1. Write 5-7 String Functions in Python Language

Python provides a wide range of built-in functions for working with strings. Here are some common string functions with example code:

1. len (): Returns the length of a string.

```
text = "Hello, World!"
length = len(text)
print("Length of String: ",length)
```

Output:

```
 Length of string: 48
```

2. str(): Converts other data types to a string.

```
num = 42
text = str(num)
print(text)
```

Output:

```
 42
```

3. upper() and lower(): Convert a string to uppercase or lowercase.

```
text = "Hello, World!"
uppercase_text = text.upper()
lowercase_text = text.lower()
print(uppercase_text)
print(lowercase_text)
```

Output:

HELLO, WORLD!

Hello, world!

4. strip(): Removes leading and trailing whitespace characters from string.

```
text  =  "  Hello,  World!  "
stripped_text = text.strip()
print(stripped_text)
```

Output:
Hello, World!

5. replace(): Replaces a substring with another substring in a string.

```
text = "Hello, World!"
new_text = text.replace("World", "Python")
print(new_text)
```

Output:
Hello, Python!

6. split(): Splits a string into a list of substrings based on a specified delimiter.

```
text = "apple,banana,cherry"
fruits = text.split(",")
print(fruits)
```

Output:

[‘apple’, ‘banana’, ‘cherry’]

7. join(): Joins a list of strings into a single string using a specified delimiter.

```
fruits = ['apple', 'banana', 'cherry']
text = ",".join(fruits)
print(text)
```

Output:

apple, banana, cherry

8. find() and index(): Search for a substring in a string and return its index.

```
text = "Hello, World!"
index1 = text.find("World")
index2 = text.index("World")
print("Find Function: ",index1)
print("Index Function: ",index2)
```

Output:
Index Function: 7

9. count(): Count the occurrences of a substring in a string.

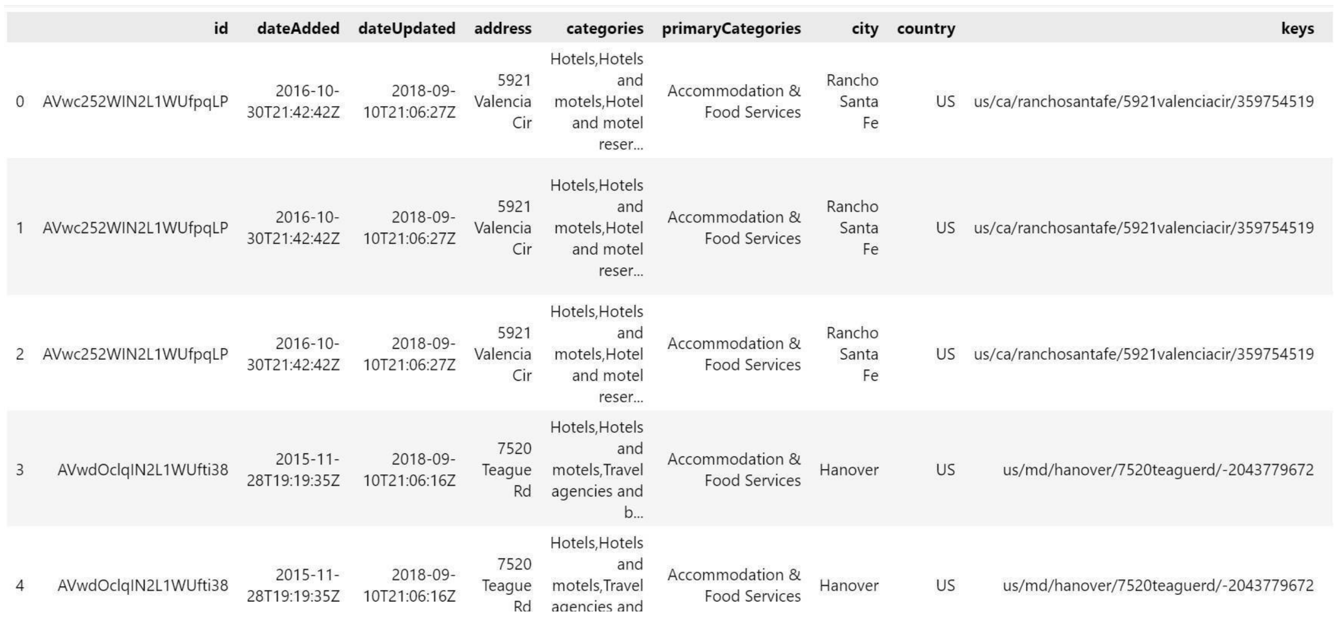
```
text = "Hello, World!"
count = text.count("l")
print("Count Function:",count)
```

Output:

```
Count Function:  3
```

2. Write 5-7 Text Pre-Processing Techniques

```
import numpy as np
import pandas as pd
import re
import nltk
import spacy
import string

pd.options.mode.chained_assignment = None
df = pd.read_csv("Datafiniti_Hotel_Reviews.csv")
df.head()
```



	id	dateAdded	dateUpdated	address	categories	primaryCategories	city	country	keys
0	AVwc252WIN2L1WUfpqLP	2016-10-30T21:42:42Z	2018-09-10T21:06:27Z	5921 Valencia Cir	Hotels,Hotels and motels,Hotel and motel reser...	Accommodation & Food Services	Rancho Santa Fe	US	us/ca/ranchosantafe/5921valenciacir/359754519
1	AVwc252WIN2L1WUfpqLP	2016-10-30T21:42:42Z	2018-09-10T21:06:27Z	5921 Valencia Cir	Hotels,Hotels and motels,Hotel and motel reser...	Accommodation & Food Services	Rancho Santa Fe	US	us/ca/ranchosantafe/5921valenciacir/359754519
2	AVwc252WIN2L1WUfpqLP	2016-10-30T21:42:42Z	2018-09-10T21:06:27Z	5921 Valencia Cir	Hotels,Hotels and motels,Hotel and motel reser...	Accommodation & Food Services	Rancho Santa Fe	US	us/ca/ranchosantafe/5921valenciacir/359754519
3	AVwdOclqIN2L1WUfti38	2015-11-28T19:19:35Z	2018-09-10T21:06:16Z	7520 Teague Rd	Hotels,Hotels and motels,Travel agencies and b...	Accommodation & Food Services	Hanover	US	us/md/hanover/7520teaguerd/-2043779672
4	AVwdOclqIN2L1WUfti38	2015-11-28T19:19:35Z	2018-09-10T21:06:16Z	7520 Teague Rd	Hotels,Hotels and motels,Travel agencies and	Accommodation & Food Services	Hanover	US	us/md/hanover/7520teaguerd/-2043779672

Lower Casing:

```
def lower_casing(text):
    return str(text).lower()

df["text_preprocessed"] = df["reviews.text"].apply(lambda x: lower_casing(x))
df[["reviews.text", "text_preprocessed"]].head()
```

Output:

	reviews.text	text_preprocessed
0	Our experience at Rancho Valencia was absolute...	our experience at rancho valencia was absolute...
1	Amazing place. Everyone was extremely warm and...	amazing place. everyone was extremely warm and...
2	We booked a 3 night stay at Rancho Valencia to...	we booked a 3 night stay at rancho valencia to...
3	Currently in bed writing this for the past hr ...	currently in bed writing this for the past hr ...
4	I live in Md and the Aloft is my Home away fro...	i live in md and the aloft is my home away fro...

Punctuation Removal:

```
# https://www.programiz.com/python-programming/methods/string/translate
PUNCT_TO_REMOVE = '!"#$%&\'()*+,-/:;<=>?@[\\]^_`{|}~' #string.punctuation
def remove_punctuation(text):
    """function to remove the punctuation"""
    return text.translate(str.maketrans('', '', PUNCT_TO_REMOVE))

print("Before: ", df["text_preprocessed"][0], "\n")
df["text_preprocessed"] = df["text_preprocessed"].apply(lambda x:
remove_punctuation(x))
print("After: ", df["text_preprocessed"][0])
```

Output:

```
Before: our experience at rancho valencia was absolutely perfect from beginning to end!!!! we felt
special and very happy during our stayed. i would come back in a heart beat!!!

After:  our experience at rancho valencia was absolutely perfect from beginning to end we felt special and
very happy during our stayed. i would come back in a heart beat
```

Stopwords Removal

The most commonly used words are called stopwords. They contribute very less to the predictions and add very little value analytically. Hence, removing stopwords will make it easier for our models to train the text data.

Examples of stop words include articles ("a", "an", "the"), prepositions ("in", "on", "at"), conjunctions ("and", "but", "or"), pronouns ("he", "she", "they"), and other frequently occurring words like "is", "am", "of", "for", and "with"

Output:

```
from nltk.corpus import stopwords

STOPWORDS = set(stopwords.words('english'))
def remove_stopwords(text):
    """function to remove the stopwords"""
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])

print("Before: ", df["text_preprocessed"][0], "\n")
df["text_preprocessed"] = df["text_preprocessed"].apply(lambda x: remove_stopwords(x))
print("After: ", df["text_preprocessed"][0])
```

Output:

```
Before: our experience at rancho valencia was absolutely perfect from beginning to end we felt special and very happy during our stayed. i would come back in a heart beat

After:  experience rancho valencia absolutely perfect beginning end felt special happy stayed. would come back heart beat
```

Stemming:

Approach: Stemming involves removing suffixes from words to obtain their root form. It uses a set of heuristic rules to chop off common suffixes.

Output: Stemming might result in words that are not actual words. The goal is to achieve linguistic normalization by reducing words to a common base form.

Speed: Stemming is generally faster as it applies simpler rules without considering the context.

Example : The stem of "running" is "run", "jumps" becomes "jump", and "happily" becomes "happi".

```
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
def stem_words(text):
    return " ".join([stemmer.stem(word) for word in text.split()])

print("Before: ", df["text_preprocessed"][0], "\n")
df["text_preprocessed"] = df["text_preprocessed"].apply(lambda x: stem_words(x))
print("After: ", df["text_preprocessed"][0])
```

Output:

```
Before: experience rancho valencia absolutely perfect beginning end felt special happy stayed. would come back heart beat

After: experi rancho valencia absolut perfect begin end felt special happi stayed. would come back heart beat
```

Lemmatization:

Approach: Lemmatization also reduces words to their base form, but it takes into account the word's context and meaning. It uses linguistic knowledge and a vocabulary (dictionary) to ensure valid words are produced.

Output: Lemmatization yields valid words that are present in the language's vocabulary. The goal is to reduce words to their canonical form while maintaining grammatical and semantic accuracy.

Speed: Lemmatization is slower compared to stemming due to its consideration of context and use of a vocabulary lookup.

Example: The lemma of "running" is "run", "jumps" becomes "jump", and "happily" becomes "happy".

```python
import nltk
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
wordnet_map = {"N":wordnet.NOUN, "V":wordnet.VERB, "J":wordnet.ADJ, "R":wordnet.ADV}
def lemmatize_words(text):
    pos_tagged_text = nltk.pos_tag(str(text).split())
    return " ".join([lemmatizer.lemmatize(word, wordnet_map.get(pos[0], wordnet.NOUN))
                     for word, pos in pos_tagged_text])

print("Before: ", df["reviews.text"][0], "\n")
df["text_preprocessed"] = df["reviews.text"].apply(lambda x: lemmatize_words(x))
print("After: ", df["text_preprocessed"][0])
```

Output:

Before: Our experience at Rancho Valencia was absolutely perfect from beginning to end!!!! We felt special and very happy during our stayed. I would come back in a heart beat!!!

After: Our experience at Rancho Valencia be absolutely perfect from begin to end!!!! We felt special and very happy during our stayed. I would come back in a heart beat!!!

Handling Emojis

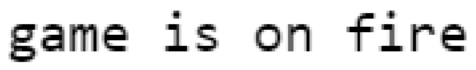
```
def remove_emoji(string):
  emoji_pattern = re.compile("["
                          u"\U0001F600-\U0001F64F" # emoticons
                          u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                          u"\U0001F680-\U0001F6FF"  # transport & map symbols
                          u"\U0001F1E0-\U0001F1FF" # flags (iOS)
                          u"\U00002702-\U000027B0"
                          u"\U000024C2-\U0001F251"
                          "]+", flags=re.UNICODE)
  return emoji_pattern.sub(r'', string)
remove_emoji("game is on 🔥")
```

Output:

```
 'game is on '
```

```
import sys
import re
from emot.emo_unicode import UNICODE_EMOJI  # For emojis

def convert_emojis(text):
    for emot in UNICODE_EMOJI:
        escaped_emot = re.escape(emot)  # Escape special characters
        text = re.sub(r'(' + escaped_emot + ')',
"_".join(UNICODE_EMOJI[emot].replace(",", "").replace(":", "").split()), text)
    return text

text = "game is on 🔥"
converted_text = convert_emojis(text)
print(converted_text)
```

Output:



```
game is on fire
```

Handling Emoticons

```python
import re
from emot.emo_unicode import EMOTICONS_EMO

def remove_emoticons(text):
    # Escape special characters in emoticons and join them into a pattern
    emoticons = [re.escape(emot) for emot in EMOTICONS_EMO]
    emoticon_pattern = re.compile('|'.join(emoticons))
    return emoticon_pattern.sub('', text)

text = "Hello :-)"
cleaned_text = remove_emoticons(text)
print(cleaned_text)
```

Output:

```
Hello
```

```python
import re
from emot.emo_unicode import EMOTICONS_EMO
def convert_emoticons(text):
    for emot in EMOTICONS_EMO:
        escaped_emot = re.escape(emot) # Escape special characters in emoticon
        text = re.sub(r'(' + escaped_emot + ')',
"_".join(EMOTICONS_EMO[emot].replace(",", "").split()), text)
    return text

text = "Hello :-) :-)"
converted_text = convert_emoticons(text)
print(converted_text)
```

Output:

```
Hello Happy_face_smiley Happy_face_smiley
```

URL Removal

```
def remove_urls(text):
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    return url_pattern.sub(r'', text)

text = "Driverless AI NLP blog post on https://www.h2o.ai/blog/detecting-sarcasm-is-
difficult-but-ai-may-have-an-answer/"
remove_urls(text)
```

Output:

```
'Driverless AI NLP blog post on '
```

HTML Removal

```
def remove_html(text):
    html_pattern = re.compile('<.*?>')
    return html_pattern.sub(r'', text)

text = """<div>
<h1> H2O</h1>
<p> AutoML</p>
<a href="https://www.h2o.ai/products/h2o-driverless-ai/"> Driverless AI</a>
</div>"""

print(remove_html(text))
```

Output:

```
H2O
AutoML
Driverless AI
```

Handling Contractions

```
import contractions

def expand_contractions(text):

    expanded_text = contractions.fix(text)
    return expanded_text

expanded_text = expand_contractions("I'm going to DHS.I don't want to be late")
print(expanded_text)
```

Output:

```
 I am going to DHS.I do not want to be late
```

Duplication Removal

```
#### Removal of duplicate characters
text = "howdyyy!!! how are you doingggg????"

#Write your code to get the unique characters
def removeDupWithoutOrder(str):
    return "".join(set(str))

removeDupWithoutOrder(text)
```

Output:

```
' gheuno!ywri?ad'
```

```
#Write your code to get all the unique characters in same order
from collections import OrderedDict
def removeDupWithOrder(str):
    return "".join(OrderedDict.fromkeys(str))

removeDupWithOrder(text)
```

Output:

```
'howdy! areuing?'
```

```python
## Write your code to replace those duplicate characters that occur more than once in
chat format
sent = []
for char in text.split():
  char = removeDupWithOrder(char)
  sent.append(char)
sent = " ".join(sent)
print(sent)
```

Output:

```
howdy! how are you doing?
```

Handling Chat words

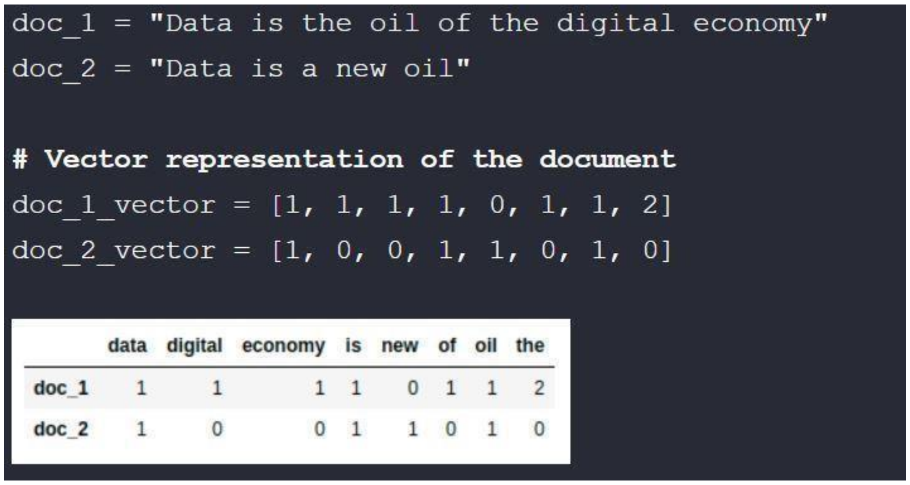
```python
# Define your custom dictionary
import nltk
from nltk import word_tokenize
CHAT_WORDS_DICT = {
    "BRB": "Be Right Back",
    "LOL": "Laughing Out Loud",
    "OMG": "Oh My God",
    "IMHO" :"In My Honest/Humble Opinion"

    # Add more chat words and full forms as needed
}

# Function to convert chat words into full forms
def convert_chat_words(text):
    # Tokenize the input text into words
    words = word_tokenize(text)

    # Initialize a list to store the converted words
    converted_words = []

    # Iterate through the words in the input text
    for word in words:
        # Check if the word is in the custom dictionary
        if word.upper() in CHAT_WORDS_DICT:
            converted_words.append(CHAT_WORDS_DICT[word.upper()])
        else:
            # If not found in the dictionary, keep the word unchanged
            converted_words.append(word)
    # Join the modified words back into a sentence
    converted_text = " ".join(converted_words)

    return converted_text

# Example usage
input_text = "IMHO, BRB, LOL! OMG, it's so funny!"
output_text = convert_chat_words(input_text)
print(output_text)
```

Output:

```
In My Honest/Humble Opinion , Be Right Back , Laughing Out Loud ! Oh My God , it 's so funny !
```

3. Write Text Similarity Techniques

```
doc_1 = "Data is the oil of the digital economy"
doc_2 = "Data is a new oil"

data = [doc_1, doc_2]
```



```
doc_1 = "Data is the oil of the digital economy"
doc_2 = "Data is a new oil"

# Vector representation of the document
doc_1_vector = [1, 1, 1, 1, 0, 1, 1, 2]
doc_2_vector = [1, 0, 0, 1, 1, 0, 1, 0]
```

	data	digital	economy	is	new	of	oil	the
doc_1	1	1	1	1	0	1	1	2
doc_2	1	0	0	1	1	0	1	0

```
from sklearn.feature_extraction.text import CountVectorizer

count_vectorizer = CountVectorizer()
vector_matrix = count_vectorizer.fit_transform(data)
vector_matrix
```

Output:

```
<2x8 sparse matrix of type '<class 'numpy.int64'>'
        with 11 stored elements in Compressed Sparse Row format>
```

```
tokens = count_vectorizer.get_feature_names_out()
tokens
```

Output:

```
array(['data', 'digital', 'economy', 'is', 'new', 'of', 'oil', 'the'],
      dtype=object)
```

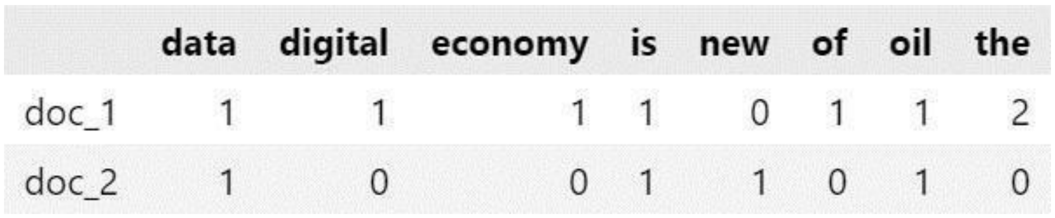
```
vector_matrix.toarray()
```

Output:

```
array([[1, 1, 1, 1, 0, 1, 1, 2],
       [1, 0, 0, 1, 1, 0, 1, 0]], dtype=int64)
```

```
import pandas as pd

def create_dataframe(matrix, tokens):

    doc_names = [f'doc_{i+1}' for i, _ in enumerate(matrix)]
    df = pd.DataFrame(data=matrix, index=doc_names, columns=tokens)
    return(df)

create_dataframe(vector_matrix.toarray(),tokens)
```

Output:

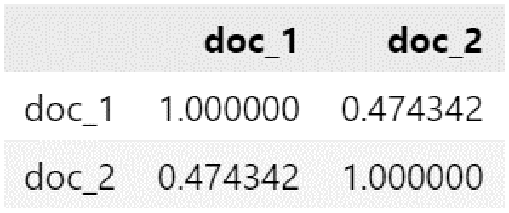


	data	digital	economy	is	new	of	oil	the
doc_1	1	1	1	1	0	1	1	2
doc_2	1	0	0	1	1	0	1	0

Cosine Similarity

```
from sklearn.metrics.pairwise import cosine_similarity

cosine_similarity_matrix = cosine_similarity(vector_matrix)
create_dataframe(cosine_similarity_matrix,['doc_1','doc_2'])
```

Output:

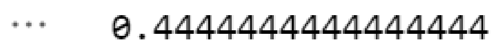


	doc_1	doc_2
doc_1	1.000000	0.474342
doc_2	0.474342	1.000000

Jaccard Similarity

```
def Jaccard_Similarity(doc1, doc2):

    # List the unique words in a document
    words_doc1 = set(doc1.lower().split())
    words_doc2 = set(doc2.lower().split())

    # Find the intersection of words list of doc1 & doc2
    intersection = words_doc1.intersection(words_doc2)

    # Find the union of words list of doc1 & doc2
    union = words_doc1.union(words_doc2)

    # Calculate Jaccard similarity score
    return float(len(intersection)) / len(union)

doc_1 = "Data is the new oil of the digital economy"
doc_2 = "Data is a new oil"

Jaccard_Similarity(doc_1,doc_2)
```

Output:

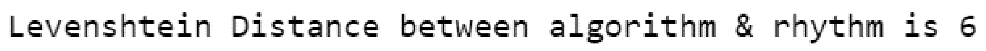


```
...    0.4444444444444444
```

Levenshtein Distance

```
import enchant
string1 = "algorithm"
string2 = "rhythm"

l_dist = enchant.utils.levenshtein(string1, string2)

print("Levenshtein Distance between "+string1+" & "+string2+" is " + str(l_dist))
```

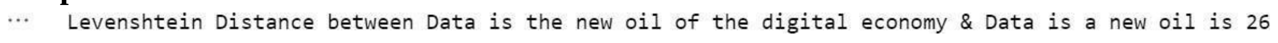
Output:



```
Levenshtein Distance between algorithm & rhythm is 6
```

```
string1 = "Data is the new oil of the digital economy" #"prime"
string2 = "Data is a new oil" #"time"
l_dist = enchant.utils.levenshtein(string1, string2)
print("Levenshtein Distance between "+string1+" & "+string2+" is " + str(l_dist))
```

Output:



```
...   Levenshtein Distance between Data is the new oil of the digital economy & Data is a new oil is 26
```

4.Generate Unigram, Bigram, N-grams using nltk library

An N-gram is an N-token sequence of words: a 2-gram (more commonly called a bigram) is a two-word se quence of words like "please turn", "turn your", or "your homework", and a 3-gram is a three-word sequ ence of words like "please turn your", or "turn your homework"
A bag-of-bigrams representation is much more powerful than a bag-of-words, and in many cases proves v ery hard to beat.

Unigram

```
from nltk.util import ngrams

n = 1
sentence = 'You will face many defeats in life, but never let yourself be defeated.'
unigrams = ngrams(sentence.split(), n)

for item in unigrams:
    print(item)
```

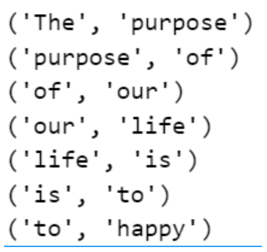
Output:

```
('You',)
('will',)
('face',)
('many',)
('defeats',)
('in',)
('life,',)
('but',)
('never',)
('let',)
('yourself',)
('be',)
('defeated.',)
```

Bigram

```
from nltk.util import ngrams

n = 2
sentence = 'The purpose of our life is to happy'
unigrams = ngrams(sentence.split(), n)
for item in unigrams:
    print(item)
```

Output:

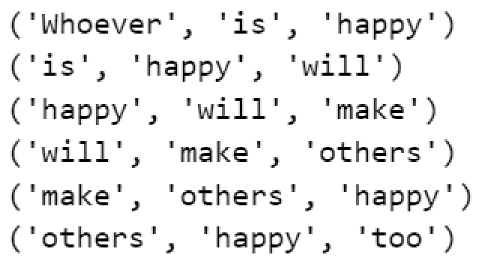


```
('The', 'purpose')
('purpose', 'of')
('of', 'our')
('our', 'life')
('life', 'is')
('is', 'to')
('to', 'happy')
```

Trigram

```python
from nltk.util import ngrams

n = 3
sentence = 'Whoever is happy will make others happy too'
unigrams = ngrams(sentence.split(), n)

for item in unigrams:
    print(item)
```

Output:



```
('Whoever', 'is', 'happy')
('is', 'happy', 'will')
('happy', 'will', 'make')
('will', 'make', 'others')
('make', 'others', 'happy')
('others', 'happy', 'too')
```

N-grams

```python
from nltk.util import ngrams

def ngram_convertor(sentence,n=3):

    ngram_sentence = ngrams(sentence.split(), n)
    for item in ngram_sentence:
        print(item)
sentence = "Life is either a daring adventure or nothing at all"
ngram_convertor(sentence,3)
```

Output:

```
('Life', 'is', 'either')
('is', 'either', 'a')
('either', 'a', 'daring')
('a', 'daring', 'adventure')
('daring', 'adventure', 'or')
('adventure', 'or', 'nothing')
('or', 'nothing', 'at')
('nothing', 'at', 'all')
```

Everygrams

```
from nltk.util import everygrams

message = "who let the dogs out"
msg_split = message.split()
list(everygrams(msg_split))
```

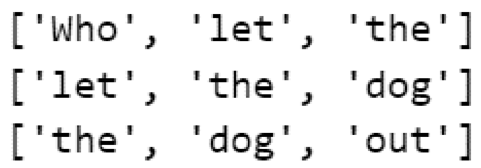
Output:

```
[('who',),
 ('who', 'let'),
 ('who', 'let', 'the'),
 ('who', 'let', 'the', 'dogs'),
 ('who', 'let', 'the', 'dogs', 'out'),
 ('let',),
 ('let', 'the'),
 ('let', 'the', 'dogs'),
 ('let', 'the', 'dogs', 'out'),
 ('the',),
 ('the', 'dogs'),
 ('the', 'dogs', 'out'),
 ('dogs',),
 ('dogs', 'out'),
 ('out',)]
```

Using TextBlob

```python
from textblob import TextBlob

data = 'Who let the dog out'
num = 3

n_grams = TextBlob(data).ngrams(num)
for grams in n_grams:
    print(grams)
```

Output:



```
['Who', 'let', 'the']
['let', 'the', 'dog']
['the', 'dog', 'out']
```

5. Text Classification using Naïve Bayes without Dataset

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a s ingle algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of featur es being classified is independent of each other.
Text Analysis is a major application field for machine learning algorithms. However, the raw data, a sequ ence of symbols (i.e. strings) cannot be fed directly to the algorithms themselves as most of them expect numerical feature vectors with a fixed size rather than the raw text documents with variable length.

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a s ingle algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of f eatures being classified is independent of each other. The dataset is divided into two parts, namely, the fe ature matrix and the response/target vector. The Feature matrix (X) contains all the vectors(rows) of the d ataset in
Each vector consists of the value of dependent features. The number of features is d i.e. X = (x1,x2,x2, xd). The Response/target vector (y) contains the value of the class/group variable for each row of the feature matrix.

Bayes' Theorem finds the probability of an event occurring given the probability of another event that ha s already occurred. Bayes' theorem is stated mathematically as follows:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

- A and B are called events.
- P(A | B) is the probability of event A, given the event B is true (has occured). Event B is also term ed as evidence.
- P(A) is the priori of A (the prior independent probability, i.e. probability of event before evidence is seen).
- P(B | A) is the probability of B given event A, i.e. probability of event B after evidence A is seen.

Given a data matrix **X** and a target vector **y,** we state our problem as:

where, **y** is **class variable** and **X** is a **dependent feature vector with dimension d i.e. X = (x1,x2,x2,xd)**

$$P(y \mid X) = \frac{P(X \mid y)P(y)}{P(X)}$$

,
where **d** is the number of variables/features of the sample.

- P(y|X) is the probability of observing the class **y** given the sample **X** with **X = (x1,x2,x2, xd),** whe re **d** is the number of variables/features of the sample.

Now the "naïve" conditional independence assumptions come into play: assume that all features in **X** ar e mutually independent, conditional on the category **y**:

$$P(y \mid x_1, \ldots, x_d) = \frac{P(y)\prod_{i=1}^{d} P(x_i|y)}{P(x_1)P(x_2)\ldots P(x_d)}$$

The denominator remains constant for a given input, so we can remove that term:

$$P(y \mid x_1, \ldots, x_d) \propto P(y)\prod_{i=1}^{d} P(x_i \mid y)$$

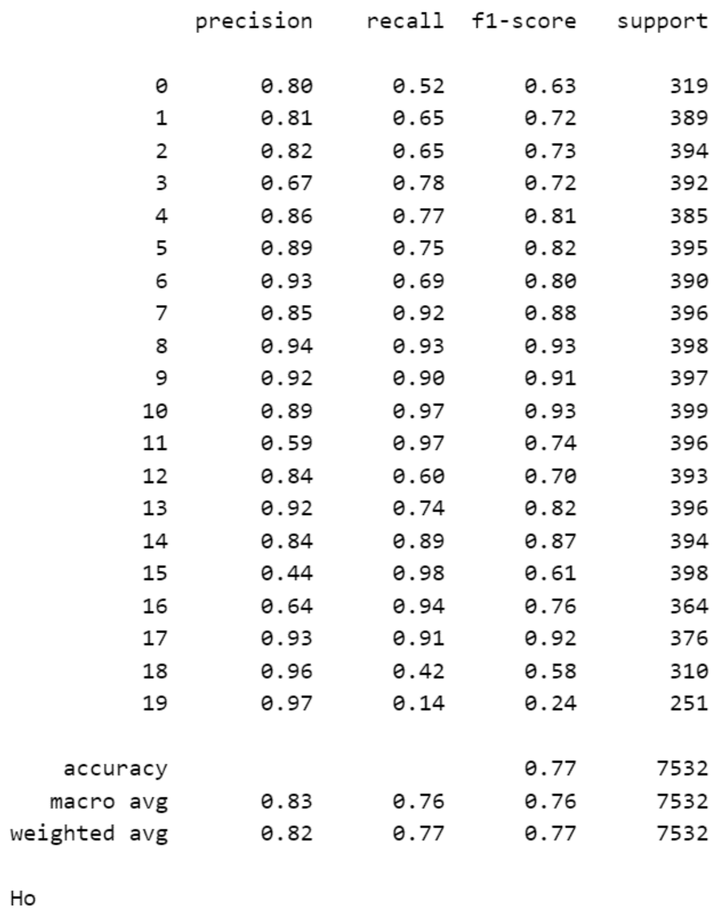
Finally, to find the probability of a given **sample** for all possible values of the class variable *y*, we just ne ed to find the output with maximum probability:

$$y = \text{argmax}_y\, P(y)\prod_{i=1}^{n} P(x_i \mid y)$$

```python
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn import metrics
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.datasets import fetch_20newsgroups

newsgroups_train = fetch_20newsgroups(subset='train')
newsgroups_test = fetch_20newsgroups(subset='test')
X_train = newsgroups_train.data
X_test = newsgroups_test.data
y_train = newsgroups_train.target
y_test = newsgroups_test.target

text_clf = Pipeline([('vect', CountVectorizer()),
                     ('tfidf', TfidfTransformer()),
                     ('clf', MultinomialNB()),
                     ])
```

```
text_clf.fit(X_train, y_train)
predicted = text_clf.predict(X_test)
print(metrics.classification_report(y_test, predicted))
```

Output:



```
              precision    recall  f1-score   support

           0       0.80      0.52      0.63       319
           1       0.81      0.65      0.72       389
           2       0.82      0.65      0.73       394
           3       0.67      0.78      0.72       392
           4       0.86      0.77      0.81       385
           5       0.89      0.75      0.82       395
           6       0.93      0.69      0.80       390
           7       0.85      0.92      0.88       396
           8       0.94      0.93      0.93       398
           9       0.92      0.90      0.91       397
          10       0.89      0.97      0.93       399
          11       0.59      0.97      0.74       396
          12       0.84      0.60      0.70       393
          13       0.92      0.74      0.82       396
          14       0.84      0.89      0.87       394
          15       0.44      0.98      0.61       398
          16       0.64      0.94      0.76       364
          17       0.93      0.91      0.92       376
          18       0.96      0.42      0.58       310
          19       0.97      0.14      0.24       251

    accuracy                           0.77      7532
   macro avg       0.83      0.76      0.76      7532
weighted avg       0.82      0.77      0.77      7532

Ho
```

6. Text Classification using Decision Tree without Dataset

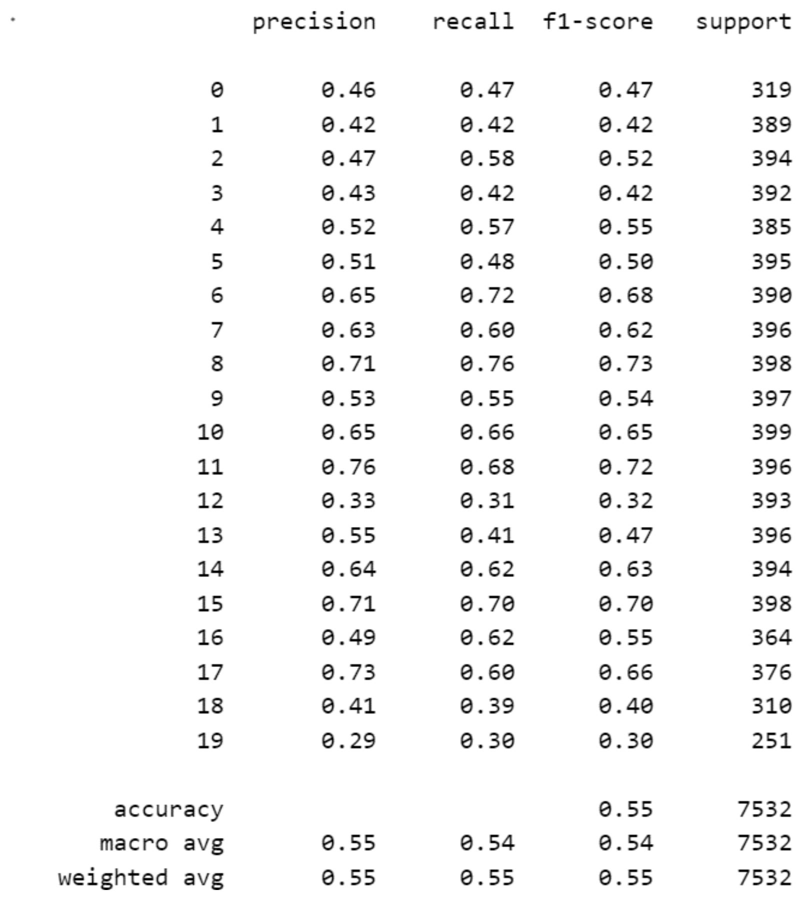
A Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

It is called a decision tree because similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure. In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.A decision tree simply asks a question and based on the answer (Yes/No), it further split the tree into subtrees.

```python
from sklearn import tree
from sklearn.pipeline import Pipeline
from sklearn import metrics
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.datasets import fetch_20newsgroups

newsgroups_train = fetch_20newsgroups(subset='train')
newsgroups_test = fetch_20newsgroups(subset='test')
X_train = newsgroups_train.data
X_test = newsgroups_test.data
y_train = newsgroups_train.target
y_test = newsgroups_test.target

text_clf = Pipeline([('vect', CountVectorizer()),
                     ('tfidf', TfidfTransformer()),
                     ('clf', tree.DecisionTreeClassifier()),
                     ])
text_clf.fit(X_train, y_train)
predicted = text_clf.predict(X_test)
print(metrics.classification_report(y_test, predicted))
```

Output:



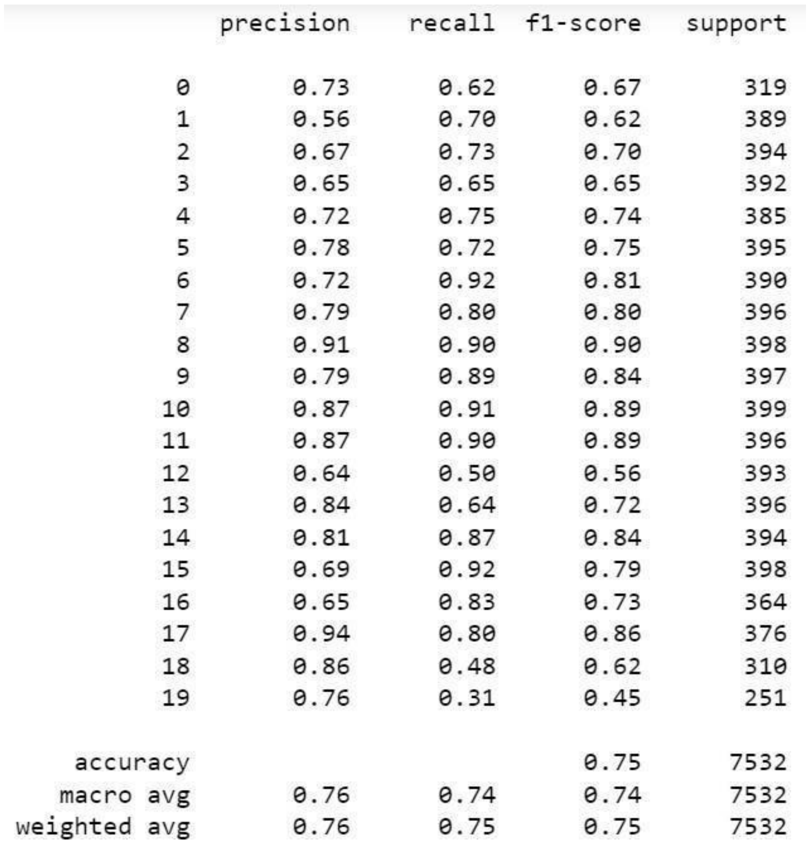
```
.                 precision    recall  f1-score   support

           0       0.46      0.47      0.47       319
           1       0.42      0.42      0.42       389
           2       0.47      0.58      0.52       394
           3       0.43      0.42      0.42       392
           4       0.52      0.57      0.55       385
           5       0.51      0.48      0.50       395
           6       0.65      0.72      0.68       390
           7       0.63      0.60      0.62       396
           8       0.71      0.76      0.73       398
           9       0.53      0.55      0.54       397
          10       0.65      0.66      0.65       399
          11       0.76      0.68      0.72       396
          12       0.33      0.31      0.32       393
          13       0.55      0.41      0.47       396
          14       0.64      0.62      0.63       394
          15       0.71      0.70      0.70       398
          16       0.49      0.62      0.55       364
          17       0.73      0.60      0.66       376
          18       0.41      0.39      0.40       310
          19       0.29      0.30      0.30       251

    accuracy                           0.55      7532
   macro avg       0.55      0.54      0.54      7532
weighted avg       0.55      0.55      0.55      7532
```

7. Text Classification using Random Forest without Dataset

Random Forest is an ensemble technique used in supervised learning for both classification and regression problems. It is primarily preferred for classification tasks. This method employs a collection of decision trees, where individual trees make predictions based on subsets of the data and features.
Random Forest introduces randomness by using bootstrapping (randomly selecting data samples with replacement) and feature selection (choosing random subsets of features for each tree). This randomness reduces overfitting and enhances model stability.
The ensemble combines the predictions of these trees through majority voting for classification and averaging for regression. This averaging helps smooth out individual tree errors, resulting in a robust and accurate predictive model.
Like decision trees, Random Forest can reveal feature importance, aiding in feature selection and data understanding. It is widely used across various domains, including finance, healthcare, image analysis, and natural language processing, due to its versatility and predictive power.

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn import metrics
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.datasets import fetch_20newsgroups

newsgroups_train = fetch_20newsgroups(subset='train')
newsgroups_test = fetch_20newsgroups(subset='test')
X_train = newsgroups_train.data
X_test = newsgroups_test.data
y_train = newsgroups_train.target
y_test = newsgroups_test.target

text_clf = Pipeline([('vect', CountVectorizer()),
                     ('tfidf', TfidfTransformer()),
                     ('clf', RandomForestClassifier(n_estimators=100)),
                     ])

text_clf.fit(X_train, y_train)
predicted = text_clf.predict(X_test)
print(metrics.classification_report(y_test, predicted))
```

Output:



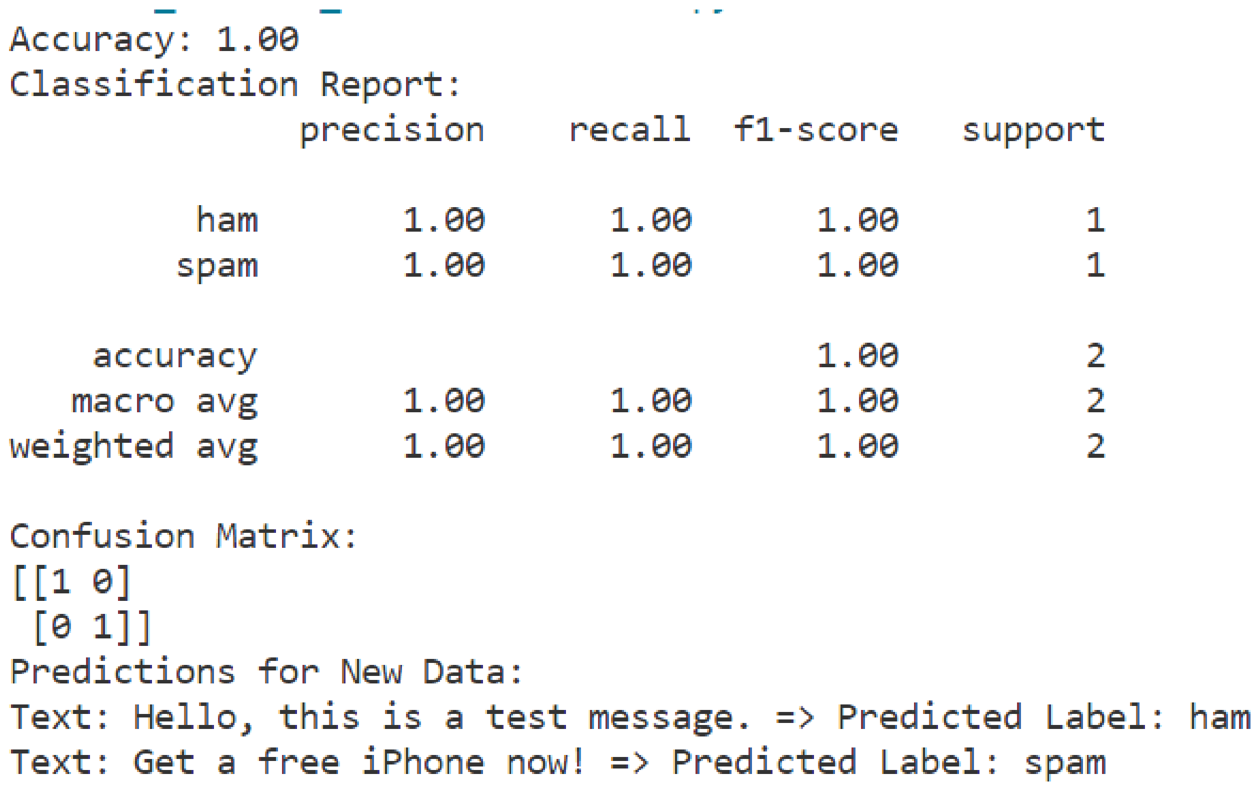
```
              precision    recall  f1-score   support

           0       0.73      0.62      0.67       319
           1       0.56      0.70      0.62       389
           2       0.67      0.73      0.70       394
           3       0.65      0.65      0.65       392
           4       0.72      0.75      0.74       385
           5       0.78      0.72      0.75       395
           6       0.72      0.92      0.81       390
           7       0.79      0.80      0.80       396
           8       0.91      0.90      0.90       398
           9       0.79      0.89      0.84       397
          10       0.87      0.91      0.89       399
          11       0.87      0.90      0.89       396
          12       0.64      0.50      0.56       393
          13       0.84      0.64      0.72       396
          14       0.81      0.87      0.84       394
          15       0.69      0.92      0.79       398
          16       0.65      0.83      0.73       364
          17       0.94      0.80      0.86       376
          18       0.86      0.48      0.62       310
          19       0.76      0.31      0.45       251

    accuracy                           0.75      7532
   macro avg       0.76      0.74      0.74      7532
weighted avg       0.76      0.75      0.75      7532
```

8. Text Classification using Naïve Bayes with Dataset

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load your CSV dataset into a pandas DataFrame
# Replace 'your_dataset.csv' with the actual path to your CSV file
df = pd.read_excel('nv.xlsx')

# Assuming your dataset has two columns: 'text' (features) and 'label' (target)
X = df['text']
y = df['label']

# 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)

# Create a CountVectorizer to convert text data into numerical format
vectorizer = CountVectorizer()
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)

# Create a Multinomial Naive Bayes classifier
nb_classifier = MultinomialNB()

# Train the classifier on the training data
nb_classifier.fit(X_train_vectorized, y_train)

# Make predictions on the test data
y_pred = nb_classifier.predict(X_test_vectorized)

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Generate a classification report
class_report = classification_report(y_test, y_pred)
print('Classification Report:')
print(class_report)

# Generate a confusion matrix
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)

new_text = ["Hello, this is a test message.", "Get a free iPhone now!"]
new_text_vectorized = vectorizer.transform(new_text)
new_predictions = nb_classifier.predict(new_text_vectorized)

# Print the predictions for new data
print('Predictions for New Data:')
for text, label in zip(new_text, new_predictions):
    print(f'Text: {text} => Predicted Label: {label}')
```

Output:



```
Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   support

         ham       1.00      1.00      1.00         1
        spam       1.00      1.00      1.00         1

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2

Confusion Matrix:
[[1 0]
 [0 1]]
Predictions for New Data:
Text: Hello, this is a test message. => Predicted Label: ham
Text: Get a free iPhone now! => Predicted Label: spam
```

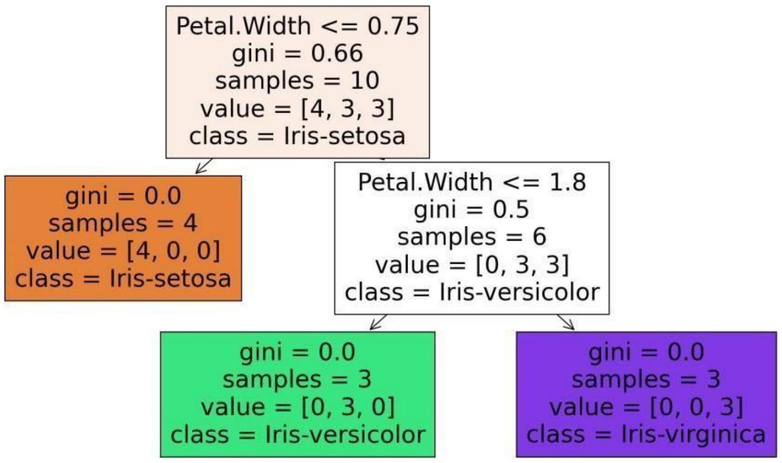
9.Text Classification using Decision Tree with Dataset

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt

# Load your CSV dataset into a pandas DataFrame
# Replace 'your_dataset.csv' with the actual path to your CSV file
df = pd.read_csv('des.csv')

# Assuming your dataset has four columns: 'feature1', 'feature2', 'feature3',
'feature4', and 'label'
X = df[['feature1', 'feature2', 'feature3', 'feature4']]  # Features
y = df['label']  # Target variable

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Create a Decision Tree classifier
dt_classifier = DecisionTreeClassifier()

# Train the classifier on the training data
dt_classifier.fit(X_train, y_train)

# Visualize the Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(dt_classifier, filled=True, feature_names=X.columns,
class_names=df['label'].unique())
plt.show()

# Make predictions on the test data
y_pred = dt_classifier.predict(X_test)

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Generate a classification report
class_report = classification_report(y_test, y_pred)
print('Classification Report:')
print(class_report)
```

```
# Generate a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)

# Make predictions on new data
new_data = pd.DataFrame({'feature1': [5.1], 'feature2': [3.5], 'feature3': [1.4],
'feature4': [0.2]})
new_predictions = dt_classifier.predict(new_data)
print('Predictions for New Data:')
print(new_predictions)
```

Output:

```
Accuracy: 0.80
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         1
Iris-versicolor       0.67      1.00      0.80         2
 Iris-virginica       1.00      0.50      0.67         2

       accuracy                           0.80         5
      macro avg       0.89      0.83      0.82         5
   weighted avg       0.87      0.80      0.79         5

Confusion Matrix:
[[1 0 0]
 [0 2 0]
 [0 1 1]]
Predictions for New Data:
['Iris-setosa']
```



10.Text Classification using K Neighbours with Dataset

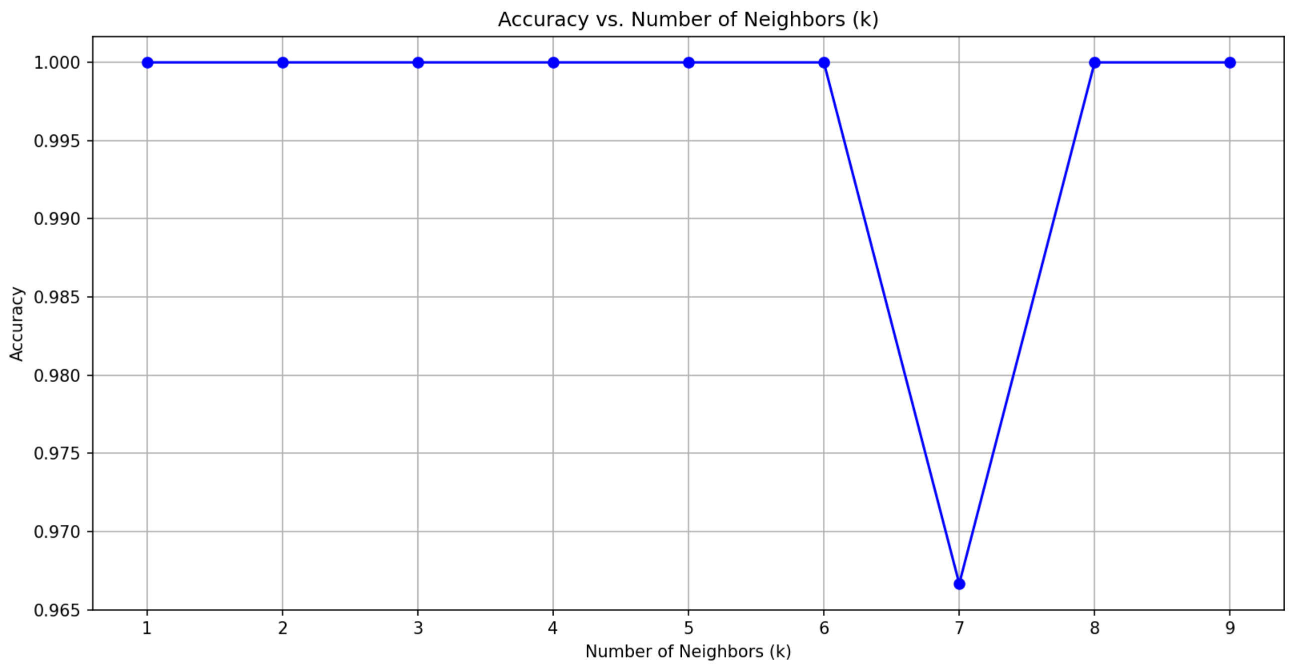
Text Classification using K-Nearest Neighbors (K-NN) is a supervised machine learning technique employed in various natural language processing tasks. K-NN is particularly well-suited for tasks like sentiment analysis, topic classification, and document categorization.
In K-NN, each document is represented as a point in a high-dimensional space, and classification is determined by examining the labels of the K-nearest neighboring documents based on similarity metrics such as cosine similarity or Euclidean distance. The class that appears most frequently among these neighbors becomes the predicted class for the new document.
This method can excel in scenarios where documents with similar content tend to share the same labels, making it valuable for text classification problems. K-NN's flexibility makes it an alternative to decision tree-based approaches, especially when dealing with non-linear decision boundaries or noisy text data. However, selecting an appropriate value for K is essential, as smaller K values might lead to overfitting, while larger K values might introduce bias. Additionally, choosing the right distance metric is critical for the algorithm's performance.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder

# Load your CSV dataset into a pandas DataFrame
# Replace 'your_dataset.csv' with the actual path to your CSV file
df = pd.read_csv('iris.csv')
tar=LabelEncoder()
y=tar.fit_transform(list(df['Species'])) # Target variable
# Assuming your dataset has four columns: 'feature1', 'feature2', 'feature3',
'feature4', and 'label'
X = df[['Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width']] # Features
# 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)
# Create a list of possible k values to try
k_values = list(range(1, 10))  # You can adjust the range as needed
# Create empty lists to store accuracy scores for different k values
accuracy_scores = []
# Iterate through different k values
for k in k_values:
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(X_train, y_train)
    y_pred = knn_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_scores.append(accuracy)

# Plot the accuracy scores for different k values
plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracy_scores, marker='o', linestyle='-', color='b')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. Number of Neighbors (k)')
plt.grid(True)
plt.show()

# Find the optimal k value (the "elbow" point)
optimal_k = k_values[np.argmax(accuracy_scores)]
print(f'Optimal k value: {optimal_k}')
# Create a KNN classifier with the optimal k value
knn_classifier = KNeighborsClassifier(n_neighbors=optimal_k)

# Train the classifier on the training data
knn_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = knn_classifier.predict(X_test)

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy with optimal k: {accuracy:.2f}')

# Generate a classification report
class_report = classification_report(y_test, y_pred)
print('Classification Report:')
print(class_report)

# Generate a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)

# Make predictions on new data
new_data = pd.DataFrame({'Sepal.Length': [5.1], 'Sepal.Width': [3.5], 'Petal.Length':
[1.4], 'Petal.Width': [0.2]})
new_predictions = knn_classifier.predict(new_data)
l=['Iris-setosa','Iris-versicolor','Iris-virginica']
```

```
print('Predictions for New Data:')
print(l[new_predictions[0]])
```

Output:

```
Optimal k value: 1
Accuracy with optimal k: 1.00
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       1.00      1.00      1.00        11

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30

Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Predictions for New Data:
['Iris-setosa']
```



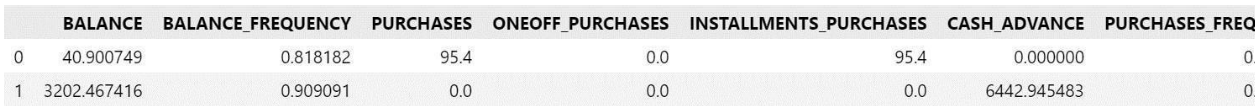
11. Text Clustering using Hierarchical Clustering

Agglomerative Clustering is also known as the bottom-up approach or hierarchical agglomerative clustering (HAC). A structure that is more informative than the unstructured set of clusters returned by flat clustering. This clustering algorithm does not require us to prespecify the number of clusters. Bottom-up algorithms treat each data as a singleton cluster at the outset and then successively agglomerate pairs of clusters until all clusters have been merged into a single cluster that contains all data.

Computing Distance Matrix: While merging two clusters we check the distance between two every pair of clusters and merge the pair with least distance/most similarity. But the question is how is that distance determined. There are different ways of defining Inter Cluster distance/similarity. Some of them are:

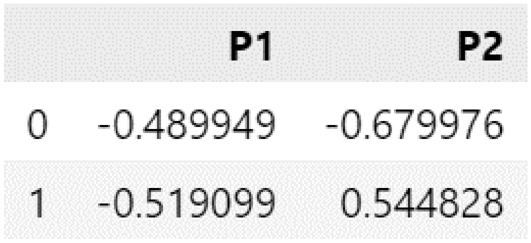
1. Min Distance: Find the minimum distance between any two points of the cluster.
2. Max Distance: Find the maximum distance between any two points of the cluster.
3. Group Average: Find the average of the distance between every two points of the clusters.
4. Ward’s Method: The similarity of two clusters is based on the increase in squared error when two clusters are merged.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as shc

raw_df = pd.read_csv('CC GENERAL.csv')
raw_df = raw_df.drop('CUST_ID', axis = 1)
raw_df.fillna(method ='ffill', inplace = True)
raw_df.head(2)
```



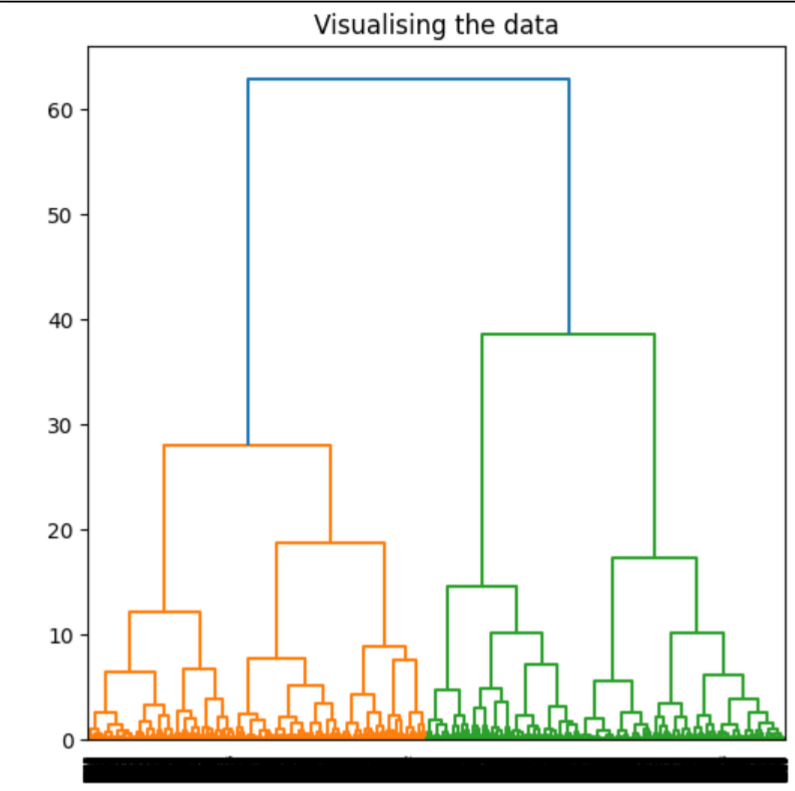
	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQ
0	40.900749	0.818182	95.4	0.0	95.4	0.000000	0.
1	3202.467416	0.909091	0.0	0.0	0.0	6442.945483	0.

```python
# Standardize  data
scaler = StandardScaler()
scaled_df = scaler.fit_transform(raw_df)

# Normalizing the  Data
normalized_df = normalize(scaled_df)

# Converting the numpy array into a pandas DataFrame
normalized_df = pd.DataFrame(normalized_df)

# Reducing the dimensions of the data
pca = PCA(n_components = 2)
X_principal = pca.fit_transform(normalized_df)
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']

X_principal.head(2)
```

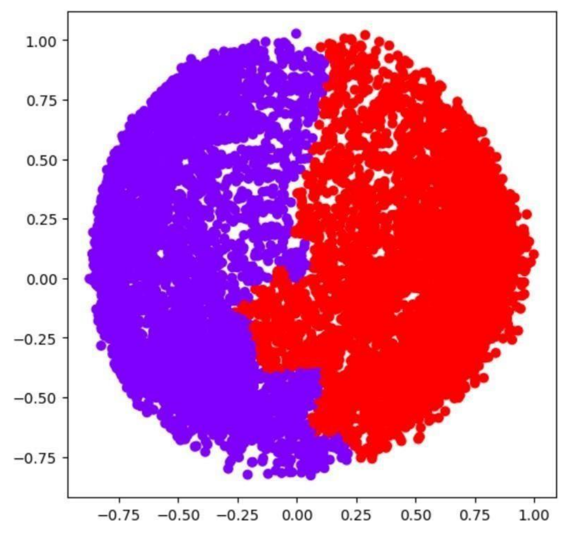


	P1	P2
0	-0.489949	-0.679976
1	-0.519099	0.544828

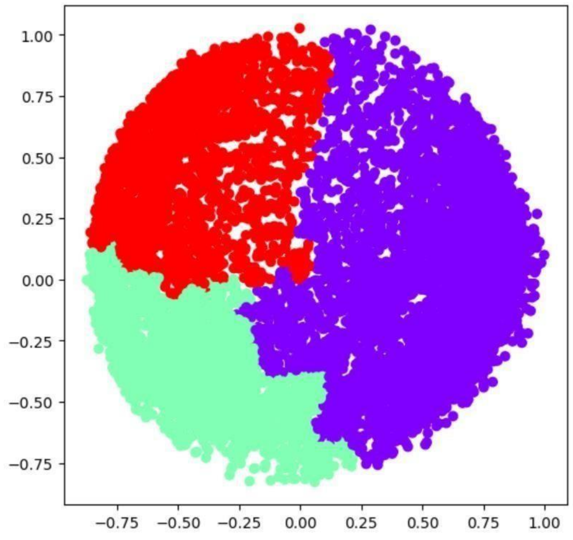
```python
plt.figure(figsize =(6, 6))
plt.title('Visualising the data')
Dendrogram = shc.dendrogram(shc.linkage(X_principal, method ='ward'))
```



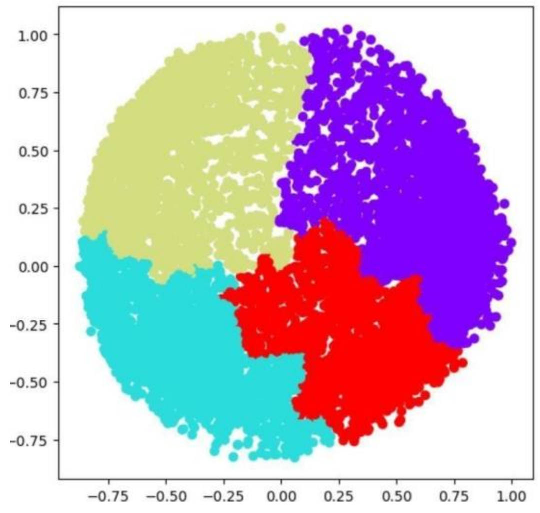
```
ac2 = AgglomerativeClustering(n_clusters = 2)
# Visualizing the clustering
plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],  c = ac2.fit_predict(X_principal),
cmap ='rainbow')
plt.show()
```



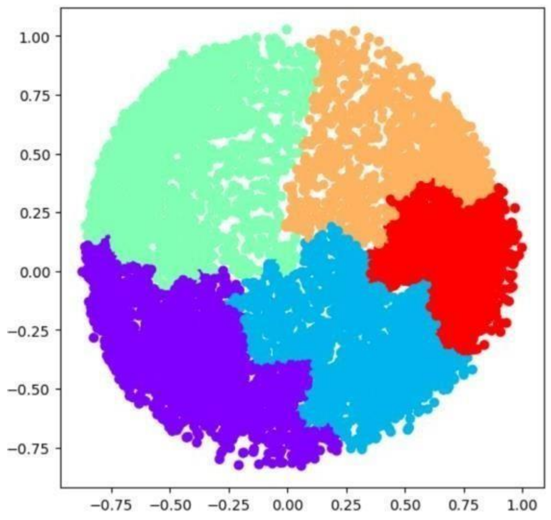
```
ac3 = AgglomerativeClustering(n_clusters = 3)
# Visualizing the clustering
plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],c = ac3.fit_predict(X_principal),
cmap ='rainbow')
plt.show()
```



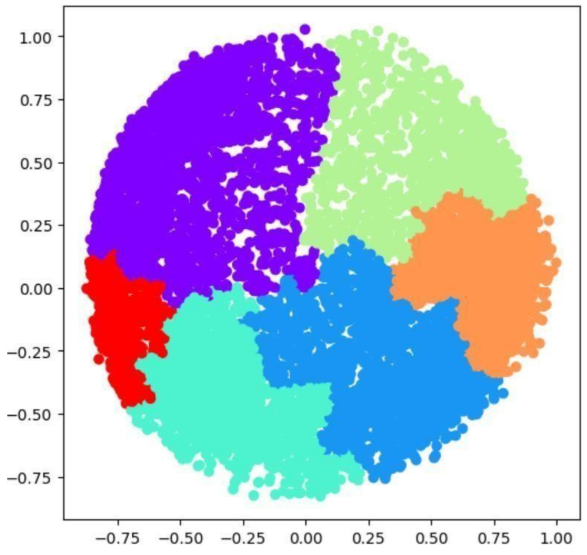
```
ac4 = AgglomerativeClustering(n_clusters = 4)
# Visualizing the clustering
plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],c = ac4.fit_predict(X_principal),
cmap ='rainbow')
plt.show()
```



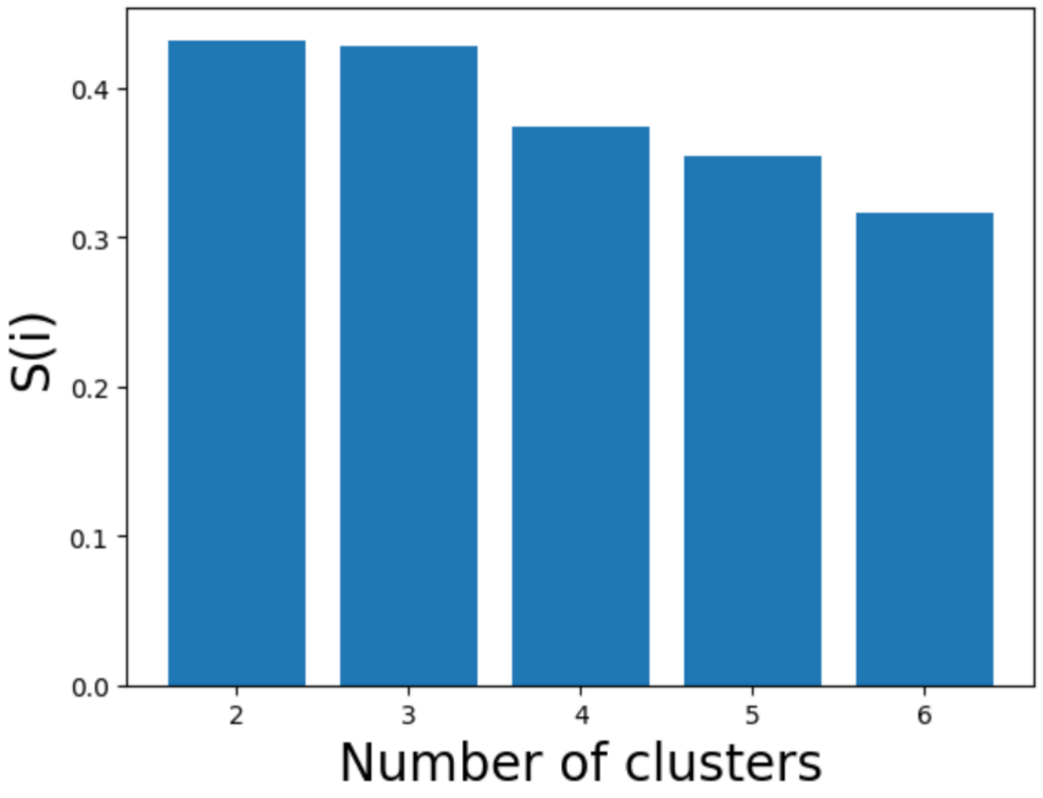
```
ac5 = AgglomerativeClustering(n_clusters = 5)
# Visualizing the clustering
plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],c = ac5.fit_predict(X_principal),
cmap ='rainbow')
plt.show()
```



```
ac6 = AgglomerativeClustering(n_clusters = 6)
# Visualizing the clustering
plt.figure(figsize =(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],c = ac6.fit_predict(X_principal),
cmap ='rainbow')
plt.show()
```



```
k = [2, 3, 4, 5, 6]
# Appending the silhouette scores of the different models to the list
silhouette_scores = []
silhouette_scores.append( silhouette_score(X_principal, ac2.fit_predict(X_principal)))
silhouette_scores.append( silhouette_score(X_principal, ac3.fit_predict(X_principal)))
silhouette_scores.append( silhouette_score(X_principal, ac4.fit_predict(X_principal)))
silhouette_scores.append( silhouette_score(X_principal, ac5.fit_predict(X_principal)))
silhouette_scores.append( silhouette_score(X_principal, ac6.fit_predict(X_principal)))
# Plotting a bar graph to compare the results
plt.bar(k, silhouette_scores)
plt.xlabel('Number of clusters', fontsize = 20)
plt.ylabel('S(i)', fontsize = 20)
plt.show()
```



12. Text Clustering using K Means Clustering

The algorithm will categorize the items into k groups or clusters of similarity. To calculate that similarity, we will use the euclidean distance as a measurement.

The algorithm works as follows:

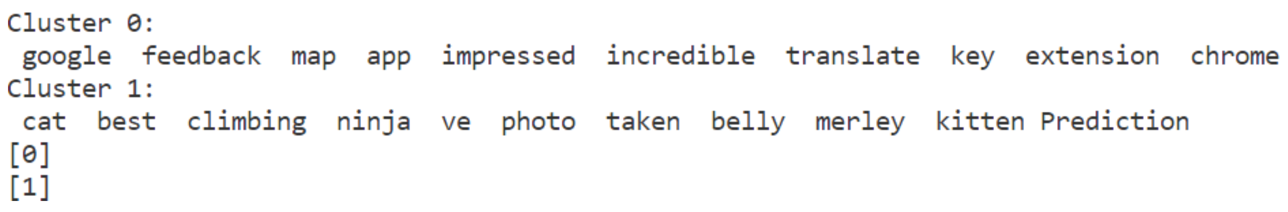
First, we initialize k points, called means or cluster centroids, randomly.

We categorize each item to its closest mean and we update the mean's coordinates, which are the averages of the items categorized in that cluster so far.

We repeat the process for a given number of iterations and at the end, we have our clusters. The "points" mentioned above are called means because they are the mean values of the items categorized in them. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another method is to initialize the means at random values between the boundaries of the data set.

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score

documents = [
"This little kitty came to play when I was eating at a restaurant." ,
"Merley has the best squooshy kitten belly.",
"Google Translate app is incredible.",
"If you open 100 tab in google you get a smiley face.",
"Best cat photo I've ever taken.",
"Climbing ninja cat.",
"Impressed with google map feedback.",
"Key promoter extension for Google Chrome."
]
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(documents)
true_k = 2
model = KMeans(n_clusters=true_k, init='k-means++', max_iter=100, n_init=1)
model.fit(X)
print("Top terms per cluster:")
order_centroids = model.cluster_centers_.argsort()[:,::-1]
terms = vectorizer.get_feature_names_out()
for i in range(true_k):
    print("\nCluster %d:" % i)
    for ind in order_centroids[i, :10]:
        print(' %s' % terms[ind],end=' ')
print("Prediction")
Y = vectorizer.transform(["chrome browser to open."])
prediction = model.predict(Y)
print(prediction)
Y = vectorizer.transform(["My cat is hungry."])
prediction =  model.predict(Y)
print(prediction)
```

Output:



```
Cluster 0:
 google  feedback  map  app  impressed  incredible  translate  key  extension  chrome
Cluster 1:
 cat  best  climbing  ninja  ve  photo  taken  belly  merley  kitten Prediction
[0]
[1]
```

13. Statistical analysis of IMDB social media

Statistical analysis is the process of collecting and analyzing data in order to discern patterns and trends. It is a method for removing bias from evaluating data by employing numerical analysis. This technique is useful for collecting the interpretations of research, developing statistical models, and planning surveys and studies.

Statistical analysis is a scientific tool that helps collect and analyze large amounts of data to identify common patterns and trends to convert them into meaningful information. In simple words, statistical analysis is a data analysis tool that helps draw meaningful conclusions from raw and unstructured data.

The conclusions are drawn using statistical analysis facilitating decision-making and helping businesses make future predictions on the basis of past trends. It can be defined as the science of collecting and analyzing data to identify trends and patterns and presenting them.

Statistical analysis involves working with numbers and is used by businesses and other institutions to make use of data to derive meaningful information.

```
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
movies = pd.read_csv('imdb_1000.csv')
movies.head()
```

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....

Check the number of rows and columns.

```
movies.shape
```

Output:

```
(979, 6)
```

Check the type of each column in the dataset.

```
movies.dtypes
```

Output:

```
star_rating       float64
title              object
content_rating     object
genre              object
duration            int64
actors_list        object
dtype: object
```

Calculate the average movie duration.

```
movies['duration'].mean()
```

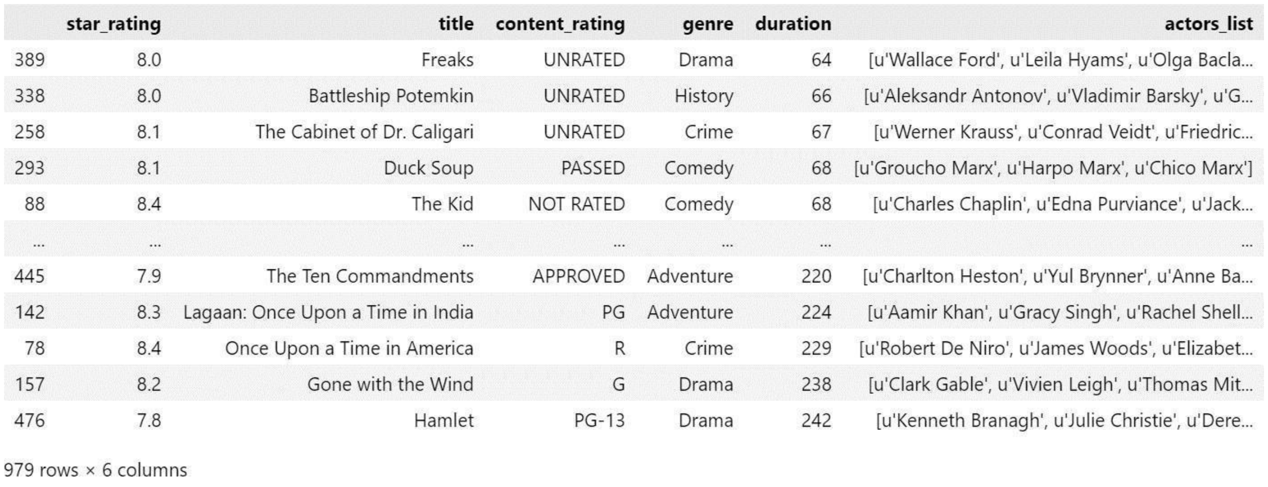
Output:

```
120.97957099080695
```

Sort the DataFrame by duration to find the shortest and longest movies.

```
movies.sort_values('duration')
```

Output:



	star_rating	title	content_rating	genre	duration	actors_list
389	8.0	Freaks	UNRATED	Drama	64	[u'Wallace Ford', u'Leila Hyams', u'Olga Bacla...
338	8.0	Battleship Potemkin	UNRATED	History	66	[u'Aleksandr Antonov', u'Vladimir Barsky', u'G...
258	8.1	The Cabinet of Dr. Caligari	UNRATED	Crime	67	[u'Werner Krauss', u'Conrad Veidt', u'Friedric...
293	8.1	Duck Soup	PASSED	Comedy	68	[u'Groucho Marx', u'Harpo Marx', u'Chico Marx']
88	8.4	The Kid	NOT RATED	Comedy	68	[u'Charles Chaplin', u'Edna Purviance', u'Jack...
...	...	...	...	...	...	...
445	7.9	The Ten Commandments	APPROVED	Adventure	220	[u'Charlton Heston', u'Yul Brynner', u'Anne Ba...
142	8.3	Lagaan: Once Upon a Time in India	PG	Adventure	224	[u'Aamir Khan', u'Gracy Singh', u'Rachel Shell...
78	8.4	Once Upon a Time in America	R	Crime	229	[u'Robert De Niro', u'James Woods', u'Elizabet...
157	8.2	Gone with the Wind	G	Drama	238	[u'Clark Gable', u'Vivien Leigh', u'Thomas Mit...
476	7.8	Hamlet	PG-13	Drama	242	[u'Kenneth Branagh', u'Julie Christie', u'Dere...

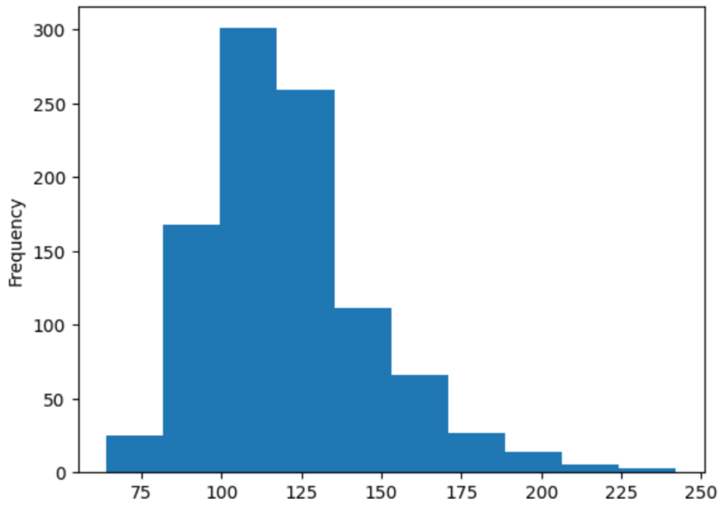
979 rows × 6 columns

Create a histogram of duration, choosing an "appropriate" number of bins.

```
movies['duration'].plot(kind='hist', bins=10)
```

Output:

```
<AxesSubplot: ylabel='Frequency'>
```

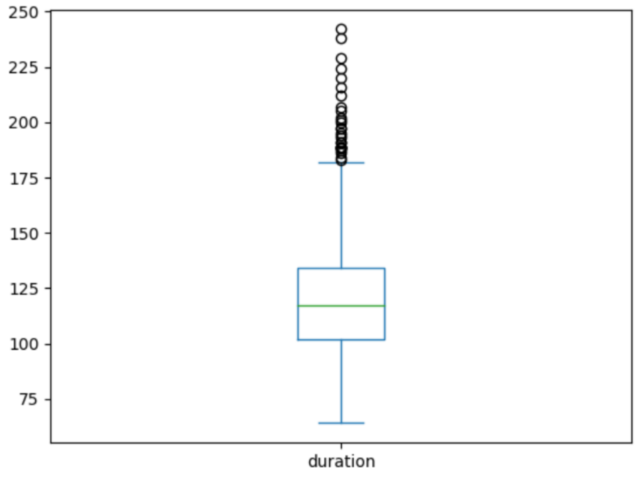


Use a box plot to display the same data.

```
movies['duration'].plot(kind='box')
```

Output:

```
<AxesSubplot: >
```



Count how many movies have each of the content ratings.

```
movies[['content_rating','title']].groupby('content_rating').count()
```

Output:

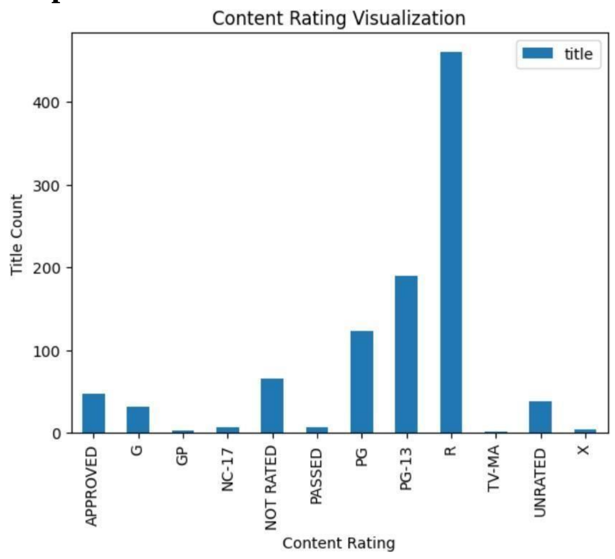
	title
content_rating	
APPROVED	47
G	32
GP	3
NC-17	7
NOT RATED	65
PASSED	7
PG	123
PG-13	189
R	460
TV-MA	1
UNRATED	38
X	4

Use a visualization to display that same data, including a title and x and y labels.

```
# Answer:
movies[['content_rating','title']].groupby('content_rating').count().plot(kind='bar',
title='Content Rating Visualization')
plt.xlabel('Content Rating')
plt.ylabel('Title Count')
```

```
Text(0, 0.5, 'Title Count')
```

Output:



Convert the following content ratings to “NC-17”:X, TV-MA.

```
movies['content_rating'].replace(['X','TV-MA'],'NC-17')
```

Output:

```
0          R
1          R
2          R
3      PG-13
4          R
       ...
974       PG
975       PG
976    PG-13
977       PG
978        R
Name: content_rating, Length: 979, dtype: object
```

Count the number of missing values in each column.

```
movies.isnull().sum(axis=0)
```

Output:

```
star_rating       0
title             0
content_rating    3
genre             0
duration          0
actors_list       0
dtype: int64
```

If there are missing values: examine them, then fill them in with “reasonable” values.

```
movies.at[187, 'content_rating'] = 'PG'
movies.at[649, 'content_rating'] = 'PG'
movies.at[936, 'content_rating'] = 'PG-13'
```

Calculate the average start rating for movies 2 hours or longer, and compare that with the average start for movies shorter than 2 hours

```
print('Avg. star rating for movies 2 hours or longer: ', movies[movies['duration'] >=
120]['star_rating'].mean(),
      '\nAvg. star rating for movies shorter than 2 hours: ',
movies[movies['duration'] < 120]['star_rating'].mean())
```

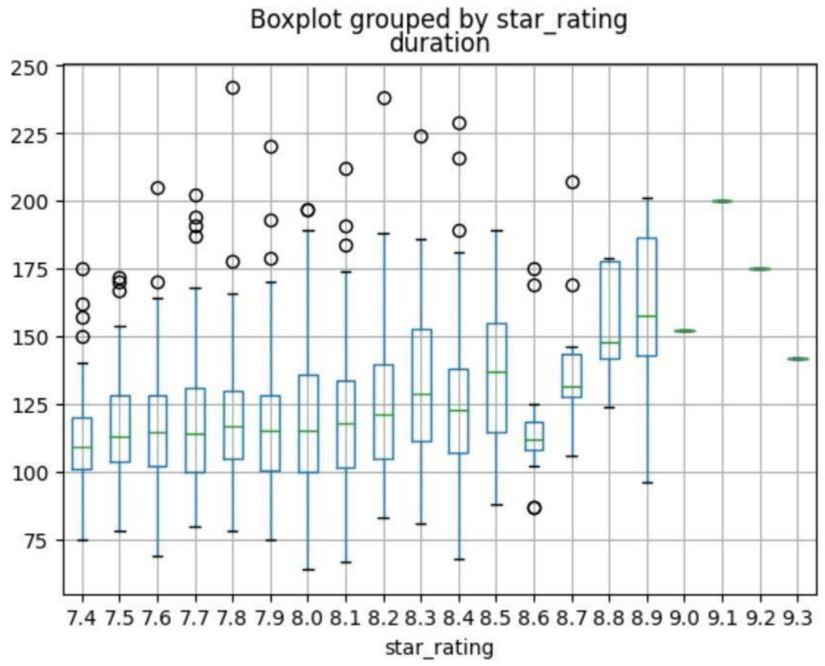
Output:

```
Avg. star rating for movies 2 hours or longer:  7.948898678414097
Avg. star rating for movies shorter than 2 hours:  7.838666666666665
```

Use a visualization to detect whether there is a relationship between duration and star rating.

```
movies.boxplot(column='duration', by='star_rating')
```

Output:



Calculate the average duration for each genre.

```
movies[['duration','genre']].groupby('genre').mean()
```

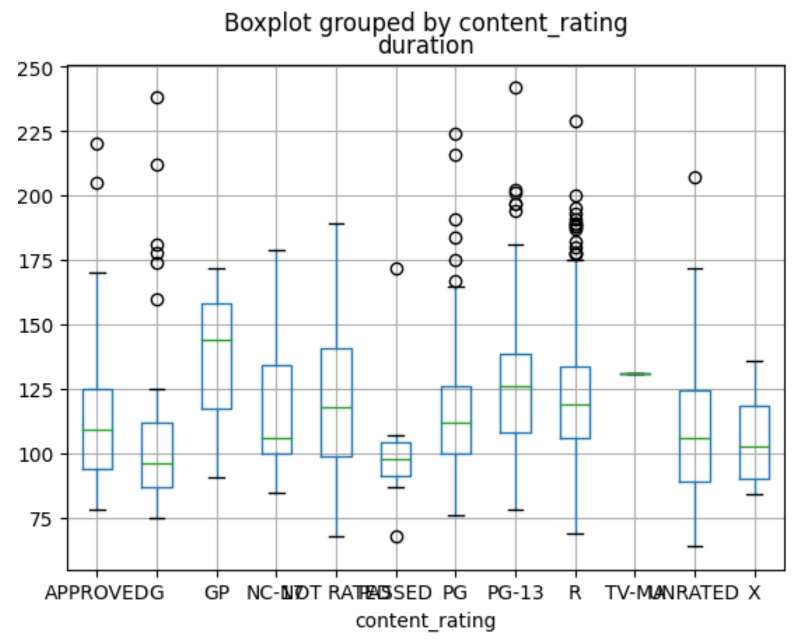
Output:

```
              duration
genre
Action      126.485294
Adventure   134.840000
Animation    96.596774
Biography   131.844156
Comedy      107.602564
Crime       122.298387
Drama       126.539568
Family      107.500000
Fantasy     112.000000
Film-Noir    97.333333
History      66.000000
Horror      102.517241
Mystery     115.625000
Sci-Fi      109.000000
Thriller    114.200000
Western     136.666667
```

Visualize the relationship between content rating and duration.

```
movies.boxplot(column='duration', by='content_rating')
```

Output:



Determine the top rated movie for each genre

```
result = movies.sort_values('star_rating', ascending=False).groupby('genre')[['title',
'star_rating']].first()
print(result)
```

Output:

```
                                                 title  star_rating
genre
Action                                 The Dark Knight          9.0
Adventure  The Lord of the Rings: The Return of the King          8.9
Animation                                Spirited Away          8.6
Biography                             Schindler's List          8.9
Comedy                                   Modern Times          8.6
Crime                        The Shawshank Redemption          9.3
Drama                                    12 Angry Men          8.9
Family                        E.T. the Extra-Terrestrial          7.9
Fantasy                         The City of Lost Children          7.7
Film-Noir                                The Third Man          8.3
History                             Battleship Potemkin          8.0
Horror                                         Psycho          8.6
Mystery                                   Rear Window          8.6
Sci-Fi                                   Blade Runner          8.2
Thriller                             Shadow of a Doubt          8.0
Western                      The Good, the Bad and the Ugly          8.9
```

Check if there are multiple movies with the same title, and if so, determine if they are duplicates.

```
result = movies[movies['title'].isin(movies[movies.duplicated(['title'])]['title'])]
result.sort_values('title')
```

Output:

	star_rating	title	content_rating	genre	duration	actors_list
703	7.6	Dracula	APPROVED	Horror	85	[u'Bela Lugosi', u'Helen Chandler', u'David Ma...
905	7.5	Dracula	R	Horror	128	[u'Gary Oldman', u'Winona Ryder', u'Anthony Ho...
678	7.7	Les Miserables	PG-13	Drama	158	[u'Hugh Jackman', u'Russell Crowe', u'Anne Hat...
924	7.5	Les Miserables	PG-13	Crime	134	[u'Liam Neeson', u'Geoffrey Rush', u'Uma Thurm...
466	7.9	The Girl with the Dragon Tattoo	R	Crime	158	[u'Daniel Craig', u'Rooney Mara', u'Christophe...
482	7.8	The Girl with the Dragon Tattoo	R	Crime	152	[u'Michael Nyqvist', u'Noomi Rapace', u'Ewa Fr...
662	7.7	True Grit	PG-13	Adventure	110	[u'Jeff Bridges', u'Matt Damon', u'Hailee Stei...
936	7.4	True Grit	PG-13	Adventure	128	[u'John Wayne', u'Kim Darby', u'Glen Campbell']

Calculate the average star rating for each genre, but only include genres with at least. 10 movies

```
genres = movies['genre'].value_counts()[movies['genre'].value_counts() > 10].index
movies[movies['genre'].isin(genres)].groupby('genre')['star_rating'].mean()
```

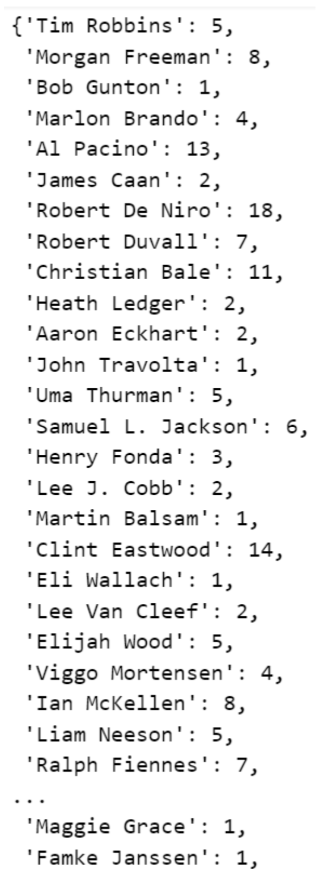
Output:

```
genre
Action       7.884559
Adventure    7.933333
Animation    7.914516
Biography    7.862338
Comedy       7.822436
Crime        7.916935
Drama        7.902518
Horror       7.806897
Mystery      7.975000
Name: star_rating, dtype: float64
```

Figure out something “interesting” using the actors data!

```
def repp(string):
    return
string.replace("[","").replace("]","").replace("u'","").replace("',",",")[:-1]
movies_series = movies['actors_list'].apply(repp)
actors_list = []
for movie_actors in movies_series:
    actors_list.append([e.strip() for e in movie_actors.split(',')])
actor_dict = {}
for actor in actors_list:
    for a in actor:
        if a in actor_dict:
            actor_dict[a] +=1
        else:
            actor_dict[a] = 1
actor_dict
```

Output:



```
{'Tim Robbins': 5,
 'Morgan Freeman': 8,
 'Bob Gunton': 1,
 'Marlon Brando': 4,
 'Al Pacino': 13,
 'James Caan': 2,
 'Robert De Niro': 18,
 'Robert Duvall': 7,
 'Christian Bale': 11,
 'Heath Ledger': 2,
 'Aaron Eckhart': 2,
 'John Travolta': 1,
 'Uma Thurman': 5,
 'Samuel L. Jackson': 6,
 'Henry Fonda': 3,
 'Lee J. Cobb': 2,
 'Martin Balsam': 1,
 'Clint Eastwood': 14,
 'Eli Wallach': 1,
 'Lee Van Cleef': 2,
 'Elijah Wood': 5,
 'Viggo Mortensen': 4,
 'Ian McKellen': 8,
 'Liam Neeson': 5,
 'Ralph Fiennes': 7,
...
 'Maggie Grace': 1,
 'Famke Janssen': 1,
```

14. Sentiment Analysis on Twitter dataset

Sentiment Analysis on Twitter datasets is a fundamental task in natural language processing, dedicated to discerning the prevailing emotional tone within tweets shared on the Twitter platform. The primary objective is to determine whether the expressed sentiments in tweets are positive, negative, or neutral. The process commences with data collection, involving the acquisition of a substantial volume of tweets either through Twitter's API or pre-existing datasets that may be curated based on specific keywords, hashtags, or user accounts of interest. Subsequently, data preprocessing becomes imperative, encompassing activities like the removal of noise, including special characters, URLs, and irrelevant information, rendering the text amenable to analysis. Tokenization then segments the tweets into smaller units, typically words or subword tokens, to enable more granular analysis. Sentiment labeling is a pivotal step, as it assigns sentiment labels to tweets, thereby categorizing them based on their emotional context. Ultimately, this process facilitates the extraction of insights and trends from Twitter data, providing valuable sentiment-related information for various applications such as brand monitoring, market analysis, and public opinion tracking.

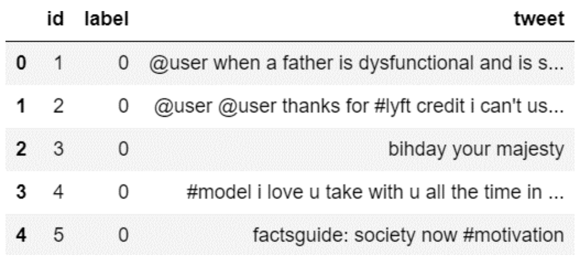
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
train = pd.read_csv('train_tweet.csv')
test = pd.read_csv('test_tweets.csv')
print(train.shape)
print(test.shape)
```

Output:

```
(31962, 3)
(17197, 2)
```

```
train.head()
```

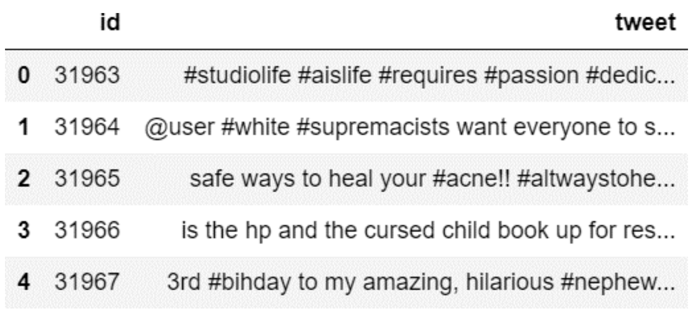
Output:



	id	label	tweet
0	1	0	@user when a father is dysfunctional and is s...
1	2	0	@user @user thanks for #lyft credit i can't us...
2	3	0	bihday your majesty
3	4	0	#model i love u take with u all the time in ...
4	5	0	factsguide: society now #motivation

```
test.head()
```

Output:



	id	tweet
0	31963	#studiolife #aislife #requires #passion #dedic...
1	31964	@user #white #supremacists want everyone to s...
2	31965	safe ways to heal your #acne!! #altwaystohe...
3	31966	is the hp and the cursed child book up for res...
4	31967	3rd #bihday to my amazing, hilarious #nephew...

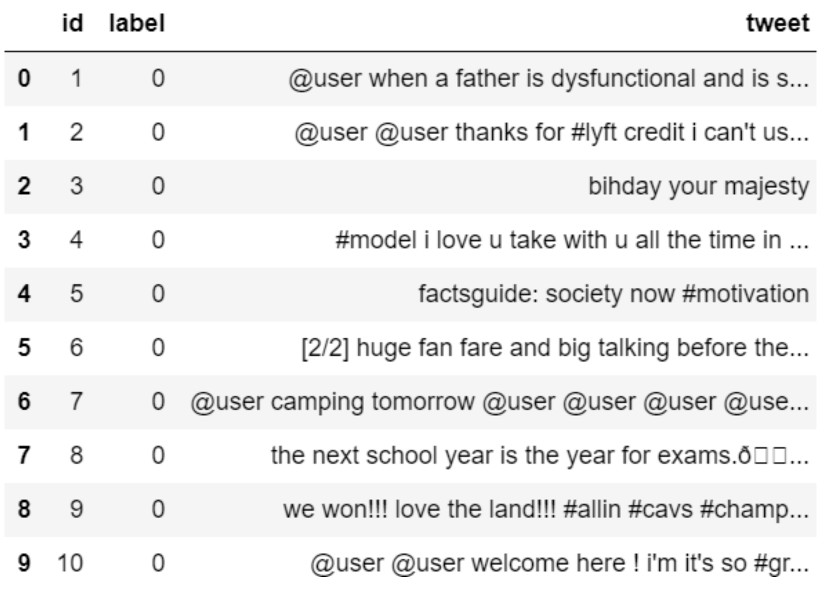
```
train.isnull().any()
test.isnull().any()
```

Output:

```
id       False
tweet    False
dtype: bool
```

```
# checking out the negative comments from the train set
train[train['label'] == 0].head(10)
```

Output:



	id	label	tweet
0	1	0	@user when a father is dysfunctional and is s...
1	2	0	@user @user thanks for #lyft credit i can't us...
2	3	0	bihday your majesty
3	4	0	#model i love u take with u all the time in ...
4	5	0	factsguide: society now #motivation
5	6	0	[2/2] huge fan fare and big talking before the...
6	7	0	@user camping tomorrow @user @user @user @use...
7	8	0	the next school year is the year for exams.ð□□...
8	9	0	we won!!! love the land!!! #allin #cavs #champ...
9	10	0	@user @user welcome here ! i'm it's so #gr...

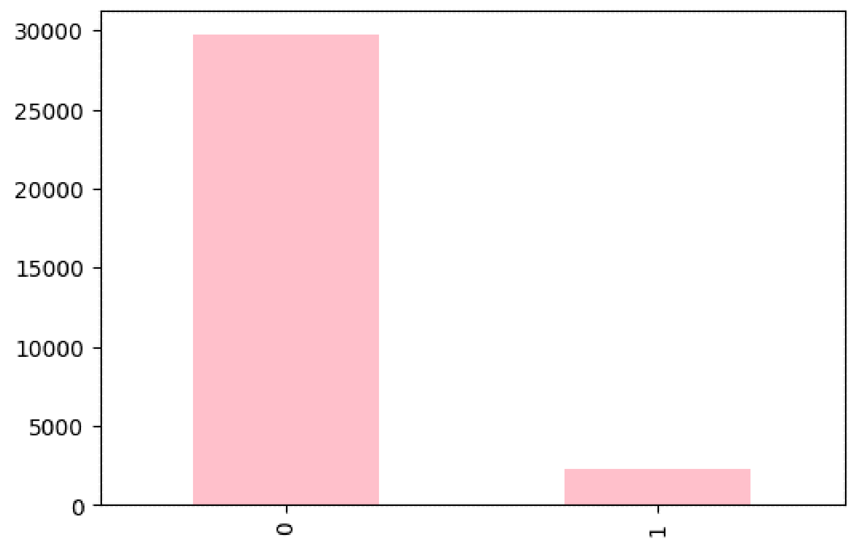
```
# checking out the postive comments from the train set
train[train['label'] == 1].head(10)
```

Output:

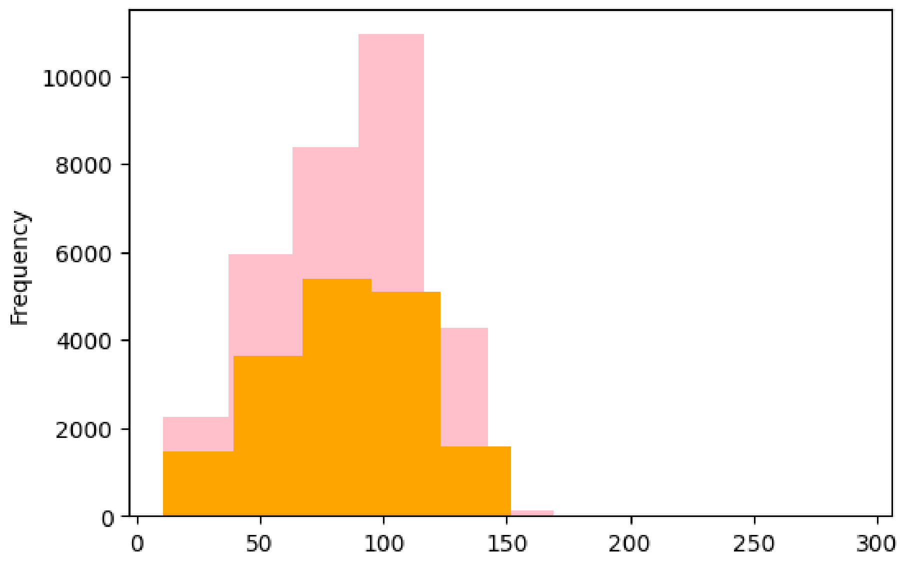
	id	label	tweet
13	14	1	@user #cnn calls #michigan middle school 'buil...
14	15	1	no comment! in #australia #opkillingbay #se...
17	18	1	retweet if you agree!
23	24	1	@user @user lumpy says i am a . prove it lumpy.
34	35	1	it's unbelievable that in the 21st century we'...
56	57	1	@user lets fight against #love #peace
68	69	1	ð□□©the white establishment can't have blk fol...
77	78	1	@user hey, white people: you can call people '...
82	83	1	how the #altright uses &amp; insecurity to lu...
111	112	1	@user i'm not interested in a #linguistics tha...

```
train['label'].value_counts().plot.bar(color = 'pink', figsize = (6, 4))
```

Output:



```python
# checking the distribution of tweets in the data

length_train = train['tweet'].str.len().plot.hist(color = 'pink', figsize = (6, 4))
length_test = test['tweet'].str.len().plot.hist(color = 'orange', figsize = (6, 4))
```



```python
# adding a column to represent the length of the tweet

train['len'] = train['tweet'].str.len()
test['len'] = test['tweet'].str.len()
train.head(10)
```

Output:

	id	label	tweet	len
0	1	0	@user when a father is dysfunctional and is s...	102
1	2	0	@user @user thanks for #lyft credit i can't us...	122
2	3	0	bihday your majesty	21
3	4	0	#model i love u take with u all the time in ...	86
4	5	0	factsguide: society now #motivation	39
5	6	0	[2/2] huge fan fare and big talking before the...	116
6	7	0	@user camping tomorrow @user @user @user @use...	74
7	8	0	the next school year is the year for exams.ð□□...	143
8	9	0	we won!!! love the land!!! #allin #cavs #champ...	87
9	10	0	@user @user welcome here ! i'm it's so #gr...	50

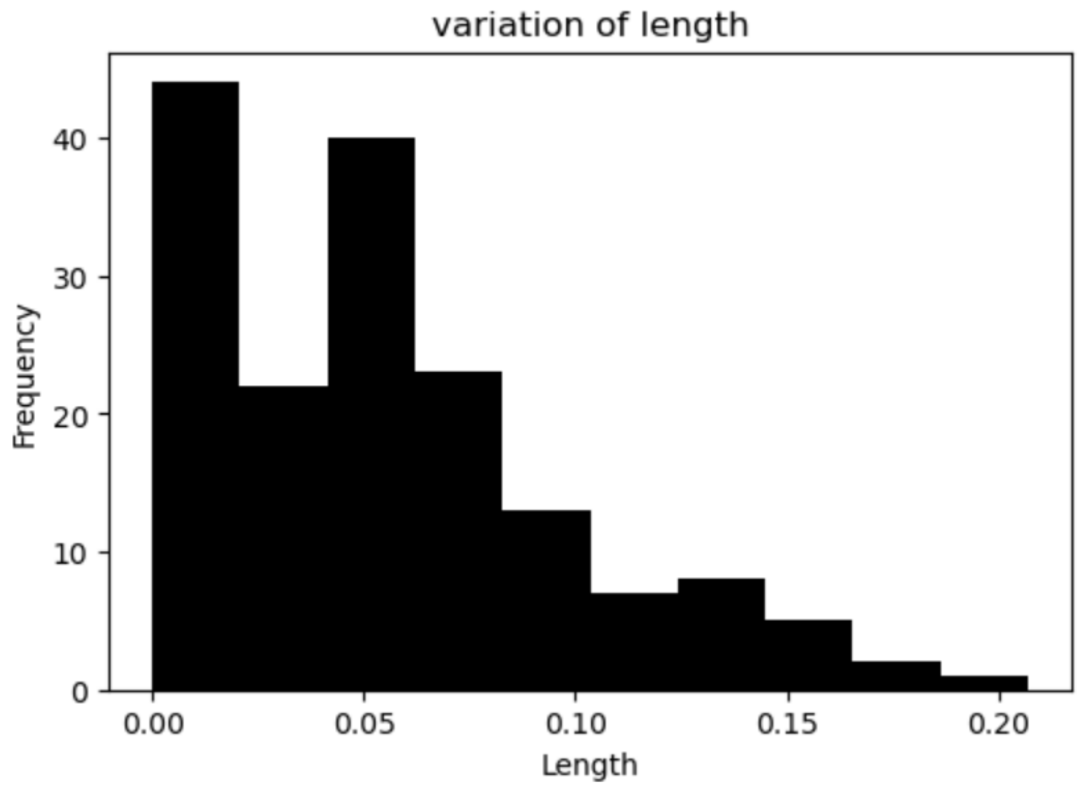
```
train.groupby('label').describe()
```

Output:

	id								len							
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max
label																
0	29720.0	15974.454441	9223.783469	1.0	7981.75	15971.5	23965.25	31962.0	29720.0	84.328634	29.566484	11.0	62.0	88.0	107.0	274.0
1	2242.0	16074.896075	9267.955758	14.0	8075.25	16095.0	24022.00	31961.0	2242.0	90.187779	27.375502	12.0	69.0	96.0	111.0	152.0

```
train.groupby('len').mean()['label'].plot.hist(color = 'black', figsize = (6, 4),)
plt.title('variation of length')
plt.xlabel('Length')
plt.show()
```

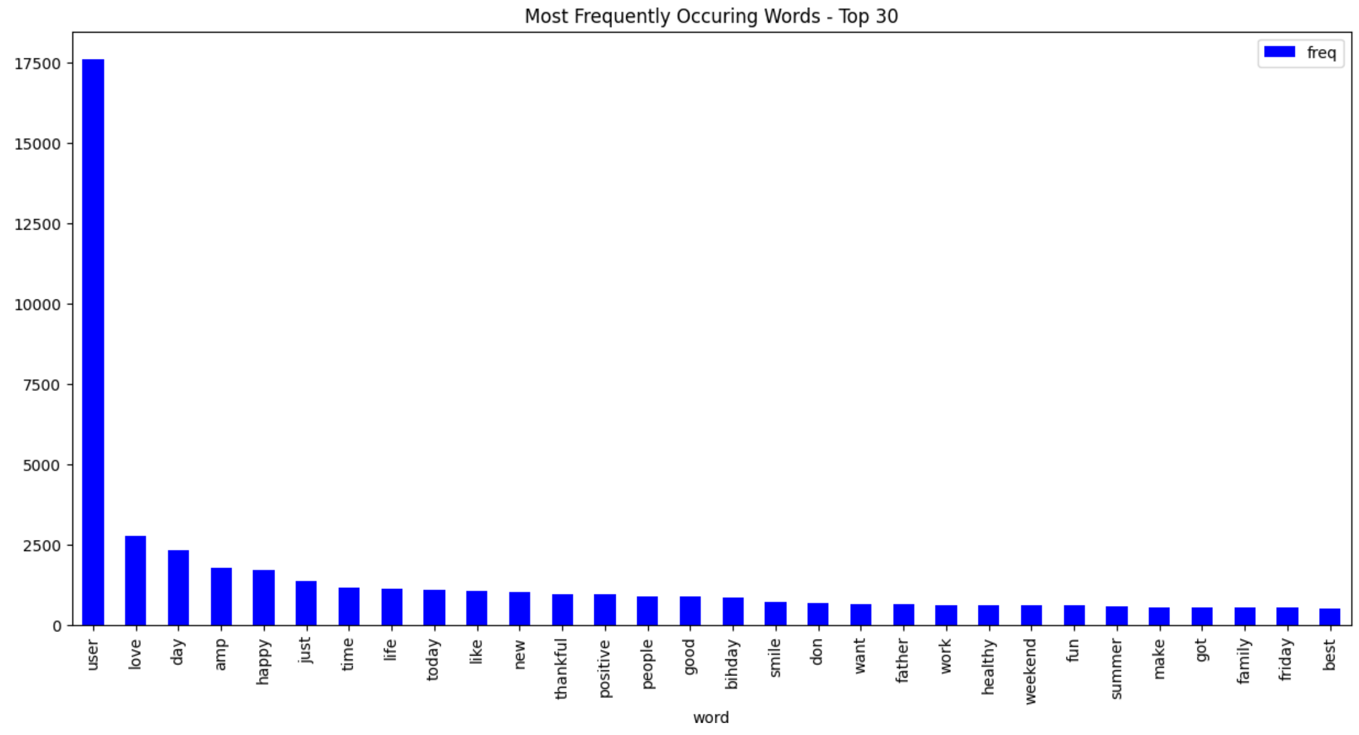
Output:



```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(stop_words = 'english')
words = cv.fit_transform(train.tweet)

sum_words = words.sum(axis=0)
```

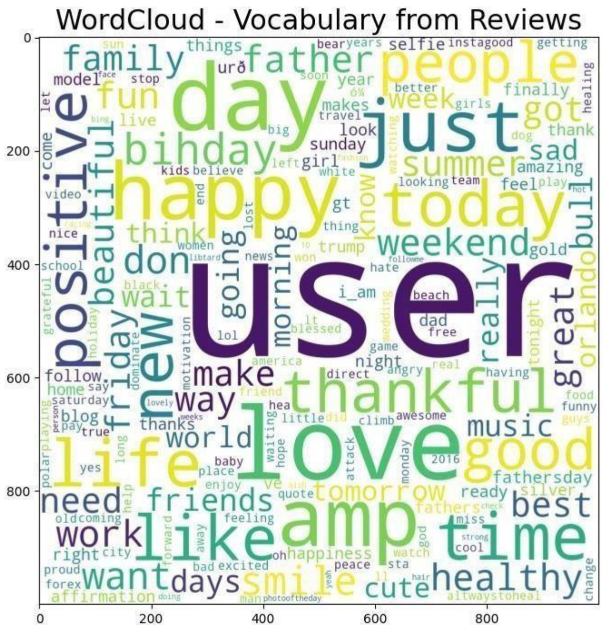
```
words_freq = [(word, sum_words[0, i]) for word, i in cv.vocabulary_.items()]
words_freq = sorted(words_freq, key = lambda x: x[1], reverse = True)

frequency = pd.DataFrame(words_freq, columns=['word', 'freq'])

frequency.head(30).plot(x='word', y='freq', kind='bar', figsize=(15, 7), color =
'blue')
plt.title("Most Frequently Occuring Words - Top 30")
```

Output:



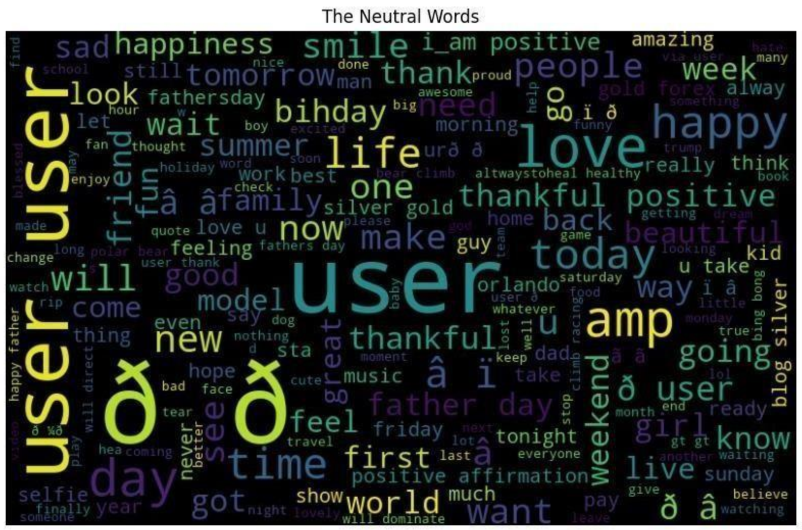
```
from wordcloud import WordCloud

wordcloud = WordCloud(background_color = 'white', width = 1000, height =
1000).generate_from_frequencies(dict(words_freq))

plt.figure(figsize=(10,8))
plt.imshow(wordcloud)
plt.title("WordCloud - Vocabulary from Reviews", fontsize = 22)
```

Output:



```
normal_words =' '.join([text for text in train['tweet'][train['label'] == 0]])
wordcloud = WordCloud(width=800, height=500, random_state = 0, max_font_size =
110).generate(normal_words)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.title('The Neutral Words')
plt.show()
```

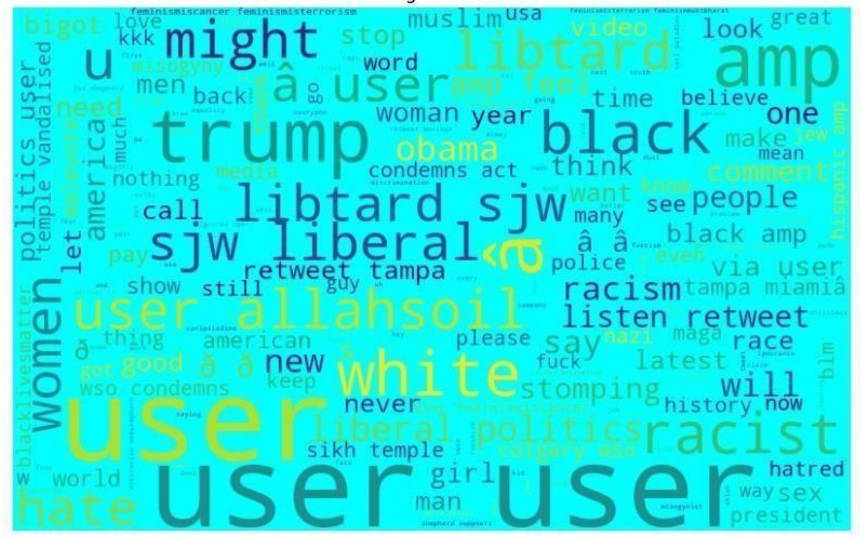
Output:



```
negative_words =' '.join([text for text in train['tweet'][train['label'] == 1]])
wordcloud = WordCloud(background_color = 'cyan', width=800, height=500, random_state =
0, max_font_size = 110).generate(negative_words)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.title('The Negative Words')
plt.show()
```

Output:

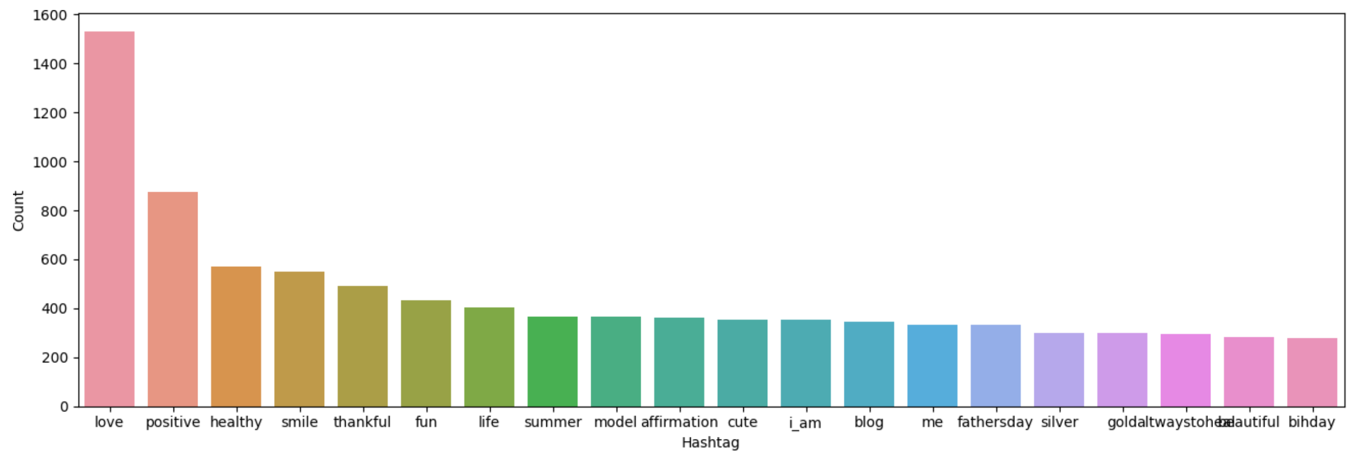
The Negative Words



```
# collecting the hashtags
import re
def hashtag_extract(x):
    hashtags = []

    for i in x:
        ht = re.findall(r"#(\w+)", i)
        hashtags.append(ht)

    return hashtags


import nltk
# extracting hashtags from non racist/sexist tweets
HT_regular = hashtag_extract(train['tweet'][train['label'] == 0])

# extracting hashtags from racist/sexist tweets
HT_negative = hashtag_extract(train['tweet'][train['label'] == 1])
```

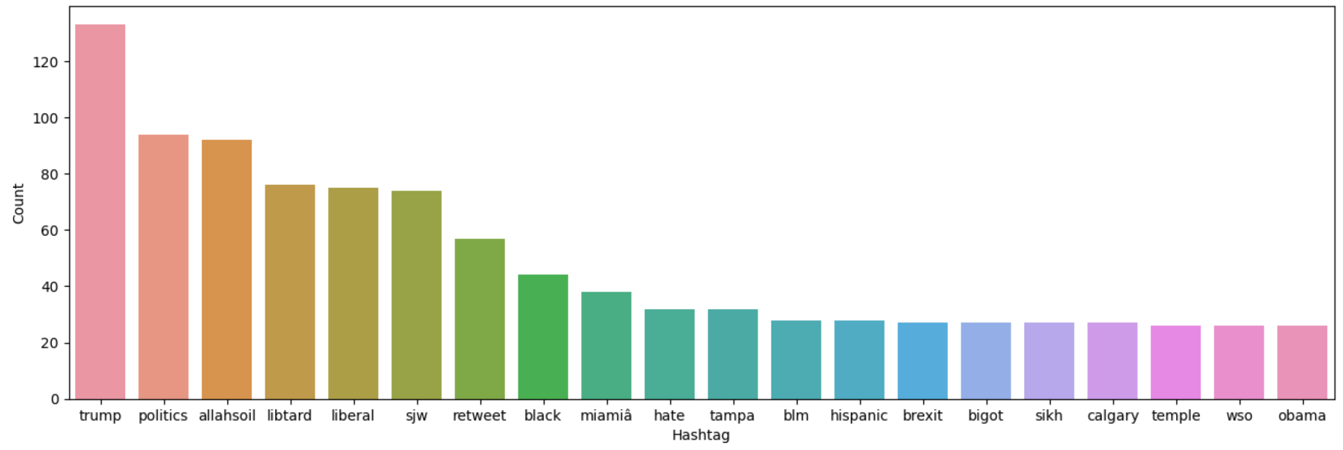
```
# unnesting list
HT_regular = sum(HT_regular,[])
HT_negative = sum(HT_negative,[])

a = nltk.FreqDist(HT_regular)
d = pd.DataFrame({'Hashtag': list(a.keys()),
                  'Count': list(a.values())})


# selecting top 20 most frequent hashtags
d = d.nlargest(columns="Count", n = 20)
plt.figure(figsize=(16,5))
ax = sns.barplot(data=d, x= "Hashtag", y = "Count")
ax.set(ylabel = 'Count')
plt.show()
```

Output:



```
a = nltk.FreqDist(HT_negative)
d = pd.DataFrame({'Hashtag': list(a.keys()),
                  'Count': list(a.values())})

# selecting top 20 most frequent hashtags
d = d.nlargest(columns="Count", n = 20)
plt.figure(figsize=(16,5))
ax = sns.barplot(data=d, x= "Hashtag", y = "Count")
ax.set(ylabel = 'Count')
plt.show()
```

Output:



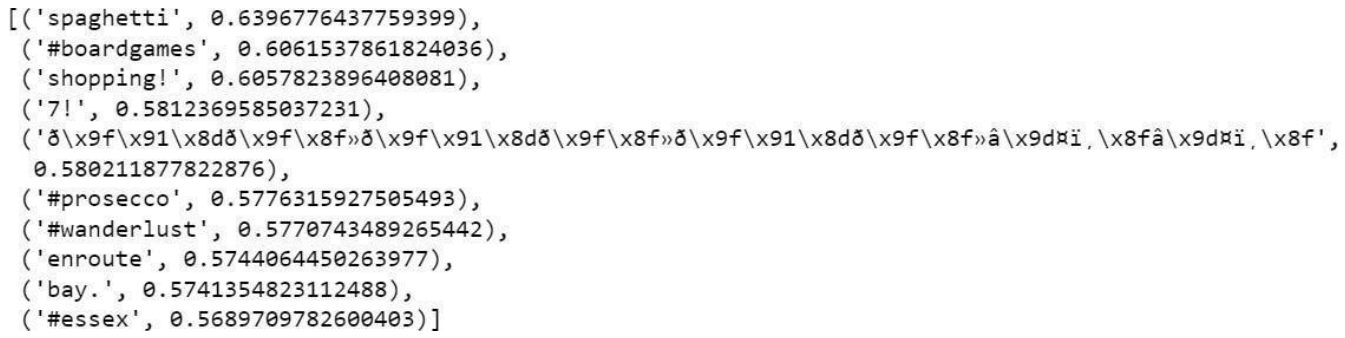
```
# tokenizing the words present in the training set
tokenized_tweet = train['tweet'].apply(lambda x: x.split())

# importing gensim
import gensim

# creating a word to vector model
model_w2v = gensim.models.Word2Vec(
            tokenized_tweet,
            vector_size=200, # desired no. of features/independent variables
            window=5, # context window size
            min_count=2,
            sg = 1, # 1 for skip-gram model
            hs = 0,
            negative = 10, # for negative sampling
            workers= 2, # no.of cores
            seed = 34)

model_w2v.train(tokenized_tweet, total_examples= len(train['tweet']), epochs=20)
```

Output:

```
(6110168, 8411580)
```

```
model_w2v.wv.most_similar(positive = "dinner")
```

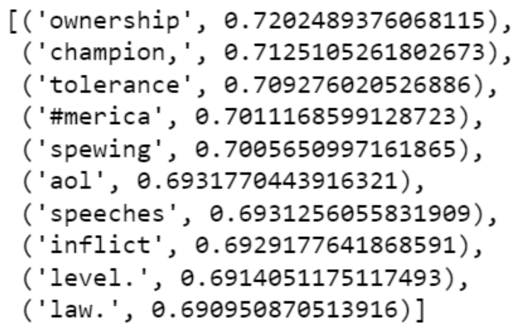
Output:



```
[('spaghetti', 0.6396776437759399),
 ('#boardgames', 0.6061537861824036),
 ('shopping!', 0.6057823896408081),
 ('7!', 0.5812369585037231),
 ('ð\x9f\x91\x8dð\x9f\x8f»ð\x9f\x91\x8dð\x9f\x8f»ð\x9f\x91\x8dð\x9f\x8f»â\x9d¤ï¸\x8fâ\x9d¤ï¸\x8f',
  0.580211877822876),
 ('#prosecco', 0.5776315927505493),
 ('#wanderlust', 0.5770743489265442),
 ('enroute', 0.5744064450263977),
 ('bay.', 0.5741354823112488),
 ('#essex', 0.5689709782600403)]
```

```
model_w2v.wv.most_similar(positive = "cancer")
```

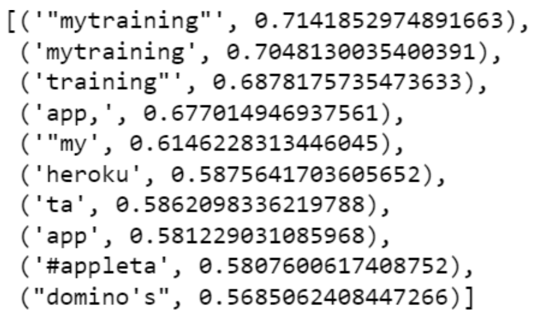
Output:



```
[('ownership', 0.7202489376068115),
 ('champion,', 0.7125105261802673),
 ('tolerance', 0.709276020526886),
 ('#merica', 0.7011168599128723),
 ('spewing', 0.7005650997161865),
 ('aol', 0.6931770443916321),
 ('speeches', 0.6931256055831909),
 ('inflict', 0.6929177641868591),
 ('level.', 0.6914051175117493),
 ('law.', 0.690950870513916)]
```

```
model_w2v.wv.most_similar(positive = "apple")
```

Output:



```
[('"mytraining"', 0.7141852974891663),
 ('mytraining', 0.7048130035400391),
 ('training"', 0.6878175735473633),
 ('app,', 0.677014946937561),
 ('"my', 0.6146228313446045),
 ('heroku', 0.5875641703605652),
 ('ta', 0.5862098336219788),
 ('app', 0.581229031085968),
 ('#appleta', 0.5807600617408752),
 ("domino's", 0.5685062408447266)]
```

```
model_w2v.wv.most_similar(negative = "hate")
```

Output:

```
[('#community', 0.05086711421608925),
 ('â\x9c\x88ï¸\x8f', 0.04701182618737221),
 ('ð\x9f\x8d»', 0.037239667028188705),
 ('lion', 0.028969956561923027),
 ('#foodie', 0.0253813024610281),
 ('#tgif', 0.014376196078956127),
 ('#newyork', 0.013427932746708393),
 ('#crafts', 0.011316018179059029),
 ('#relax', 0.010859509993001461),
 ('#hungry', 0.009565738029778004)]
```

```
from tqdm import tqdm
tqdm.pandas(desc="progress-bar")
from gensim.models.doc2vec import TaggedDocument

def add_label(twt):
    output = []
    for i, s in zip(twt.index, twt):
        output.append(TaggedDocument(s, ["tweet_" + str(i)]))
    return output

# label all the tweets
labeled_tweets = add_label(tokenized_tweet)
labeled_tweets[:6]
```

Output:

```
[TaggedDocument(words=['@user', 'when', 'a', 'father', 'is', 'dysfunctional', 'and', 'is', 'so', 'selfish', 'he', 'drags', 'hi
s', 'kids', 'into', 'his', 'dysfunction.', '#run'], tags=['tweet_0']),
 TaggedDocument(words=['@user', '@user', 'thanks', 'for', '#lyft', 'credit', 'i', "can't", 'use', 'cause', 'they', "don't", 'of
fer', 'wheelchair', 'vans', 'in', 'pdx.', '#disapointed', '#getthanked'], tags=['tweet_1']),
 TaggedDocument(words=['bihday', 'your', 'majesty'], tags=['tweet_2']),
 TaggedDocument(words=['#model', 'i', 'love', 'u', 'take', 'with', 'u', 'all', 'the', 'time', 'in', 'urð\x9f\x93±!!!', 'ð\x9f\x
98\x99ð\x9f\x98\x8eð\x9f\x91\x84ð\x9f\x91', 'ð\x9f\x92¦ð\x9f\x92¦ð\x9f\x92¦'], tags=['tweet_3']),
 TaggedDocument(words=['factsguide:', 'society', 'now', '#motivation'], tags=['tweet_4']),
 TaggedDocument(words=['[2/2]', 'huge', 'fan', 'fare', 'and', 'big', 'talking', 'before', 'they', 'leave.', 'chaos', 'and', 'pa
y', 'disputes', 'when', 'they', 'get', 'there.', '#allshowandnogo'], tags=['tweet_5'])]
```

```
# removing unwanted patterns from the data
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
```

Output:

```
[nltk_data] Error loading stopwords: <urlopen error [Errno 11001]
[nltk_data]     getaddrinfo failed>
```

```
train_corpus = []
for i in range(0, 31962):
  review = re.sub('[^a-zA-Z]', ' ', train['tweet'][i])
  review = review.lower()
  review = review.split()
  ps = PorterStemmer()
  # stemming
  review = [ps.stem(word) for word in review if not word in
set(stopwords.words('english'))]

  # joining them back with space
  review = ' '.join(review)
  train_corpus.append(review)

test_corpus = []
for i in range(0, 17197):
  review = re.sub('[^a-zA-Z]', ' ', test['tweet'][i])
  review = review.lower()
  review = review.split()
  ps = PorterStemmer()

  # stemming
  review = [ps.stem(word) for word in review if not word in
set(stopwords.words('english'))]

  # joining them back with space
  review = ' '.join(review)
  test_corpus.append(review)

# creating bag of words
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(max_features = 2500)
x = cv.fit_transform(train_corpus).toarray()
y = train.iloc[:, 1]

print(x.shape)
print(y.shape)
```

Output:

```
(31962, 2500)
(31962,)
```

```
# creating bag of words

from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features = 2500)
x_test = cv.fit_transform(test_corpus).toarray()
print(x_test.shape)
```

Output:

```
(17197, 2500)
```

```
# splitting the training data into train and valid sets

from sklearn.model_selection import train_test_split
x_train, x_valid, y_train, y_valid = train_test_split(x, y, test_size = 0.25,
random_state = 42)
print(x_train.shape)
print(x_valid.shape)
print(y_train.shape)
print(y_valid.shape)
```

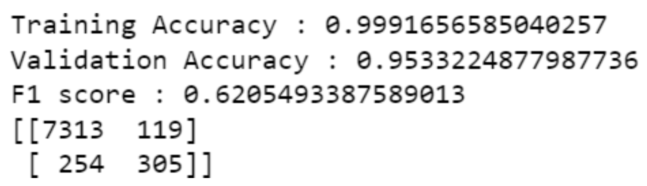
Output:

```
(23971, 2500)
(7991, 2500)
(23971,)
(7991,)
```

```
# standardization
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_valid = sc.transform(x_valid)
x_test = sc.transform(x_test)

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score

model = RandomForestClassifier()
model.fit(x_train, y_train)

y_pred = model.predict(x_valid)
```

```
print("Training Accuracy :", model.score(x_train, y_train))
print("Validation Accuracy :", model.score(x_valid, y_valid))

# calculating the f1 score for the validation set
print("F1 score :", f1_score(y_valid, y_pred))

# confusion matrix
cm = confusion_matrix(y_valid, y_pred)
print(cm)
```

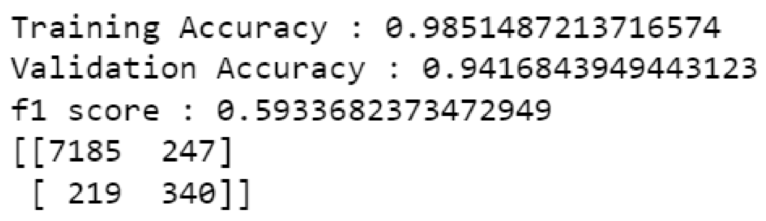
Output:



```
Training Accuracy : 0.9991656585040257
Validation Accuracy : 0.9533224877987736
F1 score : 0.6205493387589013
[[7313  119]
 [ 254  305]]
```

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_valid)

print("Training Accuracy :", model.score(x_train, y_train))
print("Validation Accuracy :", model.score(x_valid, y_valid))

# calculating the f1 score for the validation set
print("f1 score :", f1_score(y_valid, y_pred))
# confusion matrix
cm = confusion_matrix(y_valid, y_pred)
print(cm)
```

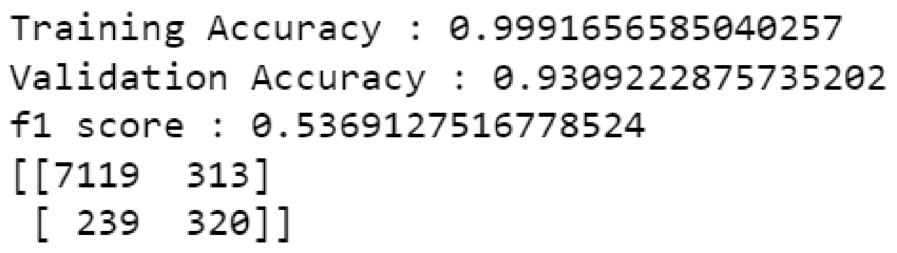
Output:



```
Training Accuracy : 0.9851487213716574
Validation Accuracy : 0.9416843949443123
f1 score : 0.5933682373472949
[[7185  247]
 [ 219  340]]
```

```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()
model.fit(x_train, y_train)

y_pred = model.predict(x_valid)

print("Training Accuracy :", model.score(x_train, y_train))
print("Validation Accuracy :", model.score(x_valid, y_valid))

# calculating the f1 score for the validation set
print("f1 score :", f1_score(y_valid, y_pred))

# confusion matrix
cm = confusion_matrix(y_valid, y_pred)
print(cm)
```

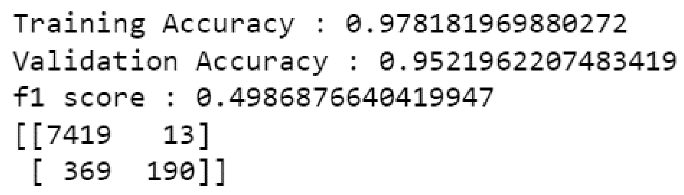
Output:



```
Training Accuracy : 0.9991656585040257
Validation Accuracy : 0.9309222875735202
f1 score : 0.5369127516778524
[[7119  313]
 [ 239  320]]
```

```
from sklearn.svm import SVC

model = SVC()
model.fit(x_train, y_train)

y_pred = model.predict(x_valid)

print("Training Accuracy :", model.score(x_train, y_train))
print("Validation Accuracy :", model.score(x_valid, y_valid))

# calculating the f1 score for the validation set
print("f1 score :", f1_score(y_valid, y_pred))

# confusion matrix
cm = confusion_matrix(y_valid, y_pred)
print(cm)
```

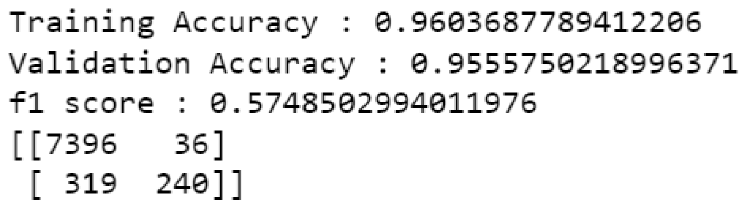
Output:



```
Training Accuracy : 0.978181969880272
Validation Accuracy : 0.9521962207483419
f1 score : 0.4986876640419947
[[7419   13]
 [ 369  190]]
```

```
from xgboost import XGBClassifier

model = XGBClassifier()
model.fit(x_train, y_train)

y_pred = model.predict(x_valid)

print("Training Accuracy :", model.score(x_train, y_train))
print("Validation Accuracy :", model.score(x_valid, y_valid))

# calculating the f1 score for the validation set
print("f1 score :", f1_score(y_valid, y_pred))

# confusion matrix
cm = confusion_matrix(y_valid, y_pred)
print(cm)
```

Output:



```
Training Accuracy : 0.9603687789412206
Validation Accuracy : 0.9555750218996371
f1 score : 0.5748502994011976
[[7396   36]
 [ 319  240]]
```

VNRVJIET

Name of the Laboratory:________________

____

Name of the Experiment :________________________

____

Experiment No:________**Date:**____________________