

Comcast Telecom Complaints Datasets

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
print('setup Completed^___^')

setup Completed^___^
```

```
###!mkdir ~/.kaggle
```

```
###!cp /kaggle.json ~/.kaggle/
```

```
###!chmod 600 ~/.kaggle/kaggle.json
```

```
###! pip install kaggle
```

```
###!pip install keras-tuner
```

```
###! kaggle datasets download -d yasserh/comcast-telecom-complaints
```

```
###! unzip /content/comcast-telecom-complaints.zip
```

```
result = pd.read_csv('/content/Comcast.csv')
result.head(2)
```

	Ticket #	Customer Complaint	Date	Date_month_year	Time	Received Via	City	Stat
0	250635	Comcast Cable Internet Speeds	22-04-15	22-Apr-15	3:53:50 PM	Customer Care Call	Abingdon	Maryland
		Payment disappear	04-		10:22:56			

```
result.columns
```

```
Index(['Ticket #', 'Customer Complaint', 'Date', 'Date_month_year', 'Time',
      'Received Via', 'City', 'State', 'Zip code', 'Status',
      'Filing on Behalf of Someone'],
      dtype='object')
```

```
result.Status.value_counts()
```

```
Solved      973
Closed      734
Open        363
Pending     154
Name: Status, dtype: int64
```

```
result.isnull().sum()
```

```
Ticket #      0
Customer Complaint  0
Date          0
```

```
Date_month_year      0
Time                  0
Received Via          0
City                  0
State                 0
Zip code              0
Status                0
Filing on Behalf of Someone  0
dtype: int64
```

```
result.shape
```

```
(2224, 11)
```

```
train = result.iloc[:1900,:]
test = result.iloc[1900:,:]
```

```
print(train.shape)
print(test.shape)
```

```
(1900, 11)
(324, 11)
```

```
train.to_csv("train.csv")
test.to_csv("test.csv")
```

```
train2 = pd.read_csv("/content/train.csv", lineterminator='\n')
```

```
test2 = pd.read_csv("/content/test.csv", lineterminator='\n')
```

```
print(train2.columns)
```

```
Index(['Unnamed: 0', 'Ticket #', 'Customer Complaint', 'Date',
      'Date_month_year', 'Time', 'Received Via', 'City', 'State', 'Zip code',
      'Status', 'Filing on Behalf of Someone'],
      dtype='object')
```

```
print(test2.columns)
```

```
Index(['Unnamed: 0', 'Ticket #', 'Customer Complaint', 'Date',
      'Date_month_year', 'Time', 'Received Via', 'City', 'State', 'Zip code',
      'Status', 'Filing on Behalf of Someone'],
      dtype='object')
```

```
print(train2.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1900 entries, 0 to 1899
Data columns (total 12 columns):
#   Column              Non-Null Count  Dtype
---  -
0   Unnamed: 0           1900 non-null  int64
1   Ticket #             1900 non-null  object
2   Customer Complaint   1900 non-null  object
3   Date                 1900 non-null  object
4   Date_month_year      1900 non-null  object
5   Time                 1900 non-null  object
6   Received Via         1900 non-null  object
7   City                 1900 non-null  object
8   State                1900 non-null  object
9   Zip code             1900 non-null  int64
10  Status               1900 non-null  object
11  Filing on Behalf of Someone 1900 non-null  object
dtypes: int64(2), object(10)
memory usage: 178.2+ KB
None
```

```
print(test2.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 324 entries, 0 to 323
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Unnamed: 0             324 non-null   int64
1   Ticket #               324 non-null   int64
2   Customer Complaint     324 non-null   object
3   Date                   324 non-null   object
4   Date_month_year        324 non-null   object
5   Time                   324 non-null   object
6   Received Via           324 non-null   object
7   City                   324 non-null   object
8   State                  324 non-null   object
9   Zip code               324 non-null   int64
10  Status                 324 non-null   object
11  Filing on Behalf of Someone 324 non-null   object
dtypes: int64(3), object(9)
memory usage: 30.5+ KB
None
```

```
train2["Date"]
```

```
0      22-04-15
1      04-08-15
2      18-04-15
3      05-07-15
4      26-05-15
...
1895   26-05-15
1896   17-06-15
1897   24-06-15
1898   30-04-15
1899   04-07-15
Name: Date, Length: 1900, dtype: object
```

```
train2["Date"] = pd.to_datetime(train2["Date"])
```

```
test2["Date"] = pd.to_datetime(test2["Date"])
```

```
train2['year'] = train2['Date'].dt.year
```

```
train2['month'] = train2['Date'].dt.month
```

```
train2['day'] = train2['Date'].dt.day
```

```
test2['year'] = test2['Date'].dt.year
test2['month'] = test2['Date'].dt.month
test2['day'] = test2['Date'].dt.day
```

```
train2.drop(columns = "Date", inplace=True)
```

```
test2.drop(columns = "Date", inplace=True)
```

```
####! pip install unicode
```

```
#####! pip install nltk
```

```
import re, unicode
from bs4 import BeautifulSoup
```

```

from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Needed only once
# import nltk
# nltk.download('stopwords')
# nltk.download('punkt')
# nltk.download('wordnet')

def remove_html_tags(text):
    soup = BeautifulSoup(text, "html.parser")
    stripped_text = soup.get_text(separator=" ")
    return stripped_text

def remove_accented_chars(text):
    text = unidecode.unidecode(text)
    return text

def remove_numbers(text):
    result = re.sub(r'\d+', '', text)
    return result

def remove_slash_with_space(text):
    return text.replace('\\', " ")

def remove_punctuation(text):
    translator = str.maketrans('', '', string.punctuation)
    return text.translate(translator)

def text_lowercase(text):
    return text.lower()

def remove_whitespace(text):
    return " ".join(text.split())

def remove_stopwords(text):
    stop_words = set(stopwords.words("english"))
    word_tokens = word_tokenize(text)
    filtered_text = [word for word in word_tokens if word not in stop_words]
    return ' '.join(filtered_text)

def stem_words(text):
    stemmer = PorterStemmer()
    word_tokens = word_tokenize(text)
    stems = [stemmer.stem(word) for word in word_tokens]
    return ' '.join(stems)

def lemmatize_words(text):
    lemmatizer = WordNetLemmatizer()
    word_tokens = word_tokenize(text)
    # provide context i.e. part-of-speech
    lemmas = [lemmatizer.lemmatize(word, pos='v') for word in word_tokens]
    return ' '.join(lemmas)

```

```

# Perform preprocessing
def perform_preprocessing(text):
    text = remove_html_tags(text)
    text = remove_accented_chars(text)
    text = remove_numbers(text)
    text = remove_stopwords(text)
    text = text_lowercase(text)
    text = remove_slash_with_space(text)
#     text = remove_punctuation(text)
    text = stem_words(text)
    text = lemmatize_words(text)
    text = remove_whitespace(text)
    return text

```

```

import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')

```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords 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 wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
True
```

```
train2.columns
```

```
Index(['Unnamed: 0', 'Ticket #', 'Customer Complaint', 'Date_month_year',
      'Time', 'Received Via', 'City', 'State', 'Zip code', 'Status',
      'Filing on Behalf of Someone', 'year', 'month', 'day'],
      dtype='object')
```

```
train2['Customer Complaint'] = train2['Customer Complaint'].apply(perform_preprocessing)
```

```
<ipython-input-36-18fdfef6dc41>:15: MarkupResemblesLocatorWarning: The input looks more like a filename than markup. You may want to use the lxml library for parsing.
soup = BeautifulSoup(text, "html.parser")
```

```
test2['Customer Complaint'] = test2['Customer Complaint'].apply(perform_preprocessing)
```

```
<ipython-input-36-18fdfef6dc41>:15: MarkupResemblesLocatorWarning: The input looks more like a filename than markup. You may want to use the lxml library for parsing.
soup = BeautifulSoup(text, "html.parser")
```

```
####! pip install --upgrade pandas
```

```
np.version.version
```

```
'1.23.5'
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
import xgboost as xgb
from sklearn.metrics import accuracy_score, top_k_accuracy_score
from sklearn import metrics
```

```
import pandas as pd
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
import re
import nltk
from collections import Counter
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB, MultinomialNB
```

```
from sklearn.svm import SVC
from sklearn import metrics
```

```
#### pip install nltk
```

```
#### pip install keras
```

```
import pandas, numpy, string, textblob
import pickle
from sklearn import model_selection, preprocessing, linear_model, naive_bayes, metrics, svm, decomposition, ensemble
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import xgboost
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.preprocessing import text, sequence
import matplotlib.pyplot as plt
```

```
#### pip install textblob
```

```
#### pip install keras
```

```
#### pip install tensorflow==2.7.0
```

```
## pip install unicode
```

```
import re, unicode
from bs4 import BeautifulSoup
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Needed only once
# import nltk
# nltk.download('stopwords')
# nltk.download('punkt')
# nltk.download('wordnet')

def remove_html_tags(text):
    soup = BeautifulSoup(text, "html.parser")
    stripped_text = soup.get_text(separator=" ")
    return stripped_text

def remove_accented_chars(text):
    text = unicode.unidecode(text)
    return text

def remove_numbers(text):
    result = re.sub(r'\d+', '', text)
    return result

def remove_slash_with_space(text):
    return text.replace('\', " ")

def remove_punctuation(text):
    translator = str.maketrans('', '', string.punctuation)
    return text.translate(translator)

def text_lowercase(text):
    return text.lower()

def remove_whitespace(text):
    return " ".join(text.split())

def remove_stopwords(text):
    stop_words = set(stopwords.words("english"))
    word_tokens = word_tokenize(text)
    filtered_text = [word for word in word_tokens if word not in stop_words]
    return ' '.join(filtered_text)

def stem_words(text):
    stemmer = PorterStemmer()
```

```

word_tokens = word_tokenize(text)
stems = [stemmer.stem(word) for word in word_tokens]
return ' '.join(stems)
def lemmatize_words(text):
    lemmatizer = WordNetLemmatizer()
    word_tokens = word_tokenize(text)
    # provide context i.e. part-of-speech
    lemmas = [lemmatizer.lemmatize(word, pos='v') for word in word_tokens]
    return ' '.join(lemmas)

```

```

# Perform preprocessing
def perform_preprocessing(text):
    text = remove_html_tags(text)
    text = remove_accented_chars(text)
    text = remove_numbers(text)
    text = remove_stopwords(text)
    text = text_lowercase(text)
    text = remove_slash_with_space(text)
    text = remove_punctuation(text)
    text = stem_words(text)
    text = lemmatize_words(text)
    text = remove_whitespace(text)
    return text

```

```

import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')

```

```

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords 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 wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
True

```

```
train2.columns
```

```

Index(['Unnamed: 0', 'Ticket #', 'Customer Complaint', 'Date_month_year',
      'Time', 'Received Via', 'City', 'State', 'Zip code', 'Status',
      'Filing on Behalf of Someone', 'year', 'month', 'day'],
      dtype='object')

```

```
#### pip install nlp_utils
```

```
#### pip install contractions
```

```
#### pip install wordcloud
```

```
train2.columns
```

```


Index(['Unnamed: 0', 'Ticket #', 'Customer Complaint', 'Date_month_year',
      'Time', 'Received Via', 'City', 'State', 'Zip code', 'Status',
      'Filing on Behalf of Someone', 'year', 'month', 'day'],
      dtype='object')

```

```
train2.rename(columns = {"Customer Complaint" : "CustomerComplaint"}, inplace=True)
```

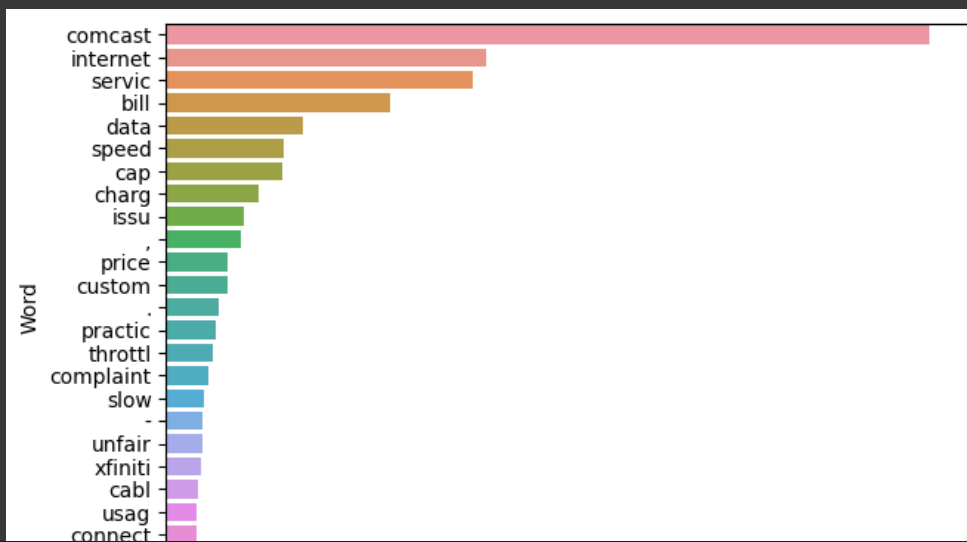
```
test2.rename(columns = {"Customer Complaint" : "CustomerComplaint"}, inplace=True)
```

```
test_texts1 = " ".join(CustomerComplaint for CustomerComplaint in test2.CustomerComplaint)
```



```
# visualising frewords
```

```
freq_df = showmostfrequentwords(train2['CustomerComplaint'],25)
```

```
freq_df.sort_values('count',ascending=False).head(10).style.background_gradient(cmap = 'plasma')
```

	word	count
0	comcast	1042
2	internet	437
7	servic	419
43	bill	306
23	data	187
3	speed	160
13	cap	158
21	charg	127
88	issu	106
98	,	102

```
#Performing tf-idf
```

```
tfidf_vectorizer = TfidfVectorizer(min_df=5)
xtrain_tfidf = tfidf_vectorizer.fit_transform(train2["CustomerComplaint"].tolist())
xtest_tfidf = tfidf_vectorizer.transform(test2["CustomerComplaint"].tolist())
```

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
plt.style.use('seaborn-bright')
```

```
def get_top_n_words(corpus, n=None):
    vec=CountVectorizer().fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]

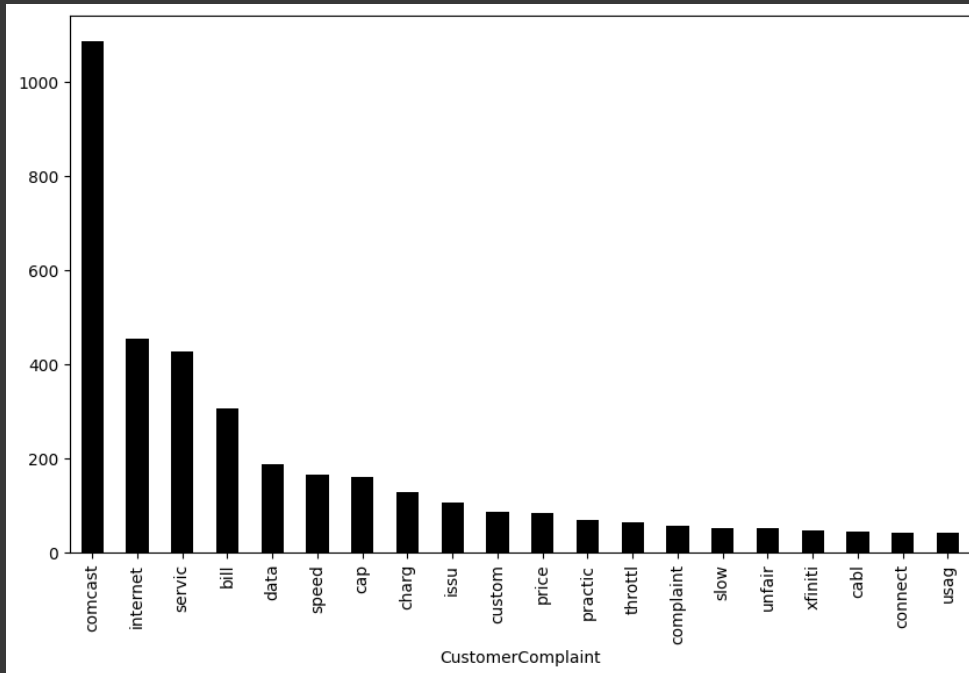
common_words = get_top_n_words(train2['CustomerComplaint'], 20)
df1 = pd.DataFrame(common_words, columns = ['CustomerComplaint', 'count'])
df1.head()
```

CustomerComplaint count

0 comcast 1087

1 internet 453

```
df1.groupby('CustomerComplaint').sum()['count'].sort_values(ascending=False).plot(kind='bar',color='black',figsize = (10, 6))
xlabel = 'Top Words'
ylabel = 'Count'
title = 'BarChart represent the Top Words Frequency(Uni-Grams Analysis)'
plt.show()
```



```
def get_top_n_bigram(corpus, n=None):
    vec = CountVectorizer(ngram_range=(2,2)).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]

common_words2 = get_top_n_bigram(train2['CustomerComplaint'], 30)
df3 = pd.DataFrame(common_words2, columns=['CustomerComplaint', "Count"])
df3.head()
```

CustomerComplaint Count

0 data cap 128

1 comcast internet 116

2 comcast data 92

3 internet servic 77

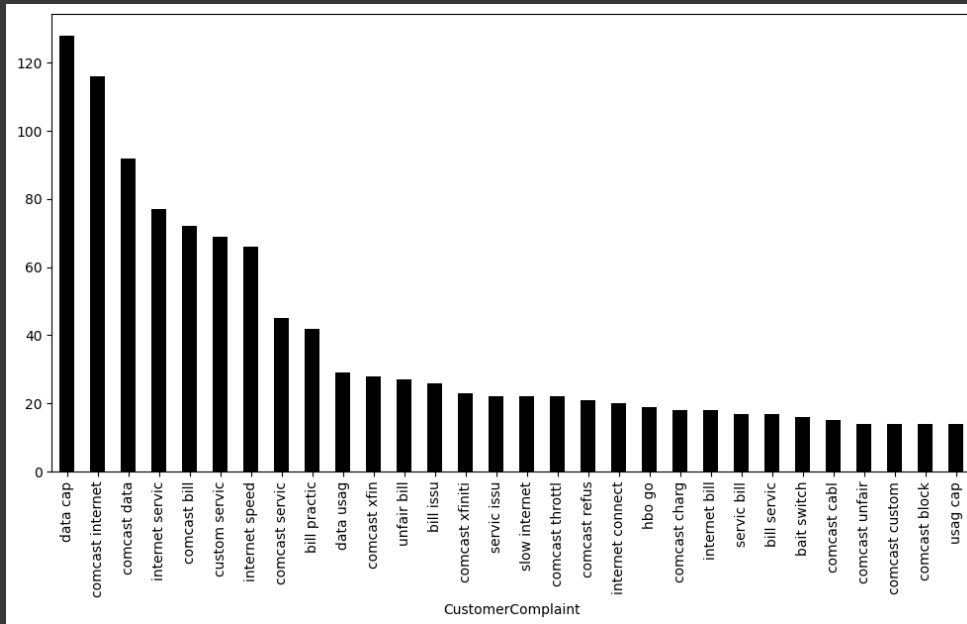
4 comcast bill 72

```
df3.groupby('CustomerComplaint').sum()['Count'].sort_values(ascending=False).plot(kind='bar',figsize=(12,6), color='black')
xlabel = "Bigram Words"
```

```

ylabel = "Count"
title = "Bar chart of Bigrams Frequency"
plt.show()

```

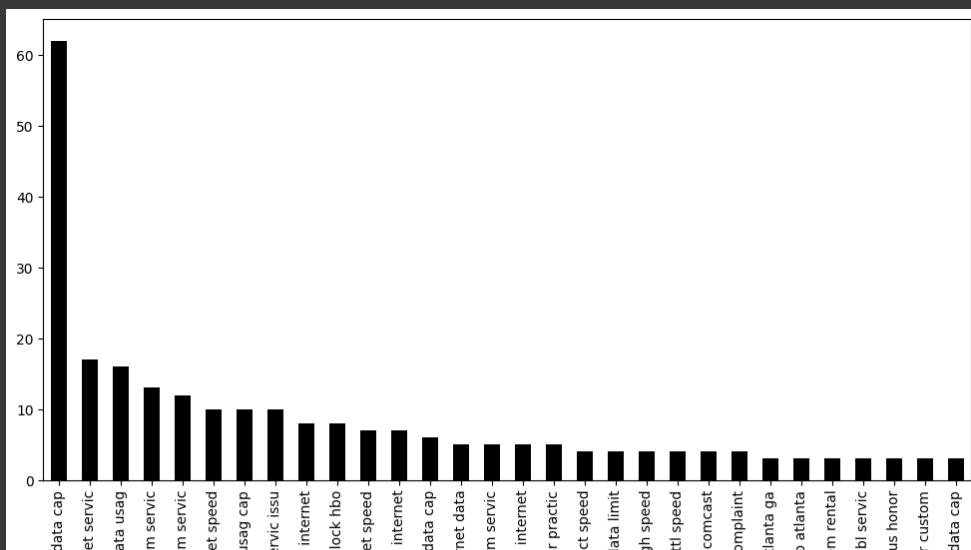


```

def get_top_n_trigram(corpus, n=None):
    vec = CountVectorizer(ngram_range=(3, 3), stop_words='english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq = sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]

common_words4 = get_top_n_trigram(train2['CustomerComplaint'], 30)
df4 = pd.DataFrame(common_words4, columns = ['CustomerComplaint' , 'Count'])
df4.groupby('CustomerComplaint').sum()['Count'].sort_values(ascending=False).plot(kind='bar',figsize=(12,6), color='black')
xlabel = "Trigram Words"
ylabel = "Count"
title = "Bar chart of Trigrams Frequency"
plt.show()

```



```
#Performing tf-idf
```

```
count_vectorizer = CountVectorizer(min_df=5)
xtrain_count_cr = count_vectorizer.fit_transform(train2["CustomerComplaint"].tolist())
xtest_count_cr = count_vectorizer.transform(test2["CustomerComplaint"].tolist())
```

```
#Performing tf-idf
```

```
tfidf_vectorizer = TfidfVectorizer(min_df=5)
xtrain_tfidf_rd = tfidf_vectorizer.fit_transform(train2["CustomerComplaint"].tolist())
xtest_tfidf_rd = tfidf_vectorizer.transform(test2["CustomerComplaint"].tolist())
```

```
train2['CustomerComplaint_len'] = train2['CustomerComplaint'].astype(str).apply(len)
train2['word_count_CustomerComplaint'] = train2['CustomerComplaint'].apply(lambda x: len(str(x).split()))
```

```
test2['CustomerComplaint_len'] = test2['CustomerComplaint'].astype(str).apply(len)
test2['word_count_CustomerComplaint'] = test2['CustomerComplaint'].apply(lambda x: len(str(x).split()))
```

```
test2['average_word_len'] = test2["CustomerComplaint"].apply(lambda x: np.mean([len(w) for w in x.split()])))
```

```
train2['average_word_len'] = train2["CustomerComplaint"].apply(lambda x: np.mean([len(w) for w in x.split()])))
```

```
from textblob import TextBlob, Word, Blobber
from nltk.stem import PorterStemmer
```

```
train2['polarity'] = train2['CustomerComplaint'].map(lambda text: TextBlob(text).sentiment.polarity)
train2.head()
```

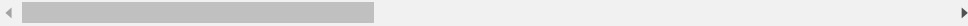
Unnamed: 0	Ticket #	CustomerComplaint	Date_month_year	Time	Received Via	City
------------	----------	-------------------	-----------------	------	--------------	------

0	0	250635	comcast cabl internet speed	22-Apr-15	3:53:50 PM	Customer Care Call	Abingd
---	---	--------	--------------------------------	-----------	---------------	-----------------------	--------

```
test2['polarity'] = test2['CustomerComplaint'].map(lambda text: TextBlob(text).sentiment.polarity)
test2.head()
```

Unnamed: 0	Ticket #	CustomerComplaint	Date_month_year	Time	Received Via	City
------------	----------	-------------------	-----------------	------	--------------	------

0	1900	257349	comcast internet service	25-Apr-15	8:09:47 PM	Internet	Silver Spring
1	1901	293593	email sdervic	16-May-15	6:05:47 PM	Customer Care Call	Sinking Spring
2	1902	373797	complaint comcast	29-Jun-15	10:37:26 PM	Internet	Skokie
3	1903	216841	internet speed	04-Mar-15	4:04:30 PM	Internet	Slc
4	1904	370691	comcast internet complaint	28-Jun-15	1:23:56 AM	Customer Care Call	Slc



```
####! pip install vaderSentiment
```

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
```

```
analyzer = SentimentIntensityAnalyzer()
```

```
train2['compound'] = [analyzer.polarity_scores(x)['compound'] for x in train2['CustomerComplaint']]
train2['neg'] = [analyzer.polarity_scores(x)['neg'] for x in train2['CustomerComplaint']]
train2['neu'] = [analyzer.polarity_scores(x)['neu'] for x in train2['CustomerComplaint']]
train2['pos'] = [analyzer.polarity_scores(x)['pos'] for x in train2['CustomerComplaint']]
```

```
test2['compound'] = [analyzer.polarity_scores(x)['compound'] for x in test2['CustomerComplaint']]
test2['neg'] = [analyzer.polarity_scores(x)['neg'] for x in test2['CustomerComplaint']]
test2['neu'] = [analyzer.polarity_scores(x)['neu'] for x in test2['CustomerComplaint']]
test2['pos'] = [analyzer.polarity_scores(x)['pos'] for x in test2['CustomerComplaint']]
```

```
test2.columns
```

```
Index(['Unnamed: 0', 'Ticket #', 'CustomerComplaint', 'Date_month_year',
      'Time', 'Received Via', 'City', 'State', 'Zip code', 'Status',
      'Filing on Behalf of Someone', 'year', 'month', 'day',
      'CustomerComplaint_len', 'word_count_CustomerComplaint',
      'average_word_len', 'polarity', 'compound', 'neg', 'neu', 'pos'],
      dtype='object')
```

```
test2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 324 entries, 0 to 323
Data columns (total 22 columns):
```

#	Column	Non-Null Count	Dtype
0	Unnamed: 0	324 non-null	int64
1	Ticket #	324 non-null	int64
2	CustomerComplaint	324 non-null	object
3	Date_month_year	324 non-null	object
4	Time	324 non-null	object
5	Received Via	324 non-null	object
6	City	324 non-null	object
7	State	324 non-null	object
8	Zip code	324 non-null	int64
9	Status	324 non-null	object
10	Filing on Behalf of Someone	324 non-null	object
11	year	324 non-null	int64
12	month	324 non-null	int64
13	day	324 non-null	int64
14	CustomerComplaint_len	324 non-null	int64
15	word_count_CustomerComplaint	324 non-null	int64
16	average_word_len	324 non-null	float64
17	polarity	324 non-null	float64
18	compound	324 non-null	float64
19	neg	324 non-null	float64
20	neu	324 non-null	float64
21	pos	324 non-null	float64

dtypes: float64(6), int64(8), object(8)
memory usage: 55.8+ KB

```
import scipy
```

```
X_train = scipy.sparse.hstack((xtrain_count_cr,
                               xtrain_tfidf_rd,
                               train2[['CustomerComplaint_len', 'word_count_CustomerComplaint', 'average_word_len', 'polarity', 'compound']
```

```
X_test = scipy.sparse.hstack((xtest_count_cr,
                               xtest_tfidf_rd,
                               test2[['CustomerComplaint_len', 'word_count_CustomerComplaint', 'average_word_len', 'polarity', 'compound'],
```

```
train2['Status'].value_counts()
```

```
Solved      831
Closed      620
Open        321
Pending     128
Name: Status, dtype: int64
```

```
train2["Encoded_Status"] = train2['Status'].astype("category").cat.codes
```

```
test2["Encoded_Status"] = test2['Status'].astype("category").cat.codes
```

```
Y_train = train2["Encoded_Status"]
```

```
Y_test = test2["Encoded_Status"]
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, recall_score, classification_report, cohen_kappa_score
from sklearn import metrics
```

```
###!pip install xgboost==0.90
```

▼ XGBClassifier

```
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
```

```
mb = XGBClassifier()
mb.fit(X_train,Y_train)
mbpred = mb.predict(X_test)
print(metrics.accuracy_score(Y_test,mbpred))
```

```
0.41358024691358025
```

```
print('Baseline: Accuracy: ', round(accuracy_score(Y_test, mbpred)*100, 2))
print('\n Classification Report:\n', classification_report(Y_test,mbpred))
```

```
Baseline: Accuracy: 41.36
```

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

    0           0.38        0.18        0.25        114
    1           0.29        0.05        0.08         42
    2           0.18        0.08        0.11         26
    3           0.44        0.77        0.56        142

 accuracy                   0.41        324
 macro avg              0.32        0.27        0.25        324
 weighted avg           0.37        0.41        0.35        324
```

▼ Random Forest Classifier

```
rf = RandomForestClassifier()
rf.fit(X_train,Y_train)
rfpred = rf.predict(X_test)
print(metrics.accuracy_score(Y_test,rf.predict(X_test)))
```

```
0.4444444444444444
```

```
#!/ pip install optuna
```

```
import optuna
import sklearn
```

```
param_grid_optuna = {
    "bootstrap": [True, False],
    "max_depth": [10, 20, 30, 40, 50, None],
    "max_features": ['auto', 'sqrt'],
    "min_samples_leaf": [1, 2, 4],
    "min_samples_split": [2, 5, 8, 10, 12],
    "n_estimators": [100, 200, 300, 400, 500, 600]
}
```

```
from sklearn.model_selection import cross_val_score
```

```
def objective(trial):
    bootstrap = trial.suggest_categorical('bootstrap',[True, False])
    max_depth = trial.suggest_int('max_depth', 10, 50)
    max_features = trial.suggest_categorical('max_features', ['auto', 'sqrt'])
    min_samples_leaf = trial.suggest_int('min_samples_leaf', 1, 4)
    min_samples_split = trial.suggest_int('min_samples_split', 2, 12)
    n_estimators = trial.suggest_int('n_estimators', 100, 600)

    clsr = RandomForestClassifier(bootstrap = bootstrap,
                                  max_depth = max_depth, max_features = max_features,min_samples_leaf = min_samples_leaf,
```

```

min_samples_split = min_samples_split, n_estimators = n_estimators)

#regr.fit(X_train, y_train)
#y_pred = regr.predict(X_val)
#return r2_score(y_val, y_pred)

score = cross_val_score(clsr, X_train, Y_train, cv=6, n_jobs=-1)
meanvalue = score.mean()

return meanvalue

```

```

#Execute optuna and set hyperparameters
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=12)

```

```

[I 2023-08-13 18:41:16,678] A new study created in memory with name: no-name-749baacc-3ca1-4749-bdda-e42a9d4f71b0
[I 2023-08-13 18:41:26,477] Trial 0 finished with value: 0.42156158074778044 and parameters: {'bootstrap': False, 'max_depth': 3}
[I 2023-08-13 18:41:34,179] Trial 1 finished with value: 0.4247261377098058 and parameters: {'bootstrap': True, 'max_depth': 14,
[I 2023-08-13 18:41:35,669] Trial 2 finished with value: 0.4263117438006629 and parameters: {'bootstrap': True, 'max_depth': 44,
[I 2023-08-13 18:41:41,829] Trial 3 finished with value: 0.42051005603694974 and parameters: {'bootstrap': False, 'max_depth': 2}
[I 2023-08-13 18:41:44,163] Trial 4 finished with value: 0.4262884505317521 and parameters: {'bootstrap': False, 'max_depth': 33}
[I 2023-08-13 18:41:48,005] Trial 5 finished with value: 0.4184020152005217 and parameters: {'bootstrap': False, 'max_depth': 36}
[I 2023-08-13 18:41:55,875] Trial 6 finished with value: 0.4205050646221831 and parameters: {'bootstrap': False, 'max_depth': 50}
[I 2023-08-13 18:41:59,492] Trial 7 finished with value: 0.4236729491940529 and parameters: {'bootstrap': False, 'max_depth': 37}
[I 2023-08-13 18:42:07,753] Trial 8 finished with value: 0.42312722117957113 and parameters: {'bootstrap': False, 'max_depth': 1}
[I 2023-08-13 18:42:10,994] Trial 9 finished with value: 0.4310419411944788 and parameters: {'bootstrap': True, 'max_depth': 13,
[I 2023-08-13 18:42:17,804] Trial 10 finished with value: 0.42841146561247995 and parameters: {'bootstrap': True, 'max_depth': 2}
[I 2023-08-13 18:42:26,946] Trial 11 finished with value: 0.42577599861571436 and parameters: {'bootstrap': True, 'max_depth': 2}

```

```

#Create an instance with tuned hyperparameters
optimised_rf = RandomForestClassifier(bootstrap = study.best_params['bootstrap'],
                                     max_depth = study.best_params['max_depth'], max_features = study.best_params['max_features'],
                                     min_samples_leaf = study.best_params['min_samples_leaf'],
                                     min_samples_split = study.best_params['min_samples_split'],
                                     n_estimators = study.best_params['n_estimators'])

#learn
optimised_rf.fit(X_train ,Y_train)

```

```

RandomForestClassifier
RandomForestClassifier(max_depth=13, min_samples_leaf=2, min_samples_split=3,
                       n_estimators=183)

```

```

optimised_rf.score(X_train ,Y_train)

```

```

0.5352631578947369

```

```

trial = study.best_trial
print('Accuracy: {}'.format(trial.value))

```

```

Accuracy: 0.4310419411944788

```

```

study.best_params

```

```

{'bootstrap': True,
 'max_depth': 13,
 'max_features': 'sqrt',
 'min_samples_leaf': 2,
 'min_samples_split': 3,
 'n_estimators': 183}

```

```

print("Best hyperparameters: {}".format(trial.params))

```

```

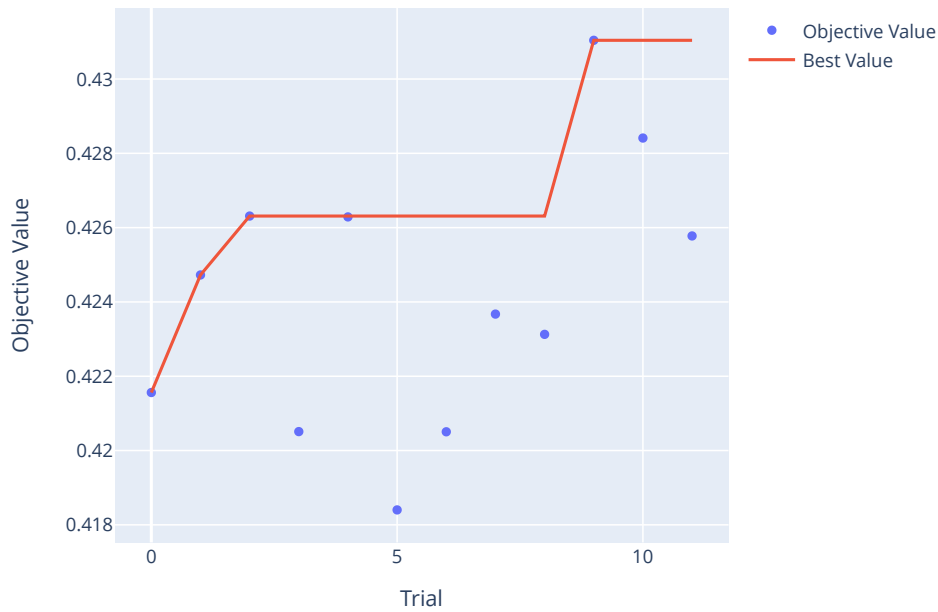
Best hyperparameters: {'bootstrap': True, 'max_depth': 13, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 3,

```



```
optuna.visualization.plot_optimization_history(study)
```

Optimization History Plot



```
y_pred_optuna = optimised_rf.predict(X_test)
```

```
y_pred_optuna = pd.DataFrame(y_pred_optuna)
```

```
y_pred_optuna.rename(columns = {0: "Predict"}, inplace=True)
```

```
y_pred_optuna.value_counts()
```

```
Predict
3      280
0       32
2        8
1         4
dtype: int64
```

```
class_names = {
3 : "Solved",
0 : "Closed",
1 : "Open",
2 : "Pending"
}
```

```
y_pred_optuna = y_pred_optuna['Predict'].map(class_names)
```

```
y_pred_optuna.value_counts()
```

```
☐➤ Solved      280
    Closed      32
    Pending      8
    Open         4
    Name: Predict, dtype: int64
```

✓ 0s completed at 12:12 AM

