Import modules

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

Importing dataset

In [3]:
```
data = pd.read_csv("E:/graduate/Admission_Predict.csv")
data.shape
```

Out[3]:
```
(400, 9)
```

Data summarization

In [4]:
```
data.head(2)
```

Out[4]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	NaN	NaN	4	4.5	4.5	9.65	1	0.92
1	2	312.0	107.0	4	4.0	4.5	8.87	1	0.76

In [5]:
```
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          399 non-null    float64
 2   TOEFL Score        399 non-null    float64
 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(6), int64(3)
memory usage: 28.2 KB
```

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

Out[6]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Resea
count	400.000000	399.000000	399.000000	400.000000	400.000000	400.000000	400.000000	400.000
mean	200.500000	316.619048	107.383459	3.087500	3.400000	3.452500	8.598925	0.547
std	115.614301	11.391182	6.053848	1.143728	1.006869	0.898478	0.597325	0.498
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.000000	6.800000	0.000
25%	100.750000	308.000000	103.000000	2.000000	2.500000	3.000000	8.167500	0.000
50%	200.500000	317.000000	107.000000	3.000000	3.500000	3.500000	8.610000	1.000
75%	300.250000	325.000000	112.000000	4.000000	4.000000	4.000000	9.072500	1.000
max	400.000000	340.000000	120.000000	5.000000	5.000000	5.000000	9.920000	1.000

In [7]:
```
data.isnull().sum()
```

Out[7]:
```
Serial No.           0
GRE Score            1
TOEFL Score          1
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

Data Pre-processing

In [8]:
```
data.drop('Serial No.', axis=1, inplace=True)
```

In [9]:
```
data.rename({'Chance of Admit ': 'Chance of Admit', 'LOR ':'LOR'}, axis=1, inplace=
```

In [10]:
```
X = data.iloc[:,:-1].values
Y = data.iloc[:,7:].values
```

In [11]:
```
print(X[:,:])
```

```
[[   nan    nan   4.   ...   4.5    9.65   1.  ]
 [312.   107.    4.   ...   4.5    8.87   1.  ]
 [316.   104.    3.   ...   3.5    8.     1.  ]
 ...
 [330.   116.    4.   ...   4.5    9.45   1.  ]
 [312.   103.    3.   ...   4.     8.78   0.  ]
 [321.   117.    4.   ...   4.     9.66   1.  ]]
```

Filling misssing values with mode

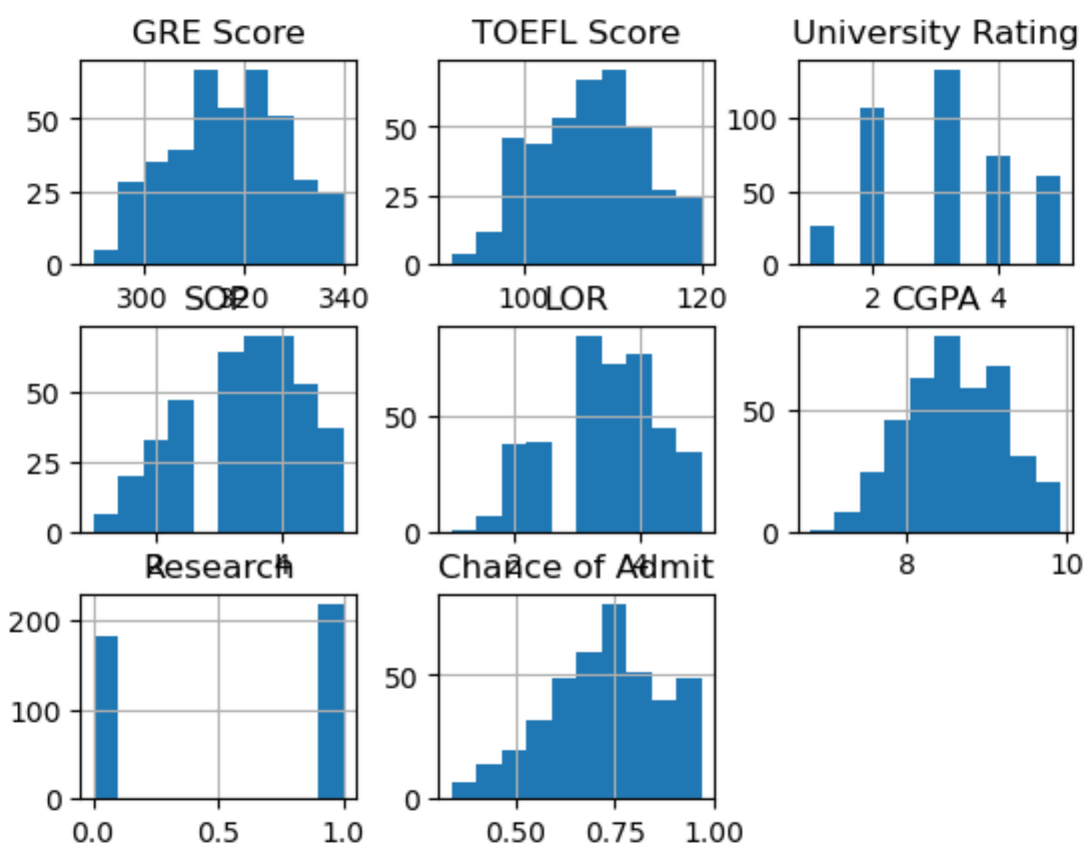
In [12]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='most_frequent')
imputer.fit(X[:, :3])
X[:, :3]= imputer.transform(X[:,:3])
print(X[0, :3])
```

```
[312. 110.   4.]
```

Data Visualization

In [13]:
```
import matplotlib.pyplot as plt
data.hist()
```

Out[13]:
```
array([[<AxesSubplot:title={'center':'GRE Score'}>,
        <AxesSubplot:title={'center':'TOEFL Score'}>,
        <AxesSubplot:title={'center':'University Rating'}>],
       [<AxesSubplot:title={'center':'SOP'}>,
        <AxesSubplot:title={'center':'LOR'}>,
        <AxesSubplot:title={'center':'CGPA'}>],
       [<AxesSubplot:title={'center':'Research'}>,
        <AxesSubplot:title={'center':'Chance of Admit'}>, <AxesSubplot:>]],
      dtype=object)
```



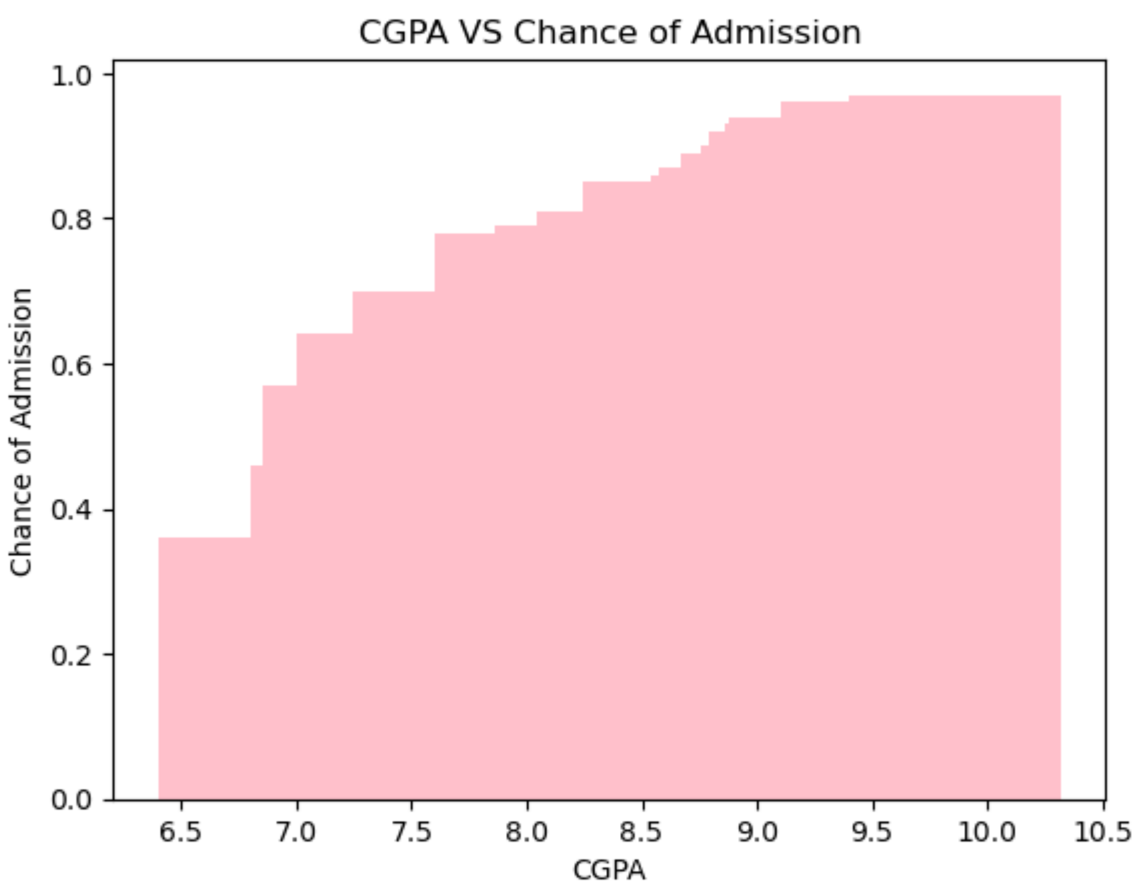
In [14]:
```
GRE = pd.DataFrame(data['GRE Score'])
GRE.describe()
```

Out[14]:

	GRE Score
count	399.000000
mean	316.619048
std	11.391182
min	290.000000
25%	308.000000
50%	317.000000
75%	325.000000
max	340.000000

In [15]:

```
plt.bar(X[:,5], Y[:,0],color = "pink")
plt.title("CGPA VS Chance of Admission")
plt.xlabel("CGPA")
plt.ylabel("Chance of Admission")
plt.show()
```



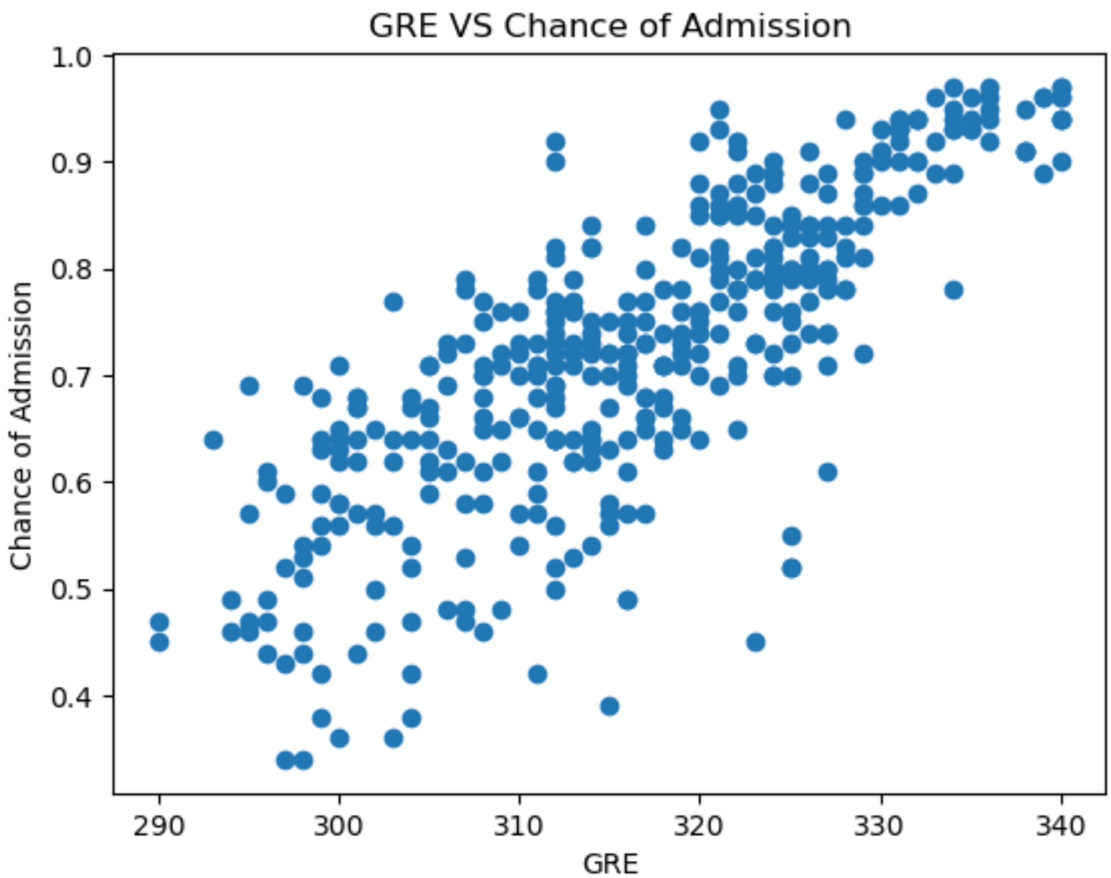
In [16]:

```
print(X)
```

```
[[312.   110.     4.   ...   4.5    9.65   1.  ]
 [312.   107.     4.   ...   4.5    8.87   1.  ]
 [316.   104.     3.   ...   3.5    8.     1.  ]
 ...
 [330.   116.     4.   ...   4.5    9.45   1.  ]
 [312.   103.     3.   ...   4.     8.78   0.  ]
 [321.   117.     4.   ...   4.     9.66   1.  ]]
```

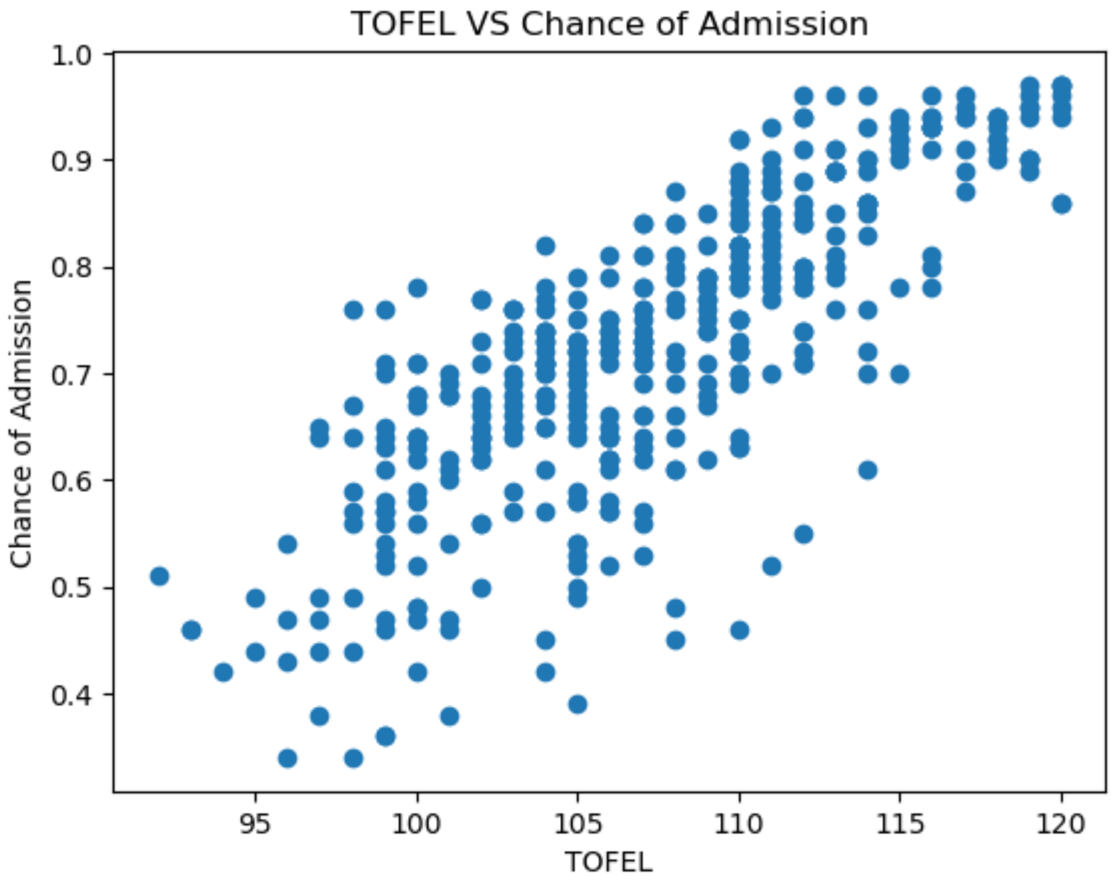
In [17]:

```
plt.scatter(X[:,0], Y)
plt.title("GRE VS Chance of Admission")
plt.xlabel("GRE")
plt.ylabel("Chance of Admission")
plt.show()
```

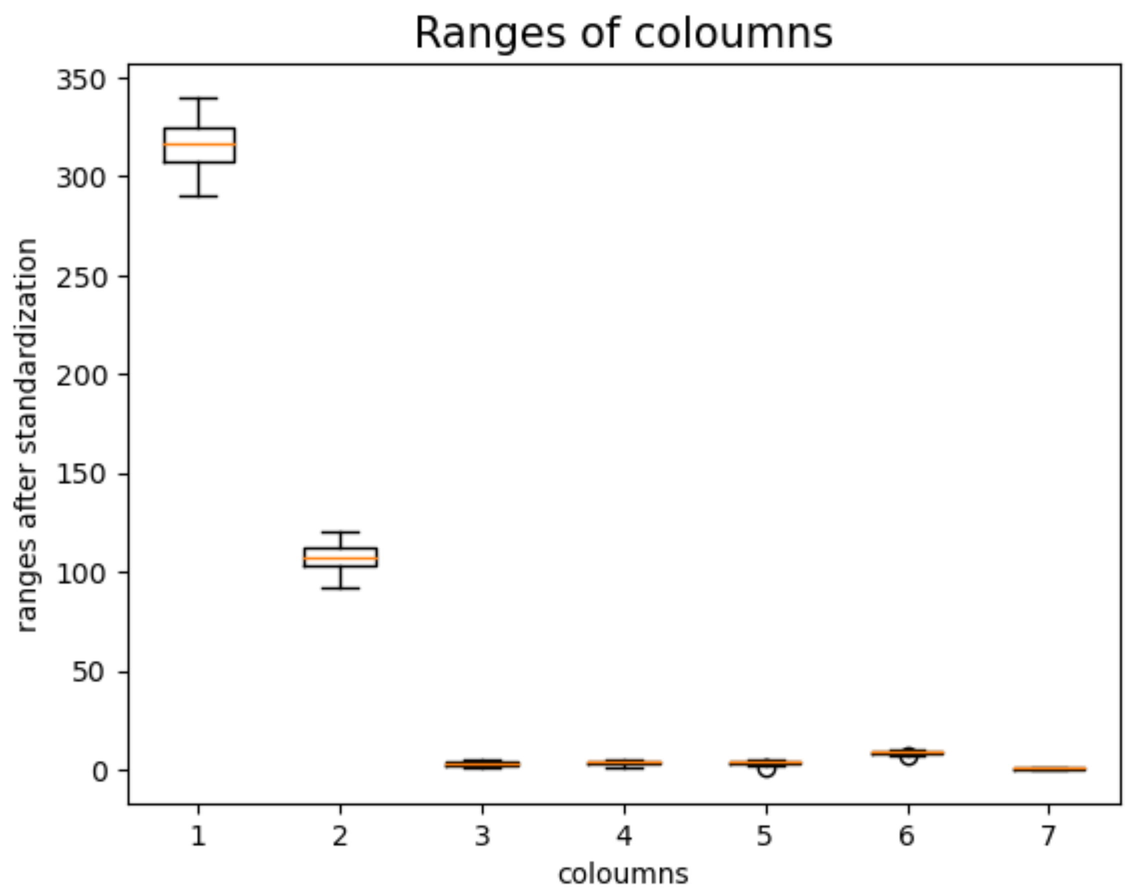


In [18]:

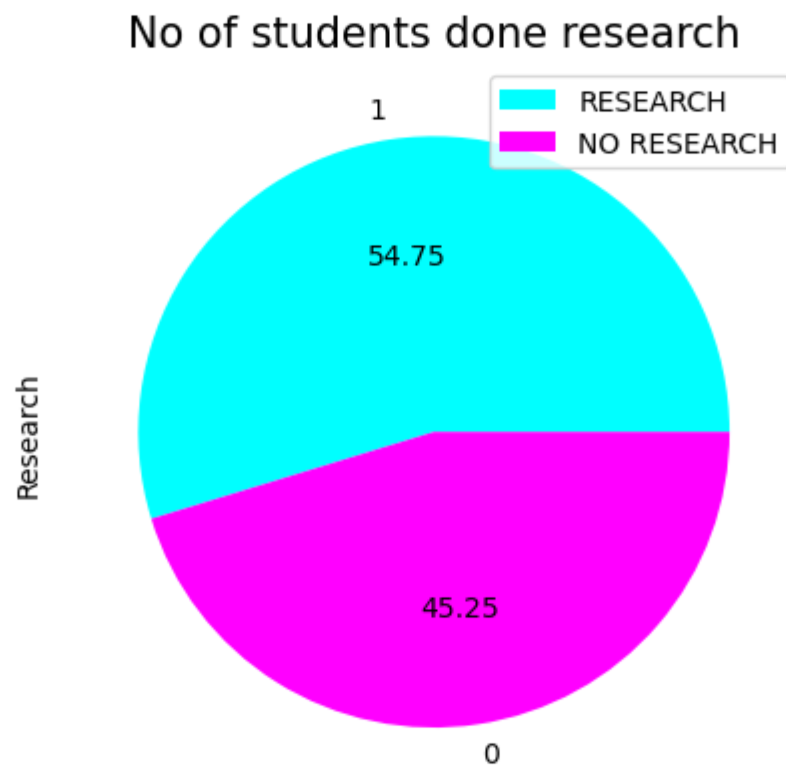
```
plt.scatter(X[:,1], Y)
plt.title("TOFEL VS Chance of Admission")
plt.xlabel("TOFEL")
plt.ylabel("Chance of Admission")
plt.show()
```



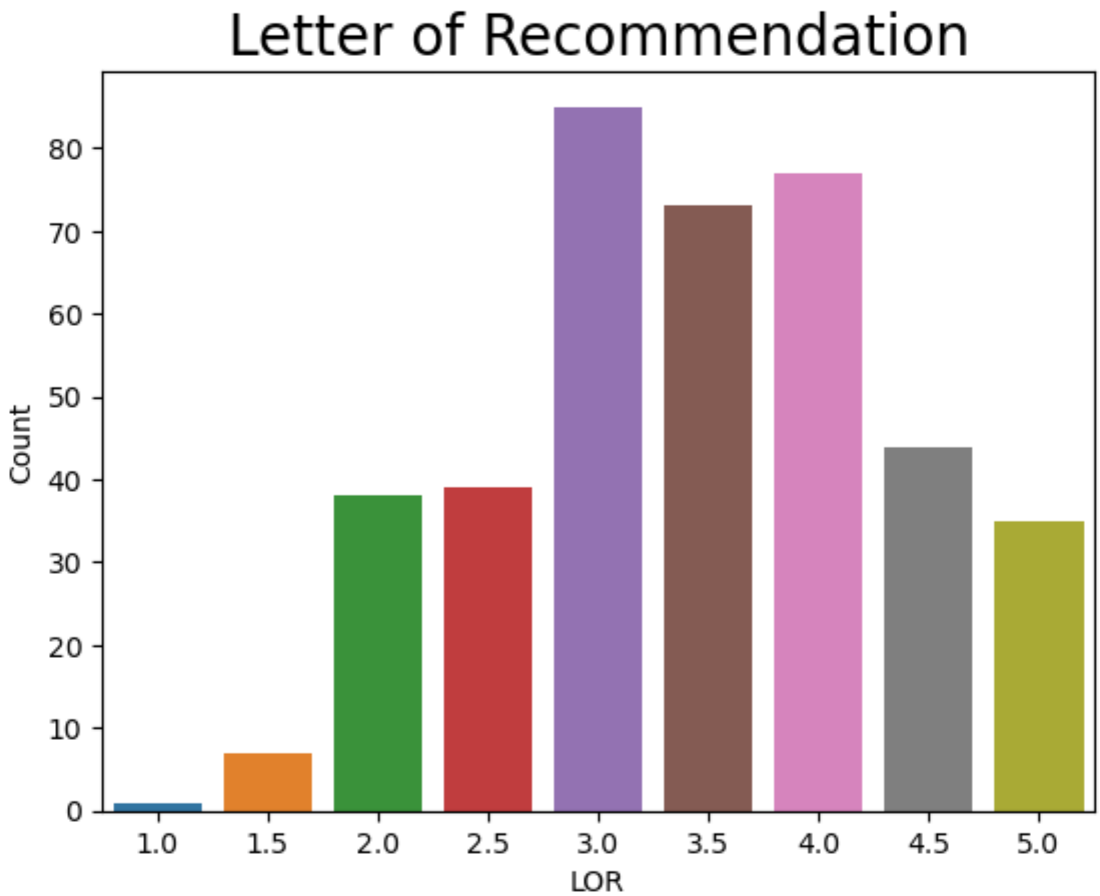
In [19]:
```
plt.boxplot(X[:,:])
plt.title('Ranges of coloumns',fontsize=15)
plt.xlabel("coloumns")
plt.ylabel("ranges after standardization")
plt.show()
```



In [20]:
```
data['Research'].value_counts().plot(kind='pie',textprops={'color':'black'},autopct
plt.title('No of students done research',fontsize=15)
plt.legend(['RESEARCH','NO RESEARCH'])
plt.show()
```



In [21]:
```
LOR = pd.DataFrame(data.groupby(['LOR']).count()['GRE Score'])
LOR.rename({'GRE Score':'Count'}, axis=1, inplace=True)
sns.barplot(x = LOR.index, y = LOR['Count']).set_title('Letter of Recommendation',
plt.show()
```



standardization

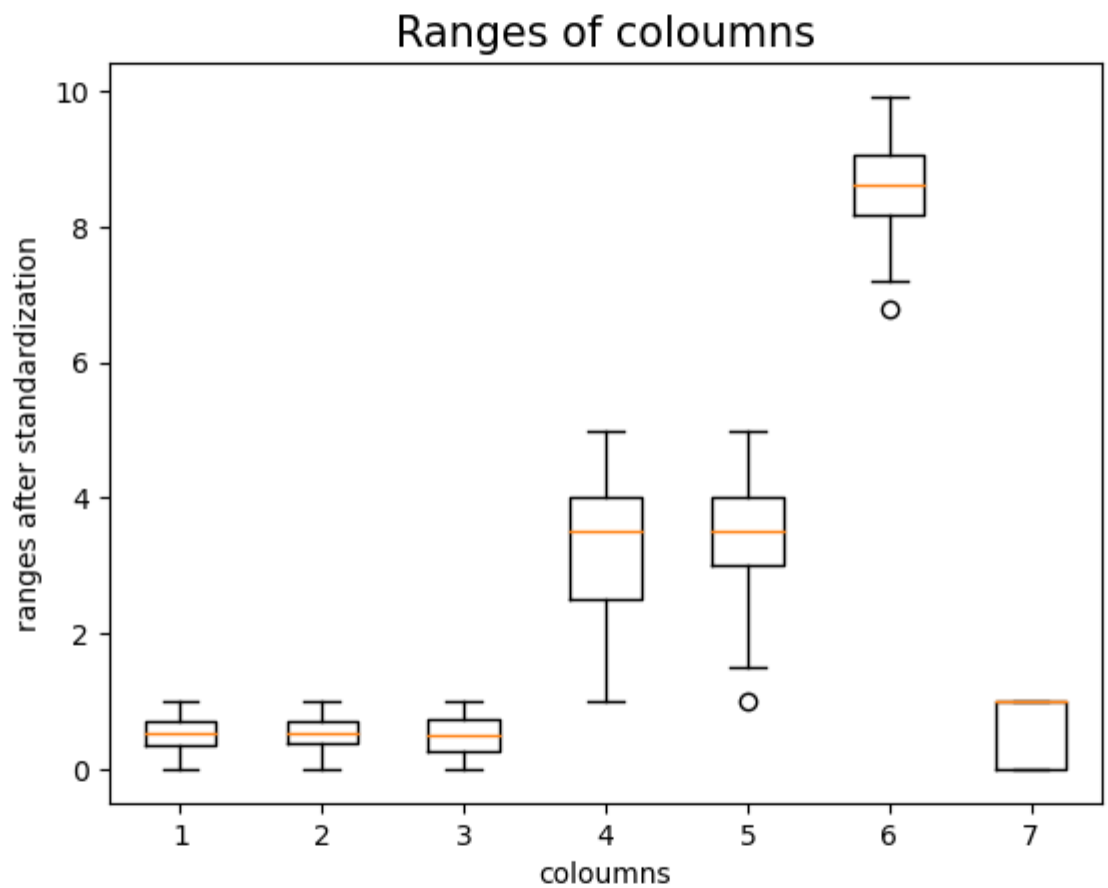
In [22]:
```
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
X[:,:3]= st_x.fit_transform(X[:,:3])
```

MinMax Scaler

In [23]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
X[:,0:3]= scaler.fit_transform(X[:,0:3])
print(X[:,0:3])
```

```
[[0.44       0.64285714 0.75      ]
 [0.44       0.53571429 0.75      ]
 [0.52       0.42857143 0.5       ]
 ...
 [0.8        0.85714286 0.75      ]
 [0.44       0.39285714 0.5       ]
 [0.62       0.89285714 0.75      ]]
```

In [24]:
```
plt.boxplot(X[:,:])
plt.title('Ranges of coloumns',fontsize=15)
plt.xlabel("coloumns")
plt.ylabel("ranges after standardization")
plt.show()
```



Saving processed dataset

In [25]:
```
arr = np.append(X, Y, axis=1)
col = ['GRE Score','TOEFL Score','University Rating','SOP','LOR','CGPA','Research',
newdf = pd.DataFrame(arr, columns=col)
print(newdf.isnull().sum())
newdf.to_csv('cleaned.csv')
```

```
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

In [26]:
```
Y_1=[1 if each > 0.82 else 0 for each in Y]
Y = np.array(Y_1)
```

Splitting Data into Train and Test

In [27]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(X, Y, test_size= 0.2, random_sta
```

Testing the following algorithms

- ## Logistic Regression

  Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression).

- ## Random Forest

  Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result. Random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

- ## Gaussian Navie Bayes

  Naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features. They are among the simplest Bayesian network models, but coupled with kernel density estimation, they can achieve higher accuracy levels. Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem.

- ## Support Vector Classifiers(SVCs)

  Support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training samples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new test samples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting).

- ## K-nearest Neighnours(KNN)

  In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

  - In k-NN classification, the output is a class membership. An object is classified by a

- In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.
- In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

LogisticRegression

In [28]:
```
from sklearn.linear_model import LogisticRegression
classifier= LogisticRegression()
classifier.fit(x_train, y_train)
```

Out[28]:
```
LogisticRegression()
```

In [29]:
```
y_logistic_pred= classifier.predict(x_test)
```

In [30]:
```
print('Train Score: ', classifier.score(x_train, y_train))
print('Test Score: ', classifier.score(x_test, y_test))
```

```
Train Score:  0.940625
Test Score:  0.9375
```

In [31]:
```
from sklearn.metrics import confusion_matrix

print(confusion_matrix(y_test, y_logistic_pred))
```

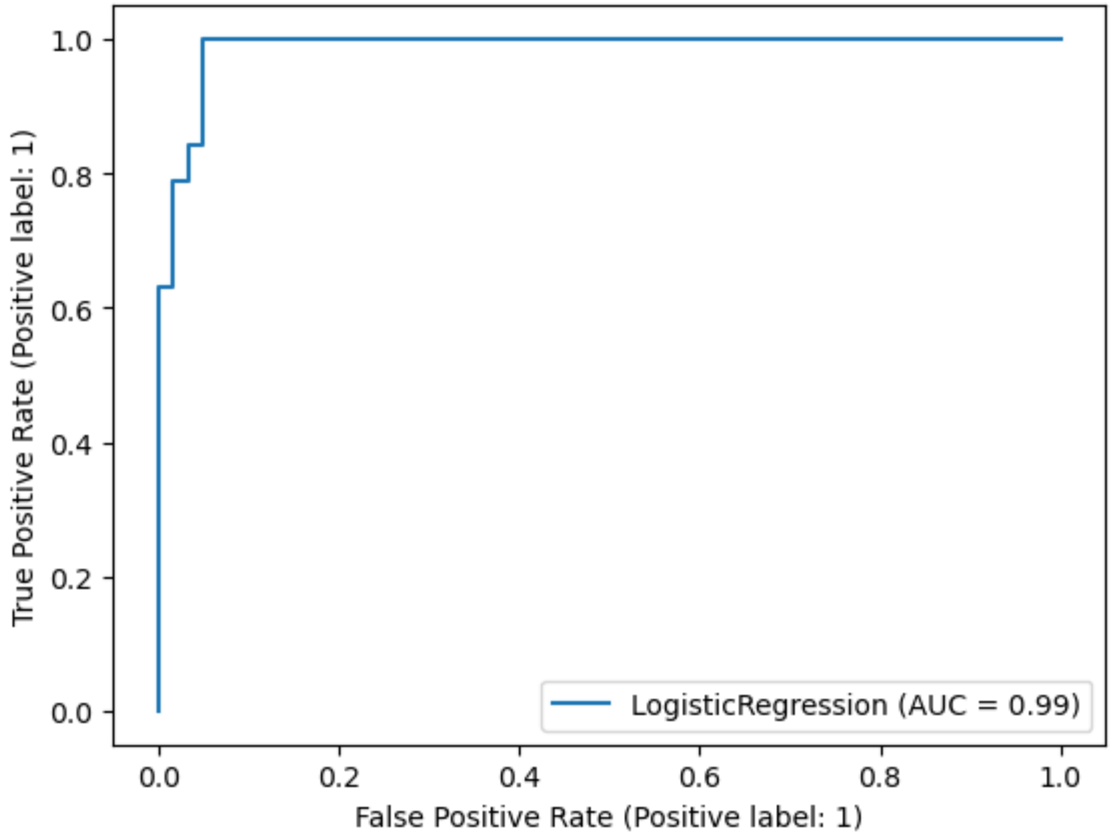
```
[[60  1]
 [ 4 15]]
```

In [32]:
```
from sklearn.metrics import precision_score, recall_score
print("precision_score: ", precision_score(y_test, y_logistic_pred))
print("recall_score: ", recall_score(y_test, y_logistic_pred))

from sklearn.metrics import f1_score
print("f1_score: ",f1_score(y_test, y_logistic_pred))
```

```
precision_score:  0.9375
recall_score:  0.7894736842105263
f1_score:  0.8571428571428572
```

ROC curve of logistic regression

In [33]:
```
from sklearn import metrics
metrics.plot_roc_curve(classifier, x_test, y_test)
plt.show()
```



SVC

In [34]:
```
from sklearn.svm import SVC
svm = SVC(random_state = 1)
svm.fit(x_train,y_train)
y_pred_svm = svm.predict(x_test)
print("score: ", svm.score(x_test,y_test))
```

```
score:  0.9375
```

In [35]:
```
from sklearn.metrics import precision_score, recall_score
print("precision_score: ", precision_score(y_test, y_pred_svm))
print("recall_score: ", recall_score(y_test, y_pred_svm))

from sklearn.metrics import f1_score
print("f1_score: ",f1_score(y_test, y_pred_svm))
```

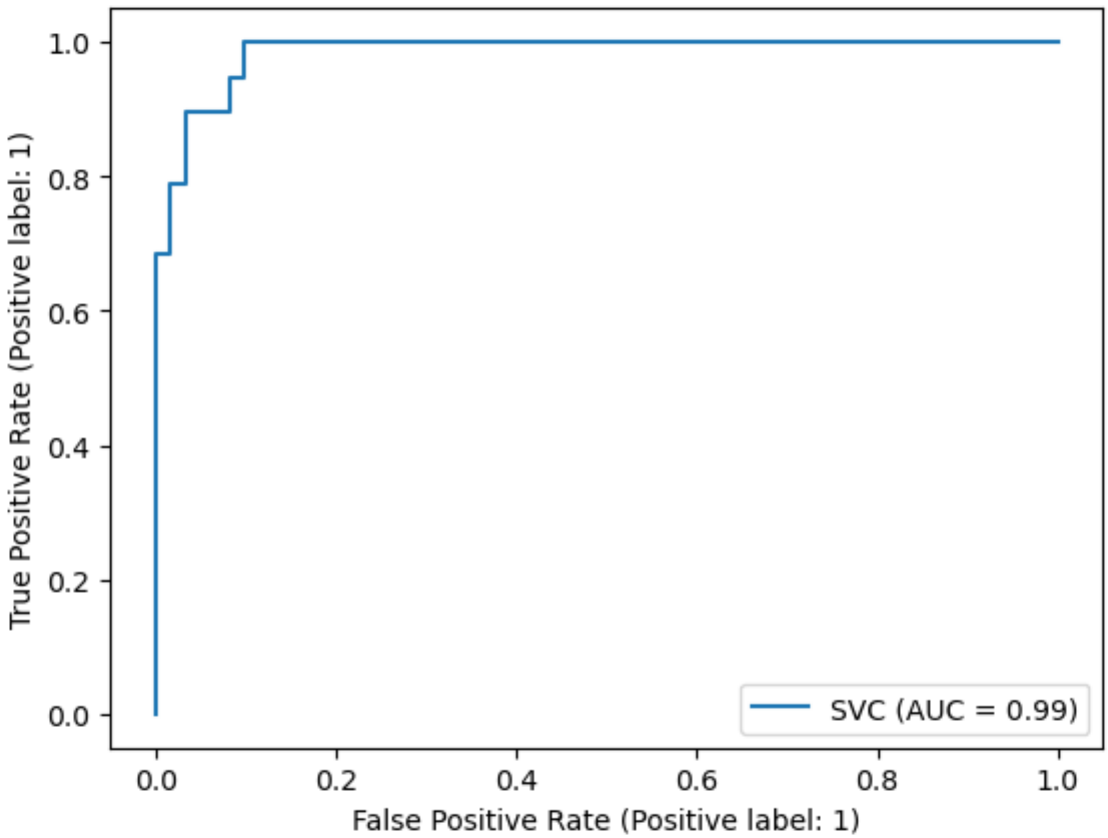
```
precision_score:  0.9375
recall_score:  0.7894736842105263
f1_score:  0.8571428571428572
```

In [36]:
```
from sklearn.metrics import confusion_matrix

print(confusion_matrix(y_test, y_pred_svm))
```

```
[[60  1]
 [ 4 15]]
```

ROC curve of SVC

In [37]:
```
from sklearn import metrics
metrics.plot_roc_curve(svm, x_test, y_test)
plt.show()
```



RandomForestClassifier

In [38]:
```
from sklearn.ensemble import RandomForestClassifier
RFC=RandomForestClassifier()
RFC.fit(x_train,y_train)
y_pred_RFC = RFC.predict(x_test)
print("score: ", RFC.score(x_test,y_test))
```

```
score:  0.925
```

In [39]:
```
from sklearn.metrics import precision_score, recall_score
print("precision_score: ", precision_score(y_test, y_pred_RFC))
print("recall_score: ", recall_score(y_test,y_pred_RFC))

from sklearn.metrics import f1_score
print("f1_score: ",f1_score(y_test, y_pred_RFC))
```

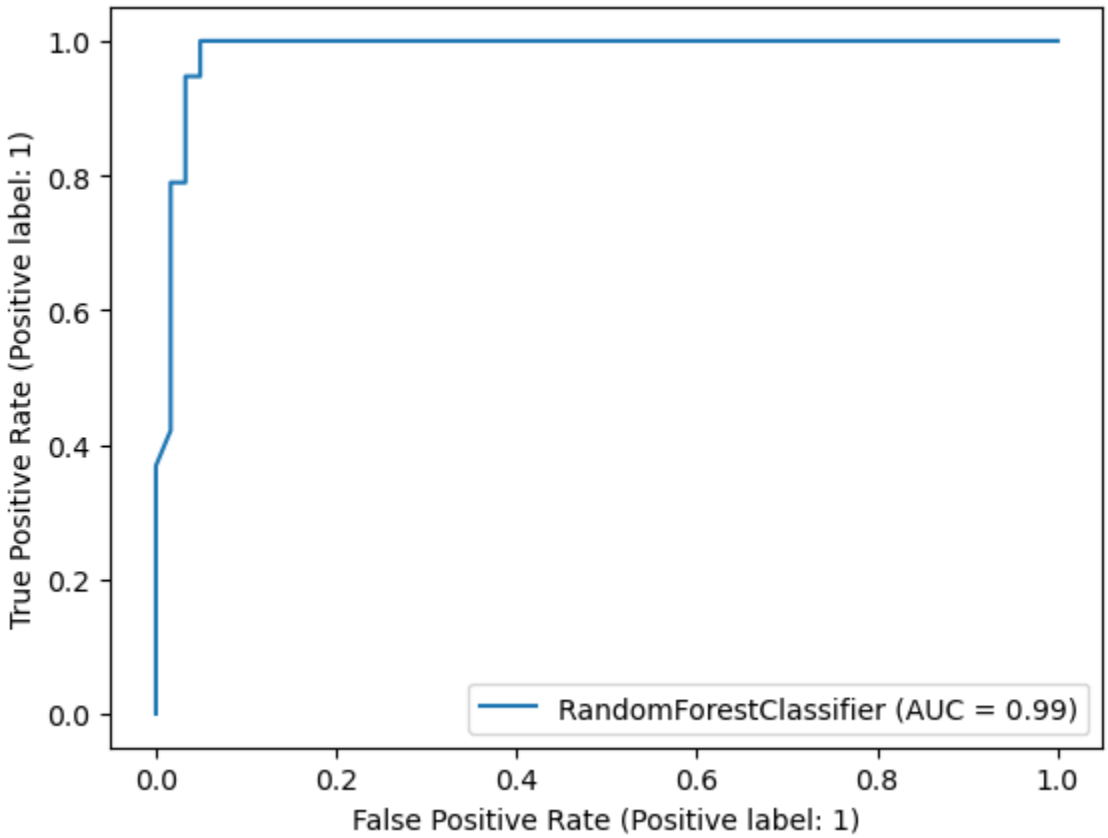
```
precision_score:  0.9333333333333333
recall_score:  0.7368421052631579
f1_score:  0.8235294117647058
```

In [40]:
```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred_RFC))
```

```
[[60  1]
 [ 5 14]]
```

ROC curve of Random forest classifier

In [41]:
```
from sklearn import metrics
metrics.plot_roc_curve(RFC, x_test, y_test)
plt.show()
```



NAVIE BAYES

In [42]:
```
from sklearn.naive_bayes import MultinomialNB
gn = MultinomialNB()
gn.fit(x_train,y_train)
gn.fit(x_train, y_train)
y_pred_gn = gn.predict(x_test)
print("score: ", gn.score(x_test,y_test))
```

```
score:  0.7625
```

In [43]:
```
from sklearn.metrics import precision_score, recall_score
print("precision_score: ", precision_score(y_test, y_pred_gn))
print("recall_score: ", recall_score(y_test,y_pred_gn))

from sklearn.metrics import f1_score
print("f1_score: ",f1_score(y_test, y_pred_gn))
```

```
precision_score:  0.0
recall_score:  0.0
f1_score:  0.0
```

Naive Bayes Classifiers

In [44]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train, y_train)
y_pred_gnb = gnb.predict(x_test)
from sklearn import metrics
print("Gaussian Naive Bayes model accuracy:",metrics.accuracy_score(y_test, y_pred_
```

```
Gaussian Naive Bayes model accuracy: 0.9375
```

In [45]:
```
from sklearn.metrics import precision_score, recall_score
print("precision_score: ", precision_score(y_test, y_pred_gnb))
print("recall_score: ", recall_score(y_test,y_pred_gnb))

from sklearn.metrics import f1_score
print("f1_score: ",f1_score(y_test, y_pred_gnb))
```

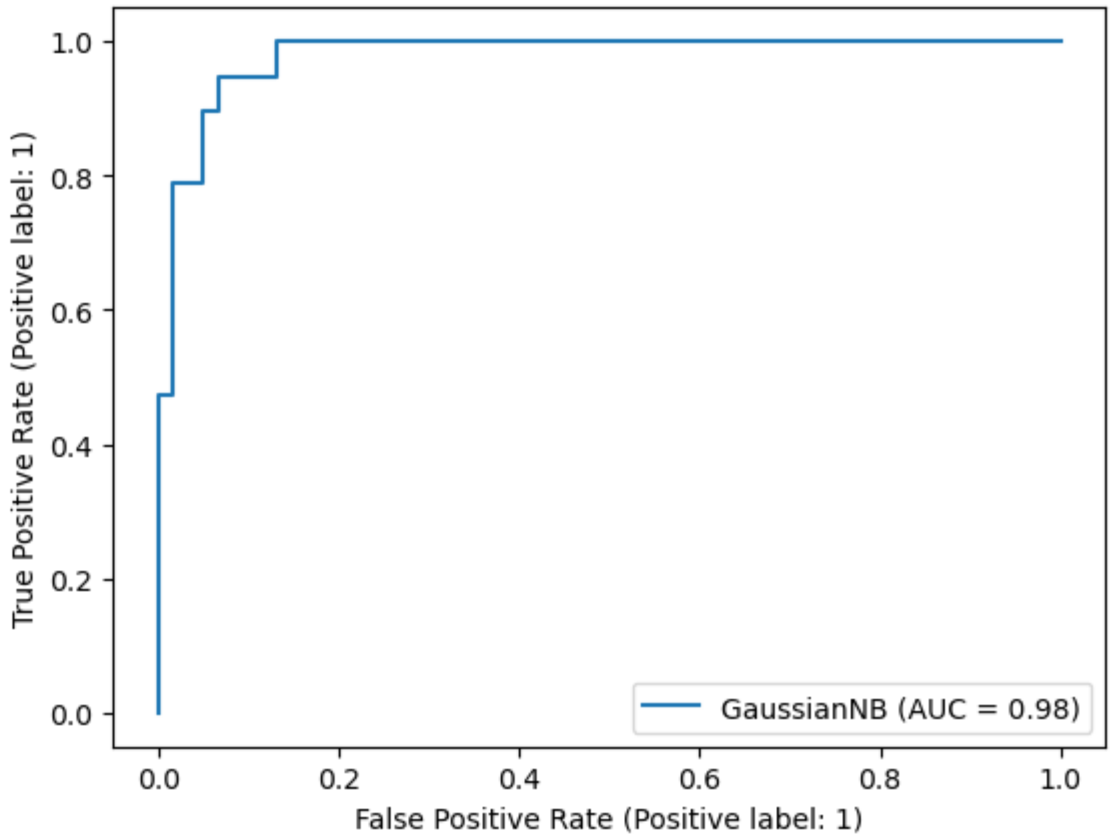
```
precision_score:  0.9375
recall_score:  0.7894736842105263
f1_score:  0.8571428571428572
```

In [46]:
```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test,  y_pred_gnb))
```

```
[[60  1]
 [ 4 15]]
```

ROC curve of Naive Bayes Classifiers

In [47]:
```
from sklearn import metrics
metrics.plot_roc_curve(gnb, x_test, y_test)
plt.show()
```



KNeighborsClassifier

In [48]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(x_train, y_train)
y_pred_knn = knn.predict(x_test)
print("KNN model accuracy:",metrics.accuracy_score(y_test, y_pred_knn))
```

```
KNN model accuracy: 0.925
```

In [49]:
```
from sklearn.metrics import precision_score, recall_score
print("precision_score: ", precision_score(y_test, y_pred_knn))
print("recall_score: ", recall_score(y_test,y_pred_knn))
from sklearn.metrics import f1_score
print("f1_score: ",f1_score(y_test, y_pred_knn))
```

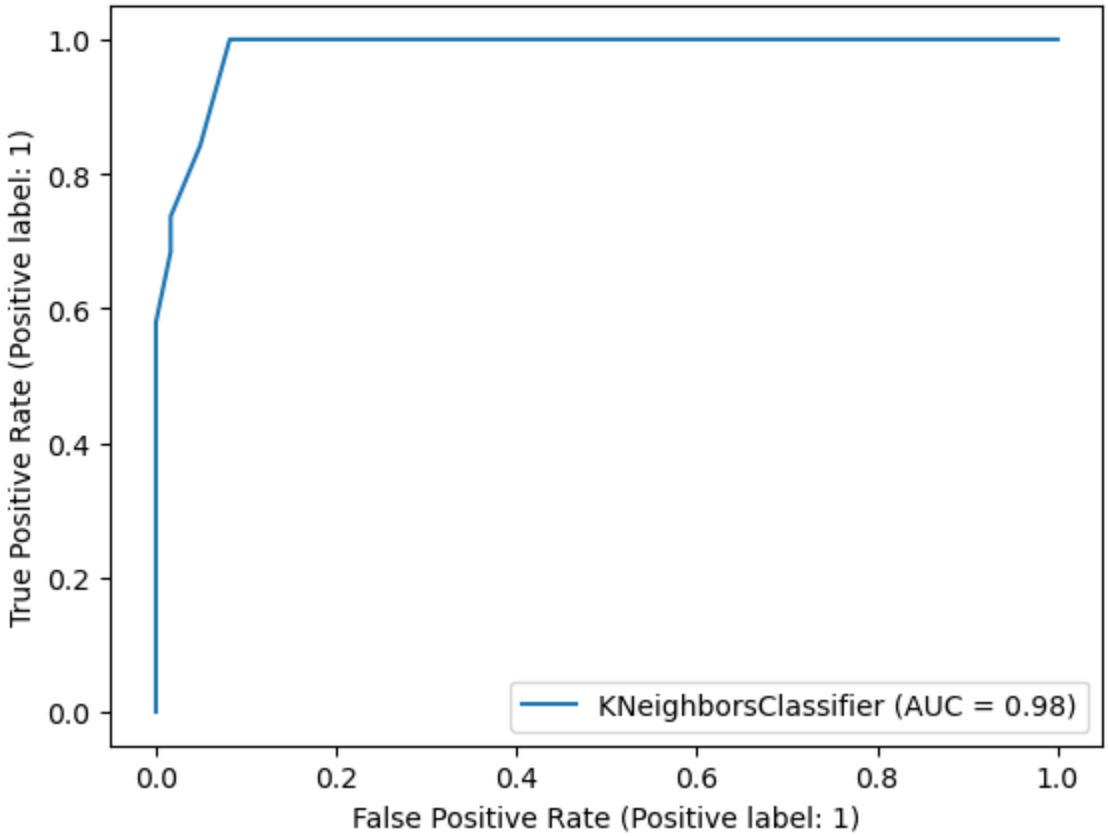
```
precision_score:  0.9333333333333333
recall_score:  0.7368421052631579
f1_score:  0.8235294117647058
```

In [50]:
```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test,  y_pred_knn))
```

```
[[60  1]
 [ 5 14]]
```

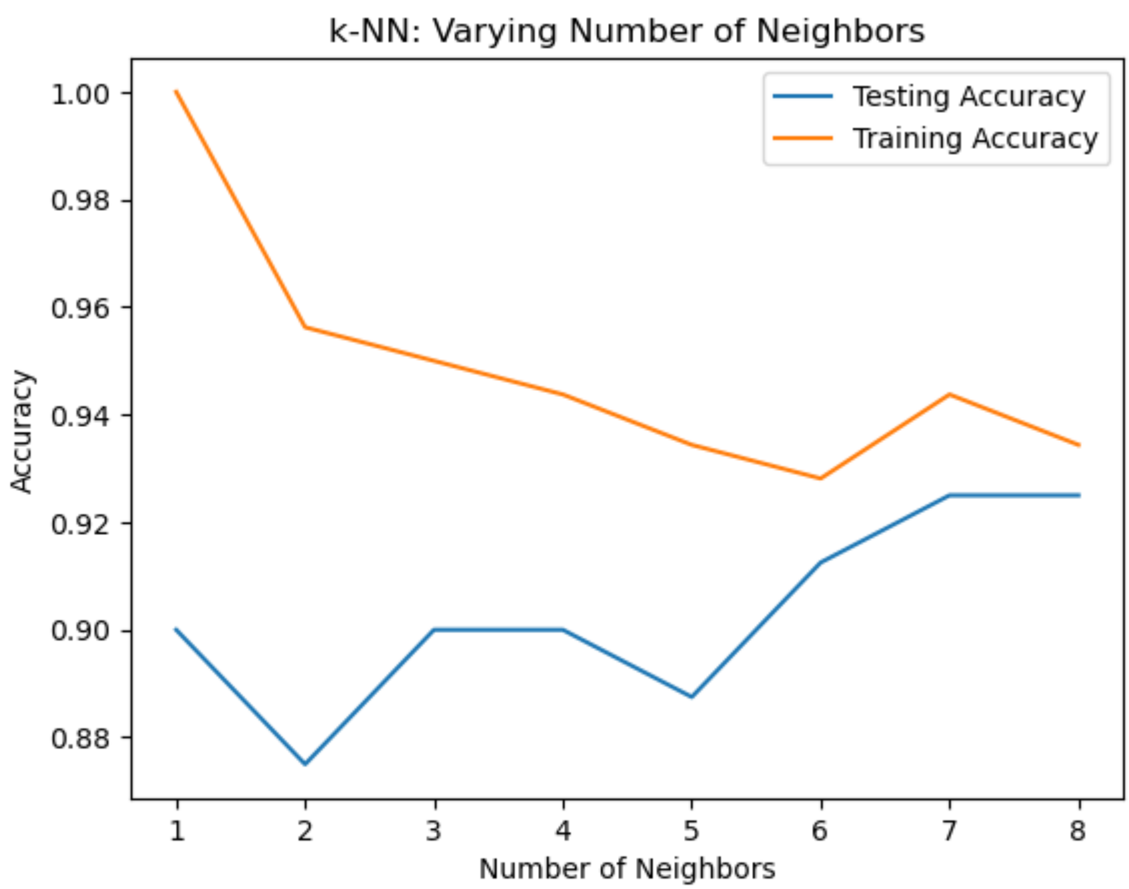
ROC curve of KNeighborsClassifier

In [51]:
```
from sklearn import metrics
metrics.plot_roc_curve(knn, x_test, y_test)
plt.show()
```



Accuracy for different values of K

In [52]:
```
no_neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(no_neighbors))
test_accuracy = np.empty(len(no_neighbors))

for i, k in enumerate(no_neighbors):
    # We instantiate the classifier
    knn = KNeighborsClassifier(n_neighbors=k)
    # Fit the classifier to the training data
    knn.fit(x_train,y_train)

    # Compute accuracy on the training set
    train_accuracy[i] = knn.score(x_train, y_train)

    # Compute accuracy on the testing set
    test_accuracy[i] = knn.score(x_test, y_test)

# Visualization of k values vs accuracy

plt.title('k-NN: Varying Number of Neighbors')
plt.plot(no_neighbors, test_accuracy, label = 'Testing Accuracy')
plt.plot(no_neighbors, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.show()
```



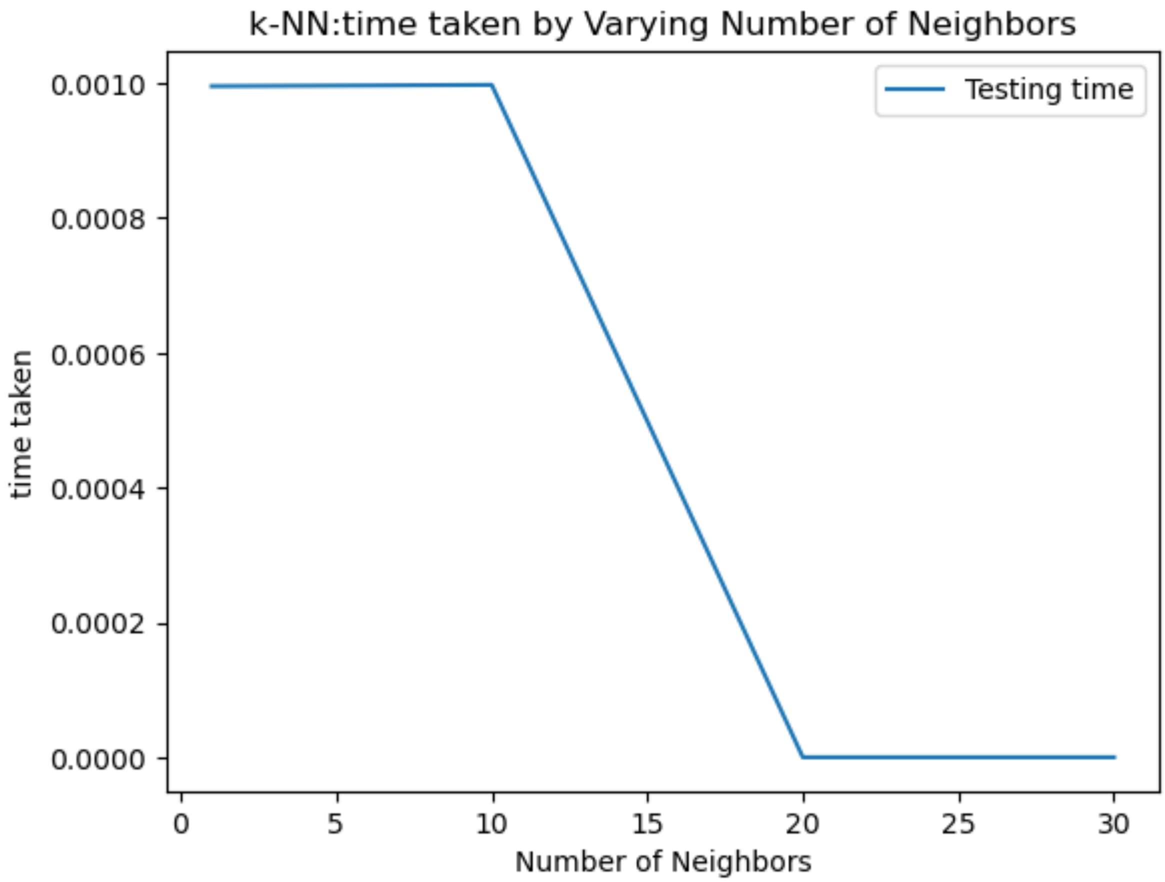
k-NN:time taken by Varying Number of K

Decision tree using cart model

In [54]:
```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(x_train,y_train)
y_pred_clf = clf.predict(x_test)
print("score: ", clf.score(x_test,y_test))
```

```
score:  0.8875
```

In [53]:
```
no_neighbors = [1,10,20,30]
time_taken = np.empty(len(no_neighbors))

import time
for i, k in enumerate(no_neighbors):
    start=time.time()
    # We instantiate the classifier
    knn = KNeighborsClassifier(n_neighbors=k)
    # Fit the classifier to the training data
    knn.fit(x_train,y_train)
    end=time.time()
    time_taken[i]=end-start

# Visualization of k values vs accuracy

plt.title('k-NN:time taken by Varying Number of Neighbors')
plt.plot(no_neighbors, time_taken, label = 'Testing time')
plt.legend()
plt.xlabel('Number of Neighbors')
plt.ylabel('time taken')
plt.show()
```



In [55]:
```
from sklearn.metrics import precision_score, recall_score
print("precision_score: ", precision_score(y_test, y_pred_clf))
print("recall_score: ", recall_score(y_test,y_pred_clf))

from sklearn.metrics import f1_score
print("f1_score: ",f1_score(y_test, y_pred_clf))
```

```
precision_score:  0.7777777777777778
recall_score:  0.7368421052631579
f1_score:  0.7567567567567567
```

CROSS VALIDATION SCORES

In [56]:
```
from sklearn.model_selection import cross_val_score
```

In [57]:
```
cross_val_score(LogisticRegression(),x_train,y_train)#average=0.93125
```

Out[57]:
```
array([0.921875, 0.890625, 0.984375, 0.921875, 0.9375  ])
```

In [58]:
```
cross_val_score(SVC(),x_train,y_train)#average=0.928125
```

Out[58]:
```
array([0.921875, 0.890625, 0.984375, 0.921875, 0.921875])
```

In [59]:
```
cross_val_score(RandomForestClassifier(),x_train,y_train)#average=0.946875
```

Out[59]:
```
array([0.921875, 0.90625 , 1.      , 0.9375  , 0.96875 ])
```

In [60]:
```
cross_val_score(GaussianNB(),x_train,y_train)#average=
```

Out[60]:
```
array([0.921875, 0.890625, 0.9375  , 0.875   , 0.9375  ])
```

In [61]:
```
cross_val_score(KNeighborsClassifier(),x_train,y_train)#average=
```

Out[61]:
```
array([0.9375  , 0.921875, 0.96875 , 0.921875, 0.90625 ])
```

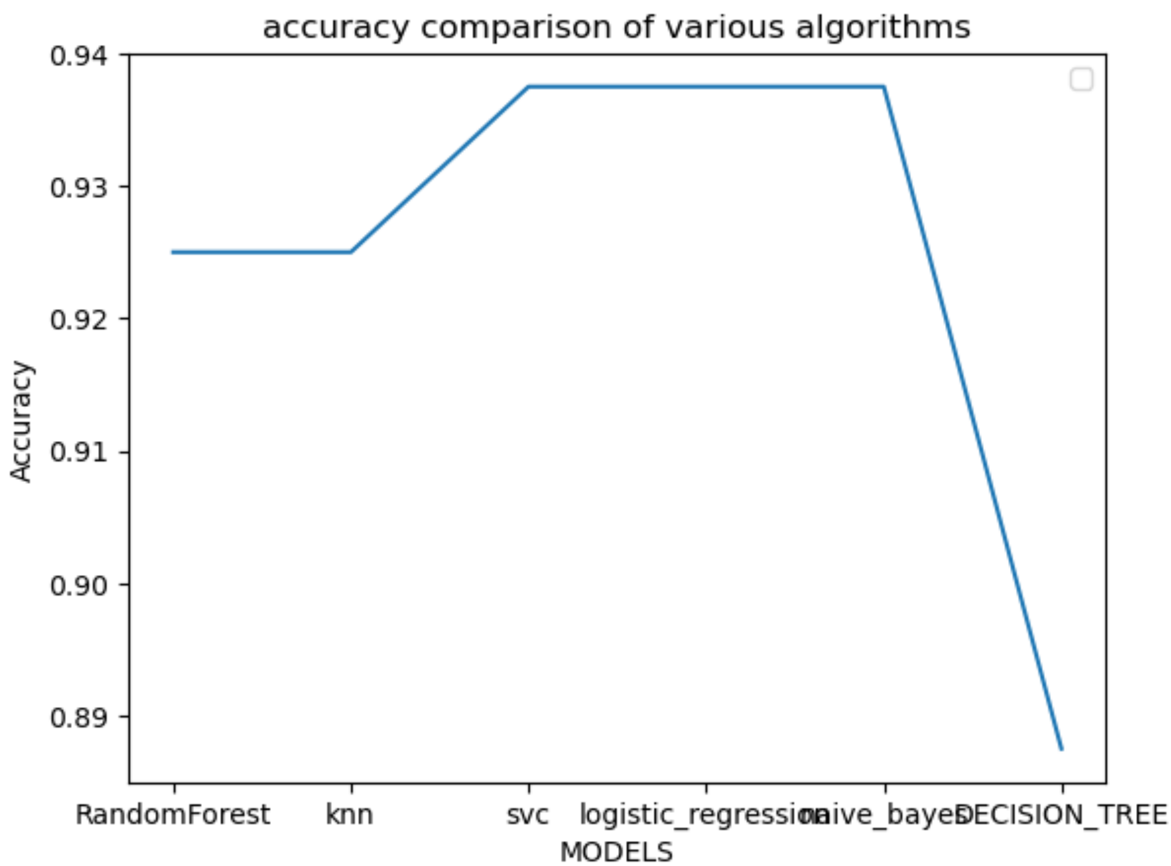
In [62]:
```
cross_val_score(DecisionTreeClassifier(),x_train,y_train)#average=
```

Out[62]:
```
array([0.9375  , 0.890625, 0.9375  , 0.90625 , 0.84375 ])
```

accuracy comparison of various algorithms

In [63]:
```
models=['RandomForest','knn','svc','logistic_regression','naive_bayes','DECISION_TR
accuracy=[metrics.accuracy_score(y_test, y_pred_RFC),metrics.accuracy_score(y_test,

plt.title('accuracy comparison of various algorithms')
plt.plot(models, accuracy)
plt.legend()
plt.xlabel('MODELS')
plt.ylabel('Accuracy')
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start
with an underscore are ignored when legend() is called with no argument.
```



So Random forest classifier is the best model

saving RFC classifier model

In [64]:
```
import pickle
with open("./college_predict.pkl", "wb") as f:
    pickle.dump(RFC, f)
```