In [92]:

```
import pandas as pd
```

In [93]:

```
data=pd.read_csv("https://raw.githubusercontent.com/aniruddhachoudhury/Red-Wine-Quality/mas
```

In [94]:

```
data
```

Out[94]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	al
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
...	...	...	...	...	...	...	...	...	...	...	
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	

1599 rows × 12 columns

In [95]:

```
data.columns
```

Out[95]:

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'densit
y',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [96]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [97]:

```
data.describe()
```

Out[97]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total su dio
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000

In [98]:

```
data.quality.unique()
```

Out[98]:

```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

In [99]:

```
data['quality'].value_counts()
```

Out[99]:

```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

In [100]:

```
from sklearn.model_selection import train_test_split
```

In [101]:

```
x=data.drop('quality',axis=1)
```

In [102]:

```
x.head()
```

Out[102]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9

In [103]:

```
y=data['quality']
```

In [104]:

```
y
```

Out[104]:

```
0       5
1       5
2       5
3       6
4       5
       ..
1594    5
1595    6
1596    6
1597    5
1598    6
Name: quality, Length: 1599, dtype: int64
```

In [105]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42)
```

In [106]:

```
from sklearn.preprocessing import StandardScaler
```

In [107]:

```
scaler=StandardScaler()
```

In [108]:

```
scaler
```

Out[108]:

```
▾ StandardScaler
StandardScaler()
```

In [109]:

```
x_train_tf=scaler.fit_transform(x_train)
```

In [110]:

```
x_train_tf
```

Out[110]:

```
array([[ 2.40069523, -1.03103722,  1.12742595, ..., -1.26096312,
         0.52726134, -0.01431863],
       [-0.93967131,  1.22920403, -1.32502245, ...,  1.52622836,
        -0.28225704,  2.24363201],
       [-0.99827424,  0.55113165, -1.37611513, ..., -0.74241587,
        -1.20742091, -0.86105011],
       ...,
       [-0.6466567 ,  0.49462562, -1.06955908, ...,  1.26695473,
        -0.68701624, -0.86105011],
       [-0.23643625, -1.87862768,  0.4121285 , ...,  0.03540501,
         0.81637505,  1.39690052],
       [-1.46709761, -1.3700734 , -0.04770558, ...,  0.48913386,
        -0.68701624,  2.90220094]])
```

In [111]:

```
from sklearn.svm import SVC
model=SVC()
```

In [112]:

```
model.fit(x_train_tf,y_train)
```

Out[112]:

```
▾ SVC
SVC()
```

In [113]:

```
model.score(x_train_tf,y_train)
```

Out[113]:

```
0.6778711484593838
```

In [114]:

```
x_test_tf=scaler.transform(x_test)
```

In [115]:

```
x_test_tf
```

Out[115]:

```
array([[-3.53642095e-01,  1.55589436e-01, -9.67373729e-01, ...,
        -4.83142240e-01,  6.85666499e-03, -7.66968836e-01],
       [-2.95039173e-01, -1.83446751e-01, -5.07539654e-01, ...,
         4.89133857e-01, -1.03395269e+00, -8.61050113e-01],
       [ 1.40444556e+00,  7.77155778e-01, -2.52076279e-01, ...,
        -2.23868614e-01,  1.85718440e+00, -4.84725007e-01],
       ...,
       [-2.02456406e-03, -1.25706134e+00,  6.16499196e-01, ...,
        -2.94133945e-02,  6.42906824e-01,  1.96138818e+00],
       [-6.06274859e-02,  4.50655383e+00, -1.37611513e+00, ...,
         1.39659155e+00, -9.76129945e-01,  4.56087756e-01],
       [ 4.66798811e-01,  7.20649747e-01, -6.09725004e-01, ...,
        -2.23868614e-01, -6.87016236e-01, -7.66968836e-01]])
```

In [116]:

```
test_predict=model.predict(x_test_tf)
```

In [117]:

```
test_predict
```

Out[117]:

```
array([5, 5, 6, 5, 6, 5, 5, 5, 6, 6, 6, 5, 6, 5, 5, 7, 5, 6, 7, 5, 5, 5,
       6, 6, 5, 5, 6, 5, 5, 6, 5, 5, 6, 5, 6, 5, 6, 6, 5, 6, 5, 5, 6, 5,
       6, 6, 6, 6, 5, 6, 5, 5, 6, 7, 5, 5, 6, 5, 6, 5, 6, 6, 5, 5, 7, 5,
       6, 5, 7, 5, 6, 5, 6, 6, 6, 5, 7, 5, 6, 7, 5, 7, 5, 5, 6, 6, 5, 6,
       6, 5, 6, 5, 5, 6, 5, 6, 5, 6, 5, 5, 5, 5, 6, 6, 6, 6, 6, 5, 6, 5,
       6, 5, 6, 5, 6, 6, 6, 5, 5, 6, 6, 6, 6, 5, 5, 5, 6, 6, 5, 6, 6, 5,
       5, 6, 6, 5, 5, 5, 5, 6, 6, 6, 6, 5, 6, 5, 6, 5, 6, 5, 6, 6, 5, 6,
       6, 6, 5, 6, 5, 6, 7, 6, 6, 5, 5, 6, 5, 5, 5, 5, 5, 5, 6, 5, 7, 6,
       6, 5, 5, 5, 5, 7, 5, 7, 5, 6, 6, 6, 7, 5, 6, 6, 5, 6, 6, 5, 5, 5,
       6, 6, 5, 5, 5, 5, 7, 6, 5, 5, 6, 6, 7, 5, 6, 6, 6, 6, 6, 5, 6, 5,
       5, 6, 6, 6, 5, 5, 5, 7, 5, 5, 5, 5, 6, 6, 5, 6, 5, 6, 6, 5, 5, 5,
       6, 6, 5, 6, 6, 5, 6, 5, 6, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 5, 7,
       6, 7, 6, 5, 6, 6, 5, 6, 5, 5, 5, 5, 6, 6, 6, 5, 7, 5, 5, 5, 5, 6,
       5, 6, 5, 6, 5, 7, 6, 5, 5, 6, 5, 6, 6, 7, 5, 5, 6, 5, 5, 5, 6, 6,
       6, 7, 5, 5, 6, 5, 5, 6, 5, 5, 6, 5, 6, 5, 6, 5, 5, 5, 6, 5, 5, 6,
       6, 7, 5, 5, 6, 6, 6, 6, 5, 5, 6, 7, 5, 5, 6, 5, 6, 5, 6, 6, 6, 6,
       5, 5, 6, 6, 5, 5, 5, 5, 5, 5, 5, 6, 5, 6, 6, 5, 5, 5, 5, 5, 6, 6,
       5, 6, 5, 6, 5, 5, 5, 6, 6, 5, 6, 6, 6, 5, 5, 6, 5, 5, 5, 6, 6, 6,
       7, 6, 5, 6, 5, 5, 6, 5, 5, 6, 7, 6, 5, 5, 6, 7, 6, 6, 6, 6, 5, 7,
       5, 6, 6, 5, 5, 5, 6, 6, 5, 5, 6, 5, 7, 5, 5, 5, 6, 5, 5, 5, 5, 6,
       6, 6, 6, 5, 5, 5, 5, 6, 6, 5, 6, 6, 5, 5, 5, 6, 7, 6, 6, 5, 5, 5,
       5, 5, 6, 5, 5, 5, 5, 6, 7, 6, 6, 6, 5, 6, 6, 6, 6, 5, 6, 6, 6, 6,
       5, 6, 6, 6, 5, 5, 6, 6, 5, 5, 6, 5, 6, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       6, 6, 6, 6, 6, 6, 5, 5, 5, 7, 6, 6, 6, 5, 5, 5, 6, 6, 7, 7, 5, 5],
      dtype=int64)
```

In [118]:

```
from sklearn.metrics import accuracy_score
```

In [119]:

```
accuracy_score(y_test,test_predict)
```

Out[119]:

```
0.5984848484848485
```

###Hyperparamter Tuning

In [120]:

```
params={'kernel':['linear','poly','rbf','sigmoid'],'degree':[3,4],'tol':[0.0001,0.001,0.01,
```

In [121]:

```
from sklearn.model_selection import GridSearchCV
```

In [122]:

```
grid=GridSearchCV(estimator=model,param_grid=params,cv=10)
```

In [123]:

```
grid.fit(x_train_tf,y_train)
```

```
C:\Users\a\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:68
4: UserWarning: The least populated class in y has only 8 members, which is
less than n_splits=10.
  warnings.warn(
```

Out[123]:

```
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

In [124]:

```
grid.best_estimator_
```

Out[124]:

```
▾    SVC
SVC(tol=1)
```

In [125]:

```
grid.best_score_
```

Out[125]:

```
0.6321651090342678
```

In [126]:

```
model2=grid.best_estimator_
```

In [127]:

```
test_pred2=model2.predict(x_test_tf)
```

In [128]:

```
test_pred2
```

Out[128]:

```
array([5, 5, 6, 5, 6, 5, 5, 5, 6, 6, 6, 5, 6, 5, 5, 7, 5, 6, 7, 5, 5, 5,
       6, 6, 5, 6, 6, 5, 5, 6, 5, 5, 6, 5, 6, 5, 6, 6, 5, 6, 5, 5, 6, 5,
       6, 6, 6, 6, 5, 6, 5, 5, 6, 7, 5, 5, 6, 5, 6, 5, 5, 6, 5, 5, 7, 5,
       7, 5, 7, 5, 6, 5, 6, 6, 6, 5, 7, 5, 6, 7, 5, 7, 5, 5, 6, 7, 5, 6,
       6, 5, 6, 5, 5, 6, 5, 6, 5, 6, 5, 5, 5, 5, 6, 6, 6, 6, 6, 5, 5, 5,
       6, 5, 6, 5, 6, 6, 6, 5, 5, 6, 6, 5, 6, 5, 5, 5, 7, 5, 5, 6, 6, 5,
       5, 6, 6, 5, 5, 5, 5, 6, 6, 6, 7, 5, 6, 5, 6, 5, 6, 5, 6, 6, 5, 6,
       6, 6, 5, 6, 5, 6, 7, 6, 6, 5, 5, 6, 5, 5, 5, 5, 5, 5, 7, 5, 7, 6,
       6, 5, 5, 5, 5, 7, 6, 7, 5, 6, 6, 6, 7, 5, 6, 6, 5, 7, 6, 5, 5, 5,
       6, 6, 5, 5, 5, 5, 7, 6, 5, 6, 6, 6, 7, 5, 6, 6, 6, 6, 6, 5, 6, 5,
       5, 6, 7, 6, 5, 5, 5, 7, 5, 5, 6, 5, 7, 6, 5, 6, 5, 6, 6, 5, 5, 5,
       6, 6, 5, 6, 6, 5, 7, 5, 6, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 5, 7,
       6, 7, 5, 5, 6, 6, 5, 6, 5, 5, 5, 5, 6, 6, 6, 5, 7, 5, 5, 5, 5, 6,
       5, 6, 5, 6, 5, 7, 6, 5, 5, 6, 5, 6, 7, 7, 5, 5, 7, 5, 5, 5, 6, 6,
       6, 7, 5, 6, 6, 5, 5, 6, 5, 5, 7, 5, 6, 5, 6, 5, 5, 5, 6, 5, 5, 6,
       6, 7, 5, 5, 6, 6, 6, 6, 5, 5, 6, 7, 5, 5, 6, 5, 6, 5, 6, 6, 6, 7,
       5, 5, 6, 6, 5, 5, 5, 5, 5, 5, 5, 6, 5, 6, 6, 5, 5, 5, 5, 5, 6, 6,
       5, 6, 5, 6, 5, 5, 5, 6, 6, 5, 7, 6, 7, 5, 5, 6, 5, 5, 5, 6, 6, 6,
       7, 6, 5, 6, 5, 5, 6, 5, 5, 6, 6, 6, 5, 5, 6, 7, 6, 6, 6, 6, 5, 7,
       5, 7, 7, 5, 5, 5, 6, 6, 5, 5, 6, 5, 7, 5, 5, 5, 7, 5, 5, 5, 5, 6,
       6, 6, 7, 5, 6, 5, 5, 6, 5, 5, 6, 6, 5, 5, 6, 6, 7, 5, 6, 5, 5, 6,
       5, 5, 6, 5, 5, 5, 5, 6, 7, 6, 6, 6, 5, 6, 6, 6, 6, 5, 6, 6, 6, 6,
       5, 6, 6, 6, 5, 5, 7, 6, 5, 5, 6, 5, 6, 5, 5, 5, 5, 5, 5, 5, 6, 5,
       6, 6, 6, 7, 6, 6, 5, 5, 5, 7, 6, 6, 6, 5, 5, 5, 6, 6, 7, 7, 5, 5],
      dtype=int64)
```

In [129]:

```
accuracy_score(y_test,test_pred2)
```

Out[129]:

```
0.5852272727272727
```

In [130]:

```
from sklearn.metrics import precision_score,recall_score,confusion_matrix
```

In [131]:

```
confusion_matrix(y_test,test_pred2)
```

Out[131]:

```
array([[  0,   0,   2,   0,   0,   0],
       [  0,   0,  13,   6,   0,   0],
       [  0,   0, 169,  45,   3,   0],
       [  0,   0,  75, 116,  22,   0],
       [  0,   0,   0,  46,  24,   0],
       [  0,   0,   0,   2,   5,   0]], dtype=int64)
```

In [132]:

```
precision_score(y_test,test_pred2,average='micro')
```

Out[132]:

```
0.5852272727272727
```

In [133]:

```
recall_score(y_test,test_pred2,average='micro')
```

Out[133]:

```
0.5852272727272727
```

# Graduate Admission Prediction Using SVR

In [47]:

```
df=pd.read_csv("https://raw.githubusercontent.com/srinivasav22/Graduate-Admission-Predictio
```

In [48]:

```
df
```

Out[48]:

	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
...	...	...	...	...	...	...	...	...	...
495	496	332	108	5	4.5	4.0	9.02	1	0.87
496	497	337	117	5	5.0	5.0	9.87	1	0.96
497	498	330	120	5	4.5	5.0	9.56	1	0.93
498	499	312	103	4	4.0	5.0	8.43	0	0.73
499	500	327	113	4	4.5	4.5	9.04	0	0.84

500 rows × 9 columns

In [49]:

```
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
```

In [50]:

```
df.shape
```

Out[50]:

```
(500, 9)
```

In [51]:

```
df.isnull().sum()
```

Out[51]:

```
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 [52]:

```
df.describe()
```

Out[52]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Re
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500
mean	250.500000	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0
std	144.481833	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0
25%	125.750000	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0
50%	250.500000	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1
75%	375.250000	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1
max	500.000000	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1

In [53]:

```
df.columns
```

Out[53]:

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

In [56]:

```
df.dtypes
```

Out[56]:

```
Serial No.             int64
GRE Score              int64
TOEFL Score            int64
University Rating      int64
SOP                  float64
LOR                  float64
CGPA                 float64
Research               int64
Chance of Admit      float64
dtype: object
```

In [57]:

```
numeric_col=[columns for columns in df.columns if df[columns].dtype!='O']
```

In [64]:

```
numeric_col=numeric_col[1:]
```

In [65]:

```
numeric_col
```

Out[65]:

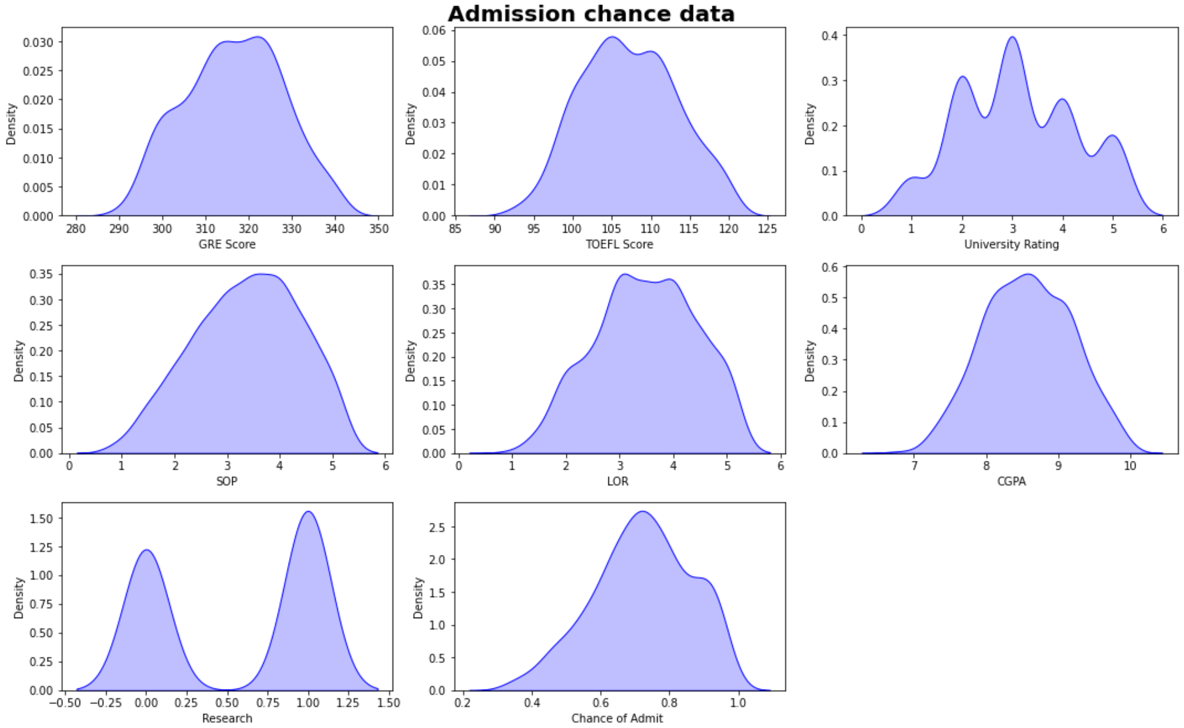
```
['GRE Score',
 'TOEFL Score',
 'University Rating',
 'SOP',
 'LOR ',
 'CGPA',
 'Research',
 'Chance of Admit ']
```

In [59]:

```
import seaborn as sns
import matplotlib.pyplot as plt
```

In [66]:

```
plt.figure(figsize=(15,15))
plt.suptitle("Admission chance data", fontsize=20, fontweight="bold")
for i in range(0,len(numeric_col)):
               plt.subplot(5,3,i+1)
               sns.kdeplot(x=df[numeric_col[i]],shade=True,color='b')
               plt.xlabel(numeric_col[i])
               plt.tight_layout()
```



In [68]:

```
x=df.drop('Chance of Admit ',axis=1)
```

In [69]:

```
y=df['Chance of Admit ']
```

In [70]:

```
x
```

Out[70]:

	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
...	...	...	...	...	...	...	...	...
495	496	332	108	5	4.5	4.0	9.02	1
496	497	337	117	5	5.0	5.0	9.87	1
497	498	330	120	5	4.5	5.0	9.56	1
498	499	312	103	4	4.0	5.0	8.43	0
499	500	327	113	4	4.5	4.5	9.04	0

500 rows × 8 columns

In [71]:

```
y
```

Out[71]:

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

In [72]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [73]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=10)
```

In [74]:

```
scale=StandardScaler()
```

In [75]:

```
x_train_tf2=scale.fit_transform(x_train)
```

In [76]:

```
x_test_tf2=scale.transform(x_test)
```

In [77]:

```
from sklearn.svm import SVR
```

In [78]:

```
svr=SVR()
```

In [81]:

```
svr.fit(x_train_tf2,y_train)
```

Out[81]:

```
▾ SVR
SVR()
```

In [82]:

```
y_test_pred=svr.predict(x_test_tf2)
```

In [83]:

```
y_test_pred
```

Out[83]:

```
array([0.87793627, 0.84529101, 0.69475389, 0.86168114, 0.70139032,
       0.77840156, 0.65246111, 0.79087535, 0.58150332, 0.79483703,
       0.84131007, 0.83250038, 0.86285841, 0.70469399, 0.81410334,
       0.73747862, 0.69284226, 0.73600249, 0.70898542, 0.64010805,
       0.72041624, 0.60225747, 0.60061743, 0.86859288, 0.49634259,
       0.87191845, 0.74087154, 0.51299302, 0.68259072, 0.77756367,
       0.89164322, 0.76456618, 0.58393949, 0.6274563 , 0.73699015,
       0.86896846, 0.83577545, 0.62013303, 0.69011353, 0.75030229,
       0.85766353, 0.60051554, 0.69115885, 0.89620826, 0.83055613,
       0.47281185, 0.59835581, 0.65763526, 0.78427726, 0.68600379,
       0.7842207 , 0.72828214, 0.56249564, 0.61706494, 0.75156514,
       0.7871982 , 0.86466527, 0.6013826 , 0.63957143, 0.8711468 ,
       0.60781057, 0.82632539, 0.75689743, 0.51246608, 0.83637061,
       0.49198243, 0.69416752, 0.74104567, 0.54967695, 0.75615813,
       0.87226857, 0.54941726, 0.54962423, 0.72439236, 0.66193436,
       0.68477606, 0.69344939, 0.74383412, 0.8494386 , 0.76947131,
       0.62386439, 0.6978024 , 0.61016821, 0.70555367, 0.82504614,
       0.75517614, 0.84124776, 0.53855504, 0.69368108, 0.84769058,
       0.8264763 , 0.6375172 , 0.52542317, 0.74717224, 0.77443104,
       0.70057109, 0.80270346, 0.83704315, 0.46284294, 0.70828786,
       0.82626981, 0.85550855, 0.73961891, 0.78044314, 0.87012807,
       0.49792623, 0.81801114, 0.4760047 , 0.82507627, 0.74229648,
       0.46335446, 0.78067591, 0.70830485, 0.58765295, 0.75613052,
       0.70215522, 0.66484084, 0.48493695, 0.83096337, 0.80025777,
       0.67980313, 0.88774933, 0.77401005, 0.84232574, 0.7902596 ])
```

In [84]:

```
from sklearn.metrics import r2_score
```

In [85]:

```
r2=r2_score(y_test,y_test_pred)
```

In [86]:

```
r2
```

Out[86]:

```
0.7538276071635838
```

In [87]:

```
ad_r2=1-((1-r2)*(len(y_test)-1))/(len(y_test)-(x_test.shape[1])-1)
```

In [88]:

```
ad_r2
```

Out[88]:

```
0.7368502007610723
```