Introduction

This is one of the most exciting projects that determine the probability of a student being accepted into a university-based on various scores, like (GRE, CGPA, TOEFL, etc).

This is a classification problem in which the target output is either False (not selected) or True (selected). Based on the seven independent attributes of a student, it predicts the target; each attribute is numerical or categorical.

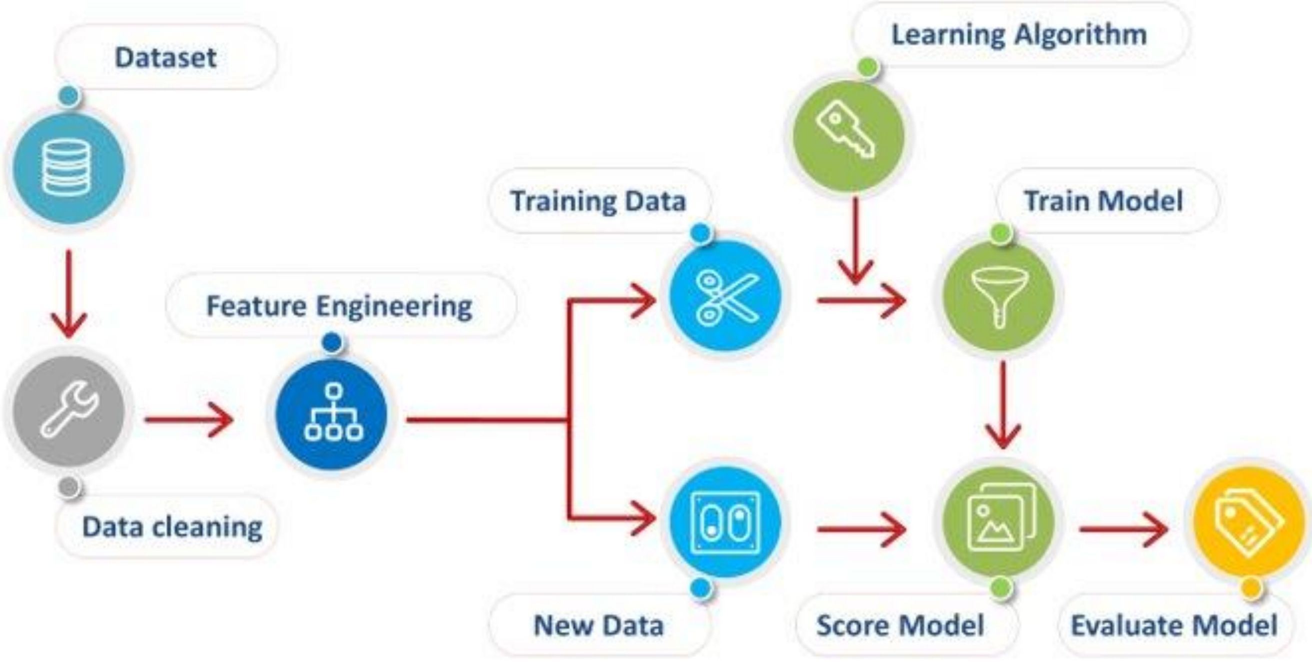
Attributes of the dataset are as follows:

GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit

Rather than using DNN, I used Machine Learning because the number of samples is 400.

Objectives

The goal is to predict whether a student will get admitted to a specific university. A variety of traditional approaches have been used to solve this problem, but none of those approaches captured the patterns (meaningful insights) that machine learning can.



Flow-digram

- Collect and load data
- Exploratory data analysis
- Data pre-processing
- Choose Optimal hyper-parameters of the algorithm
- Train and Evaluate algorithm
- Finalize the model

Project Implementation

According to the target variable, which is the probability of getting admitted into a university, the range is between 0 and 1, but I converted this problem from regression to classification by assigning 1 to the samples whose probability is greater than or equals to 0.5 and 0 to those with a probability below 0.5.

As this problem statement pertains to supervised learning, I tried 3 different Machine Learning models and selected the one that performs well on the test dataset.

Algorithms that I implemented are given as:

- Logistic Regression
- Support Vector Machine
- RandomForest Classifier

Data Pre-processing

1: Read the dataset

2. Dop unnecessary columns

3. Get statistics of the dataset and the number of null values

4. Find correlation among attributes (and remove if any attribute is not contributing towards the target attribute)

6. Convert categorical attributes to numerical format

7. Normalize the dataset to have 0 mean and 1 standard deviation.

8. Split the dataset into train-test set

Algorithms

LogisticRegression :

Logistic Regression is one of the most popular and basic techniques for solving classification problems. It uses the same underlying technique as Linear Regression. This method of classification uses the Logit function, hence the name "Logistic".

1. Tune the hyperparameter using GridSearchCV. Parameters tuned are :
   - C (inverse of regularization term)

2. Fit the model on the train set and evaluate it on the test set.

3. Measure the performance of the model using below metrics:
   - Accuracy
   - Reacll
   - ROC –AUC curve
   - Confusion matrix

4. Saved model weights

RandomForestClassifier

In a random forest classifier, different decision trees are fitted on a subset of the dataset and an average is used to increase predictive accuracy and prevent overfitting.

1. Tune the hyperparameter using GridSearchCV. Parameters tuned are :

- n_estimators (number of trees in the forest)
- max_features (number of features to consider when looking for the best split)
- max_depth (maximum depth of the tree)
- Criterion (function to measure the quality of a split)

2. Fit the model on the train set and evaluate it on the test set.

3. Measure the performance of the model using below metrics:
   - Accuracy
   - Reacll
   - ROC –AUC curve
   - Confusion matrix

4. Saved model weights

Support Vector Machine

In general, SVM modules (SVC, NuSVC, etc.) wrap the libsvm library and support different kernels while LinearSVC relies on liblinear and only supports linear kernels

1. Tune the hyperparameter using GridSearchCV. Parameters tuned are :
- C (Regularization paramete)
- gamma (Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’)
- Kernel (kernel type to be used in the algorithm)

2. Fit the model on the train set and evaluate it on the test set.

3. Measure the performance of the model using below metrics:
   - Accuracy
   - Reacll
   - ROC –AUC curve
   - Confusion matrix
4. Saved model weights

[90]

```
#import required libraries
import pickle
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,recall_score,roc_auc_score,confusion_matrix, plot_roc_c
```

[91]

```
#read the dataset
data = pd.read_csv('Admission_Predict.csv')
data.head()
```

	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

```
#drop unnecessary columns
data.drop(["Serial No."],axis=1,inplace=True)
data.head()
```

	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

```python
#get the dataset stats
data.describe()
```

	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
mean	316.807500	107.410000	3.087500	3.400000	3.452500	8.598925	0.547500	0.724350
std	11.473646	6.069514	1.143728	1.006869	0.898478	0.596317	0.498362	0.142609
min	290.000000	92.000000	1.000000	1.000000	1.000000	6.800000	0.000000	0.340000
25%	308.000000	103.000000	2.000000	2.500000	3.000000	8.170000	0.000000	0.640000
50%	317.000000	107.000000	3.000000	3.500000	3.500000	8.610000	1.000000	0.730000
75%	325.000000	112.000000	4.000000	4.000000	4.000000	9.062500	1.000000	0.830000
max	340.000000	120.000000	5.000000	5.000000	5.000000	9.920000	1.000000	0.970000

```python
1 # meta-data of the dataset
2 data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   GRE Score          400 non-null    int64  
 1   TOEFL Score        400 non-null    int64  
 2   University Rating  400 non-null    int64  
 3   SOP                400 non-null    float64
 4   LOR                400 non-null    float64
 5   CGPA               400 non-null    float64
 6   Research           400 non-null    int64  
 7   Chance of Admit    400 non-null    float64
dtypes: float64(4), int64(4)
memory usage: 25.1 KB
```

[95]

```python
1 # check the number of null values in each attribute
2 data.isnull().sum()
```

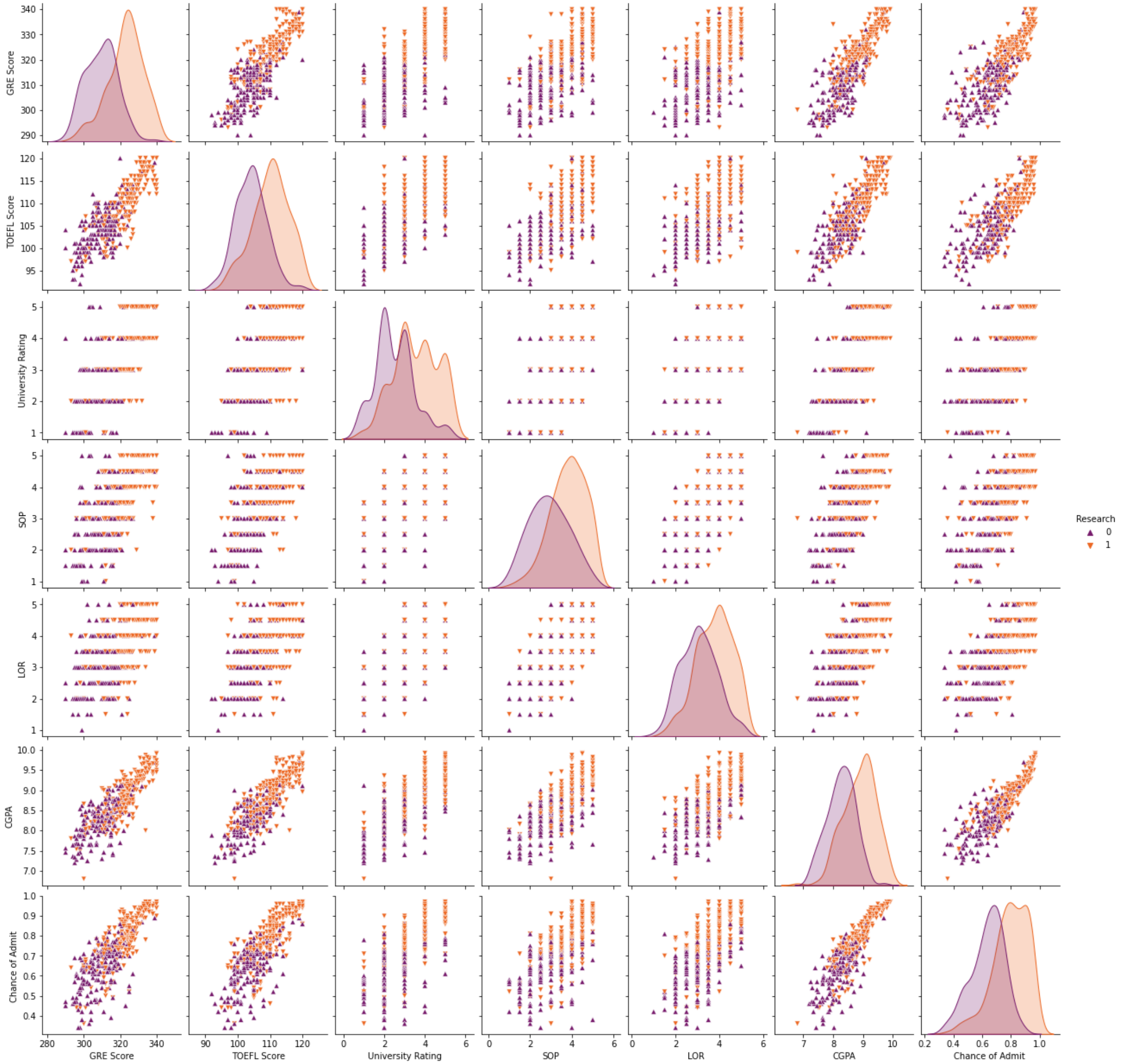
```
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

Exploratory Data Analysis

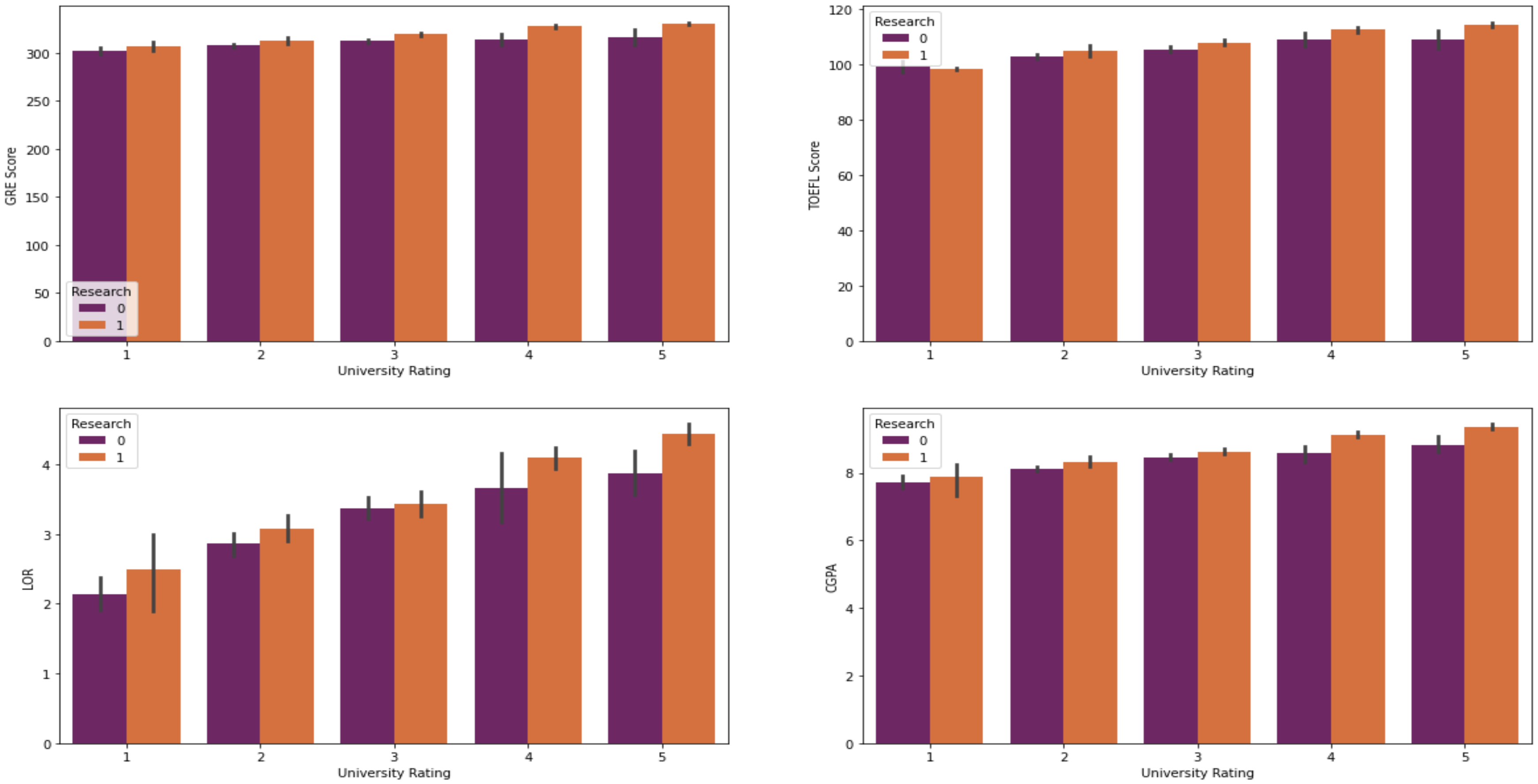
```python
# find the correlation among attributes
sns.heatmap(data.corr(),xticklabels=data.corr().columns.values,
            yticklabels=data.corr().columns.values,cmap='inferno')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff60fb97c90>
```

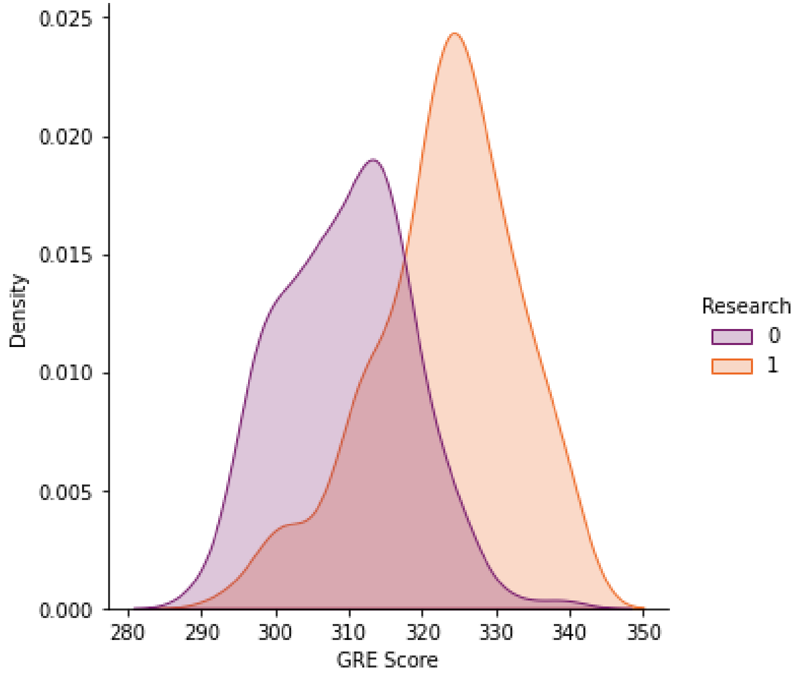
```
1 #pair-plot
2 sns.pairplot(data=data,hue='Research',markers=["^", "v"],palette='inferno')
```

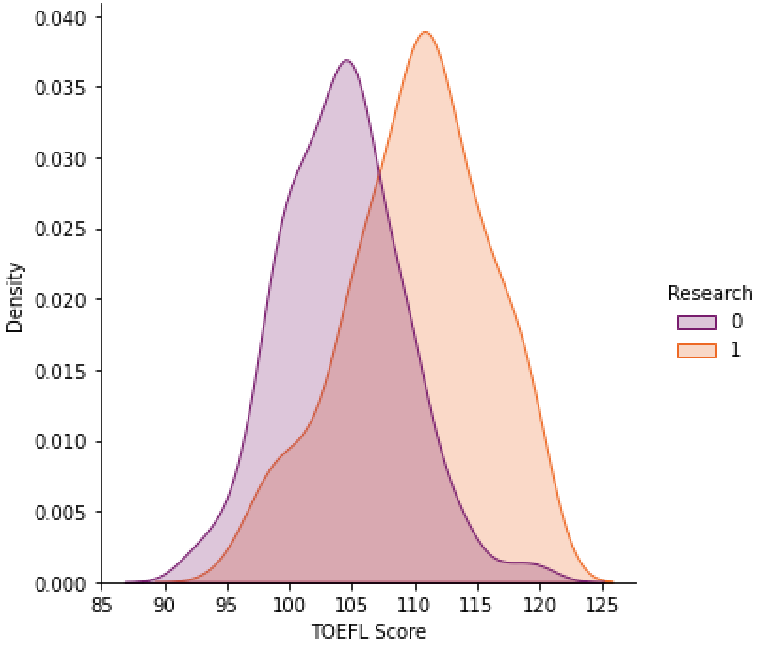


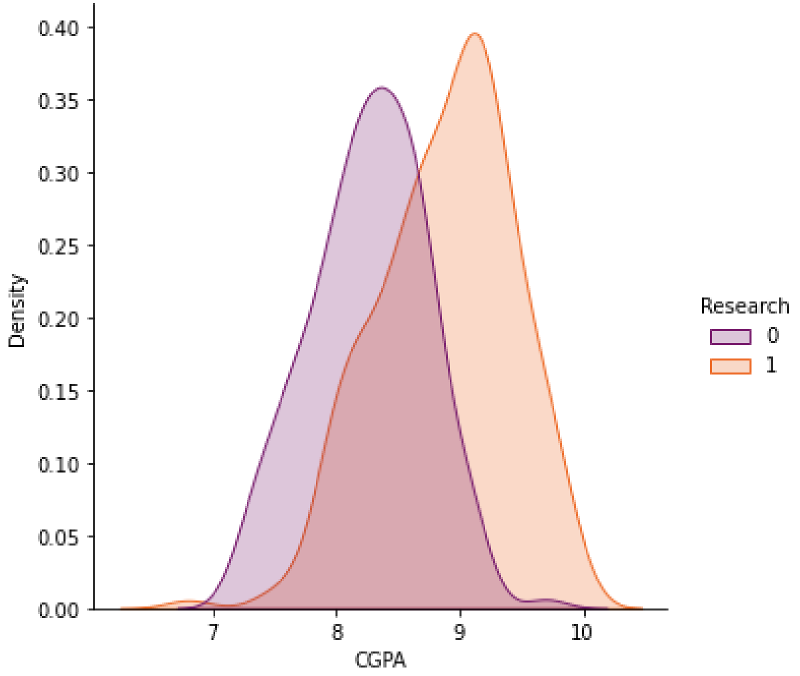
```python
#plot scores vs University ranking
fig, ax = plt.subplots(2, 2, figsize=(20, 12))

cols = list(data.columns[:2]) + list(data.columns[4:6])

k = 0
for i in range(2):
  for j in range(2):

    sns.barplot(x='University Rating',y=cols[k],data=data, hue='Research', ax=ax[i][j],
                palette='inferno')
    k += 1
```



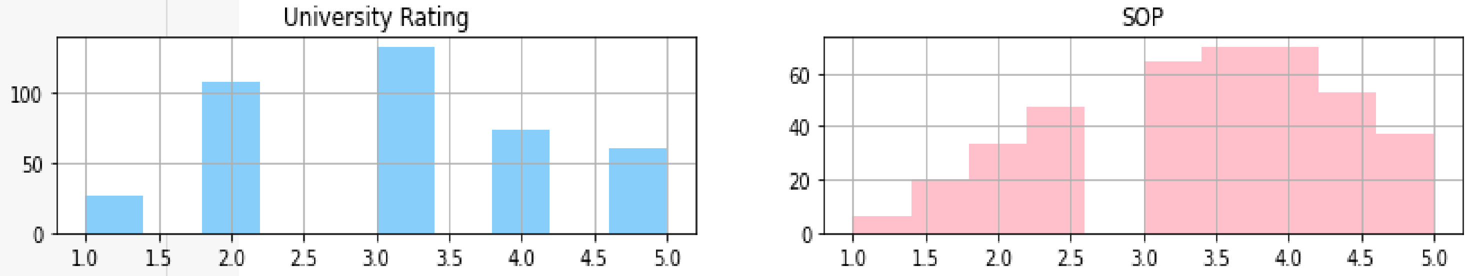
```python
# visualize the distribution of the marks
cols = ['GRE Score', 'TOEFL Score', 'CGPA']

for i in range(3):
  sns.displot(data, x=cols[i], hue="Research", kind="kde", fill=True,
                    palette='inferno')
```

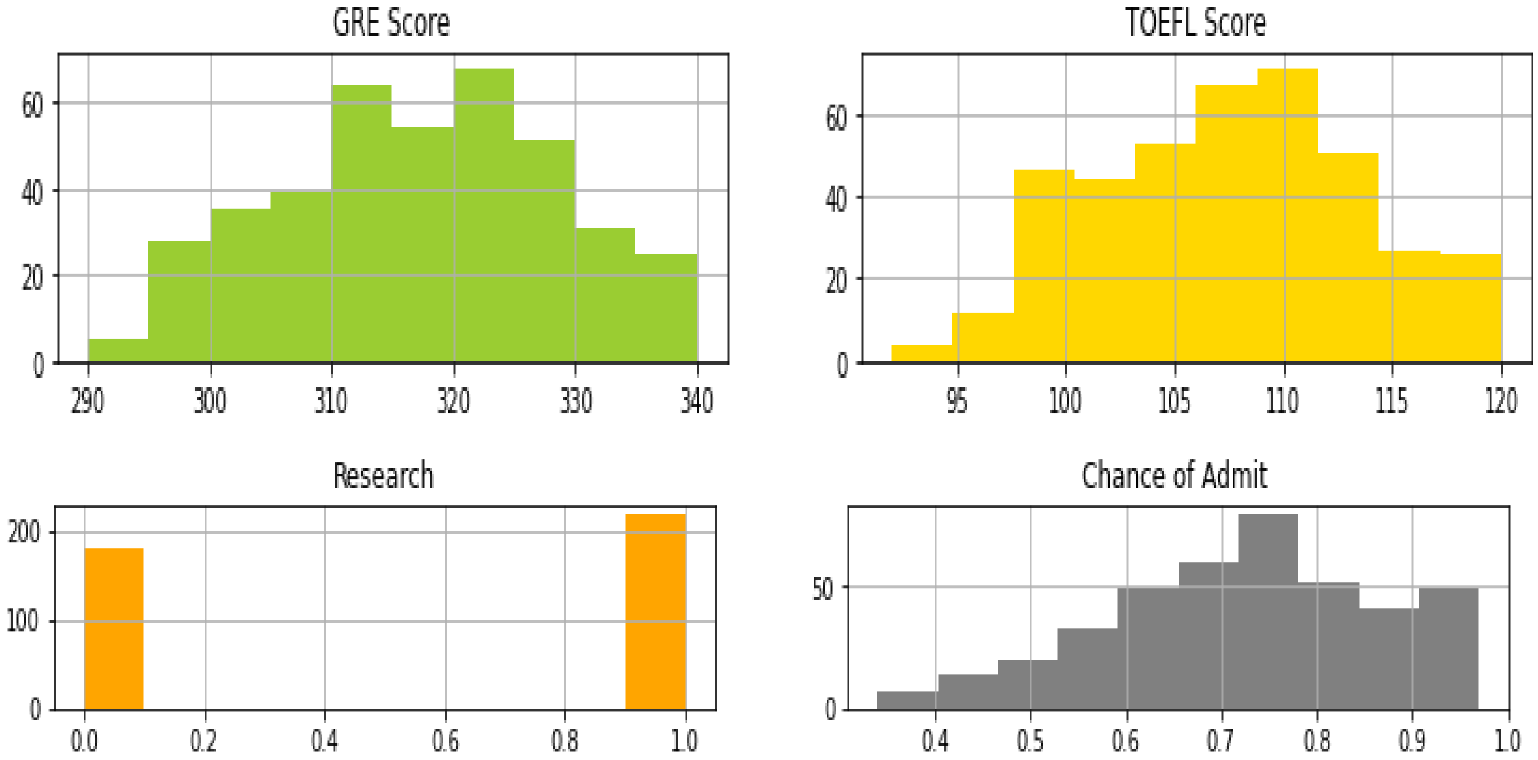






```python
category = list(data.columns)
color = ['yellowgreen','gold','lightskyblue','pink','red','purple','orange','gray']
start = True
for i in np.arange(4):
    fig = plt.figure(figsize=(14,8))
    plt.subplot2grid((4,2),(i,0))
    data[category[2*i]].hist(color=color[2*i],bins=10)
    plt.title(category[2*i])
    plt.subplot2grid((4,2),(i,1))
    data[category[2*i+1]].hist(color=color[2*i+1],bins=10)
    plt.title(category[2*i+1])

plt.subplots_adjust(hspace = 0.7, wspace = 0.2)
plt.show()
```





[101]

```
#categorical to numerical data conversion
data = pd.get_dummies(data, columns=['University Rating'])
data.head()
```

	GRE Score	TOEFL Score	SOP	LOR	CGPA	Research	Chance of Admit	University Rating_1	University Rating_2	University Rating_3	University Rating_4	University Rating_5
0	337	118	4.5	4.5	9.65	1	0.92	0	0	0	1	0
1	324	107	4.0	4.5	8.87	1	0.76	0	0	0	1	0
2	316	104	3.0	3.5	8.00	1	0.72	0	0	1	0	0
3	322	110	3.5	2.5	8.67	1	0.80	0	0	1	0	0
4	314	103	2.0	3.0	8.21	0	0.65	0	1	0	0	0

[102]
```
# independent attributes
x = data.drop('Chance of Admit ', axis=1).values

#target attribute
y = data['Chance of Admit '].values

print("Number of samples in the train-set {}".format(x.shape[0]))
```

```
Number of samples in the train-set 400
```

[103]
```
#normalize the dataset
sc = MinMaxScaler()
x = sc.fit_transform(x)
```

[104]
```
# convert probablities to the discrete value i.e, 0 or 1
y = (y>=0.5)
```

[105]
```
#split into train-test set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,
                                                    random_state=42, shuffle=True, stratify=y)
```

LogisticRegression

[106]

```
# Search optimal parameter for LogisticRegression
logistic = LogisticRegression(random_state =0)

grid_values = {'C': [0.001,0.01,0.1,1,10,100]}

grid_cv = GridSearchCV(logistic, param_grid = grid_values)
grid_cv.fit(x_train, y_train)
```

```
GridSearchCV(estimator=LogisticRegression(random_state=0),
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100]})
```

[107]

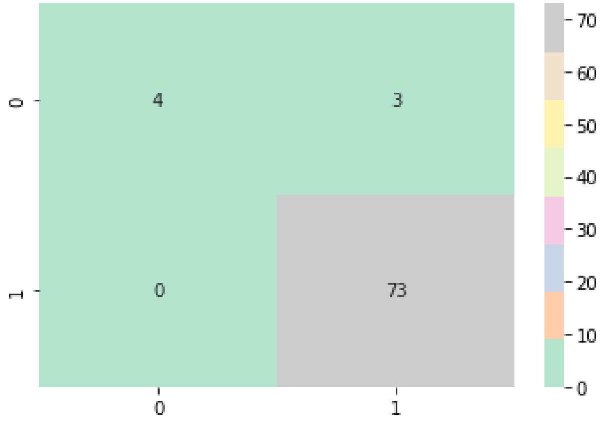
```
print("Best LogisticRegression parameters {}  Best LogisticRegression score {}".format(
    grid_cv.best_params_, grid_cv.best_score_
))
```

```
Best LogisticRegression parameters {'C': 10}  Best LogisticRegression score 0.9375
```

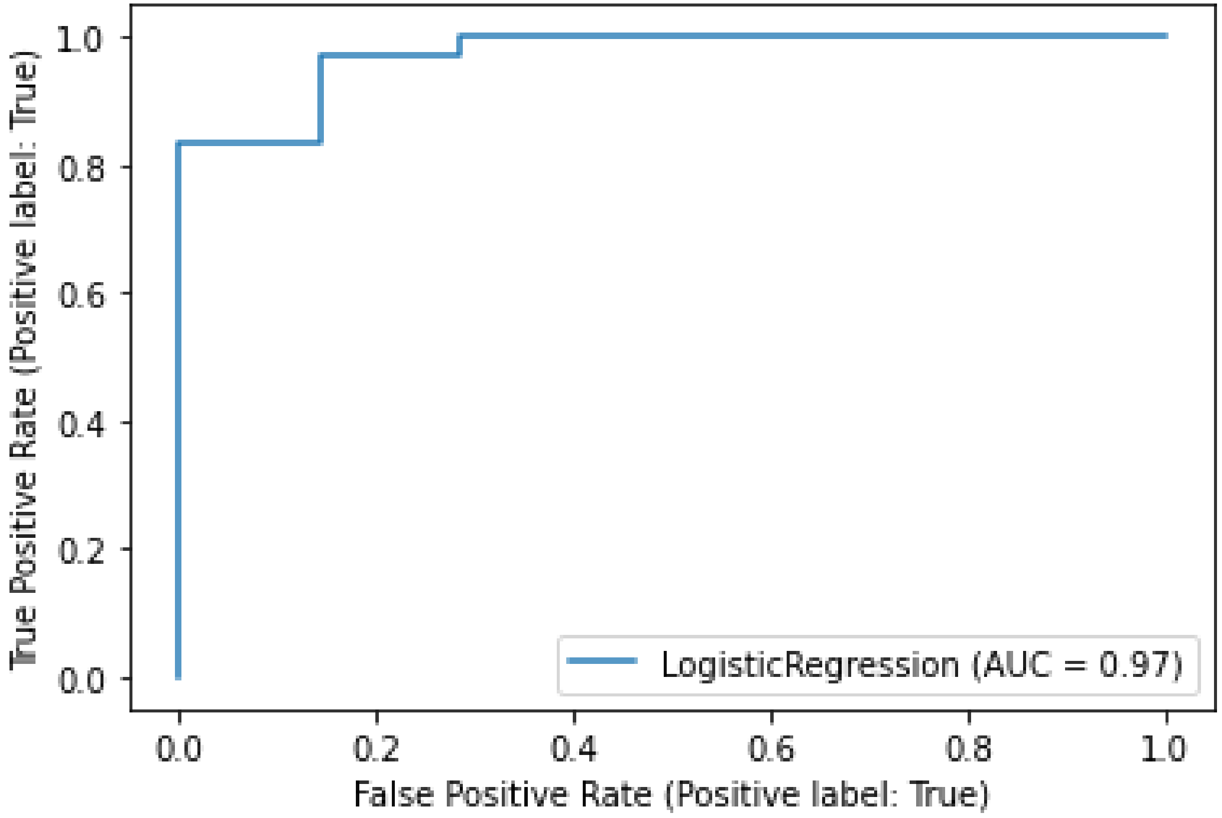
```
[109]
#train model using optimal parameter
logistic = LogisticRegression(C=10).fit(x_train, y_train)
```

```
[110]
#predict the output
y_pred =logistic.predict(x_test)

# model's performance
print("\nAccuracy score: %f" %(accuracy_score(y_test,y_pred) * 100))
print("Recall score : %f" %(recall_score(y_test,y_pred) * 100))
print("ROC score : %f\n" %(roc_auc_score(y_test,y_pred) * 100))

sns.heatmap(confusion_matrix(y_test,y_pred), annot=True, cmap='Pastel2')
```

```
Accuracy score: 96.250000
Recall score : 100.000000
ROC score : 78.571429

<matplotlib.axes._subplots.AxesSubplot at 0x7ff6085b7210>
```



```
#plot roc curve
plot_roc_curve(logistic, x_test, y_test)
```




```
#save and load the LogisticRegression model weights
pickle.dump(logistic,open('LogisticRegression.pkl','wb'))
model=pickle.load(open('LogisticRegression.pkl','rb'))
```

RandomForestClassifier

[113]

```
# Search optimal parameter for RandomForestClassifier
rf = RandomForestClassifier()

grid_values = {
    'n_estimators': [75, 100, 125, 150],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [3, 4, 5, 6, 7],
    'criterion' :['gini', 'entropy']
}

grid_cv = GridSearchCV(rf, param_grid = grid_values)
grid_cv.fit(x_train, y_train)
```

```
GridSearchCV(estimator=RandomForestClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 4, 5, 6, 7],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_estimators': [75, 100, 125, 150]})
```

[114]

```
print("Best RandomForestClassifier parameters {}  Best RandomForestClassifier score {}".format(
    grid_cv.best_params_, grid_cv.best_score_
))
```

```
Best RandomForestClassifier parameters {'criterion': 'gini', 'max_depth': 3, 'max_features': 'auto', 'n_estimators': 150}  Best RandomForestClassifier score 0.94375
```

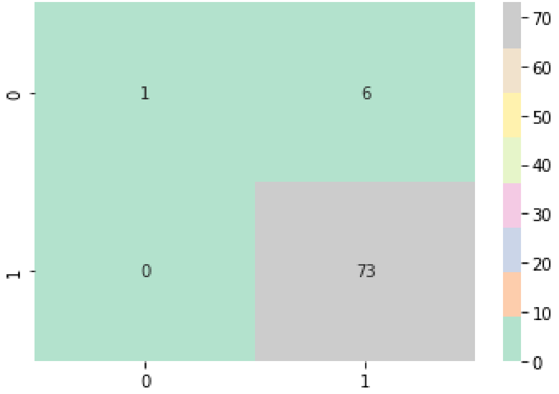
[115]

```
#train model using optimal parameter
rf = RandomForestClassifier(n_estimators=150,
                                  max_features='auto',
                                  max_depth=3,
                                  criterion='gini').fit(x_train, y_train)
```

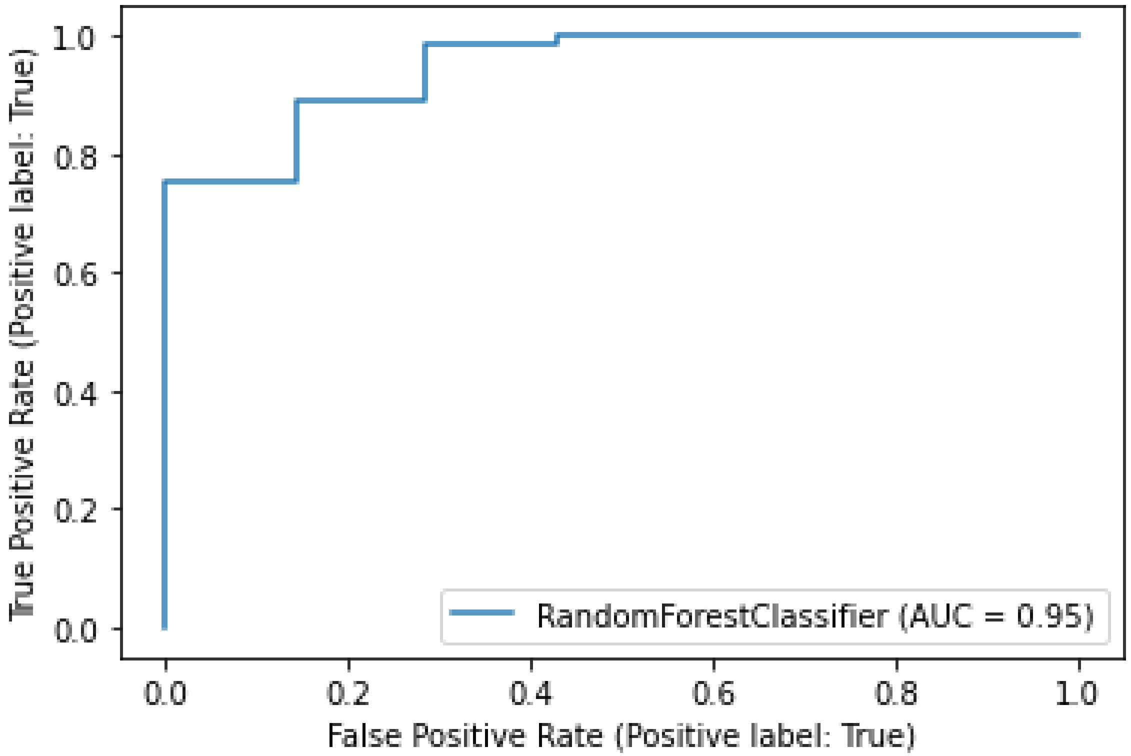
[116]

```
#predict the output
y_pred = rf.predict(x_test)

# model's performance
print("\nAccuracy score: %f" %(accuracy_score(y_test,y_pred) * 100))
print("Recall score : %f" %(recall_score(y_test,y_pred) * 100))
print("ROC score : %f\n" %(roc_auc_score(y_test,y_pred) * 100))

sns.heatmap(confusion_matrix(y_test,y_pred), annot=True, cmap='Pastel2')
```

```
Accuracy score: 92.500000
Recall score : 100.000000
ROC score : 57.142857

<matplotlib.axes._subplots.AxesSubplot at 0x7ff6125cb590>
```



```python
1 #plot roc curve
2 plot_roc_curve(rf, x_test, y_test)
```



[118]

```python
1 #save and load the RandomForestClassifier model weights
2 pickle.dump(logistic,open('RandomForestClassifier.pkl','wb'))
3 model=pickle.load(open('RandomForestClassifier.pkl','rb'))
```

SVM

[119]

```
# Search optimal parameter for SVM
svc = SVC()

grid_values = {'C': [0.01, 0.1, 1, 10, 100],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf', 'poly']}

grid_cv = GridSearchCV(svc, param_grid = grid_values)
grid_cv.fit(x_train, y_train)
```

```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [0.01, 0.1, 1, 10, 100],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf', 'poly']})
```

[120]

```
print("Best SVC parameters {}  Best SVC score {}".format(
    grid_cv.best_params_, grid_cv.best_score_
))
```

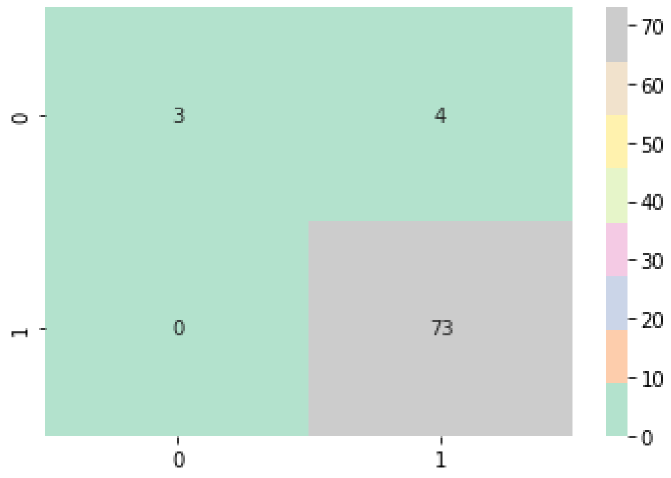
```
Best SVC parameters {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}  Best SVC score 0.94375
```

[121]

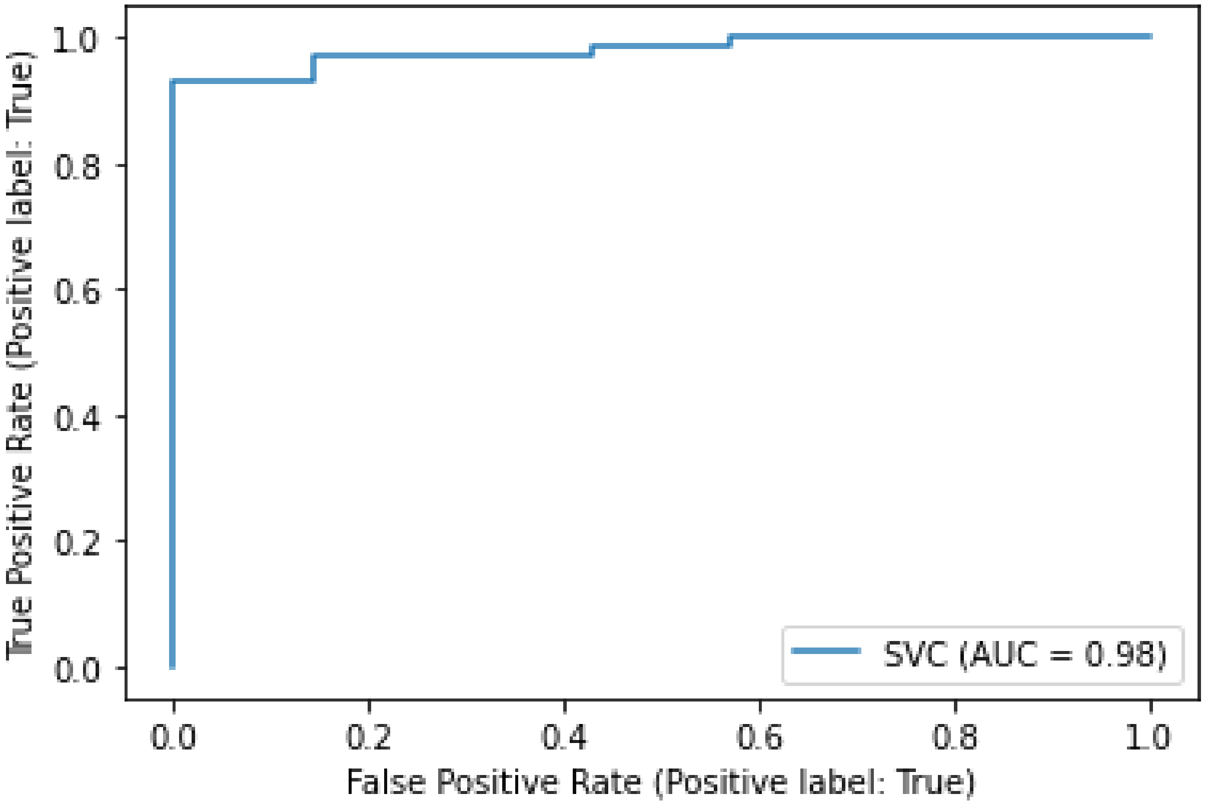
```
#train model using optimal parameter
svc = SVC(C=100, gamma=0.1, kernel='rbf').fit(x_train, y_train)
```

```python
#predict the output
y_pred =svc.predict(x_test)

# model's performance
print("\nAccuracy score: %f" %(accuracy_score(y_test,y_pred) * 100))
print("Recall score : %f" %(recall_score(y_test,y_pred) * 100))
print("ROC score : %f\n" %(roc_auc_score(y_test,y_pred) * 100))

sns.heatmap(confusion_matrix(y_test,y_pred), annot=True, cmap='Pastel2')
```

```
Accuracy score: 95.000000
Recall score : 100.000000
ROC score : 71.428571

<matplotlib.axes._subplots.AxesSubplot at 0x7ff6081764d0>
```



```
#plot roc curve
plot_roc_curve(svc, x_test, y_test)
```

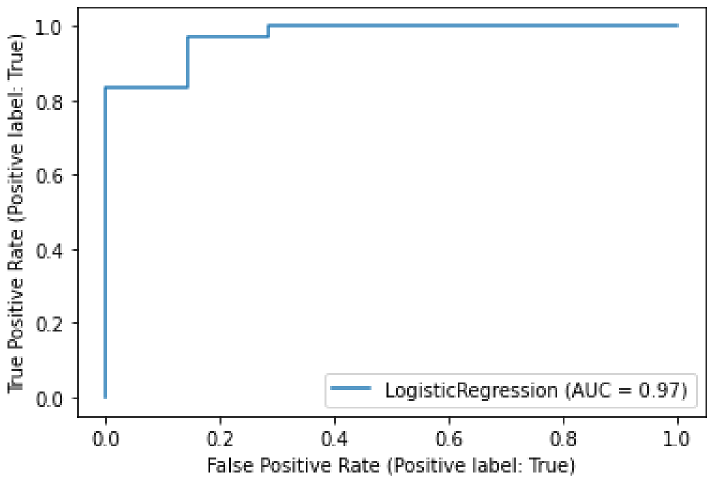


[124]

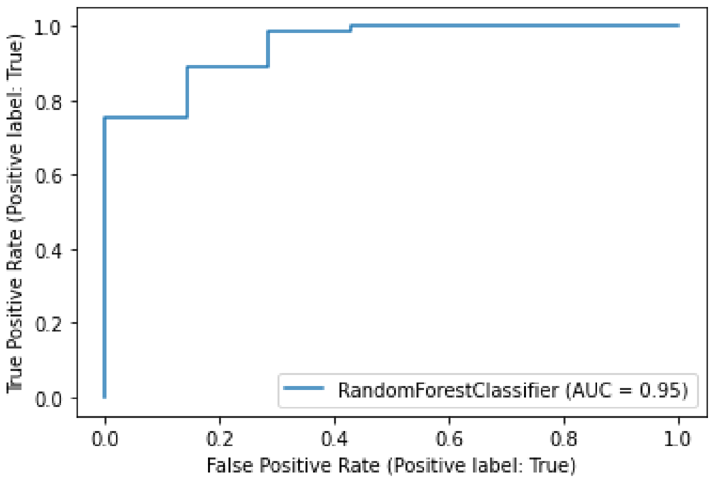
```
#save and load the svc model weights
pickle.dump(logistic,open('svc.pkl','wb'))
model=pickle.load(open('svc.pkl','rb'))
```

Result and discussion

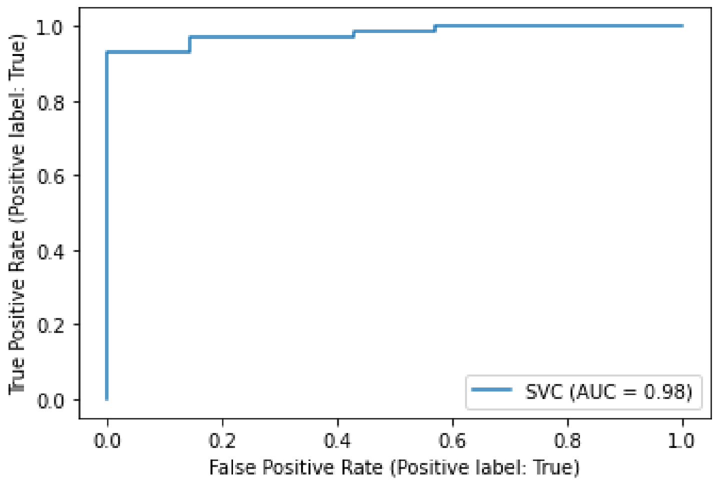
The performance of all the models was measured using various metrics. It is a good choice to use Logistic Regression first, followed by SVM and Random Forest model, since the data was not too skewed in dimensional space, which is easily separated by a linear plane



LogisticRegression



RandomForest



SVM

CONCLUSION

I presented a solution for this problem that is both efficient and effective.

A deep learning approach can help improve this problem a bit, but for that we need more data samples. In addition, to increase its effectiveness, it can be made more complex by adding more attributes, which will result in more accurate predictions.