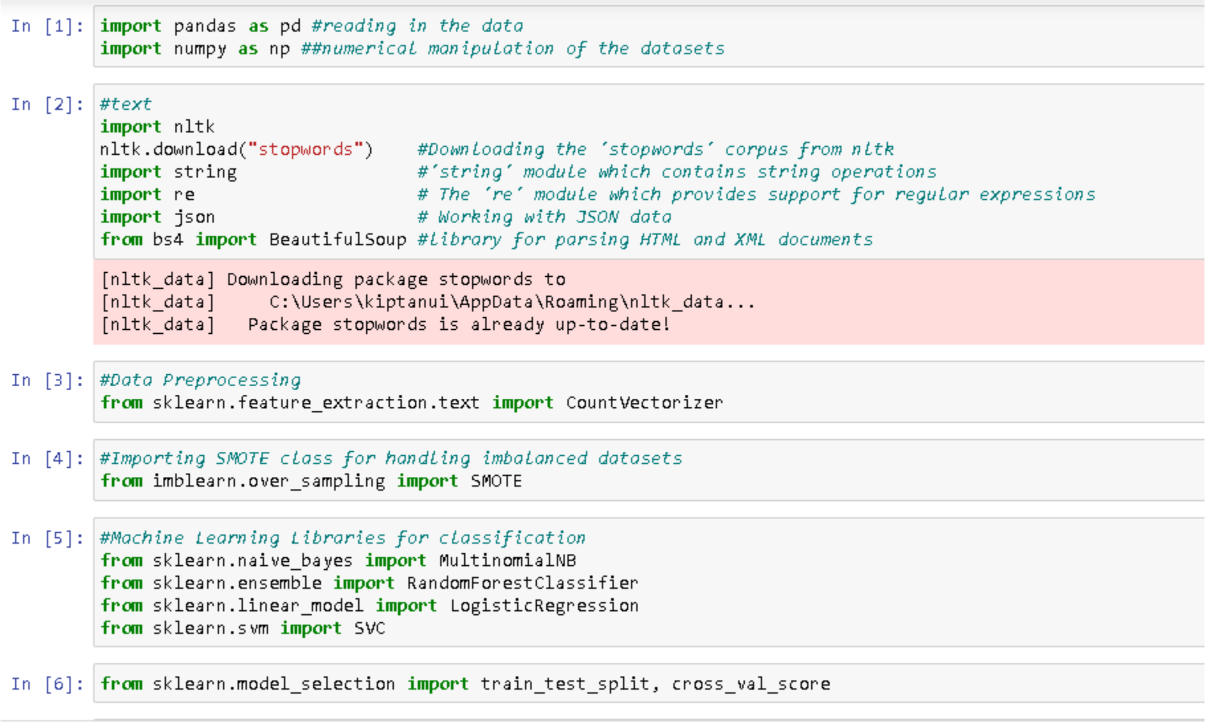
Name: Walter Kiptanui Rotich

Batch Number: LISUM29

Submission Date: 03/02/2024

Submitted To: Data Glacier

Implementing a classification model



```
In [1]: import pandas as pd #reading in the data
        import numpy as np ##numerical manipulation of the datasets
```

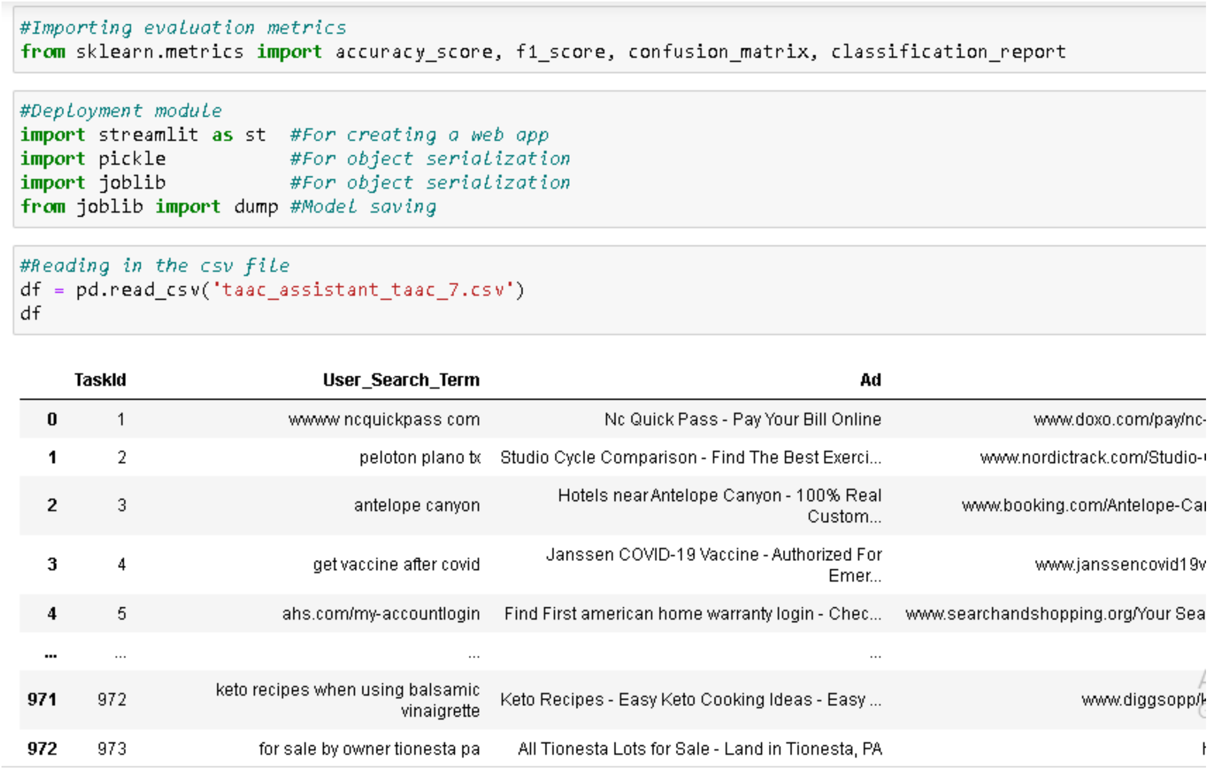
```
In [2]: #text
        import nltk
        nltk.download("stopwords")    #Downloading the 'stopwords' corpus from nltk
        import string                 #'string' module which contains string operations
        import re                     # The 're' module which provides support for regular expressions
        import json                   # Working with JSON data
        from bs4 import BeautifulSoup #Library for parsing HTML and XML documents

        [nltk_data] Downloading package stopwords to
        [nltk_data]     C:\Users\kiptanui\AppData\Roaming\nltk_data...
        [nltk_data]   Package stopwords is already up-to-date!
```

```
In [3]: #Data Preprocessing
        from sklearn.feature_extraction.text import CountVectorizer
```

```
In [4]: #Importing SMOTE class for handling imbalanced datasets
        from imblearn.over_sampling import SMOTE
```

```
In [5]: #Machine Learning Libraries for classification
        from sklearn.naive_bayes import MultinomialNB
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.linear_model import LogisticRegression
        from sklearn.svm import SVC
```

```
In [6]: from sklearn.model_selection import train_test_split, cross_val_score
```

