Spam Email Analysis with Machine Learning

In [4]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import svm
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score
```

In [5]:

```
spam = pd.read_csv('./spam.csv') # data set read
```

In [6]:

```
spam
```

Out[6]:

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ü b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

In [7]:

```
spam.shape
```

Out[7]:

```
(5572, 2)
```

In [8]:
```
spam.describe()
```

Out[8]:

	Category	Message
count	5572	5572
unique	2	5157
top	ham	Sorry, I'll call later
freq	4825	30

In [9]:
```
spam.groupby(spam['Category']).size()
```

Out[9]:
```
Category
ham     4825
spam     747
dtype: int64
```

We have a total of 5572 data. There are %83 safe and %17 spam.

In [10]:
```
spam.Category = spam.Category.apply(lambda x: 1 if x == 'spam' else 0)
```

We changed the spam values in the Category column with 1 and the raw values with 0.

In [11]:
```
spam.head()
```

Out[11]:

	Category	Message
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...

In [12]:
```
messages = spam.iloc[:,1] # Messages column select all rows and 2nd column
```

In [13]:
```
messages.head() # The code messages.head() returns the first five (head) rows
of the messages series. The head method is used to return the first n rows of
a Pandas series or dataframe. By default, head returns the first five rows, bu
t you can specify a different number by passing it as an argument, for example
messages.head(10) would return the first 10 rows.
```

Out[13]:
```
0    Go until jurong point, crazy.. Available only ...
1                        Ok lar... Joking wif u oni...
2    Free entry in 2 a wkly comp to win FA Cup fina...
3    U dun say so early hor... U c already then say...
4    Nah I don't think he goes to usf, he lives aro...
Name: Message, dtype: object
```

In [14]:

```
ifSpam = spam.iloc[:,0] # Spam column # The code ifSpam = spam.iloc[:,0] is selecting all rows (:) and first column (0) of a dataframe called spam and storing it as a series in the ifSpam variable. The .iloc accessor is used to extract data from a Pandas dataframe based on index location.
```

In [15]:

```
ifSpam.head()
```

Out[15]:

```
0    0
1    0
2    1
3    0
4    0
Name: Category, dtype: int64
```

In [16]:

```
messages_train, messages_test, ifSpam_train, ifSpam_test = train_test_split(messages, ifSpam, test_size=0.25)
```

We will use 75% of our dataset for training and 25% for testing

In [17]:

```
cv = CountVectorizer()
```

With CountVectorizer, text is analyzed and word counts are made and these are converted into vectors.

In [18]:

```
features = cv.fit_transform(messages_train)
```

In [19]:

```
features_test = cv.transform(messages_test)
```

Learning and Predicts

In [20]:

```
knModel = KNeighborsClassifier(n_neighbors=1) # creating an instance of the KNeighborsClassifier
```

In [21]:

```
knModel.fit(features, ifSpam_train) # The code knModel.fit(features, ifSpam_train) is training the knModel on the training data. The fit method is used to train a machine learning model on a given dataset.
```

Out[21]:

```
KNeighborsClassifier(n_neighbors=1)
```

In [22]:

```
knPredict = knModel.predict(features_test) # The code knPredict = knModel.predict(features_test) is using the trained knModel to make predictions on the test data. The predict method is used to make predictions using a trained machine learning model on a given dataset.
```

In [23]:

```
dtModel = tree.DecisionTreeClassifier()
```

In [24]:
```
dtModel.fit(features, ifSpam_train)
```

Out[24]:
```
DecisionTreeClassifier()
```

In [25]:
```
dtPredict = dtModel.predict(features_test)
```

In [26]:
```
svModel = svm.SVC()
```

In [27]:
```
svModel.fit(features,ifSpam_train)
```

Out[27]:
```
SVC()
```

In [28]:
```
svPredict = svModel.predict(features_test)
```

In [29]:
```
rfModel = RandomForestClassifier()
```

In [30]:
```
rfModel.fit(features, ifSpam_train)
```

Out[30]:
```
RandomForestClassifier()
```

In [31]:
```
rfPredict = rfModel.predict(features_test)
```

Visualization

In [32]:
```
from sklearn.metrics import plot_confusion_matrix,plot_precision_recall_curve,
plot_roc_curve
```

In [33]:
```
def visualization(model):
    predict = model.predict(features_test)
    plot_confusion_matrix(model,features_test,ifSpam_test)
    plot_precision_recall_curve(model,features_test,ifSpam_test)
    plot_roc_curve(model,features_test,ifSpam_test)
```

Results

K-Nearest Neighbors

In [34]:
```
print("Number of mislabeled out of a total of %d test entries: %d" % (features
_test.shape[0],
                                                                        (ifSpam_
test != knPredict).sum()))
```

```
Number of mislabeled out of a total of 1393 test entries: 70
```

In [35]:

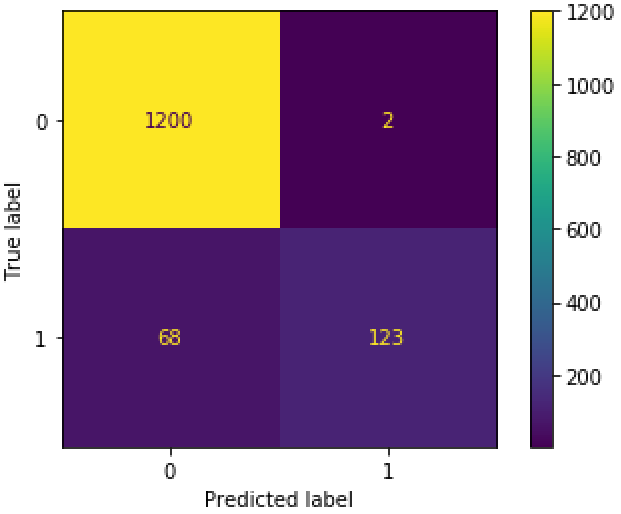
```
successRate = 100.0 * f1_score(ifSpam_test, knPredict, average='micro') # The code successRate = 100.0 * f1_score(ifSpam_test, knPredict, average='micro') is calculating the success rate of the knModel on the test data.

# The f1_score function is used to calculate the F1 score, which is a measure of a classifier's accuracy. The F1 score is the harmonic mean of precision and recall, where precision is the fraction of relevant instances among the retrieved instances, and recall is the fraction of relevant instances that have been retrieved.

# The ifSpam_test argument is the true target values of the test data, and knPredict argument is the predicted target values made by the knModel. The average parameter is set to 'micro', which specifies that a single F1 score should be returned for all classes, rather than one score per class.

# Finally, the success rate is calculated by multiplying the F1 score by 100 to convert it to a percentage. The success rate is stored in the successRate variable.
```

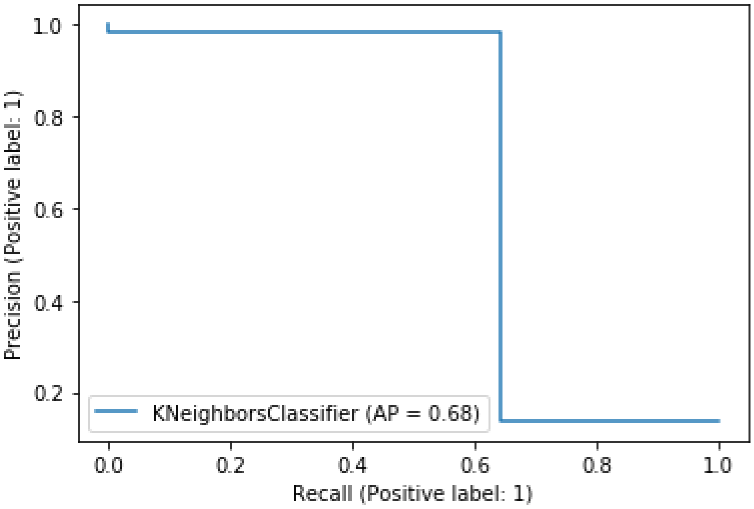
In [36]:

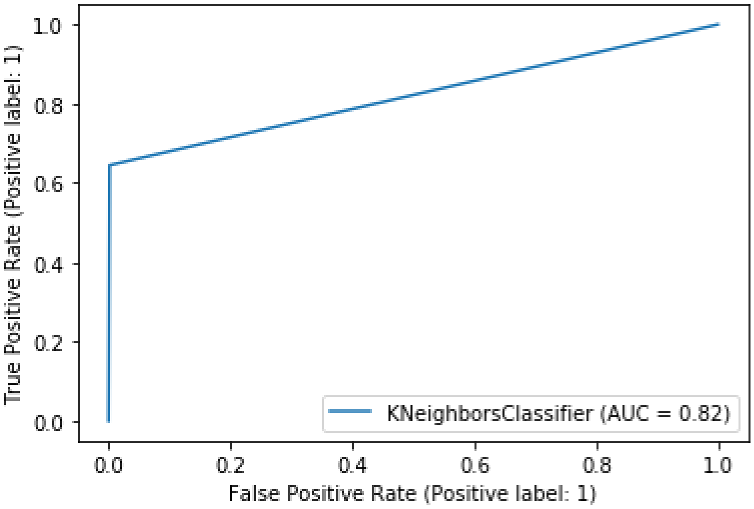
```
print("The Success Rate was calculated as % : " + str(successRate) + " with the K-Nearest-Neighbors")
```

```
The Success Rate was calculated as % : 94.9748743718593 with the K-Nearest-Neighbors
```

```
In [42]: visualization(knModel)
```







Random Forest

In [38]:

```
print("Number of mislabeled out of a total of %d test entries: %d" % (features_test.shape[0],
                                                                       (ifSpam_test != rfPredict).sum()))
```

```
Number of mislabeled out of a total of 1393 test entries: 34
```

In [39]:

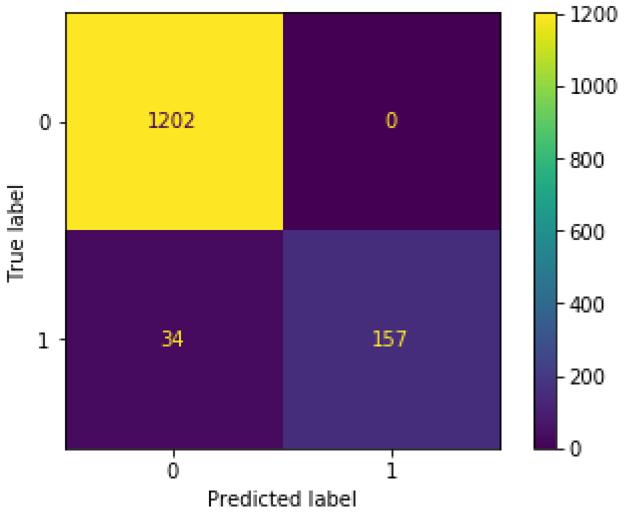
```
successRate = 100.0 * f1_score(ifSpam_test, rfPredict, average='micro')
```

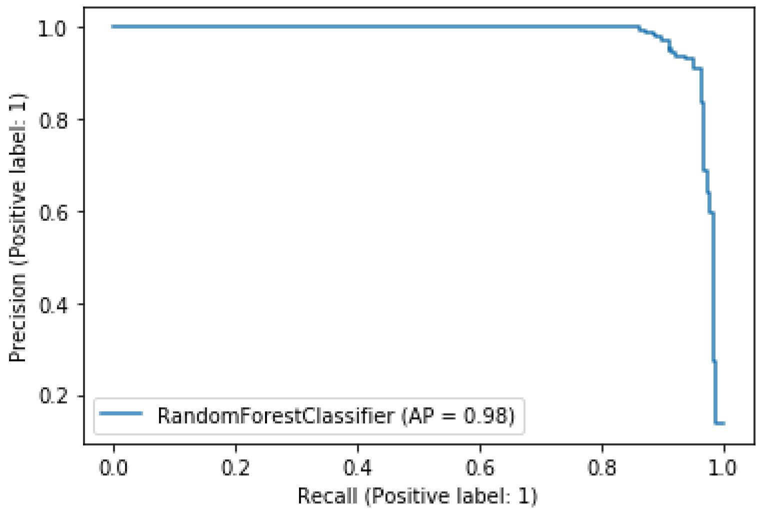
In [40]:

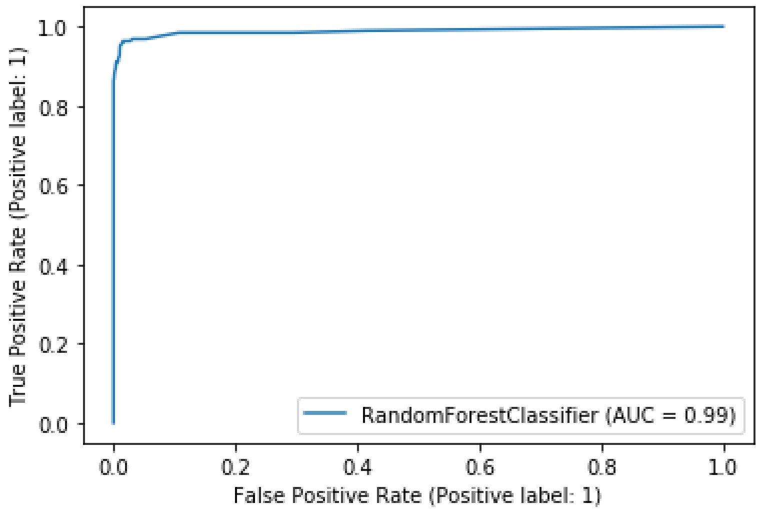
```
print("The Success Rate was calculated as % : " + str(successRate) + " with Random Forest")
```

```
The Success Rate was calculated as % : 97.5592246949031 with Random Forest
```

```
In [41]: visualization(rfModel)
```







Decision Tree

In [43]:

```
print("Number of mislabeled out of a total of %d test entries: %d" % (features
_test.shape[0],
                                                                      (ifSpam_
test != dtPredict).sum()))
```

```
Number of mislabeled out of a total of 1393 test entries: 45
```

In [44]:

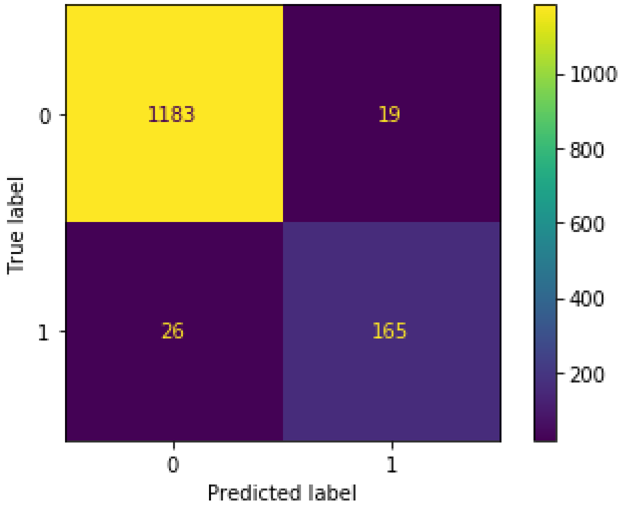
```
successRate = 100.0 * f1_score(ifSpam_test, dtPredict, average='micro')
```

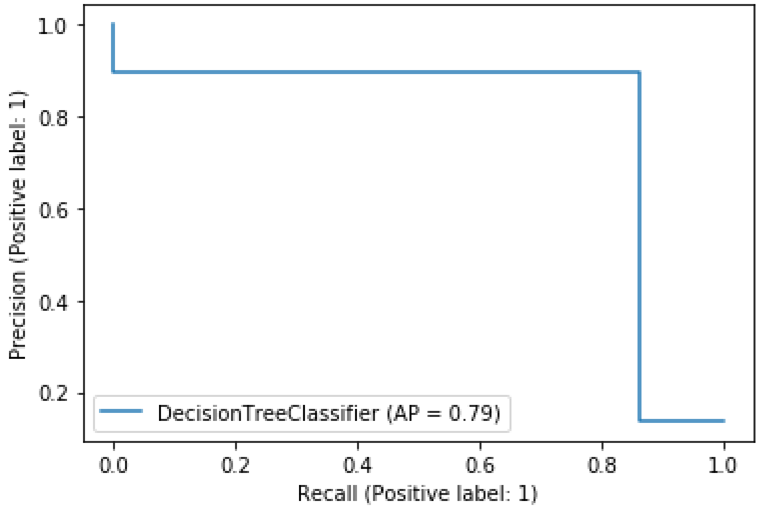
In [45]:

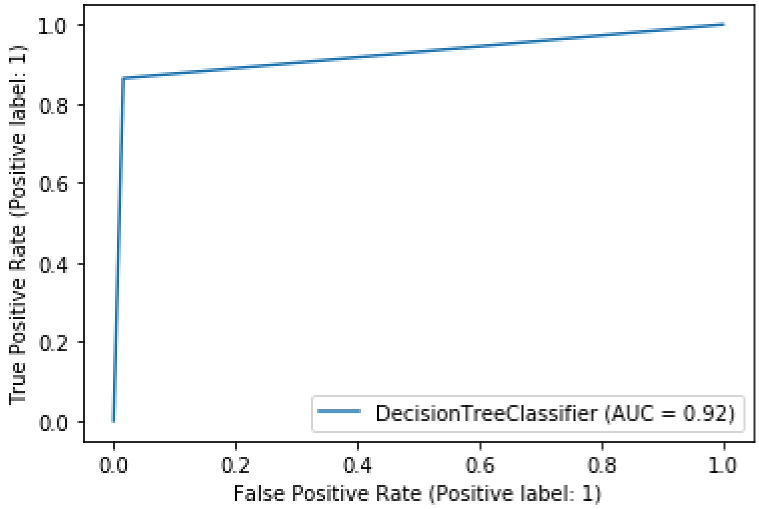
```
print("The Success Rate was calculated as % : " + str(successRate) + " with De
cision Tree")
```

```
The Success Rate was calculated as % : 96.76956209619526 with Decision Tree
```

```
In [46]: visualization(dtModel)
```







Support Vector Machine

In [47]:

```
print("Number of mislabeled out of a total of %d test entries: %d" % (features
_test.shape[0],
                                                                      (ifSpam_
test != svPredict).sum()))
```

```
Number of mislabeled out of a total of 1393 test entries: 27
```

In [48]:

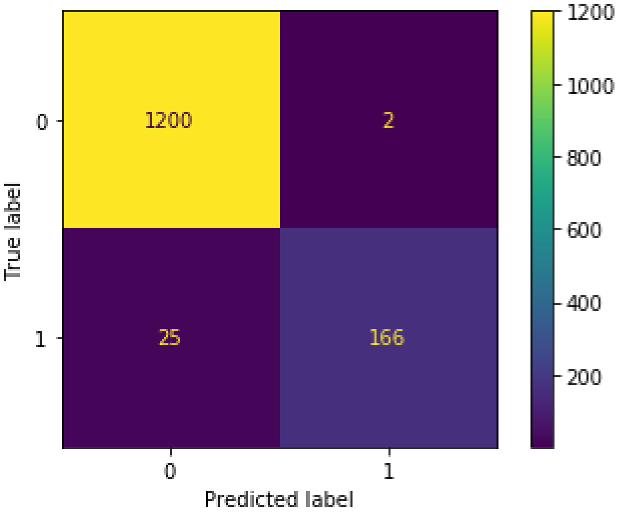
```
successRate = 100.0 * f1_score(ifSpam_test, svPredict, average='micro')
```

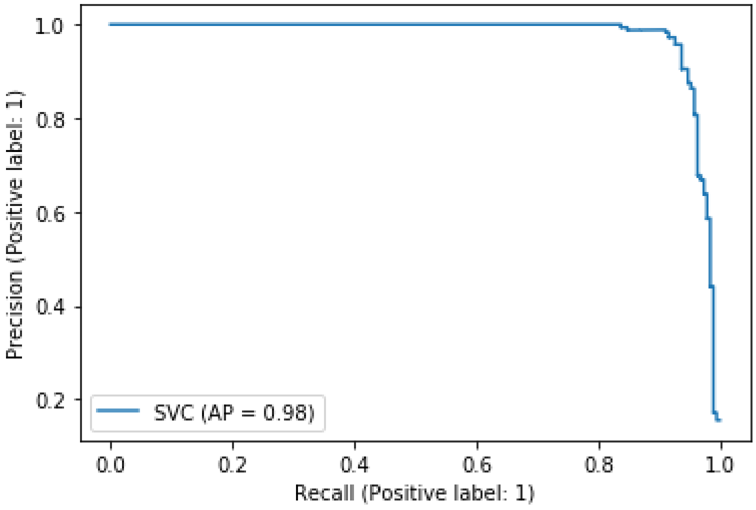
In [49]:

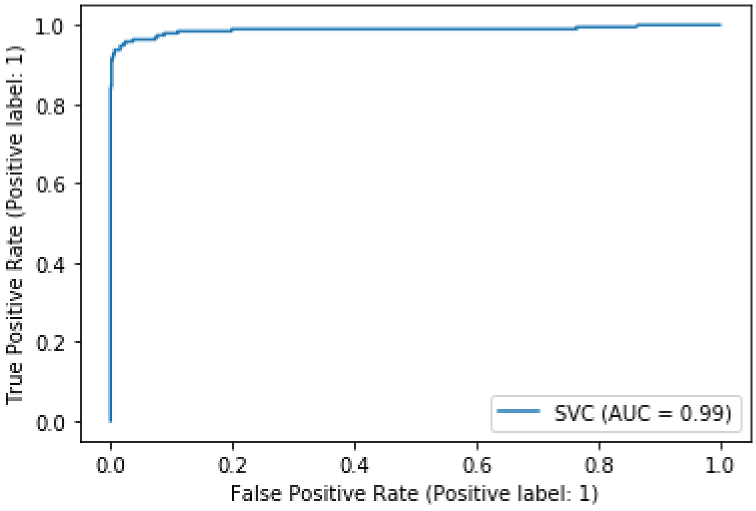
```
print("The Success Rate was calculated as % : " + str(successRate) + " with Su
pport Vector Machine")
```

```
The Success Rate was calculated as % : 98.06173725771716 with Support Vector
Machine
```

```
In [50]: visualization(svModel)
```







```
In [ ]:
```

```
In [ ]:
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: