TWITTER SENTIMENT ANALYSIS

A PROJECT REPORT

In partial fulfilment of the requirements for the award of the degree

MASTERS

IN

COMPUTER APPLICATIONS

Under the guidance of

MAHENDRA DATTA

BY

NIKHIL ARORA

HERITAGE INSTITUTE OF TECHNOLOGY.

In association with

(ISO9001:2015)

SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V, Kolkata, West Bengal 700091

(Note: All entries of the proforma of approval should be filled up with appropriate and complete information. Incomplete proforma of approval in any respect will be summarily rejected.)

1.	Title of the Project:	**TWITTER SENTIMENT ANALYSIS**
2.	Project Members:	**NIKHIL ARORA**
3.	Name of the guide:	**Mr. MAHENDRA DUTTA**
4.	Address:	Ardent Computech Pvt. Ltd (An ISO 9001:2015 Certified) SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V, Kolkata, West Bengal, 700091.

Project Version Control History

Version	Primary Author	Description of Version	Date Completed
FINAL	NIKHIL ARORA	Project Report	7TH APRIL,2020

Nikhil Arora.

Signature of Team Member

Date:

Signature of Approver

Date:

MR. MAHENDA DUTTA

Approved	**Not Approved**

DECLARATION

We hereby declare that the project work being presented in the project proposal entitled **“TWITTER SENTIMENT ANALYSIS”** in partial fulfilment of the requirements for the award of the degree of **MASTER OF COMPUTER APPLICATIONS** at **ARDENT COMPUTECH PVT. LTD, SALTLAKE, KOLKATA, WEST BENGAL,** is an authentic work carried out under the guidance of **MR. MAHENDRA DUTTA**. The matter embodied in this project work has not been submitted elsewhere for the award of any degree of our knowledge and belief.

Date:

Name of the Student: NIKHIL ARORA

Nikhil Arora.

Signature of the students:

Ardent Computech Pvt. Ltd (An ISO 9001:2015 Certified)

SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V, Kolkata, West Bengal 700091

<u>CERTIFICATE</u>

This is to certify that this proposal of minor project entitled **"TWITTER SENTIMENT ANALYSIS"** is a record of Bonafide work, carried out by ***NIKHIL ARORA*** under my guidance at **ARDENT COMPUTECH PVT LTD**. The report in its present form is in partial fulfilment of the requirements for the award of the degree of **MASTER OF COMPUTER APPLICATIONS** as per regulations of the **ARDENT®.** To the best of my knowledge, the results embodied in this report, are original in nature and worthy of incorporation in the present version of the report.

Guide / Supervisor

MR. MAHENDRA DUTTA

Project Engineer

Ardent Computech Pvt. Ltd (An ISO 9001:2015 Certified)

SDF Building, Module #132, Ground Floor, Salt Lake City, GP Block, Sector V, Kolkata, West Bengal 70repor

ACKNOWLEDGEMENT

Success of any project depends on the encouragement and guidelines of many others. I take this sincere opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project work.

I would like to show our greatest appreciation to ***Mr. MAHENDRA DUTTA***, Project Engineer at Ardent, Kolkata. I always feel motivated and encouraged every time by his valuable advice and constant inspiration; without his encouragement and guidance this project would not have materialized.

Words are inadequate in offering our thanks to the other trainees, project assistants and other members at Ardent Computech Pvt. Ltd. for their encouragement and cooperation in carrying out this project work. The guidance and support received from all the members and who are contributing to this project, was vital for the success of this project.

CONTENTS

OVERVIEW

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and has fewer syntactical constructions than other languages.

Python is interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like Perl and PHP.

Python is Interactive: You can set at a Python prompt and interact with the interpreter directly to write your programs.

Python is Object-Oriented: Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

Python is a Beginner's Language: Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

HISTORY OF PYTHON

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands. Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Small Talk, UNIX shell, and other scripting languages. Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL). Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

FEATURES OF PYTHON

1. **Easy-to-learn:** Python has few Keywords, simple structure, and clearly defined syntax. This allows a student to pick up the language quickly.

2. **Easy-to-Read:** Python code is more clearly defined and visible to the eyes.

3. **Easy -to-Maintain:** Python's source code is fairly easy-to-maintain.

4. **A broad standard library:** Python's bulk of the library is very portable and cross platform compatible on UNIX, Windows, and Macintosh.

5. **Interactive Mode**: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
6. **Portable:** Python can run on the wide variety of hardware platforms and has the same interface on all platforms.
7. **Extendable:** You can add low level modules to the python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

8. **Databases**: Python provides interfaces to all major commercial databases.

9. **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

10. **Scalable**: Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below:

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte code for building large applications.

- It provides very high-level dynamic datatypes and supports dynamic type checking.

- It supports automatic garbage collections.

BASIC SYNTAX OF PYTHON PROGRAM

Type the following text at the Python prompt and press the Enter –

```
>>> print "Hello, Python!"
```

If you are running latest version of Python, then you would need to use print statement with parenthesis as in **print ("Hello, Python!");**
However, in Python version 2.4.3, this produces the following result –

```
Hello, Python!
```

- ## Python Identifiers

A Python identifier is a name used to identify a variable, function, class, module, or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language.

- ## Python Keywords

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

and	**exec**	**not**	**assert**	**finally**
or	**break**	**for**	**pass**	**Class**
from	**print**	**continue**	**global**	**raise**
def	**if**	**return**	**del**	**import**
try	**elif**	**in**	**while**	**else**
is	**with**	**except**	**lambda**	**yield**

- <u>Lines & Indentation</u>

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced. The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example –

```
if True:
        print "True"
else:
        print "False"
```

- <u>Command Line Arguments</u>

Many programs can be run to provide you with some basic information about how they should be run. Python enables you to do this with -h –

```
$ python-h

usage: python [option]... [-c cmd|-m mod | file |-][arg]...

Options and arguments (and corresponding environment variables):

-c cmd: program passed in as string (terminates option list)

-d      : debug output from parser (also PYTHONDEBUG=x)

-E      : ignore environment variables (such as PYTHONPATH)

-h     : print this help message and exit [ etc.]
```

VARIABLE TYPES

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

- **Assigning Values to Variables**

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

```
counter=10          # An integer assignment
weight=10.60        # A floating point
name="Ardent"       # A string
```

- **Multiple Assignment**

Python allows you to assign a single value to several variables simultaneously. For example:

```
a = b = c = 1
a,b,c = 1,2,"hello"
```

- **Standard Data Types**

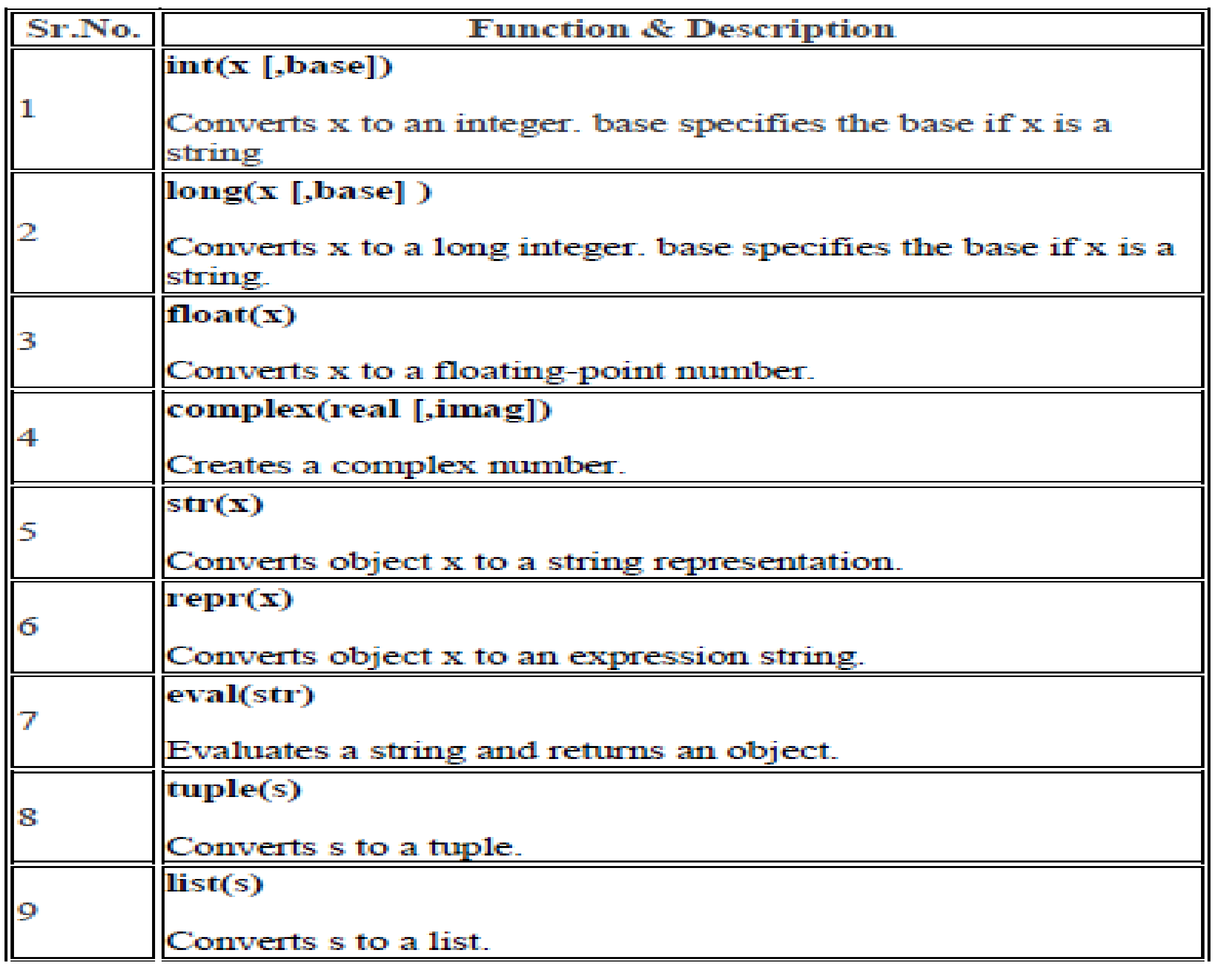
The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has five standard data types −

- String
- List
- Tuple
- Dictionary
- Number

- **Data Type Conversion**

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function.

There are several built-in functions to perform conversion from one data type to another.



Sr.No.	Function & Description
1	**int(x [,base])** Converts x to an integer. base specifies the base if x is a string
2	**long(x [,base])** Converts x to a long integer. base specifies the base if x is a string.
3	**float(x)** Converts x to a floating-point number.
4	**complex(real [,imag])** Creates a complex number.
5	**str(x)** Converts object x to a string representation.
6	**repr(x)** Converts object x to an expression string.
7	**eval(str)** Evaluates a string and returns an object.
8	**tuple(s)** Converts s to a tuple.
9	**list(s)** Converts s to a list.

FUNCTIONS

- **Defining a Function**

```
                    def function name (parameters ):
                                "function_docstring"
                                function suite
                                return [expression]
```

- **Pass by reference vs Pass by value**

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects in the calling function. For example –

```
# Function definition is here

def change me(mylist):
        "This changes a passed list into this function"
        mylist.append([1,2,3,4]);
        print"Values inside the function: ",mylist
        return

# Now you can call changeme function

mylist=[10,20,30];
change me(mylist);
print" Values outside the function: ",mylist
```

Here, we are maintaining reference of the passed object and appending values in the same object. So, this would produce the following result −

```
Values inside the function: [10, 20, 30, [1, 2, 3, 4]]
Values outside the function: [10, 20, 30, [1, 2, 3, 4]]
```

- **Global vs. Local variables**

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope. For Example-

```
total=0;                # This is global variable.

# Function definition is here

def sum (arg1, arg2):

# Add both the parameters and return them."

total= arg1 + arg2;   # Here total is local variable.
print"Inside the function local total: ", total
return total;

# Now you can call sum function

sum (10,20);
Print"Outside the function global total: ", total
```

When the above code is executed, it produces the following result –

```
Inside the function local total: 30
Outside the function global total: 0
```

MODULES

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.

The Python code for a module named *aname* normally resides in a file named *aname.py*. Here is an example of a simple module, support.py

```
def print_func( par ):
    print"Hello : ", par
    return
```

The *import* Statement

You can use any Python source file as a module by executing an import statement in some other Python source file. The *import* has the following syntax –

```
Import module1 [, module2 [… moduleN]
```

PACKAGES

A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and sub packages and sub-subpackages, and so on.

Consider a file *Pots.py* available in *Phone* directory. This file has following line of source code −

```
def Pots ():
                        print "I'm Pots Phone"
```

Similar way, we have another two files having distinct functions with the same name as above –

Phone/Isdn.py file having function Isdn ()
Phone/G3.py file having function G3 ()

Now, create one more file __init__.py in *Phone* directory −

Phone/__init__.py

To make all your functions available when you have imported Phone, you need to put explicit import statements in __init__.py as follows −

```
from Pots import Pots
from Isdn import Isdn
from G3 import
```

INTRODUCTION TO MACHINE LEARNING

Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed.

Arthur Samuel, an American pioneer in the field of computer gaming and artificial intelligence, coined the term "Machine Learning" in 1959 while at IBM. Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data

Machine learning tasks are typically classified into two broad categories, depending on whether there is a learning "signal" or "feedback" available to a learning system: -

- **SUPERVISED LEARNING**

Supervised learning is the machine learning task of inferring a function from *labelled training data*.[1] The training data consist of a set of *training examples*. In supervised learning, each example is a *pair* consisting of an input object (typically a vector) and a desired output value.

A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way.

- **UNSUPERVISED LEARNING**

Unsupervised learning is the machine learning task of inferring a function to describe hidden structure from "unlabelled" data (a classification or categorization is not included in the observations). Since the examples given to the learner are unlabelled, there is no evaluation of the accuracy of the structure that is output by the relevant algorithm—which is one way of distinguishing unsupervised learning from supervised learning and reinforcement learning.

A central case of unsupervised learning is the problem of density estimation in statistics, though unsupervised learning encompasses many other problems (and solutions) involving summarizing and explaining key features of the data.

- **NUMPY**

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with an enormous collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin.

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs if most operations work on arrays or matrices instead of scalars.

- **NUMPY ARRAY**

NumPy’s main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all the same type, indexed by a tuple of positive integers. In NumPy dimensions are called *axes*. The number of axes is *rank*.

For example, the coordinates of a point in 3D space [1, 2, 1] is an array of rank 1, because it has one axis. That axis has a length of 3. In the example pictured below, the array has rank 2 (it is 2-dimensional). The first dimension (axis) has a length of 2, the second dimension has a length of 3.

```
[[1., 0., 0.],
 [ 0., 1., 2.]]
```

NumPy’s array class is called ***ndarray***.

- **SLICING NUMPY ARRAY**

```
Import numpy as np
a = np.array ([[1, 2, 3], [3,4,5], [4,5,6]])
print 'Our array is:'
Print a
print '\n'

print 'The items in the second column are:'
print a [...,1]
print '\n'

print 'The items in the second row are:'
print a [1...]
print '\n'

print 'The items columns 1 onwards are:'
print a [...,1:]
```

OUTPUT

```
Our array is:
[[1 2 3]
[3 4 5]
[4 5 6]]

The items in the second column are:
[2 4 5]

The items in the second row are:
[3 4 5]

The items column 1 onwards are:
[[2 3]
[4 5]
[5 6]]
```

- **SCIPY**

modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

SciPy builds on the NumPy array object and is part of the NumPy stack which includes tools like Matplotlib, pandas and SymPy, and an expanding set of scientific computing libraries. This NumPy stack has similar users to other applications such as MATLAB, GNU Octave, and Scilab. The NumPy stack is also sometimes referred to as the SciPy stack.