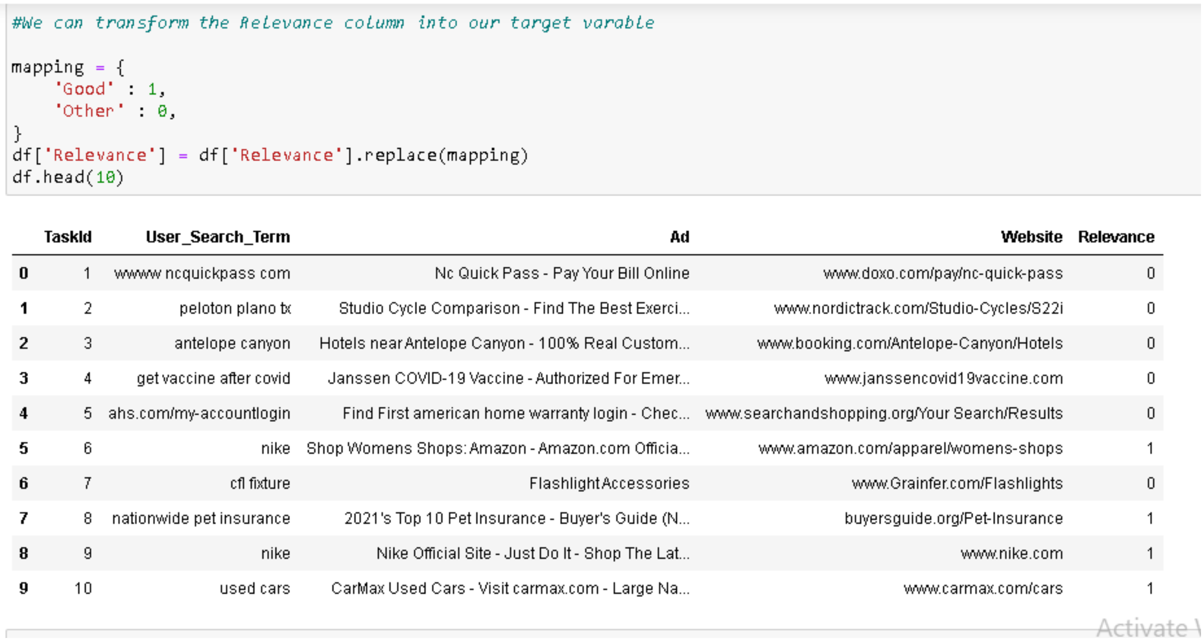


```python
#Importing evaluation metrics
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, classification_report
```

```python
#Deployment module
import streamlit as st  #For creating a web app
import pickle           #For object serialization
import joblib           #For object serialization
from joblib import dump #Model saving
```

```python
#Reading in the csv file
df = pd.read_csv('taac_assistant_taac_7.csv')
df
```

	TaskId	User_Search_Term	Ad	
0	1	wwww ncquickpass com	Nc Quick Pass - Pay Your Bill Online	www.doxo.com/pay/nc-
1	2	peloton plano tx	Studio Cycle Comparison - Find The Best Exerci...	www.nordictrack.com/Studio-
2	3	antelope canyon	Hotels near Antelope Canyon - 100% Real Custom...	www.booking.com/Antelope-Ca
3	4	get vaccine after covid	Janssen COVID-19 Vaccine - Authorized For Emer...	www.janssencovid19v
4	5	ahs.com/my-accountlogin	Find First american home warranty login - Chec...	www.searchandshopping.org/Your Sea
...	...	...	...	
971	972	keto recipes when using balsamic vinaigrette	Keto Recipes - Easy Keto Cooking Ideas - Easy ...	www.diggsopp/
972	973	for sale by owner tionesta pa	All Tionesta Lots for Sale - Land in Tionesta, PA	



```python
#We can transform the Relevance column into our target varable

mapping = {
    'Good' : 1,
    'Other' : 0,
}
df['Relevance'] = df['Relevance'].replace(mapping)
df.head(10)
```

	TaskId	User_Search_Term	Ad	Website	Relevance
0	1	wwww ncquickpass com	Nc Quick Pass - Pay Your Bill Online	www.doxo.com/pay/nc-quick-pass	0
1	2	peloton plano tx	Studio Cycle Comparison - Find The Best Exerci...	www.nordictrack.com/Studio-Cycles/S22i	0
2	3	antelope canyon	Hotels near Antelope Canyon - 100% Real Custom...	www.booking.com/Antelope-Canyon/Hotels	0
3	4	get vaccine after covid	Janssen COVID-19 Vaccine - Authorized For Emer...	www.janssencovid19vaccine.com	0
4	5	ahs.com/my-accountlogin	Find First american home warranty login - Chec...	www.searchandshopping.org/Your Search/Results	0
5	6	nike	Shop Womens Shops: Amazon - Amazon.com Officia...	www.amazon.com/apparel/womens-shops	1
6	7	cfl fixture	FlashlightAccessories	www.Grainfer.com/Flashlights	0
7	8	nationwide pet insurance	2021's Top 10 Pet Insurance - Buyer's Guide (N...	buyersguide.org/Pet-Insurance	1
8	9	nike	Nike Official Site - Just Do It - Shop The Lat...	www.nike.com	1
9	10	used cars	CarMax Used Cars - Visit carmax.com - Large Na...	www.carmax.com/cars	1

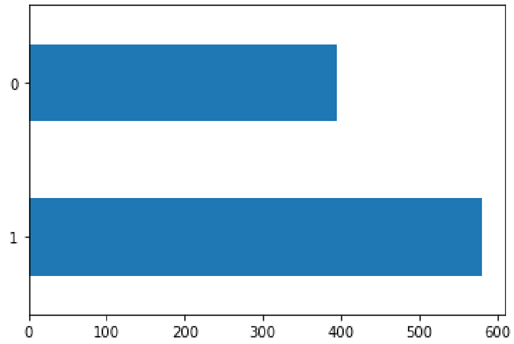
```
# Check for any missing values
df.isnull().sum().any()
```

```
False
```

```
#No missing values
```

```
#Visualization of the overall distribution of the classes
df["Relevance"].value_counts().plot(kind = 'barh')
```

```
<AxesSubplot:>
```



```
#The data is imbalanced, it has to be balanced
```

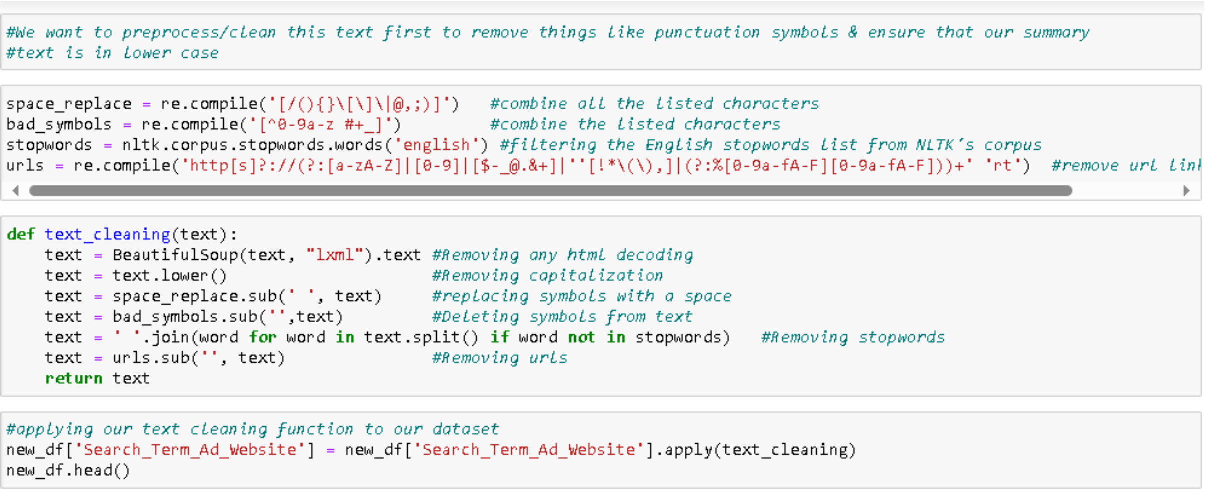
```
# Combine the three feature columns into one, separated by a space
df['Search_Term_Ad_Website'] = df['User_Search_Term'] + ' ' + df['Ad'] + ' ' + df['Website']

# Drop the individual feature columns
df = df.drop(['User_Search_Term', 'Ad', 'Website'], axis=1)

# Now, 'combined_features' will contain the combined text from the three columns
```

```
2024-01-21 15:54:02.952 INFO     numexpr.utils: NumExpr defaulting to 8 threads.
```

```
#Reading in the combined features dataset
df.head(5)
```

	TaskId	Relevance	Search_Term_Ad_Website
0	1	0	wwww ncquickpass com Nc Quick Pass - Pay Your ...
1	2	0	peloton plano tx Studio Cycle Comparison - Fin...
2	3	0	antelope canyon Hotels near Antelope Canyon - ...
3	4	0	get vaccine after covid Janssen COVID-19 Vacci...
4	5	0	ahs.com/my-accountlogin Find First american ho...

```
#Dropping an unncessary column
new_df = df.drop(columns = ['TaskId'])
new_df.head()
```

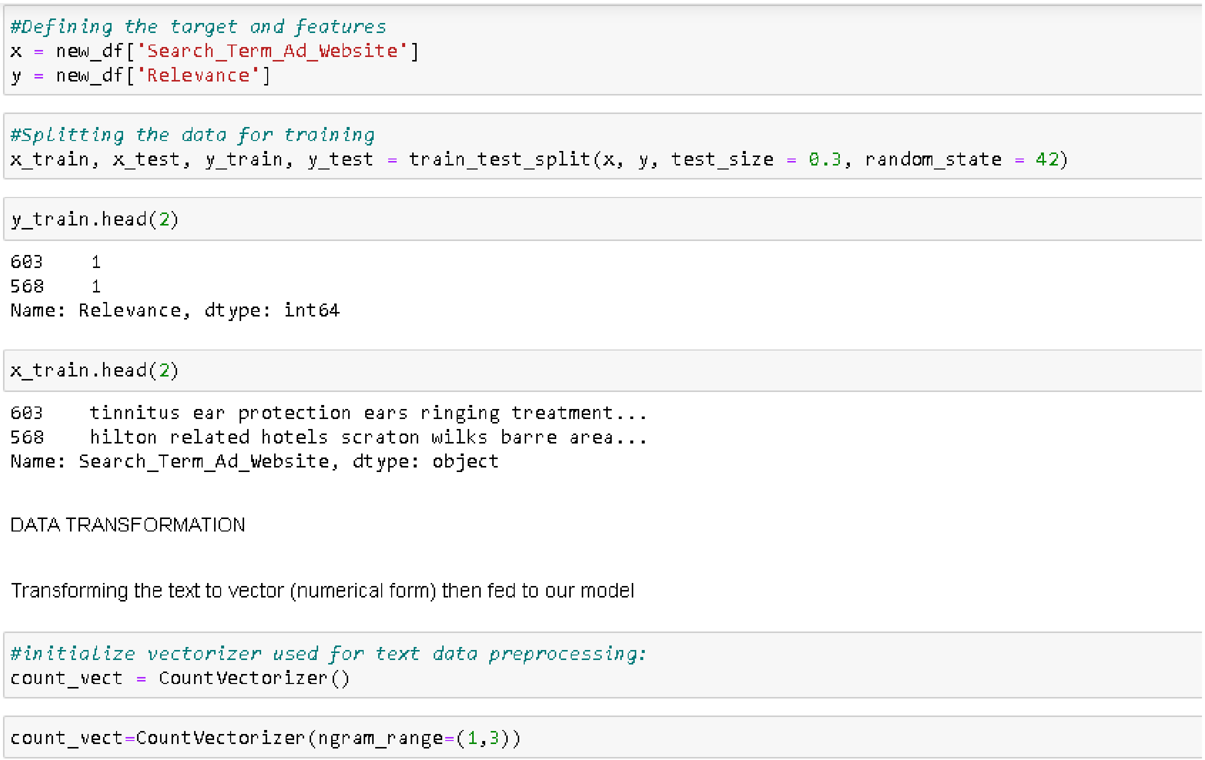


```
#We want to preprocess/clean this text first to remove things like punctuation symbols & ensure that our summary
#text is in lower case
```

```
space_replace = re.compile('[/(){}\[\]\|@,;)]')   #combine all the listed characters
bad_symbols = re.compile('[^0-9a-z #+_]')         #combine the listed characters
stopwords = nltk.corpus.stopwords.words('english') #filtering the English stopwords list from NLTK's corpus
urls = re.compile('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|''[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+' 'rt')  #remove url link
```

```
def text_cleaning(text):
    text = BeautifulSoup(text, "lxml").text #Removing any html decoding
    text = text.lower()                     #Removing capitalization
    text = space_replace.sub(' ', text)     #replacing symbols with a space
    text = bad_symbols.sub('',text)         #Deleting symbols from text
    text = ' '.join(word for word in text.split() if word not in stopwords)    #Removing stopwords
    text = urls.sub('', text)               #Removing urls
    return text
```

```
#applying our text cleaning function to our dataset
new_df['Search_Term_Ad_Website'] = new_df['Search_Term_Ad_Website'].apply(text_cleaning)
new_df.head()
```



```
#Defining the target and features
x = new_df['Search_Term_Ad_Website']
y = new_df['Relevance']
```

```
#Splitting the data for training
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 42)
```

```
y_train.head(2)
```

```
603    1
568    1
Name: Relevance, dtype: int64
```

```
x_train.head(2)
```

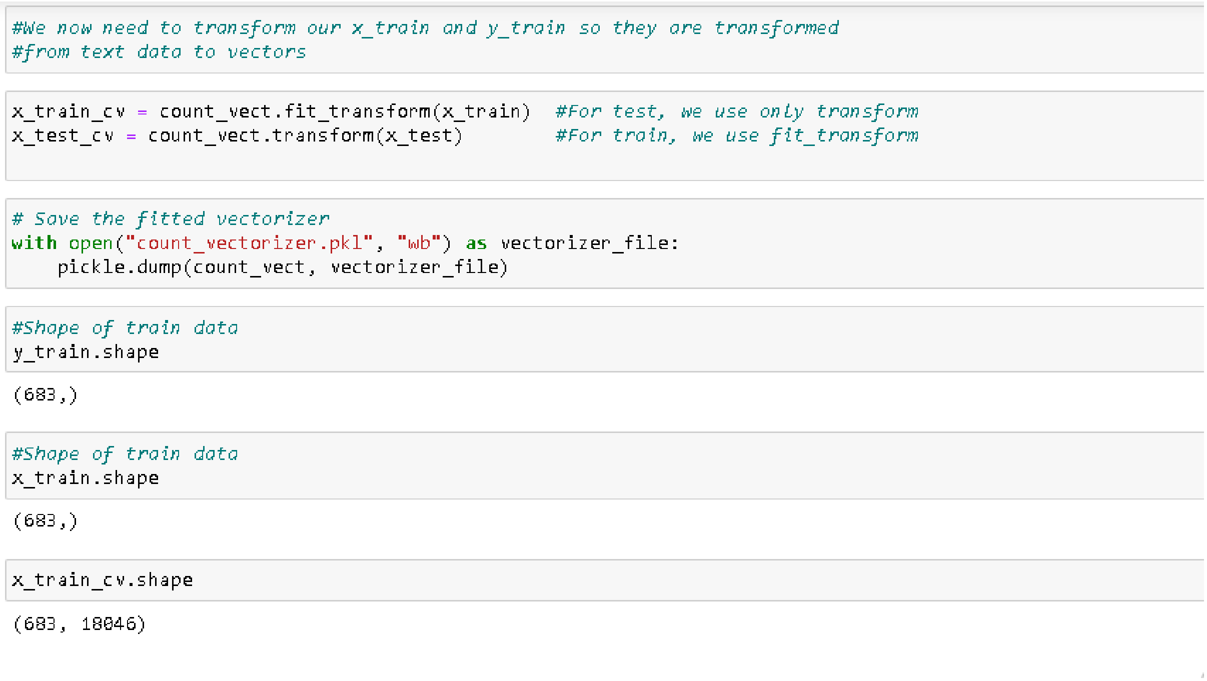
```
603    tinnitus ear protection ears ringing treatment...
568    hilton related hotels scraton wilks barre area...
Name: Search_Term_Ad_Website, dtype: object
```

DATA TRANSFORMATION

Transforming the text to vector (numerical form) then fed to our model

```
#initialize vectorizer used for text data preprocessing:
count_vect = CountVectorizer()
```

```
count_vect=CountVectorizer(ngram_range=(1,3))
```



```
#We now need to transform our x_train and y_train so they are transformed
#from text data to vectors
```

```
x_train_cv = count_vect.fit_transform(x_train)  #For test, we use only transform
x_test_cv = count_vect.transform(x_test)        #For train, we use fit_transform
```

```
# Save the fitted vectorizer
with open("count_vectorizer.pkl", "wb") as vectorizer_file:
    pickle.dump(count_vect, vectorizer_file)
```

```
#Shape of train data
y_train.shape
```

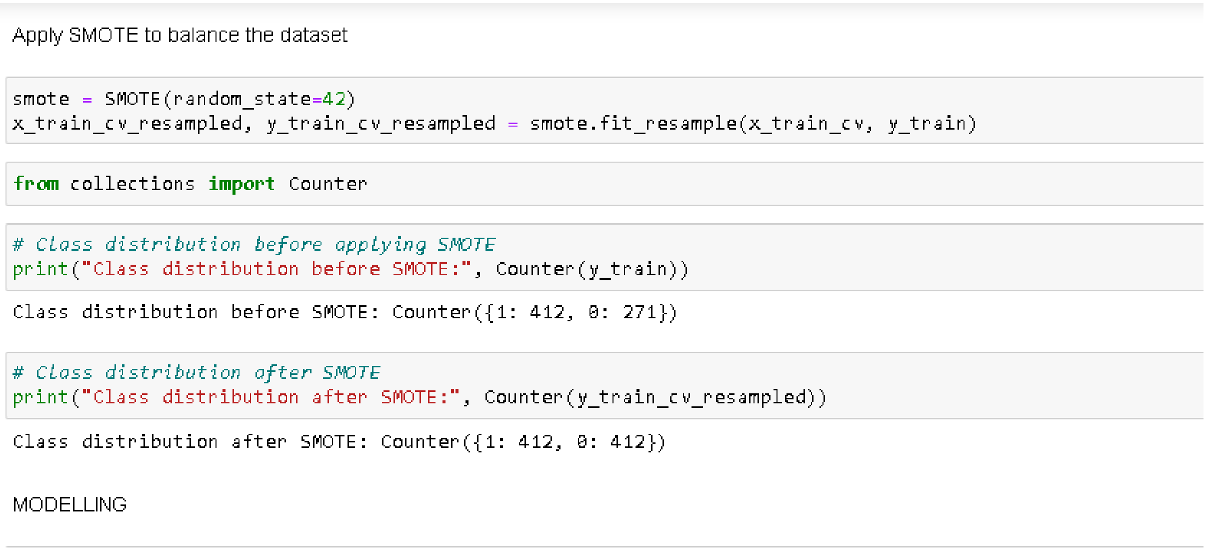
```
(683,)
```

```
#Shape of train data
x_train.shape
```

```
(683,)
```

```
x_train_cv.shape
```

```
(683, 18046)
```



Apply SMOTE to balance the dataset

```
smote = SMOTE(random_state=42)
x_train_cv_resampled, y_train_cv_resampled = smote.fit_resample(x_train_cv, y_train)
```

```
from collections import Counter
```

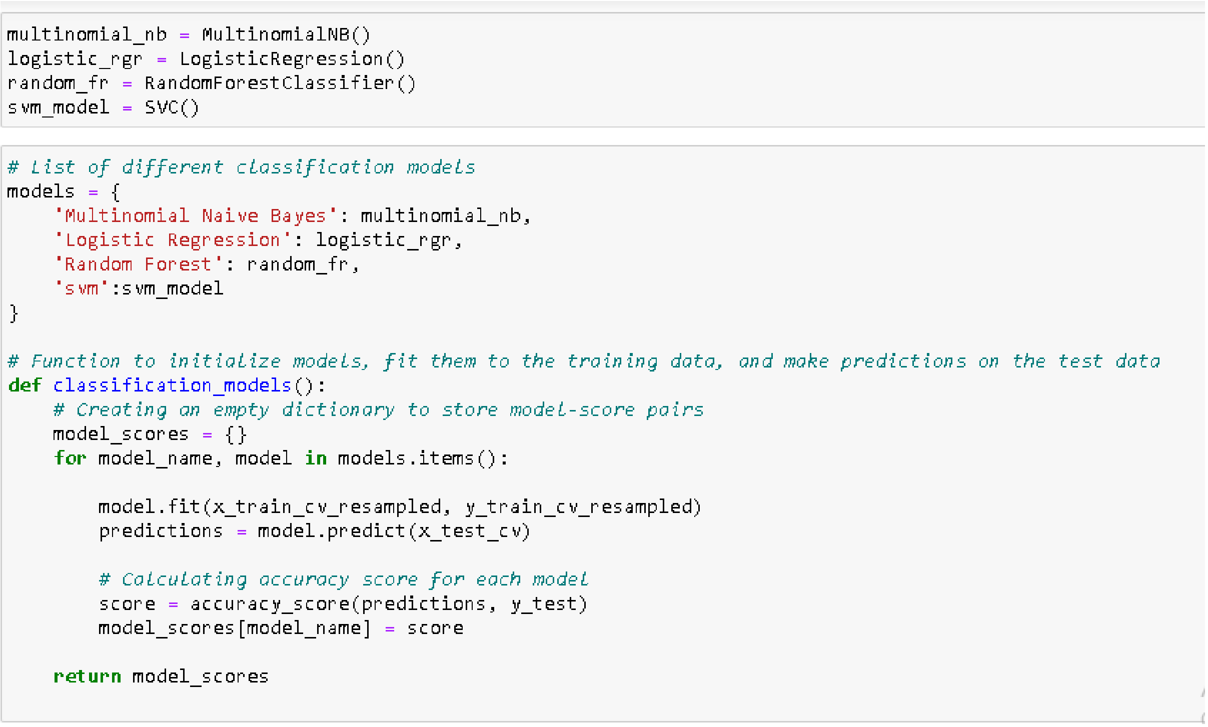
```
# Class distribution before applying SMOTE
print("Class distribution before SMOTE:", Counter(y_train))
```

```
Class distribution before SMOTE: Counter({1: 412, 0: 271})
```

```
# Class distribution after SMOTE
print("Class distribution after SMOTE:", Counter(y_train_cv_resampled))
```

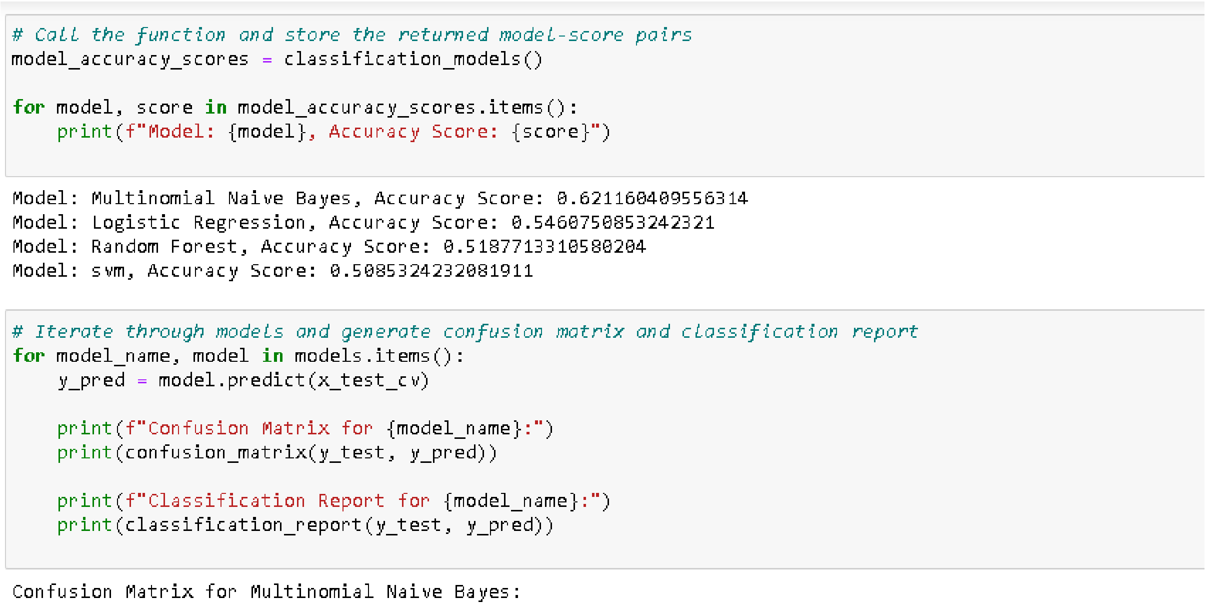
```
Class distribution after SMOTE: Counter({1: 412, 0: 412})
```

MODELLING



```python
multinomial_nb = MultinomialNB()
logistic_rgr = LogisticRegression()
random_fr = RandomForestClassifier()
svm_model = SVC()
```

```python
# List of different classification models
models = {
    'Multinomial Naive Bayes': multinomial_nb,
    'Logistic Regression': logistic_rgr,
    'Random Forest': random_fr,
    'svm':svm_model
}

# Function to initialize models, fit them to the training data, and make predictions on the test data
def classification_models():
    # Creating an empty dictionary to store model-score pairs
    model_scores = {}
    for model_name, model in models.items():

        model.fit(x_train_cv_resampled, y_train_cv_resampled)
        predictions = model.predict(x_test_cv)

        # Calculating accuracy score for each model
        score = accuracy_score(predictions, y_test)
        model_scores[model_name] = score

    return model_scores
```

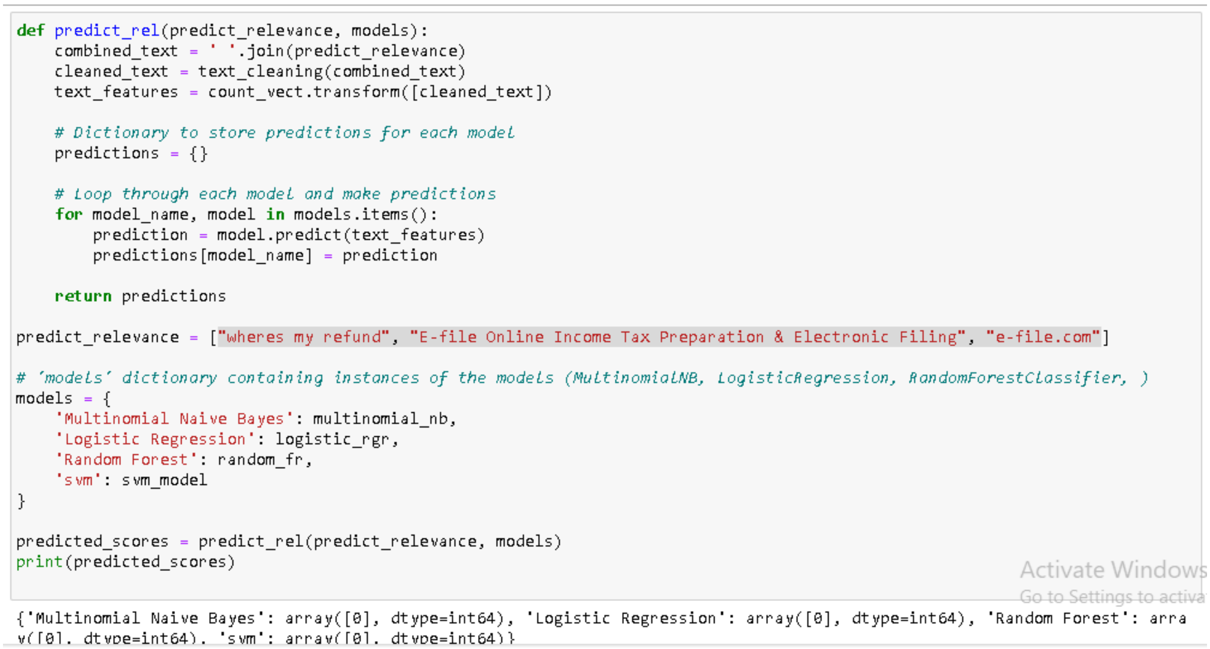


```python
# Call the function and store the returned model-score pairs
model_accuracy_scores = classification_models()

for model, score in model_accuracy_scores.items():
    print(f"Model: {model}, Accuracy Score: {score}")
```

```
Model: Multinomial Naive Bayes, Accuracy Score: 0.621160409556314
Model: Logistic Regression, Accuracy Score: 0.5460750853242321
Model: Random Forest, Accuracy Score: 0.5187713310580204
Model: svm, Accuracy Score: 0.5085324232081911
```

```python
# Iterate through models and generate confusion matrix and classification report
for model_name, model in models.items():
    y_pred = model.predict(x_test_cv)

    print(f"Confusion Matrix for {model_name}:")
    print(confusion_matrix(y_test, y_pred))

    print(f"Classification Report for {model_name}:")
    print(classification_report(y_test, y_pred))
```

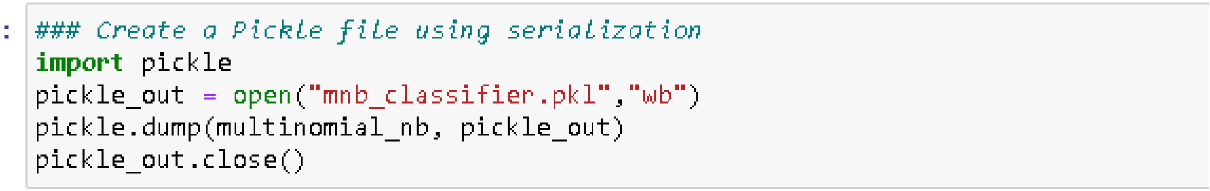
```
Confusion Matrix for Multinomial Naive Bayes:
```



```
def predict_rel(predict_relevance, models):
    combined_text = ' '.join(predict_relevance)
    cleaned_text = text_cleaning(combined_text)
    text_features = count_vect.transform([cleaned_text])

    # Dictionary to store predictions for each model
    predictions = {}

    # Loop through each model and make predictions
    for model_name, model in models.items():
        prediction = model.predict(text_features)
        predictions[model_name] = prediction

    return predictions

predict_relevance = ["wheres my refund", "E-file Online Income Tax Preparation & Electronic Filing", "e-file.com"]

# 'models' dictionary containing instances of the models (MultinomialNB, LogisticRegression, RandomForestClassifier, )
models = {
    'Multinomial Naive Bayes': multinomial_nb,
    'Logistic Regression': logistic_rgr,
    'Random Forest': random_fr,
    'svm': svm_model
}

predicted_scores = predict_rel(predict_relevance, models)
print(predicted_scores)
```

```
{'Multinomial Naive Bayes': array([0], dtype=int64), 'Logistic Regression': array([0], dtype=int64), 'Random Forest': arra
y([0], dtype=int64), 'svm': array([0], dtype=int64)}
```

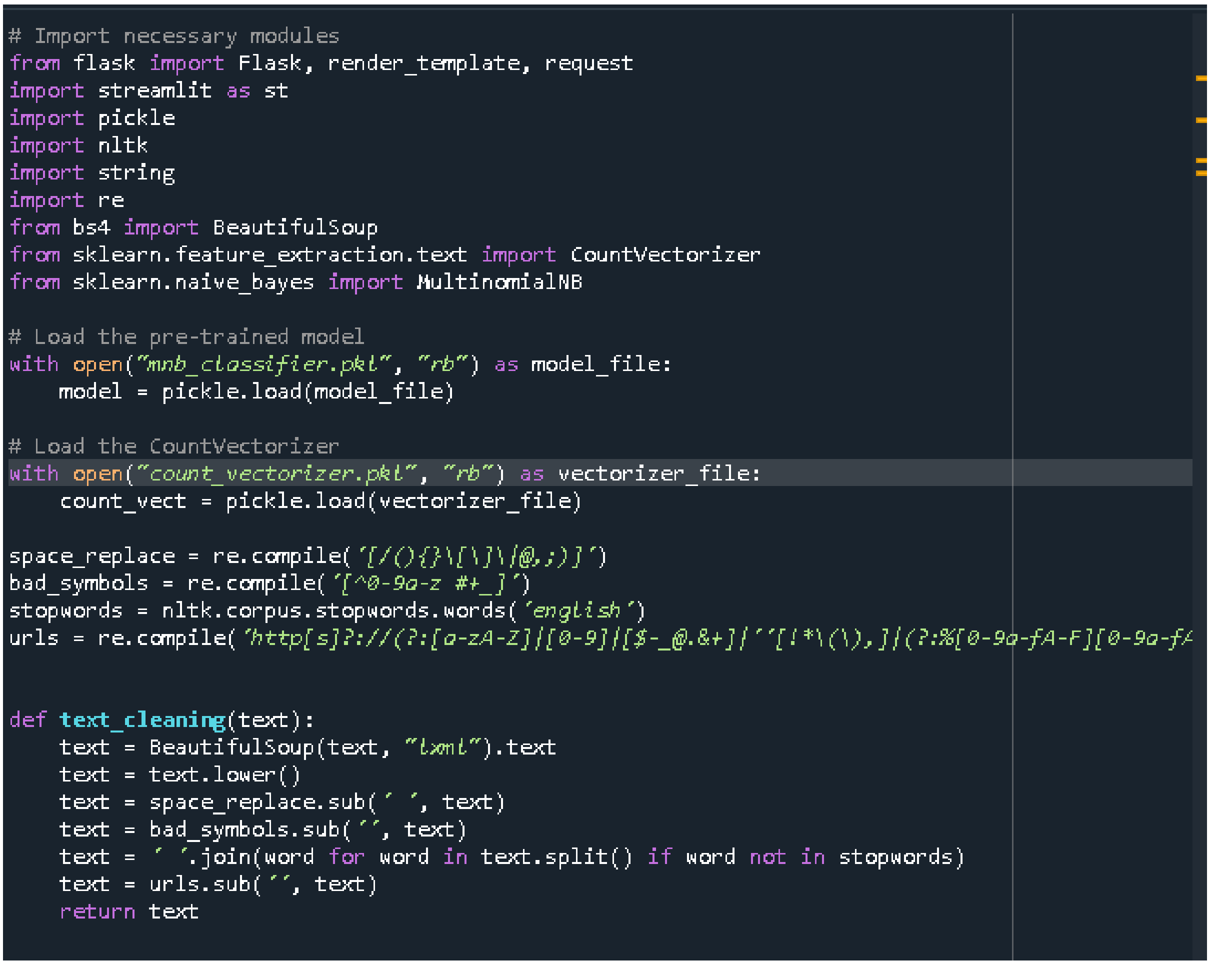
Saving the Model



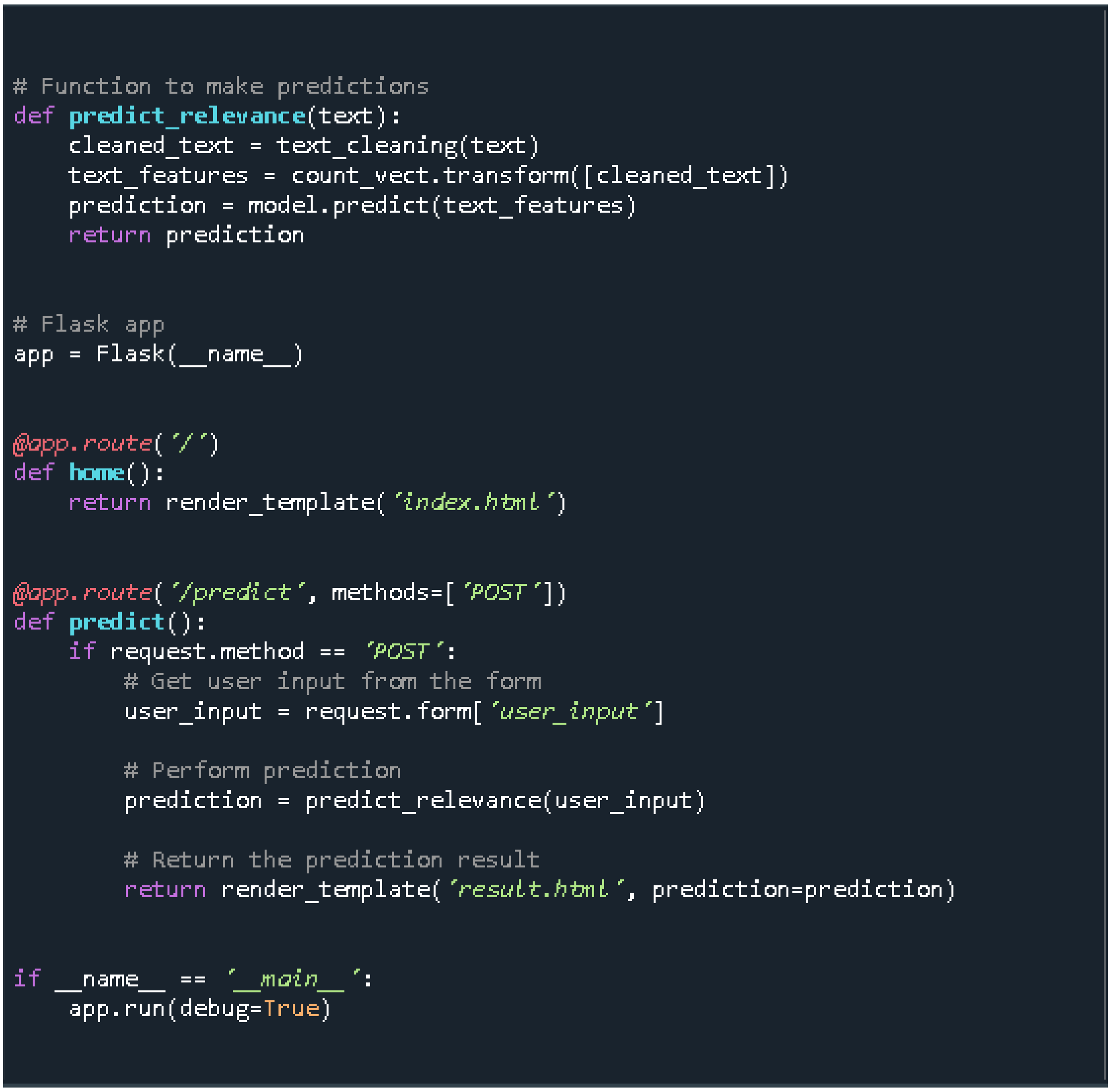
```
:   ### Create a Pickle file using serialization
    import pickle
    pickle_out = open("mnb_classifier.pkl","wb")
    pickle.dump(multinomial_nb, pickle_out)
    pickle_out.close()
```

3. Deployment the model on flask (web app)

i. Flask_app.py

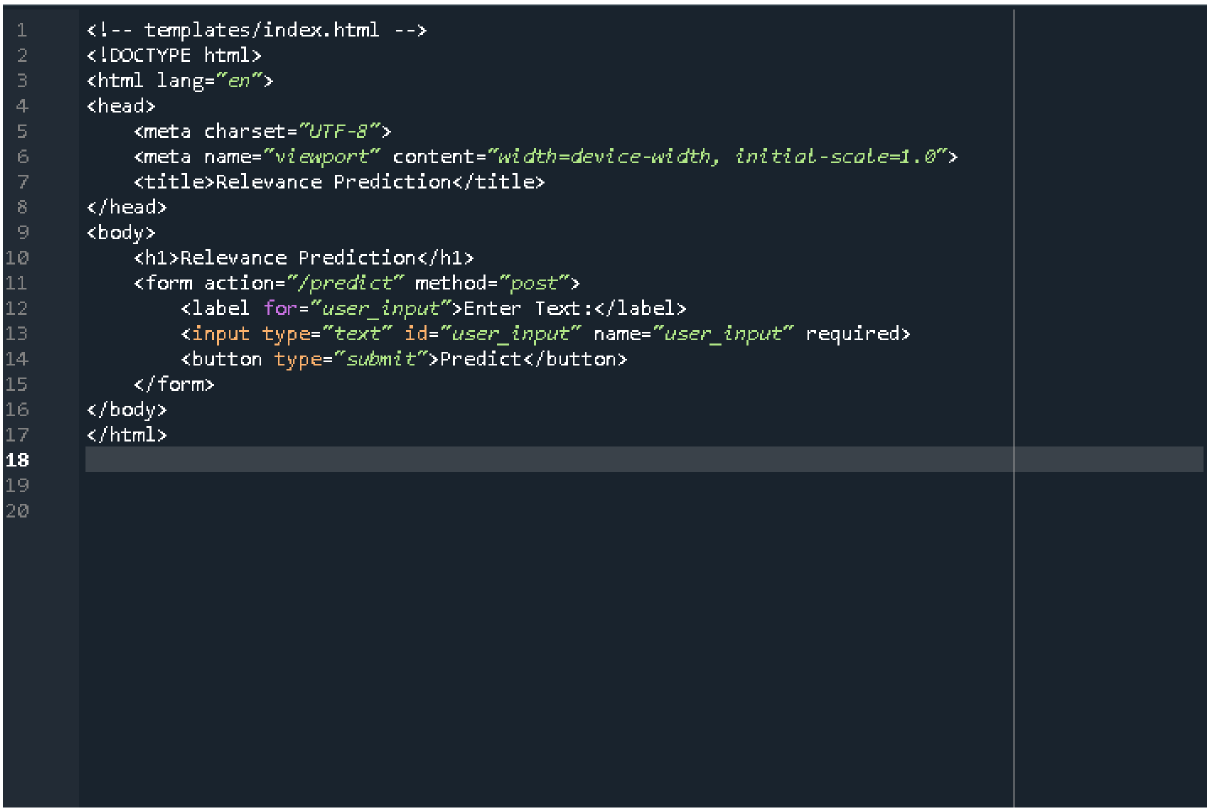


```
# Import necessary modules
from flask import Flask, render_template, request
import streamlit as st
import pickle
import nltk
import string
import re
from bs4 import BeautifulSoup
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB

# Load the pre-trained model
with open("mnb_classifier.pkl", "rb") as model_file:
    model = pickle.load(model_file)

# Load the CountVectorizer
with open("count_vectorizer.pkl", "rb") as vectorizer_file:
    count_vect = pickle.load(vectorizer_file)

space_replace = re.compile('[/(){}\[\]\|@,;)]')
bad_symbols = re.compile('[^0-9a-z #+_]')
stopwords = nltk.corpus.stopwords.words('english')
urls = re.compile('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|''[!*\(\),]|(?:%[0-9a-fA-F][0-9a-f


def text_cleaning(text):
    text = BeautifulSoup(text, "lxml").text
    text = text.lower()
    text = space_replace.sub(' ', text)
    text = bad_symbols.sub('', text)
    text = ' '.join(word for word in text.split() if word not in stopwords)
    text = urls.sub('', text)
    return text
```



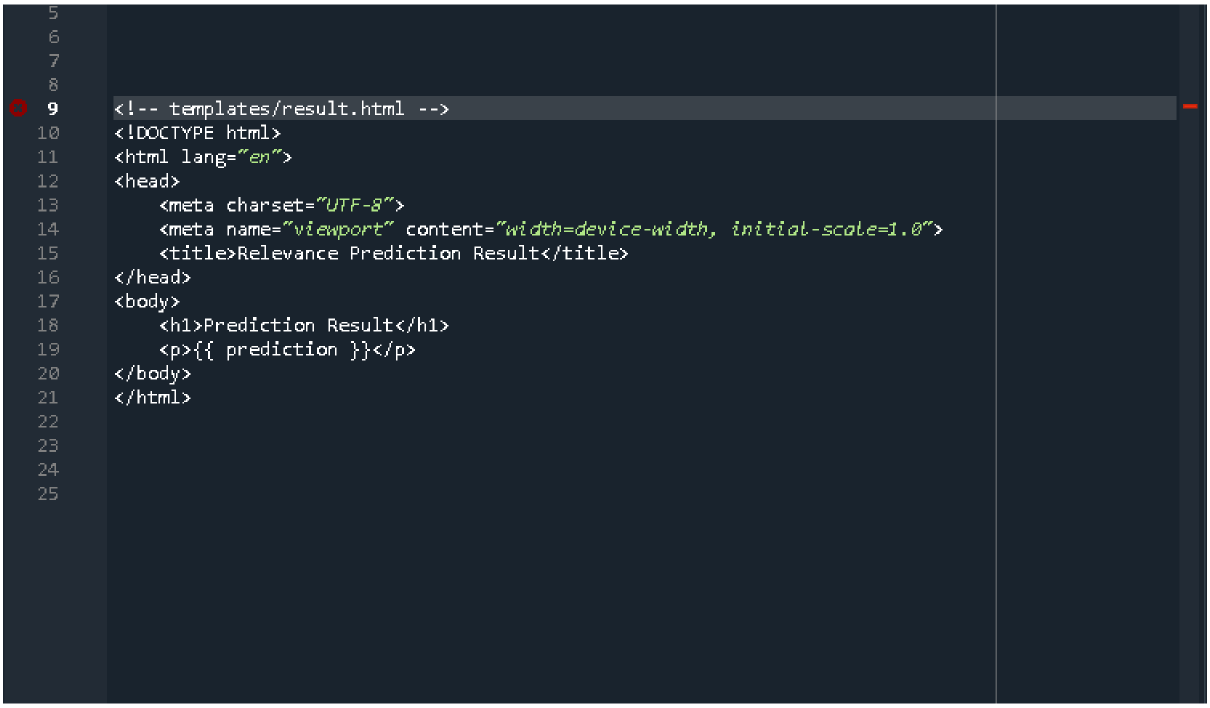
```
# Function to make predictions
def predict_relevance(text):
    cleaned_text = text_cleaning(text)
    text_features = count_vect.transform([cleaned_text])
    prediction = model.predict(text_features)
    return prediction


# Flask app
app = Flask(__name__)


@app.route('/')
def home():
    return render_template('index.html')


@app.route('/predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        # Get user input from the form
        user_input = request.form['user_input']

        # Perform prediction
        prediction = predict_relevance(user_input)

        # Return the prediction result
        return render_template('result.html', prediction=prediction)


if __name__ == '__main__':
    app.run(debug=True)
```

ii. Index.html



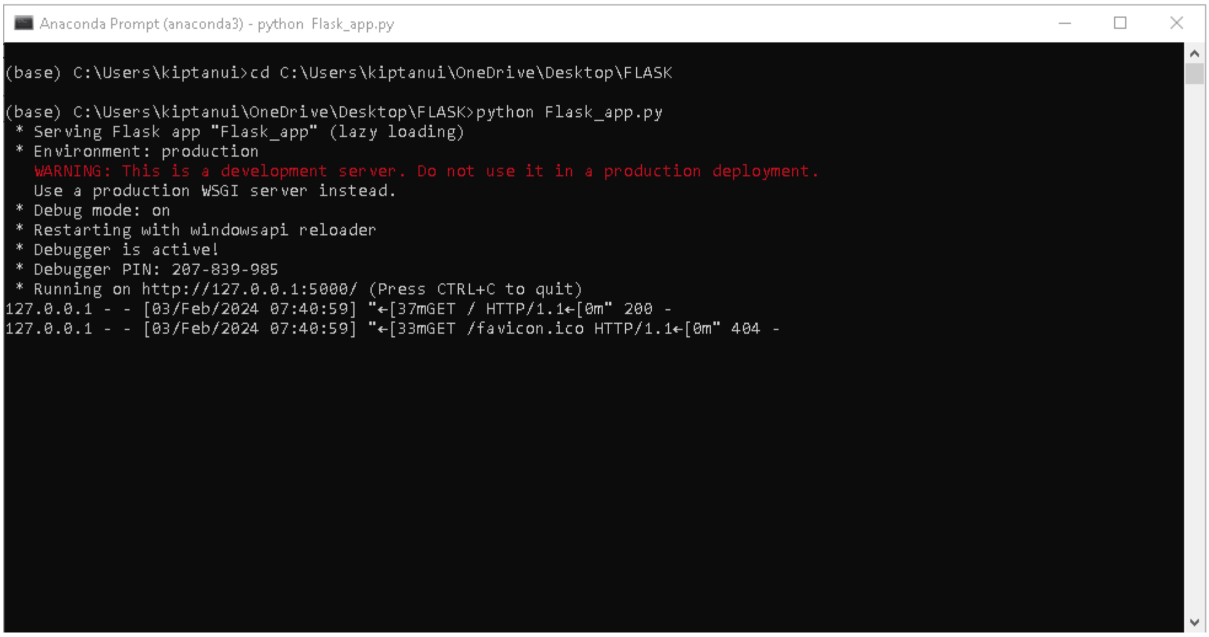
```html
<!-- templates/index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Relevance Prediction</title>
</head>
<body>
    <h1>Relevance Prediction</h1>
    <form action="/predict" method="post">
        <label for="user_input">Enter Text:</label>
        <input type="text" id="user_input" name="user_input" required>
        <button type="submit">Predict</button>
    </form>
</body>
</html>
```

iii. Result.html

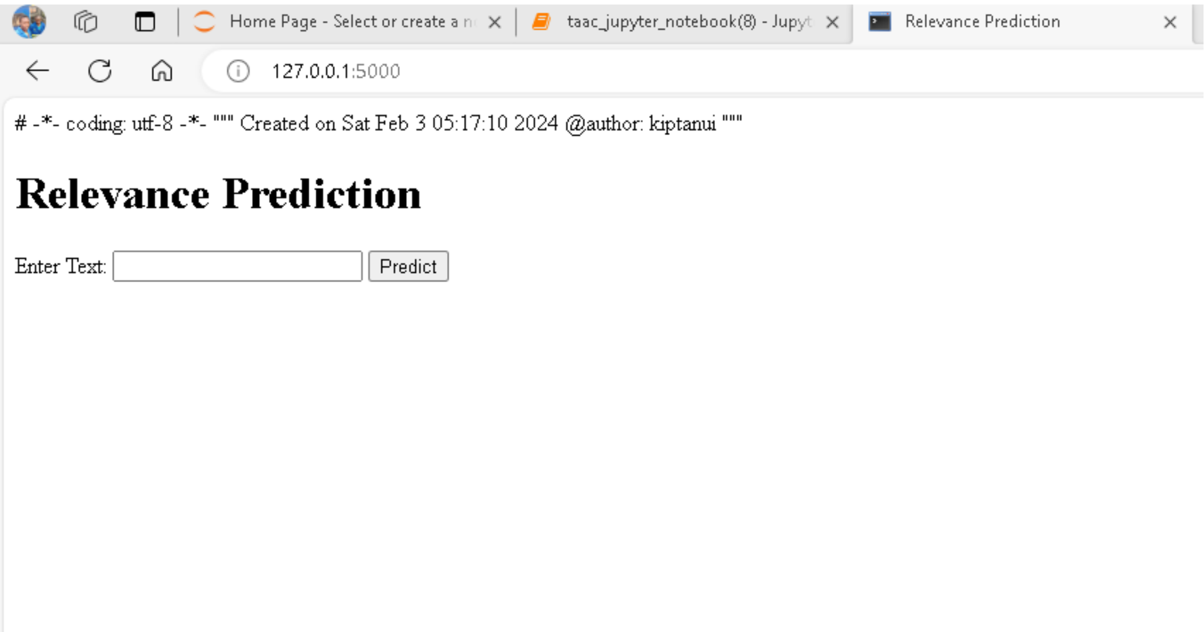


```html
<!-- templates/result.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Relevance Prediction Result</title>
</head>
<body>
    <h1>Prediction Result</h1>
    <p>{{ prediction }}</p>
</body>
</html>
```

Prediction on flask web app.



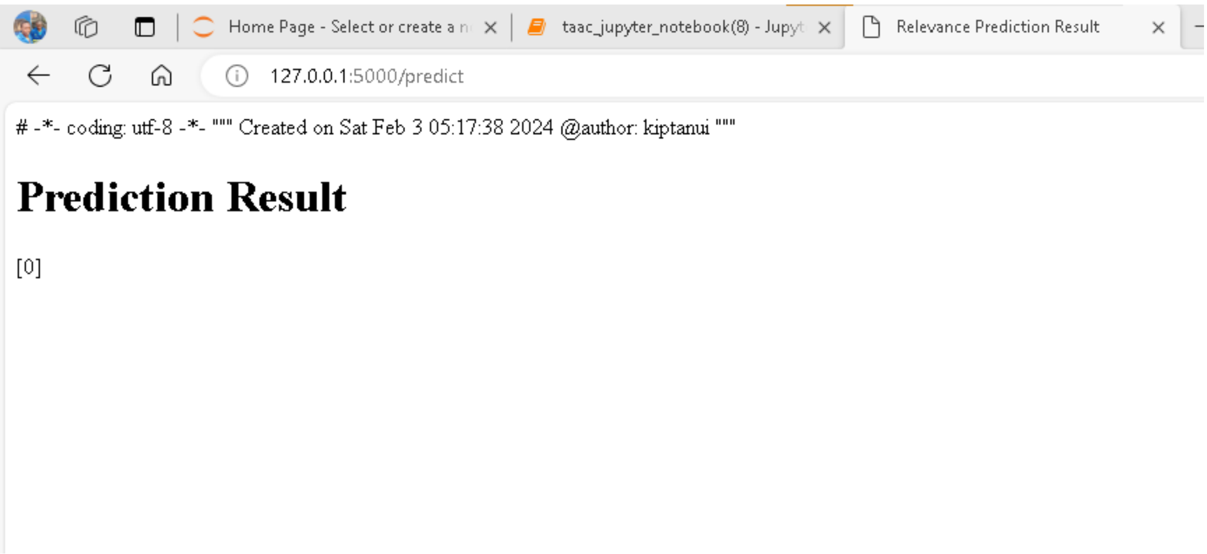
```
(base) C:\Users\kiptanui>cd C:\Users\kiptanui\OneDrive\Desktop\FLASK

(base) C:\Users\kiptanui\OneDrive\Desktop\FLASK>python Flask_app.py
 * Serving Flask app "Flask_app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with windowsapi reloader
 * Debugger is active!
 * Debugger PIN: 207-839-985
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [03/Feb/2024 07:40:59] "←[37mGET / HTTP/1.1←[0m" 200 -
127.0.0.1 - - [03/Feb/2024 07:40:59] "←[33mGET /favicon.ico HTTP/1.1←[0m" 404 -
```


Home Page - Select or create a n
taac_jupyter_notebook(8) - Jupyt
Relevance Prediction
127.0.0.1:5000

-*- coding: utf-8 -*- """ Created on Sat Feb 3 05:17:10 2024 @author: kiptanui """

Relevance Prediction

Enter Text: [ ] Predict



Home Page - Select or create a n
taac_jupyter_notebook(8) - Jupyt
Relevance Prediction Result
127.0.0.1:5000/predict

-*- coding: utf-8 -*- """ Created on Sat Feb 3 05:17:38 2024 @author: kiptanui """

Prediction Result

[0]