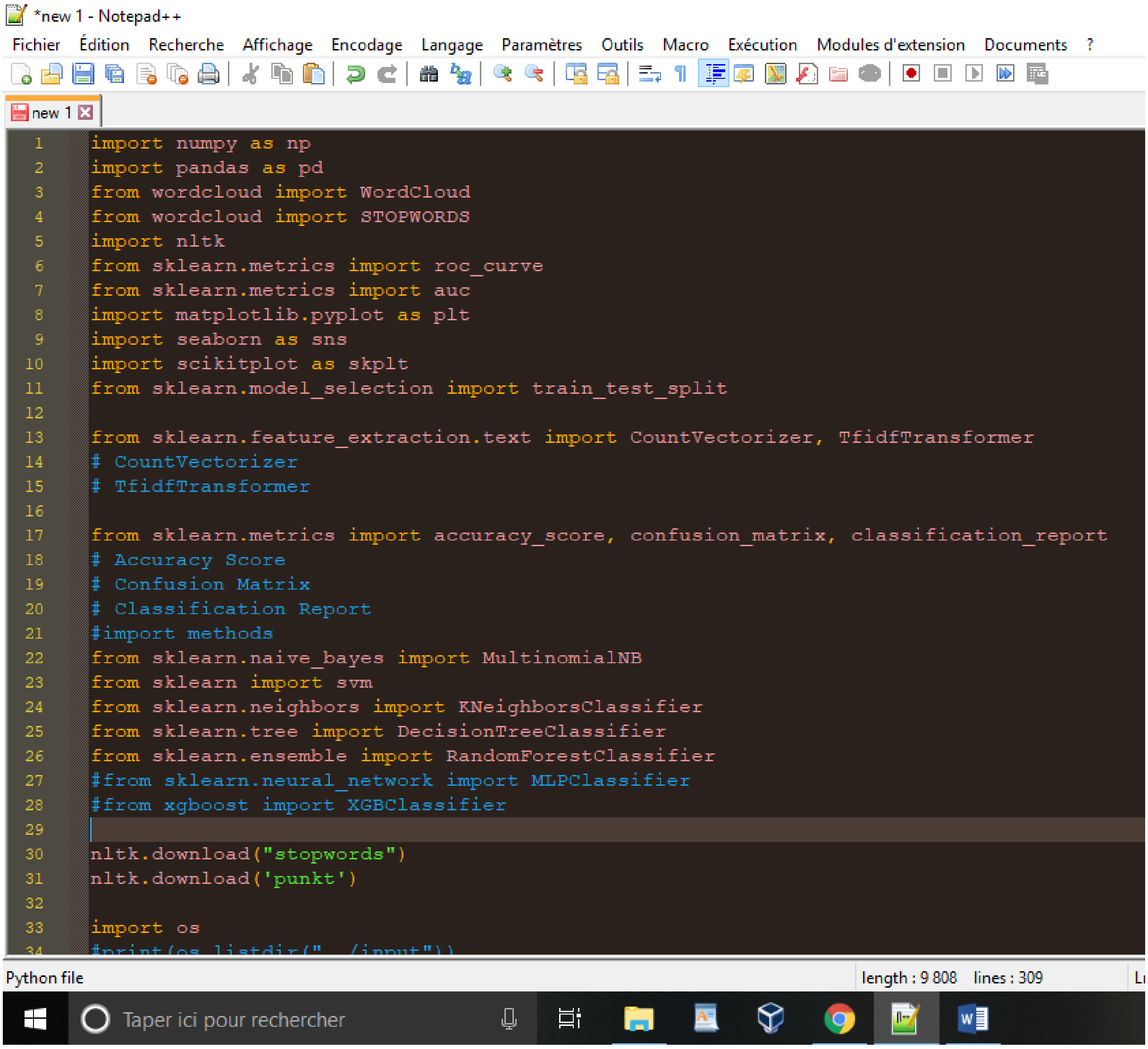
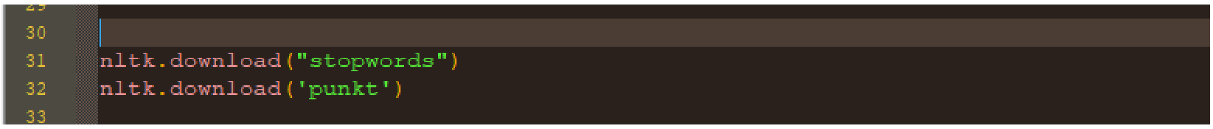
Spam Project of Machine Learning:

Members:

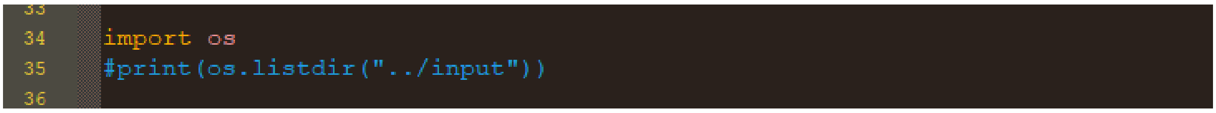
- ***Edna Batana Diane***
- ***Loukrimi Souad***
- ***Zemmouri Kenza***


*new 1 - Notepad++
Fichier Édition Recherche Affichage Encodage Langage Paramètres Outils Macro Exécution Modules d'extension Documents ?
new 1
Python file
length : 9 808 lines : 309
Taper ici pour rechercher

```python
import numpy as np
import pandas as pd
from wordcloud import WordCloud
from wordcloud import STOPWORDS
import nltk
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
import matplotlib.pyplot as plt
import seaborn as sns
import scikitplot as skplt
from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
# CountVectorizer
# TfidfTransformer

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Accuracy Score
# Confusion Matrix
# Classification Report
#import methods
from sklearn.naive_bayes import MultinomialNB
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
#from sklearn.neural_network import MLPClassifier
#from xgboost import XGBClassifier

nltk.download("stopwords")
nltk.download('punkt')

import os
#print(os.listdir("../input"))
```

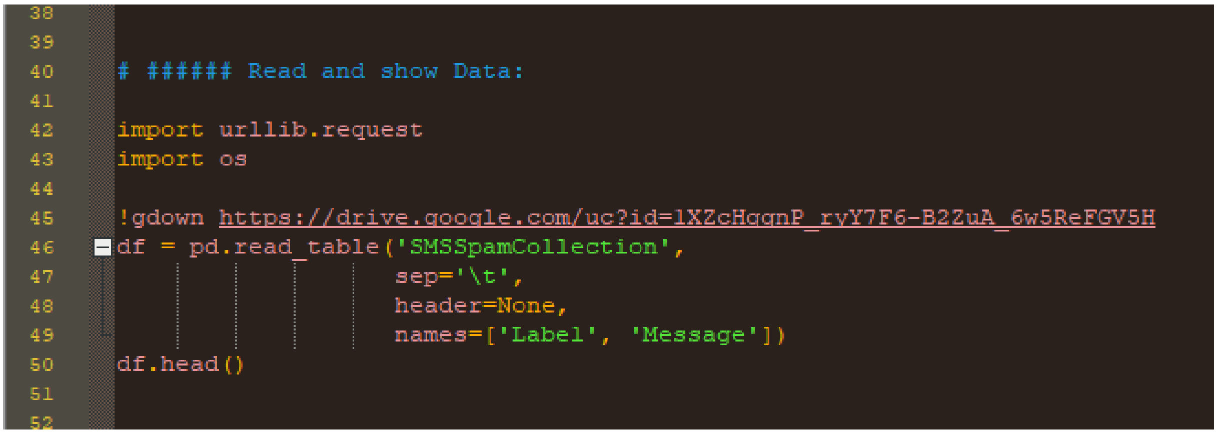


```python

nltk.download("stopwords")
nltk.download('punkt')

```

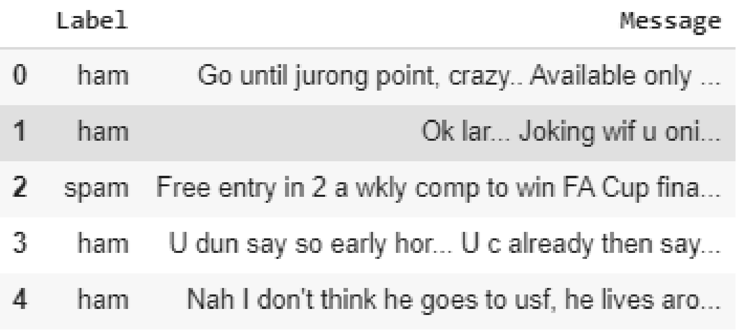


```
import os
#print(os.listdir("../input"))
```

Read and Show Data :

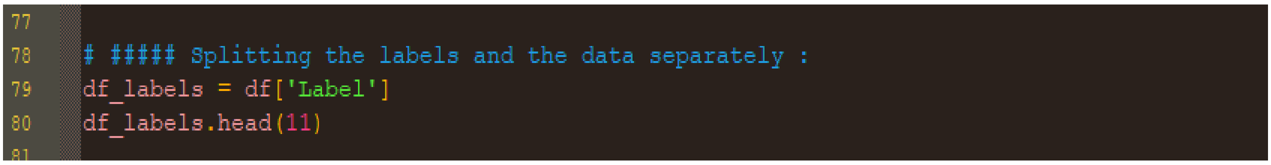


```


# ###### Read and show Data:

import urllib.request
import os

!gdown https://drive.google.com/uc?id=1XZcHqqnP_ryY7F6-B2ZuA_6w5ReFGV5H
df = pd.read_table('SMSSpamCollection',
                   sep='\t',
                   header=None,
                   names=['Label', 'Message'])
df.head()
```



	Label	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...

Splitting the labels and the data separately:

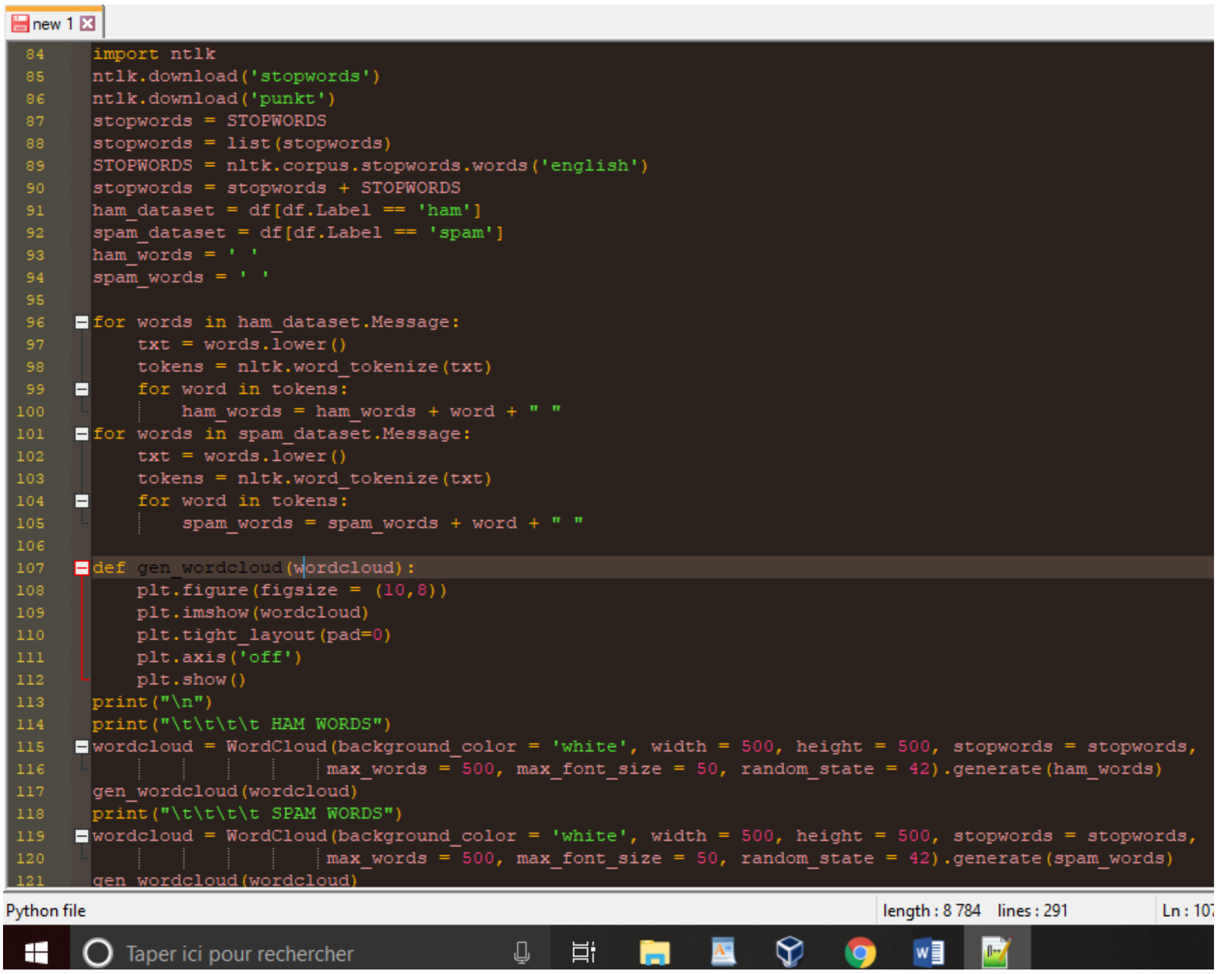


```

# ##### Splitting the labels and the data separately :
df_labels = df['Label']
df_labels.head(11)
```

```
0       ham
1       ham
2      spam
3       ham
4       ham
5      spam
6       ham
7       ham
8      spam
9      spam
10      ham
Name: Label, dtype: object
```

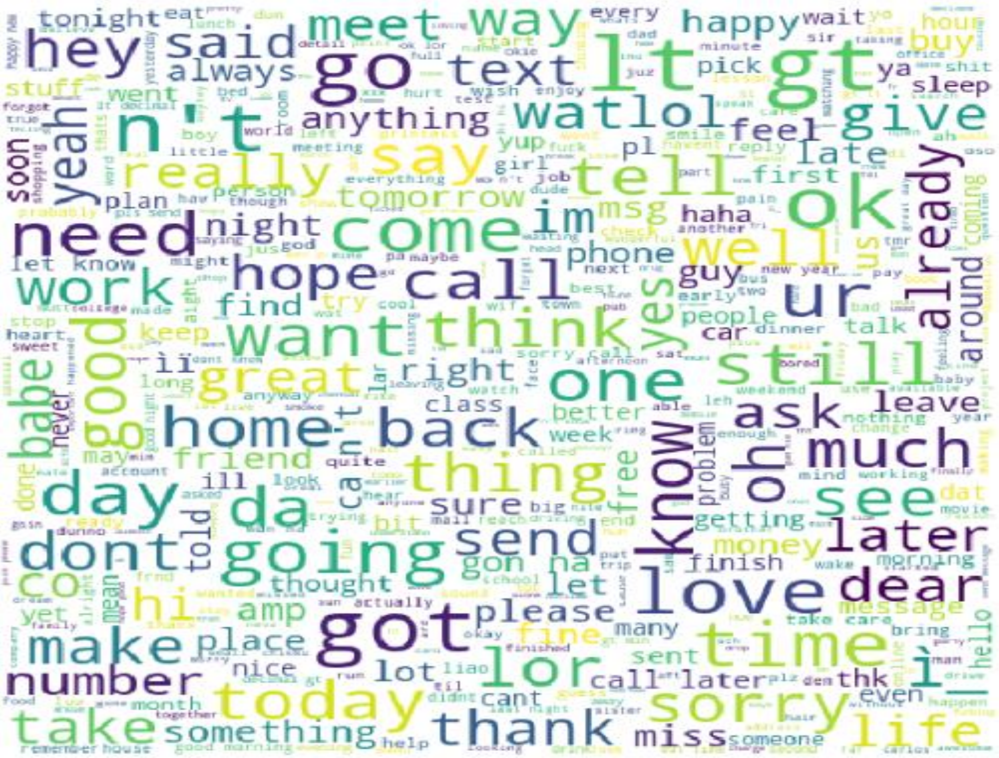
Data Visualization :

- To check the most used word in Ham sms and Spam SMS
- To visualize the percentage of Ham and Spam SMS

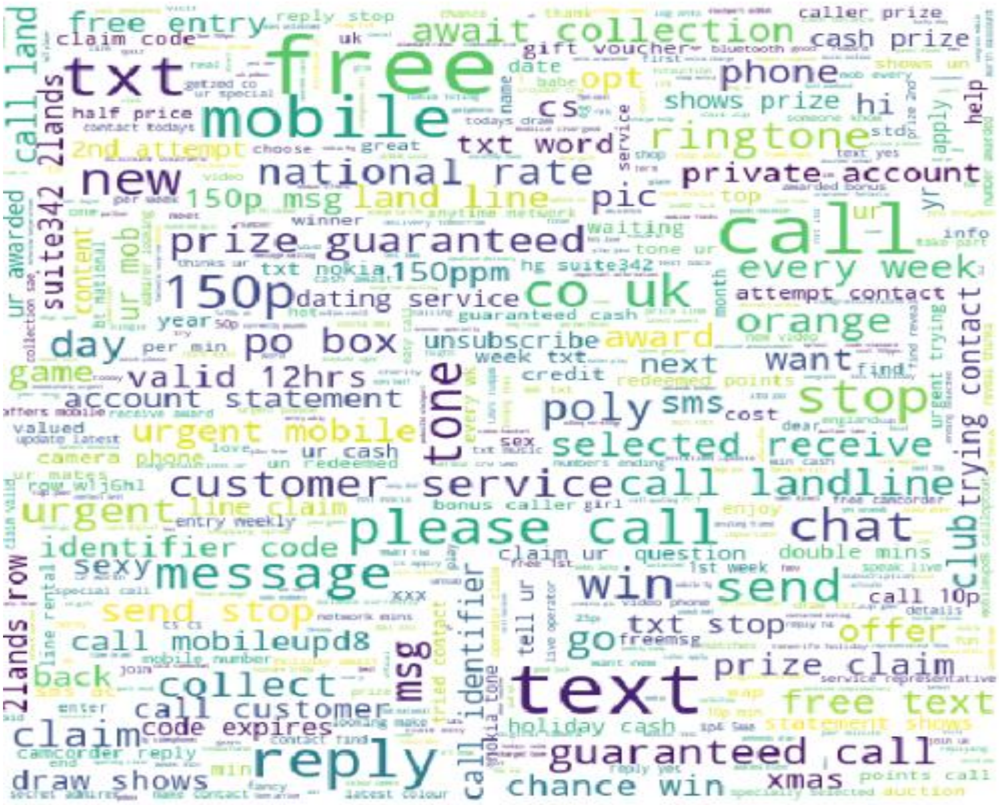


```python
import ntlk
ntlk.download('stopwords')
ntlk.download('punkt')
stopwords = STOPWORDS
stopwords = list(stopwords)
STOPWORDS = nltk.corpus.stopwords.words('english')
stopwords = stopwords + STOPWORDS
ham_dataset = df[df.Label == 'ham']
spam_dataset = df[df.Label == 'spam']
ham_words = ' '
spam_words = ' '

for words in ham_dataset.Message:
    txt = words.lower()
    tokens = nltk.word_tokenize(txt)
    for word in tokens:
        ham_words = ham_words + word + " "
for words in spam_dataset.Message:
    txt = words.lower()
    tokens = nltk.word_tokenize(txt)
    for word in tokens:
        spam_words = spam_words + word + " "

def gen_wordcloud(wordcloud):
    plt.figure(figsize = (10,8))
    plt.imshow(wordcloud)
    plt.tight_layout(pad=0)
    plt.axis('off')
    plt.show()
print("\n")
print("\t\t\t\t HAM WORDS")
wordcloud = WordCloud(background_color = 'white', width = 500, height = 500, stopwords = stopwords,
                    max_words = 500, max_font_size = 50, random_state = 42).generate(ham_words)
gen_wordcloud(wordcloud)
print("\t\t\t\t SPAM WORDS")
wordcloud = WordCloud(background_color = 'white', width = 500, height = 500, stopwords = stopwords,
                    max_words = 500, max_font_size = 50, random_state = 42).generate(spam_words)
gen_wordcloud(wordcloud)
```

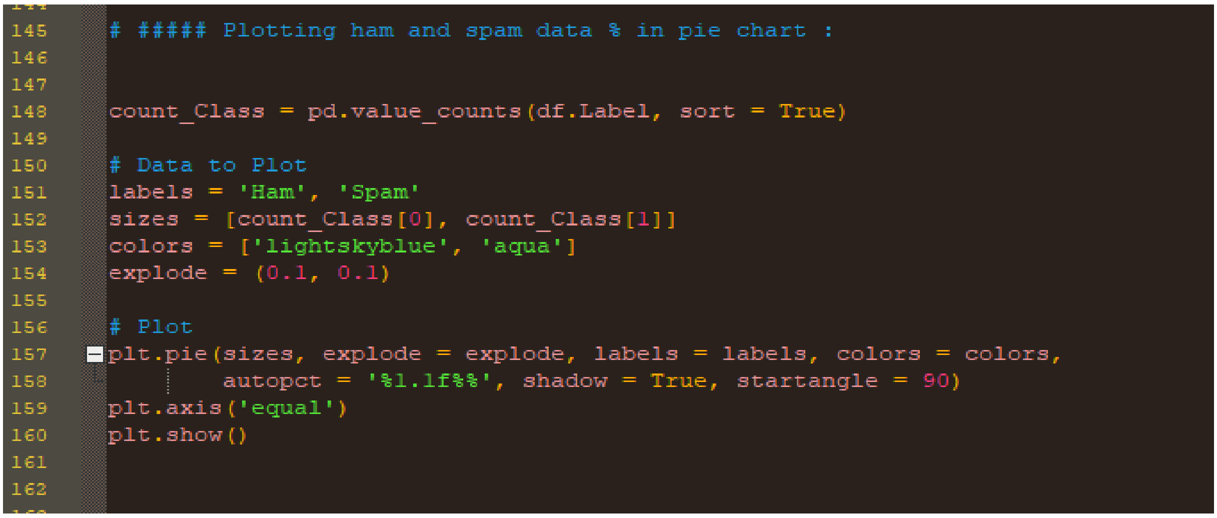
HAM WORDS



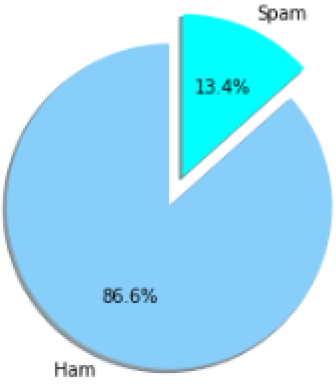
SPAM WORDS



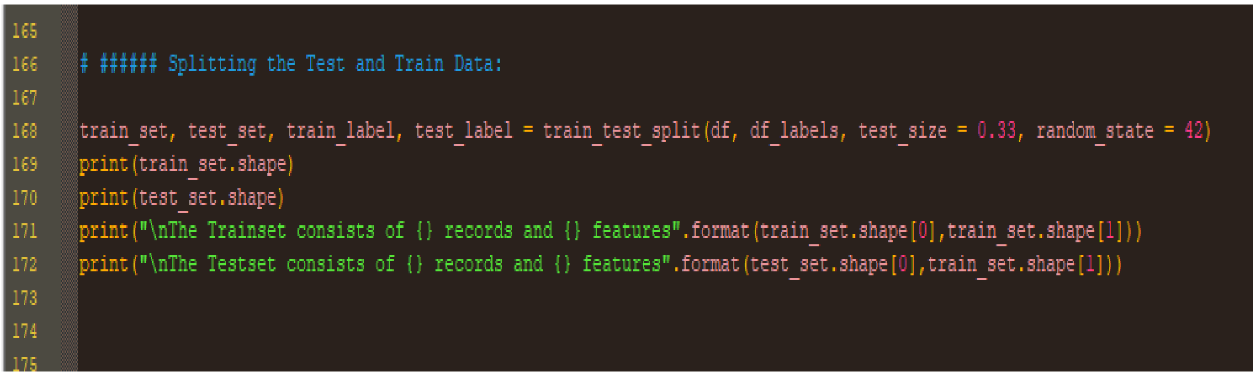
Plotting ham and spam data % in pie chart:



```
# ##### Plotting ham and spam data % in pie chart :


count_Class = pd.value_counts(df.Label, sort = True)

# Data to Plot
labels = 'Ham', 'Spam'
sizes = [count_Class[0], count_Class[1]]
colors = ['lightskyblue', 'aqua']
explode = (0.1, 0.1)

# Plot
plt.pie(sizes, explode = explode, labels = labels, colors = colors,
        autopct = '%1.1f%%', shadow = True, startangle = 90)
plt.axis('equal')
plt.show()
```



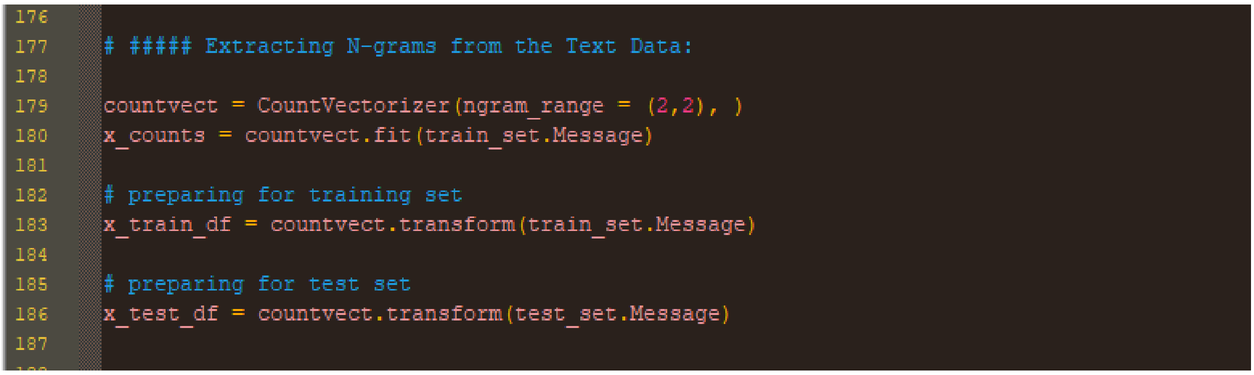
Splitting the Test and Train Data:



```
# ###### Splitting the Test and Train Data:

train_set, test_set, train_label, test_label = train_test_split(df, df_labels, test_size = 0.33, random_state = 42)
print(train_set.shape)
print(test_set.shape)
print("\nThe Trainset consists of {} records and {} features".format(train_set.shape[0],train_set.shape[1]))
print("\nThe Testset consists of {} records and {} features".format(test_set.shape[0],train_set.shape[1]))
```

```
(3733, 2)
(1839, 2)

The Trainset consists of 3733 records and 2 features

The Testset consists of 1839 records and 2 features
```

Extracting N-grams from the Text Data:



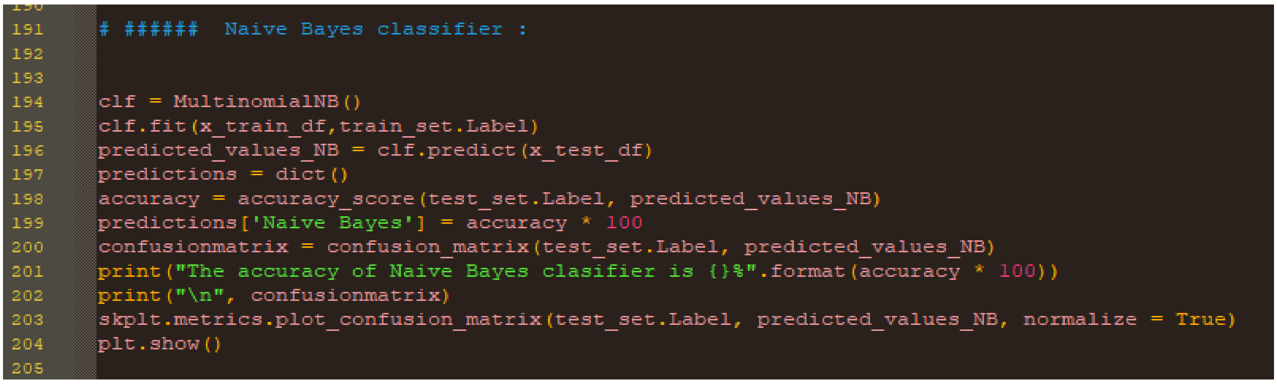
```python
# ##### Extracting N-grams from the Text Data:

countvect = CountVectorizer(ngram_range = (2,2), )
x_counts = countvect.fit(train_set.Message)

# preparing for training set
x_train_df = countvect.transform(train_set.Message)

# preparing for test set
x_test_df = countvect.transform(test_set.Message)
```

Data Model:

The Algorithms used below are:

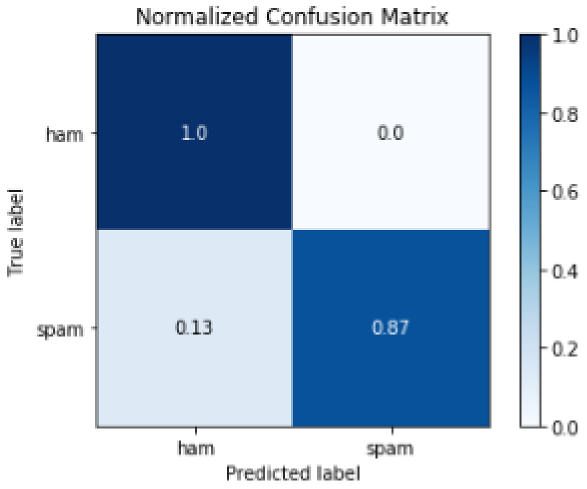
- Naive Bayes
- K-Nearest
- Decision Tree
- Support Vector Machine
- Random Forest

Naive Bayes classifier :

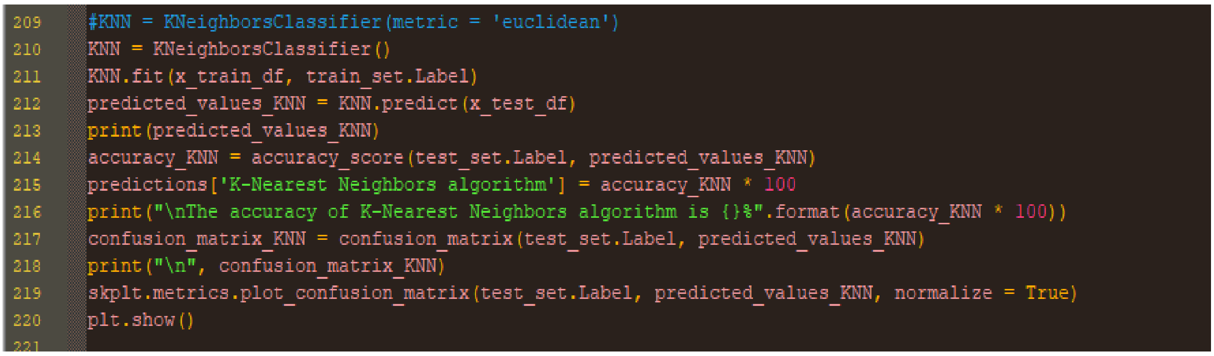


```python
# ######  Naive Bayes classifier :


clf = MultinomialNB()
clf.fit(x_train_df,train_set.Label)
predicted_values_NB = clf.predict(x_test_df)
predictions = dict()
accuracy = accuracy_score(test_set.Label, predicted_values_NB)
predictions['Naive Bayes'] = accuracy * 100
confusionmatrix = confusion_matrix(test_set.Label, predicted_values_NB)
print("The accuracy of Naive Bayes clasifier is {}%".format(accuracy * 100))
print("\n", confusionmatrix)
skplt.metrics.plot_confusion_matrix(test_set.Label, predicted_values_NB, normalize = True)
plt.show()
```

```
The accuracy of Naive Bayes clasifier is 97.87928221859707%

 [[1581     6]
 [33       219]]
```

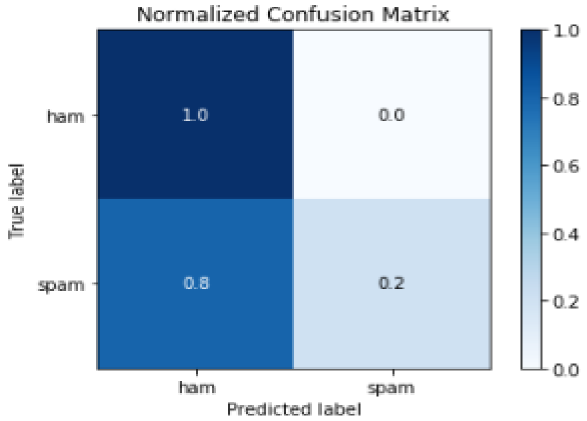


K-Nearest Neighbors algorithm :

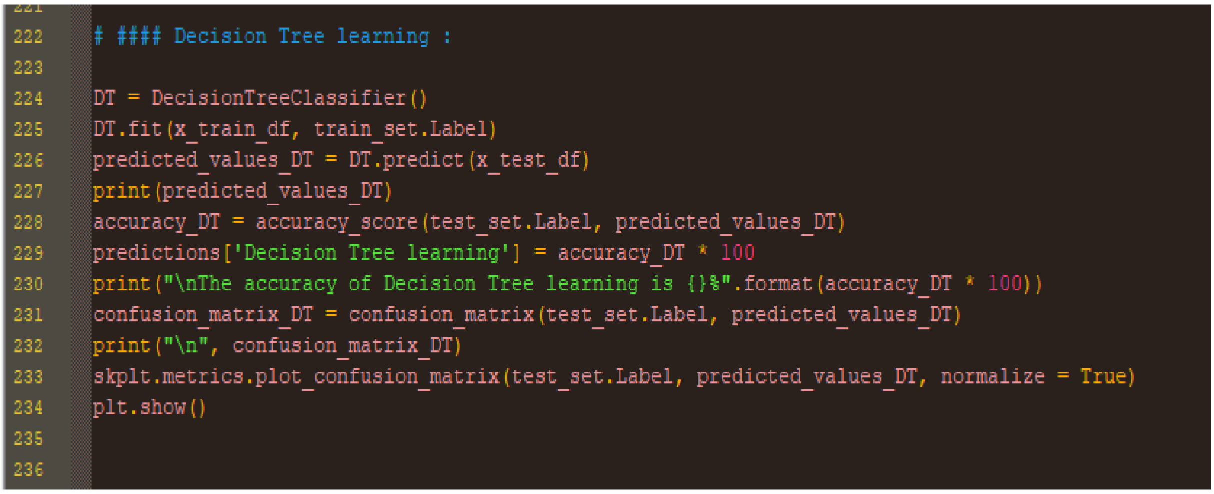


```
#KNN = KNeighborsClassifier(metric = 'euclidean')
KNN = KNeighborsClassifier()
KNN.fit(x_train_df, train_set.Label)
predicted_values_KNN = KNN.predict(x_test_df)
print(predicted_values_KNN)
accuracy_KNN = accuracy_score(test_set.Label, predicted_values_KNN)
predictions['K-Nearest Neighbors algorithm'] = accuracy_KNN * 100
print("\nThe accuracy of K-Nearest Neighbors algorithm is {}%".format(accuracy_KNN * 100))
confusion_matrix_KNN = confusion_matrix(test_set.Label, predicted_values_KNN)
print("\n", confusion_matrix_KNN)
skplt.metrics.plot_confusion_matrix(test_set.Label, predicted_values_KNN, normalize = True)
plt.show()
```

```
['ham' 'ham' 'ham' ... 'ham' 'ham' 'ham']

The accuracy of K-Nearest Neighbors algorithm is 89.07014681892332%

 [[1587    0]
 [201    51]]
```

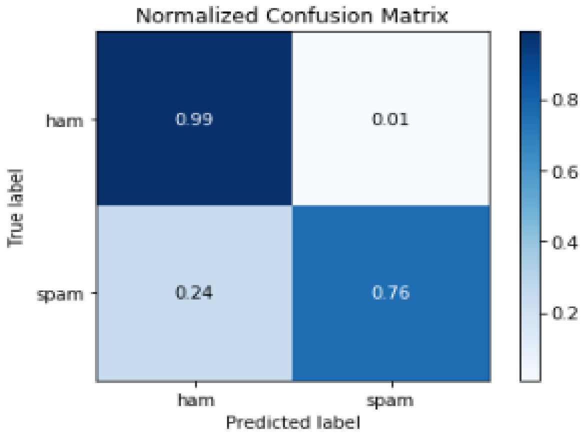


Decision Tree learning :

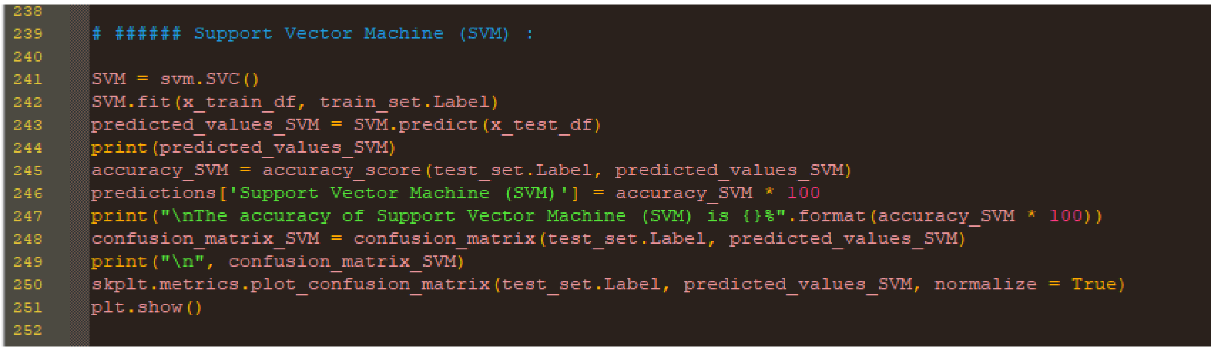


```python
# #### Decision Tree learning :

DT = DecisionTreeClassifier()
DT.fit(x_train_df, train_set.Label)
predicted_values_DT = DT.predict(x_test_df)
print(predicted_values_DT)
accuracy_DT = accuracy_score(test_set.Label, predicted_values_DT)
predictions['Decision Tree learning'] = accuracy_DT * 100
print("\nThe accuracy of Decision Tree learning is {}%".format(accuracy_DT * 100))
confusion_matrix_DT = confusion_matrix(test_set.Label, predicted_values_DT)
print("\n", confusion_matrix_DT)
skplt.metrics.plot_confusion_matrix(test_set.Label, predicted_values_DT, normalize = True)
plt.show()
```

```
['ham' 'ham' 'spam' ... 'ham' 'ham' 'spam'] The accuracy of Decision
Tree learning is 96.08482871125612%
 [[1576 11]
 [ 61 191]]
```



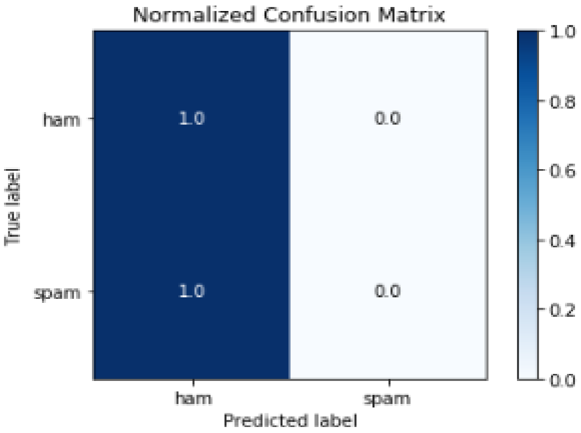
Support Vector Machine (SVM) :



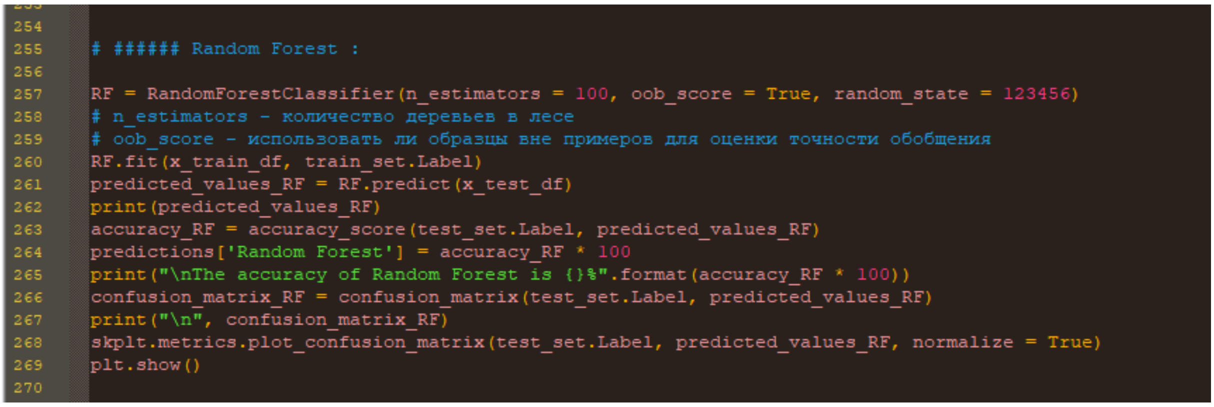
```python
# ###### Support Vector Machine (SVM) :

SVM = svm.SVC()
SVM.fit(x_train_df, train_set.Label)
predicted_values_SVM = SVM.predict(x_test_df)
print(predicted_values_SVM)
accuracy_SVM = accuracy_score(test_set.Label, predicted_values_SVM)
predictions['Support Vector Machine (SVM)'] = accuracy_SVM * 100
print("\nThe accuracy of Support Vector Machine (SVM) is {}%".format(accuracy_SVM * 100))
confusion_matrix_SVM = confusion_matrix(test_set.Label, predicted_values_SVM)
print("\n", confusion_matrix_SVM)
skplt.metrics.plot_confusion_matrix(test_set.Label, predicted_values_SVM, normalize = True)
plt.show()
```

```
['ham' 'ham' 'ham' ... 'ham' 'ham' 'ham']
```

```
The accuracy of Support Vector Machine (SVM) is 86.2969004893964%

 [[1587    0]
 [252      0]]
```

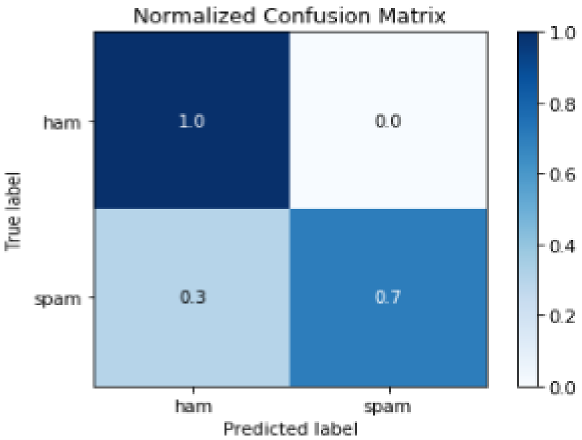


Random Forest:

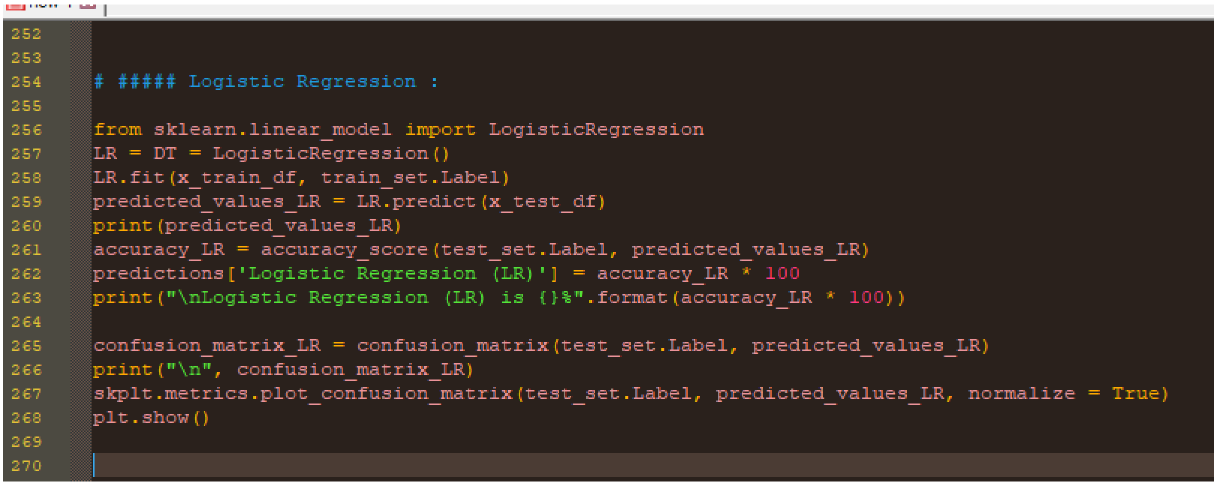


```
# ###### Random Forest :

RF = RandomForestClassifier(n_estimators = 100, oob_score = True, random_state = 123456)
# n_estimators - количество деревьев в лесе
# oob_score - использовать ли образцы вне примеров для оценки точности обобщения
RF.fit(x_train_df, train_set.Label)
predicted_values_RF = RF.predict(x_test_df)
print(predicted_values_RF)
accuracy_RF = accuracy_score(test_set.Label, predicted_values_RF)
predictions['Random Forest'] = accuracy_RF * 100
print("\nThe accuracy of Random Forest is {}%".format(accuracy_RF * 100))
confusion_matrix_RF = confusion_matrix(test_set.Label, predicted_values_RF)
print("\n", confusion_matrix_RF)
skplt.metrics.plot_confusion_matrix(test_set.Label, predicted_values_RF, normalize = True)
plt.show()
```

```
['ham' 'ham' 'ham' ... 'ham' 'ham' 'ham'] The accuracy of Random Forest
is 95.86731919521479%
[[1587 0]
[76  176]]
```

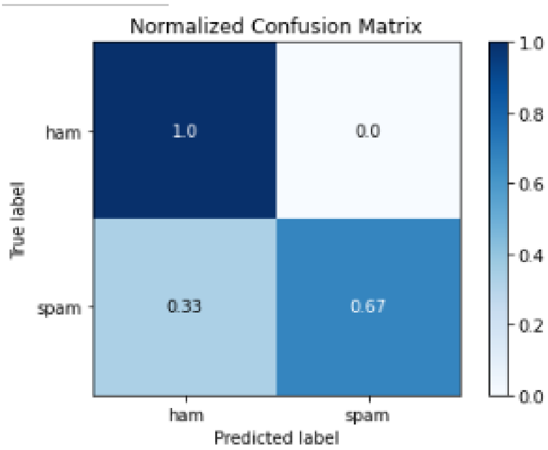


Logistic Rgression:

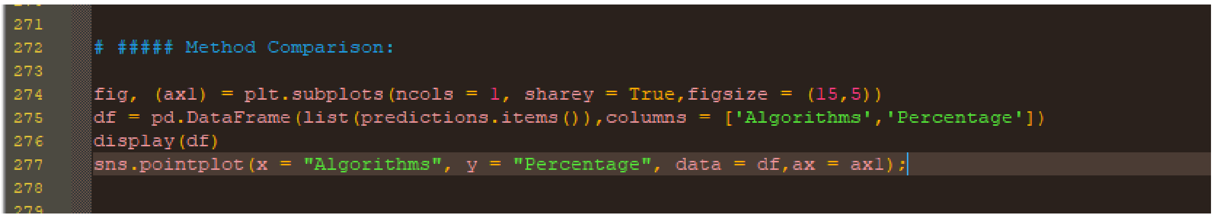


```
# ##### Logistic Regression :

from sklearn.linear_model import LogisticRegression
LR = DT = LogisticRegression()
LR.fit(x_train_df, train_set.Label)
predicted_values_LR = LR.predict(x_test_df)
print(predicted_values_LR)
accuracy_LR = accuracy_score(test_set.Label, predicted_values_LR)
predictions['Logistic Regression (LR)'] = accuracy_LR * 100
print("\nLogistic Regression (LR) is {}%".format(accuracy_LR * 100))

confusion_matrix_LR = confusion_matrix(test_set.Label, predicted_values_LR)
print("\n", confusion_matrix_LR)
skplt.metrics.plot_confusion_matrix(test_set.Label, predicted_values_LR, normalize = True)
plt.show()

```

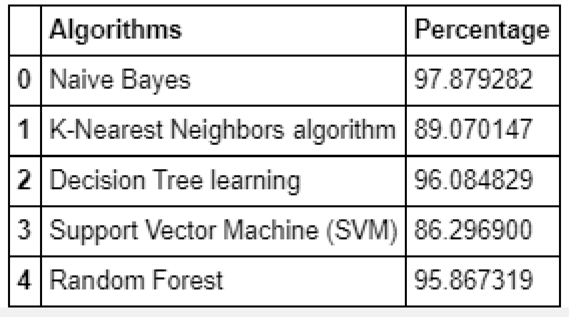
```
['ham' 'ham' 'ham' ... 'ham' 'ham' 'ham']

Logistic Regression (LR) is 95.64980967917346%

 [[1593    0]

 [80  166]]
```



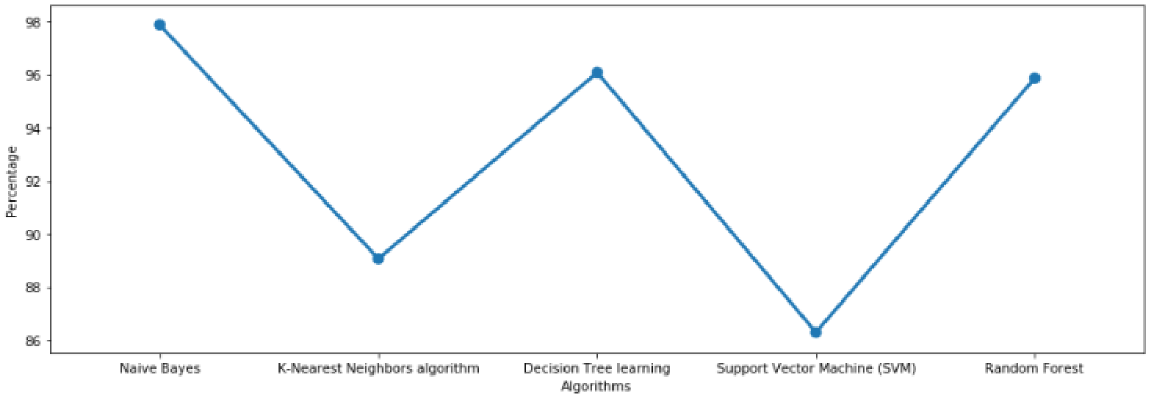
Méthodes Comparison :



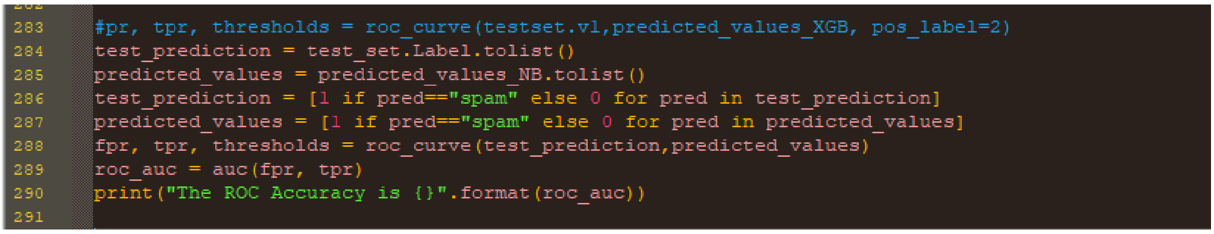
```

# ##### Method Comparison:

fig, (ax1) = plt.subplots(ncols = 1, sharey = True,figsize = (15,5))
df = pd.DataFrame(list(predictions.items()),columns = ['Algorithms','Percentage'])
display(df)
sns.pointplot(x = "Algorithms", y = "Percentage", data = df,ax = ax1);

```



	Algorithms	Percentage
0	Naive Bayes	97.879282
1	K-Nearest Neighbors algorithm	89.070147
2	Decision Tree learning	96.084829
3	Support Vector Machine (SVM)	86.296900
4	Random Forest	95.867319

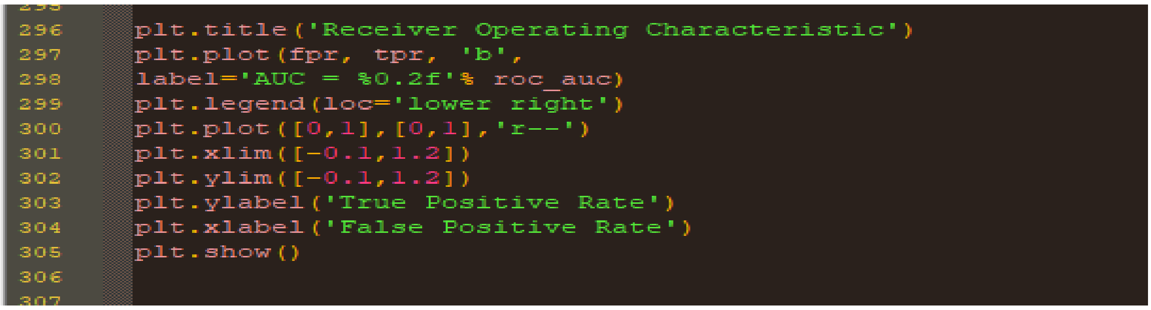


ROC Accuracy :

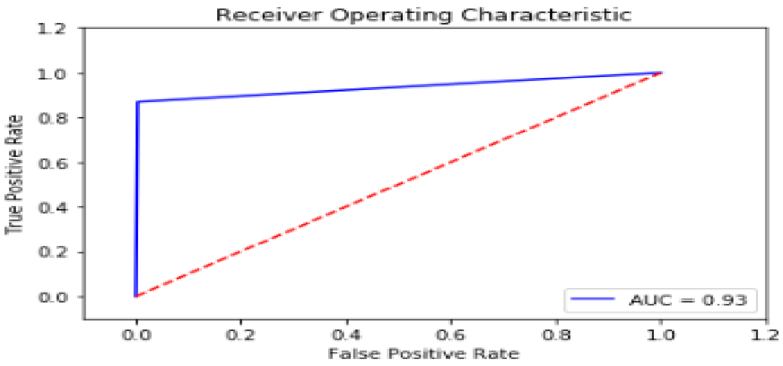


```
#pr, tpr, thresholds = roc_curve(testset.v1,predicted_values_XGB, pos_label=2)
test_prediction = test_set.Label.tolist()
predicted_values = predicted_values_NB.tolist()
test_prediction = [1 if pred=="spam" else 0 for pred in test_prediction]
predicted_values = [1 if pred=="spam" else 0 for pred in predicted_values]
fpr, tpr, thresholds = roc_curve(test_prediction,predicted_values)
roc_auc = auc(fpr, tpr)
print("The ROC Accuracy is {}".format(roc_auc))
```

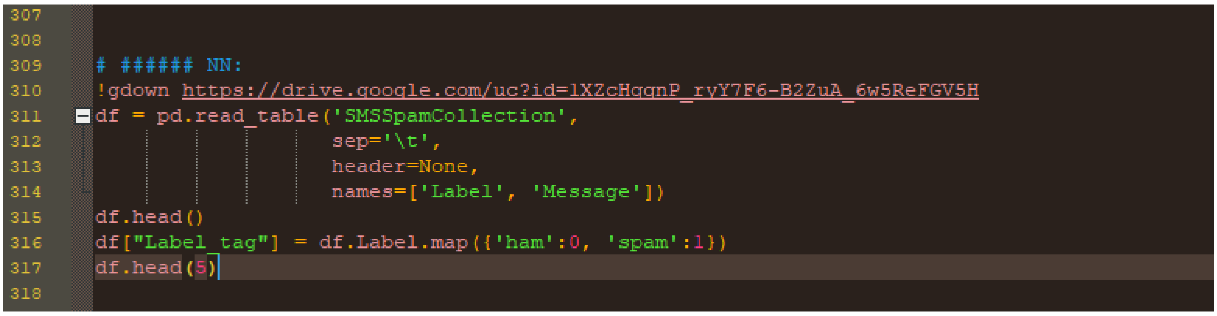
```
The ROC Accuracy is 0.9326334503555676
```



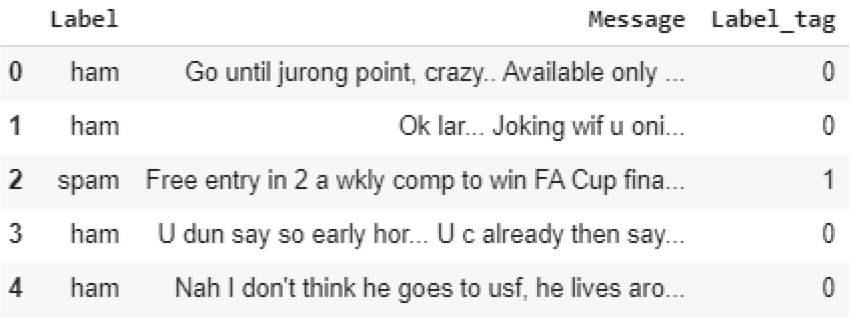
```
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b',
label='AUC = %0.2f'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.2])
plt.ylim([-0.1,1.2])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



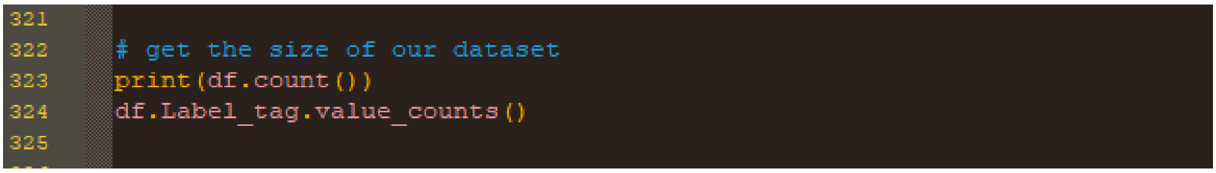
Neural Network :



```
# ###### NN:
!gdown https://drive.google.com/uc?id=1XZcHqqnP_ryY7F6-B2ZuA_6w5ReFGV5H
df = pd.read_table('SMSSpamCollection',
                   sep='\t',
                   header=None,
                   names=['Label', 'Message'])
df.head()
df["Label_tag"] = df.Label.map({'ham':0, 'spam':1})
df.head(5)
```

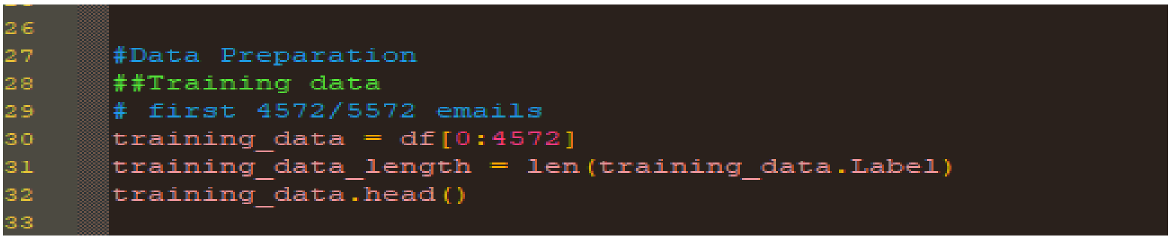


	Label	Message	Label_tag
0	ham	Go until jurong point, crazy.. Available only ...	0
1	ham	Ok lar... Joking wif u oni...	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	1
3	ham	U dun say so early hor... U c already then say...	0
4	ham	Nah I don't think he goes to usf, he lives aro...	0

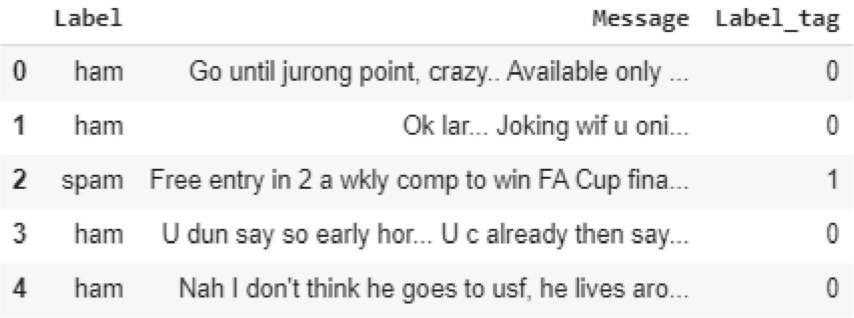


```
# get the size of our dataset
print(df.count())
df.Label_tag.value_counts()
```

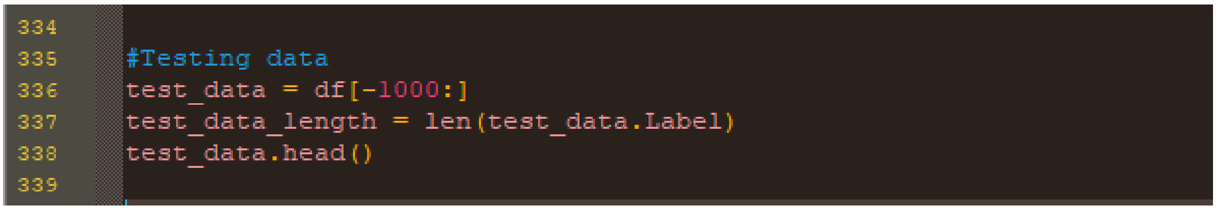
```
Label        5572
Message      5572
Label_tag    5572
dtype: int64
0    4825
1     747
Name: Label_tag, dtype: int64
```



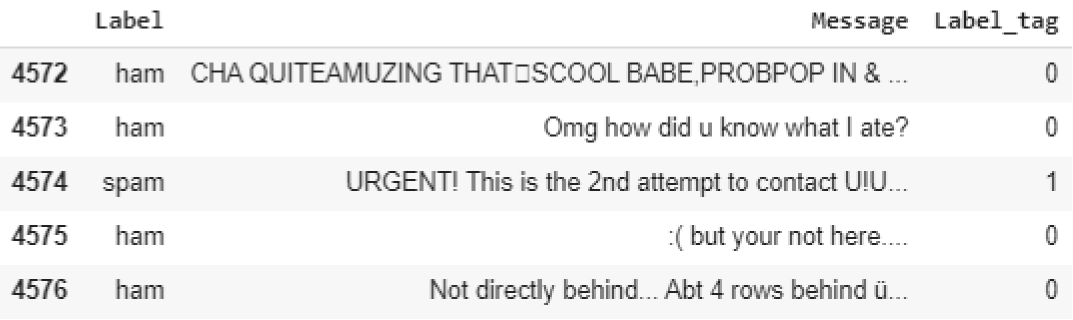
```
#Data Preparation
##Training data
# first 4572/5572 emails
training_data = df[0:4572]
training_data_length = len(training_data.Label)
training_data.head()
```



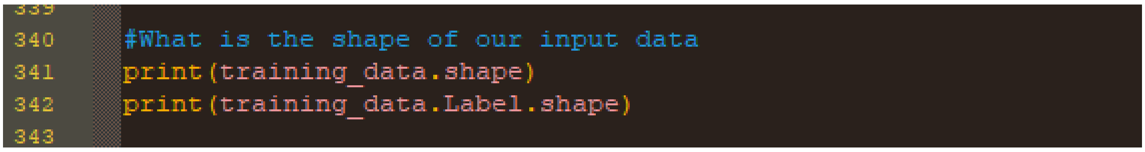
	Label	Message	Label_tag
0	ham	Go until jurong point, crazy.. Available only ...	0
1	ham	Ok lar... Joking wif u oni...	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	1
3	ham	U dun say so early hor... U c already then say...	0
4	ham	Nah I don't think he goes to usf, he lives aro...	0



```
#Testing data
test_data = df[-1000:]
test_data_length = len(test_data.Label)
test_data.head()
```

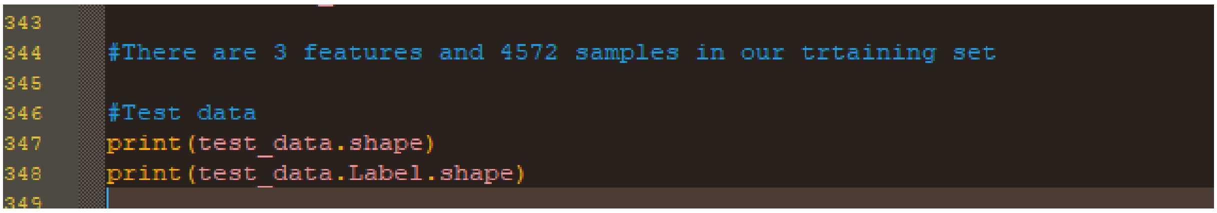


	Label	Message	Label_tag
4572	ham	CHA QUITEAMUZING THAT□SCOOL BABE,PROBPOP IN & ...	0
4573	ham	Omg how did u know what I ate?	0
4574	spam	URGENT! This is the 2nd attempt to contact U!U...	1
4575	ham	:(but your not here....	0
4576	ham	Not directly behind... Abt 4 rows behind ü...	0



```
#What is the shape of our input data
print(training_data.shape)
print(training_data.Label.shape)
```

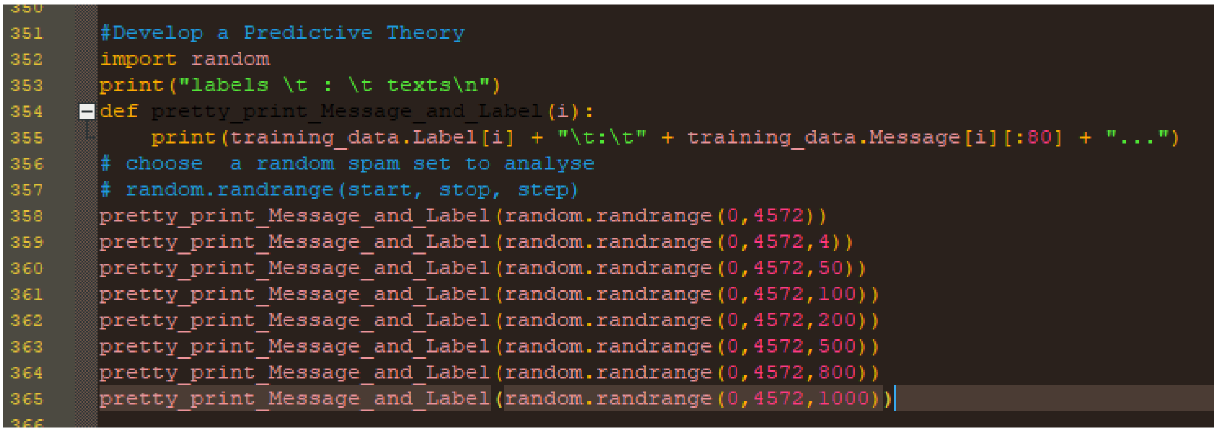
```
(4572, 3)
(4572,)
```



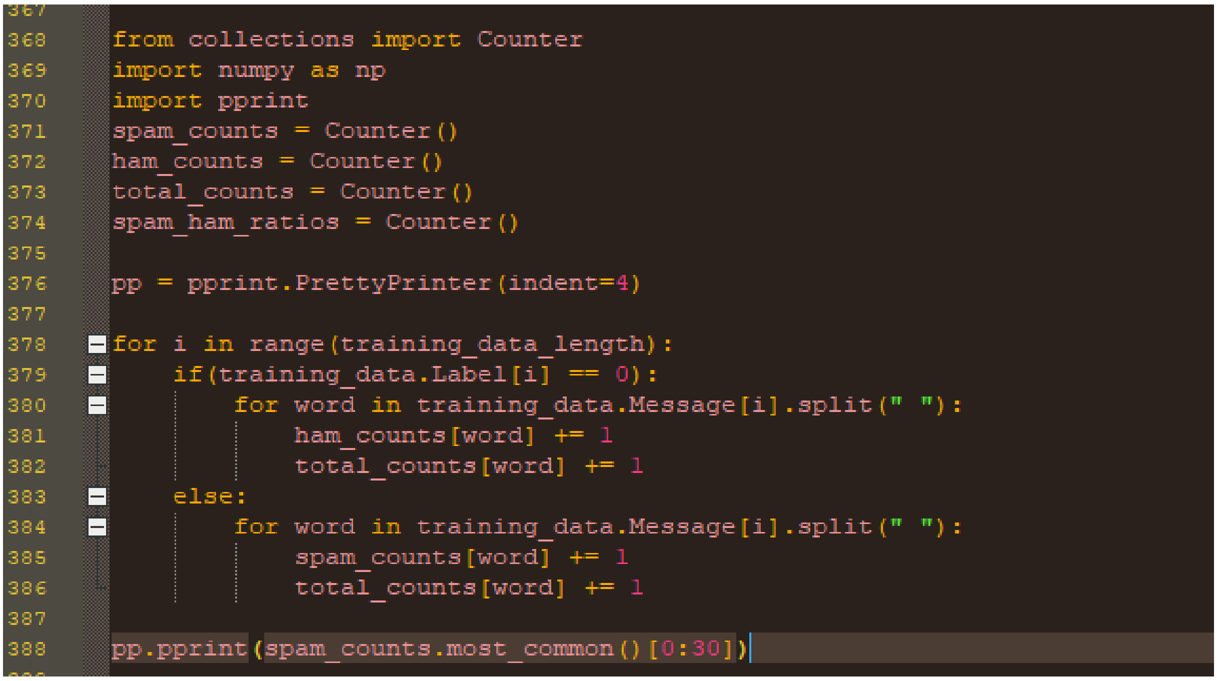
```
#There are 3 features and 4572 samples in our trtaining set

#Test data
print(test_data.shape)
print(test_data.Label.shape)
```

```
(1000, 3)
```

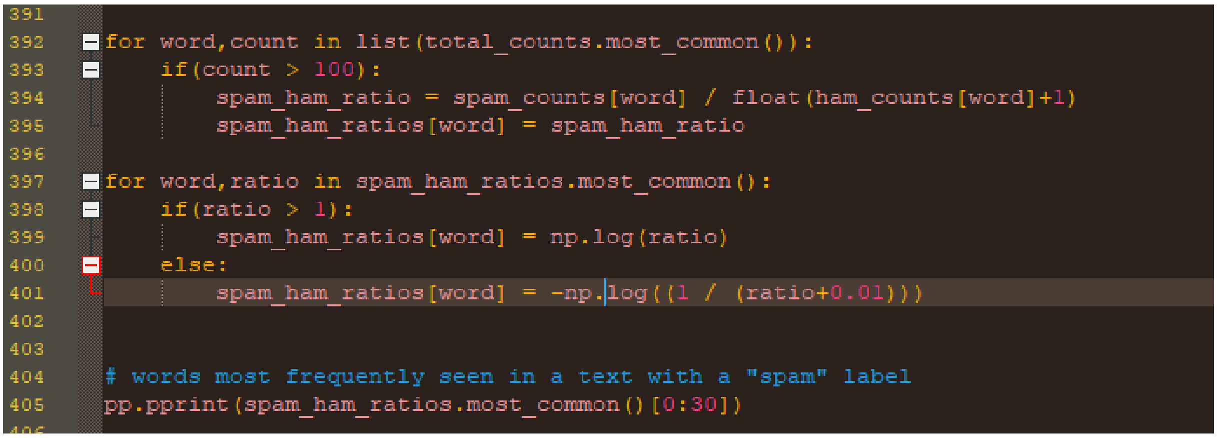
```
(1000,)
```



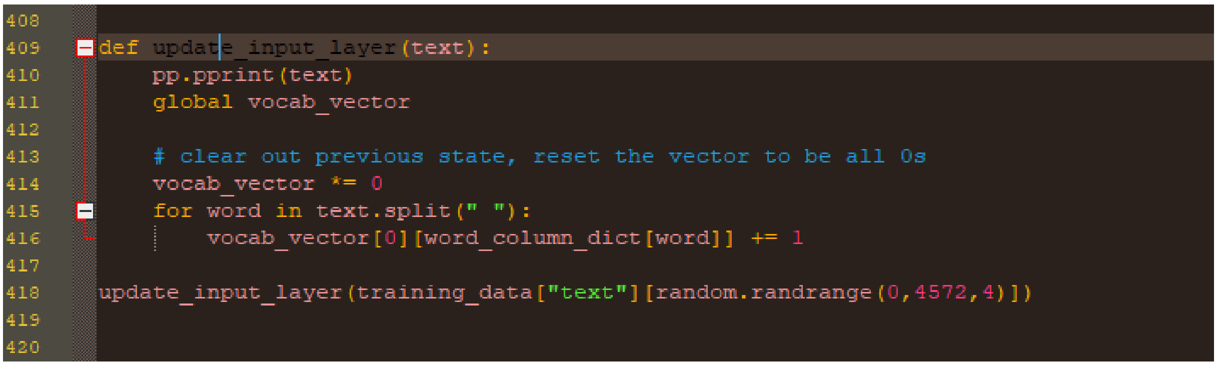
```python
#Develop a Predictive Theory
import random
print("labels \t : \t texts\n")
def pretty_print_Message_and_Label(i):
    print(training_data.Label[i] + "\t:\t" + training_data.Message[i][:80] + "...")
# choose  a random spam set to analyse
# random.randrange(start, stop, step)
pretty_print_Message_and_Label(random.randrange(0,4572))
pretty_print_Message_and_Label(random.randrange(0,4572,4))
pretty_print_Message_and_Label(random.randrange(0,4572,50))
pretty_print_Message_and_Label(random.randrange(0,4572,100))
pretty_print_Message_and_Label(random.randrange(0,4572,200))
pretty_print_Message_and_Label(random.randrange(0,4572,500))
pretty_print_Message_and_Label(random.randrange(0,4572,800))
pretty_print_Message_and_Label(random.randrange(0,4572,1000))
```



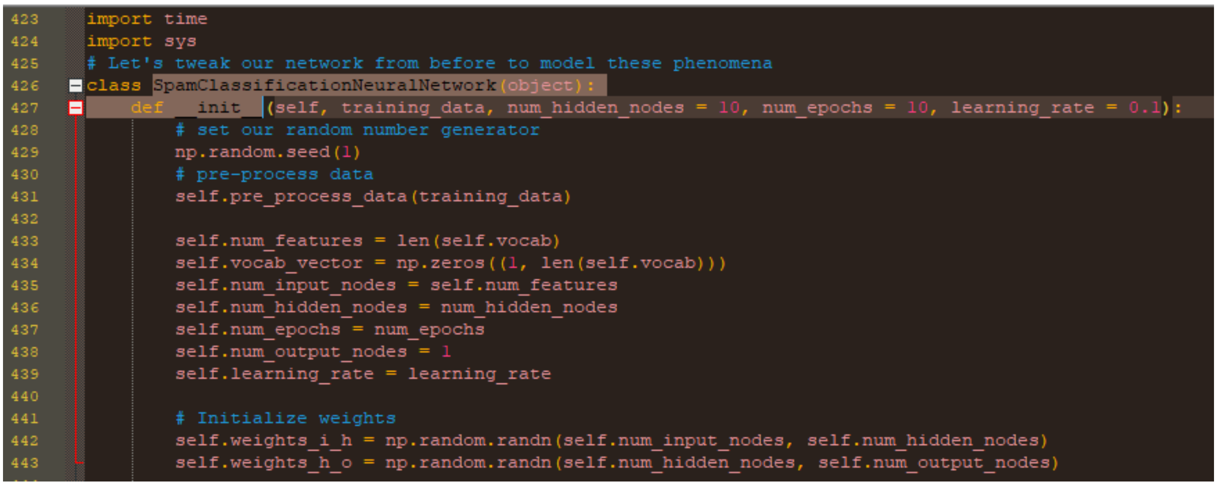
```python
from collections import Counter
import numpy as np
import pprint
spam_counts = Counter()
ham_counts = Counter()
total_counts = Counter()
spam_ham_ratios = Counter()

pp = pprint.PrettyPrinter(indent=4)

for i in range(training_data_length):
    if(training_data.Label[i] == 0):
        for word in training_data.Message[i].split(" "):
            ham_counts[word] += 1
            total_counts[word] += 1
    else:
        for word in training_data.Message[i].split(" "):
            spam_counts[word] += 1
            total_counts[word] += 1

pp.pprint(spam_counts.most_common()[0:30])
```



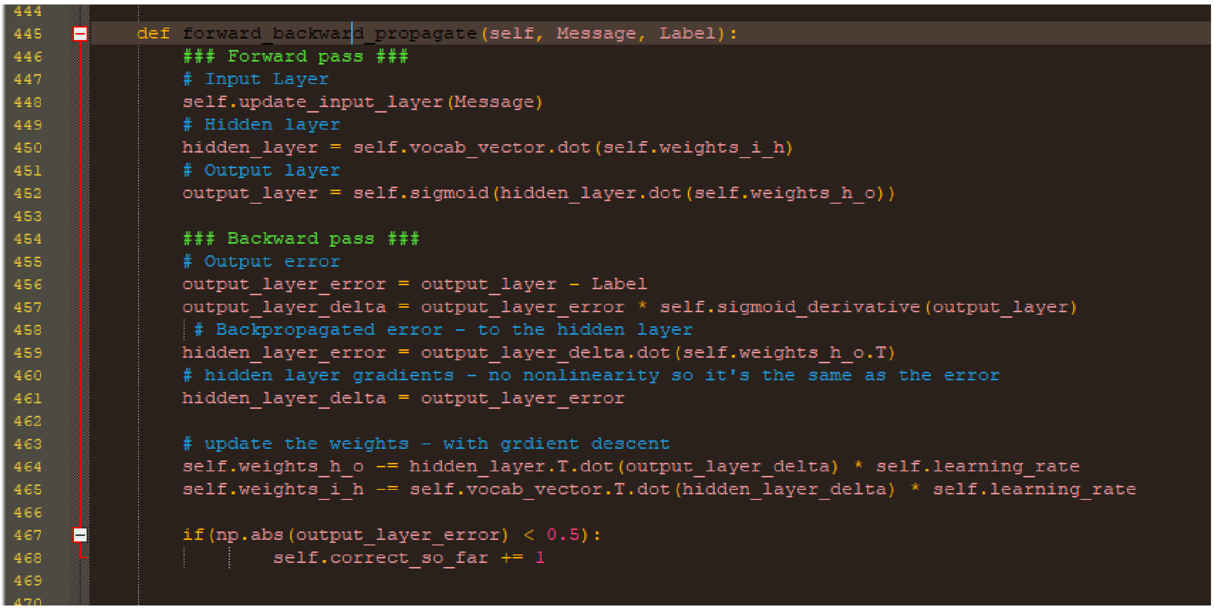
```python
for word,count in list(total_counts.most_common()):
    if(count > 100):
        spam_ham_ratio = spam_counts[word] / float(ham_counts[word]+1)
        spam_ham_ratios[word] = spam_ham_ratio

for word,ratio in spam_ham_ratios.most_common():
    if(ratio > 1):
        spam_ham_ratios[word] = np.log(ratio)
    else:
        spam_ham_ratios[word] = -np.log((1 / (ratio+0.01)))


# words most frequently seen in a text with a "spam" label
pp.pprint(spam_ham_ratios.most_common()[0:30])
```



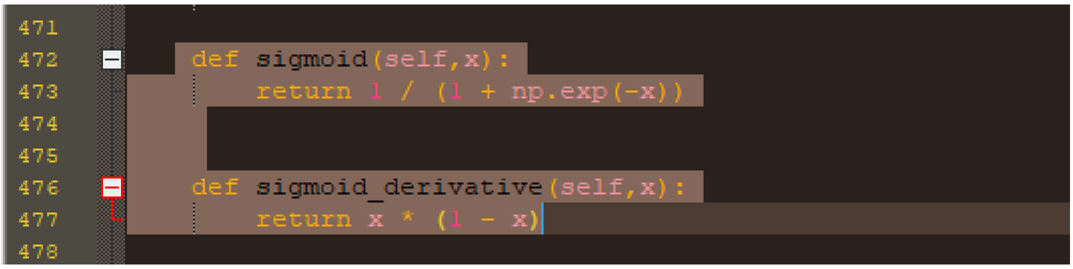
```python
def update_input_layer(text):
    pp.pprint(text)
    global vocab_vector

    # clear out previous state, reset the vector to be all 0s
    vocab_vector *= 0
    for word in text.split(" "):
        vocab_vector[0][word_column_dict[word]] += 1

update_input_layer(training_data["text"][random.randrange(0,4572,4)])

```



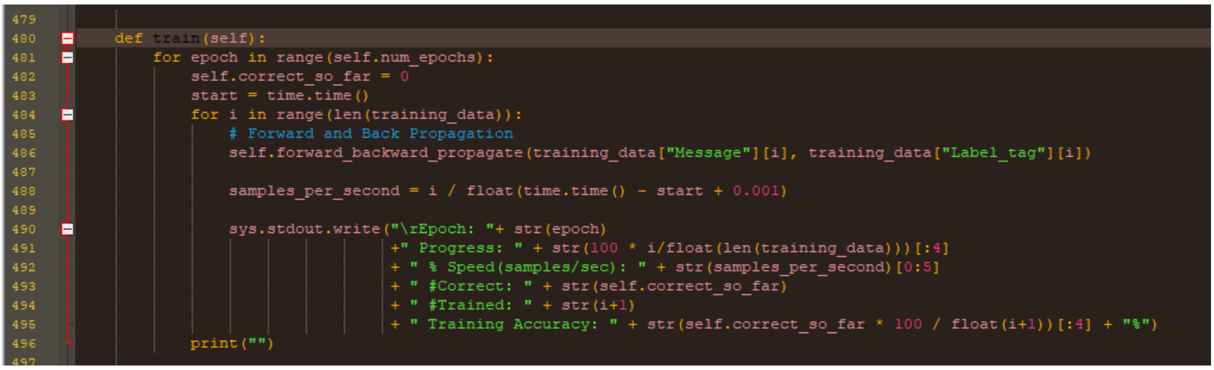
```python
import time
import sys
# Let's tweak our network from before to model these phenomena
class SpamClassificationNeuralNetwork(object):
    def __init__(self, training_data, num_hidden_nodes = 10, num_epochs = 10, learning_rate = 0.1):
        # set our random number generator
        np.random.seed(1)
        # pre-process data
        self.pre_process_data(training_data)

        self.num_features = len(self.vocab)
        self.vocab_vector = np.zeros((1, len(self.vocab)))
        self.num_input_nodes = self.num_features
        self.num_hidden_nodes = num_hidden_nodes
        self.num_epochs = num_epochs
        self.num_output_nodes = 1
        self.learning_rate = learning_rate

        # Initialize weights
        self.weights_i_h = np.random.randn(self.num_input_nodes, self.num_hidden_nodes)
        self.weights_h_o = np.random.randn(self.num_hidden_nodes, self.num_output_nodes)
```



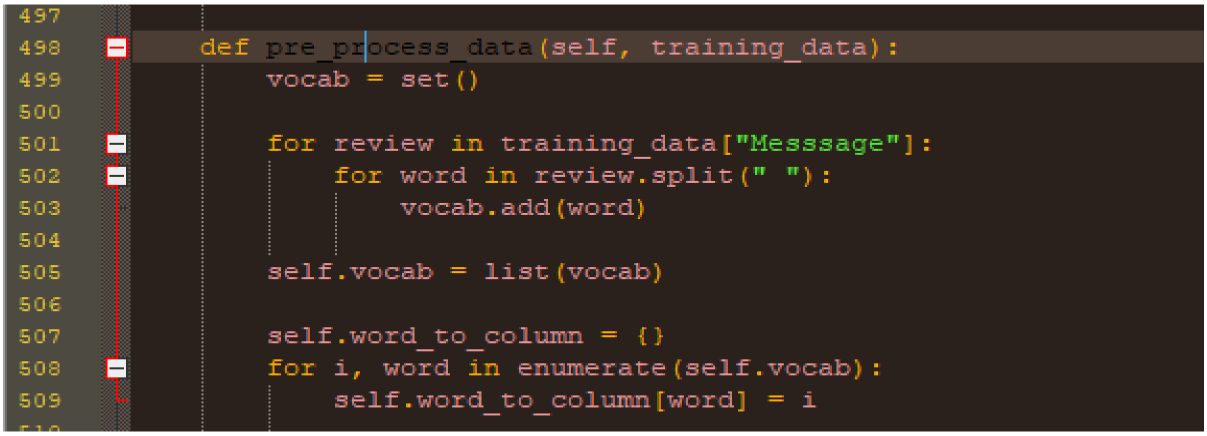
```python

    def forward_backward_propagate(self, Message, Label):
        ### Forward pass ###
        # Input Layer
        self.update_input_layer(Message)
        # Hidden layer
        hidden_layer = self.vocab_vector.dot(self.weights_i_h)
        # Output layer
        output_layer = self.sigmoid(hidden_layer.dot(self.weights_h_o))

        ### Backward pass ###
        # Output error
        output_layer_error = output_layer - Label
        output_layer_delta = output_layer_error * self.sigmoid_derivative(output_layer)
         # Backpropagated error - to the hidden layer
        hidden_layer_error = output_layer_delta.dot(self.weights_h_o.T)
        # hidden layer gradients - no nonlinearity so it's the same as the error
        hidden_layer_delta = output_layer_error

        # update the weights - with grdient descent
        self.weights_h_o -= hidden_layer.T.dot(output_layer_delta) * self.learning_rate
        self.weights_i_h -= self.vocab_vector.T.dot(hidden_layer_delta) * self.learning_rate

        if(np.abs(output_layer_error) < 0.5):
            self.correct_so_far += 1

```



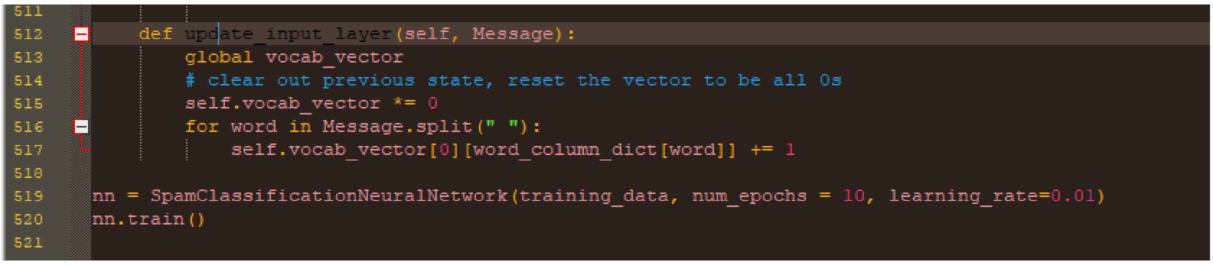
```python

    def sigmoid(self,x):
        return 1 / (1 + np.exp(-x))


    def sigmoid_derivative(self,x):
        return x * (1 - x)

```



```python
    def train(self):
        for epoch in range(self.num_epochs):
            self.correct_so_far = 0
            start = time.time()
            for i in range(len(training_data)):
                # Forward and Back Propagation
                self.forward_backward_propagate(training_data["Message"][i], training_data["Label_tag"][i])

                samples_per_second = i / float(time.time() - start + 0.001)

                sys.stdout.write("\rEpoch: "+ str(epoch)
                                 +" Progress: " + str(100 * i/float(len(training_data)))[:4]
                                 + " % Speed(samples/sec): " + str(samples_per_second)[0:5]
                                 + " #Correct: " + str(self.correct_so_far)
                                 + " #Trained: " + str(i+1)
                                 + " Training Accuracy: " + str(self.correct_so_far * 100 / float(i+1))[:4] + "%")
            print("")
```



```python
    def pre_process_data(self, training_data):
        vocab = set()

        for review in training_data["Messsage"]:
            for word in review.split(" "):
                vocab.add(word)

        self.vocab = list(vocab)

        self.word_to_column = {}
        for i, word in enumerate(self.vocab):
            self.word_to_column[word] = i
```



```python
    def update_input_layer(self, Message):
        global vocab_vector
        # clear out previous state, reset the vector to be all 0s
        self.vocab_vector *= 0
        for word in Message.split(" "):
            self.vocab_vector[0][word_column_dict[word]] += 1

nn = SpamClassificationNeuralNetwork(training_data, num_epochs = 10, learning_rate=0.01)
nn.train()
```