- **The SciPy Library/Package**

The SciPy package of key algorithms and functions core to Python's scientific computing capabilities. Available sub-packages include:

- **constants:** physical constants and conversion factors (since version 0.7.0)
- **cluster:** hierarchical clustering, vector quantization, K-means
- **fftpack:** Discrete Fourier Transform algorithms
- **integrate:** numerical integration routines
- **interpolate:** interpolation tools
- **io:** data input and output
- **lib:** Python wrappers to external libraries
- **linalg:** linear algebra routines
- **misc:** miscellaneous utilities (e.g., image reading/writing)
- **ndimage:** various functions for multi-dimensional image processing
- **optimize:** optimization algorithms including linear programming
- **signal:** signal processing tools
- **sparse:** sparse matrix and related algorithms
- **spatial:** KD-trees, nearest neighbours, distance functions
- **special:** special functions
- **stats:** statistical functions
- **weave:** tool for writing C/C++ code as Python multiline strings

- **Data Structures**

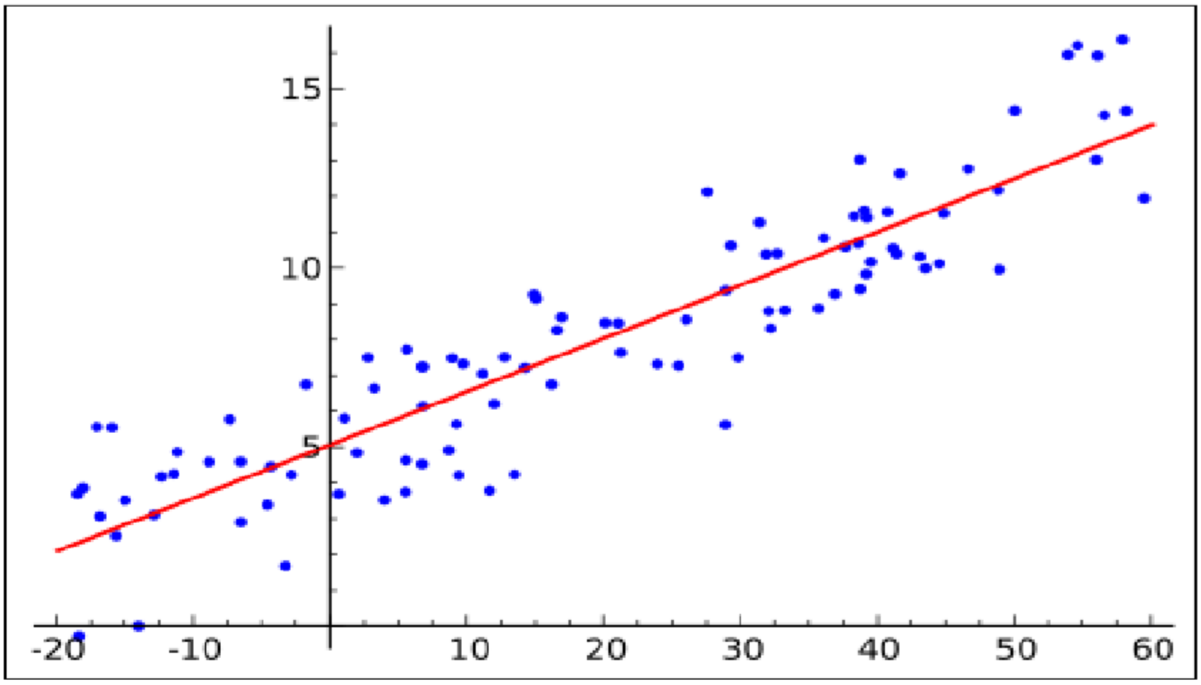
The basic data structure used by SciPy is a multidimensional array provided by the NumPy module. NumPy provides some functions for linear algebra, Fourier transforms and random number generation, but not with the generality of the equivalent functions in SciPy. NumPy can also be used as an efficient multi-dimensional container of data with arbitrary data-types. This allows NumPy to integrate with a wide variety of databases seamlessly and speedily. Older versions of SciPy used Numeric as an array type, which is now deprecated in favour of the newer NumPy array code.

- **SCIKIT-LEARN**

Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

The scikit-learn project started as scikits.learn, a Google Summer of Code project by David Cournapeau. Its name stems from the notion that it is a "SciKit" (SciPy Toolkit), a separately-developed and distributed third-party extension to SciPy.[4] The original codebase was later rewritten by other developers. In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel, all from INRIA took leadership of the project and made the first public release on February the 1st 2010[5]. Of the various scikits, scikit-learn as well as scikit-image were described as "well-maintained and popular" in November 2012.

- **REGRESSION ANALYSIS**



In statistical modelling, **regression analysis** is a set of statistical processes for estimating the relationships among variables. It includes many techniques for modelling and analysing several variables when the focus is on the relationship between a dependent variable and one or more independent variables (or 'predictors'). More specifically, regression analysis helps one understand how the typical value of the dependent variable (or 'criterion variable') changes when any one of the independent variables is varied, while the other independent variables are held fixed.

Regression analysis is widely used for prediction and forecasting, where its use has substantial overlap with the field of machine learning. Regression analysis is also used to understand which among the independent variables are related to the dependent variable, and to explore the forms of these relationships. In restricted circumstances, regression analysis can be used to infer casual relationships between the independent and dependent variables. However, this can lead to illusions or false relationships, so caution is advisable.

- **LINEAR REGRESSION**

Linear regression is a linear approach for modelling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X. The case of one explanatory variable is called *simple linear regression*. For more than one explanatory variable, the process is called *multiple linear regression*.

In linear regression, the relationships are modelled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called *linear models*.

- **LOGISTIC REGRESSION**

Logistic regression, or logit regression, or logit model [1] is a regression model where the dependent variable (DV) is categorical. This article covers the case of a binary dependent variable—that is, where the output can take only two values, "0" and "1", which represent outcomes such as pass/fail, win/lose, alive/dead or healthy/sick. Cases where the dependent variable has more than two outcome categories may be analysed in multinomial logistic regression, or, if the multiple categories are ordered, in ordinal logistic regression. In the terminology of economics, logistic regression is an example of a qualitative response/discrete choice model.

- **POLYNOMIAL REGRESSION**

Polynomial regression is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modelled as a nth degree polynomial in x.

Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y, denoted E (y | x), and has been used to describe nonlinear phenomena such as the growth rate of tissues, the distribution of carbon isotopes in lake sediments, and the progression of disease epidemics.

Although *polynomial regression* fits a nonlinear model to the data, as a statistical estimation problem it is linear, in the sense that the regression function E (y | x) is linear in the unknown parameters that are estimated from the data.

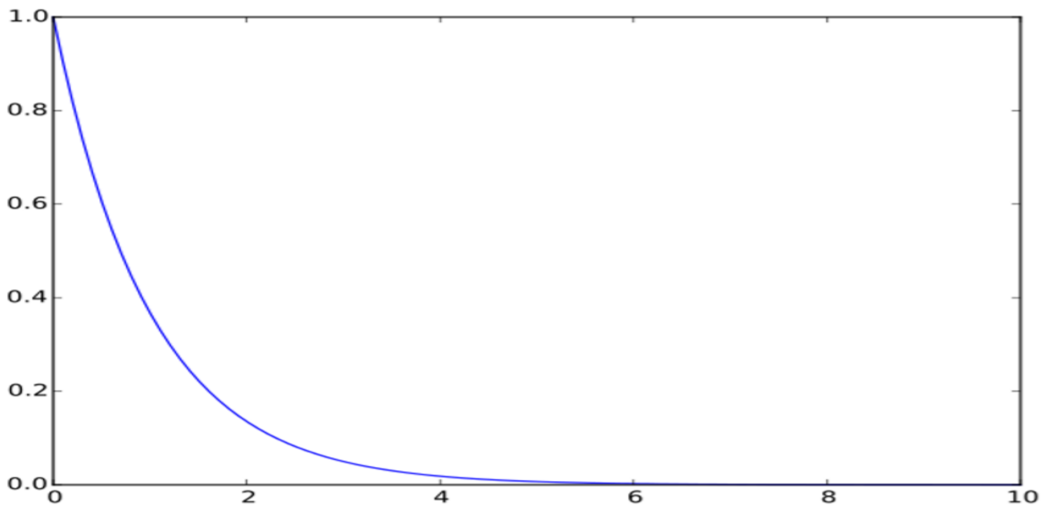
- **MATPLOTLIB**

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of matplotlib.

EXAMPLE

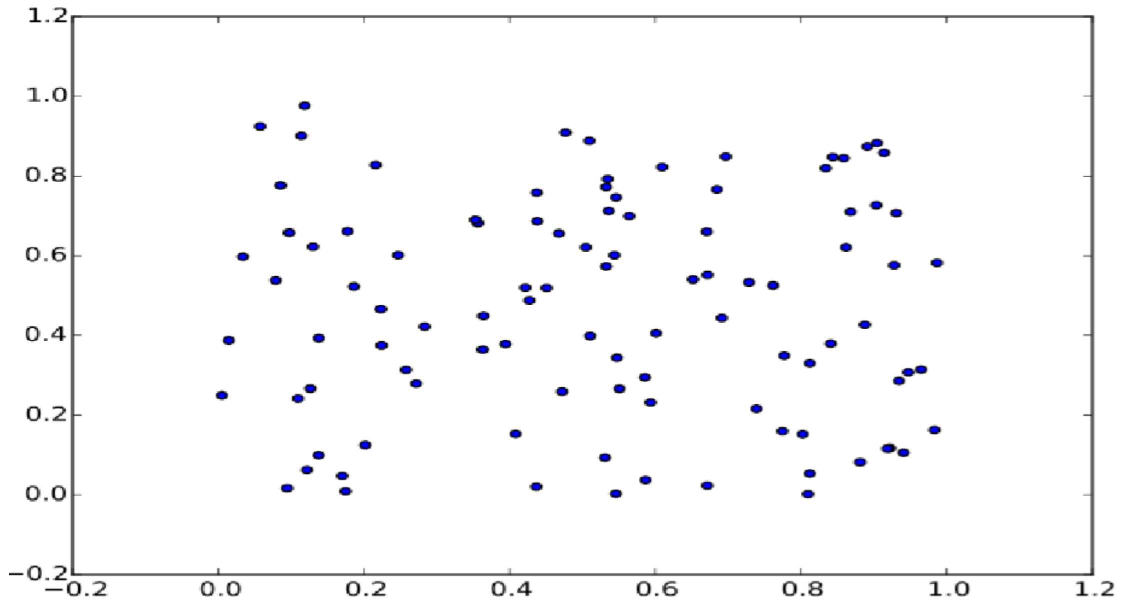
- **LINE PLOT**

```
>>>import matplotlib.pyplotasplt
>>>import numpyasnp
>>> a = np.linspace(0,10,100)
>>> b = np.exp(-a)
>>>plt.plot (a,b)
>>>plt.show ()
```



○ **<u>SCATTERPLOT</u>**

```
>>>import matplotlib.pyplotasplt
>>>from numpy.random import rand
>>> a =rand (100)
>>> b =rand (100)
>>>plt.scatter(a, b)
>>>plt.show()
```



- **PANDAS**

In computer programming, **pandas** are a software library written for the Python programming language for data manipulation and analysis. It offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. "Panel data", an econometrics term for multidimensional, structured data sets.

LIBRARY FEATURES

- Data Frame object for data manipulation with integrated indexing.
- Tools for reading and writing data between in-memory data structures and different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of data sets.
- Label-based slicing, fancy indexing, and sub setting of large data sets.
- Data structure column insertion and deletion.
- Group by engine allowing split-apply-combine operations on data sets.
- Data set merging and joining.
- Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.
- Time series-functionality: Date range generation.

- **CLUSTERING**

Cluster analysis or **clustering** is the task of grouping a set of objects in such a way that objects in the same group (called a **cluster**) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics.

Cluster analysis itself is not one specific algorithm, but the general task to be solved. It can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. Popular notions of clusters include groups with small distances among the cluster members, dense areas of the data space, intervals, or statistical distributions. Clustering can therefore be formulated as a multi-objective optimization problem.

The appropriate clustering algorithm and parameter settings (including values such as the distance function to use, a density threshold or the number of expected clusters) depend on the individual data set and intended use of the results. Cluster analysis as such is not an automatic task, but an iterative process of knowledge discovery or interactive multi-objective optimization that involves trial and failure. It is often necessary to modify data pre-processing and model parameters until the result achieves the desired properties.

NLP: NATURAL LANGUAGE PROCESSING

- **What is NLP (Natural Language Processing)?**

  Natural language processing (NLP) refers to the branch of computer science—and more specifically, the branch of artificial intelligence or AI—concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.

  NLP combines computational linguistics—rule-based modelling of human language—with statistical, machine learning, and deep learning models. Together, these technologies enable computers to process human language in the form of text or voice data and to 'understand' its full meaning, complete with the speaker or writer's intent and sentiment.

  NLP drives computer programs that translate text from one language to another, respond to spoken commands, and summarize large volumes of text rapidly—even in real time. There is a good chance you have interacted with NLP in the form of voice-operated GPS systems, digital assistants, speech-to-text dictation software, customer service chatbots, and other consumer conveniences. But NLP also plays a growing role in enterprise solutions that help streamline business operations, increase employee productivity, and simplify mission-critical business processes.

- **NLP TASK:**

  Human language is filled with ambiguities that make it incredibly difficult to write software that accurately determines the intended meaning of text or voice data. Homonyms, homophones, sarcasm, idioms, metaphors, grammar and usage exceptions, variations in sentence structure—these just a few of the irregularities of human language that take humans years to learn, but that programmers must teach natural language-driven applications to recognize and understand accurately from the start, if those applications are going to be useful.

  Several NLP tasks break down human text and voice data in ways that help the computer make sense of what it is ingesting. Some of these tasks include the following:

  - **Speech recognition:** Also called speech-to-text, is the task of reliably converting voice data into text data. Speech recognition is required for any application that follows voice commands or answers spoken questions. What makes speech recognition especially challenging is the way people talk—quickly, slurring words together, with varying emphasis and intonation, in different accents, and often using incorrect grammar.
  - **Sentiment analysis:** Attempts to extract subjective qualities—attitudes, emotions, sarcasm, confusion, suspicion—from text.
  - **Natural language generation:** Is sometimes described as the opposite of speech recognition or speech-to-text; it is the task of putting structured information into human language.
  - **Part of speech tagging:** Also called grammatical tagging, is the process of determining the part of speech of a particular word or piece of text based on its use and context. Part of speech identifies 'make' as a verb in 'I can make a paper plane,' and as a noun in 'What make of car do you own?'

- **NLP TOOLS AND APPROCH:**
  - **Python and the Natural Language Toolkit (NLTK)**

    The Python programming language provides a wide range of tools and libraries for attacking specific NLP tasks. Many of these are found in the Natural Language Toolkit, or NLTK, an open-source collection of libraries, programs, and education resources for building NLP programs.

    The NLTK includes libraries for many of the NLP tasks listed above, plus libraries for subtasks, such as sentence parsing, word segmentation, stemming and lemmatization (methods of trimming words down to their roots), and tokenization (for breaking phrases, sentences, paragraphs, and passages into tokens that help the computer better understand the text). It also includes libraries for implementing capabilities such as semantic reasoning, the ability to reach logical conclusions based on facts extracted from text.

  - **Statistical NLP, machine learning, and deep learning**

    The earliest NLP applications were hand-coded, rules-based systems that could perform certain NLP tasks, but could not easily scale to accommodate an endless stream of exceptions or the increasing volumes of text and voice data.

