USE CLASSIFICATION TECHNIQUE FOR PREDICTION OF GRADUATE ADDMISSION FROM AN PROSPECTIVE CODE

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('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.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: 28.3 KB
```

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
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
```

In [6]:
```
df.describe()
```

Out[6]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000
mean	200.500000	316.807500	107.410000	3.087500	3.400000	3.452500	8.598925	0.547500	0.724350
std	115.614301	11.473646	6.069514	1.143728	1.006869	0.898478	0.596317	0.498362	0.142609
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.000000	6.800000	0.000000	0.340000
25%	100.750000	308.000000	103.000000	2.000000	2.500000	3.000000	8.170000	0.000000	0.640000
50%	200.500000	317.000000	107.000000	3.000000	3.500000	3.500000	8.610000	1.000000	0.730000
75%	300.250000	325.000000	112.000000	4.000000	4.000000	4.000000	9.062500	1.000000	0.830000
max	400.000000	340.000000	120.000000	5.000000	5.000000	5.000000	9.920000	1.000000	0.970000

In [7]:
```
df.drop(['Serial No.'], axis=1, inplace=True)
```

In [8]:
```
df
```

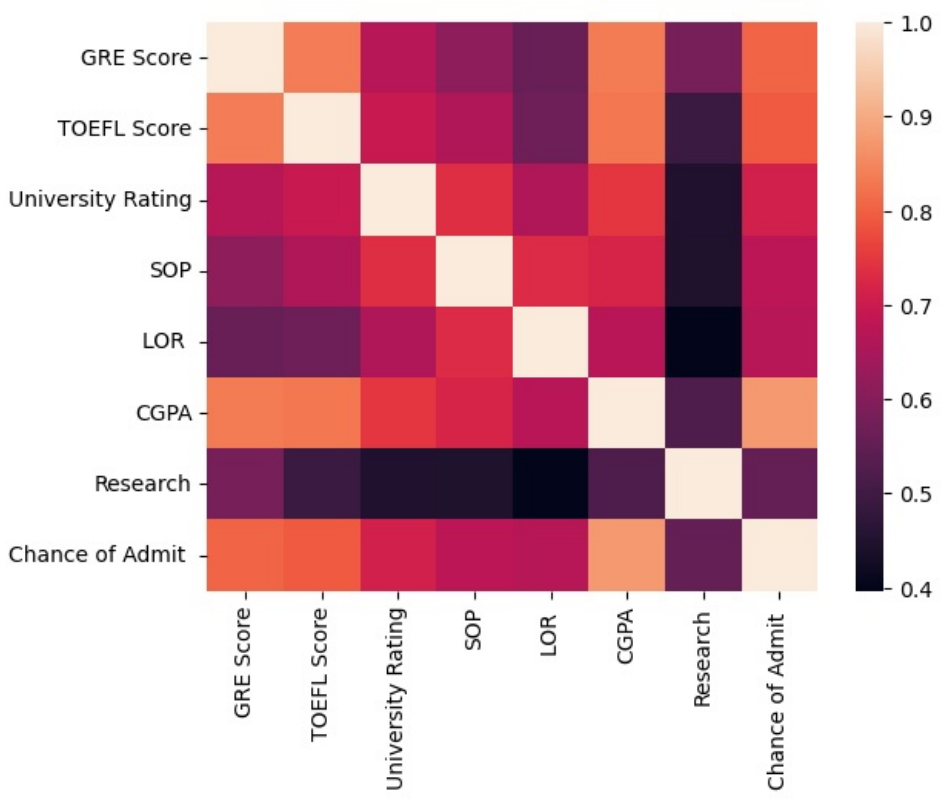
Out[8]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65
...	...	...	...	...	...	...	...	...
395	324	110	3	3.5	3.5	9.04	1	0.82
396	325	107	3	3.0	3.5	9.11	1	0.84
397	330	116	4	5.0	4.5	9.45	1	0.91
398	312	103	3	3.5	4.0	8.78	0	0.67
399	333	117	4	5.0	4.0	9.66	1	0.95

400 rows × 8 columns

In [9]:
```
sns.heatmap(df.corr())
```

Out[9]:
```
<Axes: >
```



In [10]:
```
from sklearn.model_selection import train_test_split
```

In [11]:
```
x = df.drop(['Chance of Admit '], axis = 1)
y = df['Chance of Admit ']
```

In [12]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state=101)
```

In [13]:
```
print('x_train: ',x_train.shape)
print('x_test: ',x_test.shape)
print('y_train: ',y_train.shape)
print('y_test: ',y_test.shape)
```

```
x_train:  (320, 7)
x_test:  (80, 7)
y_train:  (320,)
y_test:  (80,)
```

feature scaling

In [14]:
```
from sklearn.preprocessing import StandardScaler
```

In [15]:
```
scaler = StandardScaler()
scaler
```

Out[15]:
```
▾ StandardScaler
StandardScaler()
```

In [16]:
```
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

In [17]:
```
x_train
```

Out[17]:
```
array([[ 0.13461222, -0.28582533, -0.14928523, ..., -0.57064504,
        -1.08773332,  0.84878154],
       [ 0.7425384 ,  1.36217656, -1.03393847, ..., -0.57064504,
         0.73934146,  0.84878154],
       [-1.6891663 , -1.43942665, -1.03393847, ..., -1.68071441,
        -1.76442769, -1.17815946],
       ...,
       [ 1.26361798,  1.69177694,  1.62002124, ...,  1.64949371,
         1.39911847,  0.84878154],
       [ 0.829385  ,  0.53817562,  0.73536801, ...,  1.09445902,
         0.60400259,  0.84878154],
       [ 0.6556918 ,  0.37337543,  0.73536801, ...,  0.53942433,
         0.04572974,  0.84878154]])
```

import models

In [18]:
```
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

In [19]:
```
l_reg = LinearRegression()
l_reg.fit(x_train,y_train)

svm = SVR()
svm.fit(x_train,y_train)

r_reg = RandomForestRegressor()
r_reg.fit(x_train,y_train)

g_boost = GradientBoostingRegressor()
g_boost.fit(x_train,y_train)
print('Classification models','\n',l_reg,'\n',svm,'\n',r_reg,'\n',g_boost)
```

```
Classification models
 LinearRegression()
 SVR()
 RandomForestRegressor()
 GradientBoostingRegressor()
```

In [20]:
```
y_pred1 = l_reg.predict(x_test)
y_pred2 = svm.predict(x_test)
y_pred3 = r_reg.predict(x_test)
y_pred4 = g_boost.predict(x_test)
```

In [21]:
```
from sklearn import metrics
```

In [22]:
```
l_reg_score = metrics.r2_score(y_test,y_pred1)
svm_score = metrics.r2_score(y_test,y_pred2)
r_reg_score = metrics.r2_score(y_test,y_pred3)
g_boost_score = metrics.r2_score(y_test,y_pred4)
```

In [23]:
```
print('LinearRegression: ', l_reg_score)
print('SVR: ', svm_score)
print('RandomForestRegressor: ', r_reg_score)
print('GradientBoostingRegressor: ', g_boost_score)
```

```
LinearRegression:  0.8001063801904392
SVR:  0.720155437414667
RandomForestRegressor:  0.7795363238158549
GradientBoostingRegressor:  0.7571191366783935
```

In [24]:
```
final_data = pd.DataFrame({'Models': ['LR','SVR','RF','GBR'],
                           'R2_score':[l_reg_score,svm_score,r_reg_score,g_boost_score]})
```

In [25]:
```
final_data
```

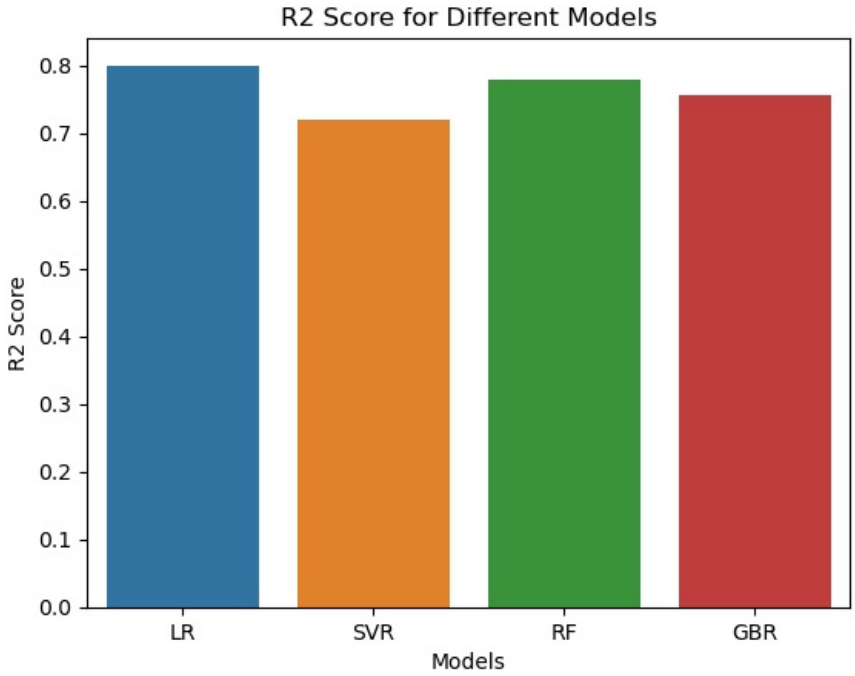
Out[25]:

	Models	R2_score
0	LR	0.800106
1	SVR	0.720155
2	RF	0.779536
3	GBR	0.757119

In [26]:
```
sns.barplot(x=final_data['Models'], y=final_data['R2_score'])

plt.xlabel('Models')
plt.ylabel('R2 Score')
plt.title('R2 Score for Different Models')

plt.show()
```



classification

In [27]:
```
df.head()
```

Out[27]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65

In [28]:
```
# list comprehension

y_train = [1 if value>0.8 else 0 for value in y_train]
y_test = [1 if value>0.8 else 0 for value in y_test]

y_train = np.array(y_train)
y_test = np.array(y_test)
```

In [29]:
```
y_train
```

Out[29]:
```
array([0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1,
       0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0])
```

In [30]:
```
y_test
```

Out[30]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0])
```

In [31]:
```
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
```

In [32]:
```
l_reg = LogisticRegression()
l_reg.fit(x_train,y_train)
y_pred1 = l_reg.predict(x_test)
print('LogisticRegression_accuracy_score_ : ',accuracy_score (y_test, y_pred1))
```

```
LogisticRegression_accuracy_score_ :  0.9375
```

In [33]:
```
svm = svm.SVC()
svm.fit(x_train,y_train)
y_pred2 = svm.predict(x_test)
print('svm_accuracy_score_ : ',accuracy_score (y_test, y_pred2))
```

```
svm_accuracy_score_ :  0.9375
```

In [34]:
```
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)
y_pred3 = knn.predict(x_test)
print('knn_accuracy_score_ : ',accuracy_score (y_test, y_pred3))
```

```
knn_accuracy_score_ :  0.9375
```

In [35]:
```
rf = RandomForestClassifier()
rf.fit(x_train,y_train)
y_pred4 = rf.predict(x_test)
print('RandomForestClassifier_accuracy_score : ',accuracy_score (y_test, y_pred4))
```

```
RandomForestClassifier_accuracy_score :  0.9375
```

In [36]:
```
g_boost = GradientBoostingClassifier()
g_boost.fit(x_train,y_train)
y_pred5 = g_boost.predict(x_test)
print('GradientBoostingClassifier_accuracy_score : ',accuracy_score (y_test, y_pred5))
```

```
GradientBoostingClassifier_accuracy_score :  0.9125
```

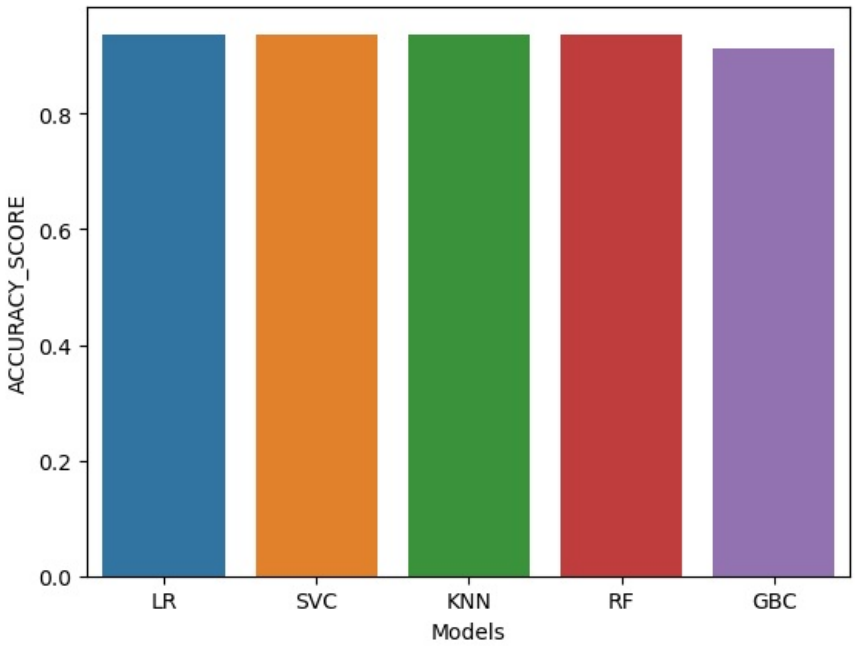
In [37]:
```
final = pd.DataFrame({"Models":['LR','SVC','KNN','RF','GBC'],
                      'ACCURACY_SCORE':[accuracy_score(y_test,y_pred1),accuracy_score(y_test,y_pred2),\
                                        accuracy_score(y_test,y_pred3),accuracy_score(y_test,y_pred4),accuracy_sc
```

In [38]:
```
final
```

Out[38]:

	Models	ACCURACY_SCORE
0	LR	0.9375
1	SVC	0.9375
2	KNN	0.9375
3	RF	0.9375
4	GBC	0.9125

In [39]:
```
sns.barplot(x=final['Models'],y=final['ACCURACY_SCORE'])
plt.show()
```



In [ ]: