Jamboree

April 17, 2022

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
import statsmodels.api as sm

pd.set_option('display.max_columns',None)
import warnings
warnings.filterwarnings('ignore')
```

1 Jamboree Education Case Study

[2]:
```
df = pd.read_csv("data/Jamboree_Admission.csv")
```

[3]:
```
df.head()
```

[3]:

	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

2 1. Define Problem Statement and perform Exploratory Data Analysis

2.1 Problem Statement:

Jamboree 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. The following are the main goals of our analysis :

1. Help Jamboree in understanding what factors are important in graduate admissions
2. How are these factors interrelated among themselves.
3. Help predict one's chances of admission given the rest of the variables.

[4]:
```
df.columns
```

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

2.2 Exploratory Data Analysis

2.2.1 Observations on shape of data, data types of all the attributes, conversion of categorical attributes to 'category', statistical summary

[5]:
```
df.shape
```

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

So we see that there are 500 rows and 9 columns.

[6]:
```
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
```

So we see that all are numerical columns.

Column Profiling:

- Serial No. (Unique row ID)
- GRE Scores (out of 340)
- TOEFL Scores (out of 120)
- University Rating (out of 5)
- Statement of Purpose (SOP) and Letter of Recommendation (LOR) Strength (out of 5)
- Undergraduate GPA (out of 10)
- Research Experience (either 0 or 1)
- Chance of Admit (ranging from 0 to 1)

We will drop the column 'Serial No.' as it is a unique identifier and does not provide any useful information for our analysis.

[7]:
```
df.drop(columns=['Serial No.'], inplace=True)
```

Conversion of categorical attributes to 'category'

[8]:
```
cat_cols = ['University Rating', 'SOP', 'LOR ','Research']

for col in cat_cols:
    df[col] = df[col].astype("category")
```

Statistical Summary

[9]:
```
df.describe().transpose()
```

[9]:

	count	mean	std	min	25%	50%	\
GRE Score	500.0	316.47200	11.295148	290.00	308.0000	317.00	
TOEFL Score	500.0	107.19200	6.081868	92.00	103.0000	107.00	
CGPA	500.0	8.57644	0.604813	6.80	8.1275	8.56	
Chance of Admit	500.0	0.72174	0.141140	0.34	0.6300	0.72	

	75%	max
GRE Score	325.00	340.00
TOEFL Score	112.00	120.00
CGPA	9.04	9.92
Chance of Admit	0.82	0.97

[10]:
```
df.describe(include=['category']).transpose()
```

[10]:

	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

So, now we have 8 columns of interest. Of these columns, 'Chance of Admit' is the response/target variable (whose values we are interested to predict) and the remaining 7 columns are independent predictor variables (using which we will predict the response).

2.2.2 Missing Value Detection

As we have already seen before, there does not seem to be any missing values. We will still check it below.

[11]:
```
percent = (df.isnull().sum()*100/df.isnull().count()).
 ↪sort_values(ascending=False)
percent_idx = percent.index
absolute = (df.isnull().sum())[percent_idx]

missing = pd.concat([percent, absolute], axis=1, keys = ['Percentage Missing',␣
 ↪'Total Missing'])
missing
```

[11]:

	Percentage Missing	Total Missing
GRE Score	0.0	0
TOEFL Score	0.0	0
University Rating	0.0	0
SOP	0.0	0
LOR	0.0	0
CGPA	0.0	0
Research	0.0	0
Chance of Admit	0.0	0

2.2.3 Unique values

[12]:
```
df.nunique()
```

[12]:
```
GRE Score            49
TOEFL Score          29
University Rating     5
SOP                   9
LOR                   9
CGPA                184
Research              2
Chance of Admit      61
dtype: int64
```

[13]:
```
for col in cat_cols:
    print("Unique Values for Column '",col,"' :", df[col].unique())
```

```
Unique Values for Column ' University Rating ' : [4, 3, 2, 5, 1]
Categories (5, int64): [1, 2, 3, 4, 5]
Unique Values for Column ' SOP ' : [4.5, 4.0, 3.0, 3.5, 2.0, 5.0, 1.5, 1.0, 2.5]
Categories (9, float64): [1.0, 1.5, 2.0, 2.5, …, 3.5, 4.0, 4.5, 5.0]
Unique Values for Column ' LOR  ' : [4.5, 3.5, 2.5, 3.0, 4.0, 1.5, 2.0, 5.0,
1.0]
Categories (9, float64): [1.0, 1.5, 2.0, 2.5, …, 3.5, 4.0, 4.5, 5.0]
```

```
Unique Values for Column ' Research ' : [1, 0]
Categories (2, int64): [0, 1]
```

[14]:
```
df.info()
```

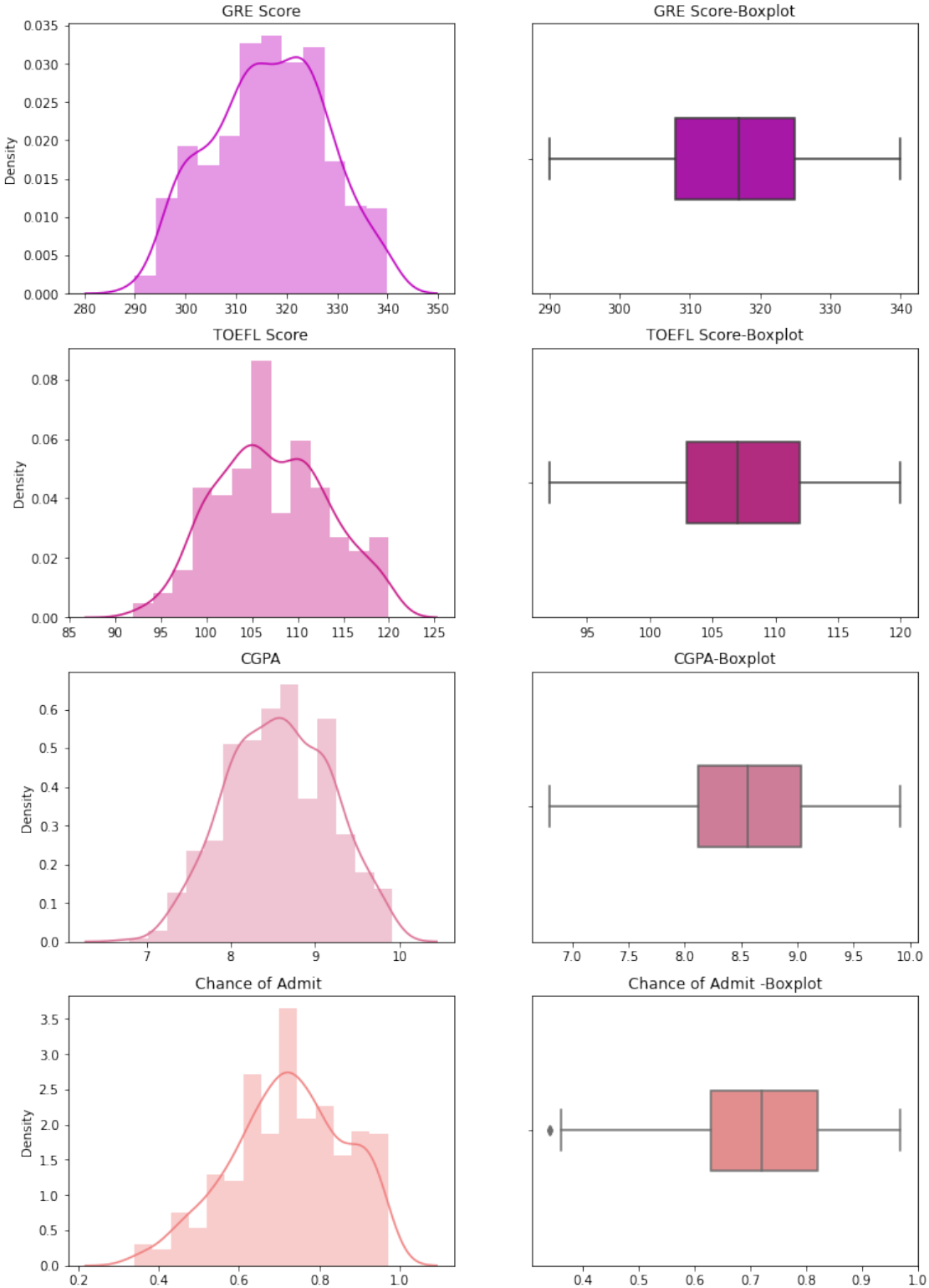
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   GRE Score          500 non-null    int64
 1   TOEFL Score        500 non-null    int64
 2   University Rating  500 non-null    category
 3   SOP                500 non-null    category
 4   LOR                500 non-null    category
 5   CGPA               500 non-null    float64
 6   Research           500 non-null    category
 7   Chance of Admit    500 non-null    float64
dtypes: category(4), float64(2), int64(2)
memory usage: 18.8 KB
```

2.2.4 Univariate Analysis

Numerical Columns - Univariate Visualizations

[15]:
```
num_cols = ['GRE Score', 'TOEFL Score', 'CGPA',  'Chance of Admit ']
colors = ['m','mediumvioletred','palevioletred','lightcoral']
fig, ax = plt.subplots(len(num_cols),2, figsize=(12,18))

for i in range(len(num_cols)):
    sns.distplot(df[num_cols[i]], ax=ax[i,0], color=colors[i])
    ax[i,0].set_title(num_cols[i])
    ax[i,0].set_xlabel('')
    sns.boxplot(df[num_cols[i]], width = 0.3, ax=ax[i,1], color=colors[i])
    ax[i,1].set_title(num_cols[i]+str("-Boxplot"))
    ax[i,1].set_xlabel('')

plt.suptitle("Univariate analysis of continuous variables")
plt.show()
```

Univariate analysis of continuous variables



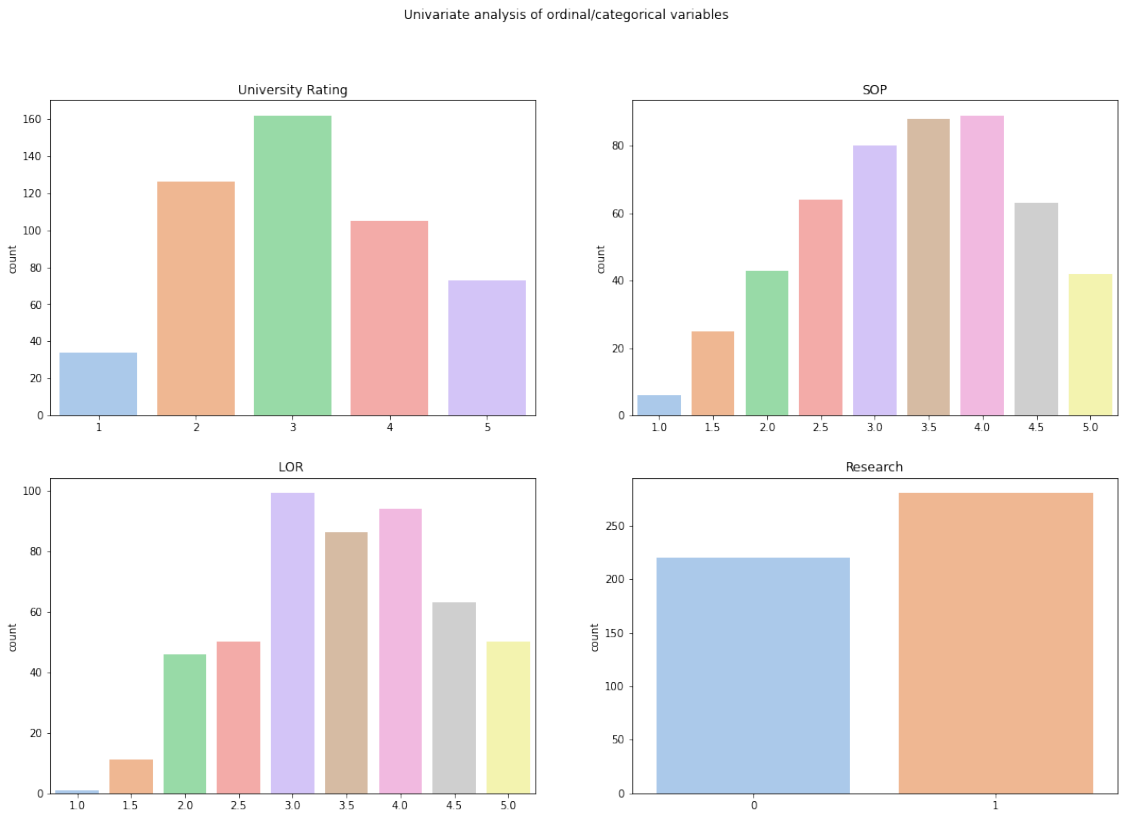
We observe the following: - Most GRE Scores lie between 310-325 - Most TOEFL Scores lie between 103-112 - Most CGPA lie between 8.2-9.0 - Most candidates end up getting prediction ranging from 60% to about 80%

Even though the variables 'University Rating', 'SOP', 'LOR' and 'Research' are not continuous, those variables are not purely cateogrical. They are ordinal and they have some inherent order (for eg. 5>4>3>2>1 for Univeristy Rating).

We will look at a few ways to treat those variables after the EDA is done.

Categorical Columns - Univariate Visualizations

[16]:
```
sns.set_palette(sns.color_palette("pastel"))

fig, ax = plt.subplots(2,2, figsize=(18,12))

for i in range(len(cat_cols)):
    sns.countplot(df[cat_cols[i]], ax=ax[i//2,i%2])
    ax[i//2,i%2].set_title(cat_cols[i])
    ax[i//2,i%2].set_xlabel('')

plt.suptitle("Univariate analysis of ordinal/categorical variables")
plt.show()
```



2.2.5 Bivariate and Multivariate Analysis

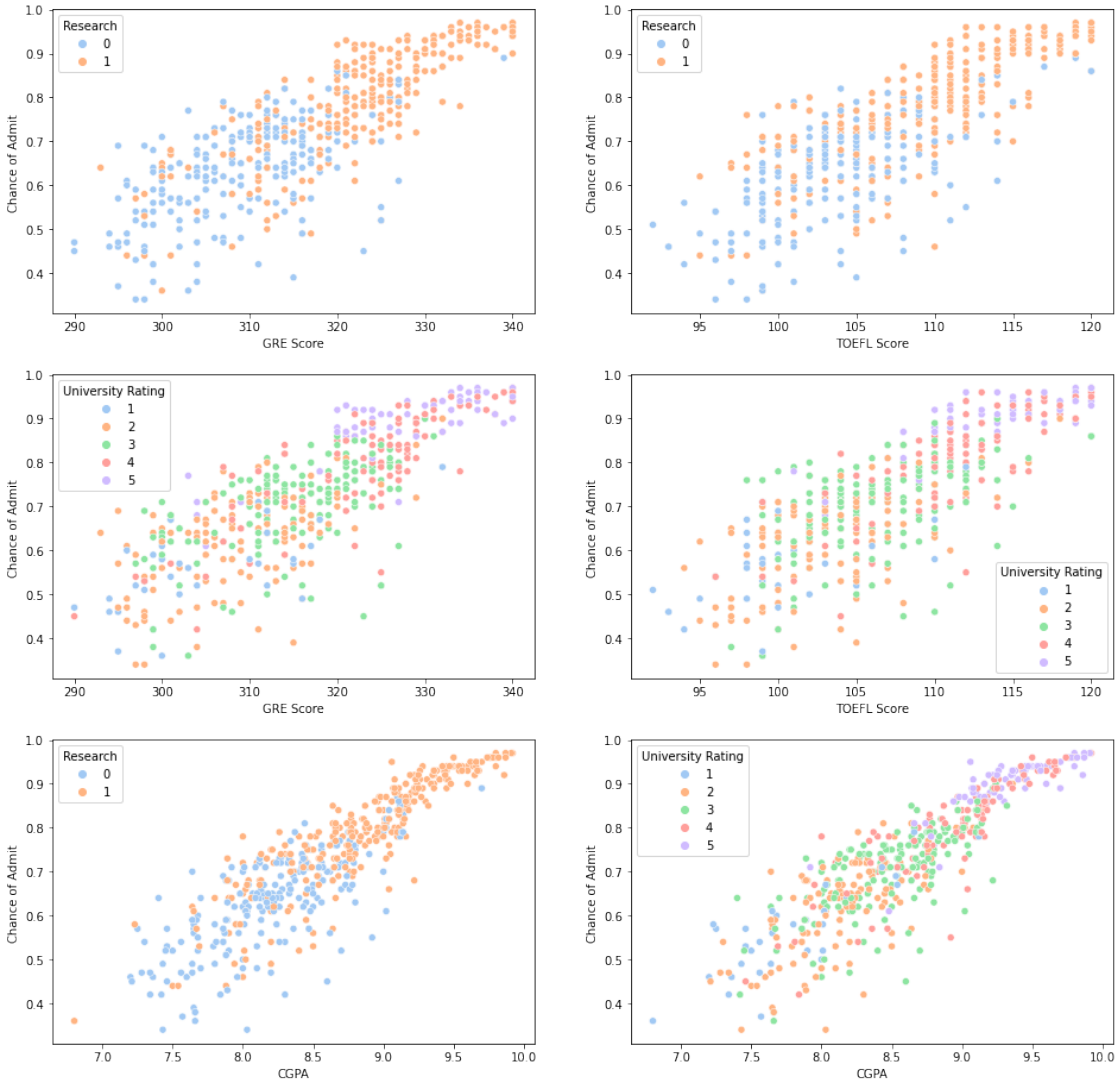
We observe the following : - Most data points are related to mid-tier universities (3), and not ambitious or safe option universities. - The self scored SOP Ratings tend to me mostly 3.5 or 4. - The self scored LOR Ratings similarly are mostly 3, 3.5 or 4. - More candidates have research experience than not.

[17]:
```
df.head()
```

[17]:
```
   GRE Score  TOEFL Score University Rating  SOP LOR   CGPA Research  \
0        337          118                 4  4.5  4.5  9.65        1
1        324          107                 4  4.0  4.5  8.87        1
2        316          104                 3  3.0  3.5  8.00        1
3        322          110                 3  3.5  2.5  8.67        1
4        314          103                 2  2.0  3.0  8.21        0

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

Since 'Chances of Admit' is the response/target variable, we will check how it depends on the other variables.

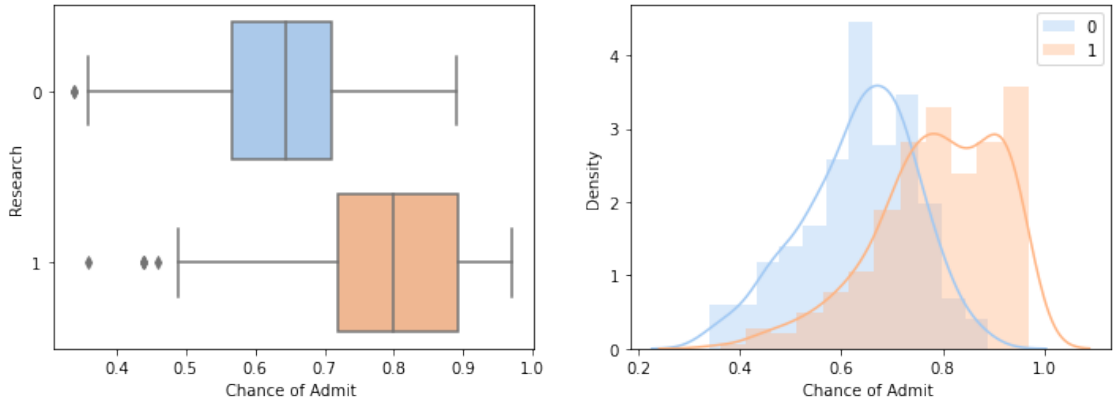
[18]:
```
fig, ax = plt.subplots(3,2, figsize=(16,16))
sns.scatterplot(x=df['GRE Score'], y=df['Chance of Admit '],␣
↪hue=df['Research'], ax=ax[0,0])
sns.scatterplot(x=df['TOEFL Score'], y=df['Chance of Admit '],␣
↪hue=df['Research'], ax=ax[0,1])
sns.scatterplot(x=df['GRE Score'], y=df['Chance of Admit '], hue=df['University␣
↪Rating'], ax=ax[1,0])
sns.scatterplot(x=df['TOEFL Score'], y=df['Chance of Admit '],␣
↪hue=df['University Rating'], ax=ax[1,1])
sns.scatterplot(x=df['CGPA'], y=df['Chance of Admit '], hue=df['Research'],␣
↪ax=ax[2,0])
sns.scatterplot(x=df['CGPA'], y=df['Chance of Admit '], hue=df['University␣
↪Rating'], ax=ax[2,1])
plt.show()
```



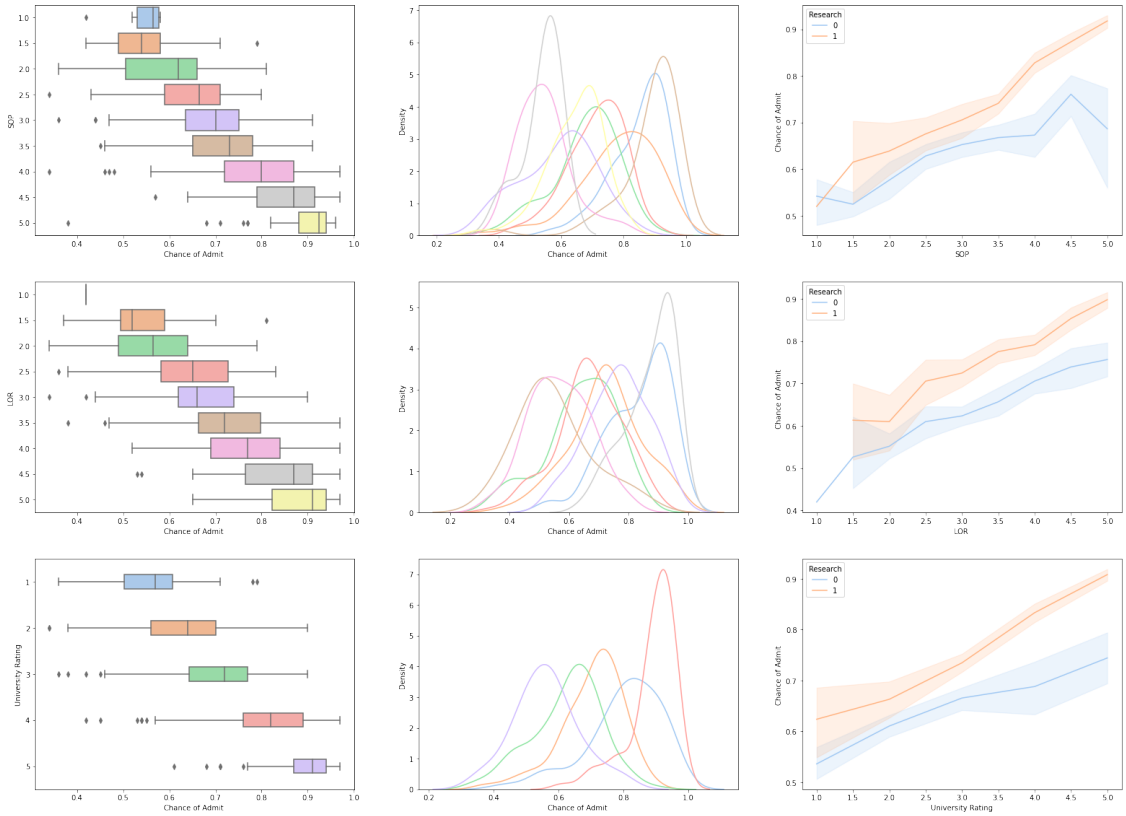
- As expected, research experience greatly increases the chances of getting admitted.
- Higher GRE, TOEFL scores and CGPA increase chance of getting admitted
- For the top most ranked universities (5 and 4), it seems that the chance of admit might never cross 80% if your scores are not high enough and you only get admit chances if your scores cross a certain threshold, below which you cannot get admitted. Or it could also be because those candidates who have very high scores, filter out only top ranked universities to calculate their chances for.

[19]:
```
fig, ax = plt.subplots(1,2, figsize=(12,4))

sns.boxplot(y=df['Research'], x=df['Chance of Admit '], orient='h', ax=ax[0])
sns.distplot(df[df['Research']==0]['Chance of Admit '], ax = ax[1], label='0')
sns.distplot(df[df['Research']==1]['Chance of Admit '], ax = ax[1], label='1')
ax[1].legend()
```

```
plt.show()
```

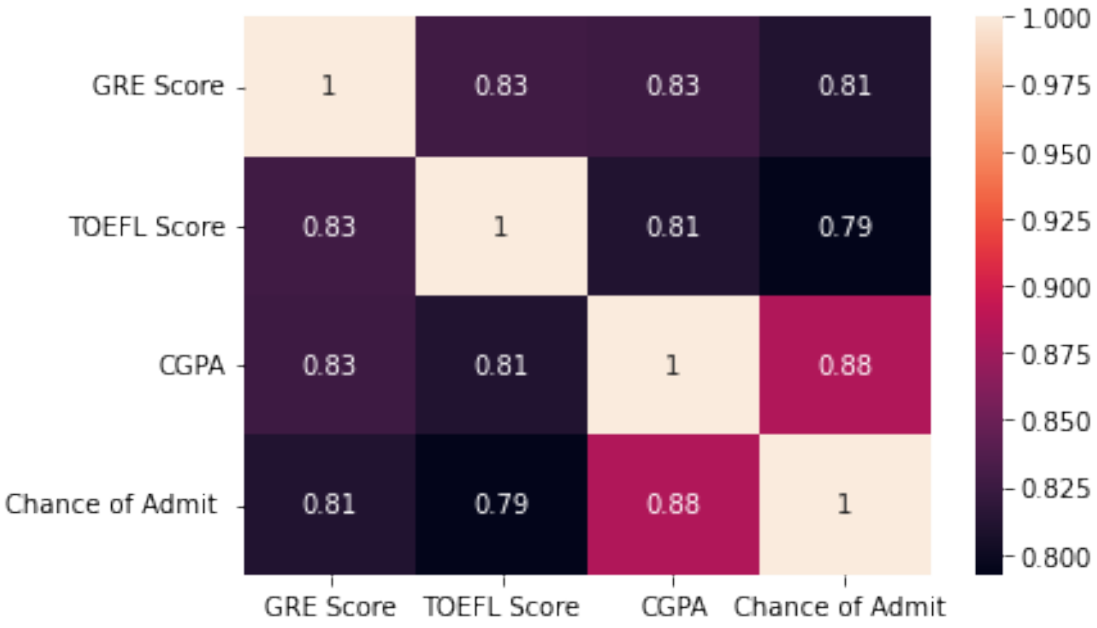


[20]:
```
def dist(df, col, hue, i, j):
    range_values = df[hue].unique()
    for val in range_values:
        sns.distplot(df[df[hue]==val][col], hist=False, ax = ax[i,j])

fig, ax = plt.subplots(3,3, figsize=(27,20))
sns.boxplot(y=df['SOP'], x=df['Chance of Admit '], orient='h', ax=ax[0,0])
dist(df, 'Chance of Admit ', 'SOP', 0, 1)
sns.lineplot(x=df['SOP'],y=df['Chance of Admit '],ax=ax[0,2],␣
 ↪hue=df['Research'])
sns.boxplot(y=df['LOR '], x=df['Chance of Admit '], orient='h', ax=ax[1,0])
dist(df, 'Chance of Admit ', 'LOR ', 1, 1)
sns.lineplot(x=df['LOR '],y=df['Chance of Admit '],ax=ax[1,2],␣
 ↪hue=df['Research'])
sns.boxplot(y=df['University Rating'], x=df['Chance of Admit '], orient='h',␣
 ↪width=0.3, ax=ax[2,0])
dist(df, 'Chance of Admit ', 'University Rating', 2, 1)
sns.lineplot(x=df['University Rating'],y=df['Chance of Admit '],ax=ax[2,2],␣
 ↪hue=df['Research'])
plt.show()
```



Correlation heatmap

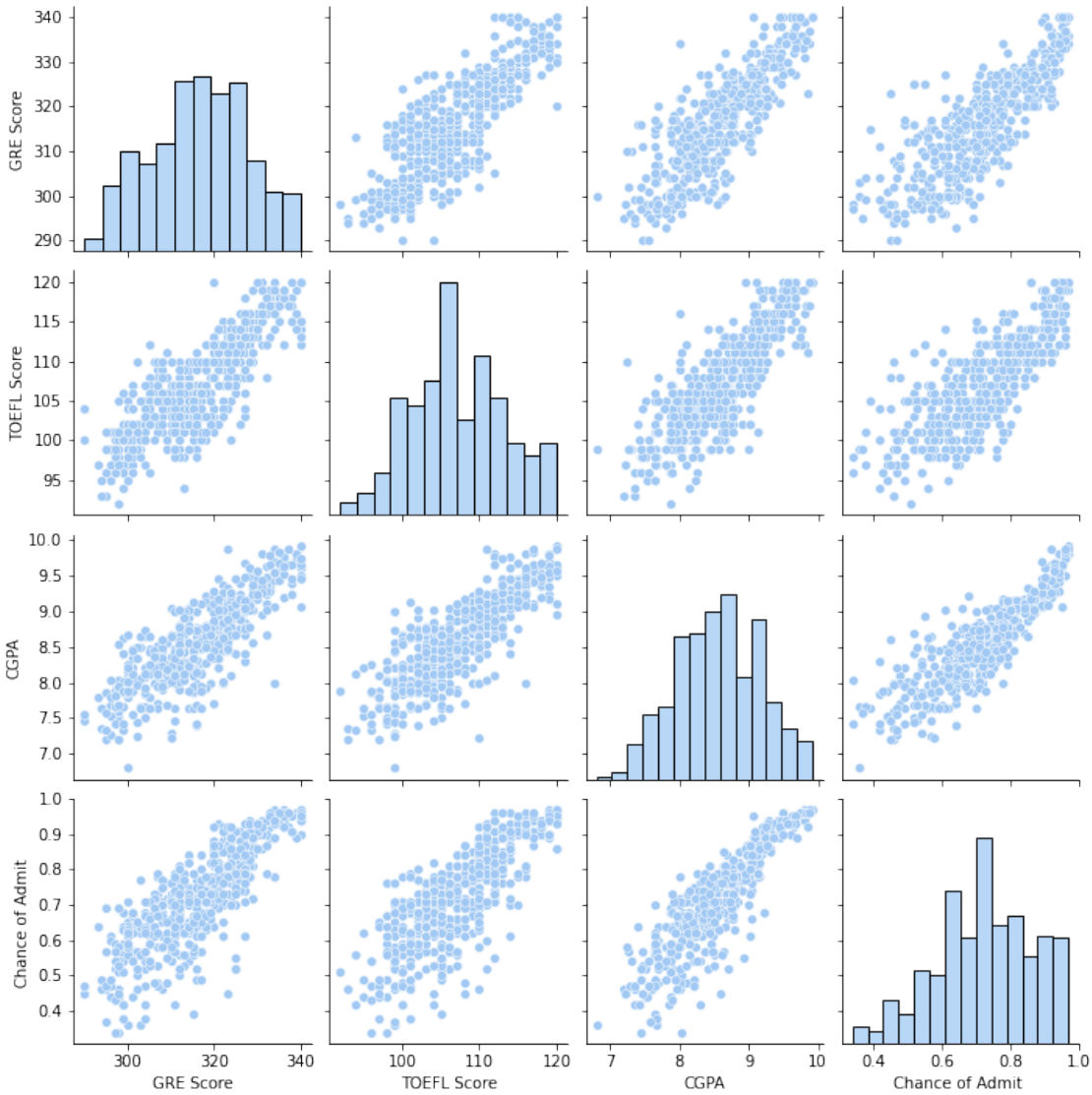
[21]:
```
sns.heatmap(df.corr(), annot=True)
plt.show()
```



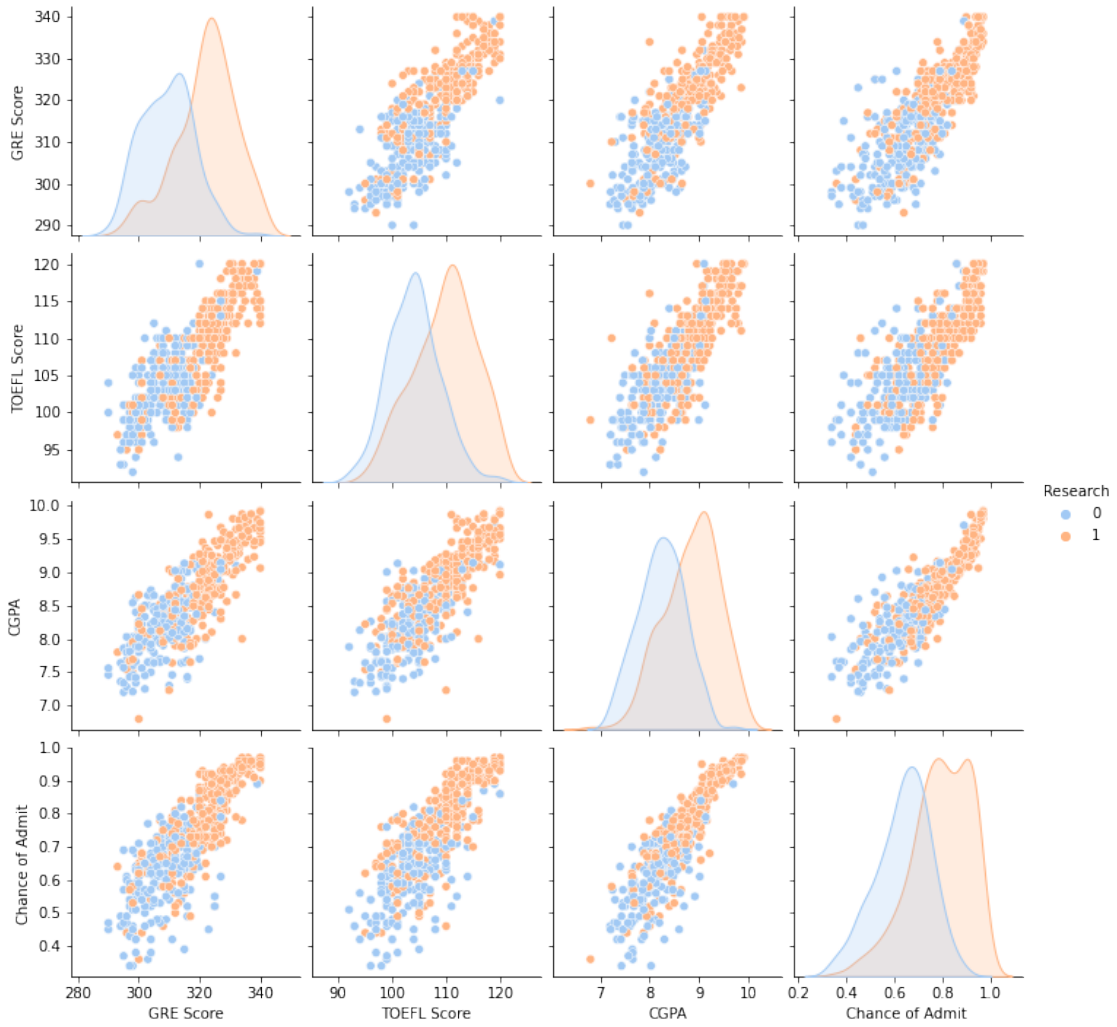
There is high correlation among variables such as GRE, TOEFL and CGPA.

Pairplots

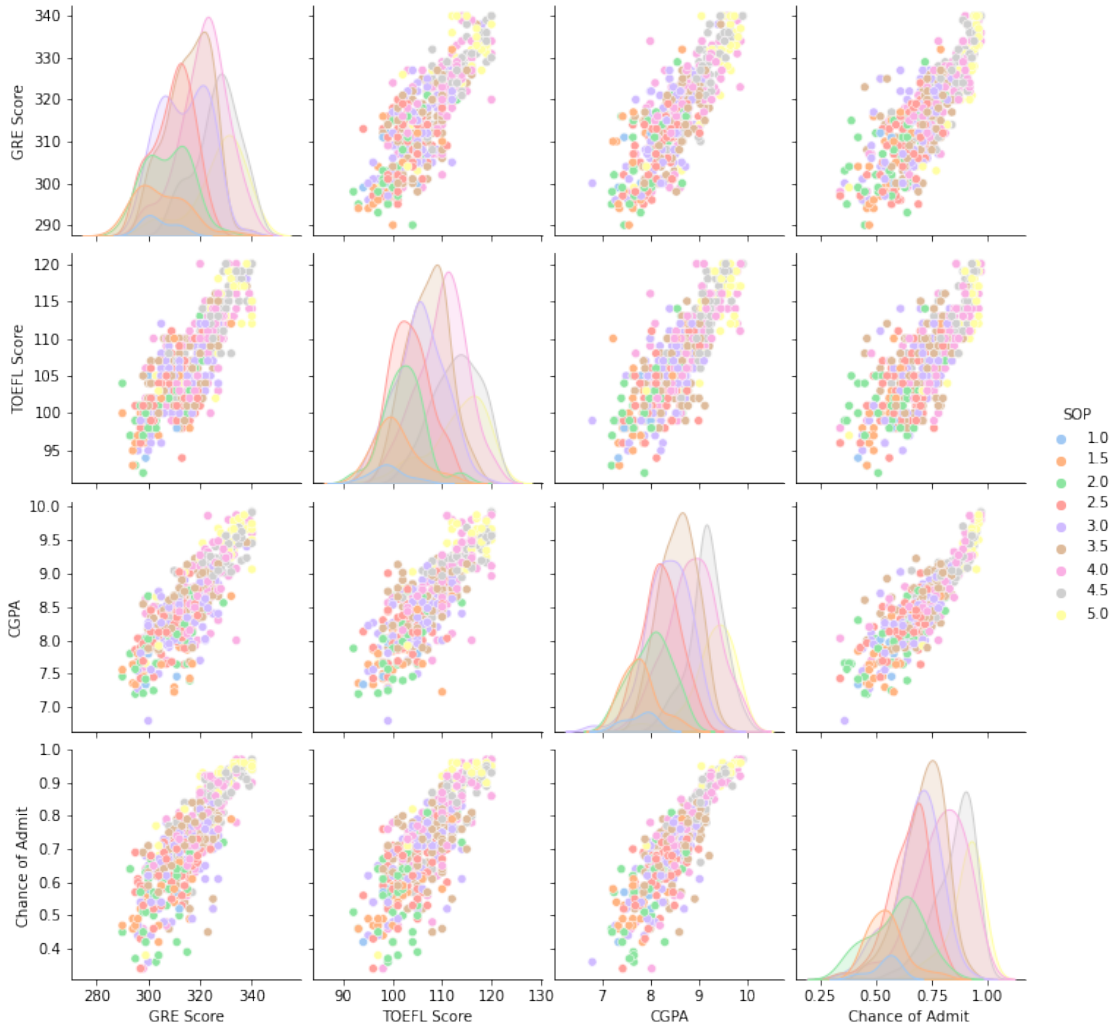
[22]:
```
sns.pairplot(df)
plt.show()
```



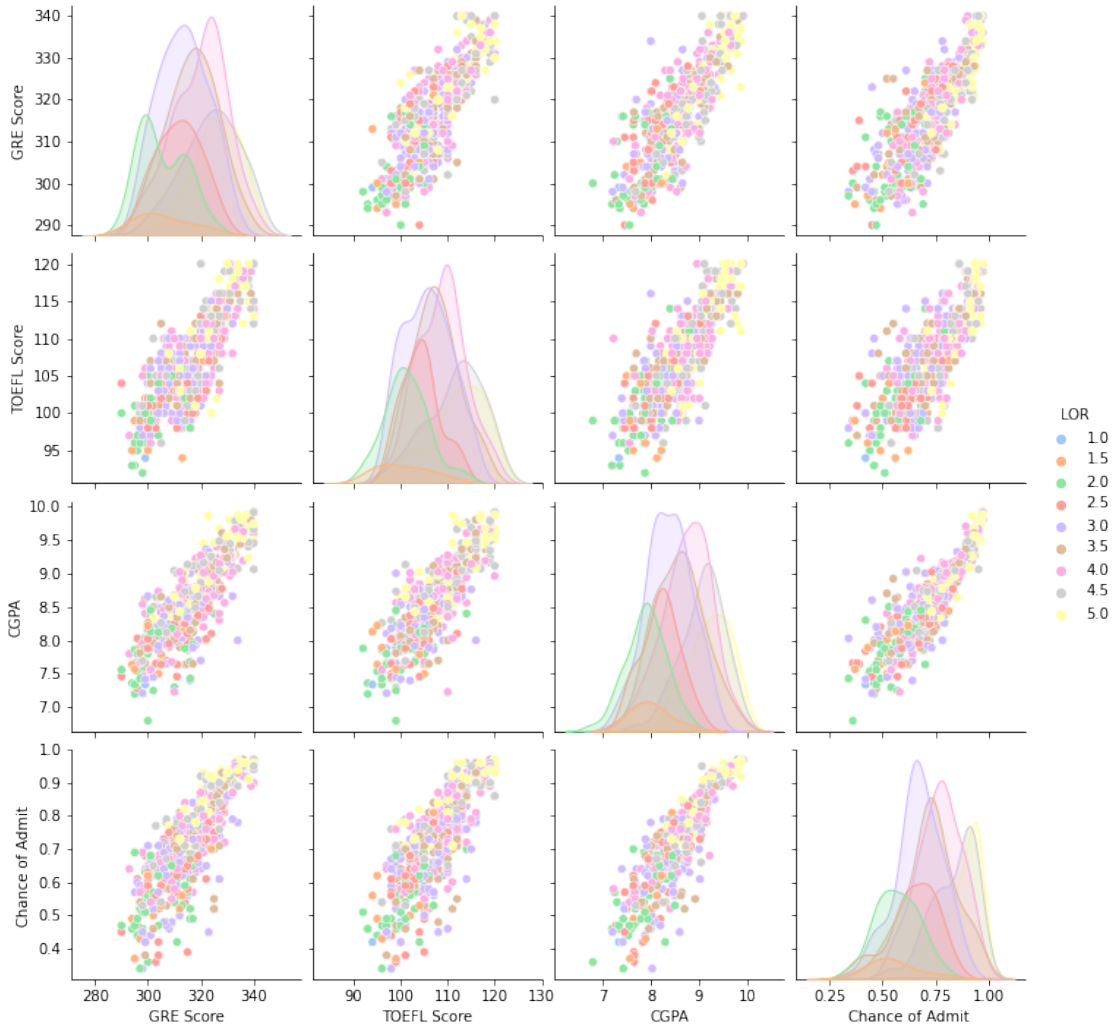
[23]:
```
sns.pairplot(df, hue='Research')
plt.show()
```



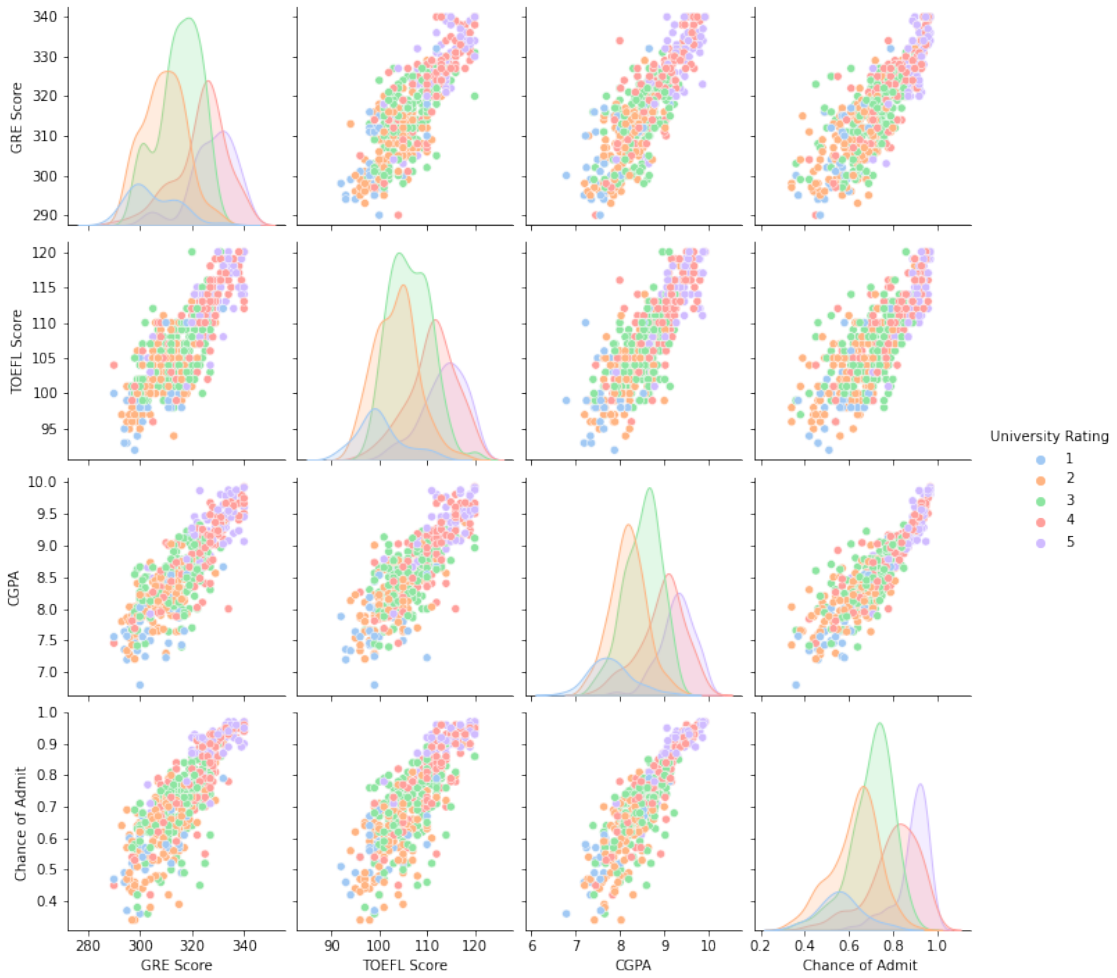
[24]:
```
sns.pairplot(df, hue="SOP")
plt.show()
```



[25]:
```
sns.pairplot(df, hue="LOR ")
plt.show()
```



[26]:
```
sns.pairplot(df, hue="University Rating")
plt.show()
```



3 2. Data Preprocessing

3.0.1 Duplicate Values Check

[27]:
```
df[df.duplicated()]
```

[27]:
```
Empty DataFrame
Columns: [GRE Score, TOEFL Score, University Rating, SOP, LOR , CGPA, Research,
Chance of Admit ]
Index: []
```

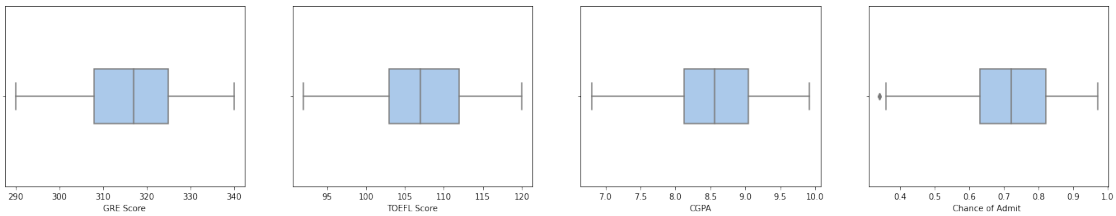
So, there are no duplicate values.

3.0.2 Missing Values

We have already checked in the previous section that there are no missing values in this dataset.

3.0.3 Outlier Detection and Handling

[28]:
```
fig, ax = plt.subplots(1,4,figsize=(24,4))
sns.boxplot(df['GRE Score'], ax=ax[0], width=0.3)
sns.boxplot(df['TOEFL Score'], ax=ax[1], width=0.3)
sns.boxplot(df['CGPA'], ax=ax[2], width=0.3)
sns.boxplot(df['Chance of Admit '], ax=ax[3], width=0.3)
plt.show()
```



So we see that there are very few outliers in the response variable. We will remove those.

[29]:
```
q25 = np.quantile(df['Chance of Admit '], 0.25)
q75 = np.quantile(df['Chance of Admit '], 0.75)
IQR = q75 - q25
drop_ind = (df['Chance of Admit ']< q25 - 1.5*IQR) | (df['Chance of Admit ']>␣
 ↪q75 + 1.5*IQR)
df[drop_ind]
```

[29]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
92	298	98	2	4.0	3.0	8.03	0	0.34
376	297	96	2	2.5	2.0	7.43	0	0.34

But if we remeber our plot for chances, we can remove anything which is less than or equal to 0.4, by considering those as outliers.

[30]:
```
drop_ind = df['Chance of Admit '] <= 0.4
df[drop_ind]
```

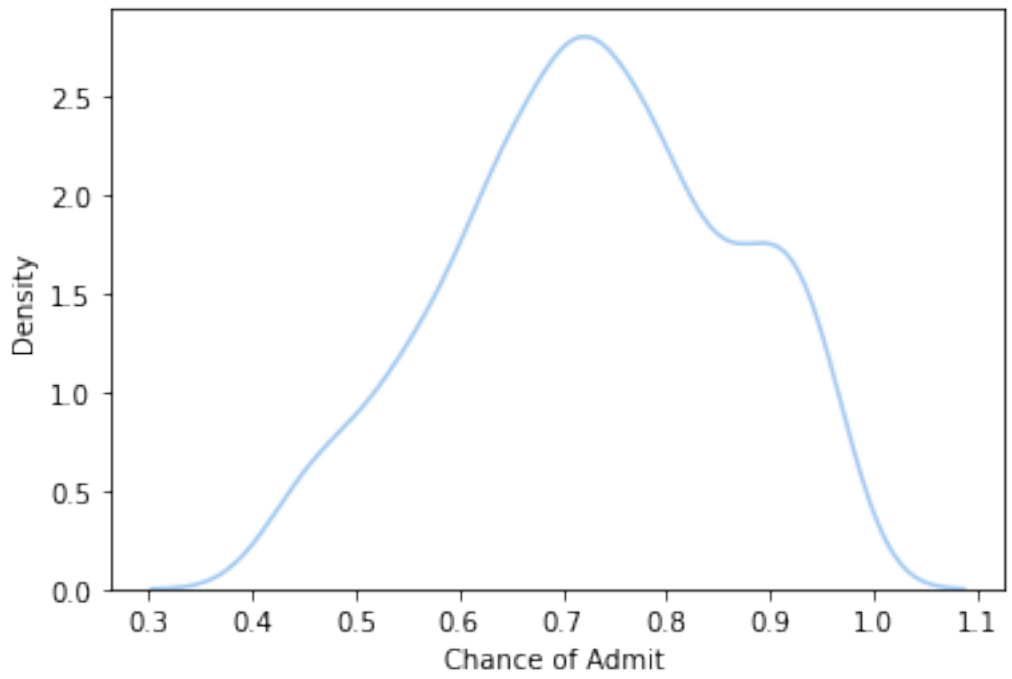
[30]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
58	300	99	1	3.0	2.0	6.80	1	0.36
91	299	97	3	5.0	3.5	7.66	0	0.39
92	298	98	2	4.0	3.0	8.03	0	0.34

```
94          303           99                  3  2.0  2.5  7.66         0
374         315          105                  2  2.0  2.5  7.65         0
375         304          101                  2  2.0  2.5  7.66         0
376         297           96                  2  2.5  2.0  7.43         0
457         295           99                  1  2.0  1.5  7.57         0

     Chance of Admit
58                0.36
91                0.38
92                0.34
94                0.36
374               0.39
375               0.38
376               0.34
457               0.37
```

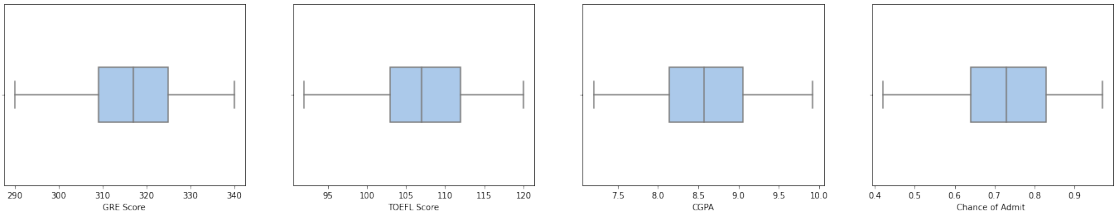
[31]:
```
df.drop(df[drop_ind].index, inplace=True)
```

[32]:
```
sns.distplot(df['Chance of Admit '], hist=False)
plt.show()
```



[33]:
```
fig, ax = plt.subplots(1,4,figsize=(24,4))
sns.boxplot(df['GRE Score'], ax=ax[0], width=0.3)
sns.boxplot(df['TOEFL Score'], ax=ax[1], width=0.3)
```

```
sns.boxplot(df['CGPA'], ax=ax[2], width=0.3)
sns.boxplot(df['Chance of Admit '], ax=ax[3], width=0.3)
plt.show()
```



There is only 1 observation with LOR rating of 1. This might also be treated as an outlier.

[34]:
```
df[df['LOR ']==1]
```

[34]:
```
     GRE Score  TOEFL Score University Rating  SOP LOR   CGPA Research  \
347        299           94                 1  1.0  1.0  7.34        0

     Chance of Admit
347             0.42
```

[35]:
```
df.drop(df[df['LOR ']==1].index, inplace=True)
df.shape
```

[35]:
```
(491, 8)
```

3.0.4 Feature Engineering and Data Preparation for Modeling.

[36]:
```
df.head(2)
```

[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

   Chance of Admit
0             0.92
1             0.76
```

[37]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 491 entries, 0 to 499
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   GRE Score          491 non-null    int64
```

```
 1   TOEFL Score        491 non-null    int64
 2   University Rating  491 non-null    category
 3   SOP                491 non-null    category
 4   LOR                491 non-null    category
 5   CGPA               491 non-null    float64
 6   Research           491 non-null    category
 7   Chance of Admit    491 non-null    float64
dtypes: category(4), float64(2), int64(2)
memory usage: 22.2 KB
```

There are 4 categorical variables : University Rating, SOP Rating, LOR Rating and Research.

Research :

[38]:
```
Research1 = pd.get_dummies(df['Research'], drop_first=True)
df['Research'] = Research1
```

[39]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 491 entries, 0 to 499
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   GRE Score          491 non-null    int64
 1   TOEFL Score        491 non-null    int64
 2   University Rating  491 non-null    category
 3   SOP                491 non-null    category
 4   LOR                491 non-null    category
 5   CGPA               491 non-null    float64
 6   Research           491 non-null    uint8
 7   Chance of Admit    491 non-null    float64
dtypes: category(3), float64(2), int64(2), uint8(1)
memory usage: 22.0 KB
```

We see that the data type of research changed because we are now treating is a dummy encoded variable. We only mneed n-1 dummy variables for categorical data having n categories. So here, n=2, and thus only 1 dummy variable will suffice.

University Rating

[40]:
```
df['University Rating'].unique()
```

[40]:
```
[4, 3, 2, 5, 1]
Categories (5, int64): [1, 2, 3, 4, 5]
```

So, we see that there are 5 categories here, however, the categories are inherently ordinal.

I have personally used the following resource as a guidance for my treatment of these ordinal variables: http://www.regorz-statistik.de/en/regression_ordinal_predictor.html

Method 1 : Staircase Encoding

[41]:
```
def staircase(x, lvl):
    if x > lvl:
        return 1
    else :
        return 0

df['UR1'] = df['University Rating'].apply(lambda x: staircase(x,1))
df['UR2'] = df['University Rating'].apply(lambda x: staircase(x,2))
df['UR3'] = df['University Rating'].apply(lambda x: staircase(x,3))
df['UR4'] = df['University Rating'].apply(lambda x: staircase(x,4))

df[['UR1','UR2','UR3','UR4']][0:10]
```

[41]:

	UR1	UR2	UR3	UR4
0	1	1	1	0
1	1	1	1	0
2	1	1	0	0
3	1	1	0	0
4	1	0	0	0
5	1	1	1	1
6	1	1	0	0
7	1	0	0	0
8	0	0	0	0
9	1	1	0	0

- UR1 encodes the difference between Lvl 1 and Lvl 2
- UR2 encodes the difference between Lvl 2 and Lvl 3
- UR3 encodes the difference between Lvl 3 and Lvl 4
- UR4 encodes the difference between Lvl 4 and Lvl 5

Now, if we try to fit an OLS model using these are predictors and the chance of admit as response and find that they are significant, we can keep this encoding otherwise we will reject these.

The possible problems with these are: - They introduce too many degrees of freedom and might reduce the power of the test. - Not all of them might be significant, so in that case we cannot use this encoding. - They might lead to overfitting.

Method 2:

We could simply treat our levels as interval (continuous). We do not need to make any additional features, but we can simply use the original feature as a predictor variable.

SOP (Statement of Purpose)

[42]:
```
df['SOP'].unique()
```

[42]:
```
[4.5, 4.0, 3.0, 3.5, 2.0, 5.0, 1.5, 1.0, 2.5]
Categories (9, float64): [1.0, 1.5, 2.0, 2.5, …, 3.5, 4.0, 4.5, 5.0]
```

I think, these values can be treated as interval and given the total number of additional features we would be introducing by encoding this ordinal feature, it might lead to overfitting since we have

only around 497 observations in the data.

LOR (Letters of Recommendation)

[43]:
```
df['LOR '].unique()
```

[43]:
```
[4.5, 3.5, 2.5, 3.0, 4.0, 1.5, 2.0, 5.0]
Categories (9, float64): [1.0, 1.5, 2.0, 2.5, …, 3.5, 4.0, 4.5, 5.0]
```

LOR Rating, once again, is similar to SOP, so we go ahead with our previous logic of treating it as interval.

[44]:
```
df.head(3)
```

[44]:

	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	

	Chance of Admit	UR1	UR2	UR3	UR4
0	0.92	1	1	1	0
1	0.76	1	1	1	0
2	0.72	1	1	0	0

Standard Scaling The variables like GRE,TOEFL can range in 300s or 100s, however some variables like research could be just 1 or 0. We could bring them under similar ranges by standardizing, but since we are not going to regularize our model, this step is not necessary.

So, We may or may not standardize. In my case, I do standardize my data.

[45]:
```
from sklearn.preprocessing import StandardScaler

X = df[df.columns]
X.drop(columns=['Chance of Admit '], inplace=True)
Y = df['Chance of Admit ']

sc = StandardScaler()
cols = X.columns
X[cols] = sc.fit_transform(X[cols])
```

Train and Test Split

[46]:
```
from sklearn.model_selection import train_test_split
import statsmodels.api as sm



X_c = sm.add_constant(X) #adding constant since we are going to use statsmodels␣
↪OLS
```

```
x_train,x_test,y_train, y_test = train_test_split(X_c,Y, test_size=0.
↪15,random_state = 1)
print("Training set shape X:", x_train.shape)
print("Training set shape Y:", y_train.shape)
print("Test set shape X:", x_test.shape)
print("Test set shape Y:", y_test.shape)
```

```
Training set shape X: (417, 12)
Training set shape Y: (417,)
Test set shape X: (74, 12)
Test set shape Y: (74,)
```

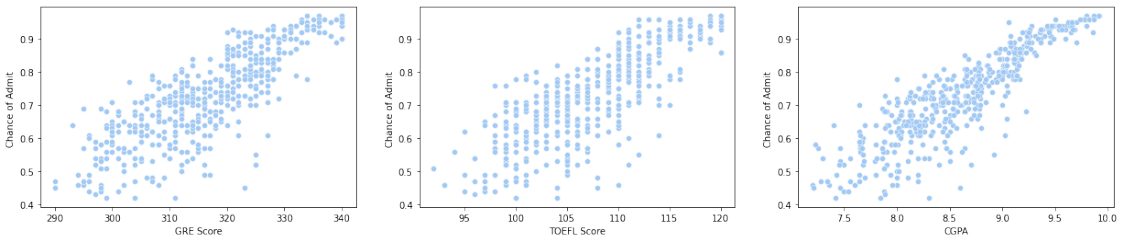
4 3. Model building

(Pre-) Assumptions: We will check the following assumptions before we even start building our model : 1. There should be a linear and additive relationship between dependent (response) variable and independent (predictor) variable(s). A linear relationship suggests that a change in response Y due to one unit change in X is constant, regardless of the value of X. An additive relationship suggests that the effect of X on Y is independent of other variables.

2. Not a necessary condition but good to check : Distribution of response variable : If it is not approximately normal, then we tend to have certain extreme errors.

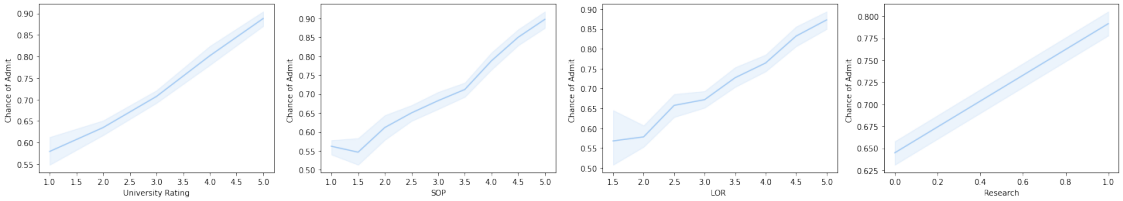
For visually checking the first assumption, we can plot the dependent variable vs independent variable graphs.

[47]:
```
indep_var_cont = ['GRE Score', 'TOEFL Score', 'CGPA']
indep_var_cat = ['University Rating', 'SOP', 'LOR ', 'Research']
dep_var =  'Chance of Admit '

fig, ax = plt.subplots(1,3, figsize=(21,4))
for i in range(len(indep_var_cont)):
    sns.scatterplot(x=df[indep_var_cont[i]], y=df[dep_var], ax=ax[i])

plt.show()
```



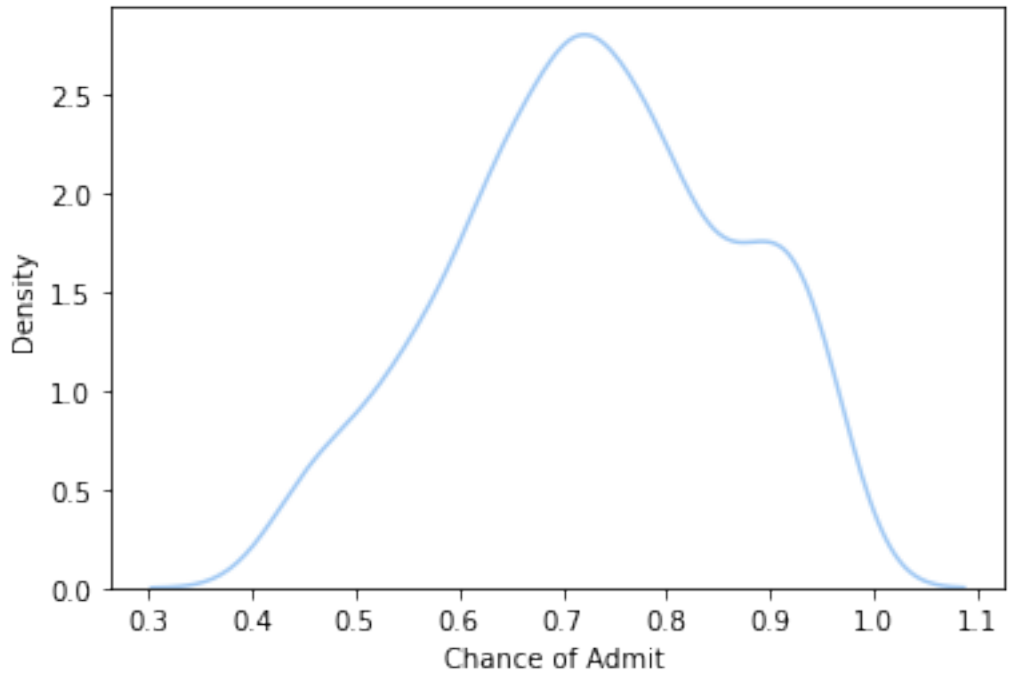
[48]:
```
fig, ax = plt.subplots(1,4, figsize=(25,4))
for i in range(len(indep_var_cat)):
```

```
    sns.lineplot(x=df[indep_var_cat[i]], y=df[dep_var], ax=ax[i])

plt.show()
```



So we see that the relationship between the independent variables and dependent variable might not be perfectly linear, but for practical purposes, we can consider them to be approximately linear.

[49]:
```
sns.distplot(df['Chance of Admit '], hist=False)
plt.show()
```



So, our target variable does not follow exact normal distribution, but for practical purposes we can assume it does approximately look normal. It is not extremely skewed or with any extreme outlier.

4.0.1 Build the Linear Regression model and comment on the model statistics

We still need to verify whether out staicase encoding for 'University Rating' variable is necessary. So we fit two versions of the model. One with only 'University Rating' (without UR1,2,3,4) and

another without 'University Rating' but including UR1,UR2,UR3,UR4 and check.

[50]:
```
sm_model0 = sm.OLS(y_train, x_train[x_train.columns.drop('University Rating')]).
↪fit()
```

[51]:
```
print(sm_model0.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       Chance of Admit   R-squared:                       0.831
Model:                           OLS   Adj. R-squared:                  0.826
Method:                Least Squares   F-statistic:                     199.0
Date:               Sun, 17 Apr 2022   Prob (F-statistic):          1.08e-149
Time:                       22:44:34   Log-Likelihood:                610.54
No. Observations:                417   AIC:                           -1199.
Df Residuals:                    406   BIC:                           -1155.
Df Model:                         10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.7276      0.003    261.491      0.000       0.722       0.733
GRE Score      0.0209      0.006      3.596      0.000       0.009       0.032
TOEFL Score    0.0180      0.005      3.312      0.001       0.007       0.029
SOP            0.0065      0.005      1.386      0.166      -0.003       0.016
LOR            0.0149      0.004      3.862      0.000       0.007       0.022
CGPA           0.0656      0.006     10.964      0.000       0.054       0.077
Research       0.0119      0.003      3.454      0.001       0.005       0.019
UR1           -0.0037      0.003     -1.155      0.249      -0.010       0.003
UR2            0.0022      0.004      0.605      0.546      -0.005       0.010
UR3           -0.0004      0.004     -0.092      0.926      -0.008       0.008
UR4            0.0072      0.003      2.079      0.038       0.000       0.014
==============================================================================
Omnibus:                      102.085   Durbin-Watson:                   1.889
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              261.511
Skew:                          -1.191   Prob(JB):                     1.64e-57
Kurtosis:                       6.061   Cond. No.                         5.97
==============================================================================

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

Display model coefficients with column names

[52]:
```
sm_model0.params
```

[52]:
```
const          0.727571
GRE Score      0.020949
```

```
TOEFL Score    0.018047
SOP            0.006548
LOR            0.014868
CGPA           0.065648
Research       0.011903
UR1           -0.003658
UR2            0.002236
UR3           -0.000380
UR4            0.007208
dtype: float64
```

We look at the P>|t| column. The null hypothesis tells us that the coefficent of that dependent variable is zero (in other words, that dependent variable is not important in predicting the response).

The following variables have p-values greater than 0.05 :

- UR1 : 0.789
- UR2 : 0.873
- UR3 : 0.538
- UR4 : 0.058
- SOP : 0.621

So, we see that these variables are not important in predicting the response. Moreover, we see that the UR's are not important either. So we can drop them and test another model with just 'University Rating'.

[53]:
```
sm_model1 = sm.OLS(y_train, x_train[x_train.columns.
↪drop(['UR1','UR2','UR3','UR4'])]).fit()
```

[54]:
```
print(sm_model1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       Chance of Admit   R-squared:                       0.828
Model:                           OLS   Adj. R-squared:                  0.825
Method:                Least Squares   F-statistic:                     281.9
Date:               Sun, 17 Apr 2022   Prob (F-statistic):          3.68e-152
Time:                       22:44:34   Log-Likelihood:                 607.77
No. Observations:                417   AIC:                            -1200.
Df Residuals:                    409   BIC:                            -1167.
Df Model:                          7
Covariance Type:           nonrobust
==============================================================================
=====
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-----
const          0.7276      0.003    261.024      0.000       0.722
0.733
```

```
GRE Score            0.0207      0.006      3.554      0.000       0.009
0.032
TOEFL Score          0.0182      0.005      3.331      0.001       0.007
0.029
University Rating    0.0052      0.004      1.151      0.250      -0.004
0.014
SOP                  0.0055      0.005      1.157      0.248      -0.004
0.015
LOR                  0.0152      0.004      3.942      0.000       0.008
0.023
CGPA                 0.0658      0.006     10.977      0.000       0.054
0.078
Research             0.0118      0.003      3.432      0.001       0.005
0.019
==============================================================================
Omnibus:                      107.847   Durbin-Watson:                   1.895
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              281.746
Skew:                          -1.251   Prob(JB):                     6.60e-62
Kurtosis:                       6.155   Cond. No.                         5.53
==============================================================================

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

Display model coefficients with column names

[55]:
```
sm_model1.params
```

[55]:
```
const              0.727552
GRE Score          0.020738
TOEFL Score        0.018178
University Rating  0.005156
SOP                0.005452
LOR                0.015209
CGPA               0.065802
Research           0.011803
dtype: float64
```

Once again we look t the P values. The following variables have p-values greater than 0.05 :

- University Rating : 0.120
- SOP : 0.704

So the University Rating variable itself is not much important for our analysis. The same goes for the SOP variable as well. So we will drop them after checking their VIFs.

But before that we drop the individual URs from our data.

[56]:
```
X.drop(columns=['UR1','UR2','UR3','UR4'], inplace=True)
X_c.drop(columns=['UR1','UR2','UR3','UR4'], inplace=True)
x_train.drop(columns=['UR1','UR2','UR3','UR4'], inplace=True)
x_test.drop(columns=['UR1','UR2','UR3','UR4'], inplace=True)
```

5 4. Testing the assumptions of the linear regression model

5.1 Multicollinearity Check

[57]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
X_t = X[X.columns]
vif['Features'] = X_t.columns
vif['VIF'] = [variance_inflation_factor(X_t.values, i) for i in range(X_t.
 ↪shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

[57]:

	Features	VIF
5	CGPA	4.69
0	GRE Score	4.34
1	TOEFL Score	3.79
3	SOP	2.90
2	University Rating	2.58
4	LOR	1.97
6	Research	1.48

Since VIF > 5 causes strong multicollinearity, and here we do not have any VIF > 5, we cannot drop columns from VIF score alone.

So, we drop SOP and University Rating from our previous analysis of p-values.

[58]:
```
X.drop(columns=['University Rating', 'SOP'], inplace=True)
X_c.drop(columns=['University Rating', 'SOP'], inplace=True)
x_train.drop(columns=['University Rating', 'SOP'], inplace=True)
x_test.drop(columns=['University Rating', 'SOP'], inplace=True)
```

We check our VIF scores again and then rerun a new OLS model.

[59]:
```
vif = pd.DataFrame()
X_t = X[X.columns]
vif['Features'] = X_t.columns
vif['VIF'] = [variance_inflation_factor(X_t.values, i) for i in range(X_t.
 ↪shape[1])]
```

```
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

[59]:

	Features	VIF
0	GRE Score	4.33
3	CGPA	4.27
1	TOEFL Score	3.68
2	LOR	1.66
4	Research	1.48

[60]:
```
sm_model2 = sm.OLS(y_train, x_train).fit()
print(sm_model2.summary())
```

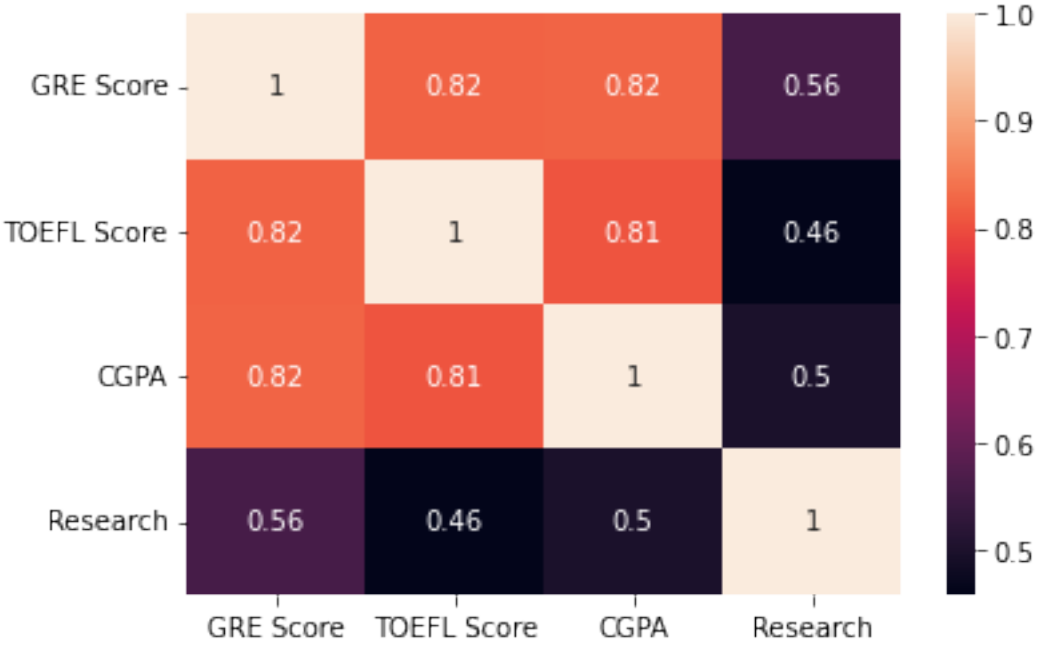
```
                            OLS Regression Results
==============================================================================
Dep. Variable:       Chance of Admit    R-squared:                       0.827
Model:                           OLS   Adj. R-squared:                  0.824
Method:                Least Squares   F-statistic:                     391.7
Date:               Sun, 17 Apr 2022   Prob (F-statistic):          7.44e-154
Time:                       22:44:35   Log-Likelihood:                 605.65
No. Observations:                417   AIC:                            -1199.
Df Residuals:                    411   BIC:                            -1175.
Df Model:                          5
Covariance Type:           nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.7275      0.003    260.332      0.000       0.722       0.733
GRE Score      0.0209      0.006      3.582      0.000       0.009       0.032
TOEFL Score    0.0199      0.005      3.690      0.000       0.009       0.031
LOR            0.0182      0.004      5.122      0.000       0.011       0.025
CGPA           0.0695      0.006     12.126      0.000       0.058       0.081
Research       0.0125      0.003      3.661      0.000       0.006       0.019
==============================================================================
Omnibus:                      102.187   Durbin-Watson:                   1.893
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              259.527
Skew:                          -1.196   Prob(JB):                     4.41e-57
Kurtosis:                       6.035   Cond. No.                         4.66
==============================================================================

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

[61]:
```
sm_model2.params
```

[61]:
```
const           0.727515
GRE Score       0.020914
TOEFL Score     0.019914
LOR             0.018217
CGPA            0.069481
Research        0.012538
dtype: float64
```

We no longer get the warning of possibility of presence of strong multi-collinearity and our VIF scores are also below 5 mostly.

[62]:
```
sns.heatmap(df[['GRE Score', 'TOEFL Score', 'LOR ', 'CGPA', 'Research']].
 ↪corr(), annot=True)
plt.show()
```



However, from the above heatmap, we see that some dependent variables are still correlated. Also, our R-squared value is 0.824 and adjusted R-squared value is 0.822. We can try dropping TOEFL Score and check.

[63]:
```
sm_model2 = sm.OLS(y_train, x_train[x_train.columns.drop(['TOEFL Score'])]).
 ↪fit()
print(sm_model2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       Chance of Admit    R-squared:                       0.821
Model:                           OLS    Adj. R-squared:                  0.819
```

```
Method:                 Least Squares   F-statistic:                     471.8
Date:                Sun, 17 Apr 2022   Prob (F-statistic):          2.58e-152
Time:                        22:44:35   Log-Likelihood:                 598.86
No. Observations:                 417   AIC:                            -1188.
Df Residuals:                     412   BIC:                            -1168.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.7276      0.003    256.471      0.000       0.722       0.733
GRE Score      0.0310      0.005      5.916      0.000       0.021       0.041
LOR            0.0189      0.004      5.256      0.000       0.012       0.026
CGPA           0.0770      0.005     14.151      0.000       0.066       0.088
Research       0.0121      0.003      3.469      0.001       0.005       0.019
==============================================================================
Omnibus:                       93.697   Durbin-Watson:                   1.878
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              226.364
Skew:                          -1.118   Prob(JB):                     7.01e-50
Kurtosis:                       5.834   Cond. No.                         4.08
==============================================================================

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

Our R-squared and adjusted R-squared have both dropped. So, we can keep TOEFL Score for our analysis.

[64]:
```
sm_model2 = sm.OLS(y_train, x_train).fit()
sm_model2.params
```

[64]:
```
const          0.727515
GRE Score      0.020914
TOEFL Score    0.019914
LOR            0.018217
CGPA           0.069481
Research       0.012538
dtype: float64
```

5.2 Mean of Residuals

[65]:
```
y_pred = sm_model2.predict(x_train).values
residuals = [y_train.values[i] - y_pred[i] for i in range(len(y_train))]
```

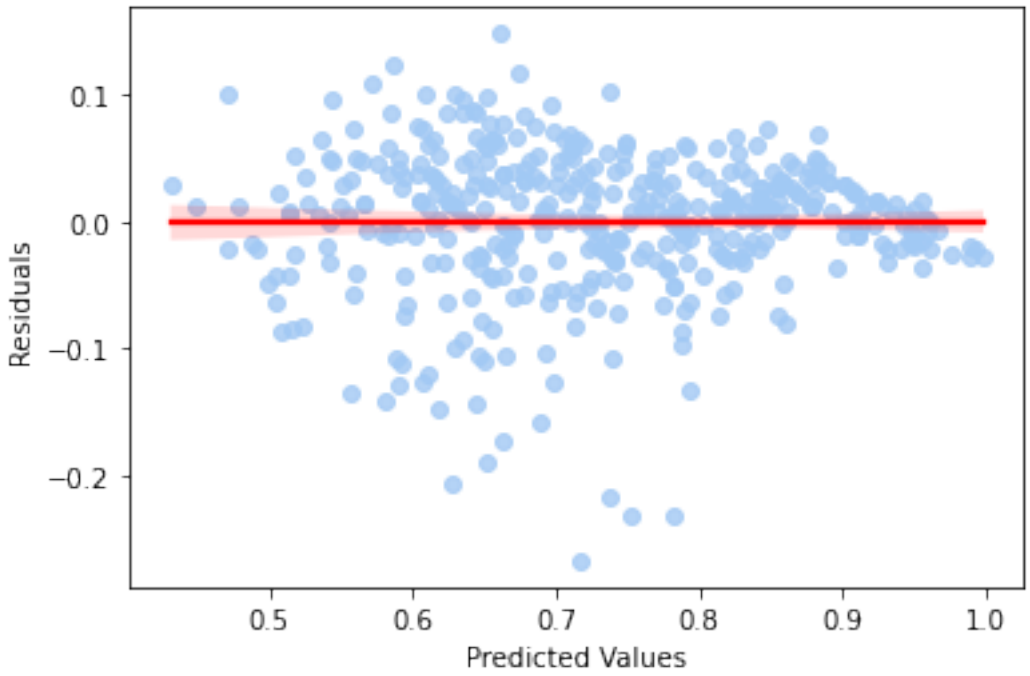
[66]:
```
np.mean(residuals)
```

[66]:
```
5.42065726171899e-16
```

So we see that the residuals are pretty small, and we can consider those to be close to zero.

5.3 Linearity of Variables and Homoscedasticity

[67]:
```
ax = sns.regplot(x=y_pred, y=residuals, line_kws={"color": "red"})
#plt.axhline(y = 0.0, color = 'red')
plt.ylabel("Residuals")
plt.xlabel("Predicted Values")
plt.show()
```

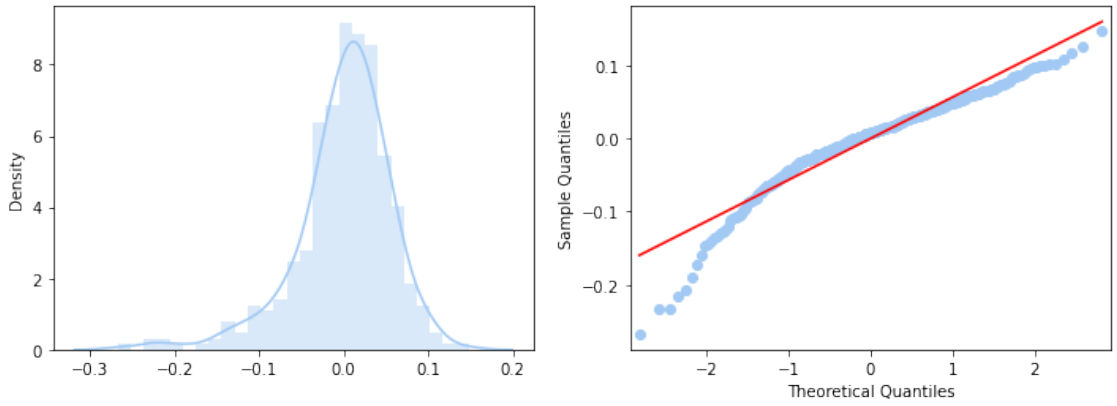


There does not seem to be any definitive pattern to the residuals. So, we can assume that linearity holds.

However, the variances are not homogeneous.The residuals do seem to have higher variance for lower values of response. They decrease as y_pred values increase. So there is heteroscedasticity.

5.3.1 Normality of Residuals

[68]:
```
fig,ax = plt.subplots(1,2, figsize=(12,4))

sns.distplot(residuals, ax=ax[0])
sm.qqplot(np.array(residuals), line='s', ax=ax[1])
plt.show()
```



According to this article, "If the error terms are non- normally distributed, confidence intervals may become too wide or narrow. Once confidence interval becomes unstable, it leads to difficulty in estimating coefficients based on minimization of least squares. Presence of non – normal distribution suggests that there are a few unusual data points which must be studied closely to make a better model."

Our residuals don't seem to follow exact normal distribution, but can be assumed to be approximately normal for practical purposes.

6 5. Model Performance Evaluation

6.0.1 R squared and Adjusted R squared.

[69]:
```
print(sm_model2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       Chance of Admit   R-squared:                       0.827
Model:                           OLS   Adj. R-squared:                  0.824
Method:                Least Squares   F-statistic:                     391.7
Date:               Sun, 17 Apr 2022   Prob (F-statistic):          7.44e-154
Time:                       22:44:36   Log-Likelihood:                 605.65
No. Observations:                417   AIC:                            -1199.
Df Residuals:                    411   BIC:                            -1175.
Df Model:                          5
Covariance Type:           nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.7275      0.003    260.332      0.000       0.722       0.733
GRE Score      0.0209      0.006      3.582      0.000       0.009       0.032
TOEFL Score    0.0199      0.005      3.690      0.000       0.009       0.031
```

```
LOR              0.0182      0.004      5.122      0.000       0.011       0.025
CGPA             0.0695      0.006     12.126      0.000       0.058       0.081
Research         0.0125      0.003      3.661      0.000       0.006       0.019
==============================================================================
Omnibus:                      102.187   Durbin-Watson:                   1.893
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              259.527
Skew:                          -1.196   Prob(JB):                     4.41e-57
Kurtosis:                       6.035   Cond. No.                         4.66
==============================================================================

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

We see that the R^2 and adjusted R^2 on the training data are 0.827 and 0.824 respectively.

[70]:
```
y_pred_test = sm_model2.predict(x_test).values
```

[71]:
```
RSS = sum([(y_pred_test[i] - y_test.values[i])**2 for i in range(len(y_test))])
y_ = np.mean(y_test.values)
TSS = sum([(y_ - y_test.values[i])**2 for i in range(len(y_test))])

testR2 = 1 - (RSS/TSS)
n = len(y_test)
k = x_test.shape[1] - 1
adjusted_testR2 = 1 - (1-testR2)*(n-1)/(n-k-1)
```

[72]:
```
print(testR2, adjusted_testR2)
```

```
0.7735741905689617 0.75692523399315
```

So, the rsquared values on test data are lower than train. This indicates that our model does not generalize as well as we had hoped it would. One solution could be to in fact, go back and drop certain highly correlated variables to avoid overfitting.

6.0.2 MAE, RMSE

[73]:
```
mae_train = np.mean([abs(y_pred[i] - y_train.values[i]) for i in␣
 ↪range(len(y_train))])
mae_test = np.mean([abs(y_pred_test[i] - y_test.values[i]) for i in␣
 ↪range(len(y_test))])

print("Train MAE:", mae_train)
print("Test MAE:", mae_test)
```

```
Train MAE: 0.04102225996737387
Test MAE: 0.039925018686229864
```

[74]:
```
rmse_train = np.sqrt(np.mean([(y_pred[i] - y_train.values[i])**2 for i in␣
↪range(len(y_train))]))
rmse_test = np.sqrt(np.mean([(y_pred_test[i] - y_test.values[i])**2 for i in␣
↪range(len(y_test))]))

print("Train RMSE:", rmse_train)
print("Test RMSE:", rmse_test)
```

```
Train RMSE: 0.05662306551205834
Test RMSE: 0.0579716433386997
```

Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors.

In our case, $RMSE_{train} < RMSE_{test}$ but only marginally, however, $MAE_{train} > MAE_{test}$. This anomaly in expected trend of MAE might be because of our larger training data size. We only have around 70 observations for test.

7 6. Business Insights and Recommendations

Business Insights and Recommendations:

- Most GRE Scores lie between 310-325
- Most TOEFL Scores lie between 103-112
- Most CGPA lie between 8.2-9.0
- Most candidates end up getting prediction ranging from 60% to about 80%

1. We can try to suggest preparation and score improvement courses for TOEFL and GRE tailored to the candiates existing scores.
2. For those candidates that have low CGPA, they can be advised to focus on and strengthen other aspects of their application.

- Most data points are related to mid-tier universities (3), and not ambitious or safe option universities.
- The self scored SOP Ratings tend to me mostly 3.5 or 4.
- The self scored LOR Ratings similarly are mostly 3, 3.5 or 4.
- More candidates have research experience than not.

3. Candidates can be recommended more universities to add to their list for predicting chances at, since they seem to be preferring mid-tier universities more. Based on their stats, candidates can be suggested to add more options from both ambitious and safe categories.

4. The self scoring of SOP and LOR might be problematic, since it is highly subjective. Instead of providing just a single score, those can be divided into sub-metrics or scoring rubrics, so that candidates can better evaluate their own SOP and LORs. For example, questions like “Does your SOP address your motivation ?”, “Does your SOP explain your weak points and your reason for choosing the program?”, “Does your SOP identify potential advisors and/or research topics?”, “Does your LOR discuss your work ethic?”, “Are your recommenders academics or professionals?”.

5. Similarly instead of making just a single Yes or No anwer to research experience, it can also be divided into submetrics, such as number of papers, tier of conference, relevance to the program where you applied, etc.

- As expected, research experience greatly increases the chances of getting admitted.
- Higher GRE, TOEFL scores and CGPA increase chance of getting admitted
- For the top most ranked universities (5 and 4), it seems that the chance of admit might never cross 80% if your scores are not high enough and you only get admit chances if your scores cross a certain threshold, below which you cannot get admitted. Or it could also be because those candidates who have very high scores, filter out only top ranked universities to calculate their chances for.

6. Self scored SOP rating and University Ratings do not seem to be as important as the other variables while performing regression analysis. This does not mean that those metrics are inherently less useful, but, the interpretation of the metrics might be lost with the current system. If those metrics are calculated from more granular level details and universities are also classified not just on the basis of a single world rank, but on different basis (for eg. based on particular research areas, based on employability of graduates, based on research environment, availability of funding), then it might be more meaningful, since a university ranked as top for computer science might not necessarily be so for social sciences.

7. The relation between chance of admit and various predictor variables might not exactly be linear, so it might make more sense to not just design better features, but to also use some non-linear model when we wish to deploy it in the real world.

8. The company could also offer candidates the option to sign up and remind them about deadlines of application for the universities that the candidates shortlist.

9. Once the current appliation cycle is over and results are out, the company can encourage the candidates to update their actual results in the system, so that the company can actually get some real validation data.

10. Improving the current system of scoring metrics might not only attract more candidates to use the tool, but also encourage them to access other services provided by the company.

11. It would help the company to run targeted services for addressing various aspects of the application, depending on the candidate's current profile.

[ ]: