Code and Outputs

The following code was implemented to divide the Symptom attribute into individual symptom attributes:

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as snb
```

```
In [2]: symp = pd.read_csv("D:\project\BVBRC_surveillance.csv")
```

```
In [3]: symp = symp.drop(['Project Identifier','Contributing Institution','Sample Transport Medium','Sample Receipt Date','Submission Dat
        ,'Collection Longitude','Pathogen Test Interpretation','Species','Type','Maintenance Medication','Types of Allergies','Influenza
```

```
In [4]: col = symp.columns.tolist()
```

```
In [6]: s = symp['Symptoms'].str.split(';', expand=True).stack()

        # Create a new dataframe from the stacked series
        temp_df = pd.DataFrame(s.values, index=s.index.droplevel(-1), columns=['key_value_pair'])

        # Split the new dataframe into two columns, key and value
        temp_df[['key', 'value']] = temp_df['key_value_pair'].str.split(':', expand=True)

        # Pivot the new dataframe to create separate columns for each key
        pivot_df = temp_df.pivot(columns='key', values='value')

        # Join the pivoted dataframe back to the original dataframe
        result_df = symp.join(pivot_df)
```

```
In [9]: result_df
```

Email Address	Collection Year	Collection Season	Collection Country	Collection State Province	Collection City	...	other symptoms	rash	running nose	short breath	sinus congestion	sudden onset	temperature	throat	vomiting	wheezing
ry@yahoo.com	2009	NaN	China	Shantou	NaN	...	NaN	NaN	Y	U	NaN	NaN	NaN	Y	NaN	NaN
ry@yahoo.com	2008	NaN	China	Shantou	NaN	...	NaN	NaN	N	U	NaN	NaN	NaN	Y	NaN	NaN
NaN	2013	NaN	USA	Massachusetts	Boston, MA	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
@niaidceirs.org	NaN	2019-2020	United States Of America	Maryland	NaN	...	Restricted Access	NaN	Y	Y	NaN	NaN	NaN	Y	Y	Y
ry@yahoo.com	2009	NaN	China	Shantou	NaN	...	NaN	NaN	N	U	NaN	NaN	NaN	N	NaN	NaN

Code to divide the Symptom attribute to each individual symptom attribute (Handling the values 'Yes' and 'No').

```
In [17]: symp = ['Fever', 'aches', 'arthalgia', 'chest pain', 'chills',
                 'conjunctivitis', 'cough', 'diarrhea', 'dyspnea', 'ear ache', 'fatigue',
                 'fever', 'headache', 'loss of appetite', 'malaise',
                 'myalgia', 'nausea', 'other symptoms', 'rash', 'running nose',
                 'short breath', 'sinus congestion', 'sudden onset',
                 'throat', 'vomiting', 'wheezing']

In [18]: for i in symp:
             pro[i] = pro[i].map({'Yes': 1, 'No': 0 , 'Y' : 1, 'N': 0 , 'Not Provided' : 'na' , 'Not Collected' : 'na', 'none' : 0 , 'Rest

In [20]: pro.fillna(0)
```

Out[20]:

	Unnamed: 0	Sample Identifier	Sequence Accession	Sample Material	Collector Institution	Contact Email Address	Collection Year	Collectio Seaso
0	0	491705	0	NS	Hong Kong University	fluquery@yahoo.com	2009	
1	1	490248	0	NS	Hong Kong University	fluquery@yahoo.com	2008	
2	2	NIGSP_YGA_00087	CY168429,CY168430,CY168427,CY168428,CY168423,C...	NS	Harvard Medical School	0	2013	
3	3	02-11-R-0661	0	NAS	Johns Hopkins University-CEIRS2	irdsupport@niaidceirs.org	0	2019-202
4	4	491450	0	NS	Hong Kong University	fluquery@yahoo.com	2009	
...	...	...	...	...	...	...	...	...

Code to map Positive and Negative value of Target variable with 1 and 0 respectively. Handling Age variable disrupted values.

```
In [27]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         import seaborn as snb

In [28]: df = pd.read_csv('influenza5.csv')

In [29]: df['Pathogen Test Result'] = df['Pathogen Test Result'].map({'Negative':0 ,'Positive':1})

In [30]: df['Pathogen Test Result'] = df['Pathogen Test Result'].fillna(0)

In [31]: df['Pathogen Test Result'] = df['Pathogen Test Result'].astype(int)

In [32]: age = df['Host Age']

In [33]: age = age.replace('Not Provided' , '0').replace('Not Collected' , '0').replace('>90' , '95')

In [34]: age = age

In [35]: age = age.astype(float)

In [36]: age = age.astype(int)

In [37]: age

Out[37]: 0         0
         1         0
         2        42
         3        26
         4         0
                  ..
         30525    58
         30526     0
         30527    74
         30528     4
         30529    42
         Name: Host Age, Length: 30530, dtype: int32

In [38]: df['Host Age'] = age
```

Code to map Male and Female value of Host Sex with 1 and 0 respectively.

```
In [18]: df.index = np.arange(len(df))

In [19]:  df = df.drop(['Unnamed: 0'],axis =1)

In [20]: df['Host Sex'] = df['Host Sex'].map({'Female' : 0 , 'Male' : 1})

In [21]: y=[]
         x=df['Host Sex'].isnull()
         for i in range(len(x)):
             if  x[i]:
                 y.append(i)
         df = df.drop(y)
```

Pre-processing Chronic Condition variable of Data set.

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib as plt
        import seaborn as snb

In [2]: pro = pd.read_csv("influenza2.csv")

        C:\Users\raman\AppData\Local\Temp\ipykernel_14176\4081774908.py:1: DtypeWarning: Columns (26) have mixed types. Specify dtype o
        ption on import or set low_memory=False.
          pro = pd.read_csv("influenza2.csv")

In [3]: pro_chro_condition = pro['Chronic Conditions'].astype(str)

In [4]: chro_list = list(pro_chro_condition)

In [5]: x=[]
        for i in chro_list:
            i=i.replace("'","")
            i=i.replace('"','')
            x.append(i)

In [6]: inf_series = pd.Series(x)
```

In [7]:
```
inf_list = []
for i in range(len(x)):
    if 'nan' in x[i]:
        inf_list.append(0)
    elif 'OTH-none' in x[i]:
        inf_list.append(0)
    elif 'OTH- none' in x[i]:
        inf_list.append(0)
    elif 'Not Provided' in x[i]:
        inf_list.append(0)
    elif 'Not Collected' in x[i]:
        inf_list.append(0)
    elif 'HPT' == x[i]:
        inf_list.append(1)
    elif 'RSP' == x[i]:
        inf_list.append(2)
    elif 'chronic lung disease' == x[i]:
        inf_list.append(3)
    elif 'END' == x[i]:
        inf_list.append(4)
    elif 'CDV' == x[i]:
        inf_list.append(5)
    elif 'ASM' == x[i]:
        inf_list.append(6)
    elif 'HEM' == x[i]:
        inf_list.append(7)
    elif 'NRL' == x[i]:
        inf_list.append(8)
    elif 'OBS' == x[i]:
        inf_list.append(9)
    elif 'IMS' == x[i]:
        inf_list.append(10)
    elif 'CAN' == x[i]:
        inf_list.append(11)
    elif 'URO' == x[i]:
        inf_list.append(12)
    elif 'FLU' == x[i]:
        inf_list.append(13)
    elif 'immuno' == x[i]:
        inf_list.append(14)
    elif 'diabetes' == x[i]:
        inf_list.append(15)
    elif 'obesity' == x[i]:
        inf_list.append(16)
    elif 'END;CAN' in x[i] :
        inf_list.append(17)
    elif 'HEM;CAN' in x[i]:
        inf_list.append(18)
```

```
    elif 'HPT;END' in x[i]:
        inf_list.append(19)
    elif 'CDV;HPT' in x[i]:
        inf_list.append(20)
    elif 'END;NRL' in x[i]:
        inf_list.append(21)
    elif 'CDV;NRL' in x[i]:
        inf_list.append(22)
    elif 'URO;END' in x[i]:
        inf_list.append(23)
    elif 'CDV;URO' in x[i]:
        inf_list.append(24)
    elif 'RSP;ASM' in x[i]:
        inf_list.append(25)
    elif 'OBS;OTH-chronic lung disease;ASM' in x[i]:
        inf_list.append(26)
    elif 'CDV;END'in x[i]:
        inf_list.append(27)
    elif 'CDV;END;NRL'in x[i]:
        inf_list.append(28)
    elif 'OTH-chronic lung disease;ASM' == x[i]:
        inf_list.append(29)
    else:
        inf_list.append(30)
```

Code for removing unnecessary variable.

In [1]:
```
import pandas as pd
```

In [2]:
```
df = pd.read_csv('influenza3.csv')
```

```
C:\Users\raman\AppData\Local\Temp\ipykernel_1980\1258131324.py:1: DtypeWarning: Columns (27) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv('influenza3.csv')
```

In [3]:
```
df.columns
```

Out[3]:
```
Index(['Unnamed: 0.1', 'Unnamed: 0', 'Sample Identifier', 'Sequence Accession',
       'Sample Material', 'Collector Institution', 'Contact Email Address',
       'Collection Year', 'Collection Season', 'Collection Country',
       'Collection State Province', 'Collection City', 'Pathogen Test Type',
       'Pathogen Test Result', 'Subtype', 'Strain', 'Host Identifier',
       'Host ID Type', 'Host Species', 'Host Common Name', 'Host Group',
       'Host Sex', 'Host Age', 'Chronic Conditions', 'Symptoms',
       'Additional Metadata', 'ALTCON', 'Acute Respiratory Distress Syndrome',
       'Fever', 'Temperature', 'aches', 'arthalgia', 'chest pain', 'chills',
       'conjunctivitis', 'cough', 'diarrhea', 'dyspnea', 'ear ache', 'fatigue',
       'fever', 'headache', 'high_temp_home', 'loss of appetite', 'malaise',
       'myalgia', 'nausea', 'other symptoms', 'rash', 'running nose',
       'short breath', 'sinus congestion', 'sudden onset', 'temperature',
       'throat', 'vomiting', 'wheezing'],
      dtype='object')
```

In [4]:
```
df = df.drop(['Unnamed: 0.1', 'Unnamed: 0', 'Sample Identifier', 'Sequence Accession',
       'Sample Material', 'Collector Institution', 'Contact Email Address','Collection Country',
       'Collection State Province', 'Collection City', 'Pathogen Test Type','Subtype', 'Strain', 'Host Identifier',
       'Host ID Type', 'Host Species', 'Host Common Name', 'Host Group','Symptoms',
       'Additional Metadata', 'ALTCON','temperature','high_temp_home','Temperature'],axis = 1)
```

In [5]:
```
df
```

Out[5]:

	Collection Year	Collection Season	Pathogen Test Result	Host Sex	Host Age	Chronic Conditions	Acute Respiratory Distress Syndrome	Fever	aches	arthalgia	...	nausea	other symptoms	rash	running nose	short breath	si congest
0	2009	NaN	Negative	Male	0	0	NaN	1	NaN	NaN	...	1	NaN	NaN	1	1	N
1	2008	NaN	Negative	Female	0	0	NaN	1	NaN	NaN	...	1	NaN	NaN	0	1	N
2	2013	NaN	Positive	Female	42	0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	N
3	NaN	2019-2020	Negative	Female	26	26	NaN	1	NaN	NaN	...	1	0	NaN	1	1	N
4	2009	NaN	Negative	Female	0	0	NaN	1	NaN	NaN	...	1	NaN	NaN	0	1	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
30525	NaN	2016-2017	Negative	Female	58	8	NaN	na	NaN	NaN	...	0	0	NaN	na	0	N

Code to merge Collection Year variable and Collection Season variable.

```
In [6]: df_year = df['Collection Year'].astype(str)

In [7]: df_season = df['Collection Season'].astype(str)

In [8]: df_year

Out[8]: 0        2009
        1        2008
        2        2013
        3         nan
        4        2009
                 ...
        30525     nan
        30526    2008
        30527     nan
        30528     nan
        30529    2014
        Name: Collection Year, Length: 30530, dtype: object

In [9]: df_season

Out[9]: 0              nan
        1              nan
        2              nan
        3        2019-2020
        4              nan
                   ...
        30525    2016-2017
        30526          nan
        30527    2018-2019
        30528         2020
        30529          nan
        Name: Collection Season, Length: 30530, dtype: object

In [10]: df_season[3][:4]

Out[10]: '2019'

In [11]: for i in range(len(df_year)):
             if df_year[i] == 'nan':
                 df_year[i] = df_season[i][:4]

In [12]: for i in range(len(df_year)):
             df_year[i] = df_year[i][:4]

In [13]: df_year.tolist()

Out[13]: ['2009',
          '2008',
          '2013',
          '2019',
          '2009',
          '2014',
          '2012',
          '2015',
          '2020',
          '2012',
          '2009',
          '2017',
          '2020',
          '2018',
          '2016',
          '2012',
          '2013',
          '2018',
          '2008',

In [14]: df['Collection Year'] = df_year

In [15]: df = df.drop(['Collection Season'],axis = 1)

In [16]: df['Collection Year']

Out[16]: 0        2009
         1        2008
         2        2013
         3        2019
         4        2009
                  ...
         30525    2016
         30526    2008
         30527    2018
         30528    2020
         30529    2014
         Name: Collection Year, Length: 30530, dtype: object
```

Code for removing null value rows, symptoms having 95% of null values and filling null values with 0 in remaing values.

```
In [17]: dict(df.isnull().sum(axis = 1))
```

```
Out[17]: {0: 16,
          1: 16,
          2: 27,
          3: 10,
          4: 16,
          5: 22,
          6: 27,
          7: 10,
          8: 10,
          9: 13,
          10: 16,
          11: 10,
          12: 10,
          13: 10,
          14: 11,
          15: 13,
          16: 27,
          17: 10,
          18: 16,
```

```
In [18]: dict(df.isnull().sum(axis = 0))
```

```
Out[18]: {'Collection Year': 0,
          'Pathogen Test Result': 0,
          'Host Sex': 102,
          'Host Age': 16,
          'Chronic Conditions': 0,
          'Acute Respiratory Distress Syndrome': 30529,
          'Fever': 2302,
          'aches': 29976,
          'arthalgia': 30530,
          'chest pain': 30006,
          'chills': 12528,
          'conjunctivitis': 13991,
          'cough': 2266,
          'diarrhea': 2408,
          'dyspnea': 30002,
          'ear ache': 30020,
          'fatigue': 29914,
          'fever': 2302,
          'headache': 2340,
          'loss of appetite': 14494,
          'malaise': 2954,
          'myalgia': 2899,
          'nausea': 2435,
          'other symptoms': 14206,
          'rash': 30094,
          'running nose': 2324,
          'short breath': 2944,
          'sinus congestion': 29945,
          'sudden onset': 30530,
          'throat': 2337,
          'vomiting': 12565,
          'wheezing': 13108}
```

```
In [19]: df  = df.drop(['Acute Respiratory Distress Syndrome','arthalgia','sudden onset','chest pain','ear ache','rash','dyspnea'],axis=1)
```

```
In [20]: df = df.drop(['sinus congestion','fatigue','aches'],axis=1)
```

```
In [21]: df = df.fillna(0)
         df = df.replace('na',0)
```

In [22]:
```
df
```

Out[22]:

	Collection Year	Pathogen Test Result	Host Sex	Host Age	Chronic Conditions	Fever	chills	conjunctivitis	cough	diarrhea	...	loss of appetite	malaise	myalgia	nausea	other symptoms	runnir nos
0	2009	Negative	Male	0	0	1	0	0	1	1	...	0	1	1	1	0	
1	2008	Negative	Female	0	0	1	0	0	0	1	...	0	1	1	1	0	
2	2013	Positive	Female	42	0	0	0	0	0	0	...	0	0	0	0	0	
3	2019	Negative	Female	26	26	1	1	0	1	0	...	1	1	1	1	0	
4	2009	Negative	Female	0	0	1	0	0	1	1	...	0	1	1	1	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
30525	2016	Negative	Female	58	8	0	1	1	0	0	...	0	0	0	0	0	
30526	2008	Negative	Male	0	0	1	0	0	1	1	...	0	1	1	1	0	
30527	2018	Positive	Female	74	26	0	1	0	1	0	...	1	1	1	0	0	
30528	2020	Negative	Male	4	0	1	0	0	1	0	...	0	0	0	0	0	
30529	2014	Positive	Male	42	0	0	1	0	1	0	...	0	0	0	0	0	

30530 rows × 22 columns

Code for splitting data set for training and testing.

In [2]:
```
# Load the dataset
df = pd.read_csv('inf.csv')
df = df.drop(['Unnamed: 0'],axis = 1)
```

In [3]:
```
df['Pathogen Test Result'].value_counts()
```

Out[3]:
```
1    10028
0     8489
Name: Pathogen Test Result, dtype: int64
```

In [4]:
```
df.describe()
```

Out[4]:

	Collection Year	Pathogen Test Result	Host Sex	Host Age	Chronic Conditions	Fever	chills	conjunctivitis	cough	diarrhea	...	
count	18517.000000	18517.000000	18517.000000	18517.000000	18517.000000	18517.000000	18517.000000	18517.000000	18517.000000	18517.000000	...	1851
mean	2015.881838	0.541556	0.471405	34.079657	7.982341	0.442944	0.400443	0.230545	0.448615	0.122752	...	
std	2.855294	0.498284	0.499195	22.497576	11.803348	0.496747	0.490001	0.421193	0.497366	0.328161	...	
min	2006.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
25%	2015.000000	0.000000	0.000000	16.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
50%	2016.000000	1.000000	0.000000	32.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
75%	2018.000000	1.000000	1.000000	51.000000	18.000000	1.000000	1.000000	0.000000	1.000000	0.000000	...	
max	2020.000000	1.000000	1.000000	97.000000	30.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	

8 rows × 22 columns

In [5]:
```
# Split the data into features and target variable
X = df.drop(['Pathogen Test Result'],axis = 1)
Y = df['Pathogen Test Result']
```

In [8]:
```
# Split the data into training and testing sets
X_train , X_test , Y_train , Y_test = train_test_split(X,Y,test_size= 0.3, random_state=2)
```

Code for implementing SVM Model with Linear Kernel.

In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score , recall_score , f1_score
from sklearn.metrics import roc_curve , auc
import matplotlib.pyplot as plt
```

In [7]:
```
st_x= StandardScaler()
X_train= st_x.fit_transform(X_train)
X_test= st_x.transform(X_test)
```

In [8]:
```
# Train the model on the training set
classifier = SVC(kernel='linear', random_state=2)
classifier.fit(X_train, Y_train)
```

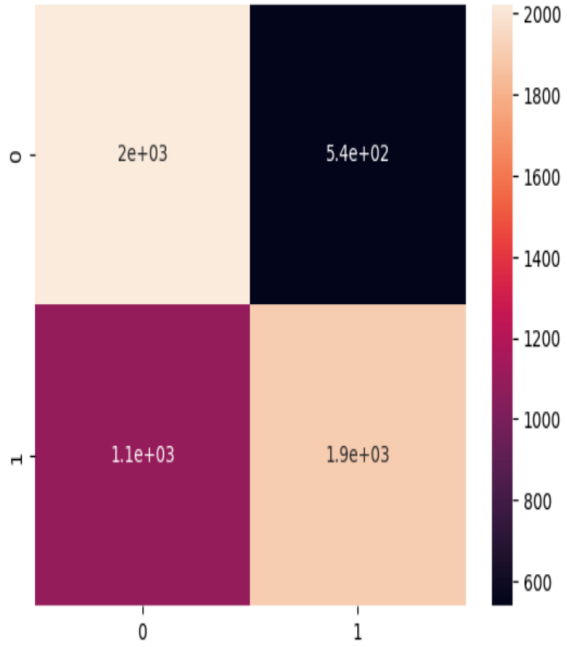
Out[8]:
```
SVC(kernel='linear', random_state=2)
```

In [9]:
```
# Use the trained model to make predictions on the testing set
y_pred = classifier.predict(X_test)
```

In [10]:
```
cm= confusion_matrix(Y_test, y_pred)
```

In [11]:
```
snb.heatmap(cm,annot = True)
cm
```

Out[11]:
```
array([[2022,  539],
       [1091, 1904]], dtype=int64)
```



In [12]:
```
score =accuracy_score(Y_test,y_pred)
print('accuracy on testing Data :',round(score*100 ,2),'%')
```

```
accuracy on testing Data : 70.66 %
```

In [13]:
```
y_pred2 =classifier.predict(X_train)
```

In [14]:
```
score2 =accuracy_score(y_pred2, Y_train)
print('accuracy on Training Data :',round(score2*100 ,2),'%')
```

```
accuracy on Training Data : 71.1 %
```

In [15]:
```
# precision for trainig and testing
print('for testing  : ' , round(precision_score(Y_test , y_pred)*100,2),'%')
print('for training : ' , round(precision_score(Y_train , y_pred2)*100,2),'%')
```

```
for testing  :  77.94 %
for training :  78.67 %
```

In [16]:
```
# Recall for trainig and testing
print('for testing  : ' , round(recall_score(Y_test , y_pred)*100,2),'%')
print('for training : ' , round(recall_score(Y_train , y_pred2)*100,2),'%')
```

```
for testing  :  63.57 %
for training :  64.13 %
```

In [17]:
```
# f1_score for trainig and testing
print('for testing  : ' , round(f1_score(Y_test , y_pred)*100,2),'%')
print('for training : ' , round(f1_score(Y_train , y_pred2)*100,2),'%')
```

```
for testing  :  70.03 %
for training :  70.66 %
```

In [18]:
```
# for sensitivity and specificity
tn, fp, fn, tp = cm.ravel()
```

In [19]:
```
specificity = tn / (tn+fp)
print('specificity for data is : ',round(specificity*100,2),'%')
```

```
specificity for data is :  78.95 %
```

In [20]:
```
sensitivity = tp / (tp+fn)
print('sensitivity for data is : ',round(sensitivity*100,2),'%')
```

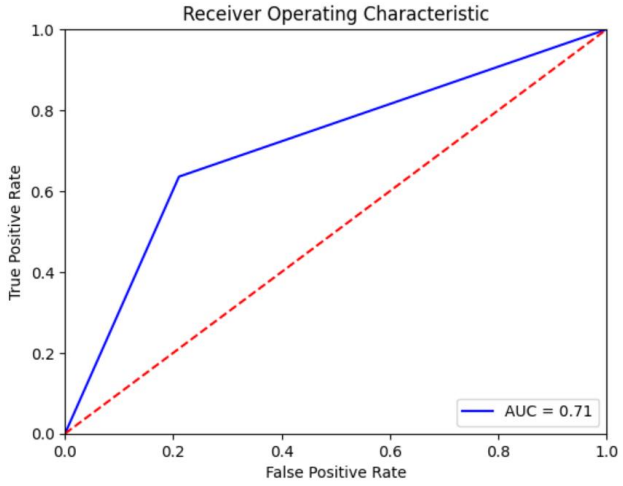
```
sensitivity for data is :  63.57 %
```

In [21]:
```
#ROC and AUC
fpr, tpr, threshold = roc_curve(Y_test,y_pred)
roc_auc = auc(fpr, tpr)
```

In [22]:
```
print('Area under curve : ',round(roc_auc*100,2),'%')
```

```
Area under curve :  71.26 %
```

In [23]:
```
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()
```



Code for implementing RandomForest Model.

In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score , recall_score , f1_score , confusion_matrix
from sklearn.metrics import roc_curve , auc
import matplotlib.pyplot as plt
import seaborn as snb
```

In [5]:
```
# Create a random forest classifier object with 100 trees
rf = RandomForestClassifier(n_estimators=100)
```

In [6]:
```
# Train the model on the training set
rf.fit(X_train, y_train)
```

Out[6]:
```
RandomForestClassifier
RandomForestClassifier()
```

In [7]:
```
# Use the trained model to make predictions on the testing set
y_pred = rf.predict(X_test)
y_pred2 = rf.predict(X_train)
```

In [8]:
```
# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", round(accuracy*100,2),'%')
```

```
Accuracy: 87.34 %
```

In [9]:
```
# precision for trainig and testing
print('for testing  : ' , round(precision_score(y_test , y_pred)*100,2),'%')
print('for training : ' , round(precision_score(y_train , y_pred2)*100,2),'%')
```

```
for testing  :  90.63 %
for training :  96.29 %
```

In [10]:
```
# Recall for trainig and testing
print('for testing  : ' , round(recall_score(y_test , y_pred)*100,2),'%')
print('for training : ' , round(recall_score(y_train , y_pred2)*100,2),'%')
```

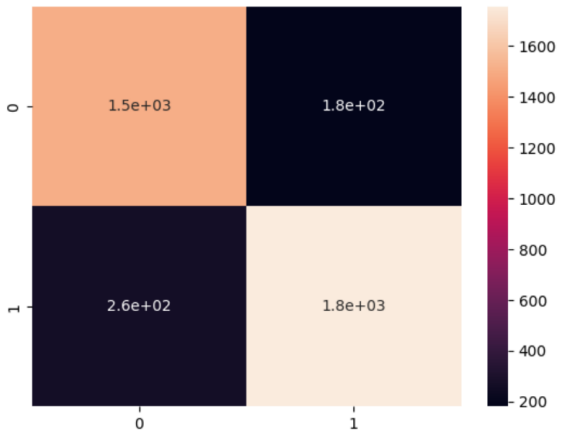
```
for testing  :  85.66 %
for training :  89.86 %
```

In [11]:
```
# f1_score for trainig and testing
print('for testing  : ' , round(f1_score(y_test , y_pred)*100,2),'%')
print('for training : ' , round(f1_score(y_train , y_pred2)*100,2),'%')
```

```
for testing  :  88.08 %
for training :  92.96 %
```

In [12]:
```
cm = confusion_matrix(y_test,y_pred)
snb.heatmap(cm, annot =True)
cm
```

Out[12]:
```
array([[1504,  180],
       [ 265, 1755]], dtype=int64)
```



In [13]:
```
# for sensitivity and specificity
tn, fp, fn, tp = cm.ravel()
```

In [14]:
```
specificity = tn / (tn+fp)
print('specificity for data is : ',round(specificity*100,2),'%')
```

```
specificity for data is :  89.31 %
```

In [15]:
```
sensitivity = tp / (tp+fn)
print('sensitivity for data is : ',round(sensitivity*100,2),'%')
```

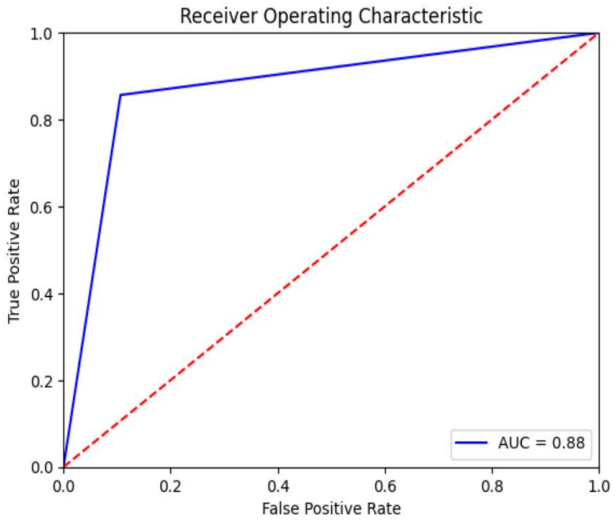
```
sensitivity for data is :  86.88 %
```

In [16]:
```
#ROC and AUC
fpr, tpr, threshold = roc_curve(y_test,y_pred)
roc_auc = auc(fpr, tpr)
```

In [17]:
```
print('Area under curve : ',round(roc_auc*100,2),'%')
```

```
Area under curve :  87.51 %
```

In [18]:
```
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()
```



Code for implementing K-NN Model.

```
In [1]: import pandas as pd
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import confusion_matrix , classification_report
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import precision_score , recall_score , f1_score
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import roc_curve , auc
        import matplotlib.pyplot as plt
        import seaborn as snb
```

```
In [5]: classifier = KNeighborsClassifier(n_neighbors= 5)
        classifier.fit(X_train,Y_train)
```

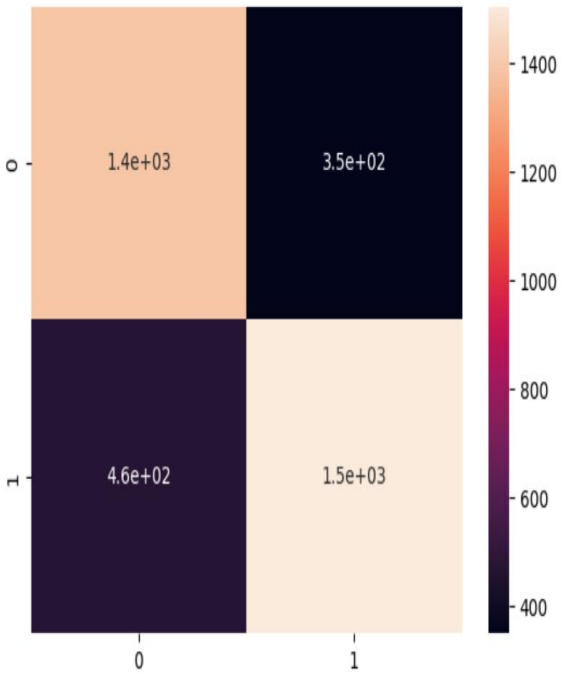
```
Out[5]: KNeighborsClassifier
        KNeighborsClassifier()
```

```
In [6]: Y_pred = classifier.predict(X_test)
```

```
In [7]: cm= confusion_matrix(Y_test, Y_pred)
```

```
In [8]: snb.heatmap(cm,annot=True)
        cm
```

```
Out[8]: array([[1387,  349],
               [ 464, 1504]], dtype=int64)
```



In [9]:
```
print(classification_report(Y_test,Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.75      0.80      0.77      1736
           1       0.81      0.76      0.79      1968

    accuracy                           0.78      3704
   macro avg       0.78      0.78      0.78      3704
weighted avg       0.78      0.78      0.78      3704
```

In [10]:
```
# for sensitivity and specificity
tn, fp, fn, tp = cm.ravel()
```

In [11]:
```
specificity = tn / (tn+fp)
print('specificity for data is : ',round(specificity*100,2),'%')
```

```
specificity for data is :  79.9 %
```

In [12]:
```
sensitivity = tp / (tp+fn)
print('sensitivity for data is : ',round(sensitivity*100,2),'%')
```

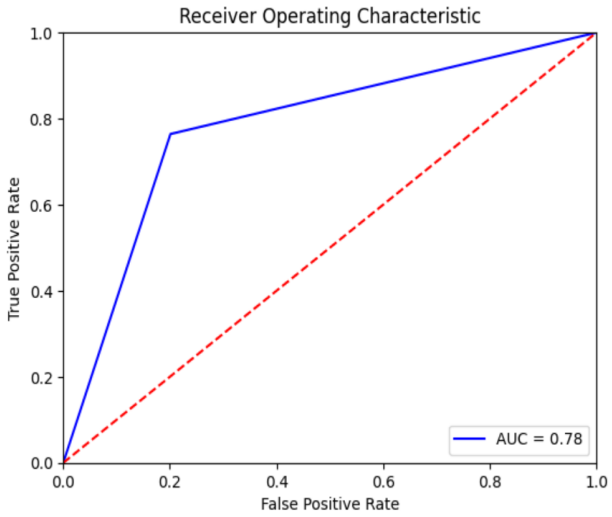
```
sensitivity for data is :  76.42 %
```

In [13]:
```
#ROC and AUC
fpr, tpr, threshold = roc_curve(Y_test,Y_pred)
roc_auc = auc(fpr, tpr)
```

In [14]:
```
print('Area under curve : ',round(roc_auc*100,2),'%')
```

```
Area under curve :  78.16 %
```

In [15]:
```
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()
```



Code for implementing ANN Model.

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score , recall_score , f1_score , confusion_matrix
from sklearn.metrics import roc_curve , auc
import matplotlib.pyplot as plt
import seaborn as snb
```

In [5]:
```
# Create a neural network model
model = Sequential()
model.add(Dense(16, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

In [6]:
```
# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [7]:
```
# Train the model on the training set
model.fit(X_train, y_train, epochs=100, batch_size=32)
```

```
Epoch 3/100
463/463 [==============================] - 1s 2ms/step - loss: 0.7170 - accuracy: 0.5289
Epoch 4/100
463/463 [==============================] - 1s 2ms/step - loss: 0.7097 - accuracy: 0.5351
Epoch 5/100
463/463 [==============================] - 1s 2ms/step - loss: 0.7118 - accuracy: 0.5353
Epoch 6/100
463/463 [==============================] - 1s 2ms/step - loss: 0.7262 - accuracy: 0.5360
Epoch 7/100
463/463 [==============================] - 1s 2ms/step - loss: 0.7176 - accuracy: 0.5412
Epoch 8/100
463/463 [==============================] - 1s 2ms/step - loss: 0.7122 - accuracy: 0.5436
Epoch 9/100
463/463 [==============================] - 1s 2ms/step - loss: 0.7247 - accuracy: 0.5347
Epoch 10/100
463/463 [==============================] - 1s 2ms/step - loss: 0.7170 - accuracy: 0.5417
Epoch 11/100
463/463 [==============================] - 1s 2ms/step - loss: 0.7136 - accuracy: 0.5453
Epoch 12/100
463/463 [==============================] - 1s 2ms/step - loss: 0.7211 - accuracy: 0.5396
```

In [8]:
```
# Use the trained model to make predictions on the testing set
y_pred = model.predict(X_test)
y_pred = np.round(y_pred)
```

```
116/116 [==============================] - 0s 1ms/step
```

In [9]:
```
y_pred2 = model.predict(X_train)
y_pred2 = np.round(y_pred)
```

```
463/463 [==============================] - 1s 1ms/step
```

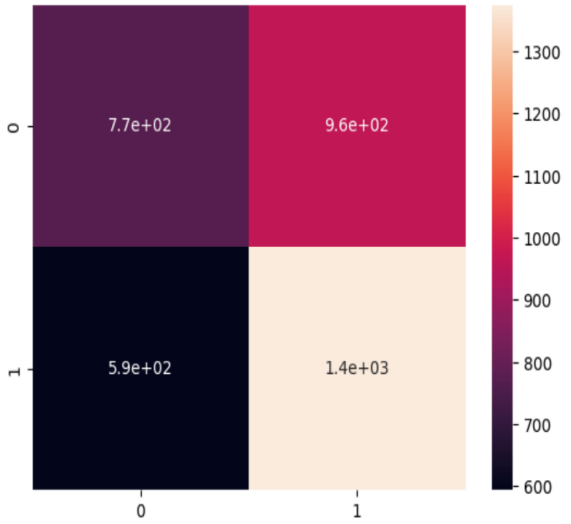
In [10]:
```
# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", round(accuracy*100,2),'%')
```

```
Accuracy: 57.91 %
```

In [11]:
```
cm = confusion_matrix(y_test, y_pred)
```

In [12]:
```
snb.heatmap(cm,annot=True)
cm
```

Out[12]:
```
array([[ 770,  965],
       [ 594, 1375]], dtype=int64)
```



In [13]:
```
# precision for testing
print('for testing : ' , round(precision_score(y_test , y_pred)*100,2),'%')
```

```
for testing :  58.76 %
```

In [14]:
```
# Recall for testing
print('for testing : ' , round(recall_score(y_test , y_pred)*100,2),'%')
```

```
for testing :  69.83 %
```

In [15]:
```
# f1_score for testing
print('for testing : ' , round(f1_score(y_test , y_pred)*100,2),'%')
```

```
for testing :  63.82 %
```

In [16]:
```
# for sensitivity and specificity
tn, fp, fn, tp = cm.ravel()
```

In [17]:
```
specificity = tn / (tn+fp)
print('specificity for data is : ',round(specificity*100,2),'%')
```

```
specificity for data is :  44.38 %
```

In [18]:
```
sensitivity = tp / (tp+fn)
print('sensitivity for data is : ',round(sensitivity*100,2),'%')
```

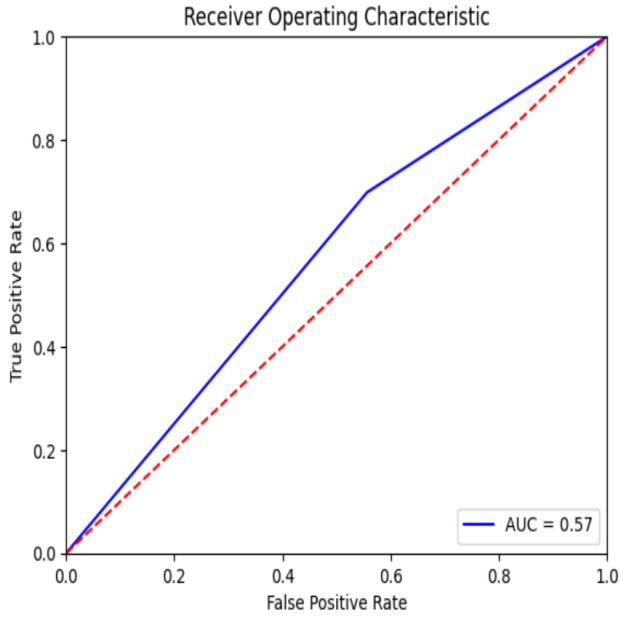
```
sensitivity for data is :  69.83 %
```

In [19]:
```
#ROC and AUC
fpr, tpr, threshold = roc_curve(y_test,y_pred)
roc_auc = auc(fpr, tpr)
```

In [20]:
```
print('Area under curve : ',round(roc_auc*100,2),'%')
```

```
Area under curve :  57.11 %
```

In [21]:
```
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()
```



Code for implementing ensemble Model (SVM-KNN).

In [1]:
```
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score , recall_score , f1_score , confusion_matrix
from sklearn.metrics import roc_curve , auc
import matplotlib.pyplot as plt
import seaborn as snb
```

In [3]:
```
# Create SVM and KNN classifiers
svm = SVC(kernel='linear', C=1)
knn = KNeighborsClassifier(n_neighbors=5)
```

In [4]:
```
# Train classifiers on training data
svm.fit(X_train, y_train)
knn.fit(X_train, y_train)
```

Out[4]:
```
KNeighborsClassifier
KNeighborsClassifier()
```

In [5]:
```
# Make predictions on testing data
svm_preds = svm.predict(X_test)
knn_preds = knn.predict(X_test)
```

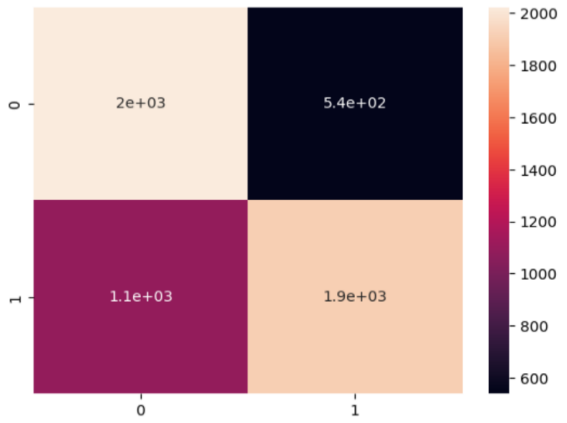
In [6]:
```
# Combine predictions using majority voting
ensemble_preds = []
for i in range(len(X_test)):
    if svm_preds[i] == knn_preds[i]:
        ensemble_preds.append(svm_preds[i])
    else:
        ensemble_preds.append(svm_preds[i])  # You can also use knn_preds here
```

In [7]:
```
# Calculate accuracy of ensemble predictions
ensemble_acc = accuracy_score(y_test, ensemble_preds)
print("Ensemble accuracy:", round(ensemble_acc*100,2),'%')
```

```
Ensemble accuracy: 70.72 %
```

In [8]:
```
cm = confusion_matrix(y_test, ensemble_preds)
snb.heatmap(cm,annot=True)
cm
```

Out[8]:
```
array([[2022,  539],
       [1088, 1907]], dtype=int64)
```



In [9]:
```
# precision for testing
print('for testing : ' , round(precision_score(y_test , ensemble_preds)*100,2),'%')
```

```
for testing :  77.96 %
```

In [10]:
```
# Recall for testing
print('for testing : ' , round(recall_score(y_test , ensemble_preds)*100,2),'%')
```

```
for testing :  63.67 %
```

In [11]:
```
# f1_score for testing
print('for testing : ' , round(f1_score(y_test , ensemble_preds)*100,2),'%')
```

```
for testing :  70.1 %
```

In [12]:
```
# for sensitivity and specificity
tn, fp, fn, tp = cm.ravel()
```

In [13]:
```
specificity = tn / (tn+fp)
print('specificity for data is : ',round(specificity*100,2),'%')
```

```
specificity for data is :  78.95 %
```

In [14]:
```
sensitivity = tp / (tp+fn)
print('sensitivity for data is : ',round(sensitivity*100,2),'%')
```

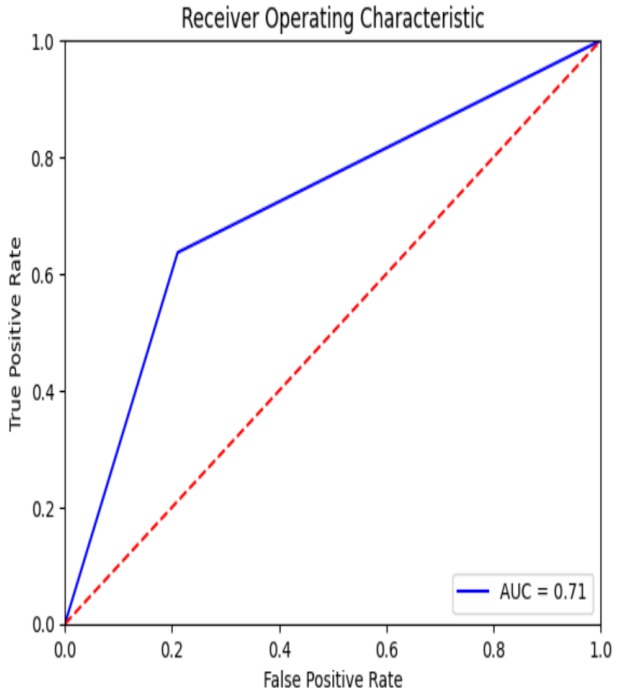
```
sensitivity for data is :  63.67 %
```

In [15]:
```
#ROC and AUC
fpr, tpr, threshold = roc_curve(y_test,ensemble_preds)
roc_auc = auc(fpr, tpr)
```

In [16]:
```
print('Area under curve : ',round(roc_auc*100,2),'%')
```

```
Area under curve :  71.31 %
```

In [17]:
```
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()
```



Code for implementing ensemble Model (KNN-RandomForest).

In [1]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score , recall_score , f1_score , confusion_matrix
from sklearn.metrics import roc_curve , auc
import matplotlib.pyplot as plt
import seaborn as snb
```

In [5]:
```
# Create KNN and Random Forest classifiers
knn = KNeighborsClassifier(n_neighbors=5)
rf = RandomForestClassifier(n_estimators=100)
```

In [6]:
```
# Train classifiers on training data
knn.fit(X_train, y_train)
rf.fit(X_train, y_train)
```

Out[6]:
```
RandomForestClassifier
RandomForestClassifier()
```

In [7]:
```
# Make predictions on testing data
knn_preds = knn.predict(X_test)
rf_preds = rf.predict(X_test)
```

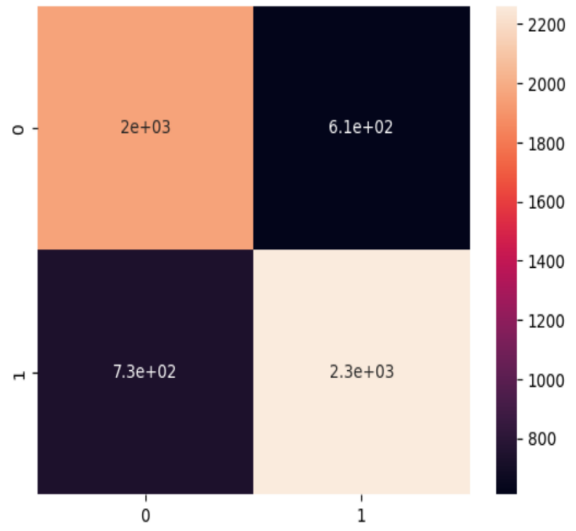
In [8]:
```
# Combine predictions using majority voting
ensemble_preds = []
for i in range(len(X_test)):
    if knn_preds[i] == rf_preds[i]:
        ensemble_preds.append(knn_preds[i])
    else:
        ensemble_preds.append(knn_preds[i])  # You can also use rf_preds here
```

In [9]:
```
# Calculate accuracy of ensemble predictions
ensemble_acc = accuracy_score(y_test, ensemble_preds)
print("Ensemble accuracy:", round(ensemble_acc*100,2),'%')
```

```
Ensemble accuracy: 75.83 %
```

In [10]:
```
cm = confusion_matrix(y_test, ensemble_preds)
snb.heatmap(cm,annot=True)
cm
```

Out[10]:
```
array([[1952,  609],
       [ 734, 2261]], dtype=int64)
```



In [11]:
```
# precision for testing
print('for testing : ' , round(precision_score(y_test , ensemble_preds)*100,2),'%')
```

```
for testing :  78.78 %
```

In [12]:
```
# Recall for testing
print('for testing : ' , round(recall_score(y_test , ensemble_preds)*100,2),'%')
```

```
for testing :  75.49 %
```

In [13]:
```
# f1_score for testing
print('for testing : ' , round(f1_score(y_test , ensemble_preds)*100,2),'%')
```

```
for testing :  77.1 %
```

In [14]:
```
# for sensitivity and specificity
tn, fp, fn, tp = cm.ravel()
```

In [15]:
```
specificity = tn / (tn+fp)
print('specificity for data is : ',round(specificity*100,2),'%')
```

```
specificity for data is :  76.22 %
```

In [16]:
```
sensitivity = tp / (tp+fn)
print('sensitivity for data is : ',round(sensitivity*100,2),'%')
```

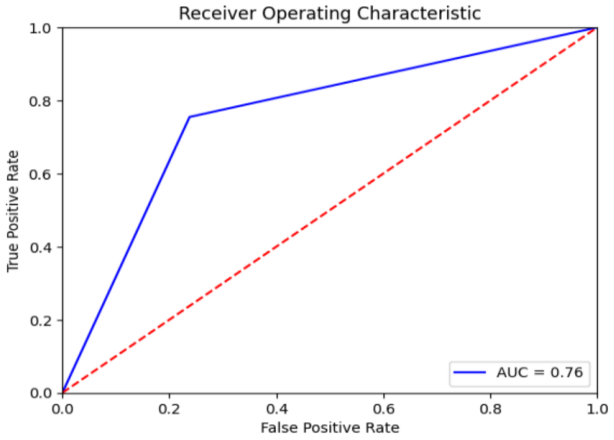
```
sensitivity for data is :  75.49 %
```

In [17]:
```
#ROC and AUC
fpr, tpr, threshold = roc_curve(y_test,ensemble_preds)
roc_auc = auc(fpr, tpr)
```

In [18]:
```
print('Area under curve : ',round(roc_auc*100,2),'%')
```

```
Area under curve :  75.86 %
```

In [19]:
```
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()
```



Code for implementing ensemble Model (RandomForest-SVM).

In [1]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score , recall_score , f1_score , confusion_matrix
from sklearn.metrics import roc_curve , auc
import matplotlib.pyplot as plt
import seaborn as snb
```

In [5]:
```
# Create Random Forest and SVM classifiers
rf = RandomForestClassifier(n_estimators=100)
svm = SVC(kernel='linear', C=1)
```

In [6]:
```
# Train classifiers on training data
rf.fit(X_train, y_train)
svm.fit(X_train, y_train)
```

Out[6]:
```
SVC(C=1, kernel='linear')
```

In [7]:
```
# Make predictions on testing data
rf_preds = rf.predict(X_test)
svm_preds = svm.predict(X_test)
```

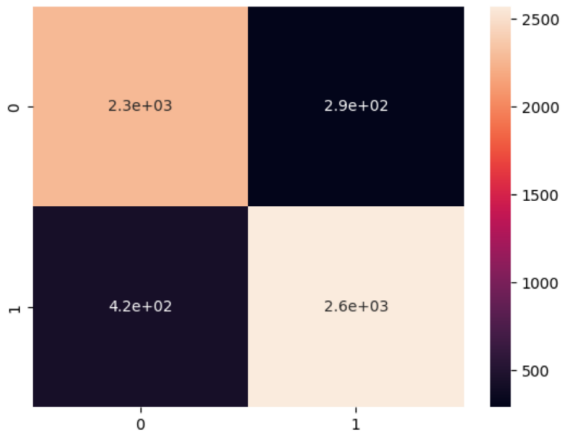
In [8]:
```
# Combine predictions using majority voting
ensemble_preds = []
for i in range(len(X_test)):
    if rf_preds[i] == svm_preds[i]:
        ensemble_preds.append(rf_preds[i])
    else:
        ensemble_preds.append(rf_preds[i])  # You can also use svm_preds here
```

In [9]:
```
# Calculate accuracy of ensemble predictions
ensemble_acc = accuracy_score(y_test, ensemble_preds)
print("Ensemble accuracy:", round(ensemble_acc*100,2),'%')
```

```
Ensemble accuracy: 87.24 %
```

In [10]:
```
cm = confusion_matrix(y_test, ensemble_preds)
snb.heatmap(cm,annot=True)
cm
```

Out[10]:
```
array([[2274,  287],
       [ 422, 2573]], dtype=int64)
```



In [11]:
```
# precision for testing
print('for testing : ' , round(precision_score(y_test , ensemble_preds)*100,2),'%')
```

```
for testing :  89.97 %
```

In [12]:
```
# Recall for testing
print('for testing : ' , round(recall_score(y_test , ensemble_preds)*100,2),'%')
```

```
for testing :  85.91 %
```

In [13]:
```
# f1_score for testing
print('for testing : ' , round(f1_score(y_test , ensemble_preds)*100,2),'%')
```

```
for testing :  87.89 %
```

In [14]:
```
# for sensitivity and specificity
tn, fp, fn, tp = cm.ravel()
```

In [15]:
```
specificity = tn / (tn+fp)
print('specificity for data is : ',round(specificity*100,2),'%')
```

```
specificity for data is :  88.79 %
```

In [16]:
```
sensitivity = tp / (tp+fn)
print('sensitivity for data is : ',round(sensitivity*100,2),'%')
```

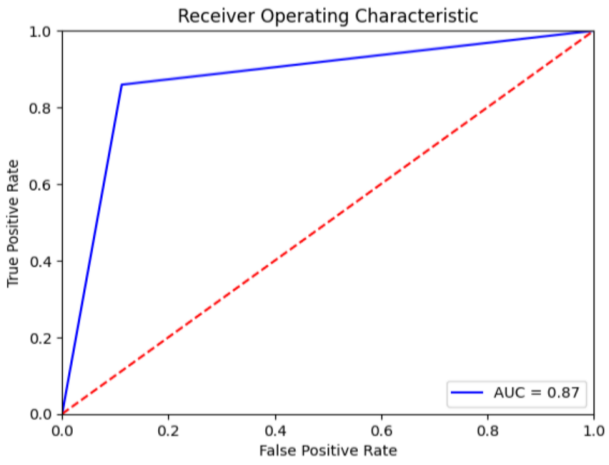
```
sensitivity for data is :  85.91 %
```

In [17]:
```
#ROC and AUC
fpr, tpr, threshold = roc_curve(y_test,ensemble_preds)
roc_auc = auc(fpr, tpr)
```

In [18]:
```
print('Area under curve : ',round(roc_auc*100,2),'%')
```

```
Area under curve :  87.35 %
```

In [19]:
```
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()
```



Code for implementing ensemble Model (RandomForest-ANN).

In [1]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score , recall_score , f1_score , confusion_matrix
from sklearn.metrics import roc_curve , auc
import matplotlib.pyplot as plt
import seaborn as snb
```

In [5]:
```
# Create Random Forest and ANN classifiers
rf = RandomForestClassifier(n_estimators=100)
ann = MLPClassifier(hidden_layer_sizes=(100, 50), activation='relu', solver='adam')
```

In [6]:
```
# Train classifiers on training data
rf.fit(X_train, y_train)
ann.fit(X_train, y_train)
```

Out[6]:
```
MLPClassifier(hidden_layer_sizes=(100, 50))
```

In [7]:
```
# Make predictions on testing data
rf_preds = rf.predict(X_test)
ann_preds = ann.predict(X_test)
```

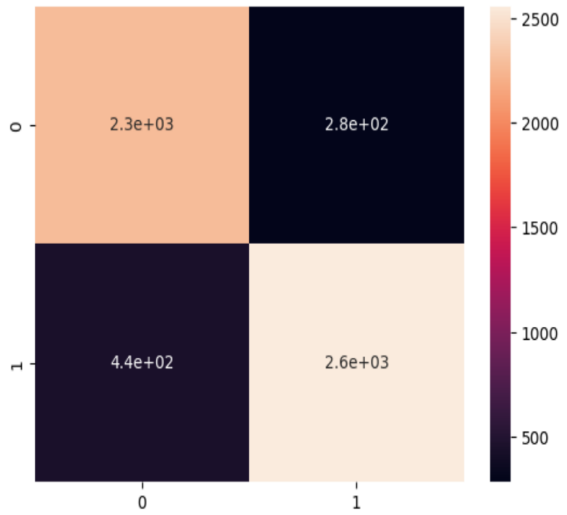
In [8]:
```
# Combine predictions using majority voting
ensemble_preds = []
for i in range(len(X_test)):
    if rf_preds[i] == ann_preds[i]:
        ensemble_preds.append(rf_preds[i])
    else:
        ensemble_preds.append(rf_preds[i])  # You can also use ann_preds here
```

In [9]:
```
# Calculate accuracy of ensemble predictions
ensemble_acc = accuracy_score(y_test, ensemble_preds)
print("Ensemble accuracy:", round(ensemble_acc*100,2),'%')
```

```
Ensemble accuracy: 87.06 %
```

In [10]:
```
cm = confusion_matrix(y_test, ensemble_preds)
snb.heatmap(cm,annot=True)
cm
```

Out[10]:
```
array([[2279,  282],
       [ 437, 2558]], dtype=int64)
```



In [11]:
```
# precision for testing
print('for testing : ' , round(precision_score(y_test , ensemble_preds)*100,2),'%')
```

```
for testing :  90.07 %
```

In [12]:
```
# Recall for testing
print('for testing : ' , round(recall_score(y_test , ensemble_preds)*100,2),'%')
```

```
for testing :  85.41 %
```

In [13]:
```
# f1_score for testing
print('for testing : ' , round(f1_score(y_test , ensemble_preds)*100,2),'%')
```

```
for testing :  87.68 %
```

In [14]:
```
# for sensitivity and specificity
tn, fp, fn, tp = cm.ravel()
```

In [15]:
```
specificity = tn / (tn+fp)
print('specificity for data is : ',round(specificity*100,2),'%')
```

```
specificity for data is :  88.99 %
```

In [16]:
```
sensitivity = tp / (tp+fn)
print('sensitivity for data is : ',round(sensitivity*100,2),'%')
```

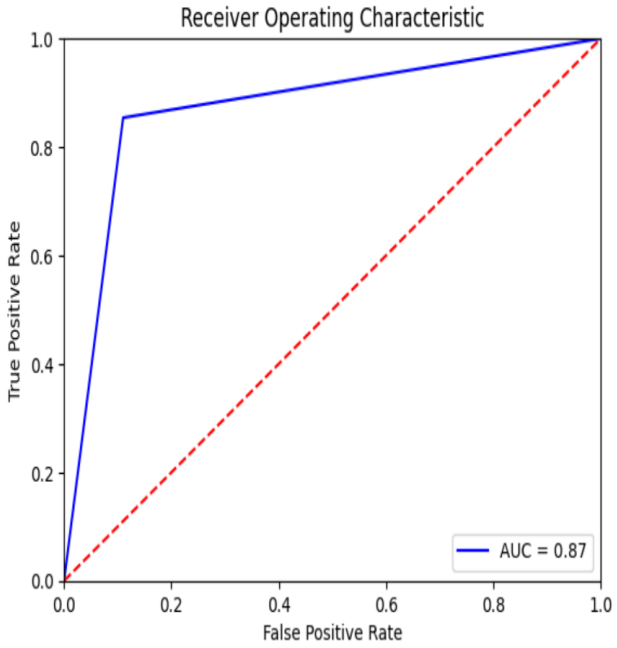
```
sensitivity for data is :  85.41 %
```

In [17]:
```
#ROC and AUC
fpr, tpr, threshold = roc_curve(y_test,ensemble_preds)
roc_auc = auc(fpr, tpr)
```

In [18]:
```
print('Area under curve : ',round(roc_auc*100,2),'%')
```

```
Area under curve :  87.2 %
```

In [19]:
```
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()
```



Code for implementing ensemble Model (KNN-ANN).

In [1]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score , recall_score , f1_score , confusion_matrix
from sklearn.metrics import roc_curve , auc
import matplotlib.pyplot as plt
import seaborn as snb
```

In [4]:
```
# Train classifiers on training data
knn.fit(X_train, y_train)
ann.fit(X_train, y_train)
```

Out[4]:
```
MLPClassifier(hidden_layer_sizes=(100, 50))
```

In [5]:
```
# Make predictions on testing data
knn_preds = knn.predict(X_test)
ann_preds = ann.predict(X_test)
```

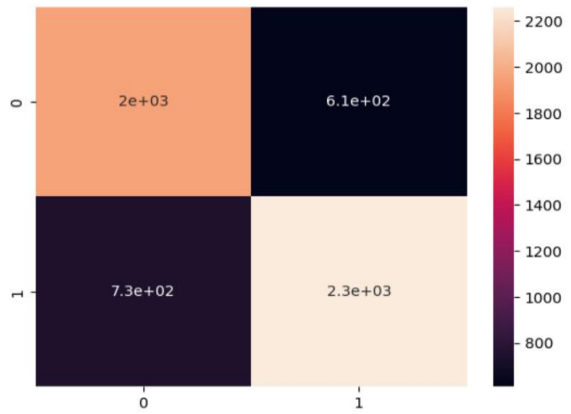
In [6]:
```
# Combine predictions using majority voting
ensemble_preds = []
for i in range(len(X_test)):
    if knn_preds[i] == ann_preds[i]:
        ensemble_preds.append(knn_preds[i])
    else:
        ensemble_preds.append(knn_preds[i])  # You can also use ann_preds here
```

In [7]:
```
# Calculate accuracy of ensemble predictions
ensemble_acc = accuracy_score(y_test, ensemble_preds)
print("Ensemble accuracy:", round(ensemble_acc*100,2),'%')
```

```
Ensemble accuracy: 75.83 %
```

In [8]:
```
cm = confusion_matrix(y_test, ensemble_preds)
snb.heatmap(cm,annot=True)
cm
```

Out[8]:
```
array([[1952,  609],
       [ 734, 2261]], dtype=int64)
```



In [9]:
```
# precision for testing
print('for testing : ' , round(precision_score(y_test , ensemble_preds)*100,2),'%')
```

```
for testing :  78.78 %
```

In [10]:
```
# Recall for testing
print('for testing : ' , round(recall_score(y_test , ensemble_preds)*100,2),'%')
```

```
for testing :  75.49 %
```

In [11]:
```
# f1_score for testing
print('for testing : ' , round(f1_score(y_test , ensemble_preds)*100,2),'%')
```

```
for testing :  77.1 %
```

In [12]:
```
# for sensitivity and specificity
tn, fp, fn, tp = cm.ravel()
```

In [13]:
```
specificity = tn / (tn+fp)
print('specificity for data is : ',round(specificity*100,2),'%')
```

```
specificity for data is :  76.22 %
```

In [14]:
```
sensitivity = tp / (tp+fn)
print('sensitivity for data is : ',round(sensitivity*100,2),'%')
```

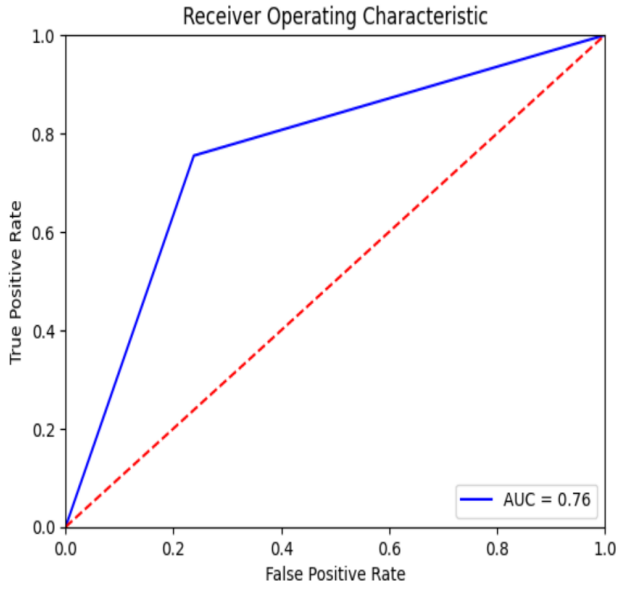
```
sensitivity for data is :  75.49 %
```

In [15]:
```
#ROC and AUC
fpr, tpr, threshold = roc_curve(y_test,ensemble_preds)
roc_auc = auc(fpr, tpr)
```

In [16]:
```
print('Area under curve : ',round(roc_auc*100,2),'%')
```

```
Area under curve :  75.86 %
```

In [17]:
```
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()
```



Code for ROC Curve of all Model Trained for influenza detection.

In [157]:
```
fig, ax = plt.subplots(1, figsize=(8, 6))
fig.suptitle('ROC curve', fontsize=15)
ax.plot(fpr_rfknn, tpr_rfknn, 'b',linestyle = '-', label = 'RF-KNN = %0.2f' % roc_auc_rfknn)
ax.plot(fpr_rfsvm, tpr_rfsvm, 'r',linestyle = '--', label = 'RF-SVM = %0.2f' % roc_auc_rfsvm)
ax.plot(fpr_rfann, tpr_rfann, 'cyan',linestyle = 'dotted', label = 'RF-ANN = %0.2f' % roc_auc_rfann)
ax.plot(fpr_knnann, tpr_knnann, 'c',linestyle = 'dashdot', label = 'KNN-ANN = %0.2f' % roc_auc_knnann)
ax.plot(fpr_svmknn, tpr_svmknn, 'k',linestyle = '--', label = 'SVM-KNN = %0.2f' % roc_auc_svmknn)
ax.plot(fpr_knn, tpr_knn, 'm',linestyle = ':', label = 'KNN = %0.2f' % roc_auc_knn)
ax.plot(fpr_rf,tpr_rf, 'b', linestyle = 'solid',label = 'RF = %0.2f' % roc_auc_rf)
ax.plot(fpr_ann, tpr_ann, 'pink', linestyle = '-',label = 'ANN = %0.2f' % roc_auc_ann)
ax.plot(fpr_svm, tpr_svm, 'g',linestyle = '--', label = 'SVM = %0.2f' % roc_auc_svm)
plt.legend(loc="lower right", title="Legend Title", frameon=False)
ax.set_xlabel('Sensitivity')
ax.set_ylabel('Specificity')
plt.show()
```