1. NLP - Introduction

Contents

1. NLP – Introduction

1. Natural Language

- ✓ We are all used communicate with natural language.
  - We may speak to each other on daily base for different purpose.
  - It is very easy to speak than to write.
  - We are surrounded by text.

- ✓ Think about how much text you see each day:
  - Signs/signals
  - Menus
  - Email
  - SMS
  - Web Pages and much more…
  - The list is endless.

2. Now think about speech.

- ✓ We may speak to each other on daily base for different purpose.
- ✓ It is very easy to speak than to write.

3. Text data

- ✓ We need understand more about text related data.
- ✓ Text data also having separate methodologies to analyse the data.

4. Natural Language Processing

- ✓ The short name for Natural Language Processing is NLP.
- ✓ NLP is a way of computers to analyse, understand and derive meaning from human languages such as English, Spanish, Hindi, etc.
- ✓ With NLP, machine learning algorithms can be applied to speech and text.

5. NLP popularity is raising everyday

- ✓ Yes, the popularity of NLP is rising every day.
- ✓ With NLP, a computer can understand the human language while speaking
- ✓ By using big data we can make communication in between human and machine.
- ✓ With NLP an intelligent system such as a robot can perform according to our instructions issued in a plain language such as English.

6. IMP components in NLP

- ✓ There are mainly two components in NLP
  - Syntax
  - Semantic analysis

Syntax

- ✓ Syntax explains about the arrangement of the words in sentence and grammar as well
- ✓ NLP makes use of syntax to analyse to get the meaning from a language depending on grammatical rules.
- ✓ Some syntax techniques that are used for this include,
  - Parsing,
  - Word segmentation,
  - Sentence breaking,
  - Morphological segmentation and stemming.

7. Semantics

- ✓ Semantic is associated with the use and the meaning of various words.
- ✓ In NLP, algorithms are used to analyse and determine the meaning and structure of sentences.
- ✓ Some NLP techniques used for semantics include word understanding, named entity recognition and natural language generation.

8. Deep learning importance

- ✓ Most of the current approaches to NLP are using deep learning.
- ✓ Deep learning is a type of artificial intelligence that relies on the patterns in data to improve the understanding of a program.
- ✓ Deep learning models usually require huge amounts of labelled data to train and identify any correlations between the data elements.

9. NLTK

- ✓ NLTK means Natural Language Toolkit.
- ✓ It is a python module to work with nlp
- ✓ It is open source library
- ✓ To install this library,
  - pip install nltk

2. NLP – Text wrangling and cleaning

Contents

2. NLP – Text wrangling and cleaning

1. Text wrangling and cleansing

- ✓ The ultimate goal of doing text processing is to, machine can understand the text data.
- ✓ Text processing is very important as it helps us understand our data better and gain insights from it.
- ✓ Text processing normally involves the following steps,
  - Tokenization
  - Lemmatization
  - Stemming
  - Stop word removal

Program Name: Downloading punkt
demo1.py

```
import nltk
nltk.download("punkt")
```

Output

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

- ✓ We have then downloaded the punkt, which is a Tokenizer for a text into a list of sentences.
- ✓ We can now do text wrangling.

2. Sentence Splitting

- ✓ Generally we used to understand sentence by reading words
- ✓ We will start reading sentence to understand paragraph.
- ✓ So, a paragraph we need to split it into a set of sentences.
- ✓ So, let's understand words, sentences and paragraph.
- ✓ This we can easily do by using nltk package

Program Name: Sentence splitting by using split() method
demo2.py

```
myString = "This is a paragraph. It should split at the end of sentence marker, such as a period. It can tell that the period in Mr.Daniel is not an end. Run it!, Hey How are you doing"

result = myString.split(".")
print(result)
```

Output

```
['This is a paragraph', ' It should split at the end of sentence marker, such as a period', ' It can tell that the period in Mr', 'Daniel is not an end', ' Run it!, Hey How are you doing']
```

Program Name: Sentence splitting
demo3.py

```
from nltk.tokenize import sent_tokenize

myString = "This is a paragraph. It should split at the end of sentence marker, such as a period. It can tell that the period in Mr.Daniel is not an end. Run it!, Hey How are you doing"

tokenized_sentence = sent_tokenize(myString)
print(tokenized_sentence)
```

Output

```
['This is a paragraph.', 'It should split at the end of sentence marker, such as a period.', 'It can tell that the period in Mr.Daniel is not an end.', 'Run it!, Hey How are you doing']
```

Note

- ✓ We can understand from above output, the paragraph was split into the exact sentences.
- ✓ It was also able to tell the difference between a period that has been used to end a sentence from one used on the name Mr.Daniel also

3. Tokenization

- ✓ Tokenization refers to the process by which a large text is broken down into various pieces.
- ✓ In NLP, a token is the minimal piece of text that a machine can be able to understand.
- ✓ A text may be tokenized into words or sentences.
    - In the case of sentence tokenization, every sentence will be identified as a token.
    - In the case of word tokenization, every word will be identified as a token.

4. Word tokenization

- ✓ There are various ways through which tokenization can be done, but the most popular way of doing it is through word tokenization.
- ✓ What happens in word tokenization is that a large text is broken down into words and the words are used as the tokens.
  - word_tokenizer
  - sent_tokenizer
  - punkt_tokenizer
  - Regexp_tokenizer

- ✓ To tokenize a text, we can still apply split() method from string

Program Name: Word tokenization
demo4.py

```
myString = "These are sentences. Let us tokenize it! Run it!"

print(myString.split())
```

Output

```
['These', 'are', 'sentences.', 'Let', 'us', 'tokenize', 'it!', 'Run', 'it!']
```

Program Name: Word tokenization with word_tokenize function
demo5.py

```
from nltk.tokenize import word_tokenize

myString = "These are sentences. Let us tokenize it! Run it!"

print(word_tokenize(myString))
```

Output

```
['These', 'are', 'sentences', '.', 'Let', 'us', 'tokenize', 'it', '!', 'Run', 'it', '!']
```

Program Name: Word tokenization with regexp_tokenize function
demo6.py

```
from nltk.tokenize import regexp_tokenize

myString = "These are sentences. Let us tokenize it! Run it!"

print(regexp_tokenize(myString, pattern="\w+"))
```

Output

```
['These', 'are', 'sentences', 'Let', 'us', 'tokenize', 'it', 'Run', 'it']
```

Program Name: Capture the digit from sentence
demo7.py

```
from nltk.tokenize import regexp_tokenize

myString = "These are 3 sentences. Let us tokenize them! Run the code!"

print(regexp_tokenize(myString, pattern="\d+"))
```

Output

```
['3']
```

5. Stemming

- ✓ This is another step in text wrangling.
- ✓ From the name, you can get the meaning, cutting down a token to its root stem.
- ✓ Stemming algorithm works by cutting the suffix from the word.
- ✓ In simple, it cuts either the beginning or end of the word.
- ✓ Consider the word "cutting".
- ✓ This word can be broken down to its root, which is "cut".

Program Name: Stemming
demo8.py

```
from nltk.stem import PorterStemmer

porter = PorterStemmer()
print(porter.stem("cutting"))
```

Output

```
cut
```

Program Name: Stemming
demo9.py

```
from nltk.stem import PorterStemmer

e_words= ["wait", "waiting", "waited", "waits"]

ps = PorterStemmer()

for w in e_words:
        rootWord = ps.stem(w)
        print(rootWord)
```

Output

```
wait
wait
wait
wait
```

Program Name: Stemming
demo10.py

```
import nltk
from nltk.stem.porter import PorterStemmer

porter_stemmer  = PorterStemmer()
text = "studies studying cries cry"
tokenization = nltk.word_tokenize(text)

for w in tokenization:
        print("Stemming for {} is {}".format(w, porter_stemmer.stem(w)))
```

Output

```
Stemming for studies is studi
Stemming for studying is studi
Stemming for cries is cri
Stemming for cry is cri
```

- ✓ Stemming is good for its simplicity when it comes to dealing with NLP problems.
- ✓ However, when more complex stemming is needed, stemming is not the best option, but we have lemmatization.

6. Lemmatization

- ✓ Lemmatization is a more advanced compared to stemming.
- ✓ It follow specific rules to get results.
- ✓ It understand the context, parts of the speech to understand the root of the word

Program Name: Lemmatization
demo11.py

```
from nltk.stem import WordNetLemmatizer

wl = WordNetLemmatizer()

print("rocks :", wl.lemmatize("rocks"))
print("corpora :", wl.lemmatize("corpora"))
print("better :", wl.lemmatize("better", pos = "a"))
```

Output

```
rocks : rock
corpora : corpus
better : good
```

Program Name: Lemmatization
demo12.py

```
import nltk
from nltk.stem import WordNetLemmatizer

wl = WordNetLemmatizer()

text = "studies studying cries cry"

tokens = nltk.word_tokenize(text)

for word in tokens:
        print("Lemmatization for {} is {}".format(word, wl.lemmatize(word)))
```

Output

```
Lemmatization for studies is study
Lemmatization for studying is studying
Lemmatization for cries is cry
Lemmatization for cry is cry
```

7. Stop word removal

- ✓ Some common words that are present in text but do not contribute in the meaning of a sentence.
- ✓ Such words are not at all important for the purpose of information retrieval or natural language processing.
- ✓ The most common stopwords are ‘a’, ‘in’ ‘the’ etc
- ✓ The good news is that nltk comes with a list of stop words and in different languages.

Program Name: list of the stopwords
demo13.py

```
import nltk
from nltk.corpus import stopwords

print(stopwords.words('english'))
```

Output

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
"you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they',
'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this',
'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been',
'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the',
'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with',
'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above',
'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again',
'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any',
'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not',
'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don',
"don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren',
"aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',
"hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
"shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn',
"wouldn't"]
```

Program Name: list of the languages
demo13.py

```
from nltk.corpus import stopwords

langs = stopwords.fileids()
print(len(langs))
```

Output

```
29
```

Program Name: Lemmatization
demo14.py

```
import nltk
from nltk.corpus import stopwords

nltk.download('stopwords')

mylist = stopwords.words('english')

line = "This is really good, how are you doing"

postPa = [word for word in line.split()]

print(postPa)
```

Output

```
['This', 'is', 'really', 'good,', 'how', 'are', 'you', 'doing']
```

Program Name: Lemmatization
demo15.py

```
import nltk
from nltk.corpus import stopwords

nltk.download('stopwords')

mylist = stopwords.words('english')

line = "This is really good, how are you doing"

postPa = [word        for word in line.split()        if word not in mylist]

print(postPa)
```

Output

```
['This', 'really', 'good,']
```

3. NLP – Replacing and Correcting words

Contents

3. NLP – Replacing and correcting words

1. Text conversions

- ✓ We can convert the text from lower to upper and upper to lower case

Program Name	Converting lower case to upper case demo1.py
	text ="hello good morning" print(text.upper())
Output	
	HELLO GOOD MORNING

Program Name	Converting upper case to lower case demo2.py
	text ="HELLO GOOD MORNING" print(text.lower())
Output	
	hello good morning

2. Removing numbers

- ✓ By using regular expression we can remove numbers from the text

Program Name: Removing numbers from the text
demo3.py

```
import re

myString = 'Box A has 4 red and 6 white balls, while Box B has 3 red and 5 blue balls.'

output = re.sub(r'\d+', '', myString)
print(output)
```

Output

```
Box A has red and white balls, while Box B has red and blue balls.
```

3. Removing punctuations

- ✓ By using regular expression we can remove the punctuations from the text

Program Name: Removing the punctuations from the text
demo4.py

```
import re

text = "Hello $@#$# Good !@#!@# morning #*#@&@#"

print("Text is:", text)

res = re.sub(r'[^\w\s]', '', text )

print("After punctuations:", res)
```

Output

```
Text is: Hello $@#$# Good !@#!@# morning #*#@&@#
After punctuations: Hello  Good  morning
```

4. Removing whitespaces

- ✓ We can remove the whitespaces in string by using strip() method.

Program Name: Removing whitespaces from text
demo5.py

```
text = "          a sample string      "

print(text)
res = text.strip()
print(res)
```

Output

```
          a sample string
a sample string
```

5. Part of Speech Tagging (POS)

- ✓ The goal of POS is to assign the various parts of a speech to every word of the provided text like nouns, adjectives, verbs, etc.
- ✓ This is normally done based on the definition and the context.
- ✓ Install textblob library,
  - pip install textblob

Program Name: Removing whitespaces from text
demo6.py

```
from textblob import TextBlob
import nltk

nltk.download('averaged_perceptron_tagger')

myString = "Parts of speech: an article, to run, fascinating, quickly, and, of"

output = TextBlob(myString)
print(output.tags)
```

Output

```
[('Parts', 'NNS'), ('of', 'IN'), ('speech', 'NN'), ('an', 'DT'), ('article', 'NN'), ('to', 'TO'), ('run', 'VB'), ('fascinating', 'VBG'), ('quickly', 'RB'), ('and', 'CC'), ('of', 'IN')]
```

Some examples are as below:

Abbreviation	Meaning
CC	coordinating conjunction
CD	cardinal digit
DT	determiner
EX	existential there
FW	foreign word
IN	preposition/subordinating conjunction
JJ	adjective (large)
JJR	adjective, comparative (larger)
JJS	adjective, superlative (largest)
LS	list market
MD	modal (could, will)
NN	noun, singular (cat, tree)
NNS	noun plural (desks)
NNP	proper noun, singular (sarah)
NNPS	proper noun, plural (indians or americans)
PDT	predeterminer (all, both, half)
POS	possessive ending (parent\ 's)
PRP	personal pronoun (hers, herself, him,himself)
PRP$	possessive pronoun (her, his, mine, my, our)
RB	adverb (occasionally, swiftly)
RBR	adverb, comparative (greater)
RBS	adverb, superlative (biggest)
RP	particle (about)
TO	infinite marker (to)
UH	interjection (goodbye)

VB	verb (ask)
VBG	verb gerund (judging)
VBD	verb past tense (pleaded)
VBN	verb past participle (reunified)
VBP	verb, present tense not 3rd person singular(wrap)
VBZ	verb, present tense with 3rd person singular (bases)
WDT	wh-determiner (that, what)
WP	wh- pronoun (who)
WRB	wh- adverb (how)

Program Name: pos example
demo7.py

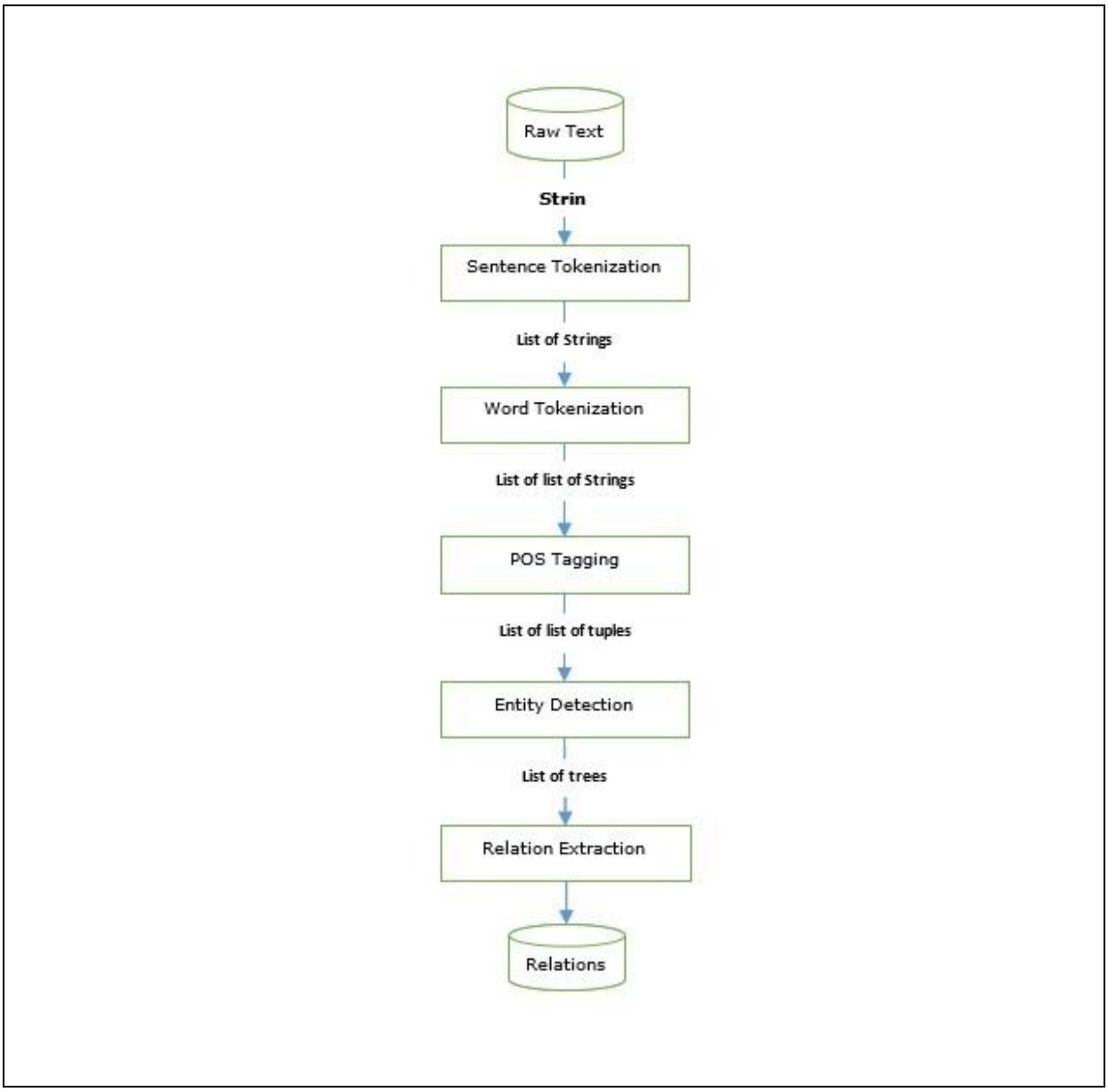
```
from nltk.corpus import wordnet

syn = wordnet.synsets('hello')[0]
print("Syn tag : ", syn.pos())

syn = wordnet.synsets('doing')[0]
print("Syn tag : ", syn.pos())

syn = wordnet.synsets('beautiful')[0]
print("Syn tag : ", syn.pos())
```

Output

```
Syn tag :  n
Syn tag :  v
Syn tag :  a
Syn tag :  r
```

6. Information Extraction

- ✓ We need to understand the tags and parsers to build information extraction engine.
- ✓ Let us see a basic information extraction pipeline



7. Information extraction has many applications including

- ✓ Business intelligence
- ✓ Resume harvesting
- ✓ Media analysis
- ✓ Sentiment detection
- ✓ Patent search
- ✓ Email scanning

8. Collocations: Bigrams and Trigrams

What is Collocations?

- ✓ Collocations are the pairs of words occurring together many times in paragraphs.
- ✓ It is calculated by the number of those pair occurring together to the overall word count of the paragraph.
- ✓ We can say that finding collocations requires calculating the frequencies of words and their appearance in the context of other words.

Bigrams and Trigrams

- ✓ Collocation can be categorized into two types,
  - Bigrams combination of two words
  - Trigrams combination of three words
- ✓ Bigrams and Trigrams provide more meaningful and useful features for the feature extraction stage.
- ✓ These are especially useful in text-based sentimental analysis.

Program Name: Bigram example
demo8.py

```
import nltk

text = "Data Science is a totally new kind of learning experience."
Tokens = nltk.word_tokenize(text)
output = list(nltk.bigrams(Tokens))

print(output)
```

Output

```
[('Data', 'Science'), ('Science', 'is'), ('is', 'a'), ('a', 'totally'), ('totally', 'new'),
('new', 'kind'), ('kind', 'of'), ('of', 'learning'), ('learning', 'experience'),
('experience', '.')]
```

Program Name: Trigram example
demo9.py

```
import nltk

text = "Data Science is a totally new kind of learning experience."
Tokens = nltk.word_tokenize(text)
output = list(nltk.trigrams(Tokens))

print(output)
```

Output

```
[('Data', 'Science', 'is'), ('Science', 'is', 'a'), ('is', 'a', 'totally'), ('a', 'totally',
'new'), ('totally', 'new', 'kind'), ('new', 'kind', 'of'), ('kind', 'of', 'learning'), ('of',
'learning', 'experience'), ('learning', 'experience', '.')]
```

9. Wordnet

- ✓ Wordnet is an NLTK lexical database for English.
- ✓ It can be used to find the meaning of words, synonym or antonym.

synset

- ✓ Synset is a special kind of a simple interface that is present in NLTK to look up words in Wordnet.
- ✓ Synset instances are the groupings of synonymous words that express the same concept.

Program Name: wordnet example
demo10.py

```
from nltk.corpus import wordnet

syn = wordnet.synsets('hello')[0]

print ("Synset name :", syn.name())
print ("Synset meaning :", syn.definition())
print ("Synset example :", syn.examples())
```

Output

```
Synset name : hello.n.01
Synset meaning : an expression of greeting
Synset example : ['every morning they exchanged polite hellos']
```

Program Name: wordnet example
demo11.py

```
from nltk.corpus import wordnet

syn = wordnet.synsets('boy')[0]

print ("Synset name :", syn.name())
print ("Synset meaning :", syn.definition())
print ("Synset example :", syn.examples())
```

Output

```
Synset name : male_child.n.01
Synset meaning : a youthful male person
Synset example : ['the baby was a boy', 'she made the boy brush his teeth
every night', 'most soldiers are only boys in uniform']
```

Program Name: wordnet example
demo12.py

```
from nltk.corpus import wordnet

syn = wordnet.synsets('good')[0]

print ("Synset name :", syn.name())
print ("Synset meaning :", syn.definition())
print ("Synset example :", syn.examples())
```

Output

```
Synset name : good.n.01
Synset meaning : benefit
Synset example: ['for your own good', "what's the good of worrying?"]
```

4. NLP – Components in NLP

Contents

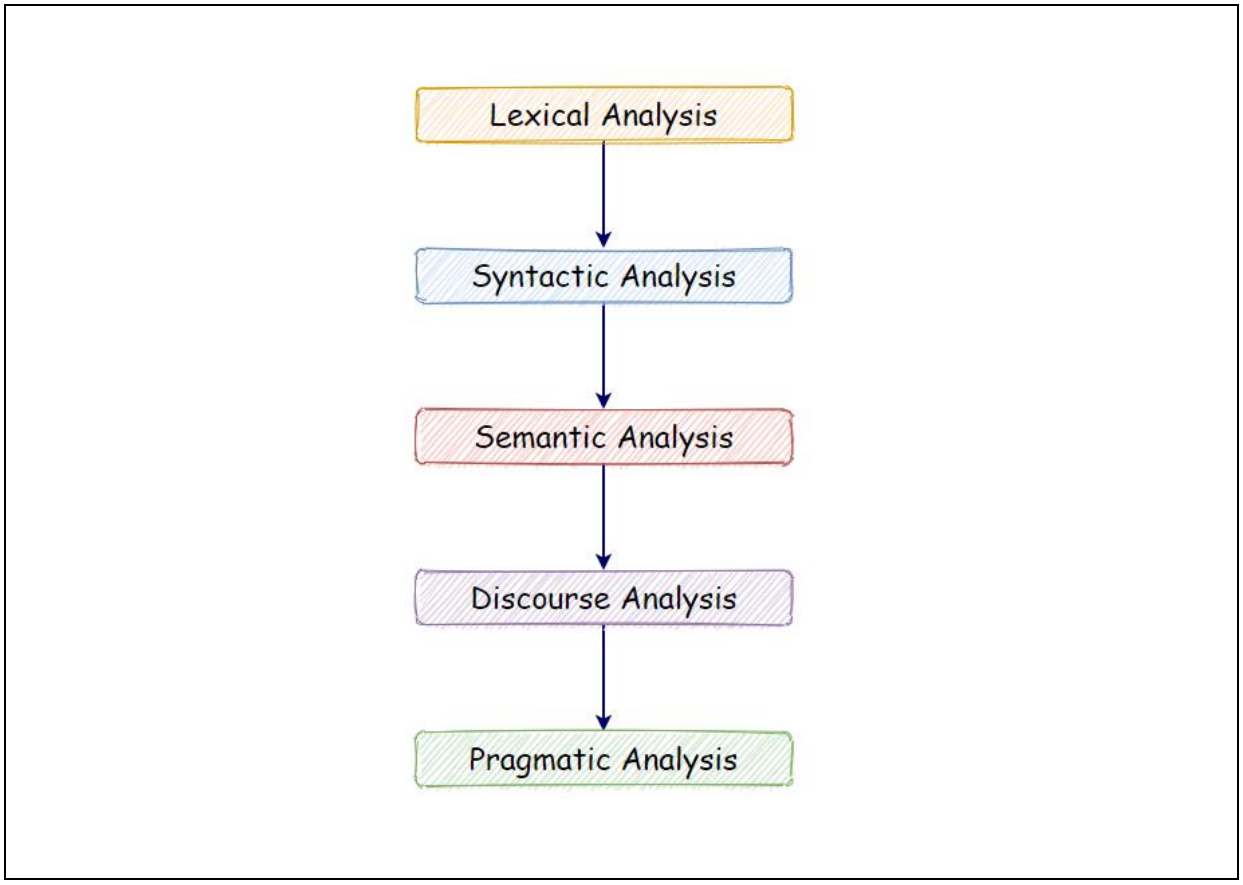
4. NLP – Components in NLP

1. Common nlp libraries

- ✓ Nltk library
- ✓ Spacy library
- ✓ TextBlob library
- ✓ Gensim framework
- ✓ Pattern framework

2. Components in NLP

- ✓ There are mainly five components in NLP
  - Lexical Analysis
  - Syntactic Analysis
  - Semantic Analysis
  - Discourse Analysis
  - Pragmatic analysis



2.1. Lexical Analysis

- ✓ Lexical means relating to the words or vocabulary of a language.
- ✓ With lexical analysis, we divide a whole chunk of text into,
  - Paragraphs.
  - Sentences.
  - Words.
- ✓ It involves identifying and analysing words' structure.

2.2. Syntactic Analysis

- ✓ Syntactic means according to the syntax
- ✓ In syntactic analysis, it is the process of analysing the words in sentence.
- ✓ Analysing the grammar and arranging the words in a manner that shows the relationship among the words.
- ✓ Example
  - The sentence "The shop goes to the house" does not pass means invalid

2.3. Semantic Analysis

- ✓ Syntactic means relating to meaning in language.
- ✓ Semantic analysis draws the exact meaning for the words, and it analyses the text to get meaningful.
- ✓ Example
  - The sentences such as "hot ice-cream" do not pass or invalid

2.4. Discourse Analysis

- ✓ Discourse means written or spoken communication.
- ✓ It considers the meaning of the sentence before it ends.
- ✓ Example
  - The sentences such as "He works at Google" in this sentence "he" should be first word in sentence

2.5. Pragmatic Analysis

- ✓ Pragmatic means practical, especially when making decisions
- ✓ Pragmatic analysis deals with overall communication and interpretation of language.
- ✓ It deals with deriving meaningful use of language in various situations.

3. Use case

- ✓ Process the below textual information

story_input.txt

Once upon a time there was an old mother pig that had three little pigs and not enough food to feed them. So, when they were old enough, she sent them out into the world to seek their fortunes.

The first little pig was very lazy. He didn't want to work at all and he built his house out of straw. The second little pig worked a little bit harder but he was somewhat lazy too and he builds his house out of sticks. Then, they sang and danced and played together and rest of the day.

The third little pig worked hard all day and built his house with bricks. It was a sturdy house complete with a fine fireplace and chimney. It looked like it could withstand the strongest winds.

Program Name: Reading textual information from file
demo1.py

```
data = open("story_input.txt")

text = data.read()

print(text)
print()
print(type(text))
print("Length of the text is:", len(text))
```

Output

```
Once upon a time there was an old mother pig that had three little pigs and not enough food to feed the
m. So, when they were old enough, she sent them out into the world to seek their fortunes.

The first little pig was very lazy. He didn't want to work at all and he built his house out of straw.
The second little pig worked a little bit harder but he was somewhat lazy too and he builds his house o
ut of sticks. Then, they sang and danced and played together and rest of the day.

The third little pig worked hard all day and built his house with bricks. It was a sturdy house complet
e with a fine fireplace and chimney. It looked like it could withstand the strongest winds.
<class 'str'>
Length of the text is: 678
```

Program Name: Sentence tokenization
demo2.py

```
import nltk
from nltk import sent_tokenize

data = open("story_input.txt")
text = data.read()

sentences = sent_tokenize(text)

print("Total sentences:", len(sentences))

for sent in sentences:
        print(sent)
```

Output

```
Total sentences: 9
Once upon a time there was an old mother pig that had three little pigs and not enough food to feed the
m.
So, when they were old enough, she sent them out into the world to seek their fortunes.
The first little pig was very lazy.
He didn't want to work at all and he built his house out of straw.
The second little pig worked a little bit harder but he was somewhat lazy too and he builds his house o
ut of sticks.
Then, they sang and danced and played together and rest of the day.
The third little pig worked hard all day and built his house with bricks.
It was a sturdy house complete with a fine fireplace and chimney.
It looked like it could withstand the strongest winds.
```

Program Name: Word tokenization
demo3.py

```
import nltk
from nltk import word_tokenize

data = open("story_input.txt")
text = data.read()

words = word_tokenize(text)
print(words)
```

Output

```
['Once', 'upon', 'a', 'time', 'there', 'was', 'an', 'old', 'mother', 'pig', 'that', 'had', 'three', 'li
ttle', 'pigs', 'and', 'not', 'enough', 'food', 'to', 'feed', 'them', '.', 'So', ',', 'when', 'they', 'w
ere', 'old', 'enough', ',', 'she', 'sent', 'them', 'out', 'into', 'the', 'world', 'to', 'seek', 'their'
, 'fortunes', '.', 'The', 'first', 'little', 'pig', 'was', 'very', 'lazy', '.', 'He', 'did', "n't", 'wa
nt', 'to', 'work', 'at', 'all', 'and', 'he', 'built', 'his', 'house', 'out', 'of', 'straw', '.', 'The',
 'second', 'little', 'pig', 'worked', 'a', 'little', 'bit', 'harder', 'but', 'he', 'was', 'somewhat', '
lazy', 'too', 'and', 'he', 'builds', 'his', 'house', 'out', 'of', 'sticks', '.', 'Then', ',', 'they', '
sang', 'and', 'danced', 'and', 'played', 'together', 'and', 'rest', 'of', 'the', 'day', '.', 'The', 'th
ird', 'little', 'pig', 'worked', 'hard', 'all', 'day', 'and', 'built', 'his', 'house', 'with', 'bricks'
, '.', 'It', 'was', 'a', 'sturdy', 'house', 'complete', 'with', 'a', 'fine', 'fireplace', 'and', 'chimn
ey', '.', 'It', 'looked', 'like', 'it', 'could', 'withstand', 'the', 'strongest', 'winds', '.']
```

Program Name: Finding word frequency
demo4.py

```
import nltk
from nltk import word_tokenize
from nltk.probability import FreqDist

data = open("story_input.txt")
text = data.read()

words = word_tokenize(text)

fdist = FreqDist(words)
result = fdist.most_common(10)
print(result)
```

Output

```
[('.', 9),
('and', 8),
('little', 5),
('a', 4),
('was', 4),
('pig', 4),
('house', 4),
('to', 3),
(',', 3),
('out', 3)]
```

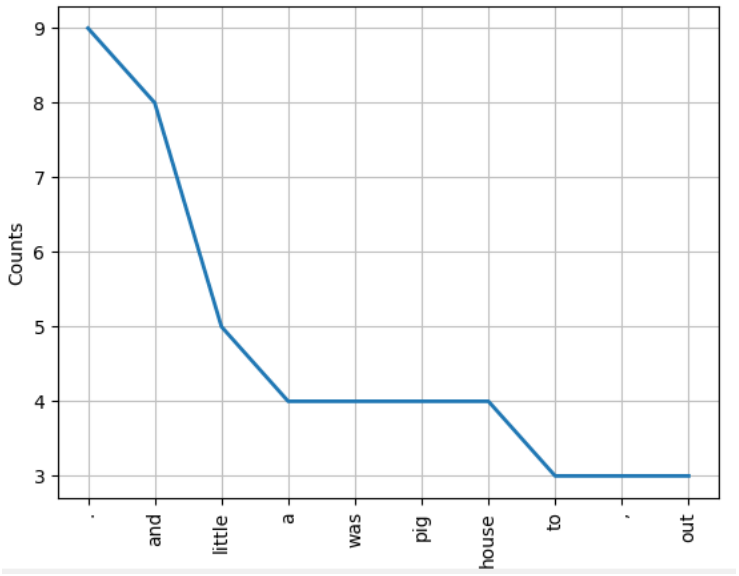
Note

- ✓ Notice that the most used words are punctuation marks and stopwords.
- ✓ We will have to remove such words to analyse the actual text.

Program Name: Plotting the frequency graph of words with punctuations
demo5.py

```
import nltk
from nltk import word_tokenize
from nltk.probability import FreqDist
import matplotlib.pyplot as plt

data = open("story_input.txt")
text = data.read()

words = word_tokenize(text)

fdist = FreqDist(words)
fdist.plot(10)
```

Output



- ✓ In the graph above, notice that a period "." is used nine times in our text.
- ✓ Analytically speaking, punctuation marks are not that important for natural language processing.
- ✓ Therefore, in the next step, we will be removing such punctuation marks.

Program Name: Removing the punctuation marks
demo6.py

```
import nltk
from nltk import word_tokenize

data = open("story_input.txt")
text = data.read()

words = word_tokenize(text)

words_no_punc = []

for w in words:
        if w.isalpha():
                words_no_punc.append(w.lower())

print(words_no_punc)
print("Length of words:", len(words_no_punc))
```

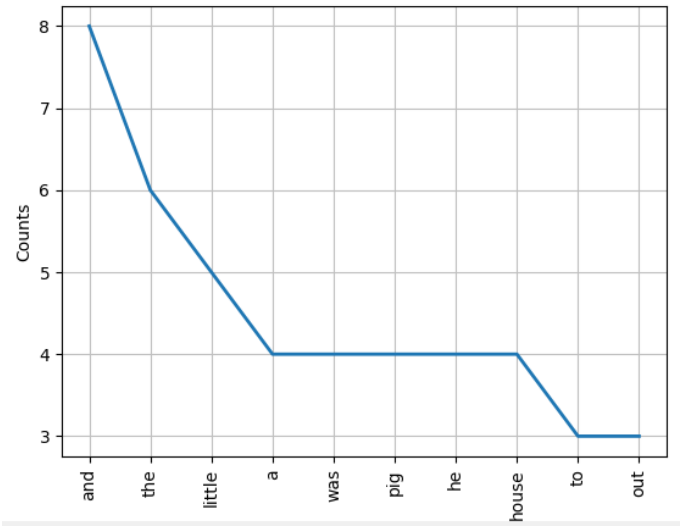
Output

```
['once', 'upon', 'a', 'time', 'there', 'was', 'an', 'old', 'mother', 'pig', 'that', 'had', 'three', 'li
ttle', 'pigs', 'and', 'not', 'enough', 'food', 'to', 'feed', 'them', 'so', 'when', 'they', 'were', 'old
', 'enough', 'she', 'sent', 'them', 'out', 'into', 'the', 'world', 'to', 'seek', 'their', 'fortunes', '
the', 'first', 'little', 'pig', 'was', 'very', 'lazy', 'he', 'did', 'want', 'to', 'work', 'at', 'all',
'and', 'he', 'built', 'his', 'house', 'out', 'of', 'straw', 'the', 'second', 'little', 'pig', 'worked',
 'a', 'little', 'bit', 'harder', 'but', 'he', 'was', 'somewhat', 'lazy', 'too', 'and', 'he', 'builds',
'his', 'house', 'out', 'of', 'sticks', 'then', 'they', 'sang', 'and', 'danced', 'and', 'played', 'toget
her', 'and', 'rest', 'of', 'the', 'day', 'the', 'third', 'little', 'pig', 'worked', 'hard', 'all', 'day
', 'and', 'built', 'his', 'house', 'with', 'bricks', 'it', 'was', 'a', 'sturdy', 'house', 'complete', '
with', 'a', 'fine', 'fireplace', 'and', 'chimney', 'it', 'looked', 'like', 'it', 'could', 'withstand',
'the', 'strongest', 'winds']
Lenght of words: 132
```

Program Name: Plotting the frequency graph of words without punctuations
demo7.py

```
import nltk
from nltk import word_tokenize
from nltk.probability import FreqDist
import matplotlib.pyplot as plt

data = open("story_input.txt")
text = data.read()

words = word_tokenize(text)

words_no_punc = []

for w in words:
        if w.isalpha():
                words_no_punc.append(w.lower())

fdist = FreqDist(words_no_punc)
fdist.plot(10)
```

Output



- ✓ Notice that we still have many words that are not very useful in the analysis of our text file sample, such as “and,” “but,” “so,” and others.
- ✓ Next, we need to remove coordinating conjunctions.

Program Name: List of the stop words
demo8.py

```
from nltk.corpus import stopwords

stopwords = stopwords.words('english')
print(stopwords)
```

Output

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you
'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', '
hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what
', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were
', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the
', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about
', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from'
, 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here',
'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', '
some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can',
 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'a
in', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'h
asn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'ne
edn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won',
 "won't", 'wouldn', "wouldn't"]
```

Program Name: Removing the stop words
demo9.py

```
import nltk
from nltk import word_tokenize
from nltk.probability import FreqDist
import matplotlib.pyplot as plt
from nltk.corpus import stopwords

data = open("story_input.txt")
text = data.read()

stopwords = stopwords.words('english')
words = word_tokenize(text)


words_no_punc = []

for w in words:
    if w.isalpha():
        words_no_punc.append(w.lower())


clean_words = []

for w in words_no_punc:
    if w not in stopwords:
        clean_words.append(w)

print(clean_words)
print("Length of clean words:", len(clean_words))
```

Output

```
['upon', 'time', 'old', 'mother', 'pig', 'three', 'little', 'pigs', 'enough', 'food', 'feed', 'old', 'e
nough', 'sent', 'world', 'seek', 'fortunes', 'first', 'little', 'pig', 'lazy', 'want', 'work', 'built',
 'house', 'straw', 'second', 'little', 'pig', 'worked', 'little', 'bit', 'harder', 'somewhat', 'lazy',
'builds', 'house', 'sticks', 'sang', 'danced', 'played', 'together', 'rest', 'day', 'third', 'little',
'pig', 'worked', 'hard', 'day', 'built', 'house', 'bricks', 'sturdy', 'house', 'complete', 'fine', 'fir
eplace', 'chimney', 'looked', 'like', 'could', 'withstand', 'strongest', 'winds']
Lenght of clean words 65
```

Program Name: Word frequency distribution
demo10.py

```
import nltk
from nltk import word_tokenize
from nltk.probability import FreqDist
import matplotlib.pyplot as plt
from nltk.corpus import stopwords

data = open("story_input.txt")
text = data.read()

stopwords = stopwords.words('english')
words = word_tokenize(text)

words_no_punc = []

for w in words:
    if w.isalpha():
        words_no_punc.append(w.lower())

clean_words = []

for w in words_no_punc:
    if w not in stopwords:
        clean_words.append(w)

fdist = FreqDist(clean_words)
result = fdist.most_common(10)
print(result)
```

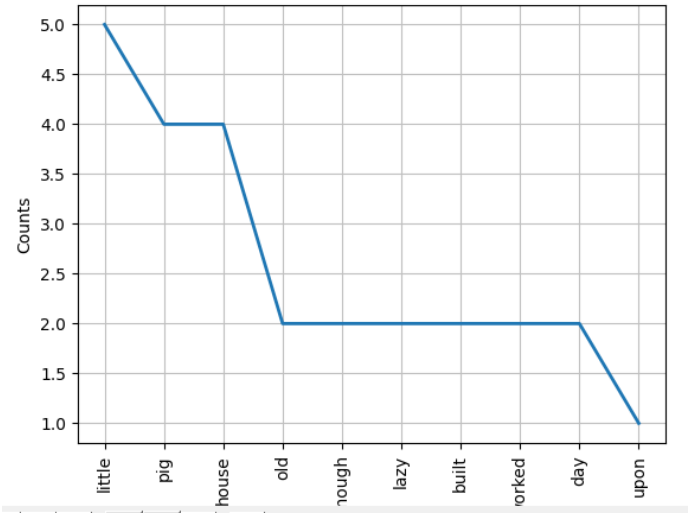
Output

```
[('little', 5),
('pig', 4),
('house', 4),
('old', 2),
('enough', 2),
('lazy', 2),
('built', 2),
('worked', 2),
('day', 2),
('upon', 1)]
```

Program Name: Plotting the useful words
demo11.py

```
import nltk
from nltk import word_tokenize
from nltk.probability import FreqDist
import matplotlib.pyplot as plt
from nltk.corpus import stopwords

data = open("story_input.txt")
text = data.read()

stopwords = stopwords.words('english')
words = word_tokenize(text)

words_no_punc = []

for w in words:
        if w.isalpha():
                words_no_punc.append(w.lower())


clean_words = []

for w in words_no_punc:
        if w not in stopwords:
                clean_words.append(w)


fdist = FreqDist(clean_words)
fdist.most_common(10)
fdist.plot(10)
```

Output



✓ As shown above, the final graph has many useful words that help us understand what our sample data is about, showing how essential it is to perform data cleaning on NLP.

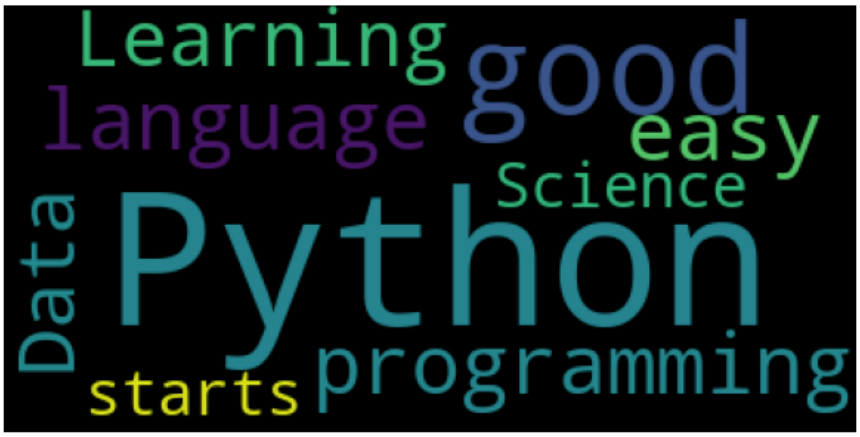
4. Word Cloud

- ✓ Word Cloud is a data visualization technique.
- ✓ In which words from a given text display on the main chart.
- ✓ In this technique, more frequent or essential words display in a larger and bolder font.
- ✓ Less frequent or essential words display in smaller or thinner fonts.
- ✓ It is a beneficial technique in NLP that gives us a glance at what text should be analysed.

Program Name: Wordcloud example
demo12.py

```
import matplotlib.pyplot as plt
from wordcloud import WordCloud

text = "Python is good programming language, Python is very easy, Learning Data Science starts from Python"

wordcloud = WordCloud().generate(text)

plt.figure(figsize = (12, 12))
plt.imshow(wordcloud)

plt.axis('off')
plt.show()
```

Output



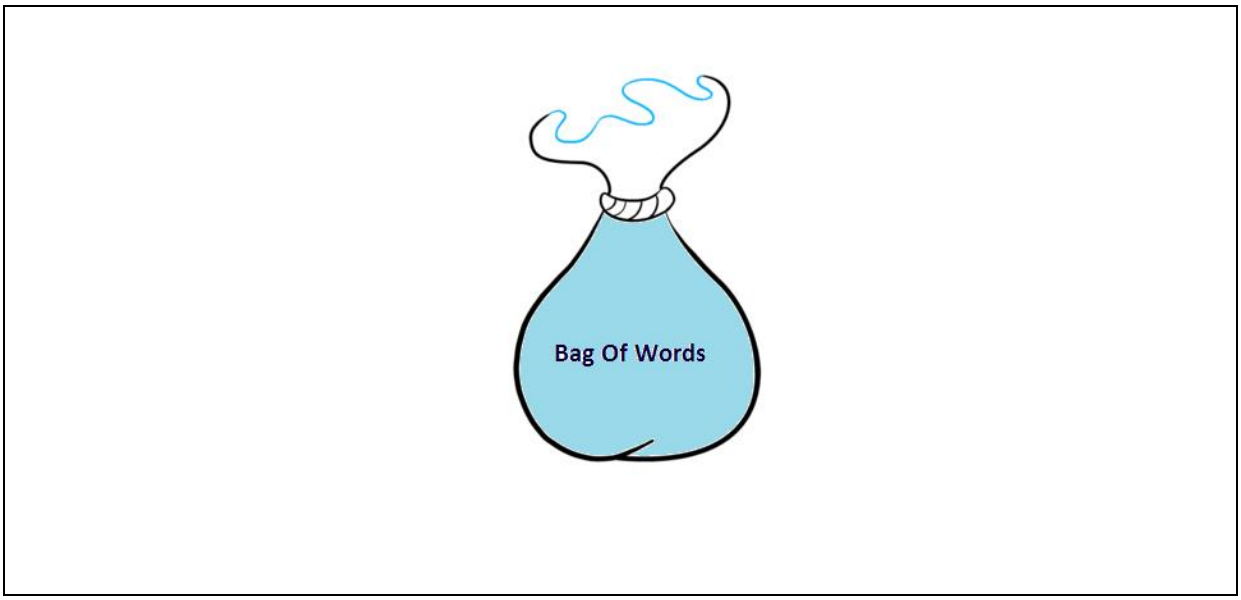
5. NLP – Bag of words, TF and IDF

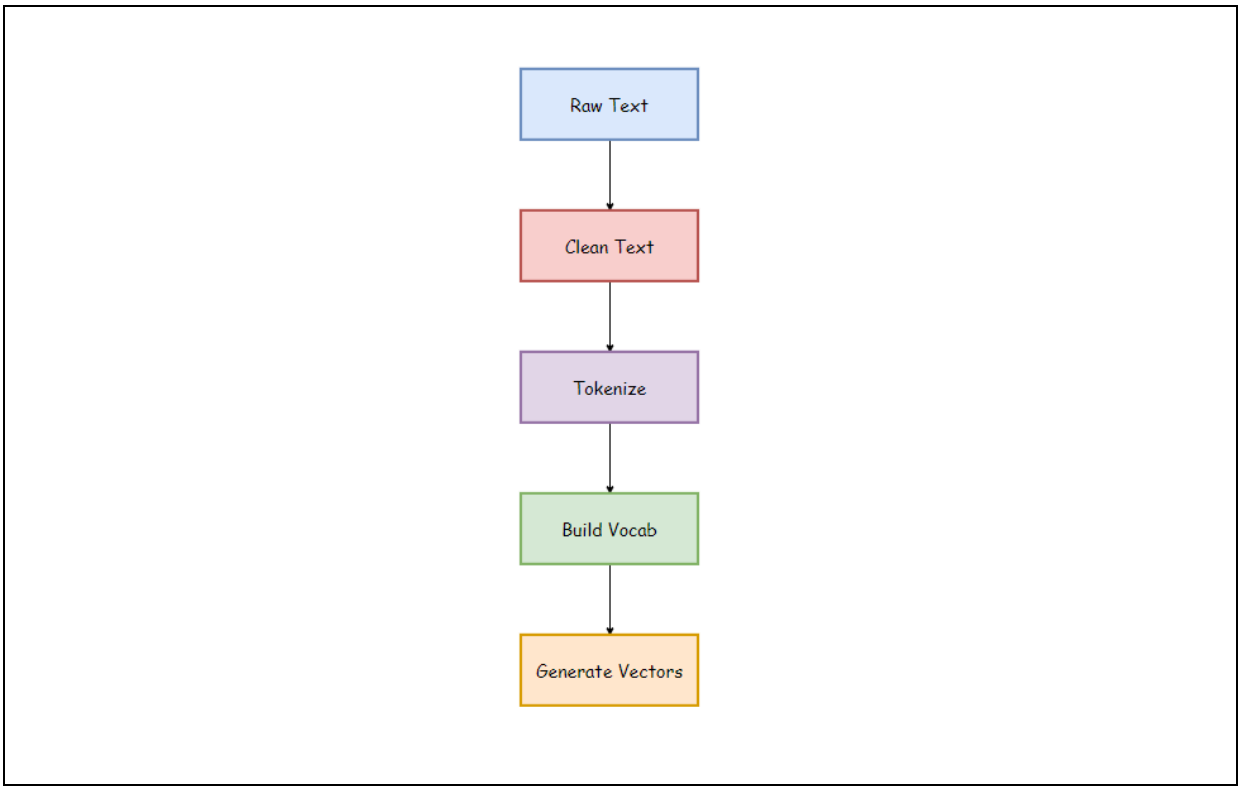
Contents

5. NLP – Bag of words, TF and IDF

1. NLP – Components in NLP Bag-of-Words

- ✓ It is a method of extracting essential features from raw text.
- ✓ So that we can use it for machine learning models.
- ✓ A bag of words model converts the raw text into words, and it also counts the frequency for the words in the text.





1.1. Raw Text

- ✓ This is the original text on which we want to perform analysis.

1.2. Clean Text

- ✓ Since our raw text contains some unnecessary data like punctuation marks and stopwords, so we need to clean up our text.

1.3. Tokenize

- ✓ Tokenization represents the sentence as a group of tokens or words.

1.4. Building Vocab

- ✓ It contains total words used in the text after removing unnecessary data.

1.5. Generate Vocab

- ✓ It contains the words along with their frequencies in the sentences.

2. Use case

Let's take few sentences

- ✓ Jim and Pam travelled by bus.
- ✓ The train was late.
- ✓ The flight was full. Travelling by flight is expensive.

2.1. Creating a basic structure

Sentence 1	Sentence2	Sentence 3
Jim	The	The
and	train	flight
Pam	was	was
travelled	late	full
by		Travelling
the		by
bus		flight
		is
		expensive

2.2. Words with frequencies

Sentence1	Count	Sentence2	Count	Sentence3	Count
Jim	1	The	1	The	1
and	1	train	1	flight	2
Pam	1	was	1	was	1
travelled	1	late	1	full	1
by	1			Travelling	1
the	1			by	1
bus	1			is	1
				expensive	1

2.3. Combining all the words

Sentence	Frequency
and	1
bus	1
by	2
expensive	1
flight	2
full	1
is	1
jim	1
late	1
pam	1
the	3
train	1
travelled	1
travelling	1
was	1

2.4. Final model

	and	bus	by	expensive	flight	full	is	jim	Late	pam	The	train	travelled	travelling	was
S-1	1	1	1	0	0	0	0	1	0	1	1	0	1	0	0
S-2	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1
S-3	0	0	1	1	2	1	1	0	0	0	1	0	0	1	1

Program Name: Bag of the words

demo1.py

```
from sklearn.feature_extraction.text import CountVectorizer

sentences = ["Jim and Pam travelled by bus.",
"The train was late",
"The flight was full. Travelling by flight is expensive"]

cv = CountVectorizer()

B_O_W = cv.fit_transform(sentences).toarray()

print(B_O_W)
```

Output

```
[[1 1 1 0 0 0 0 1 0 1 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 1 0 1 1 0 0 1]
 [0 0 1 1 2 1 1 0 0 0 1 0 0 1 1]]
```

Applications

- ✓ This concept we can use in nlp applications
- ✓ Information retrieval from documents.
- ✓ Classifications of documents.

Limitations

- ✓ Semantic meaning:
  - It does not consider the semantic meaning of a word.
- ✓ Vector size:
  - For large documents, the vector size increase, which may result in higher computational time?
- ✓ Preprocessing:
  - In preprocessing, we need to perform data cleansing before using it.

3. Term Frequency-Inverse Document Frequency (TF-IDF)

- ✓ "**T**erm **F**requency – **I**nverse **D**ocument **F**requency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection."

Here's a sample of reviews about a particular horror movie:

- ✓ Review 1: This movie is very scary and long
- ✓ Review 2: This movie is not scary and is slow
- ✓ Review 3: This movie is spooky and good

TF: Term Frequency

$$TF = \frac{Frequency\ of\ the\ word\ in\ the\ sentence}{Total\ number\ of\ words\ in\ the\ sentence}$$

Term	Review 1	Review 2	Review 3	TF (Review 1)	TF (Review 2)	TF (Review 3)
This	1	1	1	1/7	1/8	1/6
movie	1	1	1	1/7	1/8	1/6
is	1	2	1	1/7	1/4	1/6
very	1	0	0	1/7	0	0
scary	1	1	0	1/7	1/8	0
and	1	1	1	1/7	1/8	1/6
long	1	0	0	1/7	0	0
not	0	1	0	0	1/8	0
slow	0	1	0	0	1/8	0
spooky	0	0	1	0	0	1/6
good	0	0	1	0	0	1/6

Inverse Document Frequency (IDF)

- ✓ IDF is a measure of how important a term is in a sentence
- ✓ We need the IDF value because computing just the TF alone is not sufficient to understand the importance of words

$$idf_t = \log \frac{number\ of\ documents}{number\ of\ documents\ with\ term\ 't'}$$

Example: Review 2

- ✓ IDF ('movie') = log(3/3) = 0
- ✓ IDF ('is') = log (3/3) = 0
- ✓ IDF ('not') = log (3/1) = 0.48
- ✓ IDF ('scary') = log(3/2) = 0.18
- ✓ IDF ('and') = log(3/3) = 0
- ✓ IDF ('slow') = log(3/1) =0.48

Calculating IDF

Term	Review 1	Review 2	Review 3	IDF
This	1	1	1	0.00
movie	1	1	1	0.00
is	1	2	1	0.00
very	1	0	0	0.48
scary	1	1	0	0.18
and	1	1	1	0.00
long	1	0	0	0.48
not	0	1	0	0.48
slow	0	1	0	0.48
spooky	0	0	1	0.48
good	0	0	1	0.48

- ✓ We can now compute the TF-IDF score for each word in the corpus.
- ✓ Words with a higher score are more important, and lower score are less important

Calculating final TF-IDF values:

- ✓ TF-IDF = TF * IDF

Example

Review 2

TF-IDF('this') = TF('this') * IDF('this') = 1/8 * 0 = 0

Term	TF-IDF (Review 1)	TF-IDF (Review 2)	TF-IDF (Review 3)
This	0.000	0.000	0.000
movie	0.000	0.000	0.000
is	0.000	0.000	0.000
very	0.068	0.000	0.000
scary	0.025	0.022	0.000
and	0.000	0.000	0.000
long	0.068	0.000	0.000
not	0.000	0.060	0.000
slow	0.000	0.060	0.000
spooky	0.000	0.000	0.080
good	0.000	0.000	0.080

6. NLP – Twitter Sentiment Analysis – Textblob

Contents

6. NLP – Twitter Sentiment Analysis – Textblob

1. TextBlob

- ✓ TextBlob provides an API that can perform different Natural Language Processing (NLP) tasks like,
  - Part-of-Speech Tagging
  - Noun Phrase Extraction
  - Sentiment Analysis
  - Classification (Naive Bayes, Decision Tree)
  - Language Translation and Detection
  - Spelling Correction etc.

- ✓ TextBlob is built upon Natural Language Toolkit (NLTK).
- ✓ Sentiment Analysis means analysing the sentiment of a given text or document and categorizing the text/document into a specific class or category (like positive and negative).
- ✓ Basically, the classification is done for two classes: positive and negative.
- ✓ However, we can add more classes like neutral, highly positive, highly negative, etc.

2. Installing textblob library

- ✓ pip install -U textblob

3. Simple TextBlob Sentiment Analysis Example

- ✓ We can apply textblob on any text to do Sentiment Analysis
- ✓ The sentiment property gives the sentiment scores to the given text.
- ✓ There are two scores given: Polarity and Subjectivity.
  - The polarity score is a float within the range [-1.0, 1.0] where negative value indicates negative text and positive value indicates that the given text is positive.
  - The subjectivity is a float within the range [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective.

Program Name: Sentiment analysis
demo1.py

```
from textblob import TextBlob

text = TextBlob("It was a wonderful movie. I liked it very much.")

print (text.sentiment)
print ('polarity: {}'.format(text.sentiment.polarity))
print ('subjectivity: {}'.format(text.sentiment.subjectivity))
```

Output

```
Sentiment(polarity=0.62, subjectivity=0.6866666666666666)
polarity: 0.62
subjectivity: 0.6866666666666666
```

Program Name: Sentiment analysis
demo2.py

```
from textblob import TextBlob

text = TextBlob("I liked the acting of the lead actor but I didn't like the movie overall.")

print (text.sentiment)
print ('polarity: {}'.format(text.sentiment.polarity))
print ('subjectivity: {}'.format(text.sentiment.subjectivity))
```

Output

```
Sentiment(polarity=0.19999999999999998, subjectivity=0.26666666666666666)
polarity: 0.19999999999999998
subjectivity: 0.26666666666666666
```

Program Name: Sentiment analysis
demo3.py

```
from textblob import TextBlob

text = TextBlob("I liked the acting of the lead actor and I liked the movie overall.")

print (text.sentiment)
print ('polarity: {}'.format(text.sentiment.polarity))
print ('subjectivity: {}'.format(text.sentiment.subjectivity))
```

Output

```
Sentiment(polarity=0.3, subjectivity=0.4)
polarity: 0.3
subjectivity: 0.4
```

4. Using NLTK's Twitter Corpus

- ✓ We use the twitter_samples corpus to train the TextBlob's NaiveBayesClassifier.
- ✓ Using the twitter_samples corpus, we create a train set and test set containing a certain amount of positive and negative tweets.
- ✓ And, then we test the accuracy of the trained classifier.

Program Name: Getting twitter sample datasets
demo4.py

```
from nltk.corpus import twitter_samples
import nltk
nltk.download('twitter_samples')

print (twitter_samples.fileids())
```

Output

```
['negative_tweets.json', 'positive_tweets.json', 'tweets.20150430-223406.json']
```

Program Name: Getting twitter sample datasets and checking length
demo5.py

```
from nltk.corpus import twitter_samples

pos_tweets = twitter_samples.strings('positive_tweets.json')
print(len(pos_tweets))

neg_tweets = twitter_samples.strings('negative_tweets.json')
print(len(neg_tweets))
```

Output

```
5000
5000
```

Program Name: Getting twitter sample datasets and checking length
demo6.py

```
from nltk.corpus import twitter_samples

all_tweets = twitter_samples.strings('tweets.20150430-223406.json')
print (len(all_tweets))
```

Output

```
20000
```

Program Name: Getting positive tweets
demo7.py

```
from nltk.corpus import twitter_samples

pos_tweets = twitter_samples.strings('positive_tweets.json')

pos_tweets_set = []

for tweet in pos_tweets:
        pos_tweets_set.append((tweet, 'pos'))

for i in pos_tweets_set:
        print(i)
```

Output

```
('#FollowFriday @France_Inte @PKuchly57 @Milipol_Paris for being top engaged members in my community th
is week :)', 'pos')
('@Lamb2ja Hey James! How odd :/ Please call our Contact Centre on 02392441234 and we will be able to a
ssist you :) Many thanks!', 'pos')
('@DespiteOfficial we had a listen last night :) As You Bleed is an amazing track. When are you in Scot
land?!', 'pos')
('@97sides CONGRATS :)', 'pos')
('yeaaaah yippppy!!!  my accnt verified rqst has succeed got a blue tick mark on my fb profile :) in 15
 days', 'pos')
('@BhaktisBanter @PallaviRuhail This one is irresistible :)\n#FlipkartFashionFriday http://t.co/EbZ0L2V
ENM', 'pos')
("We don't like to keep our lovely customers waiting for long! We hope you enjoy! Happy Friday! - LWWF
:) https://t.co/smyYriipxI", 'pos')
('@Impatientraider On second thought, there's just not enough time for a DD :) But new shorts entering
system. Sheep must be buying.', 'pos')
```

Program Name: Getting negative tweets
demo8.py

```
from nltk.corpus import twitter_samples

neg_tweets = twitter_samples.strings('negative_tweets.json')

neg_tweets_set = []

for tweet in neg_tweets:
        neg_tweets_set.append((tweet, 'neg'))

for i in neg_tweets_set:
        print(i)
```

Output

```
('hopeless for tmr :(', 'neg')
("Everything in the kids section of IKEA is so cute. Shame I'm nearly 19 in 2 months :(", 'neg')
('@Hegelbon That heart sliding into the waste basket. :(', 'neg')
('“@ketchBurning: I hate Japanese call him "bani" :( :(”\n\nMe too', 'neg')
('Dang starting next week I have "work" :(', 'neg')
("oh god, my babies' faces :( https://t.co/9fcwGvaki0", 'neg')
('@RileyMcDonough make me smile :((', 'neg')
('@f0ggstar @stuartthull work neighbour on motors. Asked why and he said hates the updates on search :(
 http://t.co/XvmTUikWln', 'neg')
('why?:("@tahuodyy: sialan:( https://t.co/Hv1i0xcrL2"', 'neg')
('Athabasca glacier was there in #1948 :-( #athabasca #glacier #jasper #jaspernationalpark #alberta #ex
plorealberta #… http://t.co/dZZdqmf7Cz', 'neg')
("I have a really good m&amp;amp;g idea but I'm never going to meet them :(((", 'neg')
('@Rampageinthebox mare ivan :(', 'neg')
```

Program Name: Splitting Training and testing datasets
demo9.py

```
from nltk.corpus import twitter_samples

pos_tweets = twitter_samples.strings('positive_tweets.json')
neg_tweets = twitter_samples.strings('negative_tweets.json')

pos_tweets_set = []

for tweet in pos_tweets:
        pos_tweets_set.append((tweet, 'pos'))

neg_tweets_set = []

for tweet in neg_tweets:
        neg_tweets_set.append((tweet, 'neg'))

from random import shuffle

shuffle(pos_tweets_set)
shuffle(neg_tweets_set)

test_set = pos_tweets_set[:300] + neg_tweets_set[:300]
train_set = pos_tweets_set[300:600] + neg_tweets_set[300:600]

print("Training and testing datasets")
```

Output

```
Training and testing datasets
```

Program Name: Creating model and training the model
demo10.py

```
from nltk.corpus import twitter_samples

pos_tweets = twitter_samples.strings('positive_tweets.json')
neg_tweets = twitter_samples.strings('negative_tweets.json')

pos_tweets_set = []

for tweet in pos_tweets:
        pos_tweets_set.append((tweet, 'pos'))

neg_tweets_set = []

for tweet in neg_tweets:
        neg_tweets_set.append((tweet, 'neg'))

from random import shuffle

shuffle(pos_tweets_set)
shuffle(neg_tweets_set)

test_set = pos_tweets_set[:300] + neg_tweets_set[:300]
train_set = pos_tweets_set[300:600] + neg_tweets_set[300:600]

from textblob.classifiers import NaiveBayesClassifier
classifier = NaiveBayesClassifier(train_set)

print("Model got trained")
```

Output

```
Model got trained
```

Program Name: Testing model accuracy
demo11.py

```
from nltk.corpus import twitter_samples

pos_tweets = twitter_samples.strings('positive_tweets.json')
neg_tweets = twitter_samples.strings('negative_tweets.json')

pos_tweets_set = []

for tweet in pos_tweets:
        pos_tweets_set.append((tweet, 'pos'))

neg_tweets_set = []

for tweet in neg_tweets:
        neg_tweets_set.append((tweet, 'neg'))

from random import shuffle

shuffle(pos_tweets_set)
shuffle(neg_tweets_set)

test_set = pos_tweets_set[:300] + neg_tweets_set[:300]
train_set = pos_tweets_set[300:600] + neg_tweets_set[300:600]

from textblob.classifiers import NaiveBayesClassifier
classifier = NaiveBayesClassifier(train_set)

accuracy = classifier.accuracy(test_set)
print (accuracy)
```

Output

```
0.73666666666667
```

Program Name: Testing the model
demo12.py

```
from nltk.corpus import twitter_samples

pos_tweets = twitter_samples.strings('positive_tweets.json')
neg_tweets = twitter_samples.strings('negative_tweets.json')

pos_tweets_set = []

for tweet in pos_tweets:
        pos_tweets_set.append((tweet, 'pos'))

neg_tweets_set = []

for tweet in neg_tweets:
        neg_tweets_set.append((tweet, 'neg'))

from random import shuffle

shuffle(pos_tweets_set)
shuffle(neg_tweets_set)

test_set = pos_tweets_set[:300] + neg_tweets_set[:300]
train_set = pos_tweets_set[300:600] + neg_tweets_set[300:600]

from textblob.classifiers import NaiveBayesClassifier
classifier = NaiveBayesClassifier(train_set)

text = "It was a wonderful movie. I liked it very much."
print (classifier.classify(text))
```

Output

```
0.735
pos
```

Program Name: Testing the model
demo13.py

```
from nltk.corpus import twitter_samples

pos_tweets = twitter_samples.strings('positive_tweets.json')
neg_tweets = twitter_samples.strings('negative_tweets.json')

pos_tweets_set = []

for tweet in pos_tweets:
        pos_tweets_set.append((tweet, 'pos'))

neg_tweets_set = []

for tweet in neg_tweets:
        neg_tweets_set.append((tweet, 'neg'))

from random import shuffle

shuffle(pos_tweets_set)
shuffle(neg_tweets_set)

test_set = pos_tweets_set[:300] + neg_tweets_set[:300]
train_set = pos_tweets_set[300:600] + neg_tweets_set[300:600]

from textblob.classifiers import NaiveBayesClassifier
classifier = NaiveBayesClassifier(train_set)

text = "I don't like movies having happy ending."
print (classifier.classify(text))
```

Output

```
0.726666666666666
neg
```

Program Name: Testing the model
demo14.py

```
from nltk.corpus import twitter_samples
from textblob import TextBlob

pos_tweets = twitter_samples.strings('positive_tweets.json')
neg_tweets = twitter_samples.strings('negative_tweets.json')

pos_tweets_set = []

for tweet in pos_tweets:
        pos_tweets_set.append((tweet, 'pos'))

neg_tweets_set = []

for tweet in neg_tweets:
        neg_tweets_set.append((tweet, 'neg'))

from random import shuffle

shuffle(pos_tweets_set)
shuffle(neg_tweets_set)

test_set = pos_tweets_set[:300] + neg_tweets_set[:300]
train_set = pos_tweets_set[300:600] + neg_tweets_set[300:600]

from textblob.classifiers import NaiveBayesClassifier
classifier = NaiveBayesClassifier(train_set)

text = "It was a wonderful movie. I liked it very much."
print(classifier.classify(text))

blob = TextBlob(text, classifier=classifier)

print(blob)
print(blob.classify())
```

Output

```
0.695
pos
It was a wonderful movie. I liked it very much.
pos
```

7. NLP – Spacy library

Contents

6. NLP – Spacy library

1. Introduction

- ✓ There are mainly 3 types of data we have
  - Structured data
  - Semi structured data
  - Unstructured data

- ✓ Today more than 80% of the data available as Unstructured Data.
- ✓ Unstructured data is a data which cannot be represented in tabular format
  - Texts,
  - Videos,
  - Images

- ✓ We can use Natural Language processing to process and interpret the unstructured data
- ✓ NLP applications,
  - Sentiment analysis
  - Speech recognition
  - Text Classification
  - Machine Translation
  - Semantic Search
  - News/article Summarization
  - Answering Questions

- ✓ Live examples,
  - Google Assistant.
  - Amazon Echo.
  - ChatBot,
  - Language Translations,
  - Article summarization and so on.

2. Install and Load Main Python Libraries for NLP

- ✓ We do have different python libraries to work with NLP.
  - Nltk
  - Spacy
  - Genism
  - transformers

2.1. Installation

- ✓ Open a command prompt and run below commands

  - pip install nltk
  - pip install spacy
  - pip install gensim
  - pip install transformers

Note: Run below command

- ✓ python -m spacy download en_core_web_sm

3. Text Pre-processing

- ✓ The raw text data also called as text corpus, it has a lot of noise.
- ✓ Raw text having punctuation, special symbols, stop words & etc
- ✓ From the raw data we need to process the text by using nlp techniques.
- ✓ A text can be converted into nlp object of spacy.

Program Name: spacy example
demo1.py

```
import spacy

nlp = spacy.load('en_core_web_sm')

raw_text = 'NLP is a very powerful tool'

text_doc = nlp(raw_text)

print(text_doc)
print(type(text_doc))
```

Output

```
NLP is a very powerful tool
<class 'spacy.tokens.doc.Doc'>
```

4. What is Tokenization?

- ✓ The process of extracting tokens from a text file/document is referred as tokenization.

Program Name: Tokenization using spacy
demo2.py

```
import spacy

nlp = spacy.load('en_core_web_sm')

raw_text = 'NLP is a very powerful tool'

text_doc = nlp(raw_text)

print(text_doc)
print()
for word in text_doc:
        print(word.text)
```

Output

```
NLP is a very powerful tool

NLP
is
a
very
powerful
tool
```

Program Name: Checking every token either it is alphabet or not by using spacy
demo3.py

```
import spacy

nlp = spacy.load('en_core_web_sm')

mixed_text = 'My salary is $1000 dollars'
mixed_text_doc = nlp(mixed_text)

for word in mixed_text_doc:
        print(word.text.ljust(10), word.is_alpha)
```

Output

```
My         True
salary     True
is         True
$          False
1000       False
dollars    True
```

Program Name: Checking every token alpha, stop word and punctuation
demo4.py

```
import spacy

nlp = spacy.load('en_core_web_sm')

text='My salary is $1000 dollars'
text_doc=nlp(text)


print("Text",'\t\t', "Alpha", "Stop", "Punct")

for word in text_doc:
        print(word.text.ljust(15), ':', word.is_alpha, word.is_stop,
        word.is_punct)
```

Output

```
Text             Alpha Stop Punct
My              : True True False
salary          : True False False
is              : True True False
$               : False False False
1000            : False False False
dollars         : True False False
```

Program Name: Checking tokens and count of tokens by using spacy
demo5.py

```
import spacy

nlp = spacy.load('en_core_web_sm')

raw_text= 'Amazon Alexa, also known simply as Alexa, is a virtual assistant AI technology developed by Amazon, first used in the Amazon Echo smart speakers developed by Amazon Lab126. It is capable of voice interaction, music playback, making to-do lists, setting alarms, streaming podcasts, playing audio books, and providing weather, traffic, sports, and other real-time information, such as news. Alexa can also control several smart devices using itself as a home automation system. Users are able to extend the Alexa capabilities by installing "skills" additional functionality developed by third-party vendors, in other settings more commonly called apps such as weather programs and audio features. Most devices with Alexa allow users to activate the device using a wake-word (such as Alexa or Amazon); other devices (such as the Amazon mobile app on iOS or Android and Amazon Dash Wand) require the user to push a button to activate Alexa listening mode, although, some phones also allow a user to say a command, such as "Alexa" or "Alexa wake". Currently, interaction and communication with Alexa are available only in English, German, French, Italian, Spanish, Portuguese, Japanese, and Hindi. '

text_doc = nlp(raw_text)

token_count = 0

for word in text_doc:
        print(word.text)
        token_count = token_count  + 1

print("Token count is:", token_count)
```

Output

```
Amazon
Alexa
,
also
known
simply
as
```

```
Alexa
,
is
a
.....

Token count is: 235
```

Note

- ✓ So, we have a lot of tokens here. The stop words like ‘it’, ’was’ ’that’, ’to’…, so on.
- ✓ These words will not give us much information, especially for models.
- ✓ Punctuations also will not provide much info
  - While dealing with large text files, the stop words and punctuations will be repeated at high levels, misguiding us to think they are important.
- ✓ So, it is necessary to filter out the stop words.

5. Stop words in spacy library

- ✓ Spacy has an inbuilt list of stop words.

Program Name: Checking stop words by using spacy
demo6.py

```
import spacy

nlp = spacy.load('en_core_web_sm')

stopwords = spacy.lang.en.stop_words.STOP_WORDS

list_stopwords = list(stopwords)

print(list_stopwords)
```

Output

```
['re', 'through', '’ll', 'besides', 'in', 'per', 'along', 'rather', 'mostly', 'ourselves', 'around', 'f
urther', 'always', 'same', 'among', 'that', 'we', 'yet', 'n’t', 'hereupon', 'own', 'have', 'must', 'the
refore', 'except', 'into', 'some', 'too', 'her', 'could', 'are', 'but', 'hereby', "'s", '’ve', 'least',
 'indeed', 'over', 'using', 'do', 'whenever', 'nowhere', 'therein', 'call', 'wherein', 'your', 'meanwhi
le', 'due', '’s', 'go', 'part', 'no', 'themselves', 'onto', 'moreover', 'almost', 'without', 'am', 'sid
e', 'a', 'make', 'anyway', 'just', 'then', 'while', 'been', 'every', 'under', 'will', 'seem', 'none', '
latterly', 'about', 'please', 'say', 'beside', 'see', 'forty', 'unless', 'ca', 'whereupon', 'everyone',
 'become', 'had', 'why', 'though', 'eleven', 'take', 'be', 'after', 'via', 'than', 'these', 'where', 'a
nyhow', 'each', 'first', 'made', 'many', 'most', 'has', 'name', 'nobody', 'amongst', 'one', 'thru', 'us
ed', 'whence', 'very', 'everything', 'cannot', 'yourselves', 'somewhere', 'would', 'of', 'four', 'at',
'elsewhere', 'may', 'last', 'before', 'much', 'next', 'hence', 'throughout', 'for', 'twelve', 'you', 's
till', 'either', '’m', 'when', 'top', 'were', 'wherever', 'with', 'alone', 'eight', 'becomes', 'front',
 'an', 'again', 'fifty', 'whether', "'ll", 'latter', 'against', 'became', 'and', 'on', 'anyone', 'anyth
ing', 'sometime', 'thereupon', 'i', 'various', 'up', 'thence', "n't", 'here', 'less', 'seeming', '‘s',
'itself', 'out', 'sometimes', 'thereby', 'third', 'to', 'quite', 'together', '‘ve', 'yours', 'often', '
nor', 'move', 'twenty', 'my', 'also', 'all', 'keep', 'doing', 'however', 'hundred', 'back', 'since', 'o
thers', 'something', 'those', 'well', 'because', 'whereby', 'hereafter', 'himself', 'me', 'full', 'hers
elf', 'myself', 'whatever', 'seemed', 'even', 'within', 'behind', 'n‘t', 'it', 'formerly', 'perhaps', '
down', 'becoming', 'them', 'upon', 'they', 'afterwards', 'ten', 'whoever', 'us', "'ve", 'she', 'otherwi
se', 'beyond', 'ever', 'which', 'once', 'was', 'so', 'whom', '‘m', 'bottom', 'neither', 'any', 'anywher
e', 'else', 'his', 'now', 'several', '’re', 'few', "'d", 'above', 'there', 'by', 'off', 'is', 'serious'
, 'if', 'get', 'sixty', 'during', 'seems', 'everywhere', 'regarding', 'namely', '‘re', 'give', '’d', 'o
ther', '‘d', 'him', 'thereafter', 'towards', 'between', 'or', 'although', 'does', 'below', 'noone', 'be
```

Program Name: Removing **stop words** and **punctuations** from text using spacy
demo7.py

```
import spacy

nlp = spacy.load('en_core_web_sm')

raw_text = "Hello good!!! morning how are you, today climate is very cool"

text_doc = nlp(raw_text)

final = [
                token
                for token in text_doc
                if not token.is_stop and not token.is_punct
]

print("Actual text:", raw_text)

for token in final:
        print(token)
```

Output

```
Actual text: Hello good!!! morning how are you, today climate is very cool
Hello
good
morning
today
climate
cool
```

6. Stemming

- ✓ General words are,
  - ‘calculator’,
  - ‘calculating’,
  - ‘Calculation’.
- ✓ We know that these words are not very distinct from each other.
- ✓ It can be achieved through stemming.
- ✓ Stemming means reducing a word to its ‘root form’.
- ✓ We can implement this by using,
  - PorterStemmer predefined class

Program Name: Stemming example
demo8.py

```
from nltk.stem import PorterStemmer

ps = PorterStemmer()

words = ['dance ','dances', 'dancing ','danced']

for word in words:
        print(word,'------->', ps.stem(word))
```

Output

```
dance  -------> dance
dances -------> danc
dancing  -------> dancing
danced -------> danc
```

Note

- ✓ You can observe that some words were reduced to the base stems ‘danc’
- ✓ But, the word ‘danc’ is not semantically correct.
- ✓ Stemming may give us root words that are not present in the dictionary.
- ✓ How to overcome this?
  - That’s where Lemmatization comes to rescue

7. Lemmatization

- ✓ It is similar to stemming, except that the root word is correct and always meaningful.
- ✓ There are different ways to perform lemmatization.
- ✓ We will learn lemmatization by using nltk and spacy in below examples.

8. Lemmatization using spacy

- ✓ In spacy, the token object has an attribute .lemma_ which allows you to access the lemmatized version of that token.

Program Name: Lemmatization using spacy
demo9.py

```
import spacy
nlp = spacy.load('en_core_web_sm')

text = 'dance dances dancing danced'
text_doc = nlp(text)

for token in text_doc:
        print(token.text,'-------', token.lemma_)
```

Output

```
dance ------- dance
dances ------- dance
dancing ------- dance
danced ------- dance
```

Note

- ✓ Here, all words are reduced to ‘dance’ which is meaningful and just as required.
- ✓ It is highly preferred over stemming.

9. Word Frequency Analysis

- ✓ Once we removed stop words and applied lemmatization then the next important step is, we need analyse for further information about the text data.
- ✓ If any specific word repeated more times then we need to focus on that well.
- ✓ So, the words which occur more frequently those are very key concepts
- ✓ So, we need to store all tokens with their frequencies separately to analyse well

10. Counter predefined class

- ✓ We can use Counter to get the frequency of each token.
- ✓ If you provide a list to the Counter it returns a dictionary of all elements with their frequency as values.

Program Name: Counter example
demo10.py

```
import collections
from collections import Counter

import spacy
nlp = spacy.load('en_core_web_sm')


data = 'It is my birthday today. I could not have a birthday party. I felt sad'
data_doc = nlp(data)

list_of_tokens = [token.text for token in data_doc if not token.is_stop and not token.is_punct]

token_frequency = Counter(list_of_tokens)
print(token_frequency)
```

Output

```
Counter({'birthday': 2, 'today': 1, 'party': 1, 'felt': 1, 'sad': 1})
```

Note

- ✓ From above output, the word ‘birthday’ is most common.
- ✓ So, it gives us an idea that the text is about a birthday.
- ✓ Let us try with another example with more larger data
- ✓ For example,
  - o If we have a text data about a particular place, and you want to know the important factors.

Program Name: Counter example on large text
demo11.py

```
import collections
from collections import Counter

import spacy
nlp = spacy.load('en_core_web_sm')

longtext = 'Gangtok is a city, municipality, the capital and the largest town of
the Indian state of Sikkim. It is also the headquarters of the East Sikkim
district. Gangtok is in the eastern Himalayan range, at an elevation of 1,650.
The towns population of 100000 are from different ethnicities such as Bhutia,
Lepchas and Indian Gorkhas. Within the higher peaks of the Himalaya and
with a year-round mild temperate climate, Gangtok is at the centre of Sikkims
tourism industry. Gangtok rose to prominence as a popular Buddhist
pilgrimage site after the construction of the Enchey Monastery in 1840. In
1894, the ruling Sikkimese Chogyal, Thutob Namgyal, transferred the capital
to Gangtok. In the early 20th century, Gangtok became a major stopover on
the trade route between Lhasa in Tibet and cities such as Kolkata (then
Calcutta) in British India. After India won its independence from Britain in
1947, Sikkim chose to remain an independent monarchy, with Gangtok as its
capital. In 1975, after the integration with the union of India, Gangtok was
made Indias 22nd state capital. Like the rest of Sikkim, not much is known
about the early history of Gangtok. The earliest records date from the
construction of the hermitic Gangtok monastery in 1716.[7] Gangtok
remained a small hamlet until the construction of the Enchey Monastery in
1840 made it a pilgrimage center. It became the capital of what was left of
Sikkim after an English conquest in the mid-19th century in response to a
hostage crisis. After the defeat of the Tibetans by the British, Gangtok
became a major stopover in the trade between Tibet and British India at the
end of the 19th century. Most of the roads and the telegraph in the area
were built during this time. In 1894, Thutob Namgyal, the Sikkimese monarch
under British rule, shifted the capital from Tumlong to Gangtok, increasing
the citys importance. A new grand palace along with other state buildings
was built in the new capital. Following India independence in 1947, Sikkim
became a nation-state with Gangtok as its capital. Sikkim came under the
suzerainty of India, with the condition that it would retain its independence,
by the treaty signed between the Chogyal and the then Indian Prime Minister
Jawaharlal Nehru.[9] This pact gave the Indians control of external affairs on
behalf of Sikkimese. Trade between India and Tibet continued to flourish
through the Nathula and Jelepla passes, offshoots of the ancient Silk Road
near Gangtok. These border passes were sealed after the Sino-Indian War in
1962, which deprived Gangtok of its trading business.[10] The Nathula pass
```

```
was finally opened for limited trade in 2006, fuelling hopes of economic
boomIn 1975, after years of political uncertainty and struggle, including riots,
the monarchy was abrogated and Sikkim became India twenty-second state,
with Gangtok as its capital after a referendum. Gangtok has witnessed annual
landslides, resulting in loss of life and damage to property. The largest
disaster occurred in June 1997, when 38 were killed and hundreds of
buildings were destroyed'

long_text = nlp(longtext)

list_of_tokens = [token.text for token in long_text if not token.is_stop and
not token.is_punct]

token_frequency = Counter(list_of_tokens)
print(token_frequency)
```

Output

```
Counter({'Gangtok': 18, 'capital': 9, 'Sikkim': 8, 'India': 8, 'state': 5, 'Indian': 4,
'British': 4, 'construction': 3, 'Sikkimese': 3, 'century': 3, 'trade': 3, 'Tibet': 3,
'independence': 3, 'largest': 2, 'pilgrimage': 2, 'Enchey': 2, 'Monastery': 2,
'1840': 2, '1894': 2, 'Chogyal': 2, 'Thutob': 2, 'Namgyal': 2, 'early': 2, 'major': 2,
'stopover': 2, '1947': 2, 'monarchy': 2, '1975': 2, 'built': 2, 'new': 2, 'buildings':
2, 'Nathula': 2, 'passes': 2, 'city': 1, 'municipality': 1, 'town': 1, 'headquarters':
1, 'East': 1, 'district': 1, 'eastern': 1, 'Himalayan': 1, 'range': 1, 'elevation': 1,
'1,650': 1, 'towns': 1, 'population': 1, '100000': 1, 'different': 1, 'ethnicities': 1,
'Bhutia': 1, 'Lepchas': 1, 'Gorkhas': 1, 'higher': 1, 'peaks': 1, 'Himalaya': 1,
'year': 1, 'round': 1, 'mild': 1, 'temperate': 1, 'climate': 1, 'centre': 1, 'Sikkims':
1, 'tourism': 1, 'industry': 1, 'rose': 1, 'prominence': 1, 'popular': 1, 'Buddhist':
1, 'site': 1, 'ruling': 1, 'transferred': 1, '20th': 1, 'route': 1, 'Lhasa': 1, 'cities': 1,
'Kolkata': 1, 'Calcutta': 1, 'won': 1, 'Britain': 1, 'chose': 1, 'remain': 1,
'independent': 1, 'integration': 1, 'union': 1, 'Indias': 1, '22nd': 1, 'Like': 1,
'rest': 1, 'known': 1, 'history': 1, 'earliest': 1, 'records': 1, 'date': 1, 'hermitic':
1, 'monastery': 1, '1716.[7': 1, 'remained': 1, 'small': 1, 'hamlet': 1, 'center': 1,
'left': 1, 'English': 1, 'conquest': 1, 'mid-19th': 1, 'response': 1, 'hostage': 1,
'crisis': 1, 'defeat': 1, 'Tibetans': 1, 'end': 1, '19th': 1, 'roads': 1, 'telegraph': 1,
'area': 1, 'time': 1, 'monarch': 1, 'rule': 1, 'shifted': 1, 'Tumlong': 1, 'increasing':
1, 'citys': 1, 'importance': 1, 'grand': 1, 'palace': 1, 'Following': 1, 'nation': 1,
'came': 1, 'suzerainty': 1, 'condition': 1, 'retain': 1, 'treaty': 1, 'signed': 1,
'Prime': 1, 'Minister': 1, 'Jawaharlal': 1, 'Nehru.[9': 1, 'pact': 1, 'gave': 1,
'Indians': 1, 'control': 1, 'external': 1, 'affairs': 1, 'behalf': 1, 'Trade': 1,
'continued': 1, 'flourish': 1, 'Jelepla': 1, 'offshoots': 1, 'ancient': 1, 'Silk': 1,
'Road': 1, 'near': 1, 'border': 1, 'sealed': 1, 'Sino': 1, 'War': 1, '1962': 1,
'deprived': 1, 'trading': 1, 'business.[10': 1, 'pass': 1, 'finally': 1, 'opened': 1,
'limited': 1, '2006': 1, 'fuelling': 1, 'hopes': 1, 'economic': 1, 'boomIn': 1,
'years': 1, 'political': 1, 'uncertainty': 1, 'struggle': 1, 'including': 1, 'riots': 1,
```

```
'abrogated': 1, 'second': 1, 'referendum': 1, 'witnessed': 1, 'annual': 1,
'landslides': 1, 'resulting': 1, 'loss': 1, 'life': 1, 'damage': 1, 'property': 1,
'disaster': 1, 'occurred': 1, 'June': 1, '1997': 1, '38': 1, 'killed': 1, 'hundreds': 1,
'destroyed': 1})
```

Note

- ✓ As you can see, as the length or size of text data increases, it is difficult to analyse frequency of all tokens.
- ✓ So, you can print the n most common tokens using most_common function of Counter.
- ✓ see the below example on how to print the 6 most frequent words of the text

Program Name: Most common words from text
demo12.py

```
import collections
from collections import Counter

import spacy
nlp = spacy.load('en_core_web_sm')

longtext = 'Gangtok is a city, municipality, the capital and the largest town of
the Indian state of Sikkim. It is also the headquarters of the East Sikkim
district. Gangtok is in the eastern Himalayan range, at an elevation of 1,650.
The towns population of 100000 are from different ethnicities such as Bhutia,
Lepchas and Indian Gorkhas. Within the higher peaks of the Himalaya and
with a year-round mild temperate climate, Gangtok is at the centre of Sikkims
tourism industry. Gangtok rose to prominence as a popular Buddhist
pilgrimage site after the construction of the Enchey Monastery in 1840. In
1894, the ruling Sikkimese Chogyal, Thutob Namgyal, transferred the capital
to Gangtok. In the early 20th century, Gangtok became a major stopover on
the trade route between Lhasa in Tibet and cities such as Kolkata (then
Calcutta) in British India. After India won its independence from Britain in
1947, Sikkim chose to remain an independent monarchy, with Gangtok as its
capital. In 1975, after the integration with the union of India, Gangtok was
made Indias 22nd state capital. Like the rest of Sikkim, not much is known
about the early history of Gangtok. The earliest records date from the
construction of the hermitic Gangtok monastery in 1716.[7] Gangtok
remained a small hamlet until the construction of the Enchey Monastery in
1840 made it a pilgrimage center. It became the capital of what was left of
Sikkim after an English conquest in the mid-19th century in response to a
hostage crisis. After the defeat of the Tibetans by the British, Gangtok
became a major stopover in the trade between Tibet and British India at the
end of the 19th century. Most of the roads and the telegraph in the area
were built during this time. In 1894, Thutob Namgyal, the Sikkimese monarch
under British rule, shifted the capital from Tumlong to Gangtok, increasing
the citys importance. A new grand palace along with other state buildings
was built in the new capital. Following India independence in 1947, Sikkim
became a nation-state with Gangtok as its capital. Sikkim came under the
suzerainty of India, with the condition that it would retain its independence,
by the treaty signed between the Chogyal and the then Indian Prime Minister
Jawaharlal Nehru.[9] This pact gave the Indians control of external affairs on
behalf of Sikkimese. Trade between India and Tibet continued to flourish
through the Nathula and Jelepla passes, offshoots of the ancient Silk Road
near Gangtok. These border passes were sealed after the Sino-Indian War in
1962, which deprived Gangtok of its trading business.[10] The Nathula pass
was finally opened for limited trade in 2006, fuelling hopes of economic
```

```
boomIn 1975, after years of political uncertainty and struggle, including riots,
the monarchy was abrogated and Sikkim became India twenty-second state,
with Gangtok as its capital after a referendum. Gangtok has witnessed annual
landslides, resulting in loss of life and damage to property. The largest
disaster occurred in June 1997, when 38 were killed and hundreds of
buildings were destroyed'

long_text = nlp(longtext)

list_of_tokens = [
        token.text
        for token in long_text
        if not token.is_stop and not token.is_punct
]

token_frequency = Counter(list_of_tokens)

most_frequent_tokens = token_frequency.most_common(6)
print(most_frequent_tokens)
```

Output

```
[('Gangtok', 18), ('capital', 9), ('Sikkim', 8), ('India', 8), ('state', 5), ('Indian', 4)]
```

Note

- ✓ We can see that the keywords are gangtok , sikkkim, Indian and so on.
- ✓ It gives us an idea of the text to start with.

11. Part of Speech Tagging

- ✓ Each word has its own role in a sentence.
- ✓ For example,
  - ’Daniel is dancing’.
  - Daniel is the person or ‘Noun’ and dancing is the action performed by him.
  - So it is a ‘Verb’.
  - Likewise, each word can be classified.
  - This is referred as POS or Part of Speech Tagging.

12. Parts of Speech

- ✓ Noun
- ✓ Verb
- ✓ Adjective
- ✓ Pronoun
- ✓ Adverb
- ✓ Preposition
- ✓ Conjunction
- ✓ Interjection and so on.

13. Pos by using spacy

- ✓ POS can be implemented using both spaCy and nltk.
- ✓ First, let us see the method using spaCy
- ✓ In spaCy, the POS tags are present in the attribute of Token object.
- ✓ We can access the POS tag of particular token through the token.pos_ attribute.

Program Name: Pos example
demo13.py

```
import spacy
nlp = spacy.load('en_core_web_sm')

text = 'Daniel is singing loudly and his roommates are enjoying too'
text_doc = nlp(text)

for word in text_doc:
        print(word.text.ljust(10), '-----', word.pos_)
```

Output

```
Nireekshan ----- PROPN
is         ----- AUX
singing    ----- VERB
loudly     ----- ADV
and        ----- CCONJ
his        ----- DET
roommates  ----- NOUN
are        ----- AUX
enjoying   ----- VERB
too        ----- ADV
```

covid.txt

Even as the UK became the first country to grant emergency approval to Pfizer-BioNTech's Covid-19 vaccine, the US firm said it is "committed" to engaging with the Indian government to "explore opportunities" to roll it out here. The vaccine will be a challenge for India as it requires storage at -70 degrees, experts say.

Pfizer spokeswoman Roma Nair told TOI: "We are committed to advance our dialogue with the Indian government. We are working with governments across the world to understand the infrastructure requirements of each country and we have logistical plans in place. We are confident the rollout can be managed in India."

Program Name: Preprocessing the text by using spacy
demo14.py

```
import spacy
nlp = spacy.load('en_core_web_sm')

with open('covid.txt') as file:
        robot_text = file.read()

robot_doc = nlp(robot_text)

robot_doc = [
        word
        for word in robot_doc
        if not word.is_stop and not word.is_punct
]

print(robot_doc)
```

Output

[UK, country, grant, emergency, approval, Pfizer, BioNTech, Covid-19, vaccine, firm, said, committed, engaging, Indian, government, explore, opportunities, roll, vaccine, challenge, India, requires, storage, -70, degrees, experts,
, Pfizer, spokeswoman, Roma, Nair, told, TOI, committed, advance, dialogue, Indian, government, working, governments, world, understand, infrastructure, requirements, country, logistical, plans, place, confident, rollout, managed, India]

Program Name: List of nouns and verbs
demo15.py

```
import spacy
nlp = spacy.load('en_core_web_sm')

with open('covid.txt') as file:
        robot_text = file.read()

robot_doc = nlp(robot_text)

robot_doc = [
        word
        for word in robot_doc
        if not word.is_stop and not word.is_punct
]

nouns = []
verbs = []

for word in robot_doc:
        if word.pos_ == 'NOUN':
                nouns.append(word)
        if word.pos_ == 'VERB':
                verbs.append(word)

print('List of Nouns in the text:', nouns)
print()
print('List of verbs in the text:', verbs)
```

Output

```
List of Nouns in the text:
[country, emergency, approval, vaccine, firm, government, opportunities,
vaccine, challenge, storage, degrees, experts, spokeswoman, dialogue,
government, governments, world, infrastructure, requirements, country,
plans, place, rollout]

List of verbs in the text:
[grant, said, committed, engaging, explore, roll, requires, told, advance,
working, understand, managed]
```

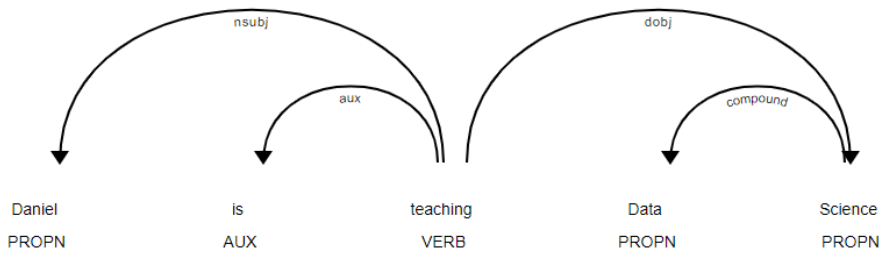
Program Name

We can use displacy function for display.
demo16.py

```
from spacy import displacy

import spacy
nlp = spacy.load('en_core_web_sm')

text = "Daniel is teaching Data Science"

doc = nlp(text)
displacy.render(doc)
```

Note

Execute above example in jupyter to see output image



14. Named Entity Recognition

Named Entity Reorganization

- ✓ Suppose you have a collection of news articles text data.
- ✓ From these we can capture,
  - companies/organization names
  - People names
- ✓ By using NER topic, we can do this

15. What is NER?

- ✓ NER is the technique of identifying named entities in the text corpus and assigning them pre-defined categories such as ' person names' , ' locations' ,'organizations', etc..
- ✓ It is a very useful method especially in the field of classification problems and search engine optimizations.

16. NER by using nltk and spacy

- ✓ NER can be implemented through both nltk and spacy'.

17. NER with spacy

- ✓ Spacy is highly flexible and advanced library.
- ✓ Named entity recognition through spacy is easy.

18. The few common labels are:

- ✓ 'ORG' : companies, organizations, etc.
- ✓ 'PERSON' : names of people
- ✓ 'GPE' : countries, states, etc.
- ✓ 'PRODUCT' : vehicles, food products and so on
- ✓ 'LANGUAGE' : Names of different languages

- ✓ There are many other labels too.

Program Name: Print all the named entities
demo20.py

```
import spacy
nlp = spacy.load('en_core_web_sm')

sentence=' The building is located at London. It is the headquarters of
Google. Mark works there. He speaks English'

doc=nlp(sentence)

for entity in doc.ents:
        print(entity.text,'--', entity.label_)
```

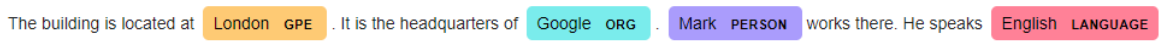
Output

```
London -- GPE
Google -- ORG
Mark -- PERSON
English -- LANGUAGE
```

Program Name: Visualize all the named entities
demo21.py

```
import spacy

nlp = spacy.load('en_core_web_sm')

sentence=' The building is located at London. It is the headquarters of Google. Mark works there. He speaks English'

doc=nlp(sentence)

displacy.render(doc, style='ent', jupyter=True)
```

Output



Use case - 1

- ✓ Let us take the text data of collection of news headlines.
- ✓ Can we get list of all names that have occurred in the news?

Program Name: Display only person names from the text
demo22.py

```
from spacy import displacy
import spacy

nlp = spacy.load('en_core_web_sm')

news_articles = 'Honoured to serve India, Naredra Modi, 68, wrote in a
Twitter post that popped up on the micro blogging site as the ceremony
started at 7 p.m. before an audience of 8,000 people, who included United
Progressive Alliance chairperson Sonia Gandhi and her son and Congress
president Rahul Gandhi, both seated on the front row. Shah, 54, whose
inclusion in the cabinet had been much speculated upon, was administered
the oaths after Naredra Modi and Rajnath Singh, indicating the order of
seniority in the new government , which took office exactly a week after
results from the 17th general elections were declared'

news_doc=nlp(news_articles)

list_of_people=[]

for token in news_doc:
        if token.ent_type_=='PERSON':
                list_of_people.append(token.text)

print(list_of_people)
```

Output

```
['Sonia', 'Gandhi', 'Rahul', 'Gandhi', 'Naredra', 'Modi', 'Rajnath', 'Singh']
```

E-Commerce Purchase Intention Model

Contents

E-Commerce Purchase Intention Model

1. E-Commerce application goal

- ✓ Any eCommerce application main goal is,
  - Converting browsers into buyers.

2. Purchase intention model

- ✓ Ecommerce purchase intention models predict the probability of each customer making a purchase or not.
- ✓ Once we identify then we can target those customers.
- ✓ We will learn here how they work and build model
- ✓ Ecommerce purchase intention models analyse click-stream consumer behaviour data from web analytics platforms.
- ✓ After analysis it can predict whether a customer will make a purchase during their visit.
- ✓ These online shopping models are used to examine real time web analytics data and predict the probability of each customer making a purchase, so the retailer can serve a carefully targeted promotion to try and persuade those less likely to purchase.

3. Google Ads

- ✓ Generally while browsing we can see some Google ads
- ✓ Surprisingly person to person these ads are different.
- ✓ The point is, Google Company implemented Analytical techniques to target customer by using Google Ads.

4. Work flow of ecommerce purchase intention models

- ✓ The problem statement is, customer will BUY product or NOT
- ✓ So, this is classification.
- ✓ E-Commerce purchase intention models are classifiers.
- ✓ These models designed to analyse web analytics data and predict whether a customer will buy product or not during their visit.
- ✓ These things needs to be monitored,
    - Number of times URLs visited
    - Information about product
    - Recorded the number of each page type visited
    - Time spent on the pages.

5. Important features from the dataset

- ✓ Below features are very important from the dataset.
- ✓ From those features we can apply feature engineering and creating new features.
- ✓ Based on our requirement few of features we can convert them into numeric.
- ✓ We need to identify the correlation of the target variable.

Feature	Description	Type
Day	Measures the closeness of the visit date to a key trading event.	Numerical
Operating system	The operating system used during the visit.	Categorical
Browser	The web browser used during the visit.	Categorical
Region	The geographic region of the visitor.	Categorical
Visitor type	Whether the customer was a new visitor or returning visitor.	Categorical
Weekend	A Boolean value indicating whether the visit fell on a weekend.	Categorical
Month	The month of the user's visit.	Categorical
Revenue	The target variable indicating whether the visit generated revenue.	Categorical

6. Install packages

```
pip install lightgbm
pip install imblearn
```

7. Loading dataset using pandas

Program Name: Loading the dataset
File: demo1.py
File: online_shoppers_intention.csv

```
import pandas as pd

df = pd.read_csv("online_shoppers_intention.csv")

print(df.head())
```

Output

```
   Administrative  Administrative_Duration  Informational  ...        VisitorType  Weekend  Revenue
0               0                      0.0              0  ...  Returning_Visitor    False    False
1               0                      0.0              0  ...  Returning_Visitor    False    False
2               0                      0.0              0  ...  Returning_Visitor    False    False
3               0                      0.0              0  ...  Returning_Visitor    False    False
4               0                      0.0              0  ...  Returning_Visitor     True    False

[5 rows x 18 columns]
```

8. Number of rows and columns

Program Name: Checking number of rows and column
File: demo2.py
File: online_shoppers_intention.csv

```
import pandas as pd

df = pd.read_csv("online_shoppers_intention.csv")

print(df.shape)
```

Output

```
(12330, 18)
```

9. Dataset information

Program Name: Checking Dataset information

File: demo3.py

File: online_shoppers_intention.csv

```
import pandas as pd

df = pd.read_csv("online_shoppers_intention.csv")

print(df.info())
```

Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           12330 non-null  int64
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  object
 16  Weekend                  12330 non-null  bool
 17  Revenue                  12330 non-null  bool
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

10. Feature engineering

- ✓ The Weekend and Revenue columns are currently set to Boolean values, so we first need to convert into binary values.

10.1. replace() method

- ✓ replace() is predefined method in Series class.
- ✓ We should access this method by using series object.
- ✓ This method replace existing values with desired values.

10.2. Weekend column

Program Name: Checking unique values in Weekend column
File: demo4.py
File: online_shoppers_intention.csv

```
import pandas as pd

df = pd.read_csv("online_shoppers_intention.csv")

print(df['Weekend'].unique())
```

Output

```
[False  True]
```

Program Name: Convert Weekend column to binary values

File: demo5.py

File: online_shoppers_intention.csv

```
import pandas as pd

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))

print(df.head())
```

Output

```
   Administrative  Administrative_Duration  Informational  ...        VisitorType  Weekend  Revenue
0               0                      0.0              0  ...  Returning_Visitor        0    False
1               0                      0.0              0  ...  Returning_Visitor        0    False
2               0                      0.0              0  ...  Returning_Visitor        0    False
3               0                      0.0              0  ...  Returning_Visitor        0    False
4               0                      0.0              0  ...  Returning_Visitor        1    False

[5 rows x 18 columns]
```

10.3. Revenue column

Program Name: Checking unique values in Revenue column
File: demo6.py
File: online_shoppers_intention.csv

```
import pandas as pd

df = pd.read_csv("online_shoppers_intention.csv")

print(df['Revenue'].unique())
```

Output

```
[False  True]
```

Program Name: Convert Revenue column to binary values

File: demo7.py

File: online_shoppers_intention.csv

```
import pandas as pd

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

print(df.head())
```

Output

```
   Administrative  Administrative_Duration  Informational  ...        VisitorType  Weekend  Revenue
0               0                      0.0              0  ...  Returning_Visitor        0        0
1               0                      0.0              0  ...  Returning_Visitor        0        0
2               0                      0.0              0  ...  Returning_Visitor        0        0
3               0                      0.0              0  ...  Returning_Visitor        0        0
4               0                      0.0              0  ...  Returning_Visitor        1        0

[5 rows x 18 columns]
```

10.4. Let's understand VisitorType column

- ✓ VisitorType contains either Returning_Visitor or New_Visitor.
- ✓ For us one value is enough because other value is opposite to existing value.

Program Name: Checking unique values in VisitorType column
File: demo8.py
File: online_shoppers_intention.csv

```
import pandas as pd

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

print(df['VisitorType'].unique())
```

Output

```
['Returning_Visitor' 'New_Visitor' 'Other']
```

10.5. VisitorType column

- ✓ VisitorType contains either Returning_Visitor or New_Visitor.
- ✓ For us one value is enough because other value is opposite to existing value.
- ✓ Let's add Returning_Visitor column to existing dataframe

Program — Adding Returning_Visitor column and Convert VisitorType column to binary values

Name — demo9.py

File — online_shoppers_intention.csv

```
import pandas as pd
import numpy as np

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

print(df.head())
```

Output

```
   Administrative  Administrative_Duration  Informational  ...  Weekend  Revenue  Returning_Visitor
0               0                      0.0              0  ...        0        0                  1
1               0                      0.0              0  ...        0        0                  1
2               0                      0.0              0  ...        0        0                  1
3               0                      0.0              0  ...        0        0                  1
4               0                      0.0              0  ...        1        0                  1

[5 rows x 19 columns]
```

Program Name: Drop VisitorType column
File: demo10.py
File: online_shoppers_intention.csv

```
import pandas as pd
import numpy as np

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

print(df.head())
```

Output

```
   Administrative  Administrative_Duration  Informational  ...  Weekend  Revenue  Returning_Visitor
0               0                      0.0              0  ...        0        0                  1
1               0                      0.0              0  ...        0        0                  1
2               0                      0.0              0  ...        0        0                  1
3               0                      0.0              0  ...        0        0                  1
4               0                      0.0              0  ...        1        0                  1

[5 rows x 19 columns]
```

10.6. Month column

- ✓ We do have Month column name in DataFrame.
- ✓ By default this column type recognised as object means string type
- ✓ Lest apply Ordinal Encoding over this column.

Program Name: Checking all columns data type
File: demo11.py
File: online_shoppers_intention.csv

```
import pandas as pd
import numpy as np

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

print(df.dtypes)
```

Output

```
Administrative             int64
Administrative_Duration    float64
Informational              int64
Informational_Duration     float64
ProductRelated             int64
ProductRelated_Duration    float64
BounceRates                float64
ExitRates                  float64
PageValues                 float64
SpecialDay                 float64
Month                      object
OperatingSystems           int64
Browser                    int64
Region                     int64
TrafficType                int64
Weekend                    int64
Revenue                    int64
Returning_Visitor          int32
dtype: object
```

Program Name: Checking unique values in Month column
File: demo12.py
File: online_shoppers_intention.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder


df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

print(df['Month'].unique())
```

Output

```
['Feb' 'Mar' 'May' 'Oct' 'June' 'Jul' 'Aug' 'Nov' 'Sep' 'Dec']
```

Program Name: Applying Ordinal Encoding on Month column
File: demo13.py
File: online_shoppers_intention.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder


df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

ordinal_encoder = OrdinalEncoder()

df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

print(df['Month'].unique())
```

Output

```
[2. 5. 6. 8. 4. 3. 0. 7. 9. 1.]
```

11. Target variable is Revenue column

- ✓ **Revenue column is the target variable**
- ✓ Let's check the value_counts() on this column.

Program Name: Let's understand the Revenue column
File: demo14.py
File: online_shoppers_intention.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder


df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

ordinal_encoder = OrdinalEncoder()

df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

print(df.Revenue.value_counts())
```

Output

```
Revenue
0    10422
1     1908
Name: count, dtype: int64
```

12. Pearson correlation

- ✓ We can check which of the features are correlated with the target variable by using Pearson correlation.

13. PageValues column

- ✓ The strongest predictor of conversion was the PageValues column.
- ✓ This column contained the Page Value metric.
- ✓ This is obviously higher for customers who have viewed product, basket, and checkout pages.
- ✓ So it makes total sense that it plays a significant role.

14. How to identify customers interest

- ✓ Customers who have viewed more product pages and spent longer looking at them were also much more likely to have purchased.
- ✓ Customers who shopped using specific browsers.
- ✓ Specific days shopping like weekday or weekend etc

Program Name: Access required columns
File: demo15.py
File: online_shoppers_intention.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder


df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

ordinal_encoder = OrdinalEncoder()

df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

result = df.columns[1:]

print(result)
```

Output

```
Index(['Administrative_Duration', 'Informational', 'Informational_Duration',
       'ProductRelated', 'ProductRelated_Duration', 'BounceRates', 'ExitRates',
       'PageValues', 'SpecialDay', 'Month', 'OperatingSystems', 'Browser',
       'Region', 'TrafficType', 'Weekend', 'Revenue', 'Returning_Visitor'],
      dtype='object')
```

Program Name: Create a DataFrame with required columns

File: demo16.py

File: online_shoppers_intention.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder


df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

ordinal_encoder = OrdinalEncoder()

df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

result = df[df.columns[1:]]

print(result)
```

Output

```
       Administrative_Duration  Informational  ...  Revenue  Returning_Visitor
0                          0.0              0  ...        0                  1
1                          0.0              0  ...        0                  1
2                          0.0              0  ...        0                  1
3                          0.0              0  ...        0                  1
4                          0.0              0  ...        0                  1
...                        ...            ...  ...      ...                ...
12325                    145.0              0  ...        0                  1
12326                      0.0              0  ...        0                  1
12327                      0.0              0  ...        0                  1
12328                     75.0              0  ...        0                  1
12329                      0.0              0  ...        0                  0

[12330 rows x 17 columns]
```

Program Name: Correlation
File: demo17.py
online_shoppers_intention.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder


df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

ordinal_encoder = OrdinalEncoder()

df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

result = df[df.columns[1:]].corr()

print(result)
```

Output

	Administrative_Duration	Informational	...	Revenue	Returning_Visitor
Administrative_Duration	1.000000	0.302710	...	0.093587	-0.022525
Informational	0.302710	1.000000	...	0.095200	0.057399
Informational_Duration	0.238031	0.618955	...	0.070345	0.045501
ProductRelated	0.289087	0.374164	...	0.158538	0.128738
ProductRelated_Duration	0.355422	0.387505	...	0.152373	0.120489
BounceRates	-0.144170	-0.116114	...	-0.150673	0.129908
ExitRates	-0.205798	-0.163666	...	-0.207071	0.171987
PageValues	0.067608	0.048632	...	0.492569	-0.115825
SpecialDay	-0.073304	-0.048219	...	-0.082305	0.087123
Month	0.029061	0.019743	...	0.080150	0.036689
OperatingSystems	-0.007343	-0.009527	...	-0.014668	-0.038345
Browser	-0.015392	-0.038235	...	0.023984	-0.058836
Region	-0.005561	-0.029169	...	-0.011595	-0.050829
TrafficType	-0.014376	-0.034491	...	-0.005113	-0.026219
Weekend	0.014990	0.035785	...	0.029295	-0.039444
Revenue	0.093587	0.095200	...	1.000000	-0.103843
Returning_Visitor	-0.022525	0.057399	...	-0.103843	1.000000

[17 rows x 17 columns]

Program Name: Correlation
File: demo18.py
online_shoppers_intention.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder


df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

ordinal_encoder = OrdinalEncoder()

df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

result = df[df.columns[1:]].corr()['Revenue']
print(result)
```

Output

```
Administrative_Duration    0.093587
Informational              0.095200
Informational_Duration     0.070345
ProductRelated             0.158538
ProductRelated_Duration    0.152373
BounceRates               -0.150673
ExitRates                 -0.207071
PageValues                 0.492569
SpecialDay                -0.082305
Month                      0.080150
OperatingSystems          -0.014668
Browser                    0.023984
Region                    -0.011595
TrafficType               -0.005113
Weekend                    0.029295
Revenue                    1.000000
Returning_Visitor         -0.103843
Name: Revenue, dtype: float64
```

Program Name: Correlation
File: demo19.py
online_shoppers_intention.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder


df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

ordinal_encoder = OrdinalEncoder()

df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

result = df[df.columns[1:]].corr()['Revenue']
result1 = result.sort_values(ascending=False)

print(result1)
```

Output

```
Revenue                    1.000000
PageValues                 0.492569
ProductRelated             0.158538
ProductRelated_Duration    0.152373
Informational              0.095200
Administrative_Duration    0.093587
Month                      0.080150
Informational_Duration     0.070345
Weekend                    0.029295
Browser                    0.023984
TrafficType               -0.005113
Region                    -0.011595
OperatingSystems          -0.014668
SpecialDay                -0.082305
Returning_Visitor         -0.103843
BounceRates               -0.150673
ExitRates                 -0.207071
Name: Revenue, dtype: float64
```

15. Create the training and test data

- ✓ Now we need to prepare the feature set, we need to define X and y.
- ✓ The X feature set will include all the features we created above
- ✓ The y feature having target variable alone.

Program Name: Create features and target

File: demo20.py

File: online_shoppers_intention.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder


df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

ordinal_encoder = OrdinalEncoder()

df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

result = df[df.columns[1:]].corr()['Revenue']
result1 = result.sort_values(ascending=False)

X = df.drop(['Revenue'], axis=1)
y = df['Revenue']

print("Features and target created")
```

Output

```
Features and target created
```

16. Train and Test data

- ✓ We need to split(randomly) data into training(70%) and testing(30%) datasets.
- ✓ We can split data by using **train_test_split**(X, y, test_size = 0.3, random_state) function.
- ✓ The random_state value ensures we get reproducible results each time we run the code.

Program Name: Creating train and test datasets
File: demo21.py
online_shoppers_intention.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

ordinal_encoder = OrdinalEncoder()

df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

result = df[df.columns[1:]].corr()['Revenue']
result1 = result.sort_values(ascending=False)

X = df.drop(['Revenue'], axis=1)
y = df['Revenue']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.3, random_state = 0)

print("Created train and test datasets")
```

Output

```
Created train and test datasets
```

17. A Machine Learning pipeline

- ✓ This is the process of automating the workflow of a complete machine learning task.
- ✓ This is having group of steps like data to be transformed and correlated together in a model that can be analysed to get the output.
- ✓ A typical pipeline includes raw data input, features, outputs, model parameters, ML models, and Predictions.
- ✓ Pipeline also having sequential steps that perform everything like data extraction and pre-processing to model training and deployment in Machine learning in a modular approach.
- ✓ It means that in the pipeline, each step is designed as an independent module, and all these modules are tied together to get the final result.

18. Create a model pipeline

- ✓ Now we need to create a model pipeline for our project
- ✓ This Pipeline handles the encoding of data using the ColumnTransformer().
- ✓ By using this we can even avoid like manually generating some of the features we created above.
- ✓ This also imputes any missing values with realistic values.
- ✓ It scales the data before we pass it to the model.

18.1. SelectKBest and SMOTE

- ✓ By using SelectKBest() we can select the optimal features. From this we are getting around 6 features as per shown in Pearson correlation.
- ✓ By using SMOTE(Synthetic Minority Oversampling Technique) we can handle class imbalance.

18.2. model_pipeline(X, model) function explanation

- ✓ model_pipleline(X, model) is a user defined function for this project.
- ✓ This function returns pipeline to pre-process data and bundle with a model.
- ✓ There are two arguments for this function X means X_train data and model means model object Ex: XGBClassifier object.

18.3. Kind note over model pipeline

- ✓ Requesting kindly spend some time to understand this pipeline code.
- ✓ Don't worry I am happy to explain every piece of line.
    - **From Daniel**

Program Name: A function to create model pipeline
daniel_model_pipeline.py

```
def model_pipeline(X, model):

    n_c = X.select_dtypes(exclude=['object']).columns.tolist()
    c_c = X.select_dtypes(include=['object']).columns.tolist()


    numeric_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='constant')),
        ('scaler', MinMaxScaler())
    ])


    categorical_pipeline = Pipeline([
        ('encoder', OneHotEncoder(handle_unknown='ignore'))
    ])


    preprocessor = ColumnTransformer([
        ('numeric', numeric_pipeline, n_c),
        ('categorical', categorical_pipeline, c_c)
    ], remainder='passthrough')



    final_steps = [
        ('preprocessor', preprocessor),
        ('smote', SMOTE(random_state=1)),
        ('feature_selection', SelectKBest(score_func = chi2, k =
6)),
        ('model', model)
    ]

    return IMBPipeline(steps = final_steps)
```

19. Select best model

19.1. Ameerpet Standard

- ✓ Generally we can select single model to proceeding in further steps.
- ✓ Many times we use to run our code repeatedly by taking different models as well.
- ✓ Instead of checking every time with different models, we can create a function to select the best model

19.2. Hi-Tech City Standard

- ✓ It would be good to create one function to select the best model.
- ✓ This function automatically test all models and finally returns the best model as per our requirement
- ✓ In this function, in very first step we are creating dictionary with XGBoost, Random Forest, Decision Tree, SVC, and a Multilayer Perceptron among others.
- ✓ We can loop through these models, run the data through the pipeline for each model.
- ✓ Use cross-validation to get good performance, reliability and validity of each model.
- ✓ Store the model results in pandas DataFrame and print the output.
- ✓ Finally selecting the **BEST MODEL** with highest ROC/AUC score

19.3. Function explanation

- ✓ select_model(X, y, pipeline = None) is user defined function created by us.
- ✓ This function takes 2 non default parameters and 1 default parameter
  - X (object) : Pandas dataframe containing X_train data.
  - y (object) : Pandas dataframe containing y_train data.
  - pipeline : Pipeline from model_pipeline().

19.4. Kind note over select best model

- ✓ Requesting kindly spend some time to understand this select best model code.
- ✓ Don't worry I am happy to explain every piece of line.
  - **From Daniel**

Program Name: A function to select model
daniel_select_model.py

```
def select_model(X, y, pipeline=None):

    classifiers = {}

    c_d1 = {"DummyClassifier":
    DummyClassifier(strategy='most_frequent')}
    classifiers.update(c_d1)


    c_d2 = {"RandomForestClassifier":
    RandomForestClassifier()}
    classifiers.update(c_d2)


    c_d3 = {"DecisionTreeClassifier": DecisionTreeClassifier()}
    classifiers.update(c_d3)


    c_d4 = {"KNeighborsClassifier": KNeighborsClassifier()}
    classifiers.update(c_d4)

    c_d5 = {"SVC": SVC()}
    classifiers.update(c_d5)

    mlpc = {
    "MLPClassifier (paper)":
    MLPClassifier(hidden_layer_sizes=(27, 50),
    max_iter=300,
    activation='relu',
    solver='adam',
    random_state=1)
    }
```

```
c_d6 = mlpc
classifiers.update(c_d6)


cols = ['model', 'run_time', 'roc_auc']
df_models = pd.DataFrame(columns = cols)


for key in classifiers:
        start_time = time.time()
        pipeline = model_pipeline(X_train, classifiers[key])

        cv = cross_val_score(pipeline, X, y, cv=10,
        scoring='roc_auc')

        row = {'model': key,
        'run_time': format(round((time.time() -
        start_time)/60,2)),
        'roc_auc': cv.mean(),
        }

        df_models = df_models.append(row,
        ignore_index=True)

        df_models = df_models.sort_values(by='roc_auc',
        ascending = False)

return df_models
```

20. let's select_model function

- ✓ Once we call select_model function then we will get best model from all models.

Program Name: Access select_model function
File Name: access_select_model.py
File: online_shoppers_intention.csv

```
############################
# Importing required librarie #
############################

print("Step 1: Required librarie imported successfully")

import time
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

from imblearn.over_sampling import SMOTE

from imblearn.pipeline import Pipeline as imbpipeline
from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score

from sklearn.svm import SVC
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.dummy import DummyClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import ExtraTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.linear_model import RidgeClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.neural_network import MLPClassifier
```

##################

```
# To ignore warning #
####################

from warnings import simplefilter
from sklearn.exceptions import ConvergenceWarning
simplefilter("ignore", category=ConvergenceWarning)




##############################################
# Loading online_shoppers_intention.csv dataset #
##############################################

print("Step 2: Created DataFrame successfully")

df = pd.read_csv("online_shoppers_intention.csv")



####################
# Feature Engineering #
####################

print("Step 3: Feature Engineering Done successfully on Weekend,
Revenue")


df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))









################################
```

```
# Added Returning_Visitor column #
################################

print("Step 4: Added Returning_Visitor column successfully")

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns=['VisitorType'])



########################################
# Applying One Hot Encoding on Month column #
########################################

print("Step 5: Applied one hot encoding successfully on Month
column")


ordinal_encoder = OrdinalEncoder()
df['Month'] = ordinal_encoder.fit_transform(df[['Month']])



###################################
# Checking correlation on Revenue column #
###################################

print("Step 6: Checking correlation done successfully")


result = df[df.columns[1:]].corr()['Revenue']

result1 = result.sort_values(ascending=False)
```

```
####################################
# Preparing Features as X and target as y #
####################################

print("Step 7: Preparing features as X and target as y done
successfully")


X = df.drop(['Revenue'], axis=1)
y = df['Revenue']



#####################################
# Preparing Train and Test Dataset#
#####################################

print("Step 8: Splitting data X_train, X_test, y_train & y_test done
successfully")


X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state = 0)









###############
```

```
# Model Pipeline #
################

print("Step 9: model_pipeline function created done successfully")

def model_pipeline(X, model):

    n_c = X.select_dtypes(exclude=['object']).columns.tolist()
    c_c = X.select_dtypes(include=['object']).columns.tolist()


    numeric_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='constant')),
        ('scaler', MinMaxScaler())
    ])


    categorical_pipeline = Pipeline([
        ('encoder', OneHotEncoder(handle_unknown='ignore'))
    ])


    preprocessor = ColumnTransformer([
        ('numeric', numeric_pipeline, n_c),
        ('categorical', categorical_pipeline, c_c)
    ], remainder='passthrough')



    final_steps = [
        ('preprocessor', preprocessor),
        ('smote', SMOTE(random_state=1)),
        ('feature_selection', SelectKBest(score_func = chi2, k =
6)),
        ('model', model)
    ]

    return IMBPipeline(steps = final_steps)
```

```
################
# Model Selection #
################

print("Step 10: select_model function created done successfully")

def select_model(X, y, pipeline=None):

    classifiers = {}

    c_d1 = {"DummyClassifier":
    DummyClassifier(strategy='most_frequent')}
    classifiers.update(c_d1)

    c_d4 = {"RandomForestClassifier":
    RandomForestClassifier()}
    classifiers.update(c_d4)

    c_d5 = {"DecisionTreeClassifier": DecisionTreeClassifier()}
    classifiers.update(c_d5)

    c_d9 = {"KNeighborsClassifier": KNeighborsClassifier()}
    classifiers.update(c_d9)

    c_d10 = {"RidgeClassifier": RidgeClassifier()}
    classifiers.update(c_d10)

    c_d14 = {"SVC": SVC()}
    classifiers.update(c_d14)

    mlpc = {
    "MLPClassifier (paper)":
    MLPClassifier(hidden_layer_sizes=(27, 50),
    max_iter=300,
    activation='relu',
    solver='adam',
    random_state=1)
    }
```

```
c_d16 = mlpc
classifiers.update(c_d16)

cols = ['model', 'run_time', 'roc_auc']
df_models = pd.DataFrame(columns = cols)

for key in classifiers:
        start_time = time.time()
        print()
        print("Step 12: model_pipeline run successfully on",
        key)

        pipeline = model_pipeline(X_train, classifiers[key])

        cv = cross_val_score(pipeline, X, y, cv=10,
        scoring='roc_auc')

        row = {'model': key,
        'run_time': format(round((time.time() -
        start_time)/60,2)),
        'roc_auc': cv.mean(),
        }

        df_models = pd.concat([df_models,
        pd.DataFrame([row])], ignore_index=True)

        df_models = df_models.sort_values(by='roc_auc',
        ascending = False)

return df_models
```

```
##################################
# Access Model select_model function #
##################################

print("Step 11: Accessing select_model function done successfully")

models = select_model(X_train, y_train)



##############################
# Let's see total model with score #
##############################

print("Step 13: Accessing select_model function done successfully")

print(models)
```

Output

	model	run_time	roc_auc
14	MLPClassifier	1.79	0.903222
15	MLPClassifier (paper)	2.86	0.900244
2	LGBMClassifier	0.04	0.897217
1	XGBClassifier	0.15	0.891174
10	SGDClassifier	0.02	0.889067
7	AdaBoostClassifier	0.1	0.888264
13	SVC	0.83	0.885963
3	RandomForestClassifier	0.27	0.884997
11	BaggingClassifier	0.07	0.863183
12	BernoulliNB	0.01	0.857851
9	RidgeClassifier	0.01	0.855441
8	KNeighborsClassifier	0.02	0.840505
6	ExtraTreesClassifier	0.01	0.770749
5	ExtraTreeClassifier	0.01	0.754475
4	DecisionTreeClassifier	0.02	0.734040
0	DummyClassifier	0.01	0.500000

21. The best model is

- ✓ The top performer was MLPClassifier(), which generated a ROC/AUC score of 0.902.
- ✓ We'll select this model as our best one and examine the results in a bit more detail to see how well it works.

22. Examine the performance of the best model

- ✓ By re-running the model_pipeline() function on our selected model as the MLPClassifier()
- ✓ We can generate predictions and assess their accuracy on the test data.

23. Best model score

- ✓ What this shows is that the MLPClassifier model generated a ROC/AUC score of 0.902 on the training data, which is really good, but a slightly lower ROC/AUC score of 0.836 on the test data, with an overall accuracy of 0.88.

Program Name: Final code with best model and result
File: final_code.py
online_shoppers_intention.csv

```
##########################
# Importing required librarie #
##########################

print("Step 1: Required librarie imported successfully")

import time
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

from imblearn.over_sampling import SMOTE

from imblearn.pipeline import Pipeline as imbpipeline
from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score

from sklearn.svm import SVC
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.dummy import DummyClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import ExtraTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.linear_model import RidgeClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.neural_network import MLPClassifier
```

```
##################
# To ignore warning #
##################

from warnings import simplefilter
from sklearn.exceptions import ConvergenceWarning
simplefilter("ignore", category=ConvergenceWarning)



##########################################
# Loading online_shoppers_intention.csv dataset #
##########################################

print("Step 2: Created DataFrame successfully")

df = pd.read_csv("online_shoppers_intention.csv")



###################
# Feature Engineering #
###################

print("Step 3: Feature Engineering Done successfully on Weekend,
Revenue")


df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))
```

```
################################
# Added Returning_Visitor column #
################################

print("Step 4: Added Returning_Visitor column successfully")

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns=['VisitorType'])



########################################
# Applying One Hot Encoding on Month column #
########################################

print("Step 5: Applied one hot encoding successfully on Month
column")


ordinal_encoder = OrdinalEncoder()
df['Month'] = ordinal_encoder.fit_transform(df[['Month']])



#####################################
# Checking correlation on Revenue column #
#####################################

print("Step 6: Checking correlation done successfully")

result = df[df.columns[1:]].corr()['Revenue']

result1 = result.sort_values(ascending=False)
```

```
####################################
# Preparing Features as X and target as y #
####################################

print("Step 7: Preparing features as X and target as y done
successfully")


X = df.drop(['Revenue'], axis=1)
y = df['Revenue']




#####################################
# Preparing Train and Test Dataset#
#####################################

print("Step 8: Splitting data X_train, X_test, y_train & y_test done
successfully")


X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state = 0)
```

```
###############
# Model Pipeline #
###############

print("Step 9: model_pipeline function created done successfully")

def model_pipeline(X, model):

        n_c = X.select_dtypes(exclude=['object']).columns.tolist()
        c_c = X.select_dtypes(include=['object']).columns.tolist()


        numeric_pipeline = Pipeline([
            ('imputer', SimpleImputer(strategy='constant')),
            ('scaler', MinMaxScaler())
        ])


        categorical_pipeline = Pipeline([
            ('encoder', OneHotEncoder(handle_unknown='ignore'))
        ])


        preprocessor = ColumnTransformer([
            ('numeric', numeric_pipeline, n_c),
            ('categorical', categorical_pipeline, c_c)
        ], remainder='passthrough')



        final_steps = [
            ('preprocessor', preprocessor),
            ('smote', SMOTE(random_state=1)),
            ('feature_selection', SelectKBest(score_func = chi2, k =
        6)),
            ('model', model)
        ]

        return IMBPipeline(steps = final_steps)
```

```
################
# Model Selection #
################

print("Step 10: select_model function created done successfully")

def select_model(X, y, pipeline=None):

    classifiers = {}

    c_d1 = {"DummyClassifier":
    DummyClassifier(strategy='most_frequent')}
    classifiers.update(c_d1)

    c_d4 = {"RandomForestClassifier":
    RandomForestClassifier()}
    classifiers.update(c_d4)

    c_d5 = {"DecisionTreeClassifier": DecisionTreeClassifier()}
    classifiers.update(c_d5)

    c_d9 = {"KNeighborsClassifier": KNeighborsClassifier()}
    classifiers.update(c_d9)

    c_d14 = {"SVC": SVC()}
    classifiers.update(c_d14)

    mlpc = {
    "MLPClassifier (paper)":
    MLPClassifier(hidden_layer_sizes=(27, 50),
    max_iter=300,
    activation='relu',
    solver='adam',
    random_state=1)
    }
    c_d16 = mlpc
    classifiers.update(c_d16)
```

```
cols = ['model', 'run_time', 'roc_auc']
df_models = pd.DataFrame(columns = cols)

for key in classifiers:
        start_time = time.time()
        print()
        print("Step 12: model_pipeline run successfully on",
        key)

        pipeline = model_pipeline(X_train, classifiers[key])

        cv = cross_val_score(pipeline, X, y, cv=10,
        scoring='roc_auc')

        row = {'model': key,
        'run_time': format(round((time.time() -
        start_time)/60,2)),
        'roc_auc': cv.mean(),
        }

        df_models = pd.concat([df_models,
        pd.DataFrame([row])], ignore_index=True)

        df_models = df_models.sort_values(by='roc_auc',
        ascending = False)

return df_models
```

```
###################################
# Access Model select_model function #
###################################

print("Step 11: Accessing select_model function done
successfully")

models = select_model(X_train, y_train)



#############################
# Let's see total model with score #
#############################

print("Step 13: Accessing select_model function done
successfully")

print(models)



#################################
# Accessing best model and training #
#################################

print("Step 14: Accessing select_model function done
successfully")

selected_model = MLPClassifier()
bundled_pipeline = model_pipeline(X_train, selected_model)
bundled_pipeline.fit(X_train, y_train)
```

```
################################
# Accessing best model and training #
################################

print("Step 15: Results predicted successfully")
y_pred = bundled_pipeline.predict(X_test)

print(y_pred)


####################
# ROC and AOC score #
####################

print("Step 16: ROC and AOC scores")

roc_auc = roc_auc_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
f1_score = f1_score(y_test, y_pred)


print('ROC/AUC:', roc_auc)
print('Accuracy:', accuracy)
print('F1 score:', f1_score)


####################
# Classification report #
####################

print("Step 17: classification report generated successfully")

classif_report = classification_report(y_test, y_pred)

print(classif_report)
```

```
##############################################
# This is done, BOSS it's a right time to celebrate :) #
##############################################
```

Output

	model	run_time	roc_auc
14	MLPClassifier	1.79	0.903222
15	MLPClassifier (paper)	2.86	0.900244
2	LGBMClassifier	0.04	0.897217
1	XGBClassifier	0.15	0.891174
10	SGDClassifier	0.02	0.889067
7	AdaBoostClassifier	0.1	0.888264
13	SVC	0.83	0.885963
3	RandomForestClassifier	0.27	0.884997
11	BaggingClassifier	0.07	0.863183
12	BernoulliNB	0.01	0.857851
9	RidgeClassifier	0.01	0.855441
8	KNeighborsClassifier	0.02	0.840505
6	ExtraTreesClassifier	0.01	0.770749
5	ExtraTreeClassifier	0.01	0.754475
4	DecisionTreeClassifier	0.02	0.734040
0	DummyClassifier	0.01	0.500000

```
ROC/AUC: 0.8346658696876629
Accuracy: 0.8764530954311976
F1 score: 0.6774876499647141
```

	precision	recall	f1-score	support
0	0.95	0.90	0.92	3077
1	0.60	0.77	0.68	622
accuracy			0.88	3699
macro avg	0.78	0.83	0.80	3699
weighted avg	0.89	0.88	0.88	3699

24. Final results and words

- ✓ Examining the confusion matrix for the selected model, it shows that we correctly predicted customers (90%) wouldn't purchase during their session, and we correctly predicted that 77% of customers who would purchase during their sessions.
- ✓ Clearly, there's still more we could do, and the overall accuracy of 88%
- ✓ The results show that it's possible to predict purchase intention from consumer behaviour data with a good degree of accuracy.

25. Happy learning

- ✓ While learning this, if having any questions then I am happy to help/support you guys.
- ✓ From Daniel