```
pip install sklearn
```

```
Requirement already satisfied: sklearn in /usr/local/lib/python3.7/dist-packages (0.0
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numpy>=1.14.6 in /usr/local/lib/python3.7/dist-package
```

```
pip install numpy
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (1.21
```

```
pip install pandas
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (1.3
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dis
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (fr
```

```
pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.7/dist-packages (3.2.5)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from nl
```

```
pip install matplotlib
```

```
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (fr
```

```
pip install imblearn
```

```
Requirement already satisfied: imblearn in /usr/local/lib/python3.7/dist-packages (0
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: scikit-learn>=0.24 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.feature_extraction.text import TfidfTransformer, CountVectorizer, TfidfVector
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

from nltk.stem.porter import PorterStemmer
import nltk
import re, string
from nltk.corpus import stopwords

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import cross_val_score

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import plot_precision_recall_curve
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import classification_report
from sklearn import metrics


url = 'https://drive.google.com/uc?export=download&id=12fBlhsa5GIdtme1jT3KlPPIgIdjzqhv1'
df = pd.read_json(url, lines= True,orient='columns')
df.head
```

```
<bound method NDFrame.head of                                                    conter
0                                Get fucking real dude.  ...    NaN
1       She is as dirty as they come  and that crook ...  ...    NaN
2       why did you fuck it up. I could do it all day...  ...    NaN
3       Dude they dont finish enclosing the fucking s...  ...    NaN
4       WTF are you talking about Men? No men thats n...  ...    NaN
...                                                   ...  ...    ...
19996    I dont. But what is complaining about it goi...  ...    NaN
19997   Bahah  yeah i&;m totally just gonna&; get pis...  ...    NaN
19998       hahahahaha >:) im evil mwahahahahahahahahaha  ...    NaN
19999            What&;s something unique about Ohio? :)  ...    NaN
20000              Who is the biggest gossiper you know?  ...    NaN
```

```
[20001 rows x 3 columns]>
```

```
for i in range(0,len(df)):
    if df.annotation[i]['label'][0] == '1':
        df.annotation[i] = 1
    else:
        df.annotation[i] = 0
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarnir
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: SettingWithCopyWarnir
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  """
```

```
df.drop(['extras'],axis = 1,inplace = True)
df
```

	content	annotation
0	Get fucking real dude.	1
1	She is as dirty as they come and that crook ...	1
2	why did you fuck it up. I could do it all day...	1
3	Dude they dont finish enclosing the fucking s...	1
4	WTF are you talking about Men? No men thats n...	1
...	...	...
19996	I dont. But what is complaining about it goi...	0
19997	Bahah yeah i&;m totally just gonna&; get pis...	0
19998	hahahahaha >:) im evil mwahahahahahahahahaha	0
19999	What&;s something unique about Ohio? :)	0
20000	Who is the biggest gossiper you know?	0

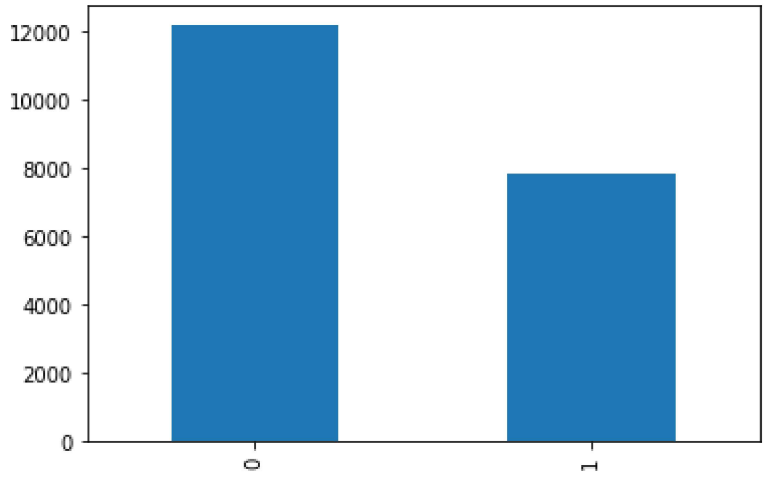
20001 rows × 2 columns

```
df.shape
```

```
(20001, 2)
```

```
df['annotation'].value_counts().sort_index().plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe963bc2450>
```




```
print("PosiNon cyber trollingtive: ", df.annotation.value_counts()[0]/len(df.annotation)*1
print("Cybertrolling: ", df.annotation.value_counts()[1]/len(df.annotation)*100,"%")
```

```
PosiNon cyber trollingtive:  60.89195540222989 %
Cybertrolling:  39.10804459777012 %
```

```
nltk.download('stopwords')
stop = stopwords.words('english')

regex = re.compile('[%s]' % re.escape(string.punctuation))

def test_re(s):
    return regex.sub('', s)

df ['content_without_stopwords'] = df['content'].apply(lambda x: ' '.join([word for word i
df ['content_without_puncs'] = df['content_without_stopwords'].apply(lambda x: regex.sub('
del df['content_without_stopwords']
del df['content']
df
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

	annotation	content_without_puncs

```
#Stemming
porter_stemmer = PorterStemmer()
#punctuations
nltk.download('punkt')
tok_list = []
size = df.shape[0]

for i in range(size):
  word_data = df['content_without_puncs'][i]
  nltk_tokens = nltk.word_tokenize(word_data)
  final = ''
  for w in nltk_tokens:
    final = final + ' ' + porter_stemmer.stem(w)
  tok_list.append(final)

df['content_tokenize'] = tok_list
del df['content_without_puncs']
df
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

	annotation	content_tokenize
0	1	get fuck real dude
1	1	she dirti come crook rengel dem fuck corrupt ...
2	1	fuck up I could day too let hour ping later s...
3	1	dude dont finish enclos fuck shower I hate ha...
4	1	wtf talk men No men that menag that gay
...	...	...
19996	0	I dont but complain go do
19997	0	bahah yeah im total gon na get piss talk you ...
19998	0	hahahahaha im evil mwahahahahahahahahaha
19999	0	what someth uniqu ohio
20000	0	who biggest gossip know

20001 rows × 2 columns

```
noNums = []
for i in range(len(df)):
  noNums.append(''.join([i for i in df['content_tokenize'][i] if not i.isdigit()]))

df['content'] = noNums
df
```

	annotation	content_tokenize	
0	1	get fuck real dude	g
1	1	she dirti come crook rengel dem fuck corrupt ...	she dirti come crook rengel de
2	1	fuck up I could day too let hour ping later s...	fuck up I could day too let h
3	1	dude dont finish enclos fuck shower I hate ha...	dude dont finish enclos fuck s
4	1	wtf talk men No men that menag that gay	wtf talk men No men tha
...	...	...	
19996	0	I dont but complain go do	I dont bu
19997	0	bahah yeah im total gon na get piss talk you ...	bahah yeah im total gon na g
19998	0	hahahahaha im evil mwahahahahahahahahaha	hahahahaha im evil mwahahal
19999	0	what someth uniqu ohio	what s
20000	0	who biggest gossip know	who big

20001 rows × 3 columns

```
tfIdfVectorizer=TfidfVectorizer(use_idf=True, sublinear_tf=True)
tfIdf = tfIdfVectorizer.fit_transform(df.content.tolist())
```

```
print(tfIdf)
```

```
  (0, 3598)     0.5682792040556577
  (0, 10534)    0.6408032598619846
  (0, 4665)     0.3314842764826402
  (0, 4896)     0.3956616014132561
  (1, 7497)     0.1421522208901913
  (1, 7670)     0.18997382467613527
  (1, 10707)    0.3380770158779807
  (1, 7868)     0.17712641457020445
  (1, 6881)     0.2707206754001475
  (1, 2649)     0.3478358132370042
  (1, 3127)     0.36956626902789813
  (1, 10686)    0.36956626902789813
  (1, 2791)     0.3609013757539863
  (1, 2453)     0.20014266836955738
  (1, 3306)     0.294004579420996
  (1, 11402)    0.24231137330135857
  (1, 4665)     0.12302268120056382
  (2, 5648)     0.26264752682375
  (2, 1476)     0.2858475342270202
  (2, 14420)    0.28761927584628644
  (2, 11156)    0.4130661580674724
  (2, 7317)     0.3061308801267633
  (2, 9784)     0.38298243181872793
  (2, 5956)     0.28144866948736874
  (2, 7434)     0.24199503289435126
  :     :
  (19997, 8527) 0.362558005670761
  (19997, 14527)        0.1829917686470462
  (19997, 364)  0.2524980709313037
```

```
  (19997, 8632) 0.19487099515279527
  (19997, 5039) 0.21529577669215724
  (19997, 14639)        0.15162817445998714
  (19997, 5311) 0.2322934882970198
  (19997, 9798) 0.22212274676003707
  (19997, 13161)        0.22711912398563924
  (19997, 6367) 0.1396437116782225
  (19997, 12782)        0.14437044050700218
  (19997, 12583)        0.21638447818263024
  (19997, 4896) 0.14751463907596812
  (19998, 8599) 0.6474267500657062
  (19998, 5355) 0.5240398795250955
  (19998, 4014) 0.5046761457592059
  (19998, 6367) 0.22698633410034566
  (19999, 13559)        0.6577171835959204
  (19999, 9144) 0.5711145182804813
  (19999, 11870)        0.38585942493978787
  (19999, 14101)        0.30388948253771536
  (20000, 5085) 0.7029240479741253
  (20000, 1246) 0.5142345992116426
  (20000, 14161)        0.4012493121480635
  (20000, 7153) 0.28365392515178917
```

```
print(tfIdf.shape) # means total rows  20001 with 14783 features
```

```
(20001, 14783)
```

```
df2 = pd.DataFrame(tfIdf[2].T.todense(), index=tfIdfVectorizer.get_feature_names(), column
df2 = df2.sort_values('TF-IDF', ascending=False)
print (df2.head(10))
```

```
         TF-IDF
sched  0.413066
ping   0.382982
later  0.306131
write  0.287619
book   0.285848
hour   0.281449
here   0.262648
let    0.241995
up     0.237401
could  0.223151
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
  warnings.warn(msg, category=FutureWarning)
```

```
dfx = pd.DataFrame(tfIdf.toarray(), columns = tfIdfVectorizer.get_feature_names())
print(dfx)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
  warnings.warn(msg, category=FutureWarning)
      aa  aaaaaaaaaa  aaaaaanndgummi  aaaagh  ...  zune  zzzz  zzzzzz  zzzzzzzz
0    0.0         0.0             0.0     0.0  ...   0.0   0.0     0.0       0.0
1    0.0         0.0             0.0     0.0  ...   0.0   0.0     0.0       0.0
2    0.0         0.0             0.0     0.0  ...   0.0   0.0     0.0       0.0
3    0.0         0.0             0.0     0.0  ...   0.0   0.0     0.0       0.0
```

```
4      0.0        0.0            0.0     0.0  ...   0.0   0.0     0.0       0.0
...    ...        ...            ...     ...  ...   ...   ...     ...       ...
19996  0.0        0.0            0.0     0.0  ...   0.0   0.0     0.0       0.0
19997  0.0        0.0            0.0     0.0  ...   0.0   0.0     0.0       0.0
19998  0.0        0.0            0.0     0.0  ...   0.0   0.0     0.0       0.0
19999  0.0        0.0            0.0     0.0  ...   0.0   0.0     0.0       0.0
20000  0.0        0.0            0.0     0.0  ...   0.0   0.0     0.0       0.0

[20001 rows x 14783 columns]
```

```
def display_scores(vectorizer, tfidf_result):
    scores = zip(vectorizer.get_feature_names(),
                 np.asarray(tfidf_result.sum(axis=0)).ravel())
    sorted_scores = sorted(scores, key=lambda x: x[1], reverse=True)
    i=0
    for item in sorted_scores:
        print ("{0:50} Score: {1}".format(item[0], item[1]))
        i = i+1
        if (i > 25):
          break


display_scores(tfIdfVectorizer, tfIdf)
```

```
hate                                               Score: 533.8157298036014
fuck                                               Score: 503.76150769255435
damn                                               Score: 482.3875012051478
suck                                               Score: 407.37790877127185
ass                                                Score: 337.54089621427744
that                                               Score: 311.6250930420745
lol                                                Score: 298.0085779872157
im                                                 Score: 296.0216055277791
like                                               Score: 287.8183474868775
you                                                Score: 284.7850587424088
it                                                 Score: 254.75722294501585
get                                                Score: 253.19747902607998
what                                               Score: 221.43673623523864
know                                               Score: 211.53595900888456
would                                              Score: 202.5073882820925
bitch                                              Score: 193.08800391463464
ye                                                 Score: 182.22364463196365
love                                               Score: 181.49014270754344
go                                                 Score: 180.2588319545915
haha                                               Score: 179.29466045019018
think                                              Score: 178.9039058038677
one                                                Score: 174.16019276608517
do                                                 Score: 160.57524593088053
time                                               Score: 160.1100301847739
gay                                                Score: 159.5820454915121
peopl                                              Score: 151.04499856119287
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
  warnings.warn(msg, category=FutureWarning)
```

```
X=tfIdf.toarray()
y = np.array(df.annotation.tolist())
```

```
#Splting
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(16000, 14783)
(16000,)
(4001, 14783)
(4001,)
```

```
#Training data biasness
unique_elements, counts_elements = np.unique(y_train, return_counts=True)
print(np.asarray((unique_elements, counts_elements)))
```

```
[[   0    1]
 [9750 6250]]
```

```
unique_elements, counts_elements = np.unique(y_test, return_counts=True)
print(np.asarray((unique_elements, counts_elements)))
```

```
[[   0    1]
 [2429 1572]]
```

```
#Random oversampling on training data
from imblearn.over_sampling import RandomOverSampler

oversample = RandomOverSampler(sampling_strategy='not majority')
X_over, y_over = oversample.fit_resample(X_train, y_train)
```

```
print(X_over.shape)
print(y_over.shape)
```

```
(19500, 14783)
(19500,)
```

```
unique_elements, counts_elements = np.unique(y_over, return_counts=True)
print(np.asarray((unique_elements, counts_elements)))
```

```
[[   0    1]
 [9750 9750]]
```

```
def getStatsFromModel(model):
  print(classification_report(y_test, y_pred))
  disp = plot_precision_recall_curve(model, X_test, y_test)
  disp.ax_.set_title('2-class Precision-Recall curve: '
                     'AP={0:0.2f}')

  logit_roc_auc = roc_auc_score(y_test, model.predict(X_test))
  fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_test)[:,1])
```

```
    plt.figure()
    plt.plot(fpr, tpr, label='(area = %0.2f)' % logit_roc_auc)
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.savefig('Log_ROC')
    plt.show()


#Supervised Methods
# 3 normal methods
# 2 ensemble methods
gnb = GaussianNB()
gnbmodel = gnb.fit(X_over, y_over)
y_pred = gnbmodel.predict(X_test)
print ("Score:", gnbmodel.score(X_test, y_test))
print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))
score_gau=gnbmodel.score(X_test, y_test)
getStatsFromModel(gnb)
```

```
Score: 0.6163459135216196
Confusion Matrix:
 [[ 925 1504]
 [  31 1541]]
              precision    recall  f1-score   support

           0       0.97      0.38      0.55      2429
           1       0.51      0.98      0.67      1572

    accuracy                           0.62      4001
   macro avg       0.74      0.68      0.61      4001
weighted avg       0.79      0.62      0.59      4001

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
  warnings.warn(msg, category=FutureWarning)
```

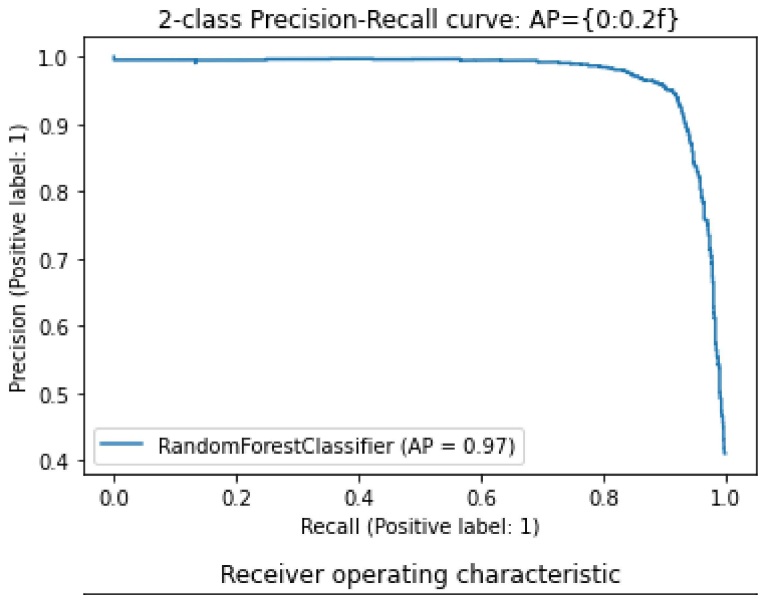
```
dtc = DecisionTreeClassifier()
dtc.fit(X_over, y_over)
y_pred = dtc.predict(X_test)
print("Accuracy: ",metrics.accuracy_score(y_test, y_pred))
print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))
getStatsFromModel(dtc)
score_dec=metrics.accuracy_score(y_test, y_pred)
```

```
Accuracy:  0.8437890527368158
Confusion Matrix:
 [[1873  556]
 [  69 1503]]
              precision    recall  f1-score   support

           0       0.96      0.77      0.86      2429
           1       0.73      0.96      0.83      1572

    accuracy                           0.84      4001
   macro avg       0.85      0.86      0.84      4001
weighted avg       0.87      0.84      0.85      4001
```

```
#Ensemble methods from here
#abc = AdaBoostClassifier()
#abc.fit(X_over, y_over)
#y_pred = abc.predict(X_test)
#print("Accuracy: ",metrics.accuracy_score(y_test, y_pred))
#print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))
#score_ada=metrics.accuracy_score(y_test, y_pred);
#getStatsFromModel(abc)
```

```
rfc = RandomForestClassifier(verbose=True) #uses randomized decision trees
rfcmodel = rfc.fit(X_over, y_over)
y_pred = rfc.predict(X_test)
print ("Score:", rfcmodel.score(X_test, y_test))
print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))
getStatsFromModel(rfc)
score_rfc=rfcmodel.score(X_test, y_test)
output = pd.DataFrame({'Predicted':y_pred}) # Heart-Disease yes or no? 1/0
print(output.head())
```

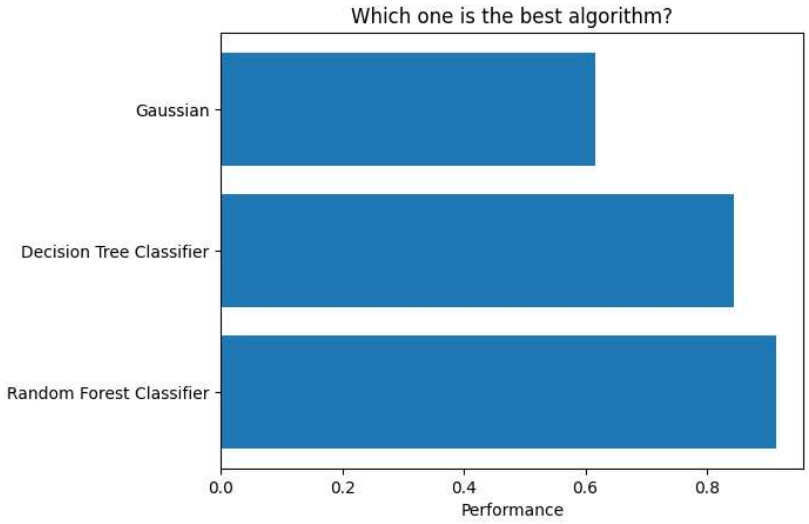
```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:  8.6min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    1.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    1.2s finished
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
  warnings.warn(msg, category=FutureWarning)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
Score: 0.9132716820794802
Confusion Matrix:
 [[2164  265]
 [  82 1490]]
              precision    recall  f1-score   support

           0       0.96      0.89      0.93      2429
           1       0.85      0.95      0.90      1572

    accuracy                           0.91      4001
   macro avg       0.91      0.92      0.91      4001
weighted avg       0.92      0.91      0.91      4001

[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    1.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    1.3s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    1.2s finished
```



Receiver operating characteristic

```
plt.rcdefaults()
fig, ax = plt.subplots()
algorithms = ('Gaussian', 'Decision Tree Classifier','Random Forest Classifier')
y_pos = np.arange(len(algorithms))
x = (score_gau, score_dec, score_rfc) # scores
ax.barh(y_pos, x, align='center')
ax.set_yticks(y_pos)
ax.set_yticklabels(algorithms)
ax.invert_yaxis() # labels read top-to-bottom
ax.set_xlabel('Performance')
ax.set_title('Which one is the best algorithm?')
for i, v in enumerate(x):
    ax.text(v + 1, i, str(v), color='black', va='center', fontweight='normal')
plt.show()
```

```
plt.show()
```



0.61634591

```
results=pd.DataFrame(columns=['score'])
results.loc['Gaussian']=[score_gau]
results.loc['Random Forest Classifier']=[score_rfc]
results.loc['Decision Tree Classifier']=[score_dec]
```

```
results.sort_values('score',ascending=False).style.background_gradient(cmap='Greens',subse
```

	score
Random Forest Classifier	0.913272
Decision Tree Classifier	0.843789
Gaussian	0.616346

```
output.to_csv('output.csv', index=False)
print("Success!")
```

```
Success!
```

```
output.head(10)
```

	Predicted
0	0
1	1
2	0
3	0
4	1
5	1
6	1
9	1