Importing basic libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
```

```
<frozen importlib._bootstrap>:228: RuntimeWarning: scipy._lib.messagestream.MessageStream size changed, may indicate binary incompatibility. Expected 56 from C header, got 64 from PyObject
```

In [2]:
```
# Reading the data
df = pd.read_csv('https://raw.githubusercontent.com/Jhoie/Admission-Prediction/main/Admission_Predict.csv
```

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

Out[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

In [4]:
```
df.shape
```

Out[4]:
```
(400, 9)
```

In [5]:
```
df.nunique()
```

Out[5]:
```
Serial No.           400
GRE Score             49
TOEFL Score           29
University Rating      5
SOP                    9
LOR                    9
CGPA                 168
Research               2
Chance of Admit       60
dtype: int64
```

In [6]:
```
df.columns
```

Out[6]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

In [7]:
```
# Removing the extra space in the column name

df.columns = [x.strip() for x in df.columns]
df.columns
```

Out[7]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR', 'CGPA', 'Research', 'Chance of Admit'],
      dtype='object')
```

In [8]:
```
# Checking for null values
df.isnull().sum()
```

Out[8]:
```
Serial No.           0
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

Statistical Analysis

In [9]:
```
df.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	75%	max
Serial No.	400.0	200.500000	115.614301	1.00	100.75	200.50	300.2500	400.00
GRE Score	400.0	316.807500	11.473646	290.00	308.00	317.00	325.0000	340.00
TOEFL Score	400.0	107.410000	6.069514	92.00	103.00	107.00	112.0000	120.00
University Rating	400.0	3.087500	1.143728	1.00	2.00	3.00	4.0000	5.00
SOP	400.0	3.400000	1.006869	1.00	2.50	3.50	4.0000	5.00
LOR	400.0	3.452500	0.898478	1.00	3.00	3.50	4.0000	5.00
CGPA	400.0	8.598925	0.596317	6.80	8.17	8.61	9.0625	9.92
Research	400.0	0.547500	0.498362	0.00	0.00	1.00	1.0000	1.00
Chance of Admit	400.0	0.724350	0.142609	0.34	0.64	0.73	0.8300	0.97

In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         400 non-null    int64  
 1   GRE Score          400 non-null    int64  
 2   TOEFL Score        400 non-null    int64  
 3   University Rating  400 non-null    int64  
 4   SOP                400 non-null    float64
 5   LOR                400 non-null    float64
 6   CGPA               400 non-null    float64
 7   Research           400 non-null    int64  
 8   Chance of Admit    400 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 28.2 KB
```

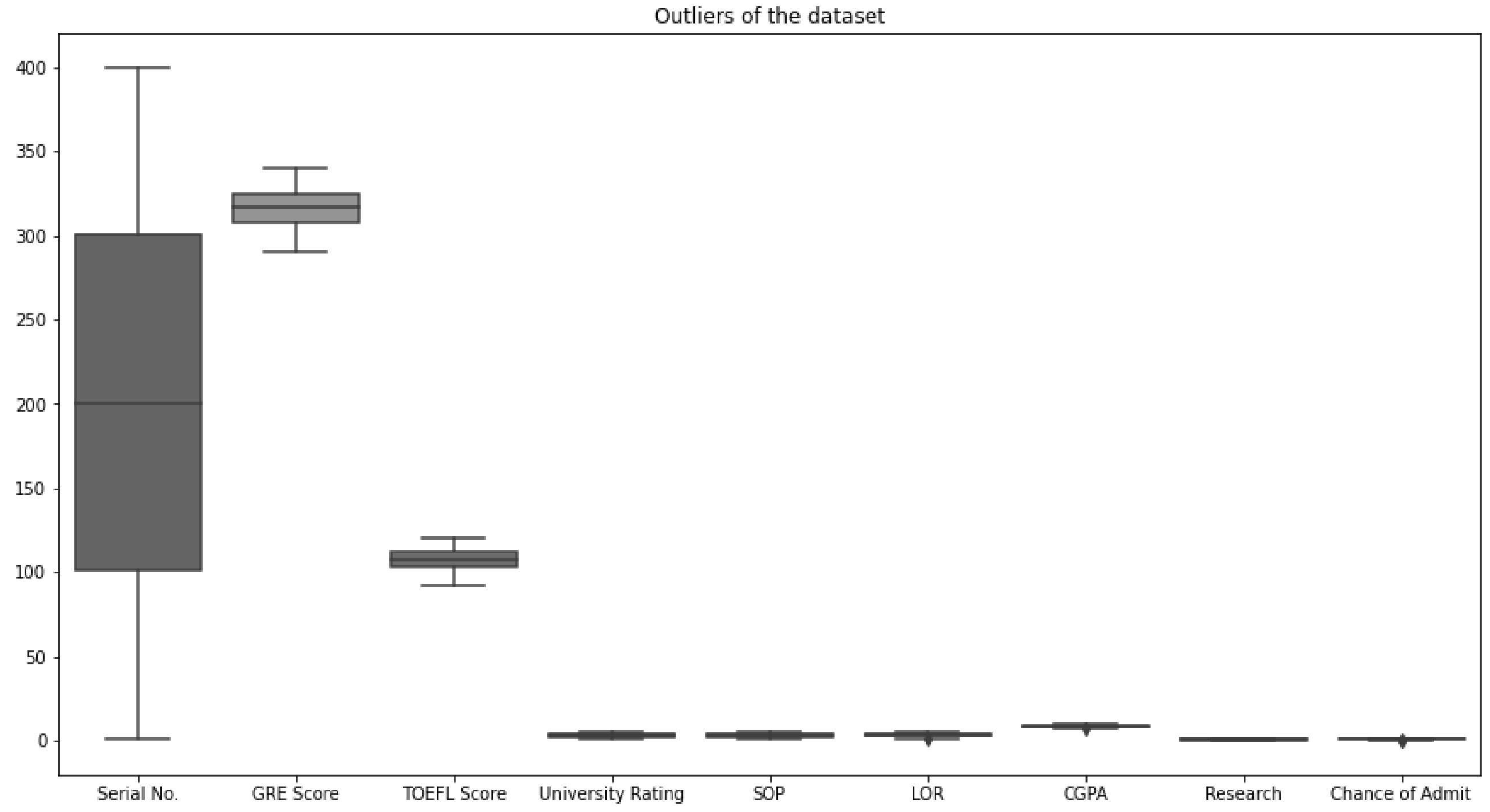
In [11]:
```
## Univariate Analysis
for i in df.columns:
    print('-----------')
    print(i)
    print('-----------')
    print(df[i].value_counts())
```

```
336    5
296    5
303    5
302    5
335    4
295    4
338    4
297    4
333    4
339    3
294    2
290    2
337    1
293    1
Name: GRE Score, dtype: int64
-----------
TOEFL Score
-----------
110    37
105    28
```

In [12]:

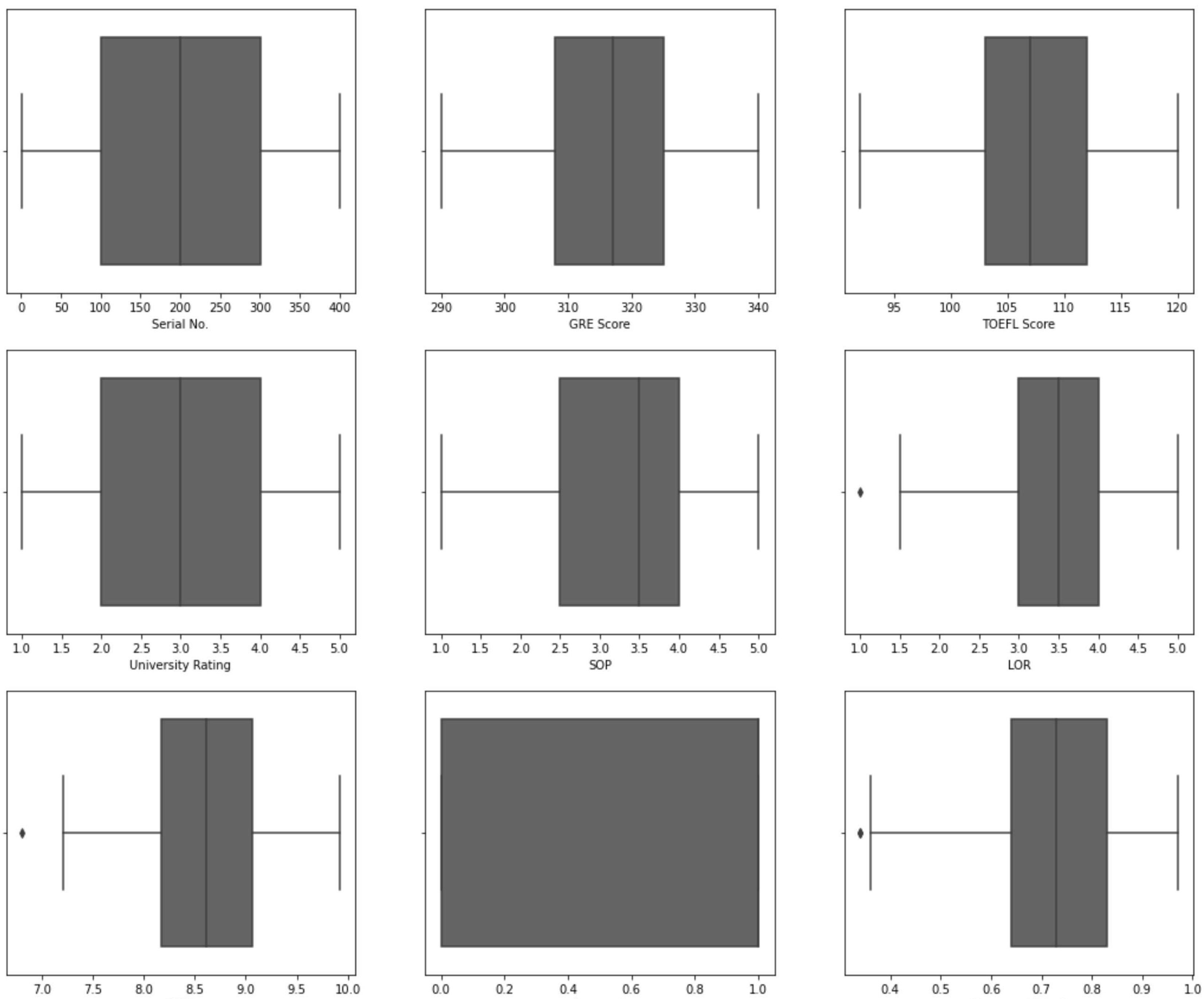
```
# checking the outlier

plt.figure(figsize=(15,8))
sns.boxplot(data = df)
plt.title('Outliers of the dataset')
```

Out[12]: Text(0.5, 1.0, 'Outliers of the dataset')



In [13]:

```
plt.figure(figsize = (18,15))
plotnumber=1
for i in df.columns:
    if plotnumber<=9:
        ax = plt.subplot(3,3,plotnumber)
        sns.boxplot(df[i])
    plotnumber+=1
plt.show()

'''

plt.figure(figsize=(18,15))
plotnumber=1
for i in df.columns:
    if plotnumber<=6:
        ax = plt.subplot(3,2,plotnumber)
        sns.boxplot(df[i])
    plotnumber+=1
plt.show()
'''
```

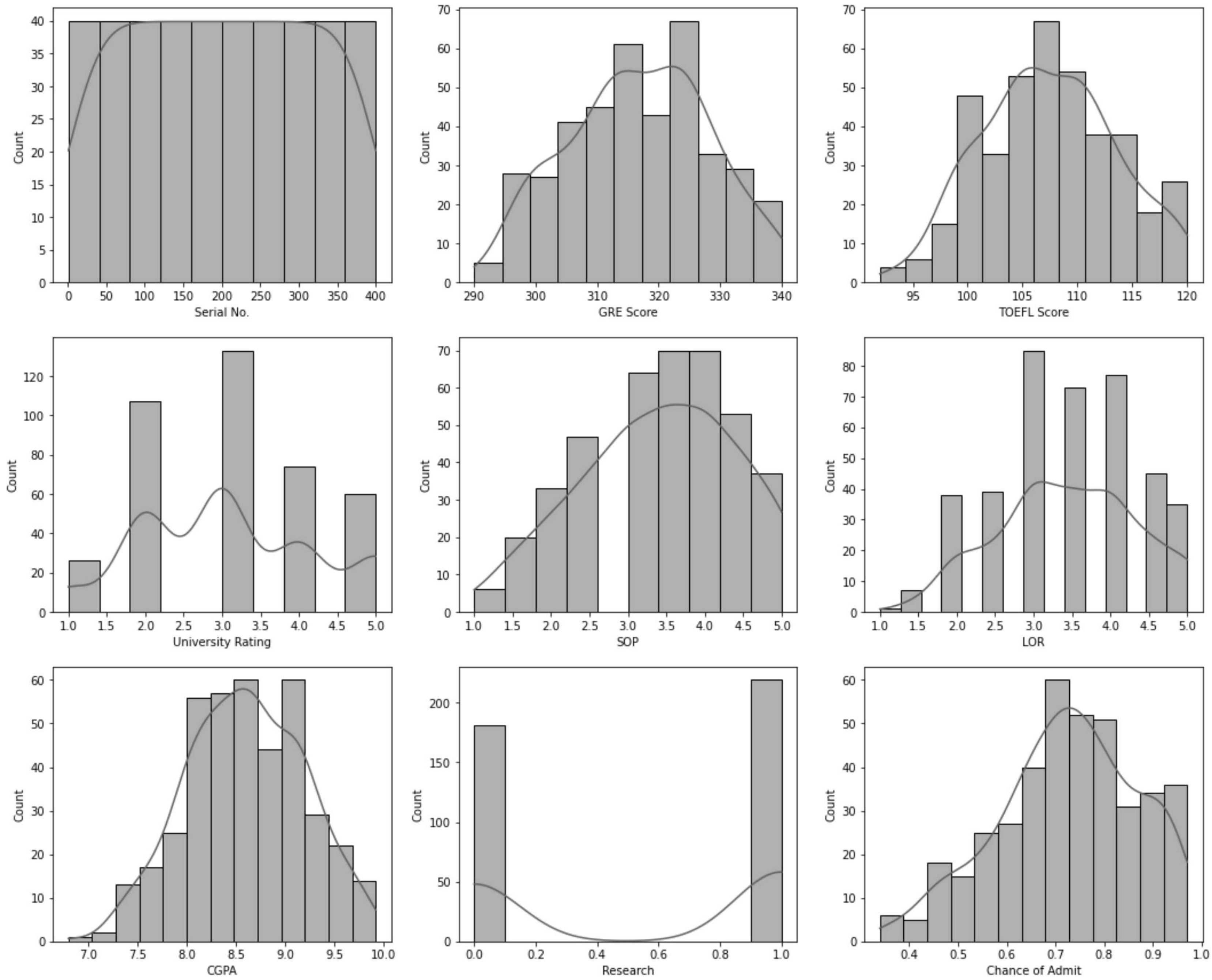


Out[13]:

```
'\n\nplt.figure(figsize=(18,15))\nplotnumber=1\nfor i in df.columns:\n    if plotnumber<=6:\n        ax = pl
t.subplot(3,2,plotnumber)\n        sns.boxplot(df[i])\n    plotnumber+=1\nplt.show()\n'
```

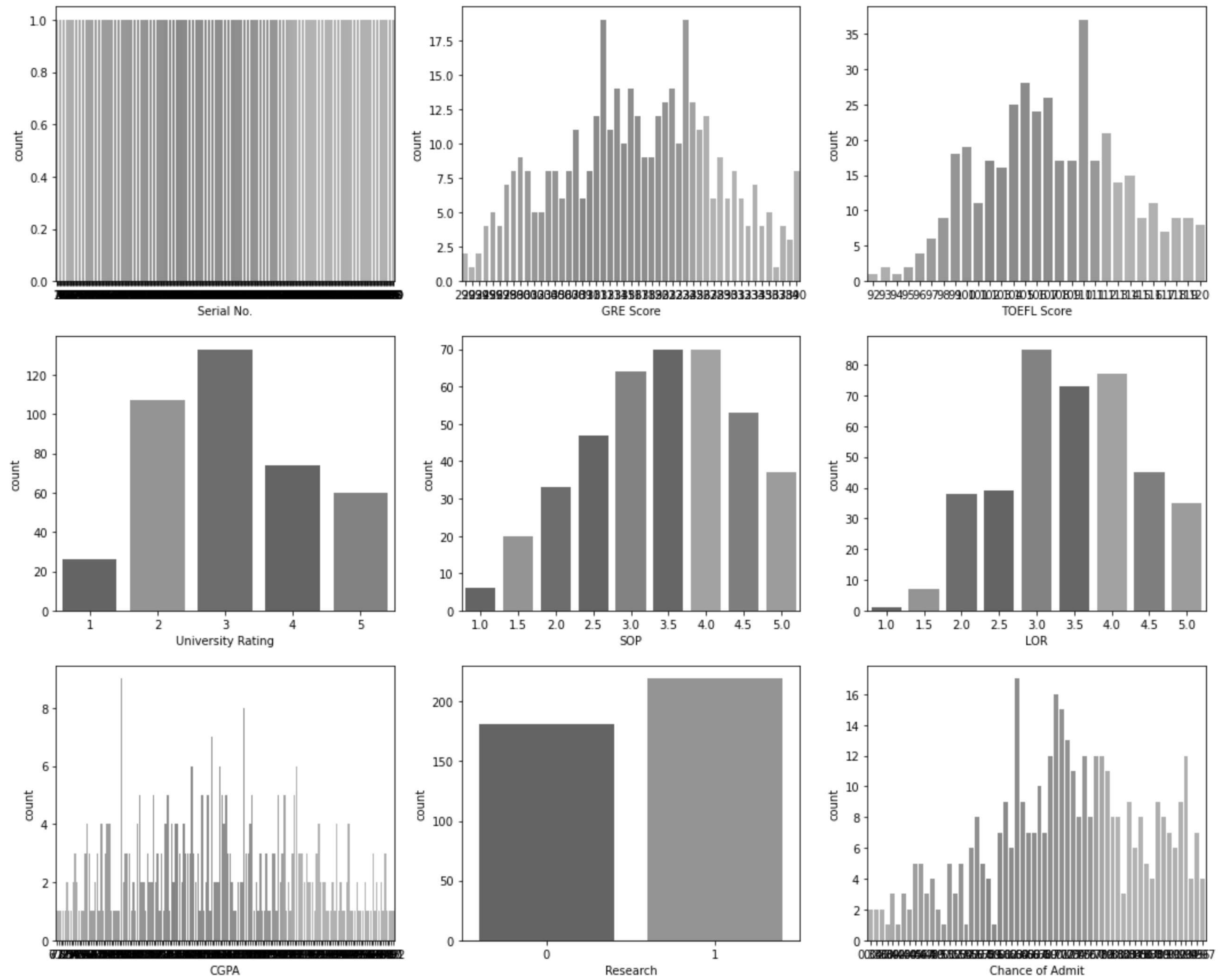
In [14]:

```
# Getting histplot of the features

plt.figure(figsize=(18,15))
plotnumber=1
for i in df.columns:
    if plotnumber<=9:
        ax=plt.subplot(3,3,plotnumber)
        sns.histplot(df[i],kde=True)
    plotnumber+=1
plt.show()
```



In [15]:

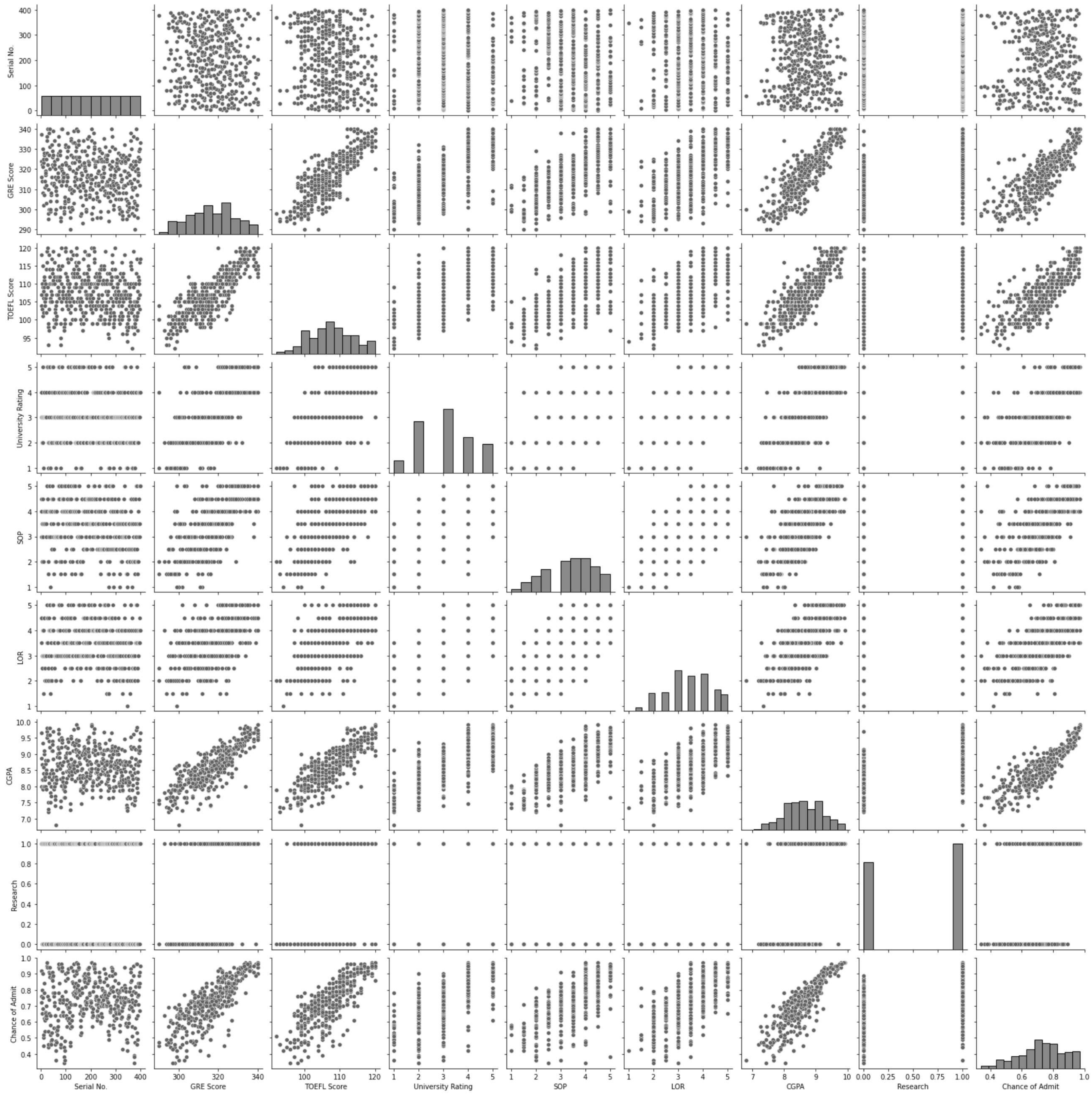
```
# Getting countplot of the features

plt.figure(figsize=(18,15))
plotnumber=1
for i in df.columns:
    if plotnumber<=9:
        ax=plt.subplot(3,3,plotnumber)
        sns.countplot(df[i])
    plotnumber+=1
plt.show()
```



In [16]:

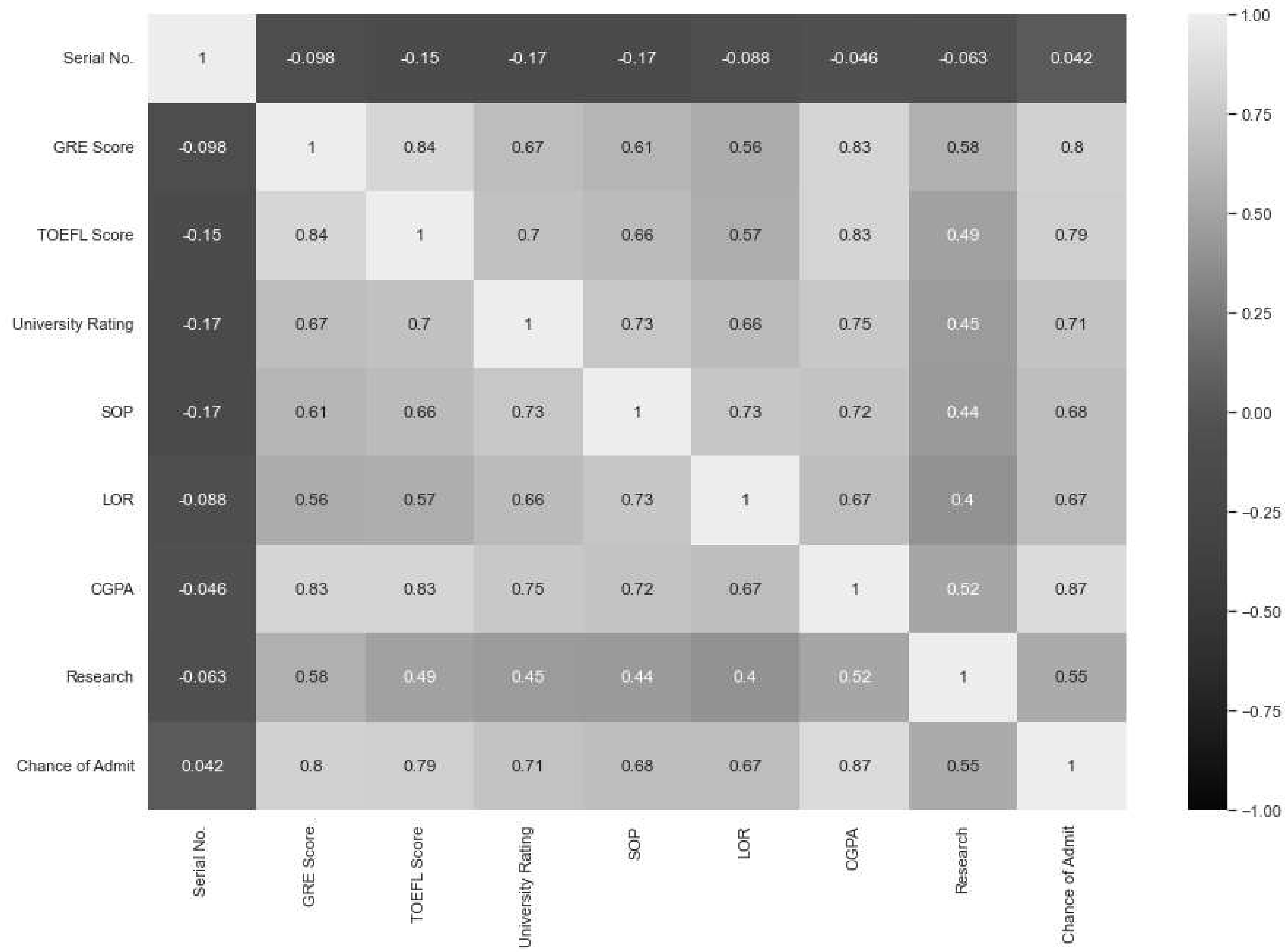
```
## Bivariate Analysis
sns.pairplot(df)
```

Out[16]: <seaborn.axisgrid.PairGrid at 0x1ef5e7d5850>



In [17]:
```python
sns.set(rc={'figure.figsize':(15,10)})
sns.heatmap(data=df.corr(),annot=True, vmin=-1, vmax=1)
```

Out[17]: <AxesSubplot:>




```
Observation:- Drop 'Serial No.' feature from the dataset, because it is of no use.
GRE Score and TOFEL Score are highly correlated with CGPA
```

In [18]:
```python
df.columns
```

Out[18]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR', 'CGPA', 'Research', 'Chance of Admit'],
      dtype='object')
```

In [19]:
```python
# df.drop(['Serial No.','GRE Score','TOEFL Score'],axis=1,inplace=True)
```

In [20]:
```python
## Segregating dependent and independent feature

X = df.iloc[ :, :-1]
y = df.iloc[ :, -1]
```

In [21]:
```python
X.head()
```

Out[21]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	1	337	118	4	4.5	4.5	9.65	1
1	2	324	107	4	4.0	4.5	8.87	1
2	3	316	104	3	3.0	3.5	8.00	1
3	4	322	110	3	3.5	2.5	8.67	1
4	5	314	103	2	2.0	3.0	8.21	0

Train Test Split

In [22]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.33,random_state=4)
```

```python
# Checking VIF

from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [23]:
```python
# Scaling
sc = StandardScaler()

X_train_tf = sc.fit_transform(X_train)
X_test_tf = sc.transform(X_test) # on;y transform to avoid data leakage
```

Model Creation

In [24]:
```python
# SVR Model
model_svr = SVR()
```

In [25]:
```python
model_svr.fit(X_train_tf,y_train)
```

Out[25]:
```
▾ SVR
SVR()
```

In [26]:
```python
## Train Accuracy
train_score = model_svr.score(X_train_tf,y_train)
train_score
```

Out[26]: 0.7947034504013842

In [27]:
```python
# Store model using pickle
import pickle

with open('AdmissionSVR.pkl','wb') as f:
    pickle.dump(model_svr,f)
```

In [28]:
```python
model_svr_load = pickle.load(open('AdmissionSVR.pkl','rb'))
```

In [29]:
```python
y_pred_svr = model_svr.predict(X_test_tf)
```

In [30]:
```python
print('MSE : ',round(mean_squared_error(y_test,y_pred_svr),2))
print('MAE : ',round(mean_absolute_error(y_test,y_pred_svr),2))
```

```
MSE :  0.0
MAE :  0.06
```

In [31]:
```python
svr_r2_score = r2_score(y_test, y_pred_svr)
svr_adj_r2_score = 1 - ((1-svr_r2_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
```

In [32]:
```python
print('R-squared Accuracy : ',round(svr_r2_score*100,2))
print('Adjusted R-squared Accuracy : ',round(svr_adj_r2_score*100,2))
```

```
R-squared Accuracy :  76.61
Adjusted R-squared Accuracy :  75.08
```

In [33]:
```python
## GridSearchCV

from sklearn.model_selection import GridSearchCV
```

In [34]:
```python
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel':['linear','rbf']}
```

In [35]:
```python
model_grid_svr = GridSearchCV(SVR(), param_grid, verbose=3)
model_grid_svr.fit(X_train_tf, y_train)
```

```
[CV 2/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.793 total time=   0.0s
[CV 3/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.718 total time=   0.0s
[CV 4/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.659 total time=   0.0s
[CV 5/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.651 total time=   0.0s
[CV 1/5] END .C=0.1, gamma=0.001, kernel=linear;, score=0.848 total time=   0.0s
[CV 2/5] END .C=0.1, gamma=0.001, kernel=linear;, score=0.866 total time=   0.0s
[CV 3/5] END .C=0.1, gamma=0.001, kernel=linear;, score=0.718 total time=   0.0s
[CV 4/5] END .C=0.1, gamma=0.001, kernel=linear;, score=0.678 total time=   0.0s
[CV 5/5] END .C=0.1, gamma=0.001, kernel=linear;, score=0.650 total time=   0.0s
[CV 1/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.600 total time=   0.0s
[CV 2/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.572 total time=   0.0s
[CV 3/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.599 total time=   0.0s
[CV 4/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.475 total time=   0.0s
[CV 5/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.569 total time=   0.0s
[CV 1/5] END .......C=1, gamma=1, kernel=linear;, score=0.848 total time=   0.0s
[CV 2/5] END .......C=1, gamma=1, kernel=linear;, score=0.866 total time=   0.0s
[CV 3/5] END .......C=1, gamma=1, kernel=linear;, score=0.718 total time=   0.0s
[CV 4/5] END .......C=1, gamma=1, kernel=linear;, score=0.673 total time=   0.0s
[CV 5/5] END .......C=1, gamma=1, kernel=linear;, score=0.650 total time=   0.0s
[CV 1/5] END .........C=1, gamma=1, kernel=rbf;, score=0.251 total time=   0.0s
```

In [36]:

```
print(model_grid_svr.best_estimator_)
```

```
SVR(C=0.1, gamma=1, kernel='linear')
```

In [37]:

```
# Train Accuracy

model_grid_svr.score(X_train_tf,y_train)
```

Out[37]: 0.8027928629188807

In [38]:

```
# Test Accuracy

y_pred_grid = model_grid_svr.predict(X_test_tf)
```

In [39]:

```
print('MSE : ',round(mean_squared_error(y_test,y_pred_grid),2))
print('MAE : ',round(mean_absolute_error(y_test,y_pred_grid),2))
```

```
MSE :  0.0
MAE :  0.05
```

In [40]:

```
grid_svr_r2_score = r2_score(y_test, y_pred_grid)
grid_svr_adj_r2_score = 1 - ((1-grid_svr_r2_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
```

In [41]:

```
print('R-squared Accuracy : ',round(grid_svr_r2_score*100,2))
print('Adjusted R-squared Accuracy : ',round(grid_svr_adj_r2_score*100,2))
```

```
R-squared Accuracy :  80.07
Adjusted R-squared Accuracy :  78.78
```

In [42]:

```
## Linear Regression Model

lin_reg = LinearRegression()
lin_reg
```

Out[42]:

```
▾ LinearRegression
LinearRegression()
```

In [43]:

```
lin_reg.fit(X_train_tf, y_train)
```

Out[43]:

```
▾ LinearRegression
LinearRegression()
```

In [44]:

```
# Train Accuracy

lin_reg.score(X_train_tf, y_train)
```

Out[44]: 0.8168014873134735

In [45]:

```
lin_reg_pred = lin_reg.predict(X_test_tf)
```

In [46]:

```
print('MSE : ',round(mean_squared_error(y_test,lin_reg_pred),2))
print('MAE : ',round(mean_absolute_error(y_test,lin_reg_pred),2))
```

```
MSE :  0.0
MAE :  0.05
```

In [47]:

```
lin_reg_r2_score = r2_score(y_test, lin_reg_pred)
lin_reg_adj_r2_score = 1 - ((1-lin_reg_r2_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
```

In [48]:

```
print('R-squared Accuracy : ',round(lin_reg_r2_score*100,2))
print('Adjusted R-squared Accuracy : ',round(lin_reg_adj_r2_score*100,2))
```

```
R-squared Accuracy :  81.09
Adjusted R-squared Accuracy :  79.86
```

In [49]:

```
# Ridge Regression Model

ridge_reg = Ridge()
ridge_reg
```

Out[49]:

```
▾ Ridge
Ridge()
```

In [50]:
```
ridge_reg.fit(X_train_tf, y_train)
```

Out[50]:

▾ Ridge

Ridge()

In [51]:
```
# Train Accuracy
ridge_reg.score(X_train_tf, y_train)
```

Out[51]: 0.8167919100113752

In [52]:
```
ridge_reg_pred = ridge_reg.predict(X_test_tf)
```

In [53]:
```
print('MSE : ',round(mean_squared_error(y_test,ridge_reg_pred),2))
print('MAE : ',round(mean_absolute_error(y_test,ridge_reg_pred),2))
```

```
MSE :  0.0
MAE :  0.05
```

In [54]:
```
ridge_reg_r2_score = r2_score(y_test, ridge_reg_pred)
ridge_reg_adj_r2_score = 1 - ((1-ridge_reg_r2_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
```

In [55]:
```
print('R-squared Accuracy : ',round(ridge_reg_r2_score*100,2))
print('Adjusted R-squared Accuracy : ',round(ridge_reg_adj_r2_score*100,2))
```

```
R-squared Accuracy :  81.11
Adjusted R-squared Accuracy :  79.88
```

In [56]:
```
## Lasso Regressionj Model
lasso_reg = Lasso()
lasso_reg.fit(X_train_tf, y_train)
```

Out[56]:

▾ Lasso

Lasso()

In [57]:
```
lasso_reg_pred = lasso_reg.predict(X_test_tf)
```

In [58]:
```
print('MSE : ',round(mean_squared_error(y_test,lasso_reg_pred),2))
print('MAE : ',round(mean_absolute_error(y_test,lasso_reg_pred),2))
```

```
MSE :  0.02
MAE :  0.12
```

In [59]:
```
lasso_reg_r2_score = r2_score(y_test, lasso_reg_pred)
lasso_reg_adj_r2_score = 1 - ((1-lasso_reg_r2_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
```

In [60]:
```
print('R-squared Accuracy : ',round(lasso_reg_r2_score*100,2))
print('Adjusted R-squared Accuracy : ',round(lasso_reg_adj_r2_score*100,2))
```

```
R-squared Accuracy :  -1.32
Adjusted R-squared Accuracy :  -7.91
```

In [61]:
```
## Elastic Net Regression Model

elastic_reg = ElasticNet()
elastic_reg
```

Out[61]:

▾ ElasticNet

ElasticNet()

In [62]:
```
elastic_reg.fit(X_train_tf, y_train)
```

Out[62]:

▾ ElasticNet

ElasticNet()

In [63]:
```
# Train Accuracy
elastic_reg.score(X_train_tf,y_train )
```

Out[63]: 0.0

In [64]:
```
elastic_reg_pred = elastic_reg.predict(X_test_tf)
```

In [65]:

```
print('MSE : ',round(mean_squared_error(y_test,elastic_reg_pred),2))
print('MAE : ',round(mean_absolute_error(y_test,elastic_reg_pred),2))
```

```
MSE :  0.02
MAE :  0.12
```

In [66]:

```
elastic_reg_r2_score = r2_score(y_test, elastic_reg_pred)
elastic_reg_adj_r2_score = 1 - ((1-elastic_reg_r2_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
```

In [67]:

```
print('R-squared Accuracy : ',round(elastic_reg_r2_score*100,2))
print('Adjusted R-squared Accuracy : ',round(elastic_reg_adj_r2_score*100,2))
```

```
R-squared Accuracy :  -1.32
Adjusted R-squared Accuracy :  -7.91
```

Comparison of all Models

In [68]:

```
print('-----------------------')
print('MSE : \n1. SVR : ',round(mean_squared_error(y_test, y_pred_svr),2))
print('2. Linear Regression : ',round(mean_squared_error(y_test,lin_reg_pred),2))
print('3. Ridge Regression : ',round(mean_squared_error(y_test, ridge_reg_pred),2))
print('4. Elastic Net Regression : ',round(mean_squared_error(y_test, elastic_reg_pred),2))
print('5. GridRegressionSVR : ',round(mean_squared_error(y_test, y_pred_grid),2))

print('-----------------------')
print('MAE : \n1. SVR : ',round(mean_absolute_error(y_test, y_pred_svr),2))
print('2. Linear Regression : ',round(mean_absolute_error(y_test,lin_reg_pred),2))
print('3. Ridge Regression : ',round(mean_absolute_error(y_test, ridge_reg_pred),2))
print('4. Elastic Net Regression : ',round(mean_absolute_error(y_test, elastic_reg_pred),2))
print('5. GridRegressionSVR : ',round(mean_absolute_error(y_test, y_pred_grid),2))
```

```
-----------------------
MSE :
1. SVR :  0.0
2. Linear Regression :  0.0
3. Ridge Regression :  0.0
4. Elastic Net Regression :  0.02
5. GridRegressionSVR :  0.0
-----------------------
MAE :
1. SVR :  0.06
2. Linear Regression :  0.05
3. Ridge Regression :  0.05
4. Elastic Net Regression :  0.12
5. GridRegressionSVR :  0.05
```

In [76]:

```
print('Train Accuracy')
print('----------------------')
print('R-squared Accuracy : \n1.SVR : ',round(svr_r2_score*100,2))
print('2. Linear Regression : ',round(lin_reg_r2_score*100,2))
print('3. Ridge Regression : ',round(ridge_reg_r2_score*100,2))
print('4. Lasso Regression : ',round(lasso_reg_r2_score*100,2))
print('5. Elastic Net Regression : ',round(elastic_reg_r2_score*100,2))
print('6. GridSearchCV : ',round(grid_svr_r2_score*100,2))
```

```
Train Accuracy
----------------------
R-squared Accuracy :
1.SVR :  76.61
2. Linear Regression :  81.09
3. Ridge Regression :  81.11
4. Lasso Regression :  -1.32
5. Elastic Net Regression :  -1.32
6. GridSearchCV :  80.07
```

In [77]:

```
print('Test Accuracy')
print('---------------------')
print('Adjusted R-squared Accuracy : \n1.SVR : ',round(svr_r2_score*100,2))
print('2. Linear Regression : ',round(lin_reg_adj_r2_score*100,2))
print('3. Ridge Regression : ',round(ridge_reg_adj_r2_score*100,2))
print('4. Lasso Regression : ',round(lasso_reg_adj_r2_score*100,2))
print('5. Elastic Net Regression : ',round(elastic_reg_adj_r2_score*100,2))
print('6. GridSearchCV : ',round(grid_svr_adj_r2_score*100,2))
```

```
Test Accuracy
---------------------
Adjusted R-squared Accuracy :
1.SVR :  76.61
2. Linear Regression :  79.86
3. Ridge Regression :  79.88
4. Lasso Regression :  -7.91
5. Elastic Net Regression :  -7.91
6. GridSearchCV :  78.78
```

Observation :-

```
We can simply choose the best fit model according to this peformance score
```

In [ ]:

```

```