In [71]:
```
import pandas as pd
```

In [72]:
```
data = pd.read_csv('Admission_Predict.csv')
```

1. Display Top 5 Rows of The Dataset

In [6]:
```
data.head()
```

Out[6]:

	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. Check Last 5 Rows of The Dataset

In [7]:
```
data.tail()
```

Out[7]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
395	396	324	110	3	3.5	3.5	9.04	1	0.82
396	397	325	107	3	3.0	3.5	9.11	1	0.84
397	398	330	116	4	5.0	4.5	9.45	1	0.91
398	399	312	103	3	3.5	4.0	8.78	0	0.67
399	400	333	117	4	5.0	4.0	9.66	1	0.95

3. Find Shape of Our Dataset (Number of Rows And Number of Columns)

In [8]:
```
data.shape
```

Out[8]:
```
(400, 9)
```

In [9]:
```
print("Number of Rows",data.shape[0])
print("Number of Columns",data.shape[1])
```

```
Number of Rows 400
Number of Columns 9
```

4. Get Information About Our Dataset Like Total Number Rows, Total Number of Columns, Datatypes of Each Column And Memory Requirement

In [10]:
```
data.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
```

5.Check Null Values In The Dataset

In [11]:
```
data.isnull().sum()
```

Out[11]:
```
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
```

6. Get Overall Statistics About The Dataset

In [12]:
```
data.describe()
```

Out[12]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Res
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.0
mean	200.500000	316.807500	107.410000	3.087500	3.400000	3.452500	8.598925	0.5
std	115.614301	11.473646	6.069514	1.143728	1.006869	0.898478	0.596317	0.4
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.000000	6.800000	0.0
25%	100.750000	308.000000	103.000000	2.000000	2.500000	3.000000	8.170000	0.0
50%	200.500000	317.000000	107.000000	3.000000	3.500000	3.500000	8.610000	1.0
75%	300.250000	325.000000	112.000000	4.000000	4.000000	4.000000	9.062500	1.0
max	400.000000	340.000000	120.000000	5.000000	5.000000	5.000000	9.920000	1.0

7. Dropping Irrelevant Features

In [13]:
```
data.columns
```

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

In [14]:
```
data = data.drop('Serial No.',axis=1)
```

In [15]:
```
data.columns
```

Out[15]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGP
A',
       'Research', 'Chance of Admit '],
      dtype='object')
```

8. Store Feature Matrix In X and Response(Target) In Vector y

In [16]:
```
data.head(1)
```

Out[16]:

	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

In [17]:
```
data.columns
```

Out[17]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGP
A',
       'Research', 'Chance of Admit '],
      dtype='object')
```

In [18]:
```
X = data.drop('Chance of Admit ',axis=1)
```

In [19]:
```
y = data['Chance of Admit ']
```

In [ ]:
```

```

9. Splitting The Dataset Into The Training Set And Test Set

In [20]:
```
from sklearn.model_selection import train_test_split
```

In [21]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state=
```

In [22]:
```
y_train
```

Out[22]:
```
3      0.80
18     0.63
202    0.97
250    0.74
274    0.58
       ...
71     0.96
106    0.87
270    0.72
348    0.57
102    0.62
Name: Chance of Admit , Length: 320, dtype: float64
```

10. Feature Scaling

In [23]:
```
data.head()
```

Out[23]:

	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 [24]:
```
from sklearn.preprocessing import StandardScaler
```

In [25]:
```
sc = StandardScaler()
```

In [26]:
```
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

In [27]:
```
X_train
```

Out[27]:
```
array([[ 0.45711129,  0.42466178, -0.057308  , ..., -1.05965163,
         0.13986648,  0.92761259],
       [ 0.1022887 ,  0.42466178, -0.057308  , ..., -0.50194025,
         0.36110014, -1.07803625],
       [ 2.05381293,  2.08593034,  1.6892215 , ...,  1.17119391,
         2.25009529,  0.92761259],
       ...,
       [-0.96217907, -0.40597251, -0.93057275, ..., -0.50194025,
        -0.62594237,  0.92761259],
       [-1.31700165, -1.40273364, -1.8038375 , ..., -1.61736302,
        -2.27668588, -1.07803625],
       [-0.25253389, -0.23984565, -0.93057275, ...,  0.05577114,
        -0.57488845, -1.07803625]])
```

11. Import The models

In [28]:
```
data.head()
```

Out[28]:

	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 [29]:
```
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

12. Model Training

In [30]:
```
lr =LinearRegression()
lr.fit(X_train,y_train)

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

rf = RandomForestRegressor()
rf.fit(X_train,y_train)

gr = GradientBoostingRegressor()
gr.fit(X_train,y_train)
```

Out[30]:
```
GradientBoostingRegressor

GradientBoostingRegressor()
```

13. Prediction on Test Data

In [31]:
```
y_pred1 = lr.predict(X_test)
y_pred2 = svm.predict(X_test)
y_pred3 = rf.predict(X_test)
y_pred4 = gr.predict(X_test)
```

14. Evaluating the Algorithm

In [32]:
```
from sklearn import metrics
```

In [33]:
```
score1 = metrics.r2_score(y_test,y_pred1)
score2 = metrics.r2_score(y_test,y_pred2)
score3 = metrics.r2_score(y_test,y_pred3)
score4 = metrics.r2_score(y_test,y_pred4)
```

In [34]:
```
print(score1,score2,score3,score4)
```

```
0.8212082591486991 0.7597814848647668 0.8085279522234895 0.7969609389648314
```

In [35]:
```
final_data = pd.DataFrame({'Models':['LR','SVR','RF','GR'],
                           'R2_SCORE':[score1,score2,score3,score4]})
```

In [36]:
```
final_data
```

Out[36]:

	Models	R2_SCORE
0	LR	0.821208
1	SVR	0.759781
2	RF	0.808528
3	GR	0.796961

In [39]:
```
import seaborn as sns
```

In [40]:
```
sns.barplot(final_data['Models'],final_data['R2_SCORE'])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[40], line 1
----> 1 sns.barplot(final_data['Models'],final_data['R2_SCORE'])

TypeError: barplot() takes from 0 to 1 positional arguments but 2 were given
```

Classification

In [41]:
```
data.head()
```

Out[41]:

	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 [42]:
```
import numpy as np
```

In [43]:
```
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 [44]:
```
y_train
```

Out[44]:
```
array([0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1,
       0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0])
```

15. Import The models

In [45]:
```
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
```

16. Model Training & Evaluation

In [46]:
```
lr = LogisticRegression()
lr.fit(X_train,y_train)
y_pred1= lr.predict(X_test)
print(accuracy_score(y_test,y_pred1))
```

```
0.925
```

In [47]:
```
svm = svm.SVC()
svm.fit(X_train,y_train)
y_pred2 = svm.predict(X_test)
print(accuracy_score(y_test,y_pred2))
```

```
0.925
```

In [48]:
```
knn=KNeighborsClassifier()
knn.fit(X_train,y_train)
y_pred3 = knn.predict(X_test)
print(accuracy_score(y_test,y_pred3))
```

```
0.8875
```

In [49]:
```
rf = RandomForestClassifier()
rf.fit(X_train,y_train)
y_pred4 = rf.predict(X_test)
print(accuracy_score(y_test,y_pred4))
```

```
0.95
```

In [50]:
```
gr = GradientBoostingClassifier()
gr.fit(X_train,y_train)
y_pred5 = gr.predict(X_test)
print(accuracy_score(y_test,y_pred5))
```

```
0.975
```

In [51]:
```
final_data = pd.DataFrame({'Models':['LR','SVC','KNN','RF','GBC'],
                           'ACC_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_score(y_test,y_pred5)]})
```

In [52]:
```
final_data
```

Out[52]:

	Models	ACC_SCORE
0	LR	0.9250
1	SVC	0.9250
2	KNN	0.8875
3	RF	0.9500
4	GBC	0.9750

In [53]:
```
import seaborn as sns
```

In [54]:
```
sns.barplot(final_data['Models'],final_data['ACC_SCORE'])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[54], line 1
----> 1 sns.barplot(final_data['Models'],final_data['ACC_SCORE'])

TypeError: barplot() takes from 0 to 1 positional arguments but 2 were given
```

17. Save The Model

In [55]:
```
data.columns
```

Out[55]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGP
A',
       'Research', 'Chance of Admit '],
      dtype='object')
```

In [56]:
```
X = data.drop('Chance of Admit ',axis=1)
```

In [57]:
```
y = data['Chance of Admit ']
```

In [58]:
```
y  = [1 if value>0.8 else 0 for value in y]
```

In [59]:
```
y = np.array(y)
```

In [60]:
```
y
```

Out[60]:
```
array([1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1])
```

In [61]:
```
X = sc.fit_transform(X)
```

In [62]:
```
X
```

Out[62]:
```
array([[ 1.76210664,  1.74697064,  0.79882862, ...,  1.16732114,
         1.76481828,  0.90911166],
       [ 0.62765641, -0.06763531,  0.79882862, ...,  1.16732114,
         0.45515126,  0.90911166],
       [-0.07046681, -0.56252785, -0.07660001, ...,  0.05293342,
        -1.00563118,  0.90911166],
       ...,
       [ 1.15124883,  1.41704229,  0.79882862, ...,  1.16732114,
         1.42900622,  0.90911166],
       [-0.41952842, -0.72749202, -0.07660001, ...,  0.61012728,
         0.30403584, -1.09997489],
       [ 1.41304503,  1.58200646,  0.79882862, ...,  0.61012728,
         1.78160888,  0.90911166]])
```

In [63]:
```
gr = GradientBoostingClassifier()
gr.fit(X,y)
```

Out[63]:
```
GradientBoostingClassifier

GradientBoostingClassifier()
```

In [64]:
```
import joblib
```

In [65]:
```
joblib.dump(gr,'admission_model')
```

Out[65]:
```
['admission_model']
```

In [66]:
```
model = joblib.load('admission_model')
```

In [67]:
```
data.columns
```

Out[67]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGP
A',
       'Research', 'Chance of Admit '],
      dtype='object')
```

In [68]:
```
model.predict(sc.transform([[337,118,4,4.5,4.5,9.65,1]]))
```

```
D:\anaconda\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not ha
ve valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

Out[68]:
```
array([1])
```

In [ ]:
```

```

GUI

In [69]:
```
from tkinter import *
import joblib
from sklearn.preprocessing import StandardScaler
```

In [70]:
```
def show_entry():

    p1 = float(e1.get())
    p2 = float(e2.get())
    p3 = float(e3.get())
    p4 = float(e4.get())
    p5 = float(e5.get())
    p6 = float(e6.get())
    p7 = float(e6.get())

    model = joblib.load('admission_model')
    result = model.predict(sc.transform([[p1,p2,p3,p4,p5,p6,p7]]))

    if result == 1:
        Label(master, text="High Chance of getting admission").grid(row=31)
    else:
        Label(master, text="You may get admission").grid(row=31)

master =Tk()
master.title("Graduate Admission Analysis and Prediction")
label = Label(master,text = "Graduate Admission Analysis and Prediction",bg = "
              fg = "white").grid(row=0,columnspan=2)

Label(master,text = "Enter Your GRE Score").grid(row=1)
Label(master,text = "Enter Your TOEFL Score").grid(row=2)
Label(master,text = "Enter University Rating").grid(row=3)
Label(master,text = "Enter SOP").grid(row=4)
Label(master,text = "Enter LOR").grid(row=5)
Label(master,text = "Enter Your CPGA").grid(row=6)
Label(master,text = "Research").grid(row=7)


e1 = Entry(master)
e2 = Entry(master)
e3 = Entry(master)
e4 = Entry(master)
e5 = Entry(master)
e6 = Entry(master)
e7 = Entry(master)

e1.grid(row=1,column=1)
e2.grid(row=2,column=1)
e3.grid(row=3,column=1)
e4.grid(row=4,column=1)
e5.grid(row=5,column=1)
e6.grid(row=6,column=1)
e7.grid(row=7,column=1)


Button(master,text="Predict",command=show_entry).grid()

mainloop()
```

```
D:\anaconda\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not ha
ve valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

In [ ]: