# Assisted Practice: Analyzing the Sentiments

In this demo, we will develop a simple application of sentiment analysis using natural language processing techniques.

Sentiment analysis is one of the most common applications in natural language processing. With sentiment analysis, we can decide with what emotion a text is written.

With the widespread use of social media, the need to analyze the content that people share over social media is increasing day by day. Considering the volume of data coming through social media, it is quite difficult to do this with mere manpower. Therefore, the need for applications that can quickly detect and respond to the positive or negative comments that people write is increasing.

Let’s look at the steps in detail that needs to be performed:

**Step 1: Import the libraries**

Here we are going to use list of libraries as shown below as part of our project:

import pandas as pd

import numpy as np

import pickle

import sys

import os

import io

import re

from sys import path

import numpy as np

import pickle

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelBinarizer

import matplotlib.pyplot as plt

from string import punctuation, digits

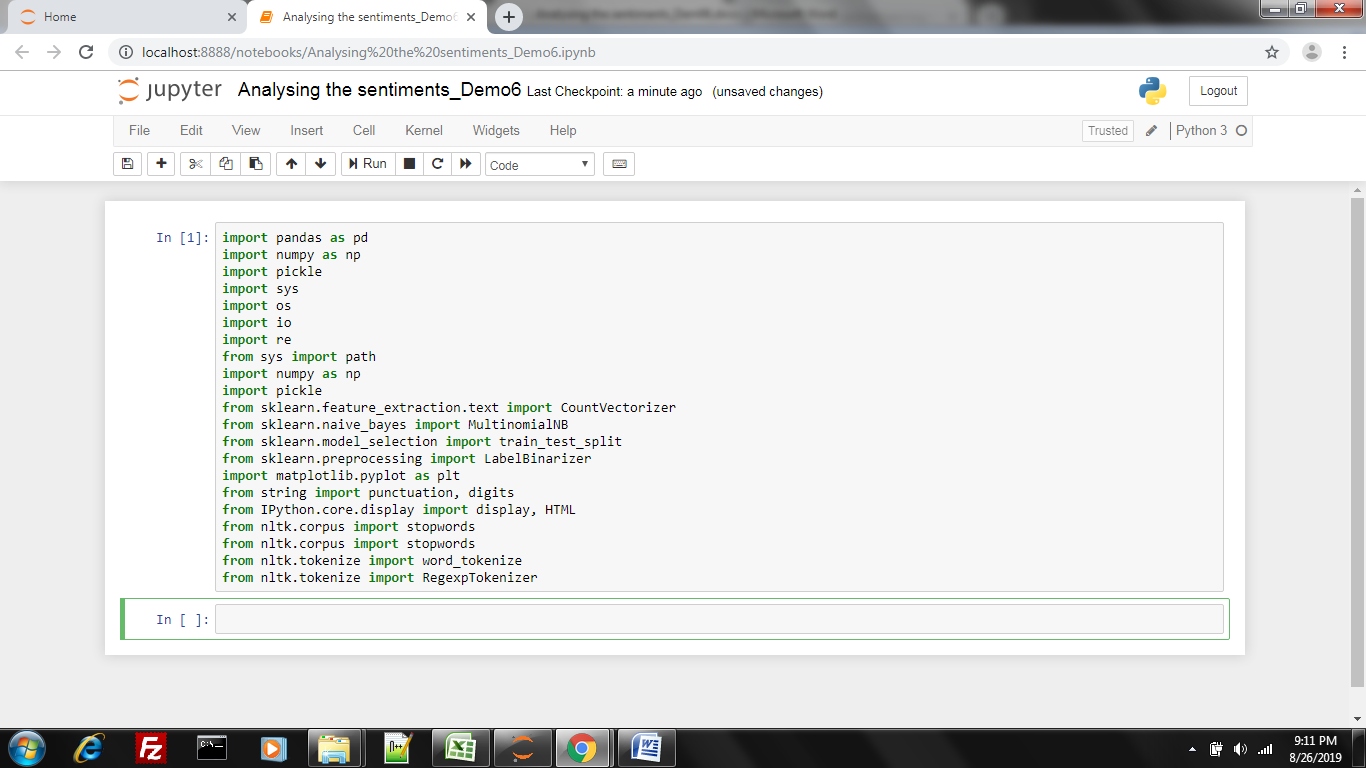
from IPython.core.display import display, HTML

from nltk.corpus import stopwords

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from nltk.tokenize import RegexpTokenizer



**Step 2: Dataset**

Here we have created the dataset with user reviews collected via 3 different websites (Amazon, Yelp, and IMDb). These comments consist reviews for restaurants, films, and products. Each record in the dataset is labeled with two different emoticons. These are 1: Positive, 0: Negative.

We will create a sentiment analysis model using the dataset that we have.

Now let's upload and see the dataset.

#Amazon Data

input\_file = "../data/amazon\_cells\_labelled.txt"

amazon = pd.read\_csv(input\_file,delimiter='\t',header=None)

amazon.columns = ['Sentence','Class']

#Yelp Data

input\_file = "../data/yelp\_labelled.txt"

yelp = pd.read\_csv(input\_file,delimiter='\t',header=None)

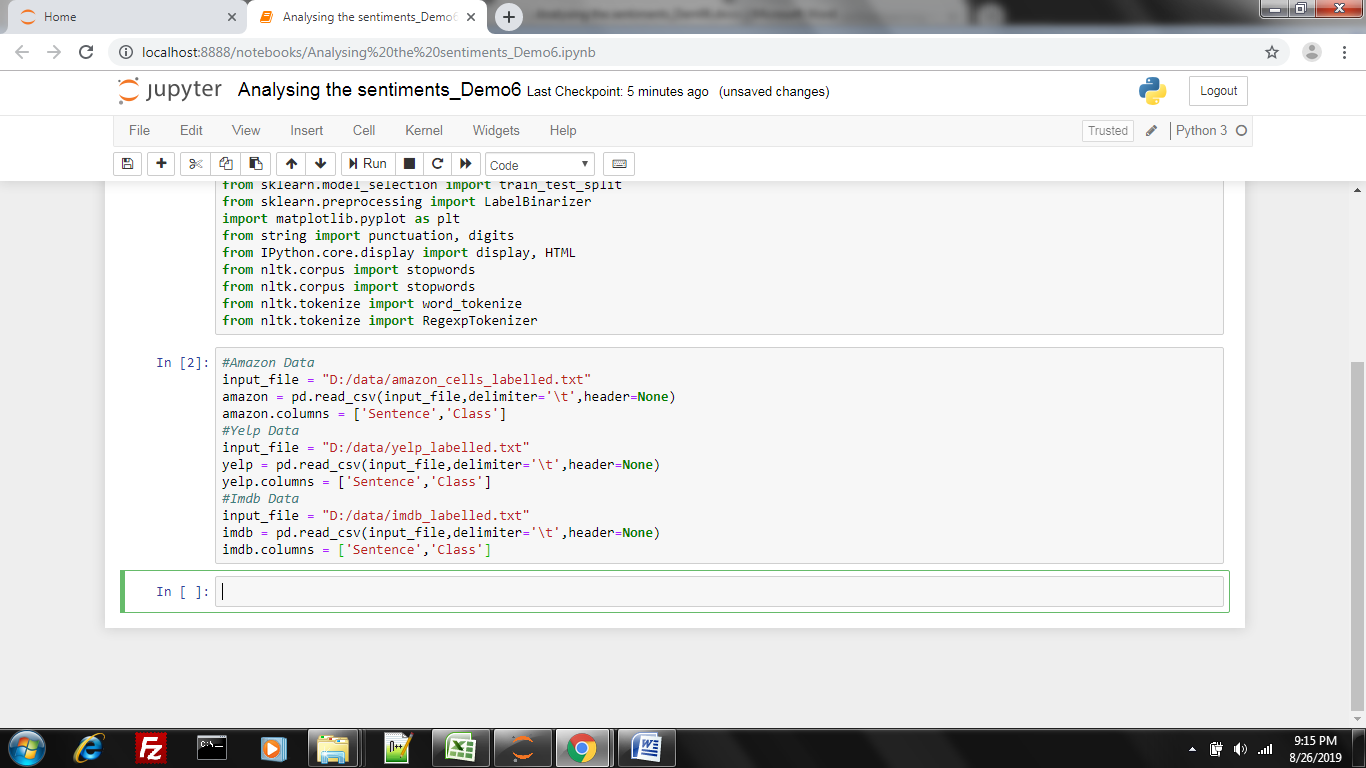
yelp.columns = ['Sentence','Class']

#Imdb Data

input\_file = "../data/imdb\_labelled.txt"

imdb = pd.read\_csv(input\_file,delimiter='\t',header=None)

imdb.columns = ['Sentence','Class']



**Step 3: Combine all datasets**

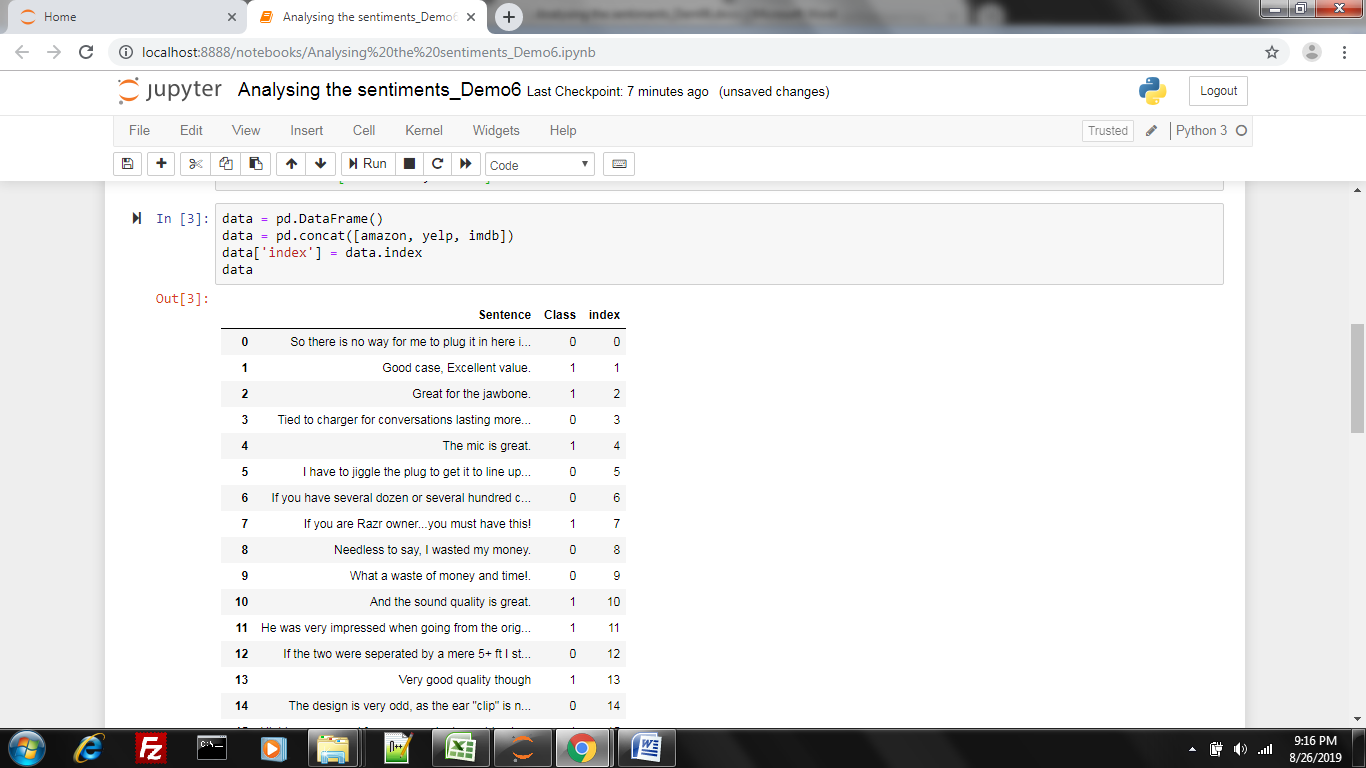
Now let's combine all the datasets into a single dataframe:

data = pd.DataFrame()

data = pd.concat([amazon, yelp, imdb])

data['index'] = data.index

data



**Step 4: Statistics**

In the above step, we imported the data and viewed it. Now, let's look at the statistics about the data.

#Total Count of Each Category

pd.set\_option('display.width', 4000)

pd.set\_option('display.max\_rows', 1000)

distOfDetails = data.groupby(by='Class', as\_index=False).agg({'index': pd.Series.nunique}).sort\_values(by='index', ascending=False)

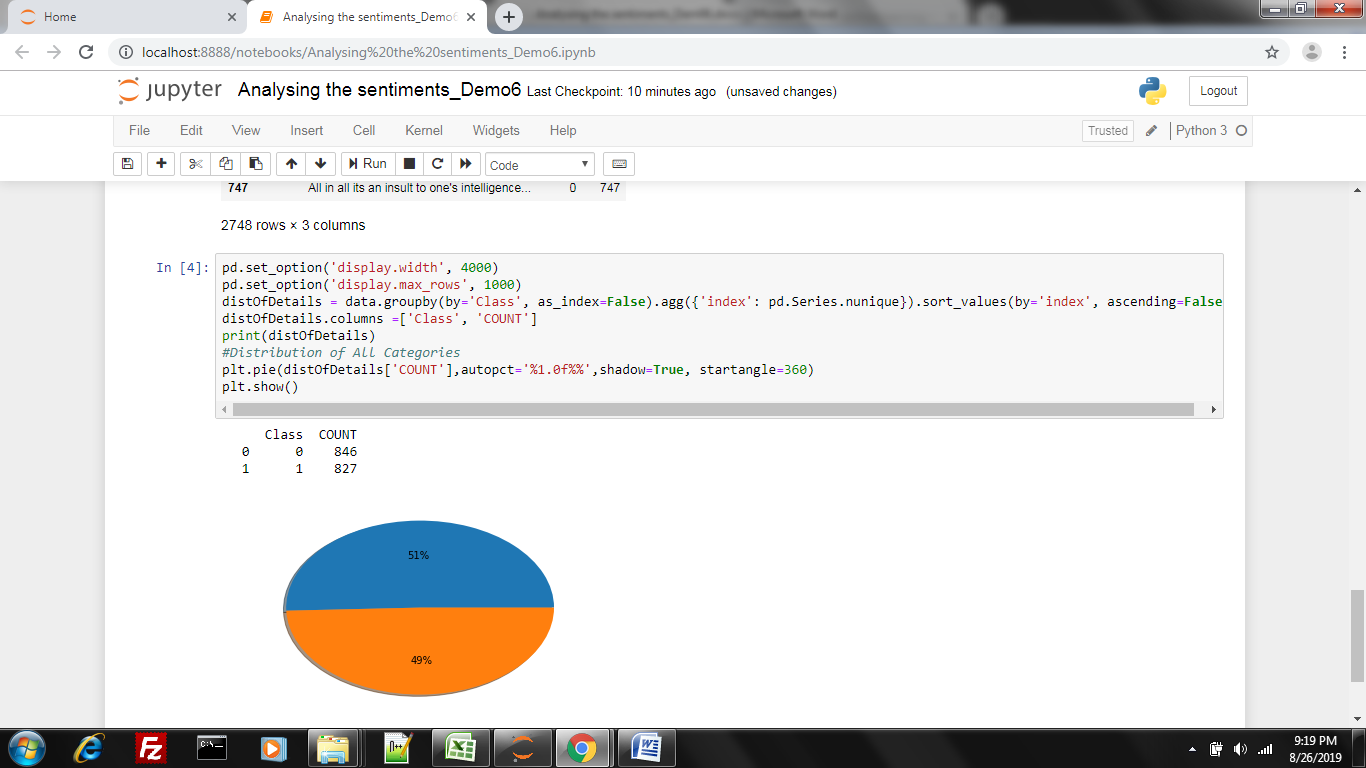
distOfDetails.columns =['Class', 'COUNT']

print(distOfDetails)

#Distribution of All Categories

plt.pie(distOfDetails['COUNT'],autopct='%1.0f%%',shadow=True, startangle=360)

plt.show()



**Step 5: Data Preprocessing**

As you can see, the data set is very balanced. There are almost equal numbers of positive and negative reviews.

Now, before using the dataset in the model, let's do a few things to clear the text preprocessing.

#Text Preprocessing

columns = ['index','Class', 'Sentence']

df\_ = pd.DataFrame(columns=columns)

#lower string

data['Sentence'] = data['Sentence'].str.lower()

#remove email adress

data['Sentence'] = data['Sentence'].replace('[a-zA-Z0-9-\_.]+@[a-zA-Z0-9-\_.]+', '', regex=True)

#remove IP address

data['Sentence'] = data['Sentence'].replace('((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)(\.|$)){4}', '', regex=True)

#remove punctaitions and special chracters

data['Sentence'] = data['Sentence'].str.replace('[^\w\s]','')

#remove numbers

data['Sentence'] = data['Sentence'].replace('\d', '', regex=True)

#remove stop words

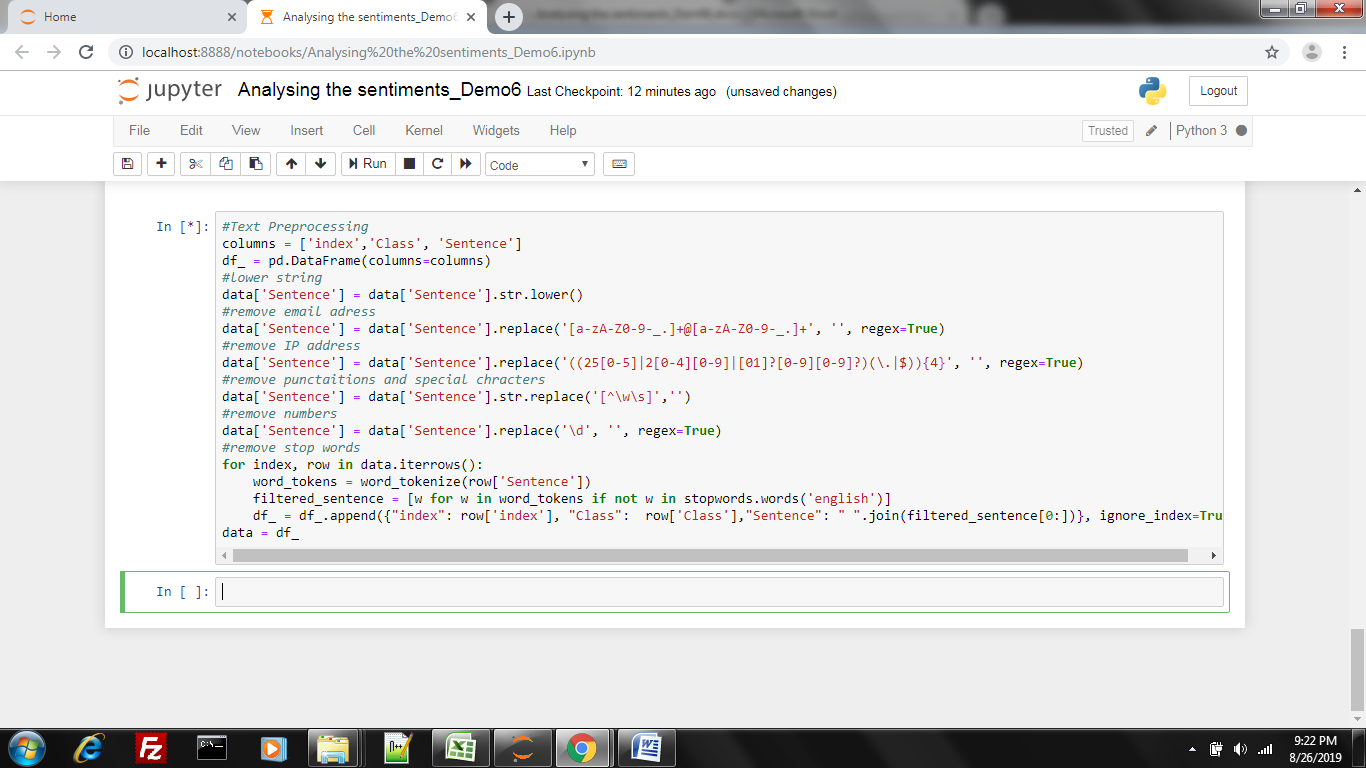
for index, row in data.iterrows():

word\_tokens = word\_tokenize(row['Sentence'])

filtered\_sentence = [w for w in word\_tokens if not w in stopwords.words('english')]

df\_ = df\_.append({"index": row['index'], "Class": row['Class'],"Sentence": " ".join(filtered\_sentence[0:])}, ignore\_index=True)

data = df\_

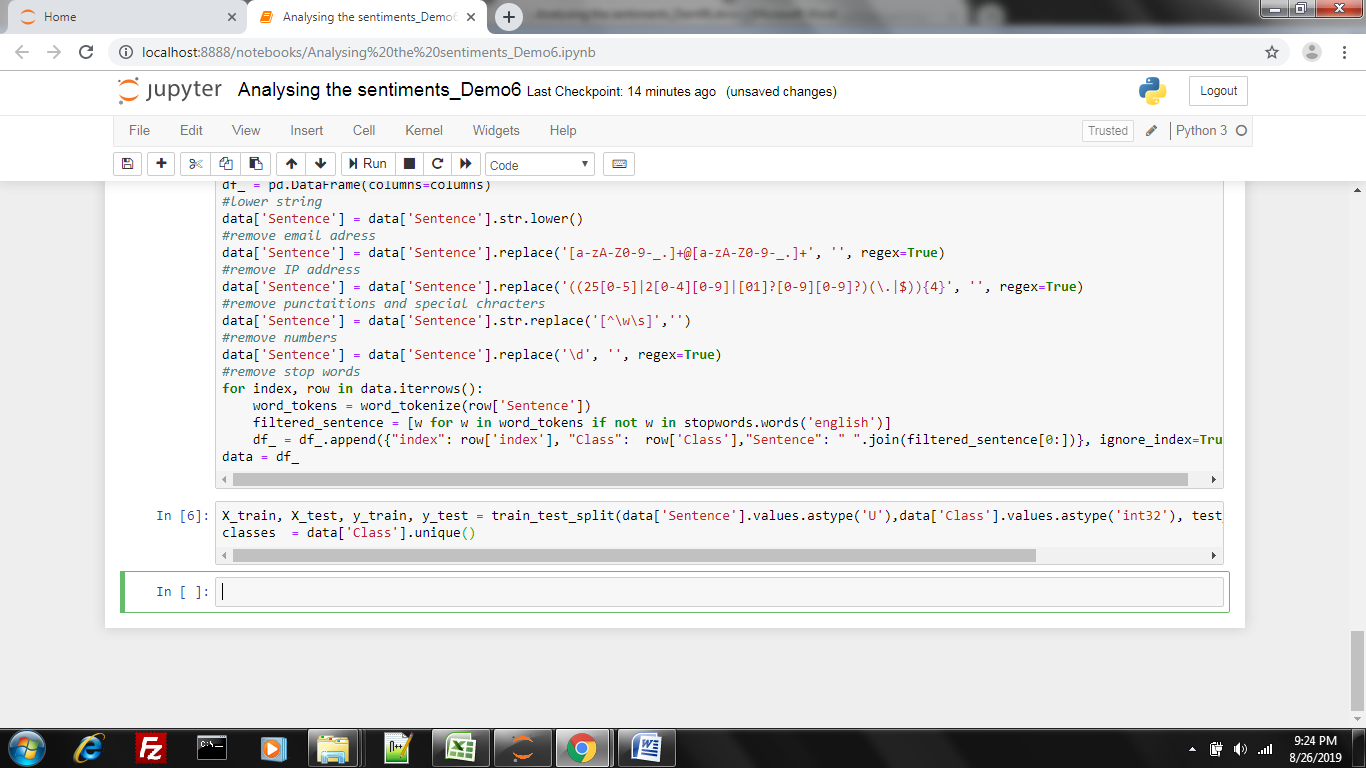


**Step 6: Data Split—Training vs. Test**

Now, before we build our model, let's split our dataset into test (10%) and training (90%).

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data['Sentence'].values.astype('U'),data['Class'].values.astype('int32'), test\_size=0.10, random\_state=0)

classes = data['Class'].unique()



**Step 7: Create a Model**

Now, we can create our model using our training data. In creating the model, we will use the TF-IDF as the vectorizer and the Stochastic Gradient Descent algorithm as the classifier. The methods and the parameters in these methods use grid search.

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import cross\_val\_score

from sklearn.metrics import accuracy\_score

from sklearn.neural\_network import MLPClassifier

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import SGDClassifier

#grid search result

vectorizer = TfidfVectorizer(analyzer='word',ngram\_range=(1,2), max\_features=50000,max\_df=0.5,use\_idf=True, norm='l2')

counts = vectorizer.fit\_transform(X\_train)

vocab = vectorizer.vocabulary\_

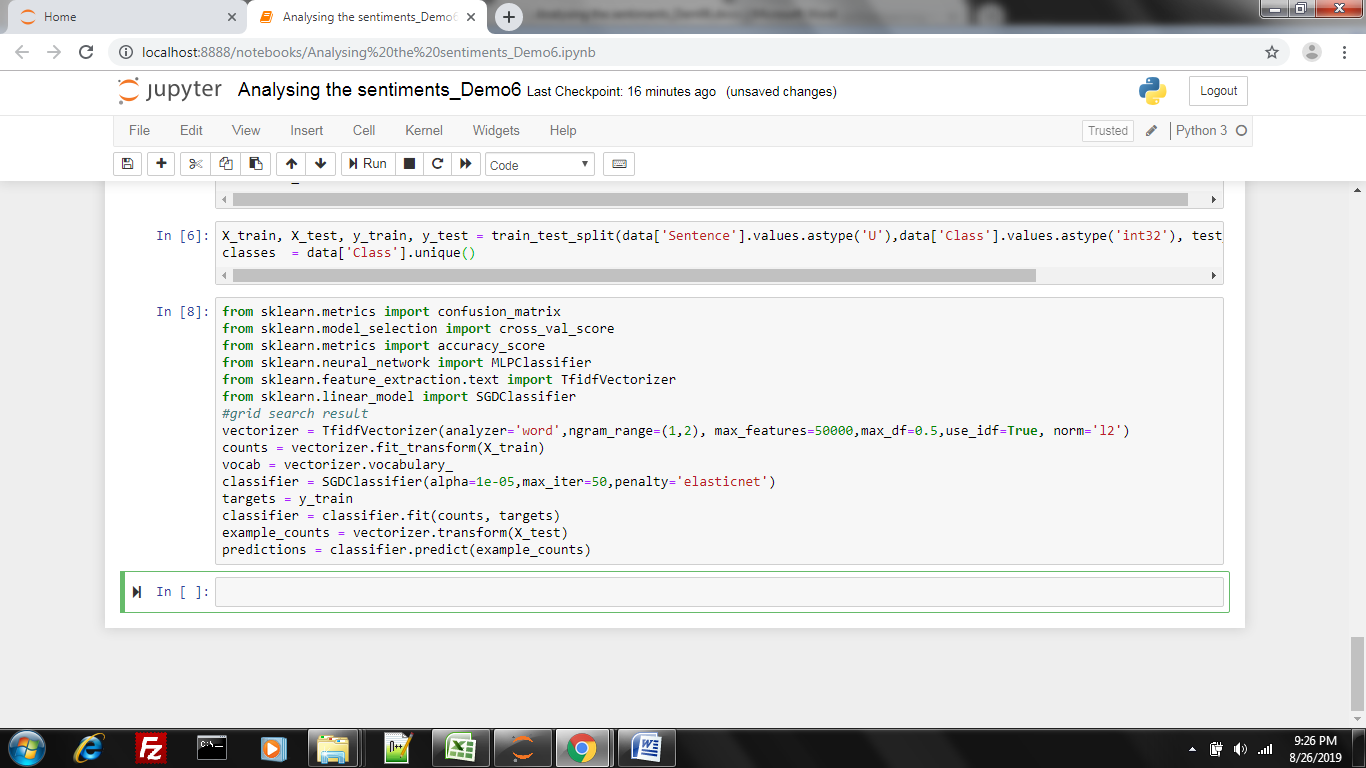
classifier = SGDClassifier(alpha=1e-05,max\_iter=50,penalty='elasticnet')

targets = y\_train

classifier = classifier.fit(counts, targets)

example\_counts = vectorizer.transform(X\_test)

predictions = classifier.predict(example\_counts)



**Step 8: Test a Model**

Our model has been created. Now, let's test our model with test data. Let's examine the accuracy, precision, recall, and f1 results.

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import classification\_report

#Model Evaluation

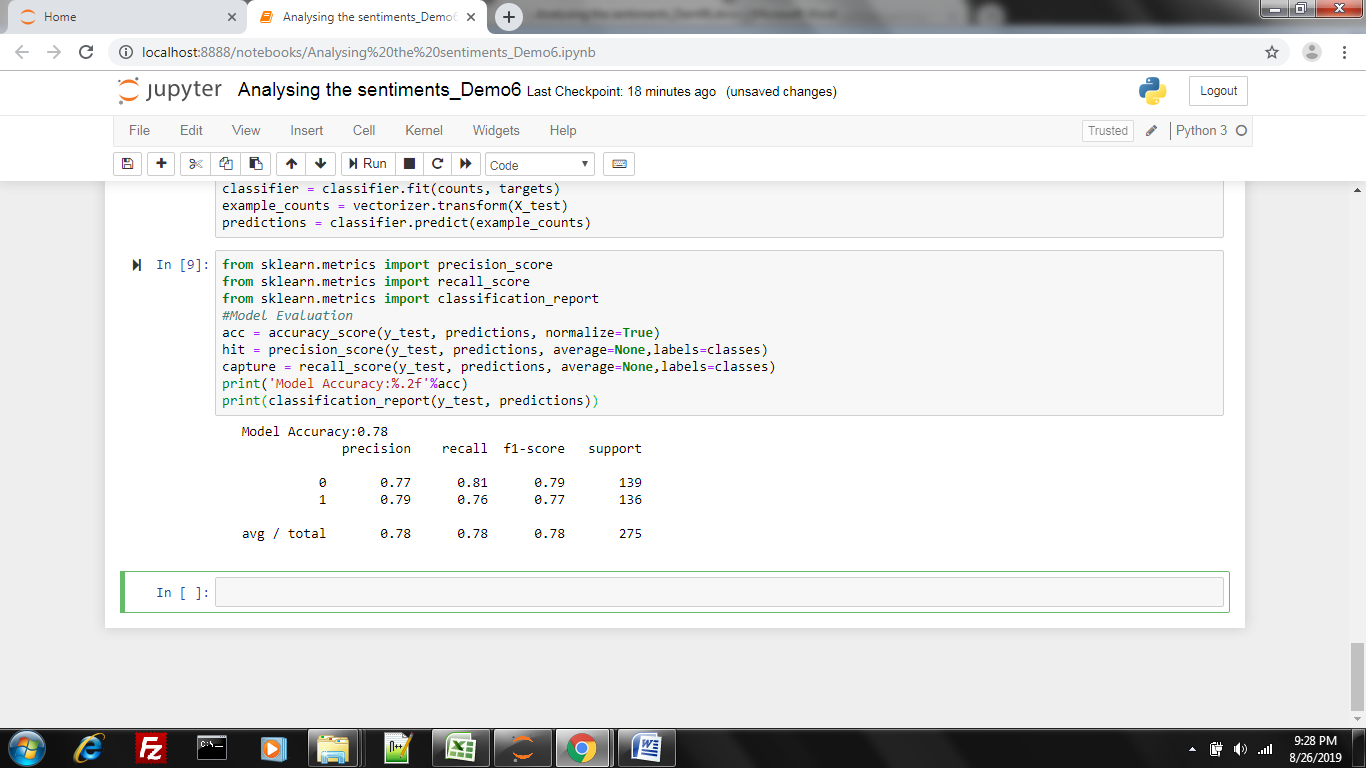
acc = accuracy\_score(y\_test, predictions, normalize=True)

hit = precision\_score(y\_test, predictions, average=None,labels=classes)

capture = recall\_score(y\_test, predictions, average=None,labels=classes)

print('Model Accuracy:%.2f'%acc)

print(classification\_report(y\_test, predictions))



This is how we can perform sentimental analysis using NLP.