```
! pip install nltk
! pip install keras
! pip install lifelines

```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: nltk in /usr/local/lib/python3.9/dist-packages (3.8.1)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.9/dist-packages (from nltk) (2022.10.31)
Requirement already satisfied: joblib in /usr/local/lib/python3.9/dist-packages (from nltk) (1.1.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages (from nltk) (4.65.0)
Requirement already satisfied: click in /usr/local/lib/python3.9/dist-packages (from nltk) (8.1.3)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: keras in /usr/local/lib/python3.9/dist-packages (2.12.0)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: lifelines in /usr/local/lib/python3.9/dist-packages (0.27.4)
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.9/dist-packages (from lifelines) (1.4.4)
Requirement already satisfied: autograd>=1.5 in /usr/local/lib/python3.9/dist-packages (from lifelines) (1.5)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.9/dist-packages (from lifelines) (1.22.4)
Requirement already satisfied: scipy>=1.2.0 in /usr/local/lib/python3.9/dist-packages (from lifelines) (1.10.1)
Requirement already satisfied: matplotlib>=3.0 in /usr/local/lib/python3.9/dist-packages (from lifelines) (3.7.1)
Requirement already satisfied: autograd-gamma>=0.3 in /usr/local/lib/python3.9/dist-packages (from lifelines) (0.5.0)
Requirement already satisfied: formulaic>=0.2.2 in /usr/local/lib/python3.9/dist-packages (from lifelines) (0.5.2)
Requirement already satisfied: future>=0.15.2 in /usr/local/lib/python3.9/dist-packages (from autograd>=1.5->lifelines) (
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/python3.9/dist-packages (from formulaic>=0.2.2-
Requirement already satisfied: astor>=0.8 in /usr/local/lib/python3.9/dist-packages (from formulaic>=0.2.2->lifelines) ((
Requirement already satisfied: interface-meta>=1.2.0 in /usr/local/lib/python3.9/dist-packages (from formulaic>=0.2.2->li
Requirement already satisfied: wrapt>=1.0 in /usr/local/lib/python3.9/dist-packages (from formulaic>=0.2.2->lifelines) (1
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0->lifeline
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0->lifelir
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0->lifelines
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0->lifelines) (
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0->lifelir
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0->lifeline
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0->lifelines)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0->life
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.(
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=1.0.0->lifelines) (2(
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-packages (from importlib-resources>=3.2.0->ma
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packages (from python-dateutil>=2.7->matplotlib>
```

```
!pip install -q wordcloud
import wordcloud
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

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

```
#Importing the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.svm import SVC
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from scipy.sparse import csr_matrix
from sklearn.metrics import roc_curve, auc
from keras.models import Sequential
from keras.layers import Dense, Conv1D, MaxPooling1D, Flatten
import numpy as np
```

```
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt
# Importing the warnings
import warnings
warnings.filterwarnings('ignore')
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve, average_precision_score

```

```
#Loading the dataset
df = pd.read_csv("messages.csv",encoding='latin-1')
```

```
df.head()
```

	subject	message	label
0	job posting - apple-iss research center	content - length : 3386 apple-iss research cen...	0
1	NaN	lang classification grimes , joseph e . and ba...	0
2	query : letter frequencies for text identifica...	i am posting this inquiry for sergei atamas (...	0
3	risk	a colleague and i are researching the differin...	0
4	request book information	earlier this morning i was on the phone with a...	0

```
#Checking information of dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2893 entries, 0 to 2892
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   subject  2831 non-null   object
 1   message  2893 non-null   object
 2   label    2893 non-null   int64
dtypes: int64(1), object(2)
memory usage: 67.9+ KB
```

```
#Checking the shape of the dataset
print("Shape of the dataset:", df.shape)
```

```
Shape of the dataset: (2893, 3)
```

```
#Checking for the null values
df.isnull().values.any()
```

```
True
```

```
#Checkin for the null values in columns
df.isnull().sum()
```

```
subject    62
message     0
label       0
dtype: int64
```

```
# replace null values with empty strings
df["message"] = df["message"].replace(np.nan, "", regex=True)

# remove messages with empty strings
df = df[df["message"].str.strip().astype(bool)]

# update csv file
df.to_csv("messages_cleaned.csv", index=False)

```

```
# 62 row are missing in the subject columns that means 62 emails are without subject heading.
# Here, not dropping Nan rows for subject column as it of no use in building model.
```

```
#Checking total number of mails
print("Count of label:\n",df['label'].value_counts())
```

```
Count of label:
 0    2412
1     481
Name: label, dtype: int64
```

```
# Note:- Here in our dataset 1 stands for Spam mail and 0 stands for not a spam mail.
#Checking the Ratio of labels
print("Not a Spam Email Ratio i.e. 0 label:",round(len(df[df['label']==0])/len(df['label']),2)*100,"%")
print("Spam Email Ratio that is 1 label:",round(len(df[df['label']==1])/len(df['label']),2)*100,"%")
```

```
Not a Spam Email Ratio i.e. 0 label: 83.0 %
Spam Email Ratio that is 1 label: 17.0 %
```

```
# so here 17 % of the data is a spam email
```

```
#Creating the new column for length of message column
df['length'] = df.message.str.len()
df.head()
```

	subject	message	label	length
0	job posting - apple-iss research center	content - length : 3386 apple-iss research cen...	0	2856
1	NaN	lang classification grimes , joseph e . and ba...	0	1800
2	query : letter frequencies for text identifica...	i am posting this inquiry for sergei atamas (...	0	1435
3	risk	a colleague and i are researching the differin...	0	324
4	request book information	earlier this morning i was on the phone with a...	0	1046

```
#Converting all messages to lower case
df['message'] = df['message'].str.lower()
df.head()
```

	subject	message	label	length
0	job posting - apple-iss research center	content - length : 3386 apple-iss research cen...	0	2856
1	NaN	lang classification grimes , joseph e . and ba...	0	1800
2	query : letter frequencies for text identifica...	i am posting this inquiry for sergei atamas (...	0	1435
3	risk	a colleague and i are researching the differin...	0	324
4	request book information	earlier this morning i was on the phone with a...	0	1046

```
# regular expressions
# Replace email addresses with 'email'
df['message'] = df['message'].str.replace(r'^.+@[^\.].*\.[a-z]{2,}$','emailaddress')

# Replace URLs with 'webaddress'
df['message'] = df['message'].str.replace(r'^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)?$','webaddress')

# Replace currency symbols with 'moneysymb' (£ can by typed with ALT key + 156)
df['message'] = df['message'].str.replace(r'£|\$', 'dollers')

# Replace 10 digit phone numbers (formats include paranthesis, spaces, no spaces, dashes) with 'phonenumber'
df['message'] = df['message'].str.replace(r'^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?[\d]{4}$','phonenumber')

# Replace numeric characters with 'numbr'
df['message'] = df['message'].str.replace(r'\d+(\.\d+)?', 'numbr')
```

```
# Remove punctuation
df['message'] = df['message'].str.replace(r'[^\w\d\s]', ' ')

# Replace whitespace between terms with a single space
df['message'] = df['message'].str.replace(r'\s+', ' ')

# Remove leading and trailing whitespace
df['message'] = df['message'].str.replace(r'^\s+|\s+?$', '')
```

```
# now re-checking the data
df.head()
```

	subject	message	label	length
0	job posting - apple-iss research center	content length numbr apple iss research center...	0	2856
1	NaN	lang classification grimes joseph e and barbar...	0	1800

```
#Removing the stopwords
import string
import nltk
from nltk.corpus import stopwords

stop_words = set(stopwords.words('english') + ['u', 'ü', 'ur', '4', '2', 'im', 'dont', 'doin', 'ure'])

df['message'] = df['message'].apply(lambda x: " ".join(term for term in x.split() if term not in stop_words))
```

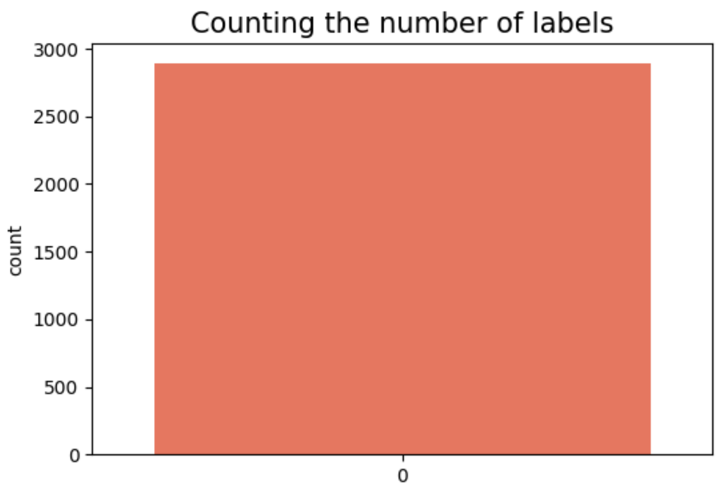
```
# New column (clean_length) after puncuations,stopwords removal
df['clean_length'] = df.message.str.len()
df.head()
```

	subject	message	label	length	clean_length
0	job posting - apple-iss research center	content length numbr apple iss research center...	0	2856	2179
1	NaN	lang classification grimes joseph e barbara f ...	0	1800	1454
2	query : letter frequencies for text identifica...	posting inquiry sergei atamas satamas umabnet ...	0	1435	1064
3	risk	colleague researching differing degrees risk p...	0	324	210
4	request book information	earlier morning phone friend mine living south...	0	1046	629

```
#Total length removal
print("Original Length:",df.length.sum())
print("Cleaned Length:",df.clean_length.sum())
print("Total Words Removed:",(df.length.sum()) - (df.clean_length.sum()))
```

```
Original Length: 9344743
Cleaned Length: 6767857
Total Words Removed: 2576886
```

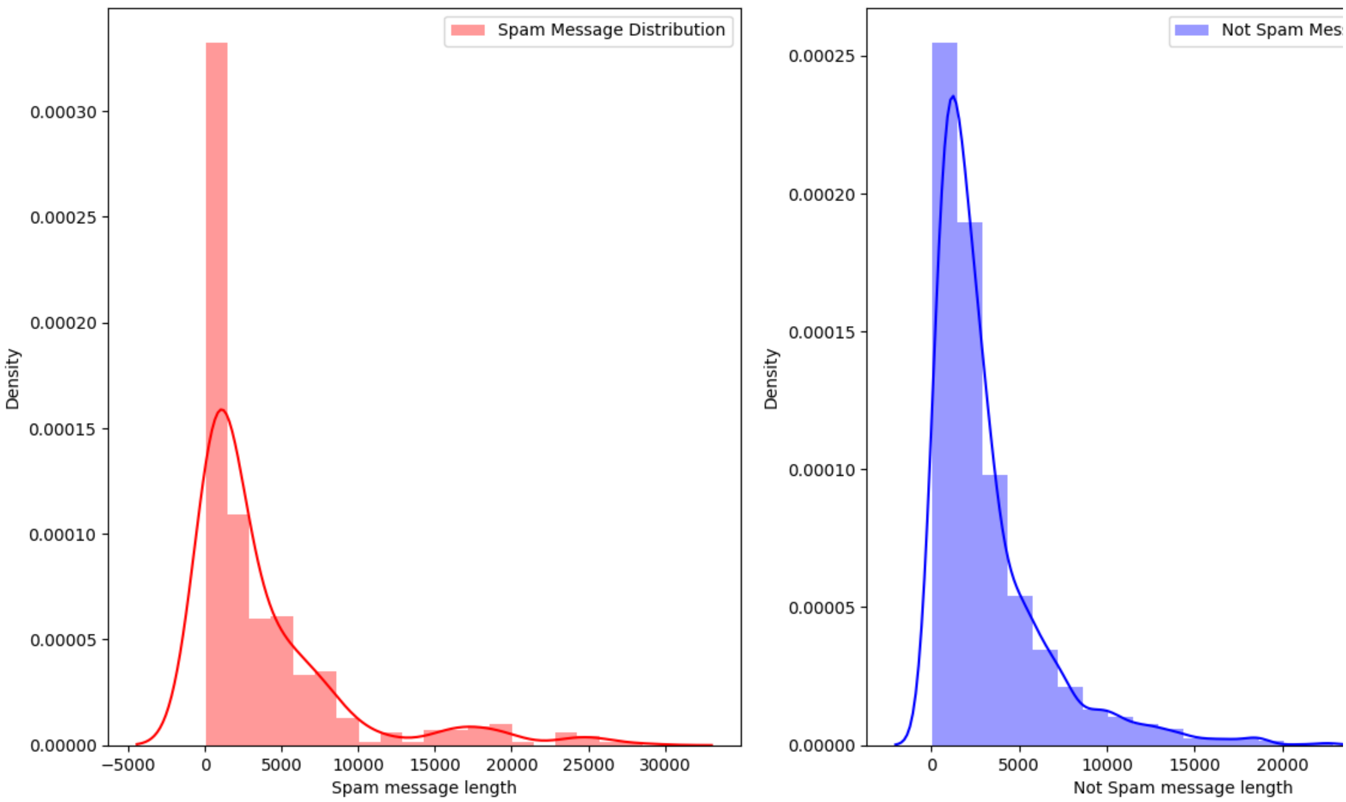
```
#Graphical Visualisation for counting number of labels.
plt.figure(figsize=(6,4))
sns.countplot(df['label'],palette= 'Reds')
plt.title("Counting the number of labels",fontsize=15)
plt.xticks(rotation='horizontal')
plt.show()

print(df.label.value_counts())
```



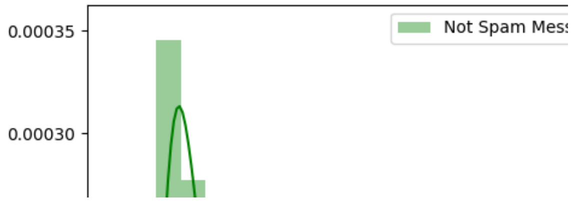

```
0    2412
1     481
Name: label, dtype: int64
```

```
#Message distribution before cleaning
f,ax = plt.subplots(1,2,figsize=(15,8))

sns.distplot(df[df['label']==1]['length'],bins=20, ax=ax[0],label='Spam Message Distribution',color='r')
ax[0].set_xlabel('Spam message length')
ax[0].legend()

sns.distplot(df[df['label']==0]['length'],bins=20, ax=ax[1],label='Not Spam Message Distribution',color='b')
```

```
ax[1].set_xlabel('Not Spam message length')
ax[1].legend()

plt.show()
```

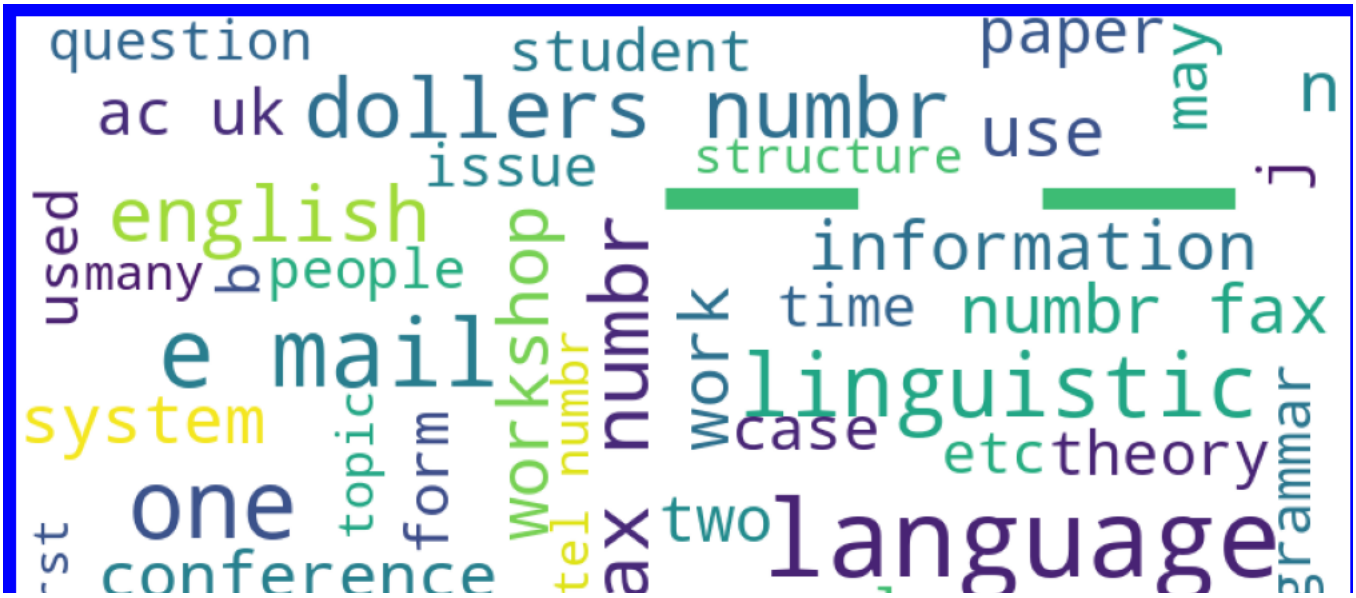



```
#Message distribution after cleaning
f,ax = plt.subplots(1,2,figsize=(15,8))

sns.distplot(df[df['label']==1]['clean_length'],bins=20, ax=ax[0],label='Spam Message Distribution',color='r')
ax[0].set_xlabel('Spam message length')
ax[0].legend()

sns.distplot(df[df['label']==0]['clean_length'],bins=20, ax=ax[1],label='Not Spam Message Distribution',color='g')
ax[1].set_xlabel('Not a Spam message length')
ax[1].legend()

plt.show()
```




```
#Getting sense of loud words in spam
from wordcloud import WordCloud


spams = df['message'][df['label']==1]

spam_cloud = WordCloud(width=800,height=500,background_color='white',max_words=50).generate(' '.join(spams))

plt.figure(figsize=(10,8),facecolor='b')
plt.imshow(spam_cloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```


```
#Getting sense of loud words in not-spam
from wordcloud import WordCloud

not_spams = df['message'][df['label']==0]

spam_cloud = WordCloud(width=800,height=500,background_color='white',max_words=50).generate(' '.join(not_spams))

plt.figure(figsize=(10,8),facecolor='b')
plt.imshow(spam_cloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```




```
# Converting the text into vectors using TF-IDF, as text cannot be the input in the model
# 1. Convert text into vectors using TF-IDF
# 2. Instantiate MultinomialNB classifier
# 3. Split feature and label


tf_vec = TfidfVectorizer()

naive = MultinomialNB()

SVM = SVC(C=1.0, kernel='linear', degree=3 , gamma='auto')

decision = DecisionTreeClassifier()
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")

clf = LogisticRegression()


features = tf_vec.fit_transform(df['message'])

X = features
y = df['label']
```

```
# Train and predict for naive bayes model
X_train,x_test,Y_train,y_test = train_test_split(X,y,random_state=42)

#test_size=0.20 random_state=42 test_size=0.15

naive.fit(X_train,Y_train)
y_pred= naive.predict(x_test)



print ('Final score = > ', accuracy_score(y_test,y_pred))
```

```
Final score = >  0.8342541436464088
```

```
# Train and predict for SVM model
X_train,x_test,Y_train,y_test = train_test_split(X,y,random_state=42)

#test_size=0.20 random_state=42 test_size=0.15



SVM.fit(X_train,Y_train)
y_pred = SVM.predict(x_test)

print ('Final score = > ', accuracy_score(y_test,y_pred))
```

```
Final score = >  0.9875690607734806
```

```
# train and predict for the Decision tree model
X_train,x_test,Y_train,y_test = train_test_split(X,y,random_state=42)
decision.fit(X_train,Y_train)
#test_size=0.20 random_state=42 test_size=0.15

y_pred = decision.predict(x_test)
print ('Final score = > ', accuracy_score(y_test,y_pred))


```

```
Final score = >  0.9571823204419889
```

```
# train and predict uisng random forest classifier
X_train,x_test,Y_train,y_test = train_test_split(X,y,random_state=42)
classifier.fit(X_train, Y_train)
#test_size=0.20 random_state=42 test_size=0.15
y_pred= classifier.predict(x_test)
print ('Final score = > ', accuracy_score(y_test,y_pred))

```

```
Final score = >  0.9585635359116023
```

```
# train and predict uisng logistic regression
X_train,x_test,Y_train,y_test = train_test_split(X,y,random_state=42)
clf.fit(X_train, Y_train)
#test_size=0.20 random_state=42 test_size=0.15
y_pred= clf.predict(x_test)
print ('Final score = > ', accuracy_score(y_test,y_pred))

```

```
Final score = >  0.9475138121546961
```

```
# train and predict uisng KNN
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create KNN model and fit it to the training data
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = knn.predict(X_test)

# Measure accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

```

```
Accuracy: 0.9758203799654577
```

```
# train and predict uisng Neural Network
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a neural network model and fit it to the training data
nn = MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000)
nn.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = nn.predict(X_test)

# Measure accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

```

```
Accuracy: 0.9930915371329879
```

```
# train and predict uisng Gradient Boosting Algorithm
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create gradient boosting model and fit it to the training data
gb = GradientBoostingClassifier()
gb.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = gb.predict(X_test)

# Measure accuracy of the model
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9740932642487047
```

```
# train and predict uisng Gaussian Naive Bayes model
# Convert sparse matrix X to dense numpy array
X_dense = X.toarray()

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_dense, y, test_size=0.2, random_state=42)

# Create Gaussian Naive Bayes model and fit it to the training data
nb = GaussianNB()
nb.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = nb.predict(X_test)

# Measure accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9430051813471503
```

```
# train and predict uisng CNN

# Convert sparse matrix X to dense numpy array
X_dense = X.toarray()

# Reshape X_dense for use in 1D Convolutional Neural Network
X_reshaped = np.expand_dims(X_dense, axis=2)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_reshaped, y, test_size=0.2, random_state=42)

# Define the CNN model
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))

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

# Fit the model to the training data
model.fit(X_train, y_train, epochs=10, batch_size=32)

# Make predictions on the testing data
y_pred_prob = model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)

# Measure accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Epoch 1/10
73/73 [==============================] - 84s 1s/step - loss: 0.4445 - accuracy: 0.8392
Epoch 2/10
73/73 [==============================] - 95s 1s/step - loss: 0.2247 - accuracy: 0.9097
Epoch 3/10
73/73 [==============================] - 82s 1s/step - loss: 0.0910 - accuracy: 0.9831
Epoch 4/10
73/73 [==============================] - 84s 1s/step - loss: 0.0458 - accuracy: 0.9948
Epoch 5/10
73/73 [==============================] - 85s 1s/step - loss: 0.0259 - accuracy: 0.9991
Epoch 6/10
73/73 [==============================] - 82s 1s/step - loss: 0.0153 - accuracy: 0.9991
Epoch 7/10
73/73 [==============================] - 84s 1s/step - loss: 0.0100 - accuracy: 0.9996
Epoch 8/10
73/73 [==============================] - 92s 1s/step - loss: 0.0065 - accuracy: 1.0000
Epoch 9/10
73/73 [==============================] - 102s 1s/step - loss: 0.0045 - accuracy: 1.0000
Epoch 10/10
73/73 [==============================] - 87s 1s/step - loss: 0.0033 - accuracy: 1.0000
19/19 [==============================] - 7s 341ms/step
Accuracy: 0.8013816925734024
```

```
1 y_pred
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0])
```

```
1 tree.plot_tree(decision)
```

```
[Text(0.8589449541284404, 0.9821428571428571, 'x[18089] <= 0.015\ngini = 0.266\nsamples = 2169\nvalue = [1827, 342]'),
 Text(0.7766055045871559, 0.9464285714285714, 'x[41154] <= 0.009\ngini = 0.157\nsamples = 1901\nvalue = [1738, 163]'),
 Text(0.7146788990825688, 0.9107142857142857, 'x[32052] <= 0.076\ngini = 0.12\nsamples = 1853\nvalue = [1734, 119]'),
 Text(0.6642201834862386, 0.875, 'x[8614] <= 0.053\ngini = 0.1\nsamples = 1828\nvalue = [1732, 96]'),
 Text(0.6073394495412844, 0.8392857142857143, 'x[19527] <= 0.068\ngini = 0.08\nsamples = 1804\nvalue = [1729, 75]'),
 Text(0.5669724770642202, 0.8035714285714286, 'x[23195] <= 0.019\ngini = 0.065\nsamples = 1780\nvalue = [1720, 60]'),
 Text(0.5302752293577981, 0.7678571428571429, 'x[49044] <= 0.029\ngini = 0.057\nsamples = 1771\nvalue = [1719, 52]'),
 Text(0.5009174311926605, 0.7321428571428571, 'x[13671] <= 0.05\ngini = 0.049\nsamples = 1760\nvalue = [1716, 44]'),
 Text(0.48623853211009177, 0.6964285714285714, 'x[7268] <= 0.044\ngini = 0.043\nsamples = 1755\nvalue = [1716, 39]'),
 Text(0.45688073394495415, 0.6607142857142857, 'x[11366] <= 0.082\ngini = 0.039\nsamples = 1750\nvalue = [1715, 35]'),
 Text(0.44220183486238535, 0.625, 'x[39836] <= 0.082\ngini = 0.036\nsamples = 1747\nvalue = [1715, 32]'),
 Text(0.3963302752293578, 0.5892857142857143, 'x[30114] <= 0.082\ngini = 0.032\nsamples = 1741\nvalue = [1713, 28]'),
 Text(0.3486238532110092, 0.5535714285714286, 'x[11408] <= 0.076\ngini = 0.027\nsamples = 1735\nvalue = [1711, 24]'),
 Text(0.29724770642201837, 0.5178571428571429, 'x[35084] <= 0.123\ngini = 0.023\nsamples = 1728\nvalue = [1708, 20]'),
 Text(0.26788990825688075, 0.48214285714285715, 'x[49688] <= 0.023\ngini = 0.02\nsamples = 1724\nvalue = [1707, 17]'),
 Text(0.25321100917431194, 0.44642857142857145, 'x[28346] <= 0.135\ngini = 0.017\nsamples = 1722\nvalue = [1707, 15]'),
 Text(0.21284403669724772, 0.4107142857142857, 'x[39102] <= 0.076\ngini = 0.015\nsamples = 1719\nvalue = [1706, 13]'),
 Text(0.1761467889908257, 0.375, 'x[6931] <= 0.052\ngini = 0.013\nsamples = 1716\nvalue = [1705, 11]'),
 Text(0.14678899082568808, 0.3392857142857143, 'x[40841] <= 0.015\ngini = 0.01\nsamples = 1713\nvalue = [1704, 9]'),
 Text(0.13211009174311927, 0.30357142857142855, 'x[30559] <= 0.122\ngini = 0.009\nsamples = 1712\nvalue = [1704, 8]'),
 Text(0.11743119266055047, 0.26785714285714285, 'x[48312] <= 0.173\ngini = 0.008\nsamples = 1711\nvalue = [1704, 7]'),
 Text(0.10275229357798166, 0.23214285714285715, 'x[53919] <= 0.087\ngini = 0.007\nsamples = 1710\nvalue = [1704, 6]'),
 Text(0.08807339449541285, 0.19642857142857142, 'x[29608] <= 0.159\ngini = 0.006\nsamples = 1709\nvalue = [1704, 5]'),
 Text(0.07339449541284404, 0.16071428571428573, 'x[8947] <= 0.102\ngini = 0.005\nsamples = 1708\nvalue = [1704, 4]'),
 Text(0.05871559633027523, 0.125, 'x[50332] <= 0.287\ngini = 0.004\nsamples = 1707\nvalue = [1704, 3]'),
 Text(0.044036697247706424, 0.08928571428571429, 'x[24241] <= 0.129\ngini = 0.002\nsamples = 1706\nvalue = [1704, 2]'),
 Text(0.029357798165137616, 0.05357142857142857, 'x[36100] <= 0.069\ngini = 0.001\nsamples = 1705\nvalue = [1704, 1]'),
 Text(0.014678899082568808, 0.017857142857142856, 'gini = 0.0\nsamples = 1704\nvalue = [1704, 0]'),
 Text(0.044036697247706424, 0.017857142857142856, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.05871559633027523, 0.05357142857142857, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.07339449541284404, 0.08928571428571429, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.08807339449541285, 0.125, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.10275229357798166, 0.16071428571428573, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.11743119266055047, 0.19642857142857142, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.13211009174311927, 0.23214285714285715, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.14678899082568808, 0.26785714285714285, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.1614678899082569, 0.30357142857142855, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.20550458715596331, 0.3392857142857143, 'x[8377] <= 0.094\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.1908256880733945, 0.30357142857142855, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.22018348623853212, 0.30357142857142855, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.24954128440366974, 0.375, 'x[6002] <= 0.03\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.23486238532110093, 0.3392857142857143, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.26422018348623855, 0.3392857142857143, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.29357798165137616, 0.4107142857142857, 'x[53983] <= 0.049\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.27889908256880735, 0.375, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.30825688073394497, 0.375, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.28256880733944956, 0.44642857142857145, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.326605504587156, 0.48214285714285715, 'x[1745] <= 0.092\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
 Text(0.3119266055045872, 0.44642857142857145, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.3412844036697248, 0.44642857142857145, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.4, 0.5178571428571429, 'x[35511] <= 0.059\ngini = 0.49\nsamples = 7\nvalue = [3, 4]'),
 Text(0.3853211009174312, 0.48214285714285715, 'x[33644] <= 0.062\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
 Text(0.3706422018348624, 0.44642857142857145, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.4, 0.44642857142857145, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.41467889908256883, 0.48214285714285715, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.44403669724770645, 0.5535714285714286, 'x[54053] <= 0.018\ngini = 0.444\nsamples = 6\nvalue = [2, 4]'),
 Text(0.42935779816513764, 0.5178571428571429, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.45871559633027525, 0.5178571428571429, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.48807339449541287, 0.5892857142857143, 'x[12037] <= 0.017\ngini = 0.444\nsamples = 6\nvalue = [2, 4]'),
 Text(0.47339449541284406, 0.5535714285714286, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.5027522935779817, 0.5535714285714286, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.47155963302752296, 0.625, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.5155963302752293, 0.6607142857142857, 'x[9817] <= 0.012\ngini = 0.32\nsamples = 5\nvalue = [1, 4]'),
 Text(0.5009174311926605, 0.625, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.5302752293577981, 0.625, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.5155963302752293, 0.6964285714285714, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(0.5596330275229358, 0.7321428571428571, 'x[10456] <= 0.017\ngini = 0.397\nsamples = 11\nvalue = [3, 8]'),
 Text(0.544954128440367, 0.6964285714285714, 'gini = 0.0\nsamples = 8\nvalue = [0, 8]'),
 Text(0.5743119266055046, 0.6964285714285714, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.6036697247706422, 0.7678571428571429, 'x[6950] <= 0.023\ngini = 0.198\nsamples = 9\nvalue = [1, 8]'),
 Text(0.5889908256880734, 0.7321428571428571, 'gini = 0.0\nsamples = 8\nvalue = [0, 8]'),
 Text(0.618348623853211, 0.7321428571428571, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.6477064220183486, 0.8035714285714286, 'x[22172] <= 0.021\ngini = 0.469\nsamples = 24\nvalue = [9, 15]'),
 Text(0.6330275229357798, 0.7678571428571429, 'gini = 0.0\nsamples = 7\nvalue = [7, 0]'),
 Text(0.6623853211009174, 0.7678571428571429, 'x[51183] <= 0.018\ngini = 0.208\nsamples = 17\nvalue = [2, 15]'),
```

```
1 # Checking Classification report
2 print(classification_report(y_test, y_pred))
```

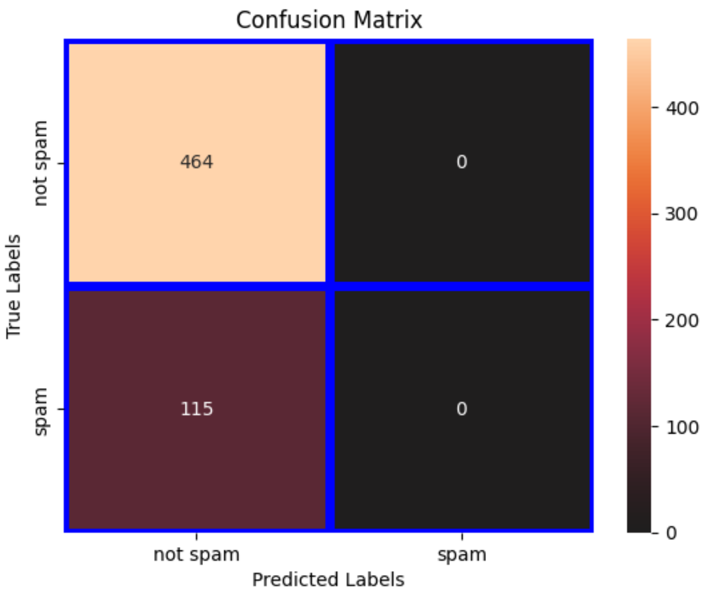
	precision	recall	f1-score	support
0	0.80	1.00	0.89	464
1	0.00	0.00	0.00	115
accuracy			0.80	579
macro avg	0.40	0.50	0.44	579
weighted avg	0.64	0.80	0.71	579

```
 Text(0.8385321100917431, 0.8392857142857143, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
```

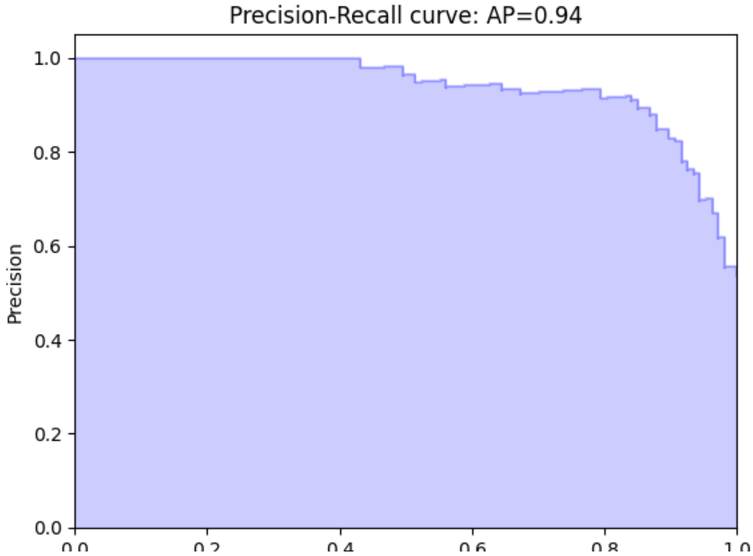
```
from sklearn.metrics import confusion_matrix

```

```
Text(0.8678899082568807, 0.8392857142857143, 'x[47827] <= 0.016\ngini = 0.083\nsamples = 185\nvalue = [8, 177]'),
```

```
conf_mat = confusion_matrix(y_test,y_pred)

ax = plt.subplot()

sns.heatmap(conf_mat, annot=True, ax=ax, linewidths=5, linecolor='b', center=0, fmt='g')

ax.set_xlabel('Predicted Labels')
ax.set_ylabel('True Labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(['not spam','spam'])
ax.yaxis.set_ticklabels(['not spam','spam'])

plt.show()

```




```


# Generate a random classification dataset
X, y = make_classification(n_samples=1000, n_classes=2, random_state=42)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a logistic regression model on the training set
model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)

# Use the model to make predictions on the testing set
y_pred = model.predict_proba(X_test)[:, 1]
y_true = y_test


#pr curve

# Assume y_true and y_pred are the true labels and predicted probabilities, respectively
precision, recall, thresholds = precision_recall_curve(y_true, y_pred)
average_precision = average_precision_score(y_true, y_pred)

# Plot the PR curve
plt.step(recall, precision, color='b', alpha=0.2, where='post')
plt.fill_between(recall, precision, step='post', alpha=0.2, color='b')

# Add labels and a legend to the plot
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Precision-Recall curve: AP={0:0.2f}'.format(average_precision))
plt.show()

```




```
### END ###
```