In [109…
```
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import nltk
from nltk.corpus import stopwords
from sklearn.preprocessing import LabelEncoder
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
import re
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split, cross_val_score
os.chdir("C:/Users/shaik/.jupyter")
info=pd.read_csv("d_d(DS).csv")
```

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

In [110…
```
info.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7613 entries, 0 to 7612
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   tweets  7613 non-null   object
 1   target  7613 non-null   int64
dtypes: int64(1), object(1)
memory usage: 119.1+ KB
```

In [111…
```
info.columns = ["tweets", "target"]
```

In [112…
```
print(info.head())
```

```
                                              tweets  target
0  Our Deeds are the Reason of this #earthquake M...       1
1             Forest fire near La Ronge Sask. Canada       1
2  All residents asked to 'shelter in place' are ...       1
3  13,000 people receive #wildfires evacuation or...       1
4  Just got sent this photo from Ruby #Alaska as ...       1
```

In [113…
```
# Check for null values
null_values = info.isnull().sum()
print(null_values)
```

```
tweets    0
target    0
dtype: int64
```

In [114…
```
# Remove rows with null values
info = info.dropna()
print(info)
```

```
                                                 tweets  target
0     Our Deeds are the Reason of this #earthquake M...       1
1                Forest fire near La Ronge Sask. Canada       1
2     All residents asked to 'shelter in place' are ...       1
3     13,000 people receive #wildfires evacuation or...       1
4     Just got sent this photo from Ruby #Alaska as ...       1
...                                                 ...     ...
7608  Two giant cranes holding a bridge collapse int...       1
7609  @aria_ahrary @TheTawniest The out of control w...       1
7610  M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt...       1
7611  Police investigating after an e-bike collided ...       1
7612  The Latest: More Homes Razed by Northern Calif...       1

[7613 rows x 2 columns]
```

In [115…
```
# Fill null values in a specific column (e.g., 'target') with the mean
info['target'].fillna(info['target'].mean(), inplace=True)
```

In [116…
```
selected_columns = ['tweets']
```

In [117…
```
selected_data = info[selected_columns]
```

In [118…
```
print(selected_data)
```

```
                                                 tweets
0     Our Deeds are the Reason of this #earthquake M...
1                Forest fire near La Ronge Sask. Canada
2     All residents asked to 'shelter in place' are ...
3     13,000 people receive #wildfires evacuation or...
4     Just got sent this photo from Ruby #Alaska as ...
...                                                 ...
7608  Two giant cranes holding a bridge collapse int...
7609  @aria_ahrary @TheTawniest The out of control w...
7610  M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt...
7611  Police investigating after an e-bike collided ...
7612  The Latest: More Homes Razed by Northern Calif...

[7613 rows x 1 columns]
```

In [119…
```
sentences = selected_data.apply(lambda row: ', '.join(row.astype(str)), axis=1)
```

In [120…
```
# for sentence in sentences:
#     print(sentence)
```

In [121…
```
# Tokenize the text into words
words = word_tokenize(sentence)
print(words)
```

```
['The', 'Latest', ':', 'More', 'Homes', 'Razed', 'by', 'Northern', 'California', 'Wildfire', '-', 'ABC', 'News', 'http', ':', '//t.co/YmY4rSkQ3d']
```

In [122…
```
# Convert the text to lowercase
lowercase_text = sentence.lower()
print(lowercase_text)
```

```
the latest: more homes razed by northern california wildfire - abc news http://t.co/ymy4rskq3d
```

In [123…
```
# Removing puntuations
re_pun = re.sub("[^A-Za-z0-9]"," ",sentence)
print(re_pun)
```

```
The Latest  More Homes Razed by Northern California Wildfire   ABC News http   t co YmY4rSkQ3d
```

In [124…
```
# Tokenize the text into words
words = nltk.word_tokenize(sentence)

# Define a list of English stopwords
stop_words = set(stopwords.words('english'))

# Remove stopwords from the tokenized words
filtered_words = [word for word in words if word.lower() not in stop_words]

# Join the filtered words into a sentence
filtered_text = ' '.join(filtered_words)

# Print the text without stop words
print(filtered_text)
```

```
Latest : Homes Razed Northern California Wildfire - ABC News http : //t.co/YmY4rSkQ3d
```

