# Experiment No: 1

**Aim:** Text preprocessing level-1

**Software:** Sypder/ Jupyter Notebook/ Google Colab

## Procedure:

* 1. To implement operations such as change of case, sentence tokenization, word tokenization, stop word removal, punctuation mark removal, stemming, lemmatization, Parts of Speech (PoS) tagging using NLTK (Natural Language Tool Kit) platform.
  2. To implement tokenization without using built in function of nltk.
  3. To comprehend the difference between stemming and lemmatization.
  4. To count frequency of each word in the given document.

## Code:

* 1. **Tokenization Using NLTK**
     + Natural Language Toolkit (NLTK) is library written in python for natural language processing.
     + NLTK has module word\_tokenize() for word tokenization and sent\_tokenize() for sentence tokenization.

# Word Tokenization

!pip install --user -U nltk

import nltk nltk.download('punkt')

from nltk.tokenize import word\_tokenize

text = """There are multiple ways we can perform tokenization on given text data. We can choose any method based on langauge, library and purpose of modeling."""

tokens = word\_tokenize(text) print(tokens)

""" # Sentence Tokenization"""

from nltk.tokenize import sent\_tokenize

text = """Characters like periods, exclamation point and newline char are used

to separate the sentences. But one drawback with split() method, that we can only use one separator at a time! So sentence tokenization won’t be foolproof with split() method.""" sent\_tokenize(text)

""" # Tokenization Using spaCy"""

* + - * spaCy is an open-source software library for advanced natural language processing, written in the programming languages Python and Cython \* in spaCy we create language model object, which then used for word and sentence tokenization
      * Syntax to install spaCy library and English model is as below
      * !pip install spacy
      * !python -m spacy download en # Word Tokenization

import spacy

# Load English model from spacy from spacy.lang.en import English

# Load English tokenizer.

# nlp object will be used to create 'doc' object which uses preprecoessing pipeline's components such as tagger, parser, NER and word vectors

nlp = English()

text = """There are multiple ways we can perform tokenization on given text data. We can choose any method based on langauge, library and purpose of modeling."""

# Now we will process above text using 'nlp' object. Which is use to create documents with linguistic annotations and various nlp properties

my\_doc = nlp(text)

# Above step has already tokenized our text but its in doc format, so lets write fo loop to create list of it

token\_list = []

for token in my\_doc: token\_list.append(token.text)

print(token\_list)

"""# Sentence Tokenization"""

nlp = spacy.load('en\_core\_web\_sm')

# Load English tokenizer, tager, parser, NER and word vectors nlp = English()

# Create the pipeline 'sentencizer' component #sbd = nlp.create\_pipe('sentencizer')

# Add component to the pipeline nlp.add\_pipe('sentencizer')

text = """Characters like periods, exclamation point and newline char are used to separate the sentences. But one drawback with split() method, that we can only use one separator at a time! So sentence tokenization wont be foolproof with split() method."""

# nlp object is used to create documents with linguistic annotations doc = nlp(text)

# Create list of sentence tokens sentence\_list =[]

for sentence in doc.sents:

sentence\_list.append(sentence.text) print(sentence\_list)

## Tokenization without any Tool

# Word Tokenization

text = """There are multiple ways we can perform tokenization on given text data. We can choose any method based on langauge, library and purpose of modeling.""" # Split text by whitespace

tokens = text.split() print(tokens)

"""# Sentence Tokenization"""

# Lets split the given text by full stop (.)

text = """Characters like periods, exclamation point and newline char are used to separate the sentences. But one drawback with

split() method, that we can only use one separator at a time! So sentence tonenization wont be foolproof with split() method. Try now.""" text.split(". ")

# Note the space after the full stop makes sure that we dont get empty element at the end of list.

"""# Tokenization Using Regular Expressions(RegEx)"""

* A regular expression is a sequence of characters that define a search pattern.
* Using RegEx we can match character combinations in string and perform word/sentence tokenization.
* We can use Python's re library for RegeEx related operations.

# Word Tokenization

#Split by Whitespace import re

text = """There are multiple ways we can perform tokenization on given text data. We can choose any method based on langauge, library and purpose of modeling."""

tokens = re.findall("[\w]+", text) print(tokens)

import re

text = 'I\'m with you for entire life in M.P.!' words = re.split(r'\W+', text) print(words[:100])

#Select Words

words = re.split(r'\W+', text) print(words[:100])

import string import re

# split into words by white space words = text.split()

# prepare regex for char filtering

re\_punc = re.compile('[%s]' % re.escape(string.punctuation)) # remove punctuation from each word

stripped = [re\_punc.sub('', w) for w in words] print(stripped[:100])

# string.printable inverse of string.punctuation

re\_print = re.compile('[^%s]' % re.escape(string.printable)) result = [re\_print.sub('', w) for w in words]

print(result)

# Normalizing Case

# split into words by white space

words = text.split() # convert to lower case words = [word.lower() for word in words] print(words[:100])

"""# Sentence Tokenization"""

text = """Characters like periods, exclamation point and newline char are used to separate the sentences. But one drawback with split() method, that we can only use one separator at a time! So sentence tonenization wont be foolproof with split() method."""

tokens\_sent = re.compile('[.!?] ').split(text)

# Using compile method to combine RegEx patterns tokens\_sent

## To comprehend the difference between stemming and lemmatization:

#Stemming import nltk

nltk.download('punkt')

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)))

#Lemmatization import nltk

nltk.download('wordnet')

from nltk.stem import WordNetLemmatizer wordnet\_lemmatizer = WordNetLemmatizer() text = "studies studying cries cry" tokenization = nltk.word\_tokenize(text)

for w in tokenization:

print("Lemma for {} is {}".format(w, wordnet\_lemmatizer.lemmatize(w)))

## To count frequency of each word in the given document

import nltk nltk.download()

from nltk.probability import FreqDist

text1 = """ I have three visions for India. In 3000 years of our history, people from all over the world have come and invaded us, captured our lands, conquered our minds. From Alexander onwards, the Greeks, the Turks, the Moguls, the Portuguese, the British, the French, the Dutch, all of them came and looted us, took over what was ours. Yet we have not done this to any other nation. We have not conquered anyone. We have not grabbed their land, their culture, their history and tried to enforce our way of life on them. Why?

Because we respect the freedom of others.That is why my first vision is that of freedom. I believe that India got its first vision of this in 1857, when we started the War of Independence. It is this freedom that we must protect and nurture and build on. If we are not free, no one will respect us. My second vision for India’s development. For fifty years we have been a developing nation. It is time we see ourselves as a developed nation. We are among the top 5 nations of the world in terms of GDP. We have a 10 percent growth rate in most areas. Our poverty levels are falling. Our achievements are being globally recognised today. Yet we lack the self-confidence to see ourselves as a developed nation, self-reliant and self-assured. Isn’t this incorrect? I have a third vision. India must stand up to the world.

Because I believe that unless India stands up to the world, no one will respect us. Only strength respects strength. We must be strong not only as a military power but also as an economic power. Both must go hand-in-hand. My good fortune was to have worked with three great minds. Dr. Vikram Sarabhai of the Dept. of space, Professor Satish Dhawan, who succeeded him and Dr. Brahm Prakash, father of nuclear material. I was lucky to have worked with all three of them closely and consider this the great opportunity of my life. I see four milestones in my career

"""

print(text1)

# Tokenizing words

words = nltk.word\_tokenize(text1) print(words)

len(words)

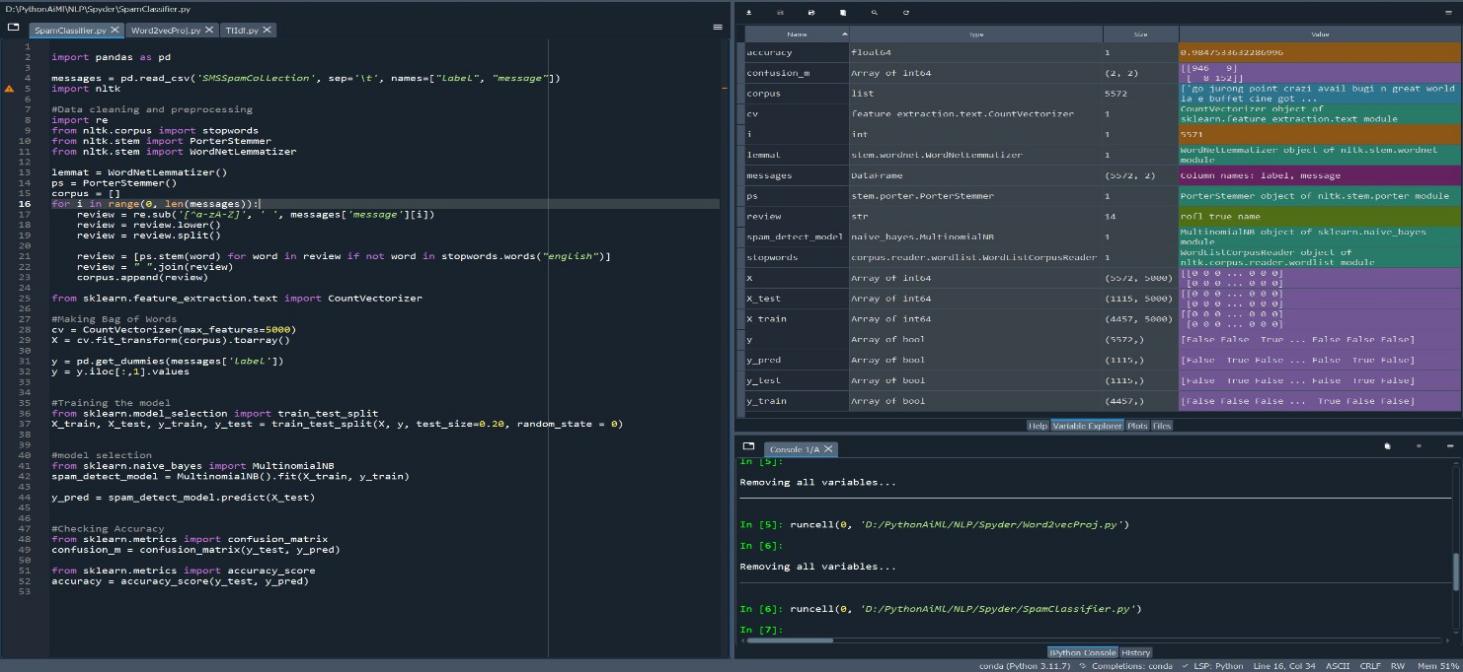
#Frequency Distribution of "words" word2 = FreqDist(words)

print(word2) len(word2)

#10 most common words and their occurrence word2.most\_common(10)

#Frequency of particular word word2.freq("and")

# Output:



**Experiment No: 2**

**Aim:** Text pre-processing level-2

**Software:** Sypder/ Jupyter Notebook/ Google Colab

## Code:

* 1. **To implement label encoding and one hot encoding on textual data**

#Label Encoding

#for label encoding we import LabelEncoder module from sklearn library from sklearn.preprocessing import LabelEncoder path=r"D:\NMIMS\NLP Material\Lab Work\Code\DataSet\Data1.xlsx"

#loading excel into a dataset variable using pandas dataset = pd.read\_csv(path)

#print top 5 rows print(dataset.head())

#print bottom 5 rows print(dataset.tail())

#datatypes of varialbes print(dataset.dtypes)

#lets print unique values of "State" variable print(dataset["State"].unique())

#dataset.mean() will get the mean of the columns (only those columns which has the numeric values)

#dataset.fillna() will replace the "nan" values with whatever we have written as argument dataset = dataset.fillna(dataset.mean())

dataset\_new = dataset.copy()

#create an object of Class LabelEncoder and save the value in a variable label\_encoder = LabelEncoder()

#Label encoding for "State column" dataset\_new["State\_new"]=label\_encoder.fit\_transform(dataset\_new["State"])

#Label encoding for "Purchase column" dataset\_new["Purchase\_new"]=label\_encoder.fit\_transform(dataset\_new["Purchase"])

#One Hot Encoding

import pandas as pd import numpy as np

#for label encoding we import LabelEncoder module from sklearn library from sklearn.preprocessing import OneHotEncoder

path=r"D:\MU\NLP Material\Lab Work\Code\DataSet\Data1.xlsx" #loading excel into a dataset variable using pandas

dataset = pd.read\_excel(path) print(dataset)

#dataset.mean() will get the mean of the columns (only those columns which has the numeric values)

#dataset.fillna() will replace the "nan" values with whatever we have written as argument dataset = dataset.fillna(dataset.mean()) dataset\_new = dataset.copy()

#One hot encoding for column "State"

df\_dummies = pd.get\_dummies(dataset\_new, columns = ["State"])

#to remove prefix from State-Names

df\_dummies = pd.get\_dummies(dataset\_new, columns = ["State"], prefix="")

#to remove "\_" from State-Names

df\_dummies = pd.get\_dummies(dataset\_new, columns = ["State"], prefix="", prefix\_sep="") print(df\_dummies)

## To implement Bag of Words (BoW) feature engineering technique on textual data

# -\*- coding: utf-8 -\*- import nltk

paragraph = """I have three visions for India. In 3000 years of our history, people from all over the world have come and invaded us, captured our lands, conquered our minds. From Alexander onwards, the Greeks, the Turks, the Moguls, the Portuguese, the British, the French, the Dutch, all of them came and looted us, took over what was ours. Yet we have not done this to any other nation. We have not conquered anyone. We have not grabbed their land, their culture, their history and tried to enforce our way of life on them. Why? Because we respect the freedom of others.That is why my first vision is that of freedom. I believe that India got its first vision of this in 1857, when we started the War of Independence. It is this freedom that we must protect and nurture and build on. If we are not free, no one will respect us. My second vision for India’s development. For fifty years we have been a developing nation. It is time we see ourselves as a developed nation. We are among the top 5 nations of the world in terms of GDP. We have a 10 percent growth rate in most areas. Our poverty levels are falling. Our achievements are being globally recognised today. Yet we lack the self-

confidence to see ourselves as a developed nation, self-reliant and self-assured. Isn’t this incorrect? I have a third vision. India must stand up to the world. Because I believe that unless India stands up to the world, no one will respect us. Only strength respects strength. We must be strong not only as a military power but also as an economic power. Both must go hand-in-hand. My good fortune was to have worked with three great minds. Dr. Vikram Sarabhai of the Dept. of space, Professor Satish Dhawan, who succeeded him and Dr. Brahm Prakash, father of nuclear material. I was lucky to have worked with all three of them closely and consider this the great opportunity of my life. I see four milestones in my career"""

# Cleaning the texts import re

from nltk.corpus import stopwords

from nltk.stem.porter import PorterStemmer from nltk.stem import WordNetLemmatizer

ps = PorterStemmer() wordnet=WordNetLemmatizer() sentences = nltk.sent\_tokenize(paragraph) corpus = []

for i in range(len(sentences)):

review = re.sub('[^a-zA-Z]', ' ', sentences[i]) review = review.lower()

review = review.split()

review = [wordnet.lemmatize(word) for word in review if not word in set(stopwords.words('english'))]

review = ' '.join(review) corpus.append(review)

# Creating the Bag of Words model

from sklearn.feature\_extraction.text import CountVectorizer

# create the transform cv = CountVectorizer()

# tokenize and build vocab cv.fit(corpus)

#creating vacabulary with index vocab = cv.vocabulary\_

#creating an array for Bag of Word

x = cv.fit\_transform(corpus).toarray()

## To implement TF-IDF feature engineering technique

import nltk

paragraph = """I have three visions for India. In 3000 years of our history, people from all over the world have come and invaded us, captured our lands, conquered our minds. From Alexander onwards, the Greeks, the Turks, the Moguls, the Portuguese, the British, the French, the Dutch, all of them came and looted us, took over what was ours. Yet we have not done this to any other nation. We have not conquered anyone. We have not grabbed their land, their culture, their history and tried to enforce our way of life on them. Why? Because we respect the freedom of others.That is why my first vision is that of freedom. I believe that India got its first vision of this in 1857, when we started the War of Independence. It is this freedom that we must protect and nurture and build on. If we are not free, no one will respect us. My second vision for India’s development. For fifty years we have been a developing nation. It is time we see ourselves as a developed nation. We are among the top 5 nations of the world in terms of GDP. We have a 10 percent growth rate in most areas. Our poverty levels are falling. Our achievements are being globally recognised today. Yet we lack the self- confidence to see ourselves as a developed nation, self-reliant and self-assured. Isn’t this incorrect? I have a third vision. India must stand up to the world. Because I believe that unless India stands up to the world, no one will respect us. Only strength respects strength. We must be strong not only as a military power but also as an economic power. Both must go hand-in-hand. My good fortune was to have worked with three great minds. Dr. Vikram Sarabhai of the Dept. of space, Professor Satish Dhawan, who succeeded him and Dr. Brahm Prakash, father of nuclear material. I was lucky to have worked with all three of them closely and consider this the great opportunity of my life. I see four milestones in my career"""

# Cleaning the texts import re

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

wordnet=WordNetLemmatizer() sentences = nltk.sent\_tokenize(paragraph) corpus = []

for i in range(len(sentences)):

review = re.sub('[^a-zA-Z]', ' ', sentences[i]) review = review.lower()

review = review.split()

review = [wordnet.lemmatize(word) for word in review if not word in set(stopwords.words('english'))]

review = ' '.join(review) corpus.append(review)

# Creating the Bag of Words model

from sklearn.feature\_extraction.text import TfidfVectorizer # create the transform

cv = TfidfVectorizer()

# tokenize and build vocab cv.fit(corpus)

#creating vacabulary with index vocab = cv.vocabulary\_

#creating an array for Bag of Word

x = cv.fit\_transform(corpus).toarray()

## To analyze and comprehend the effect of various approaches to convert text into vectors

import pandas as pd

path=r"D:\MU\NLP Material\Lab Work\Code\DataSet\Iris.csv"

#loading excel into a dataset variable using pandas dataset = pd.read\_csv(path)

#print top 5 rows print(dataset.head())

#print bottom 5 rows print(dataset.tail())

#datatypes of varialbes print(dataset.dtypes)

#lets print unique values of "Class" variable print(dataset["Class"].unique())

#lets print value count of "Class" variable print(dataset["Class"].value\_counts())

dataset\_new = dataset.copy()

#for label encoding we import LabelEncoder module from sklearn library from sklearn.preprocessing import LabelEncoder

#create an object of Class LabelEncoder and save the value in a variable label\_encoder = LabelEncoder() print(label\_encoder)

#Label encoding for target variable "Class" which is an object dataset\_new["Class\_new"]=label\_encoder.fit\_transform(dataset\_new["Class"])

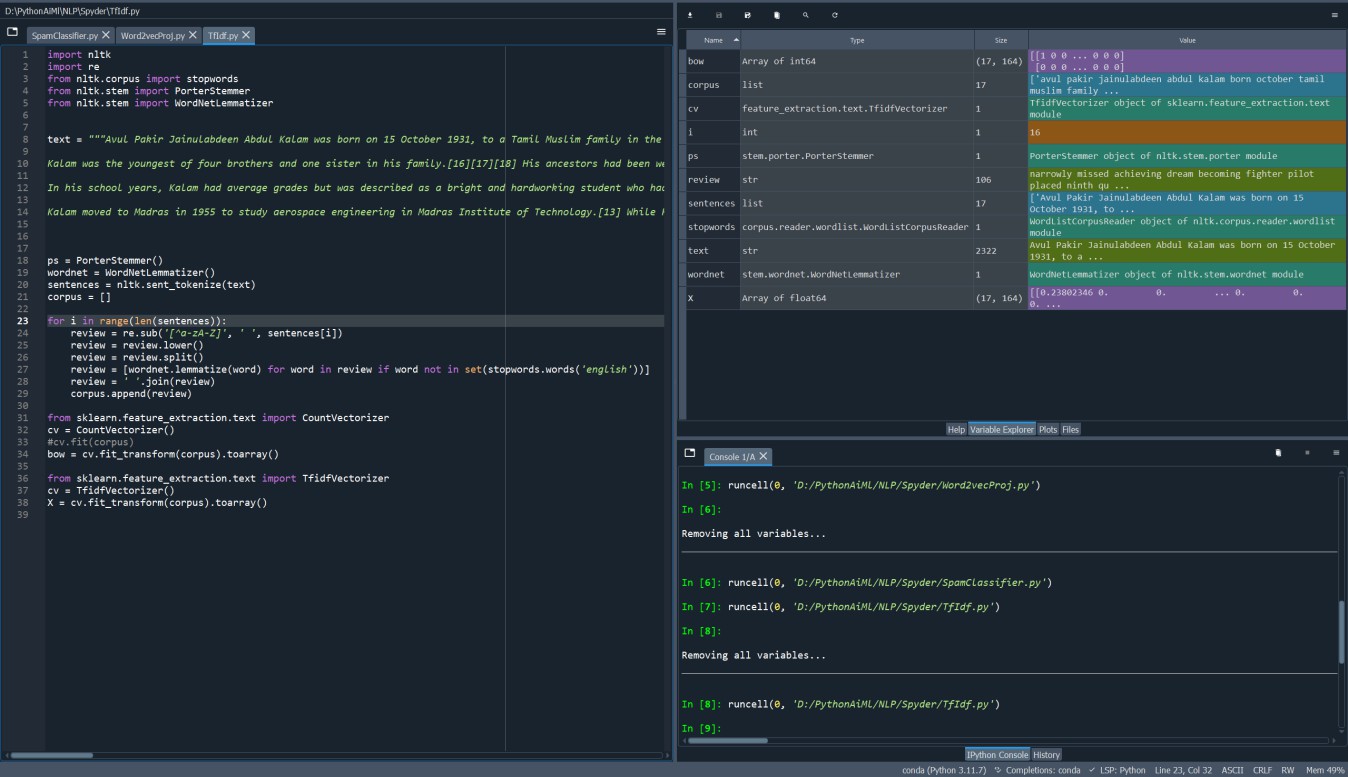
#print top 5 rows print(dataset.head())

#datatypes of varialbes print(dataset.dtypes)

#lets print unique values of "Class" variable print(dataset["Class"].unique())

#lets print value count of "Class" variable print(dataset["Class"].value\_counts()

**Output:**



# Experiment No: 3

**Aim:** Analysing Gutenberg and Brown corpus with python

**Software:** Sypder/ Jupyter Notebook/ Google Colab

## Procedure:

* 1. **Working with corpus file lists**

import nltk

#to see the file identifiers in gutenberg corpus nltk.corpus.gutenberg.fileids()

#Let's pick out the first of these texts — Emma by Jane Austen — and give it a short name, #emma, then find out how many words it contains:

emma = nltk.corpus.gutenberg.words('austen-emma.txt') print(emma)

len(emma)

#Similarly, we can use other text

shakespeare = nltk.corpus.gutenberg.words('shakespeare-macbeth.txt') print(shakespeare)

len(shakespeare)

#Concerdance View of text

emma = nltk.Text(nltk.corpus.gutenberg.words('austen-emma.txt')) emma.concordance("surprize")

"""When we defined emma, we invoked the words() function of the gutenberg object in NLTK's corpus package. Butsince it is cumbersome to type such long names all the time, Python provides another version of the import statement,as follows:"""

from nltk.corpus import gutenberg gutenberg.fileids()

emma = gutenberg.words('austen-emma.txt') print(emma)

emma2 = gutenberg.words('austen-persuasion.txt') print(emma2)

#to see whole text we can use raw() function emma2 = gutenberg.raw('austen-persuasion.txt') print(emma2)

#to see the sentences in a particular text we can use sents() function emma2 = gutenberg.sents('austen-persuasion.txt')

print(emma2) len(emma2)

#to see a particular sentence

emma2\_sentence = gutenberg.sents('austen-persuasion.txt') print(emma2\_sentence[15])

#to see the longest sentence of the text we can ue following code: longest\_len = max(len(s) for s in emma2\_sentence)

[s for s in emma2\_sentence if len(s) == longest\_len]

"""Let's write a short program to display other information about each text, by looping over all the values of fileid corresponding to the gutenberg file identifiers listed earlier and then computing statistics for each text. For a compactoutput display, we will round each number to the nearest integer, using round ()."""

for fileid in gutenberg.fileids():

num\_chars = len(gutenberg.raw(fileid)) num\_words = len(gutenberg.words(fileid)) num\_sents = len(gutenberg.sents(fileid))

num\_vocab = len(set(w.lower() for w in gutenberg.words(fileid))) print(round(num\_chars/num\_words), round(num\_words/num\_sents), round(num\_words/num\_vocab), fileid)

"""Above program displays three statistics for each text: average word length, average sentence length, and the number of times each vocabulary item appears in the text on average (our lexical diversity score)."""

