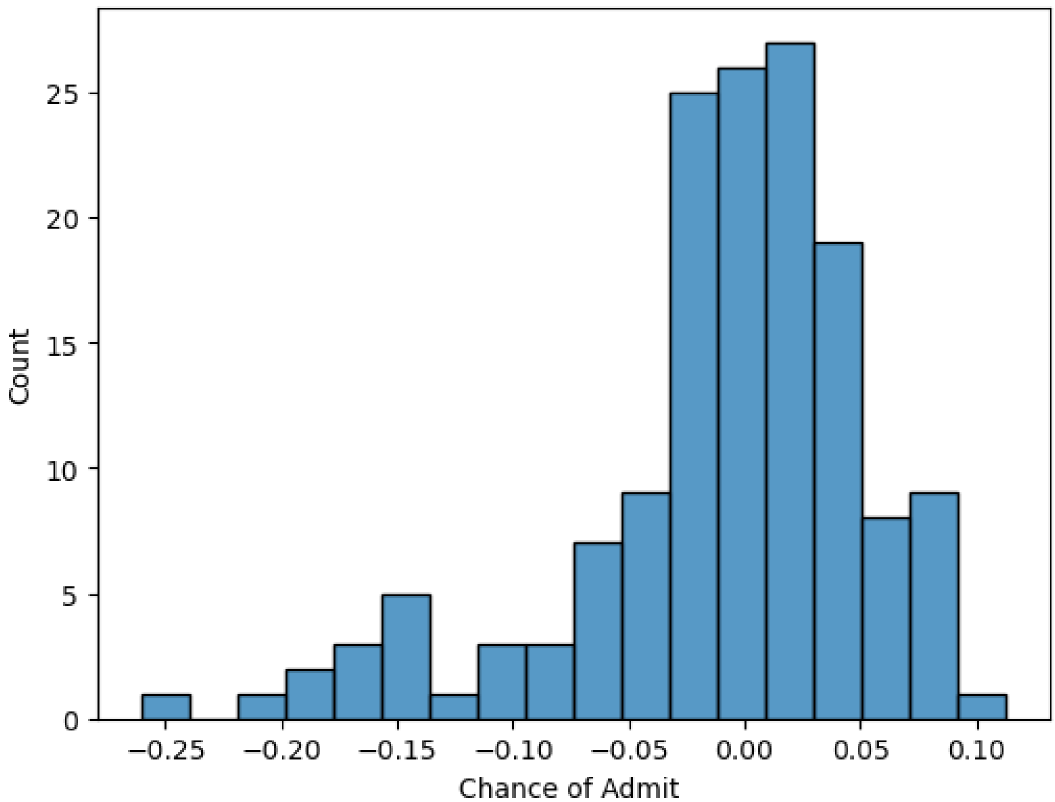
[89]:
```
model = LinearRegression()
model.fit(X_train, Y_train)

Y_train_pred = model.predict(X_train)
print("Mean of residual for training :", np.round(np.mean(np.power(Y_train_pred␣
↪- Y_train,2)),4))

Y_test_pred = model.predict(X_test)
print("Mean of residual for test :", np.round(np.mean(np.power(Y_test -␣
↪Y_test_pred,2)),4))
```

```
Mean of residual for training : 0.0041
Mean of residual for test : 0.0043
```

Assumption 3 : Linearity of variables It is quite clear from EDA that independent variables are linearly dependent on the target variables.

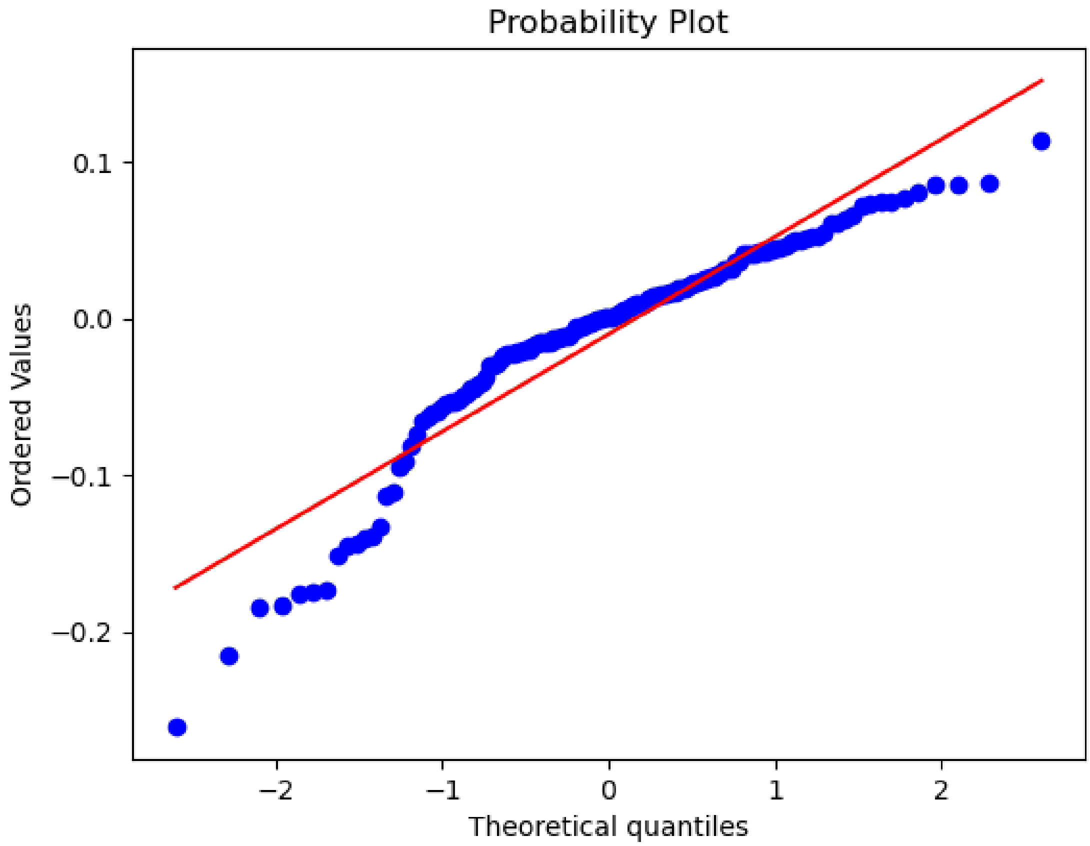
Assumption 4: Normality of Residuals

[90]:
```
y_pred = model.predict(X_test)
residuals = (Y_test - y_pred)
sns.histplot(residuals)
plt.show()
```



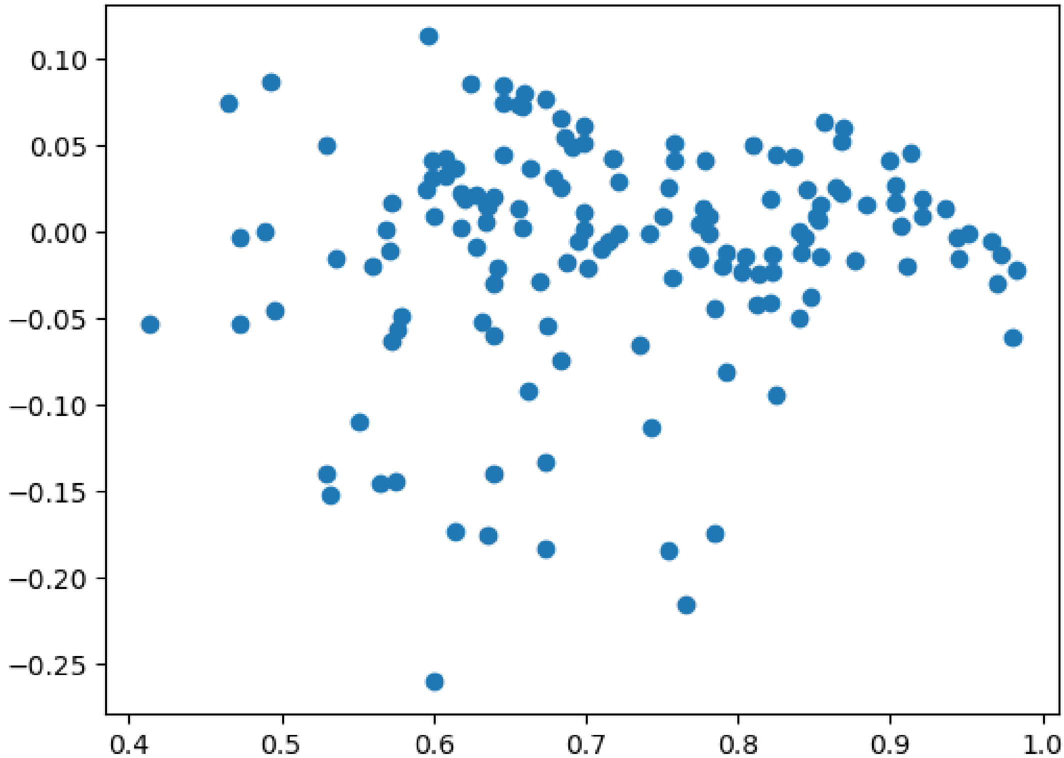
[91]:

```
stats.probplot(residuals, plot=plt)
plt.show()
```



Assumption 5: Test for Homoscedasticity

[92]:
```
plt.scatter(y_pred, residuals)
plt.show()
```



Insights & Recommendation

Insights:

The data exhibits multicollinearity. Following the removal of collinear features, only two variables remain significant in predicting the target variable. The independent variables demonstrate linear correlation with the dependent variable. Recommendations:

It is suggested that CGPA and Research are the sole important variables in predicting the Chance of Admit. CGPA emerges as the most influential variable in predicting the Chance of Admit. Final model results on the test data:

Root Mean Squared Error (RMSE): 0.07 Mean Absolute Error (MAE): 0.05 R-squared Score (R2_score): 0.81 Adjusted R-squared Score (Adjusted_R2): 0.80

[ ]:

```

```

[ ]:

```

```

[ ]:

```

```