In [125…
```
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

# Perform stemming on the tokenized words
stemmed_words = [stemmer.stem(word) for word in words]

# Perform lemmatization on the tokenized words
lemmatized_words = [lemmatizer.lemmatize(word) for word in words]

# Print the stemmed and lemmatized words
print("Stemmed words:", stemmed_words)
print("Lemmatized words:", lemmatized_words)
```

```
Stemmed words: ['the', 'latest', ':', 'more', 'home', 'raze', 'by', 'northern', 'california', 'wildfir', '-', 'abc', 'new', 'http', ':', '//t.co/ymy4rskq3d']
Lemmatized words: ['The', 'Latest', ':', 'More', 'Homes', 'Razed', 'by', 'Northern', 'California', 'Wildfire', '-', 'ABC', 'News', 'http', ':', '//t.co/YmY4rSkQ3d']
```

In [126…
```
# Initialize TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=5000)

# Transform the tweets into TF-IDF vectors
X = tfidf_vectorizer.fit_transform(info['tweets'])
```

In [127…
```

y = info['target']
print(X)
print(y)
```

```
  (0, 259)	0.22934460014171543
  (0, 4636)	0.26954223522594756
  (0, 1737)	0.46109772655004433
  (0, 260)	0.398291882605956
  (0, 2716)	0.2854410541967898
  (0, 1414)	0.31983093238985116
  (0, 4416)	0.1982297826595741
  (0, 3027)	0.1320326140509247
  (0, 3542)	0.3595883496673916
  (0, 4395)	0.11198025927232072
  (0, 346)	0.2085353799174595
  (0, 3103)	0.27870065955172785
  (1, 744)	0.5320305661213975
  (1, 2473)	0.4958432886180763
  (1, 2929)	0.4322914462990095
  (1, 1675)	0.32674179337541426
  (1, 1733)	0.42124436585369884
  (2, 1554)	0.22573526118502604
  (2, 3091)	0.23479856627992443
  (2, 3086)	0.1461632214844059
  (2, 1524)	0.18983785382797244
  (2, 3099)	0.19686789942060762
  (2, 2968)	0.13900740890739524
  (2, 3036)	0.24757257735680932
  (2, 710)	0.116060227320936
  :	:
  (7611, 4427)	0.263097242087632
  (7611, 2312)	0.23989056176491008
  (7611, 223)	0.14693331335892051
  (7611, 3884)	0.2443612145044651
  (7611, 2585)	0.20241850659857624
  (7611, 3668)	0.2727288846553381
  (7611, 2557)	0.1816929732347834
  (7611, 4868)	0.12289038953673968
  (7611, 3295)	0.16975143811027446
  (7611, 760)	0.18206292354720247
  (7611, 288)	0.14799206285004485
  (7611, 2239)	0.0819518393655636
  (7612, 3516)	0.36429314556195475
  (7612, 2122)	0.3304816365839914
  (7612, 4838)	0.3201701855586726
  (7612, 167)	0.3756423578265971
  (7612, 2978)	0.3259547091343627
  (7612, 2501)	0.3304816365839914
  (7612, 2851)	0.2582990718942193
  (7612, 2949)	0.2655886836780667
  (7612, 907)	0.09324492463172784
  (7612, 2152)	0.09885743929128046
  (7612, 725)	0.2917530049663031
  (7612, 710)	0.20834898003631017
  (7612, 4395)	0.12140591680777017
0       1
1       1
2       1
3       1
4       1
       ..
7608    1
7609    1
7610    1
7611    1
7612    1
Name: target, Length: 7613, dtype: int64
```

In [128…
```
# Initialize TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=5000)

# Transform the tweets into TF-IDF vectors
X = tfidf_vectorizer.fit_transform(info['tweets'])


X = X.toarray()  # Convert to array if needed
y = info['target']
```

In [129…
```
# Split the dataset into training and testing sets
X = info["tweets"]
y = info["target"]
```

In [130…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [131…
```
# Initialize TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=5000)

# Transform the tweets into TF-IDF vectors
X = tfidf_vectorizer.fit_transform(info['tweets'])


X = X.toarray()  # Convert to array if needed
y = info['target']
```

In [132…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [133…
```
# multinomialNB()
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)
```

Out[133]:
```
▾ MultinomialNB
MultinomialNB()
```

In [134…
```
nb_pred = nb_classifier.predict(X_test)
```

In [135…
```
nb_cm = confusion_matrix(y_test, nb_pred)
nb_report = classification_report(y_test, nb_pred)
```

In [136…
```
print("Multinomial Naïve Bayes Confusion Matrix:\n", nb_cm)
```

```
Multinomial Naïve Bayes Confusion Matrix:
 [[794  80]
 [214 435]]
```

In [137…
```
print("Multinomial Naïve Bayes Classification Report:\n", nb_report)
```

```
Multinomial Naïve Bayes Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.91      0.84       874
           1       0.84      0.67      0.75       649

    accuracy                           0.81      1523
   macro avg       0.82      0.79      0.80      1523
weighted avg       0.81      0.81      0.80      1523
```

In [138…
```
# LOGISTICREGRESSION
logistic_classifier = LogisticRegression()
logistic_classifier.fit(X_train, y_train)
```

Out[138]:
```
▾ LogisticRegression
LogisticRegression()
```

In [139…
```
logistic_pred = logistic_classifier.predict(X_test)
```

In [140…
```
logistic_cm = confusion_matrix(y_test, logistic_pred)
logistic_report = classification_report(y_test, logistic_pred)
```

In [141…
```
print("\nLogistic Regression Confusion Matrix:\n", logistic_cm)
```

```
Logistic Regression Confusion Matrix:
 [[773 101]
 [196 453]]
```

In [142…
```
print("Logistic Regression Classification Report:\n", logistic_report)
```

```
Logistic Regression Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.88      0.84       874
           1       0.82      0.70      0.75       649

    accuracy                           0.80      1523
   macro avg       0.81      0.79      0.80      1523
weighted avg       0.81      0.80      0.80      1523
```

In [143…
```
# KNEIGHBORSCLASSIFIER
knn_classifier = KNeighborsClassifier()
knn_classifier.fit(X_train, y_train)
```

Out[143]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [144…
```
knn_pred = knn_classifier.predict(X_test)
```

In [145…
```
knn_cm = confusion_matrix(y_test, knn_pred)
knn_report = classification_report(y_test, knn_pred)
```

In [146…
```
print("\nKNN Classification Confusion Matrix:\n", knn_cm)
```

```
KNN Classification Confusion Matrix:
 [[869   5]
 [507 142]]
```

In [147…
```
print("KNN Classification Classification Report:\n", knn_report)
```

```
KNN Classification Classification Report:
               precision    recall  f1-score   support

           0       0.63      0.99      0.77       874
           1       0.97      0.22      0.36       649

    accuracy                           0.66      1523
   macro avg       0.80      0.61      0.56      1523
weighted avg       0.77      0.66      0.60      1523
```

In [148…
```
# BEST ACCURACY
classifiers = [
    ('MultinomialNB', MultinomialNB()),
    ('LogisticRegression', LogisticRegression()),
    ('KNeighborsClassifier', KNeighborsClassifier())
]

# Iterate through classifiers and find the best model
best_model = None
best_accuracy = 0.0

for name, clf in classifiers:
    # Perform cross-validation on the current classifier
    cv_scores = cross_val_score(clf, X_train, y_train, cv=5)
    mean_accuracy = np.mean(cv_scores)

    print(f'{name} Cross-Validation Mean Accuracy: {mean_accuracy}')

    # Update best model if the current model has higher accuracy
    if mean_accuracy > best_accuracy:
        best_accuracy = mean_accuracy
        best_model = clf

# Train the best model on the full training set
best_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = best_model.predict(X_test)

# Evaluate the best model's accuracy on the test set
test_accuracy = accuracy_score(y_test, y_pred)
print(f'Best Model Test Accuracy: {test_accuracy}')
```

```
MultinomialNB Cross-Validation Mean Accuracy: 0.799014778325123
LogisticRegression Cross-Validation Mean Accuracy: 0.7983579638752052
KNeighborsClassifier Cross-Validation Mean Accuracy: 0.6461412151067323
Best Model Test Accuracy: 0.8069599474720945
```

In [149…
```
# BEST ACCURACY
models = [

    ("MultinomialNB", MultinomialNB()),
    ("LogisticRegression", LogisticRegression()),
    ("KNeighborsClassifier", KNeighborsClassifier())

]

best_model_name = None
best_accuracy = 0


for model_name, model in models:

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)


    accuracy = accuracy_score(y_test, y_pred)


    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_model_name = model_name


print(f"Best Model: {best_model_name}")
print(f"Accuracy: {best_accuracy * 100:.2f}%")
```

```
Best Model: MultinomialNB
Accuracy: 80.70%
```

In [ ]:

In [ ]: