Step 1: Gather your data

To start with a simple example, let's say that your goal is to build a logistic regression model in Python in order to determine whether candidates would get admitted to a prestigious university.

Here, there are two possible outcomes: Admitted (represented by the value of '1') vs. Rejected (represented by the value of '0').

You can build a logistic regression in Python, where:

The dependent variable represents whether a person gets admitted; and

The 3 independent variables are the GMAT score, GPA and Years of work experience

Step 2: Import the needed Python packages

In [30]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import seaborn as sn
import matplotlib.pyplot as plt
```

Step 3: Build a dataframe

In [2]:

```
candidates = {'gmat': [780,750,690,710,680,730,690,720,740,690,610,690,710,680,770,6
              'gpa': [4,3.9,3.3,3.7,3.9,3.7,2.3,3.3,3.3,1.7,2.7,3.7,3.7,3.3,3.3,3,2.
              'work_experience': [3,4,3,5,4,6,1,4,5,1,3,5,6,4,3,1,4,6,2,3,2,1,4,1,2,
              'admitted': [1,1,0,1,0,1,0,1,1,0,0,1,1,0,1,0,0,1,0,0,1,0,0,0,0,1,1,0,1
              }

df = pd.DataFrame(candidates,columns= ['gmat', 'gpa','work_experience','admitted'])
print (df)
```

```
    gmat  gpa  work_experience  admitted
0    780  4.0                3         1
1    750  3.9                4         1
2    690  3.3                3         0
3    710  3.7                5         1
4    680  3.9                4         0
5    730  3.7                6         1
6    690  2.3                1         0
7    720  3.3                4         1
8    740  3.3                5         1
9    690  1.7                1         0
10   610  2.7                3         0
11   690  3.7                5         1
12   710  3.7                6         1
13   680  3.3                4         0
14   770  3.3                3         1
15   610  3.0                1         0
16   580  2.7                4         0
17   650  3.7                6         1
18   540  2.7                2         0
19   590  2.3                3         0
20   620  3.3                2         1
21   600  2.0                1         0
```

```
22   550  2.3           4         0
23   550  2.7           1         0
24   570  3.0           2         0
25   670  3.3           6         1
26   660  3.7           4         1
27   580  2.3           2         0
28   650  3.7           6         1
29   660  3.3           5         1
30   640  3.0           1         0
31   620  2.7           2         0
32   660  4.0           4         1
33   660  3.3           6         1
34   680  3.3           5         1
35   650  2.3           1         0
36   670  2.7           2         0
37   580  3.3           1         0
38   590  1.7           4         0
39   690  3.7           5         1
```

In [3]:

```
df.describe()
```

Out[3]:

	gmat	gpa	work_experience	admitted
count	40.000000	40.000000	40.000000	40.000000
mean	654.000000	3.095000	3.425000	0.475000
std	61.427464	0.631218	1.737778	0.505736
min	540.000000	1.700000	1.000000	0.000000
25%	607.500000	2.700000	2.000000	0.000000
50%	660.000000	3.300000	4.000000	0.000000
75%	690.000000	3.700000	5.000000	1.000000
max	780.000000	4.000000	6.000000	1.000000

In [5]:

```
print(df.shape)
```

```
(40, 4)
```

Step 4: Create the dependent & independent variables for logistic regression

In [6]:

```
X = df[['gmat', 'gpa','work_experience']]
y = df['admitted']
```

Apply 'train_test_split'. For example, you can set the test size to 0.25, and therefore the model testing will be based on 25% of the dataset, while the model training will be based on 75% of the dataset

In [7]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=0)
```

In [8]:

```
print(X_train.shape)
```

```
(30, 3)
```

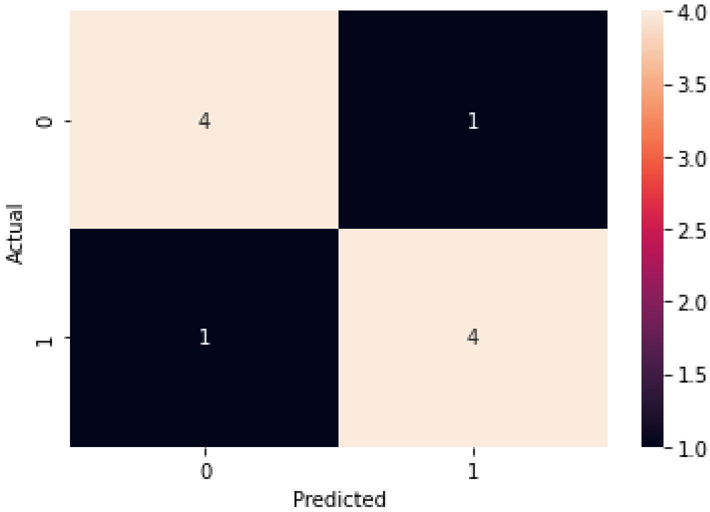
Apply the logistic regression

In [9]:
```
logistic_regression= LogisticRegression()
logistic_regression.fit(X_train,y_train)
y_pred=logistic_regression.predict(X_test)
```

Creating the Confusion Matrix

In [10]:
```
confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predi
sn.heatmap(confusion_matrix, annot=True)
```

Out[10]: <AxesSubplot:xlabel='Predicted', ylabel='Actual'>



print the Accuracy and plot the Confusion Matrix

In [11]:
```
print('Accuracy: ',metrics.accuracy_score(y_test, y_pred))
plt.show()
```

```
Accuracy:  0.8
```

As can be observed from the matrix:

TP = True Positives = 4 TN = True Negatives = 4 FP = False Positives = 1 FN = False Negatives = 1

Accuracy = (TP+TN)/Total = (4+4)/10 = 0.8

Diving Deeper into the Results

In [14]:
```
print (X_test)

##Recall that our original dataset (from step 1) had 40 observations. Since we set t
```

```
    gmat  gpa  work_experience
22   550  2.3                4
20   620  3.3                2
25   670  3.3                6
4    680  3.9                4
10   610  2.7                3
15   610  3.0                1
28   650  3.7                6
```

```
11   690  3.7                5
18   540  2.7                2
29   660  3.3                5
```

In [15]:

```
#The prediction was also made for those 10 records (where 1 = admitted, while 0 = re

print (y_pred)
```

```
[0 0 1 1 0 0 1 1 0 1]
```

In the actual dataset (from step-1), you'll see that for the test data, we got the correct results 8 out of 10 times

Index	gmat	gpa	work_experience	admitted - actual results	admitted - predicted results	Matching
22	550	2.3	4	0	0	TRUE
20	620	3.3	2	1	0	FALSE
25	670	3.3	6	1	1	TRUE
4	680	3.9	4	0	1	FALSE
10	610	2.7	3	0	0	TRUE
15	610	3	1	0	0	TRUE
28	650	3.7	6	1	1	TRUE
11	690	3.7	5	1	1	TRUE
18	540	2.7	2	0	0	TRUE
29	660	3.3	5	1	1	TRUE

Checking the Prediction for a New Set of Data

Let's say that you have a new set of data, with 5 new candidates

Goal is to use the existing logistic regression model to predict whether the new candidates will get admitted

In [16]:

```
##Creating the new candidates dataframe

new_candidates = {'gmat': [590,740,680,610,710],
                  'gpa': [2,3.7,3.3,2.3,3],
                  'work_experience': [3,4,6,1,5]
                  }

df2 = pd.DataFrame(new_candidates,columns= ['gmat', 'gpa','work_experience'])
```

In [17]:

```
df2.describe()
```

Out[17]:

	gmat	gpa	work_experience
count	5.000000	5.00000	5.000000
mean	666.000000	2.86000	3.800000
std	64.265076	0.70214	1.923538
min	590.000000	2.00000	1.000000
25%	610.000000	2.30000	3.000000
50%	680.000000	3.00000	4.000000

	gmat	gpa	work_experience
75%	710.000000	3.30000	5.000000
max	740.000000	3.70000	6.000000

In [18]:

```
y_pred=logistic_regression.predict(df2)
```

In [19]:

```
print (df2)
```

```
   gmat  gpa  work_experience
0   590  2.0                3
1   740  3.7                4
2   680  3.3                6
3   610  2.3                1
4   710  3.0                5
```

In [20]:

```
print (y_pred)
```

```
[0 1 1 0 1]
```

The first and fourth candidates are not expected to be admitted, while the other candidates are expected to be admitted

Calculating ROC Curve

In [23]:

```
import sklearn.metrics as metrics

# calculate the fpr and tpr for all thresholds of the classification
probs = logistic_regression.predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)
```

Difference between "The predict() method" and "The predict_proba() method"

The predict() method

All supervised estimators in scikit-learn implement the predict() method that can be executed on a trained model in order to predict the actual label (or class) over a new set of data.

The method accepts a single argument that corresponds to the data over which the predictions will be made and it returns an array containing the predicted label for each data point.

In [25]:

```
predictions = logistic_regression.predict(X_test)
predictions
```

Out[25]:

```
array([0, 0, 1, 1, 0, 0, 1, 1, 0, 1], dtype=int64)
```

The predict_proba() method

In the context of classification tasks, some sklearn estimators also implement the predict_proba method that returns the class probabilities for each data point.

The method accepts a single argument that corresponds to the data over which the probabilities will be computed and returns an array of lists containing the class probabilities for the input data points.

In [27]:

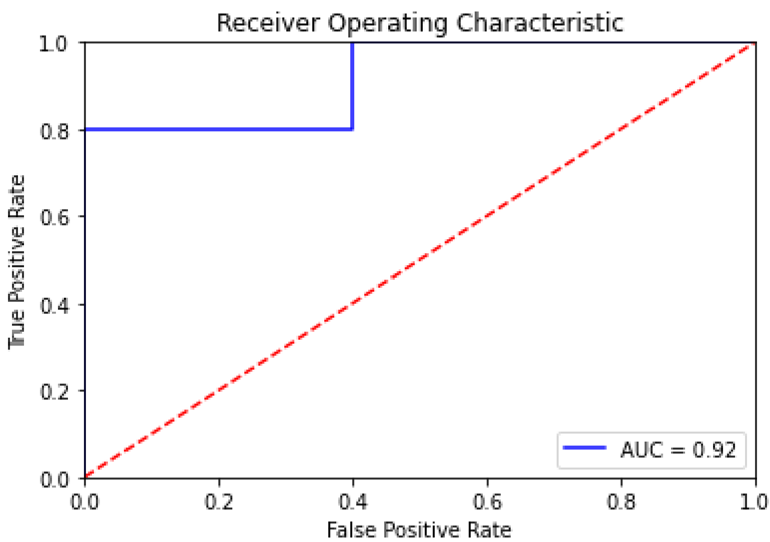
```
predictions = logistic_regression.predict_proba(X_test)
print(predictions)
```

```
[[9.91609541e-01 8.39045850e-03]
 [9.82686248e-01 1.73137518e-02]
 [4.82945926e-02 9.51705407e-01]
 [2.31228082e-01 7.68771918e-01]
 [9.72097119e-01 2.79028812e-02]
 [9.97314934e-01 2.68506615e-03]
 [7.39280072e-02 9.26071993e-01]
 [6.18376418e-02 9.38162358e-01]
 [9.99411602e-01 5.88398313e-04]
 [2.11035600e-01 7.88964400e-01]]
```

In [28]:

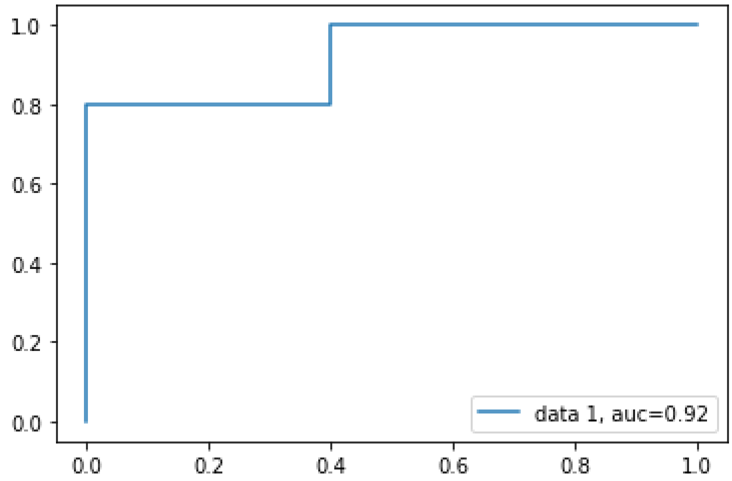
```
# method using: plt

import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [29]:

```
y_pred_proba = logistic_regression.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
data 1, auc=0.92