#Let's take another text say 'blake-poems.txt' blake = gutenberg.words('blake-poems.txt') print(blake)

blake = gutenberg.raw('blake-poems.txt') print(blake)

blake = gutenberg.sents('blake-poems.txt') print(blake) len(blake)

blake\_sentence = gutenberg.sents('blake-poems.txt') print(blake\_sentence[15])

* 1. **Working with file contents** import nltk nltk.download('brown')

from nltk.corpus import brown

brown.categories() brown.words(categories='news') brown.words(fileids=['cg22'])

brown.sents(categories=['news','editorial','reviews'])

"""The Brown Corpus is a convenient resource for studying systematic differences between genres, a kind of linguistic inquiry known as stylistics. Let's compare genres in their usage of modal verbs. The first step is to produce the countsfor a particular genre. Remember to

import nltk before doing the following: """

import nltk nltk.download('brown')

from nltk.corpus import brown

news\_text = brown.words(categories='news')

fdist = nltk.FreqDist(w.lower() for w in news\_text) modals = ['can','could','may','might','must','will'] for m in modals:

print(m +':', fdist[m], end=' ')

## Visualization

!pip install wordcloud

!pip install wikipedia

from wordcloud import WordCloud, STOPWORDS , ImageColorGenerator import pandas as pd

import wikipedia

import matplotlib.pylab as plt

from PIL import Image import numpy as np stop\_word = set(STOPWORDS) info = wikipedia.summary("Programming")

print(info) word\_cloud = WordCloud(stopwords = stop\_word , width=1600 , height=800, background\_color="White",colormap="Set2").generate(info) plt.figure(figsize=(20,10),facecolor='k') plt.imshow(word\_cloud,interpolation='bilinear') plt.axis('off') plt.tight\_layout (pad=0)

word\_cloud.to\_file ('D:/nasirsoft\_word\_cloud.png') plt.show()

# Experiment No: 4

**Aim:** Exploratory data analysis for textual data.

**Software:** Sypder/ Jupyter Notebook/ Google Colab

## Procedure:

* 1. **Most frequent words distribution, average chapter length, most frequent phrases (bi, tri and quad-grams), names of characters, places, and events.**

!pip install gensim

# Commented out IPython magic to ensure Python compatibility. import base64

import numpy as np import pandas as pd # Plotly imports

import plotly.offline as py py.init\_notebook\_mode(connected=True) import plotly.graph\_objs as go

import plotly.tools as tls # Other imports

from pprint import pprint

from collections import Counter # from scipy.misc import imread

from sklearn.feature\_extraction.text import TfidfVectorizer, CountVectorizer from sklearn.decomposition import NMF, LatentDirichletAllocation

from matplotlib import pyplot as plt import matplotlib

# %matplotlib inline

from os import path from wordcloud import WordCloud, STOPWORDS,

ImageColorGenerator import nltk nltk.download('stopwords') nltk.download('punkt')

nltk.download('averaged\_perceptron\_tagger')

nltk.download('maxent\_ne\_chunker') nltk.download('words') nltk.download('vader\_lexicon') nltk.download('opinion\_lexicon') from nltk.corpus import stopwords

stop\_words = stopwords.words('english') from nltk import ngrams, FreqDist

from nltk.sentiment import SentimentIntensityAnalyzer

import re

#from gensim.summarization import summarize #from gensim.summarization import keywords import spacy

import seaborn as sns

print("all imports worked nicely") """### File parse

Parsing the file as a python dictionary of key, value pairs as ChapterName and ChapterContent. This dictionary can then be used to generate a \*\*Pandas Dataframe\*\*.

"""

# opening the text file and reading it's contents.

with open('/content/sample\_data/War\_and\_Peace.txt', 'r', encoding='latin-1') as file: txt = file.readlines()

print(txt) chap\_name = "" content = {}

# parsing the opened file into a dictionary, where "key" = Chapter Name and "value" = Contents of chapter.

"""for line in txt :

if "CHAPTER" in line: #print line,

chap\_name = line.strip() content[chap\_name] = chap\_name

else:

content[chap\_name] = content[chap\_name] + line

# for key, value in content.iteritems() :

# print key, value

# Testing the parsing works as expected.""" print(len(content))

content['CHAPTER I']

# calculating the length of each chapter in terms of number of words df = pd.DataFrame(content.items(), columns=['ChapterName', 'ChapterContent'])

word\_count = np.zeros(28) chapter = 0

for text in df['ChapterContent']: count = text.split()

word\_count[chapter] = len(count) chapter = chapter + 1

# getting word count for individual chapters print(word\_count) """### Word Distribution

Chapterwise word distribution. \*\*WordClouds\*\* - show most frequently occuring words,

with higher frequency words corelating to high font size. Generating wordCloud showing the most frequently occurring words per Chapter.

"""

# import matplotlib as mpl

# # Just making the plots look better # mpl.style.use('ggplot')

my\_colors = 'rgbkymc'

plt.figure(figsize=(5,7))

# plt.rcParams["figure.figsize"] = [16,9] plt.ylabel('Chapters') plt.title('Word Counts by Chapter')

# plt.bar(df.WordCount, df.ChapterName) opacity = 0.4

df['WordCount'] = word\_count.tolist()

plt.barh(df.ChapterName, df.WordCount, alpha = opacity, color=['black', 'red', 'green', 'blue', 'cyan'])

# plt.barh(df.ChapterName, df.WordCount, color= my\_colors)

plt.show()

# df = df.drop('wordCount', 1) df.head(30)

# WordClouds - show most frequently occuring words, with higher frequency words correlating to high font size

# generating wordCloud showing the most frequently occurring words per Chapter

color\_flag = True chapter\_count = 0

for text in df['ChapterContent']: if color\_flag:

wordcloud = WordCloud(stopwords=stop\_words).generate(text)

else:

wordcloud = WordCloud(stopwords=stop\_words, background\_color="white").generate(text)

# Display the generated image:

plt.title("WordCloud " + df['ChapterName'][chapter\_count]) plt.imshow(wordcloud, interpolation='bilinear') plt.axis("off")

plt.show()

color\_flag = not color\_flag chapter\_count = chapter\_count + 1

"""### Phrase Distribution

Performing Phrase distribution(s) throughout the whole book and then for individual Chapter(s).

"""

# data = df['ChapterContent'][0]

# Phrase distribution(s) throughout the whole book data = ''.join(txt)

all\_counts = dict()

# counting till n-grams, n ranging from 2..5 for size in 2, 3, 4, 5:

all\_counts[size] = FreqDist(ngrams(data.split(), size)) for count in 2, 3, 4, 5:

data = all\_counts[count].most\_common(5) phrase\_count = [x[1] for x in data] phrase\_content = [' '.join(x[0]) for x in data] fig, ax = plt.subplots()

bar\_width = 0.35

opacity = 0.4

plt.barh(phrase\_content, phrase\_count, bar\_width, alpha = opacity, color=['black', 'red', 'green', 'blue', 'cyan'])

plt.xlabel('Count') plt.ylabel('Phrases')

plt.title('Occurrences of %s-grams' % count) plt.legend()

plt.tight\_layout() plt.show()

# Phrase distribution(s) throughout individual Chapters, eg. Chapter XI data = df['ChapterContent'][0] all\_counts = dict()

for size in 2, 3, 4, 5:

all\_counts[size] = FreqDist(ngrams(data.split(), size)) for count in 2, 3, 4, 5:

data = all\_counts[count].most\_common(5) phrase\_count = [x[1] for x in data] phrase\_content = [' '.join(x[0]) for x in data] bar\_width = 0.35

opacity = 0.4

plt.barh(phrase\_content, phrase\_count, bar\_width, alpha = opacity, color=['black', 'red', 'green', 'blue', 'cyan'])

plt.xlabel('Count') plt.ylabel('Phrases')

plt.title('Occurrences of %s-grams' % count) plt.legend()

plt.tight\_layout() plt.show()

# NOTE:

The process of Stop Word removal happens next, as "phrase" distribution should be done with stop words taken into account.

# Commented out IPython magic to ensure Python compatibility. def tokenize\_and\_stopWordsRemoval(text):

# first tokenize by sentence, then by word to ensure that punctuation is caught as it's own token

tokens = [word for sent in nltk.sent\_tokenize(text) for word in nltk.word\_tokenize(sent)] filtered\_tokens = []

# filter out any tokens not containing letters (e.g., numeric tokens, raw punctuation) and words whose length is less than 2 characters - removes

<br/a> formating from data

for token in tokens:

if re.search('[a-zA-Z]', token) and len(token) > 2 and token not in stop\_words : filtered\_tokens.append(token)

return filtered\_tokens #tokenize

# %time tokenized\_text = [tokenize\_and\_stopWordsRemoval(text) for text in txt] print len(tokenized\_text)

print tokenized\_text[11] print txt[11]

"""# Name and Place Identification

This can be achieved by Named Entity Recognition (NER). Here I first utilize Spacy for this purpose but after seeing quiet a few exceptions in the results, I have also shown the same task using NLTK's NER.

Top 10 results for both the categories(Name of people and places) are shown below. """

import spacy

nlp = spacy.load('en') doc\_str = ""

for item in tokenized\_text:

doc\_str = doc\_str + " " + " ".join(item) doc = nlp(doc\_str.decode('utf-8'))

print type(doc)

# print([(X.text, X.label\_) for X in doc.ents]) list\_person = []

list\_place = []

for X in doc.ents:

if (X.label\_ == u'PERSON'): list\_person.append((X.text, X.label\_))

elif (X.label\_ == u'GPE')| (X.label\_ == u'LOC') : list\_place.append((X.text, X.label\_))

# Set operation on lists to remove the duplicate entries list\_person = list(set(list\_person))

list\_place = list(set(list\_place))

# printing first 10 items in each category print list\_person[:10]

print list\_place[:10] nltk\_personList = [] nltk\_placeList = [] import nltk

for sent in nltk.sent\_tokenize(doc\_str):

for chunk in nltk.ne\_chunk(nltk.pos\_tag(nltk.word\_tokenize(sent))): if hasattr(chunk, 'label'):

if (chunk.label() == 'PERSON'):

nltk\_personList.append((chunk.label(), ' '.join(c[0] for c in chunk))) elif (chunk.label() == 'GPE'):

nltk\_placeList.append((chunk.label(), ' '.join(c[0] for c in chunk)))

# Set operation on lists to remove the duplicate entries nltk\_personList = list(set(nltk\_personList)) nltk\_placeList = list(set(nltk\_placeList))

# printing first 10 items in each category print nltk\_personList[:10]

print nltk\_placeList[:10]

"""#Sentiment Ananlysis

Here I attempt to get the sentiment of each line in the whole document based on \*\*NLTK's Vader Analyzer\*\*.

To keep the task simplistic, I am refraining from training a Neural network architecture (LSTM or CNN)

for this, as it will need the training to be done on some other labelled data set and then utilize transfer learning to work on this piece of text.

Instead, I use NLTK, and label sentences polarity based on the polarity score of its individual words.

The sentiment scoring consists of 4 tags: Neu, Neg, Pos and compound. The first three represent the sentiment score percentage of each category in our sentence, and the compound single number that scores the sentiment.

`compound` ranges from -1 (Extremely Negative) to 1 (Extremely Positive).

I have considered sentences with a compound value greater than 0.2 as positive and less than -0.2 as negative. The \*\*label\*\* column in data frame refers to the sentiment of the sentence, 0 being neutral, +1 positive and -1 negative respectively. """

from nltk.sentiment import SentimentIntensityAnalyzer vader\_analyzer = SentimentIntensityAnalyzer() sentiments = []

for text in txt[:10]:

for sent in nltk.sent\_tokenize(text): print sent

print(vader\_analyzer.polarity\_scores(sent)) sent\_dic = vader\_analyzer.polarity\_scores(sent) sent\_dic["sentence"] = sent sentiments.append(sent\_dic)

df\_sentiments = pd.DataFrame.from\_records(sentiments) df\_sentiments['label'] = 0

df\_sentiments.loc[df\_sentiments['compound'] > 0.2, 'label'] = 1

df\_sentiments.loc[df\_sentiments['compound'] < -0.2, 'label'] = -1

df\_sentiments.head() df\_sentiments.head(10) import seaborn as sns

sns.set(style='darkgrid', context='talk', palette='Dark2') # printing first 5 positive sentences from the dataframe print("Positive sentences:\n")

print (list(df\_sentiments[df\_sentiments['label'] == 1].sentence)[:5], width=200) # printing first 5 negative sentences from the dataframe

print("\nNegative sentences:\n")

print (list(df\_sentiments[df\_sentiments['label'] == -1].sentence)[:5], width=200)

## Findings through word-clouds, bar plots and histograms.

fig, ax = plt.subplots(figsize=(5, 5))

counts = df\_sentiments.label.value\_counts(normalize=True) \* 100 sns.barplot(x=counts.index, y=counts, ax=ax) ax.set\_xticklabels(['Negative', 'Neutral', 'Positive']) ax.set\_ylabel("Percentage")

plt.show()

"""# Text Summarizer

Here I am creating summary of each Chapter, using \*\*Gensim's summarizer\*\*.

This summarizer is based on \*\*TextRank algorithm\*\* which is losely similar to PageRank algorithm used by Google. Text is converted to graph with vertices and edges and the more a sentence is linked by other sentence the higher it's score grows. Summary is generated by top high scoring sentences. """

print type(df.ChapterContent[0]) chapter\_count = 0

for text in df['ChapterContent']:

# Display the generated Chapterwise summary:

print ('Summary:%s' % df['ChapterName'][chapter\_count])

summary = summarize(df.ChapterContent[chapter\_count]) print (summarize(df.ChapterContent[chapter\_count]))

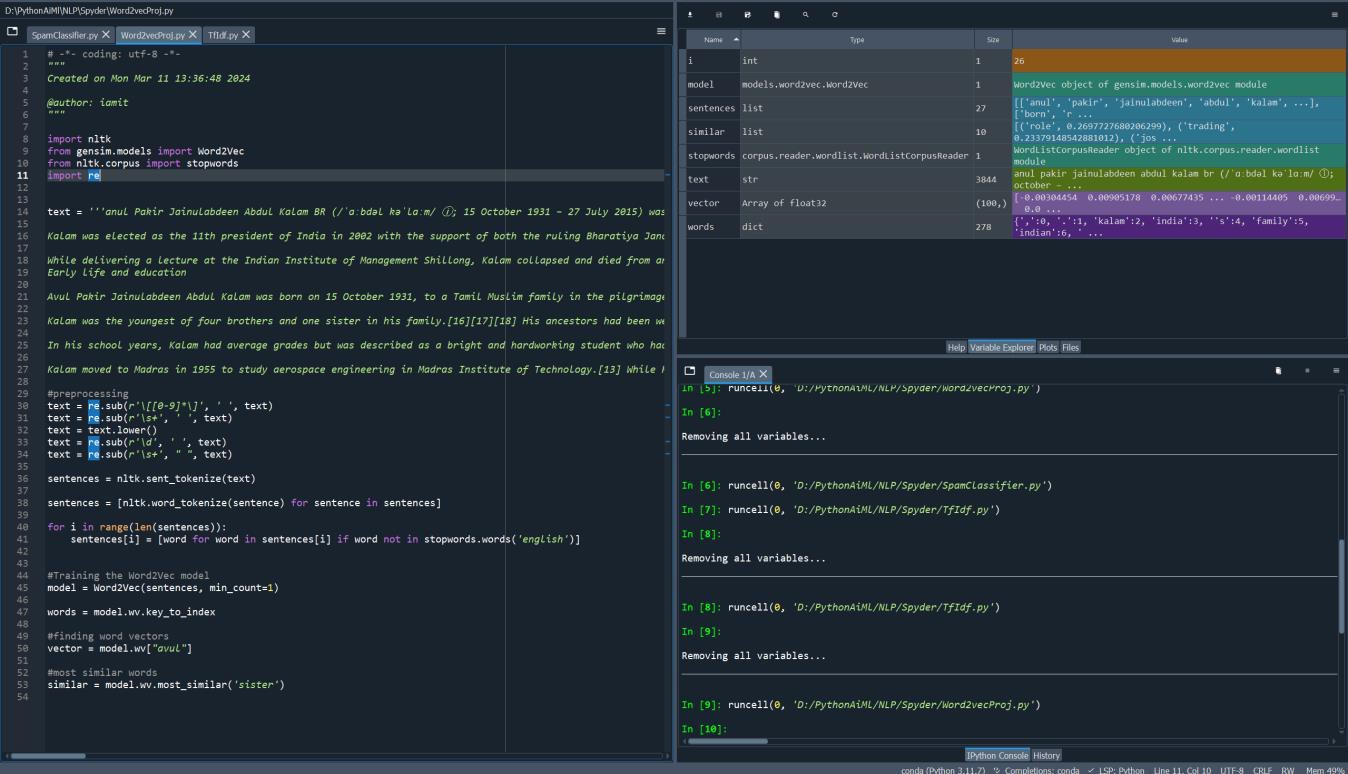
print

print "Length of original text %d, length of summary %d characters." % (len(df['ChapterContent'][chapter\_count]), len(summary))

print

chapter\_count = chapter\_count + 1

**Output:**



# Experiment No: 5

**Aim:** Information extraction -Part of Speech (POS)tagging and Named Entity Recognition (NER)

**Software:** Sypder/ Jupyter Notebook/ Google Colab

## Procedure:

1. **Identify the Part of speech like noun, verb, adjective, adverb and tag it**

#POS Tagging Example1

import nltk nltk.download()

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize, sent\_tokenize text = word\_tokenize("And now for something completely different")

nltk.pos\_tag(text)

#POS Tagging Example2 import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize, sent\_tokenize text = word\_tokenize("They refuse to permit us to obtain the refuse permit")

nltk.pos\_tag(text) import nltk nltk.download()

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize, sent\_tokenizestop\_words set(stopwords.words('english'))

##Dummy text

txt = "Sukanya, Rajib and Naba are my good friends. "

\"Sukanya is getting married next year. " \ "Marriage is a big step in one’s life."

\"It is both exciting and frightening. " \ "But friendship is a sacred bond between people."

\"It is a special kind of love between us. " \ "Many of you must have tried searching for a friend "\ "but never found the right one."

# sent\_tokenize is one of instances of PunktSentenceTokenizer from the nltk.tokenize.punkt module

tokenized = sent\_tokenize(txt) for i in tokenized:

# Word tokenizers is used to find the words and punctuation in a string wordsList = nltk.word\_tokenize(i)

# removing stop words

from wordList wordsList = [w for w in wordsList if not w in stop\_words] # Using a Tagger. Which is part-of-speech # tagger or POS-tagger. tagged = nltk.pos\_tag(wordsList)

print(tagged**)**

## Identify Named Entity in text data

import re, nltk

sentence = [("the","DT"), ("little","JJ"), ("yellow","JJ"), ("dog","NN"), ("barked","VBD"), ("at","IN"), ("the","DT"), ("cat","NN")]

grammar = "NP: {?\*}"

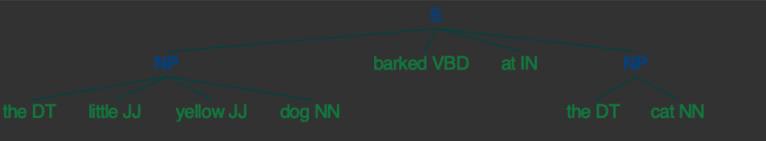
cp = nltk.RegexpParser(grammar) result = cp.parse(sentence) print(result)

result.draw()

text =''' he PRP B-NP accepted VBD BVPthe DT B-NP position NN I-NP of IN B-PP vice NN B-NP chairman NN I-NP of IN B-PP Carlyle NNP B-NP Group NNP I-NP ,,O a DT B- NP merchant NN I-NP banking NN I-NP concern NN I-NP .. O'''

nltk.chunk.conllstr2tree(text, chunk\_types=[‘NP']).draw()

**Output:**



# Experiment No: 6

**Aim:** Study and use of libraries in Python for data import, preprocessing and Machine Learning.

## Theory:

1. **NumPy:**

**Purpose:** NumPy, short for Numerical Python, is a fundamental package for numerical computingin Python.

**Features:** It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.

**Importance**: NumPy is essential for scientific computing and data analysis tasks in Python due to its efficient array operations and mathematical functions. It serves as the foundation for many other libraries in the Python scientific ecosystem.

## Pandas:

**Purpose:** Pandas is a fast, powerful, and flexible open-source data analysis and manipulation library built on top of NumPy.

**Features:** It offers data structures and functions designed to make working with structured (tabular, multidimensional, potentially heterogeneous) data intuitive and straightforward.

**Importance:** Pandas simplifies data manipulation, analysis, and exploration tasks, making it an essential tool for data scientists and analysts. It is commonly used for data preprocessing, cleaning, transformation, and analysis.

## Scikit-learn (sklearn):

**Purpose:** Scikit-learn is a simple and efficient tool for data mining and data analysis in Python.

**Features:** It features various machine learning algorithms and tools for classification, regression, clustering, dimensionality reduction, and more.

**Importance:** Scikit-learn is widely used for implementing machine learning algorithms and tasks, offering ease of use, consistent interfaces, and integration with other Python libraries. It is suitablefor both beginners and experienced machine learning practitioners.

## Matplotlib:

**Purpose:** Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

**Features:** It provides a MATLAB-like interface for creating a wide variety of plots and charts, including line plots, scatter plots, bar charts, histograms, and more.

**Importance:** Matplotlib is widely used in various fields for visualizing data and communicating results effectively, making it a crucial tool for data scientists, engineers, researchers, and analysts.

## PyTorch:

**Purpose:** PyTorch is an open-source machine learning library developed by Facebook's AI Research lab.

**Features:** It is known for its dynamic computational graph capabilities, which allow for flexible and intuitive model development.

**Importance:** PyTorch is widely used in research and production settings for building and training deep learning models, especially in fields such as computer vision, natural language processing, and reinforcement learning. It offers strong support for GPU acceleration and distributed computing.

## Keras:

**Purpose:** Keras is a high-level neural networks API written in Python and capable of running on top of TensorFlow, Theano, or Microsoft Cognitive Toolkit (CNTK).

**Features:** It provides a simple and intuitive interface for building and training neural networks with minimal code.

**Importance:** Keras is widely used for rapid prototyping and experimentation with deep learning models, offering ease of use, modularity, and extensibility. It emphasizes user-friendliness and allows for quick development and testing of neural network architectures.

## Seaborn:

**Purpose:** Seaborn is a Python data visualization library designed to simplify the creation of statistical graphics for data exploration and analysis.

**Features**: It offers a high-level interface for creating a variety of statistical plots, such as histograms, scatterplots, and heatmaps. It also integrates seamlessly with Pandas DataFrames for easy data manipulation and visualization. It incorporates statistical estimation techniques for visualizing summary statistics and confidence intervals.

**Importance**: It facilitates exploratory data analysis by visualizing data distributions, patterns, and relationships. It aids in model evaluation and interpretation by visualizing evaluation metrics and modelperformance. It also enhances the communication of insights derived from data analysis through visually appealing andinformative plots.

## Procedure:

1. Handle missing values, duplicates, and outliers while preserving data integrity.
2. Engineer features to capture additional information without losing original context.
3. Select features judiciously to maintain data representation and minimize information loss.
4. Split data into subsets for training, validation, and testing while retaining overall data distribution.
5. Scale or normalize features to maintain relative relationships and avoid distortion.
6. Handle class imbalance and perform data augmentation to enhance dataset diversity while preserving class proportions.
7. Validate preprocessed data rigorously to ensure fidelity to original dataset and document transformations comprehensively.

## Data Set:



**Code:**

**#** Importing the libraries

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

#Importing the dataset

dataset = pd.read\_csv('Data.csv')

X = dataset.iloc[:, :-1].values y = dataset.iloc[:, -1].values print(X)

[['France' 44.0 72000.0]

['Spain' 27.0 48000.0]

['Germany' 30.0 54000.0]

['Spain' 38.0 61000.0]

['Germany' 40.0 nan]

['France' 35.0 58000.0]

['Spain' nan 52000.0]

['France' 48.0 79000.0]

['Germany' 50.0 83000.0]

['France' 37.0 67000.0]]

print(y)

['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']

# Taking care of missing data

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing\_values=np.nan, strategy='mean') imputer.fit(X[:, 1:3])

X[:, 1:3] = imputer.transform(X[:, 1:3]) print(X)

[['France' 44.0 72000.0]

['Spain' 27.0 48000.0]

['Germany' 30.0 54000.0]

['Spain' 38.0 61000.0]

['Germany' 40.0 63777.77777777778]

['France' 35.0 58000.0]

['Spain' 38.77777777777778 52000.0]

['France' 48.0 79000.0]

['Germany' 50.0 83000.0]

['France' 37.0 67000.0]]

**#** Encoding the Independent Variable

from sklearn.compose import ColumnTransformer from sklearn.preprocessing import OneHotEncoder

ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough') X = np.array(ct.fit\_transform(X))

print(X)

[[1.0 0.0 0.0 44.0 72000.0]

[0.0 0.0 1.0 27.0 48000.0]

[0.0 1.0 0.0 30.0 54000.0]

[0.0 0.0 1.0 38.0 61000.0]

[0.0 1.0 0.0 40.0 63777.77777777778]

[1.0 0.0 0.0 35.0 58000.0]

[0.0 0.0 1.0 38.77777777777778 52000.0]

[1.0 0.0 0.0 48.0 79000.0]

[0.0 1.0 0.0 50.0 83000.0]

[1.0 0.0 0.0 37.0 67000.0]]

# Encoding the Dependent Variable

from sklearn.preprocessing import LabelEncoder le = LabelEncoder()

y = le.fit\_transform(y) print(y)

[0 1 0 0 1 1 0 1 0 1]

# Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 1) print(X\_train)

[[0.0 0.0 1.0 38.77777777777778 52000.0]

[0.0 1.0 0.0 40.0 63777.77777777778]

[1.0 0.0 0.0 44.0 72000.0]

[0.0 0.0 1.0 38.0 61000.0]

[0.0 0.0 1.0 27.0 48000.0]

[1.0 0.0 0.0 48.0 79000.0]

[0.0 1.0 0.0 50.0 83000.0]

[1.0 0.0 0.0 35.0 58000.0]]

print(X\_test)

[[0.0 1.0 0.0 30.0 54000.0]

[1.0 0.0 0.0 37.0 67000.0]]

print(y\_train) [0 1 0 0 1 1 0 1]

print(y\_test) [0 1]

# Feature Scaling

from sklearn.preprocessing import StandardScaler sc = StandardScaler()

X\_train[:, 3:] = sc.fit\_transform(X\_train[:, 3:])

X\_test[:, 3:] = sc.transform(X\_test[:, 3:]) print(X\_train)

[[0.0 0.0 1.0 -0.19159184384578545 -1.0781259408412425]

[0.0 1.0 0.0 -0.014117293757057777 -0.07013167641635372]

[1.0 0.0 0.0 0.566708506533324 0.633562432710455]

[0.0 0.0 1.0 -0.30453019390224867 -0.30786617274297867]

[0.0 0.0 1.0 -1.9018011447007988 -1.420463615551582]

[1.0 0.0 0.0 1.1475343068237058 1.232653363453549]

[0.0 1.0 0.0 1.4379472069688968 1.5749910381638885]

[1.0 0.0 0.0 -0.7401495441200351 -0.5646194287757332]]

print(X\_test)

[[0.0 1.0 0.0 -1.4661817944830124 -0.9069571034860727]

[1.0 0.0 0.0 -0.44973664397484414 0.2056403393225306]

# Experiment No: 7(A)Simple Linear Regression

**Aim:** To Implement Linear Regression and Logistic Regression.

**Theory:** Simple linear regression is a statistical method used to model the relationship between two continuous variables: one independent variable (x) and one dependent variable (y). The goal of simple linear regression is to find the best-fitting linear equation that describes the relationship between (x) and (y).

The linear equation for simple linear regression is represented as:

[ y = mx + b] where:

* (y) is the dependent variable (the variable being predicted or explained).
* (x) is the independent variable (the variable used to make predictions).
* (m) is the slope of the line, which represents the rate of change in (y) for a unit change in (x).
* (b) is the y-intercept, the value of (y) when (x) is 0.

Simple linear regression is widely used in various fields such as economics, finance, social sciences,and engineering for tasks such as predicting house prices based on square footage, analyzing the impact of temperature on crop yields, or predicting sales based on advertising expenditure.

## Procedure:

The process of simple linear regression involves the following steps:

1. **Model Fitting:** The algorithm aims to find the best-fitting line that minimizes the difference between the observed values of (y) and the values predicted by the linear equation.
2. **Parameter Estimation**: The algorithm estimates the values of (m) and (b) that minimize the error between the observed and predicted values. This is typically done using optimization techniques like least squares estimation.
3. **Model Evaluation:** The performance of the model is evaluated using metrics such as the coefficient of determination (R^2) and mean squared error (MSE). These metrics assess how well the model fits the observed data.
4. **Prediction:** Once the model parameters (m) and (b) are determined, the linear equation can be used to make predictions for new values of (x). The model recognizes patterns in the data and can predict corresponding values of (y).

## Code:

#Importing the libraries

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

#Importing the dataset

dataset = pd.read\_csv('Salary\_Data.csv')

X = dataset.iloc[:, :-1].values y = dataset.iloc[:, -1].values

#Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 1/3, random\_state = 0)

#Training the Simple Linear Regression model on the Training set from sklearn.linear\_model import LinearRegression

regressor = LinearRegression() regressor.fit(X\_train, y\_train)

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None, normalize=False) # Predicting the Test set results

y\_pred = regressor.predict(X\_test)

# Visualising the Training set results plt.scatter(X\_train, y\_train, color = 'red')

plt.plot(X\_train, regressor.predict(X\_train), color = 'blue') plt.title('Salary vs Experience (Training set)') plt.xlabel('Years of Experience')

plt.ylabel('Salary') plt.show()



# Visualising the Test set results plt.scatter(X\_test, y\_test, color = 'red')

plt.plot(X\_train, regressor.predict(X\_train), color = 'blue') plt.title('Salary vs Experience (Test set)') plt.xlabel('Years of Experience')

plt.ylabel('Salary') plt.show()



# Experiment No: 7(B)Logistic Regression

**Theory:** Logistic regression is a statistical method used for binary classification, where the target variable (dependent variable) is categorical with two possible outcomes. It's a type of regression analysis used when the dependent variable is binary. In logistic regression, the coefficients (parameters) are estimated using maximum likelihood estimation. The goal is to find the parameter values that maximize the likelihood of observing the given data.

## Procedure:

The process of multiple linear regression involves the following steps:

1. **Data Collection and Preprocessing**: Gather the dataset containing the input features and the binary outcome variable. Preprocess the data by handling missing values, encoding categorical variables, and scaling numerical features if necessary.
2. **Model Training:** Split the dataset into training and testing sets (or use cross-validation). Fit the logistic regression model to the training data. During training, the model estimates the coefficients that best fit the data.
3. **Model Evaluation**: Evaluate the performance of the trained model using the testing set. Common evaluation metrics for binary classification include accuracy, precision, recall, F1-score, and ROC-AUC.
4. **Model Interpretation**: Interpret the coefficients of the logistic regression model to understand the impact of each feature on the probability of the binary outcome. Positive coefficients indicate a positive association with the outcome, while negative coefficients indicate a negative association.
5. **Prediction**: Once the model is trained and evaluated, it can be used to make predictions on new data. Given a set of input features, the model predicts the probability of the binary outcome using the sigmoid function.
6. **Model Optimization**: Depending on the evaluation results, you might need to fine-tune the model hyperparameters or consider feature engineering to improve the model's performance.
7. **Deployment**: Deploy the trained logistic regression model into production for making real-time predictions.

## Code:

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('/content/Social\_Network\_Ads 1.csv')

X = dataset.iloc[:, :- 1].valuesy = dataset.iloc[:, -1].values

# Splitting the dataset into the Training set and Test set from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

#Feature Scaling

from sklearn.preprocessing import StandardScaler sc= StandardScaler()

X\_train= sc.fit\_transform(X\_train) X\_test=sc.tranform(X\_test)

# Training the Logistic Regression model on the Training set from sklearn.linear\_model import LogisticRegression classifier = LogisticRegression(random\_state = 0) classifier.fit(X\_train, y\_train)

# Predicting the Test set results y\_pred = classifier.predict(X\_test)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix, accuracy\_score cm = confusion\_matrix(y\_test, y\_pred)

accuracy\_score(y\_test, y\_pred)

.

# Visualising the Training set results

from matplotlib.colors import ListedColormap X\_set, y\_set = sc.inverse\_transform(X\_train), y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 0.25), np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 0.25)) plt.contourf(X1,X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

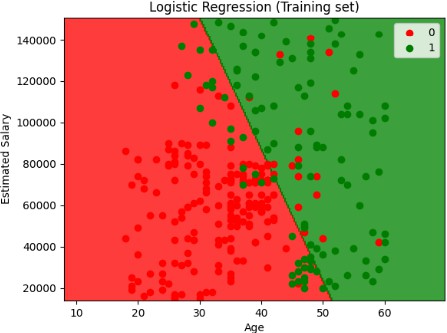
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j) plt.title('Logistic Regression (Training set)')

plt.xlabel('Age') plt.ylabel('Estimated Salary') plt.legend()

plt.show()



# Visualising the Test set results

from matplotlib.colors import ListedColormap X\_set, y\_set = sc.inverse\_transform(X\_test), y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 0.25), np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 0.25)) plt.contourf(X1,X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

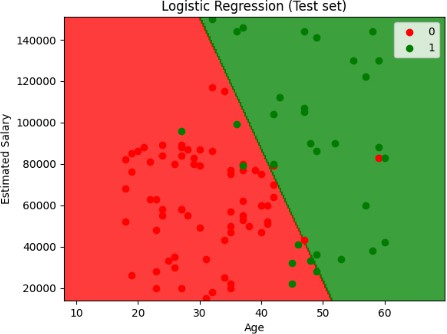
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j) plt.title('Logistic Regression (Test set)')

plt.xlabel('Age') plt.ylabel('Estimated Salary') plt.legend()

plt.show()



# Experiment No: 8

**Aim:** Implement K-Nearest Neighbours.

**Theory:** K-Nearest Neighbors (KNN) is a simple and intuitive algorithm used for classification and regression tasks. In KNN, the prediction for a new data point is based on the majority class (for classification) or the average value (for regression) of its k nearest neighbors in the feature space. The distance metric, typically Euclidean distance, determines the proximity of points.

During prediction, the algorithm calculates distances between the new data point and all training examples, selects the k nearest neighbors, and assigns the new point to the class most common among its neighbors. For regression, it calculates the average value of the target variable for the k nearest neighbors. KNN's simplicity makes it easy to understand and implement, but its performance can be sensitive to the choice of k and the distance metric. Cross-validation is often used to select optimal values for these parameters.

## Procedure:

1. **Data Collection and Preprocessing:** Collect and preprocess the dataset, handling missing values, encoding categorical variables, and scaling features if necessary. Split the dataset into training and testing sets.
2. **Choose k:** Determine the number of nearest neighbors, k, through experimentation or cross- validation.
3. **Calculate Distances:** Calculate distances (e.g., Euclidean distance) between each point in the testing set and all points in the training set.
4. **Find Nearest Neighbors:** Select the k nearest neighbors based on the calculated distances.
5. **Predict:** For classification, determine the majority class among the k nearest neighbors. For regression, calculate the average of the target variable values for the k nearest neighbors.
6. **Evaluate:** Assess the model's performance using metrics like accuracy (for classification) or mean squared error (for regression) on the testing set.
7. **Parameter Tuning:** Fine-tune k and explore different distance metrics, such as Manhattan distance, through techniques like grid search or cross-validation.
8. **Deployment:** Deploy the trained model for making predictions on new data.

## Code:

# Importing the libraries import numpy as np

import matplotlib.pyplot as plt import pandas as pd

from sklearn.model\_selection import train\_test\_split, GridSearchCV from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report from matplotlib.colors import ListedColormap

# Importing the dataset

dataset = pd.read\_csv('/content/Social\_Network\_Ads.csv')

X = dataset.iloc[:, :-1].values y = dataset.iloc[:, -1].values

# Splitting the dataset into the Training set and Test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=0)

# Feature Scaling

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train) X\_test = sc.transform(X\_test)

# Parameter tuning using Grid Search

parameters = {'n\_neighbors': [3, 5, 7, 9], 'metric': ['euclidean', 'manhattan']}

grid\_search=GridSearchCV(estimator=KNeighborsClassifier(), param\_grid=parameters, scoring='accuracy', cv=5)

grid\_search = grid\_search.fit(X\_train, y\_train) best\_accuracy = grid\_search.best\_score\_ best\_parameters = grid\_search.best\_params\_

# Training the K-NN model on the Training set using the best parameters

classifier=KNeighborsClassifier(n\_neighbors=best\_parameters['n\_neighbors'], metric=best\_parameters['metric'])

classifier.fit(X\_train, y\_train)

# Predicting the Test set results y\_pred = classifier.predict(X\_test)

# Evaluating the model

cm = confusion\_matrix(y\_test, y\_pred) accuracy = accuracy\_score(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred)

print("Confusion Matrix:") print(cm)

print("Accuracy:", accuracy) print("Classification Report:") print(classification\_rep)

# Visualizing the decision boundary (optional) def plot\_decision\_boundary(X, y, classifier, title):

X1, X2 = np.meshgrid(np.arange(start=X[:, 0].min() - 1, stop=X[:, 0].max() + 1, step=0.01),

np.arange(start=X[:, 1].min() - 1, stop=X[:, 1].max() + 1, step=0.01))

plt.contourf(X1,X2,classifier.predict(np.array([X1.ravel(),X2.ravel()]).T).reshape(X1.shape), alpha=0.75, cmap=ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y)):

plt.scatter(X[y == j, 0], X[y == j, 1], c=ListedColormap(('red', 'green'))(i), label=j) plt.title(title)

plt.xlabel('Age') plt.ylabel('Estimated Salary') plt.legend()

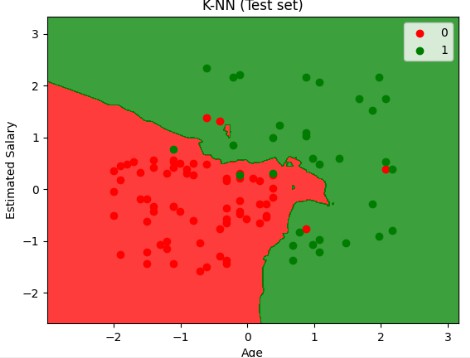
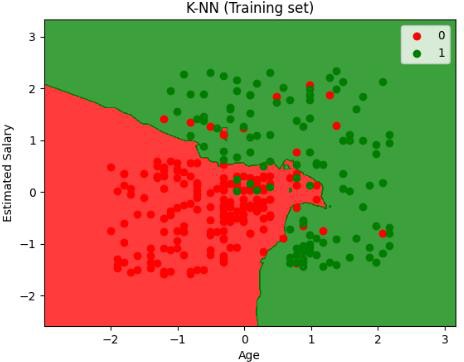
plt.show()

# Visualizing the Training set results

plot\_decision\_boundary(X\_train, y\_train, classifier, 'K-NN (Training set)')

# Visualizing the Test set results

plot\_decision\_boundary(X\_test, y\_test, classifier, 'K-NN (Test set)')



# Experiment No: 9(A)Decision Trees

**Aim:** To Implement Decision Trees and Random Forest Classifier.

**Theory:**Decision tree is a popular machine learning algorithm used for both classification and regression tasks. It recursively partitions the input space into regions based on feature values, creating a tree-like structure where each internal node represents a decision based on a feature, and each leaf node represents the predicted outcome. Decision trees are easy to interpret and understand, making them particularly useful for exploratory data analysis. They can handle both numerical and categorical data and can capture complex decision boundaries. However, they are prone to overfitting, especially with deep trees, which can be mitigated through techniques like pruning and ensemble methods.

## Procedure:

1. **Data Collection and Preprocessing:** Gather the dataset containing the input features and the target variable. Preprocess the data by handling missing values, encoding categorical variables, and scaling numerical features if necessary.
2. **Tree Construction:** Start with the root node and recursively split the data into subsets based on the feature that best separates the classes (for classification) or minimizes the variance of the target variable (for regression). This process continues until one of the stopping criteria is met, such as reaching a maximum depth, having minimum samples per leaf, or no further improvement in purity/impurity measure.
3. **Stopping Criteria:** Define the conditions for when to stop splitting, such as reaching a maximum tree depth, having a minimum number of samples in a node, or when further splitting does not significantly improve the model's performance.
4. **Splitting Criteria:** Choose the best feature and threshold for splitting at each node, typically based on measures like Gini impurity or entropy (for classification) or variance reduction (for regression).
5. **Tree Evaluation:** Assess the performance of the decision tree using metrics such as accuracy, precision, recall, F1-score (for classification), or mean squared error, R-squared (for regression) on a separate validation set or through cross-validation.
6. **Parameter Tuning:** Fine-tune hyperparameters such as maximum depth, minimum samples per leaf, or the splitting criterion to optimize the model's performance. This can be done through techniques like grid search or random search.
7. **Deployment:** Once the decision tree model is trained and evaluated satisfactorily, deploy it into production for making predictions on new, unseen data.

## Code:

# Importing the libraries import numpy as np

import matplotlib.pyplot as plt import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('/content/Social\_Network\_Ads.csv')

X = dataset.iloc[:, [2, 3]].values y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train) X\_test = sc.transform(X\_test)

# Fitting Decision Tree Classification to the Training set from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier(criterion = 'entropy', random\_state = 0) classifier.fit(X\_train, y\_train)

# Predicting the Test set results y\_pred = classifier.predict(X\_test)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix cm = confusion\_matrix(y\_test, y\_pred)

# Visualising the Training set results

from matplotlib.colors import ListedColormap X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01), np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Decision Tree Classification (Training set)') plt.xlabel('Age')

plt.ylabel('Estimated Salary') plt.legend() plt.show()

# Visualising the Test set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

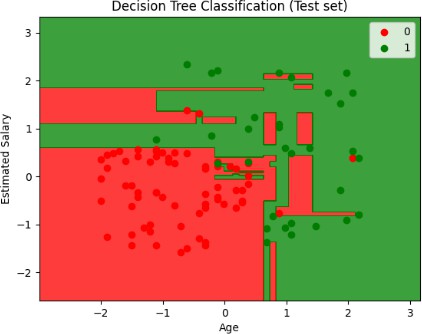
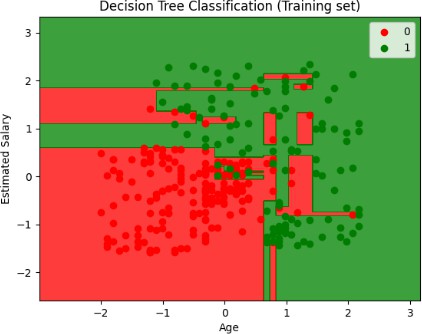
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j) plt.title('Decision Tree Classification (Test set)')

plt.xlabel('Age') plt.ylabel('Estimated Salary') plt.legend()

plt.show()



# Experiment No: 9(B)Random Forest

## Theory:

Random Forest is an ensemble learning method that combines the predictions of multiple individual decision trees to improve predictive accuracy and reduce overfitting. It constructs a multitude of decision trees during training, each tree being trained on a random subset of the data and using a random subset of features. The final prediction is then made by averaging or taking a vote among the predictions of all the trees. Random Forest is highly robust and effective for both classification and regression tasks, capable of handling large datasets with high-dimensional feature spaces, and provides estimates of feature importance.

## Procedure:

1. **Data Collection and Preprocessing**: Gather the dataset containing the input features and the target variable. Preprocess the data by handling missing values, encoding categorical variables, and scaling numerical features if necessary. Split the dataset into training and testing sets.
2. **Bootstrapping:** Randomly sample subsets of the training data with replacement to create multiple bootstrap samples. Each bootstrap sample is used to train an individual decision tree.
3. **Tree Construction:** For each bootstrap sample, construct a decision tree using a subset of features randomly selected at each node. Grow each tree to its maximum depth without pruning.
4. **Voting or Averaging:** For classification tasks, aggregate the predictions of all decision trees by either taking a majority vote (for classification) or averaging the predicted probabilities (for probability estimates). For regression tasks, take the average of the predicted values from all decision trees.
5. **Model Evaluation:** Assess the performance of the Random Forest model using metrics such as accuracy, precision, recall, F1-score (for classification), or mean squared error, R-squared (for regression) on the testing set or through cross-validation.
6. **Parameter Tuning**: Fine-tune hyperparameters such as the number of trees (n\_estimators), maximum depth of trees, minimum samples per leaf, and the number of features considered at each split to optimize the model's performance. This can be done through techniques like grid search or random search.
7. **Feature Importance:** Analyze the importance of features in the Random Forest model by examining the decrease in node impurity (e.g., Gini impurity or entropy) or mean decrease in accuracy when a feature is removed.
8. **Deployment:** Once the Random Forest model is trained and evaluated satisfactorily, deploy it into production for making predictions on new, unseen data.

## Code:

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('/content/Social\_Network\_Ads.csv')

X = dataset.iloc[:, [2, 3]].values y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train) X\_test = sc.transform(X\_test)

# Fitting Random Forest Classification to the Training set from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n\_estimators = 10, criterion = 'entropy', random\_state = 0) classifier.fit(X\_train, y\_train)

# Predicting the Test set results y\_pred = classifier.predict(X\_test)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix cm = confusion\_matrix(y\_test, y\_pred)

# Visualising the Training set results

from matplotlib.colors import ListedColormap X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01), np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

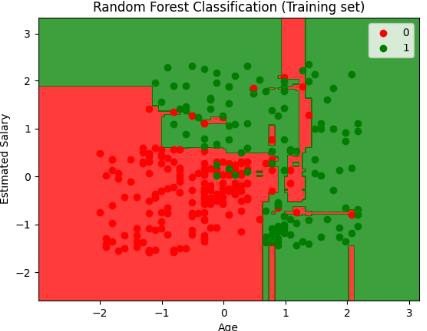
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max()) plt.ylim(X2.min(), X2.max()) for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j) plt.title('Random Forest Classification (Training set)')

plt.xlabel('Age') plt.ylabel('Estimated Salary') plt.legend()

plt.show()



# Visualising the Test set results

from matplotlib.colors import ListedColormap X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01), np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

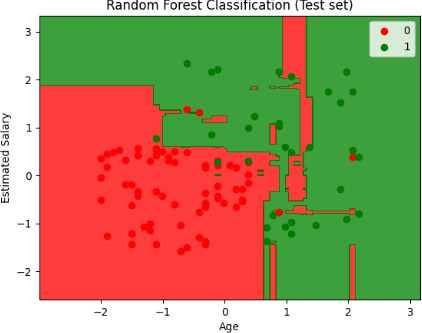
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j) plt.title('Random Forest Classification (Test set)')

plt.xlabel('Age') plt.ylabel('Estimated Salary') plt.legend()

plt.show()



# Experiment No: 10

**Aim:** To implement Support Vector Machine.

## Theory:

Support Vector Machine (SVM) is a powerful supervised learning algorithm used for both classification and regression tasks. It works by finding the optimal hyperplane that best separates classes in the feature space while maximizing the margin between the classes. SVM can handle linear and non-linear decision boundaries using different kernel functions. It is effective in high- dimensional spaces, robust against overfitting, and is particularly useful when dealing with small to medium-sized datasets.

## Procedure:

1. **Data Collection and Preprocessing:** Gather the dataset containing the input features and the target variable. Preprocess the data by handling missing values, encoding categorical variables, and scaling numerical features if necessary. Split the dataset into training and testing sets.
2. **Selecting the Kernel and Parameters:** Choose the appropriate kernel function (linear, polynomial, radial basis function, etc.) and tune the hyperparameters (e.g., C, gamma for RBF kernel) based on cross-validation or grid search to optimize the SVM's performance.
3. **Model Training:** Train the SVM model on the training data by finding the hyperplane that best separates the classes while maximizing the margin. This involves solving the optimization problem to find the optimal separating hyperplane or decision boundary.
4. **Model Evaluation:** Assess the performance of the SVM model using metrics such as accuracy, precision, recall, F1-score (for classification) or mean squared error, R-squared (for regression) on the testing set.
5. **Parameter Tuning:** Fine-tune hyperparameters such as the kernel type, regularization parameter (C), and kernel parameters (e.g., gamma for RBF kernel) to optimize the model's performance. This can be done through techniques like grid search or random search.
6. **Handling Imbalanced Data (Optional):** If dealing with imbalanced classes, consider techniques like class weighting or using different performance metrics (e.g., AUC-ROC) to account for class imbalance.
7. **Deployment:** Once the SVM model is trained and evaluated satisfactorily, deploy it into production for making predictions on new, unseen data.

## Code:

# Importing the libraries import numpy as np

import matplotlib.pyplot as plt import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('/content/Social\_Network\_Ads - Social\_Network\_Ads.csv')

X = dataset.iloc[:, [2, 3]].values y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train) X\_test = sc.transform(X\_test)

# Fitting classifier to the Training set # Create your classifier here classifier=svm.SVC(kernel='linear') classifier.fit(X\_train,y\_train)

# Predicting the Test set results y\_pred = classifier.predict(X\_test)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix cm = confusion\_matrix(y\_test, y\_pred)

# Visualising the Training set results

from matplotlib.colors import ListedColormap X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01), np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

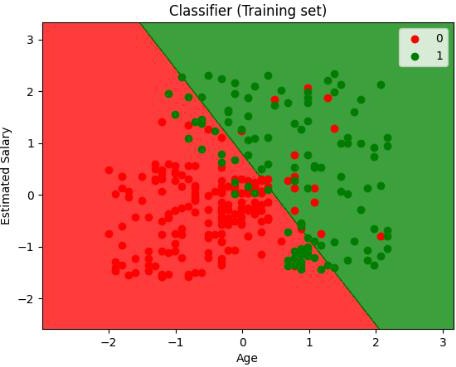
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j) plt.title('Classifier (Training set)')

plt.xlabel('Age') plt.ylabel('Estimated Salary') plt.legend()

plt.show()



# Visualising the Test set results

from matplotlib.colors import ListedColormap X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01), np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

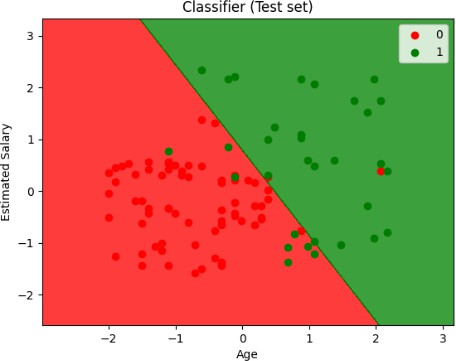
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j) plt.title('Classifier (Test set)')

plt.xlabel('Age') plt.ylabel('Estimated Salary') plt.legend()

plt.show()



# Experiment No: 11

**Aim:** To implement Naive Bayes Classifier.

## Theory:

Naive Bayes Classifier is a probabilistic machine learning algorithm based on Bayes' theorem. It assumes that features are conditionally independent given the class label, simplifying the calculation of the posterior probability. Naive Bayes is particularly effective for text classification tasks, such as spam filtering and sentiment analysis, due to its simplicity, scalability, and ability to handle high- dimensional data efficiently. Despite its simplicity and the "naive" assumption of feature independence, Naive Bayes often performs well in practice and serves as a strong baseline for many classification tasks.

## Procedure:

1. **Data Collection and Preprocessing:** Gather the dataset containing the input features and the target variable. Preprocess the data by handling missing values, encoding categorical variables, and performing feature scaling if necessary. For text classification tasks, feature extraction techniques like TF-IDF or Bag-of-Words are commonly used.
2. **Model Training:** Calculate the prior probabilities and conditional probabilities of each feature given the class label from the training data using Bayes' theorem. For continuous features, probability density functions (such as Gaussian distribution for Gaussian Naive Bayes) are estimated.
3. **Model Evaluation:** Assess the performance of the Naive Bayes Classifier using metrics such as accuracy, precision, recall, F1-score, or ROC-AUC on a separate validation set or through cross- validation.
4. **Handling Zero Probabilities:** Apply techniques like Laplace smoothing (additive smoothing) to avoid zero probabilities for features not observed in the training data.
5. **Model Interpretation:** Interpret the learned probabilities to understand the importance of different features in the classification decision.
6. **Parameter Tuning:** Depending on the variant of Naive Bayes used (e.g., Gaussian Naive Bayes, Multinomial Naive Bayes, Bernoulli Naive Bayes), fine-tune hyperparameters such as smoothing parameter (alpha) or feature distribution assumptions to optimize the model's performance.
7. **Deployment:** Once the Naive Bayes Classifier is trained and evaluated satisfactorily, deploy it into production for making predictions on new, unseen data.

## Code:

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('/content/Social\_Network\_Ads.csv')

X = dataset.iloc[:, [2, 3]].values y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler s c = StandardScaler()

X\_train = sc.fit\_transform(X\_train) X\_test = sc.transform(X\_test)

# Fitting Naive Bayes to the Training set from sklearn.naive\_bayes import GaussianNB classifier = GaussianNB() classifier.fit(X\_train, y\_train)

# Predicting the Test set results y\_pred = classifier.predict(X\_test)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix cm = confusion\_matrix(y\_test, y\_pred)

# Visualising the Training set results

from matplotlib.colors import ListedColormap X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01), np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

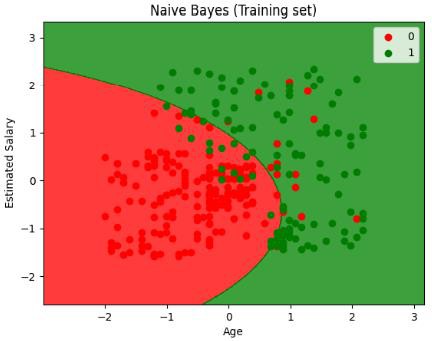
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j) plt.title('Naive Bayes (Training set)')

plt.xlabel('Age') plt.ylabel('Estimated Salary') plt.legend()

plt.show()



# Visualising the Test set results

from matplotlib.colors import ListedColormap X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01), np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

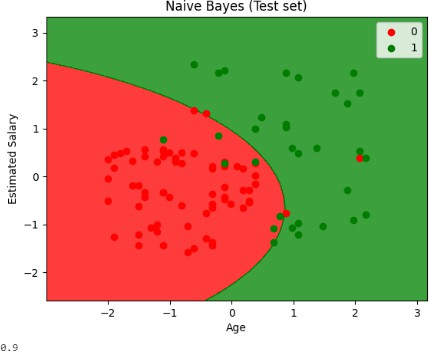
for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j) plt.title('Naive Bayes (Test set)')

plt.xlabel('Age') plt.ylabel('Estimated Salary') plt.legend()

plt.show()

from sklearn.metrics import accuracy\_score score = accuracy\_score(y\_test,y\_pred) print(score)



# Experiment No: 12

**Aim:** To implement K-Means clustering.

## Theory:

K-means clustering is an unsupervised machine learning algorithm used for partitioning a dataset into a predetermined number of clusters. It aims to minimize the within-cluster variance by iteratively assigning data points to the nearest cluster centroid and updating the centroids based on the mean of the points assigned to each cluster. The algorithm converges when the centroids no longer change significantly or after a specified number of iterations. K-means is widely used for clustering applications due to its simplicity, scalability, and efficiency, although it may be sensitive to the initial choice of centroids and can converge to local optima.

## Procedure:

1. **Initial Centroid Selection:** Choose the number of clusters (k) and randomly initialize k centroids within the feature space. Alternatively, k centroids can be selected based on a heuristic method or using k-means++ initialization to improve convergence.
2. **Assign Data Points to Clusters:** For each data point in the dataset, calculate the distance to each centroid and assign the point to the nearest centroid's cluster.
3. **Update Centroids:** Recalculate the centroids of each cluster by taking the mean of all data points assigned to that cluster.
4. **Repeat Assignment and Update:** Iterate steps 2 and 3 until convergence criteria are met. Convergence criteria can include a maximum number of iterations, a small change in centroids between iterations, or when no data points change clusters.
5. **Evaluate Clustering:** Assess the quality of the clustering solution using metrics such as the within-cluster sum of squares (WCSS) or silhouette score. These metrics can help determine the optimal number of clusters and evaluate the clustering's compactness and separation.
6. **Refinement (Optional**): Depending on the application, refine the clustering solution by adjusting the number of clusters, reinitializing centroids, or experimenting with different distance metrics.
7. **Interpretation and Visualization:** Interpret the resulting clusters and their centroids to gain insights into the underlying structure of the data. Visualize the clusters in the feature space using scatter plots or other visualization techniques.
8. **Deployment:** Once satisfied with the clustering solution, deploy the model to assign new data points to clusters in real-world applications.

## Code:

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('/content/Mall\_Customers.csv')

X = dataset.iloc[:, [3, 4]].values y = dataset.iloc[:, 3].values

# Splitting the dataset into the Training set and Test set from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0) y\_train = y\_train.reshape(-1, 1)

# Feature Scaling

from sklearn.preprocessing import StandardScaler sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train) X\_test = sc\_X.transform(X\_test)

sc\_y = StandardScaler()

y\_train = sc\_y.fit\_transform(y\_train)

# Using the elbow method to find the optimal number of clusters from sklearn.cluster import KMeans

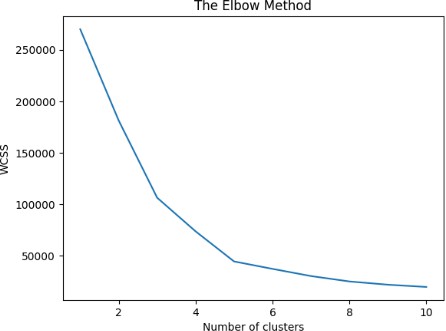
wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 42) kmeans.fit(X)

wcss.append(kmeans.inertia\_) plt.plot(range(1, 11), wcss) plt.title('The Elbow Method') plt.xlabel('Number of clusters') plt.ylabel('WCSS')

plt.show()



# Fitting K-Means to the dataset

kmeans = KMeans(n\_clusters = 5, init = 'k-means++', random\_state = 42) y\_kmeans = kmeans.fit\_predict(X)

# Visualising the clusters

plt.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')

plt.scatter(X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')

plt.scatter(X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')

plt.scatter(X[y\_kmeans == 3, 0], X[y\_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

plt.scatter(X[y\_kmeans == 4, 0], X[y\_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 300, c = 'yellow', label = 'Centroids')

plt.title('Clusters of customers') plt.xlabel('Annual Income (k$)') plt.ylabel('Spending Score (1-100)') plt.legend()

plt.show()

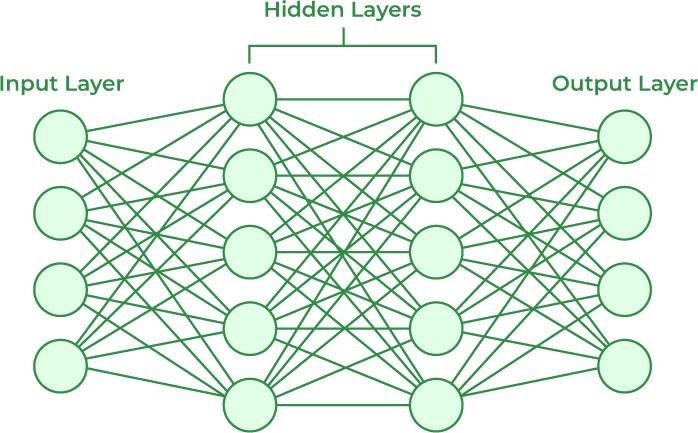


# Experiment No: 13

**Aim:** ANN (Artificial Neural Networks): A Practical Implementation in Python.

## Theory:

Artificial Neural Networks contain artificial neurons which are called units. These units are arranged in a series of layers that together constitute the whole Artificial Neural Network in a system. A layer can have only a dozen units or millions of units as this depends on how the complex neural networks will be required to learn the hidden patterns in the dataset. Commonly, Artificial Neural Network has an input layer, an output layer as well as hidden layers. The input layer receives data from the outside world which the neural network needs to analyze or learn about. Then this data passes through one or multiple hidden layers that transform the input into valuable data for the output layer. Finally, the output layer provides an output in the formof a response of the Artificial Neural Networks to input data provided. In the majority of neural networks, units are interconnected from one layer to another. Each of these connections has weights that determine the influence of one unit on another unit. As the data transfers from one unit to another, the neural network learns more and more about the data which eventually results in an output from the output layer.



The structures and operations of human neurons serve as the basis for artificial neural networks. It is also known as neural networks or neural nets. The input layer of an artificial neural network is the first layer, and it receives input from external sources and releases it to the hidden layer, which is the second layer. In the hidden layer, each neuron receives input from the previous layer neurons, computes the weighted sum, and sends it to the neurons in the next layer.

## Code:

import numpy as np import pandas as pd

forest=pd.read\_csv('forestfires.csv') forest.head()

forest.pop('month') forest.head()

forest.pop('day') forest.head()

forest.isna().sum()

from sklearn import preprocessing label\_encoder=preprocessing.LabelEncoder() forest['size\_category']=label\_encoder.fit\_transform(forest['size\_category']) forest.head()

x=forest.iloc[:,0:28] y=forest.iloc[:,28]

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=0)

from sklearn.preprocessing import StandardScaler sc=StandardScaler() x\_train=sc.fit\_transform(x\_train) x\_test=sc.fit\_transform(x\_test)

#Neural Network - ANN

from keras.models import Sequential from keras.layers import Dense

from keras.layers import Dropout

#initializing ANN model=Sequential()

#adding input and 1st hidden layer model.add(Dense(units=10,activation='relu',kernel\_initializer='he\_uniform',input\_dim=28)) #adding 2nd hidden layer model.add(Dense(units=8,activation='relu',kernel\_initializer='he\_uniform'))

#adding output layer model.add(Dense(units=1,kernel\_initializer='glorot\_uniform',activation='sigmoid'))

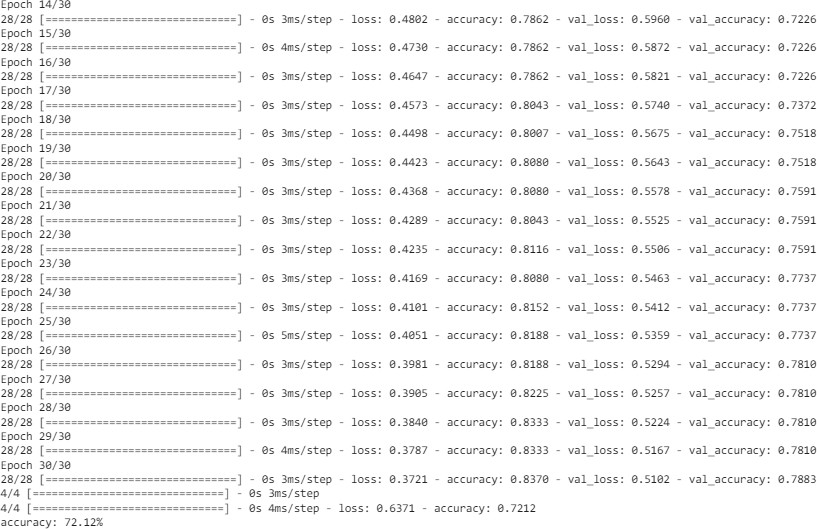
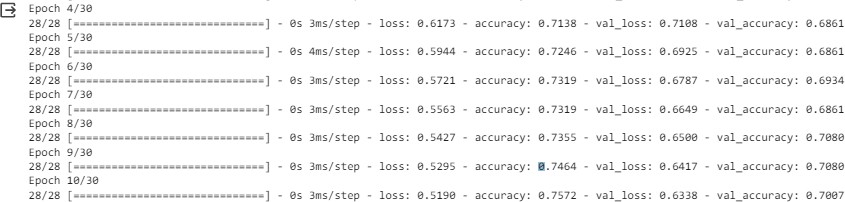
#compile the model model.compile(optimizer='adam',loss='binary\_crossentropy',metrics=['accuracy']) #fit the model model.fit(x\_train,y\_train,batch\_size=10,epochs=30,validation\_split=0.33) y\_pred=model.predict(x\_test)

y\_pred

#evaluating the model

scores = model.evaluate(x\_test, y\_test)

print("%s: %.2f%%" % (model.metrics\_names[1], scores[1]\*100))



# Experiment No: 14

**Aim:** CNN (Convolution Neural Networks): A Practical Implementation in Python.

**Theory:**

A Convolutional Neural Network (CNN) is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data.

When it comes to Machine Learning, [Artificial Neural Networks](https://www.geeksforgeeks.org/implementing-ann-training-process-in-python/) perform really well. Neural Networks are used in various datasets like images, audio, and text. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use [Recurrent Neural Networks](https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/) more precisely an [LSTM](https://www.geeksforgeeks.org/understanding-of-lstm-networks/), similarly for image classification we use Convolution Neural networks. In this blog, we are going to build a basic building block for CNN.

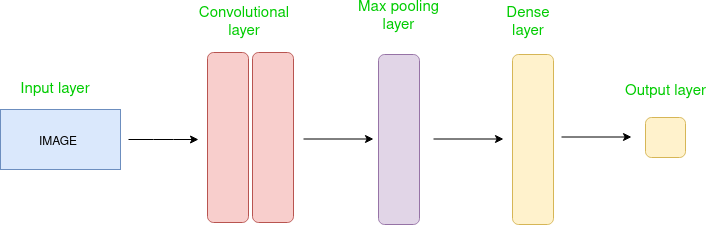
## In a regular Neural Network, there are three types of layers:

1. **Input Layers:** It’s the layer where we give input to our model. The number of neuronsin this layer is equal to the total number of features in our data (number of pixels in the case of an image).
2. **Hidden Layer:** The input from the Input layer is then fed into the hidden layer. There can be many hidden layers depending on our model and data size. Each hidden layer can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of the output of the previous layer with learnable weights of that layer and then by the addition of learnable biases followed by activation function which makes the network nonlinear.
3. **Output Layer:** The output from the hidden layer is then fed into a logistic function like sigmoid or softmax which converts the output of each class into the probability score of each class.

The data is fed into the model and output from each layer is obtained from the above step is called **[feedforward](https://www.geeksforgeeks.org/understanding-multi-layer-feed-forward-networks/)**[,](https://www.geeksforgeeks.org/understanding-multi-layer-feed-forward-networks/) we then calculate the error using an error function, some common error functions are cross-entropy, square loss error, etc. The error function measures how well the network is performing. After that, we backpropagate into the model by calculating the derivatives. This step is called **[Backpropagation](https://www.geeksforgeeks.org/backpropagation-in-data-mining/)** which is used to minimize the lo3

## CNN Architecture:

Convolutional Neural Network consists of multiple layers like the input layer, Convolutional layer, Pooling layer, and fully connected layers.



The Convolutional layer applies filters to the input image to extract features, the Pooling layer downsamples the image to reduce computation, and the fully connected layer makes the final prediction. The network learns the optimal filters through back propagation and gradient descent.

## Advantages of Convolutional Neural Networks (CNNs):

1. Good at detecting patterns and features in images, videos, and audio signals.
2. Robust to translation, rotation, and scaling invariance.
3. End-to-end training, no need for manual feature extraction.
4. Can handle large amounts of data and achieve high accuracy.

## Disadvantages of Convolutional Neural Networks (CNNs):

1. It is computationally expensive to train and requires a lot of memory.
2. Can be prone to overfitting if not enough data or proper regularization is used.
3. It requires large amounts of labeled data.
4. Interpretability is limited, it’s hard to understand what the network has learned.

## Code:

from google.colab import drive drive.mount('/content/drive') Mounted at /content/drive

#importing the libraries import keras

from keras.models import Sequential from keras.layers import Convolution2D from keras.layers import MaxPooling2D from keras.layers import Flatten

from keras.layers import Dense

classifier= Sequential() # Initialise the CNN

# Ist step of Convoltional layer to get feature maps using feature detector classifier.add(Convolution2D(filters=32, # output feature maps kernel\_size=(3,3), # matrix size for feature detector

input\_shape=(64, 64, 3), # input image shape, 3 is for rgb

coloured kernel\_initializer='he\_uniform', # weights distriution activation='relu')) # activation function

# 2nd Pooling layer classifier.add(MaxPooling2D(pool\_size=(2,2))) classifier.add(Convolution2D(filters=32,

kernel\_size=(3,3), kernel\_initializer='he\_uniform', activation='relu')) classifier.add(MaxPooling2D(pool\_size=(2,2)))

# Step 3 - Flattening classifier.add(Flatten())

#Step 4 full connection in which input we have from flattening classifier.add(Dense(units=128,kernel\_initializer='glorot\_uniform', activation='relu'))

#step 5 output layer classifier.add(Dense(units=1,kernel\_initializer='glorot\_uniform',activation='sigmoid'))

# Compiling the CNN

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy]

#Fitting the model

from keras.preprocessing.image import ImageDataGenerator #applying all the transformation we want to apply to training data set

train\_datagen = ImageDataGenerator(rescale = 1./255, shear\_range = 0.2, zoom\_range = 0.2, horizontal\_flip = True)

#Recycling the test data set images to use for validation. test\_datagen= ImageDataGenerator(rescale=1./255)

#Getting My training data ready for validation, so it will read all the data with the px size training\_set= train\_datagen.flow\_from\_directory(directory= '/content/drive/MyDrive/dataset/traini

target\_size=(64,64), # As we choose 64\*64 for our batch\_size=50, class\_mode='binary' # for 2 class binary )

Found 2118 images belonging to 2 classes.

#Getting My test data ready for validation, so it will read all the data with the px size

test\_set= test\_datagen.flow\_from\_directory(directory= '/content/drive/MyDrive/dataset/test\_set target\_size=(64,64), # As we choose 64\*64 for our batch\_size=50, class\_mode='binary' # for 2 class binary )

Found 351 images belonging to 2 classes.

classifier.fit\_generator(training\_set, #training data to fit steps\_per\_epoch=8000, # Data in training set epochs=5, # No of epochs to run validation\_data=test\_set, # Test or validation set validation\_steps=2000 # no of data point for validation )

1: UserWarning: `Model.fit\_generator` is deprecated and will classifier.fit\_generator(training\_set, #training data to fit Epoch 1/5 43/8000 [ ] - ETA: 15:53:22 - loss: 0.7943 - accuracy: 0.5274

WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your da8000/8000 [==============================] - 466s 57ms/step - loss: 0.7943 - accuracy: 0.527

# Part 3 - Making new predictions import numpy as np

from keras.preprocessing import image

test\_image = image.load\_img('/content/drive/MyDrive/dataset/test\_set/cats/cat.4793.jpg', target\_s # Loading the image and converting the pixels into array whcih will be used as input to predict. test\_image = image.img\_to\_array(test\_image)

test\_image = np.expand\_dims(test\_image, axis = 0) result = classifier.predict(test\_image) training\_set.class\_indices

if result[0][0] == 1:

prediction = 'dog'

else:

prediction = 'cat'

1/1 [==============================] - 0s 166ms/step