    Enter statistical NLP, which combines computer algorithms with machine learning and deep learning models to automatically extract, classify, and label elements of text and voice data and then assign a statistical likelihood to each meaning of those elements. Today, deep learning models and learning techniques based on convolutional neural networks (CNNs) and recurrent neural networks (RNNs) enable NLP systems that 'learn' as they work and extract ever more accurate meaning from huge volumes of raw, unstructured, and unlabelled text and voice data sets.

- **NLP USE CASE:**

Natural language processing is the driving force behind machine intelligence in many modern real-world applications. Here are a few examples:

  - **Spam detection:**

    You may not think of spam detection as an NLP solution, but the best spam detection technologies use NLP's text classification capabilities to scan emails for language that often indicates spam or phishing. These indicators can include overuse of financial terms, characteristic bad grammar, threatening language, inappropriate urgency, misspelled company names, and more. Spam detection is one of a handful of NLP problems that experts consider 'mostly solved' (although you may argue that this does not match your email experience).

  - **Machine translation:**

    Google Translate is an example of widely available NLP technology at work. Useful machine translation involves more than replacing words in one language with words of another. Effective translation must capture accurately the meaning and tone of the input language and translate it to text with the same meaning and desired impact in the output language. Machine translation tools are making substantial progress in terms of accuracy. A terrific way to test any machine translation tool is to translate text to one language and then back to the original. An oft-cited classic example: Not long ago, translating "The spirit is willing but the flesh is weak" from English to Russian and back yielded "The vodka is good but the meat is rotten." Today, the result is "The spirit desires, but the flesh is weak," which is not perfect, but inspires much more confidence in the English-to-Russian translation.

  - **Virtual agents and chatbots:**

    Virtual agents such as Apple's Siri and Amazon's Alexa use speech recognition to recognize patterns in voice commands and natural language generation to respond with appropriate action or helpful comments. Chatbots perform the same magic in response to typed text entries. The best of these also learn to recognize contextual clues about human requests and use them to provide even better responses or options over time.

TWITTER SENTIMENT ANALYSIS

- ## Introduction:

Sentiment analysis, also refers as opinion mining, is a sub machine learning task where we want to determine which is the general sentiment of a given document. Using machine learning techniques and natural language processing we can extract the subjective information of a document and try to classify it according to its polarity such as positive, neutral or negative. It is a useful analysis since we could determine the overall opinion about a selling object, or predict stock markets for a given company like, if most people think positive about it, possibly its stock markets will increase, and so on. Sentiment analysis is far from to be solved since the language is very complex (objectivity/subjectivity, negation, vocabulary, grammar...) but it is also why it is very interesting to working on. In this project I choose to try to classify tweets from Twitter into "positive" or "negative" sentiment by building a model based on probabilities. Twitter is a microblogging website where people can share their feelings quickly and spontaneously by sending a tweet limited by 140 characters. You can directly address a tweet to someone by adding the target sign "@" or participate to a topic by adding a hashtag "#" to your tweet. Because of the usage of Twitter, it is a perfect source of data to determine the current overall opinion about anything.

- **<u>Data Collection:</u>**

As our task of Sentiment Analysis is one that focuses heavily on textual data, one would expect there to be a lot of text processing. This is correct. In fact, both our Test and Training data will merely comprise of text.

I chose to start with the Test set to get you all warmed up for the Training set extraction part, as it will rely more on the API. Here is a bit of an overview of what we are about to do:

1- Register Twitter application to get our own credentials.

2- Authenticate our Python script with the API using the credentials.

3- Create function to download tweets based on a search keyword.

Registering an application with Twitter is critical, as it is the only way to get authentication credentials. As soon as we get our credentials, we will start writing code. Step 3 is where the Test set lies. We will be downloading tweets based on the term that we are trying to analyse the sentiment on.

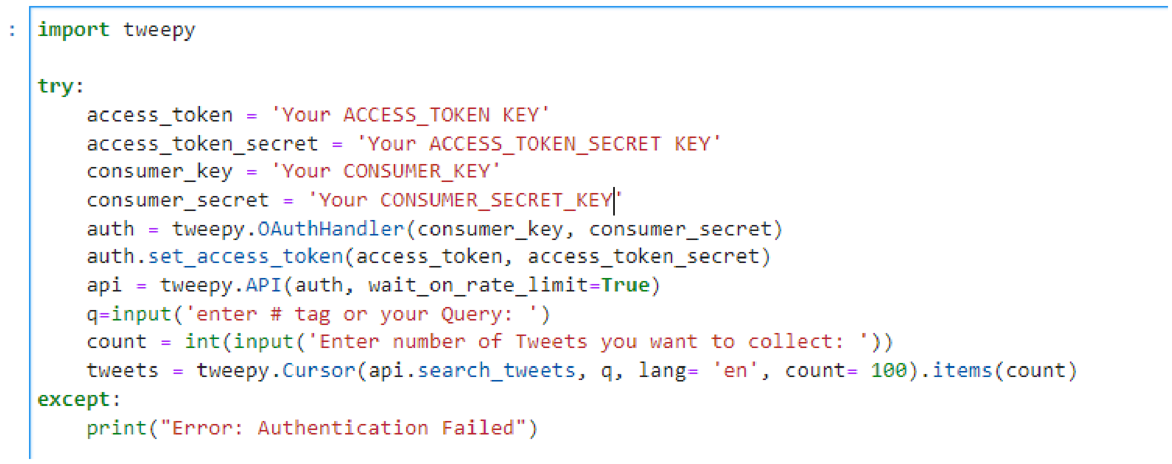
- **<u>Step 1</u>:** First off, we need to visit the Twitter Developer website on the top-right corner, click on the Apps button, Create an App, Apply and then Continue

- **<u>Step 2</u>:** Next, we will choose the "I am requesting access for my own personal use" option on the same web page, scroll-down a bit and input your Account name and Country of operation then click Continue, and you will be redirected to the next web page. Here, you can choose any Use Cases you're interested in. For our case, I chose "Student Project/Learning to Code"

- **<u>Step 3</u>:** After you make your choice, scroll-down and fill out the use case interest paragraph required. Twitter takes this very seriously now (guess they learned from Facebook's mistakes), so make sure you emphasize on the application being a self-learning/academic-related project. Choose "No" for the government involvement question, and press "Continue". On the next web page, read the Terms and Conditions list, agree to them then "Submit Application".

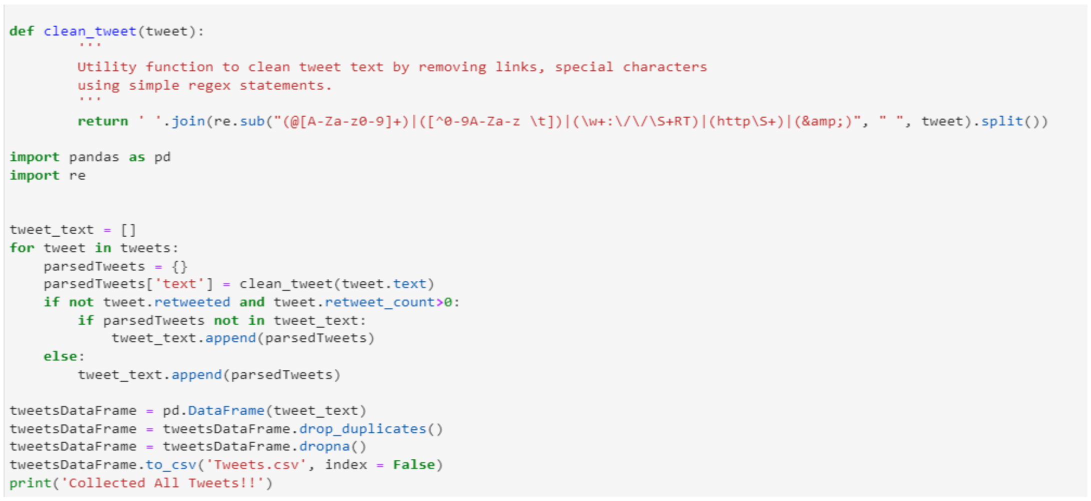
- **Step 4:** On the next web page, click "Create an app" from the top-right corner. After you are redirected, fill out the required app details, including — if you'd like — that it is for self-learning purposes. Click "Create". The next web page will include the app details that you just input, access tokens and permissions. Proceed to the "Keys and tokens" tab. Copy the API key as well as the API secret key into a safe place (a text file, if you would like), as we will be using them in a bit.

- **Step 5:** Authenticating our Python script which looks something like below:



```
import tweepy

try:
    access_token = 'Your ACCESS_TOKEN KEY'
    access_token_secret = 'Your ACCESS_TOKEN_SECRET KEY'
    consumer_key = 'Your CONSUMER_KEY'
    consumer_secret = 'Your CONSUMER_SECRET_KEY'
    auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_token, access_token_secret)
    api = tweepy.API(auth, wait_on_rate_limit=True)
    q=input('enter # tag or your Query: ')
    count = int(input('Enter number of Tweets you want to collect: '))
    tweets = tweepy.Cursor(api.search_tweets, q, lang= 'en', count= 100).items(count)
except:
    print("Error: Authentication Failed")
```

- **Step 6:** Collect Tweet based on the string provided by the user using Tweepy search_tweet method. After Collecting Tweets, we clean the and remove all the special character and hyperlinks and finally store all tweets in .csv format:



```
def clean_tweet(tweet):
        '''
        Utility function to clean tweet text by removing links, special characters
        using simple regex statements.
        '''
        return ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+RT)|(http\S+)|(&amp;)", " ", tweet).split())

import pandas as pd
import re


tweet_text = []
for tweet in tweets:
    parsedTweets = {}
    parsedTweets['text'] = clean_tweet(tweet.text)
    if not tweet.retweeted and tweet.retweet_count>0:
        if parsedTweets not in tweet_text:
            tweet_text.append(parsedTweets)
    else:
        tweet_text.append(parsedTweets)

tweetsDataFrame = pd.DataFrame(tweet_text)
tweetsDataFrame = tweetsDataFrame.drop_duplicates()
tweetsDataFrame = tweetsDataFrame.dropna()
tweetsDataFrame.to_csv('Tweets.csv', index = False)
print('Collected All Tweets!!')
```

- **Approach For Twitter Sentiment Analysis:**

The main goal of sentiment analysis is to automatically determine whether a text leaves a positive, negative, or neutral impression. It is often used to analyse customer feedback on brands, products, and services found in online reviews or on social media platforms.

Millions of tweets are posted every second. It helps us know how the public is responding to a particular event. To get the sentiments of tweets, we are going to discuss lexical-based sentiment analysis (Natural Language Processing) and can use the Naive Bayes classification algorithm (Machine Learning Approach), which is simply the application of Bayes rule.

A. Lexical Based Approach:

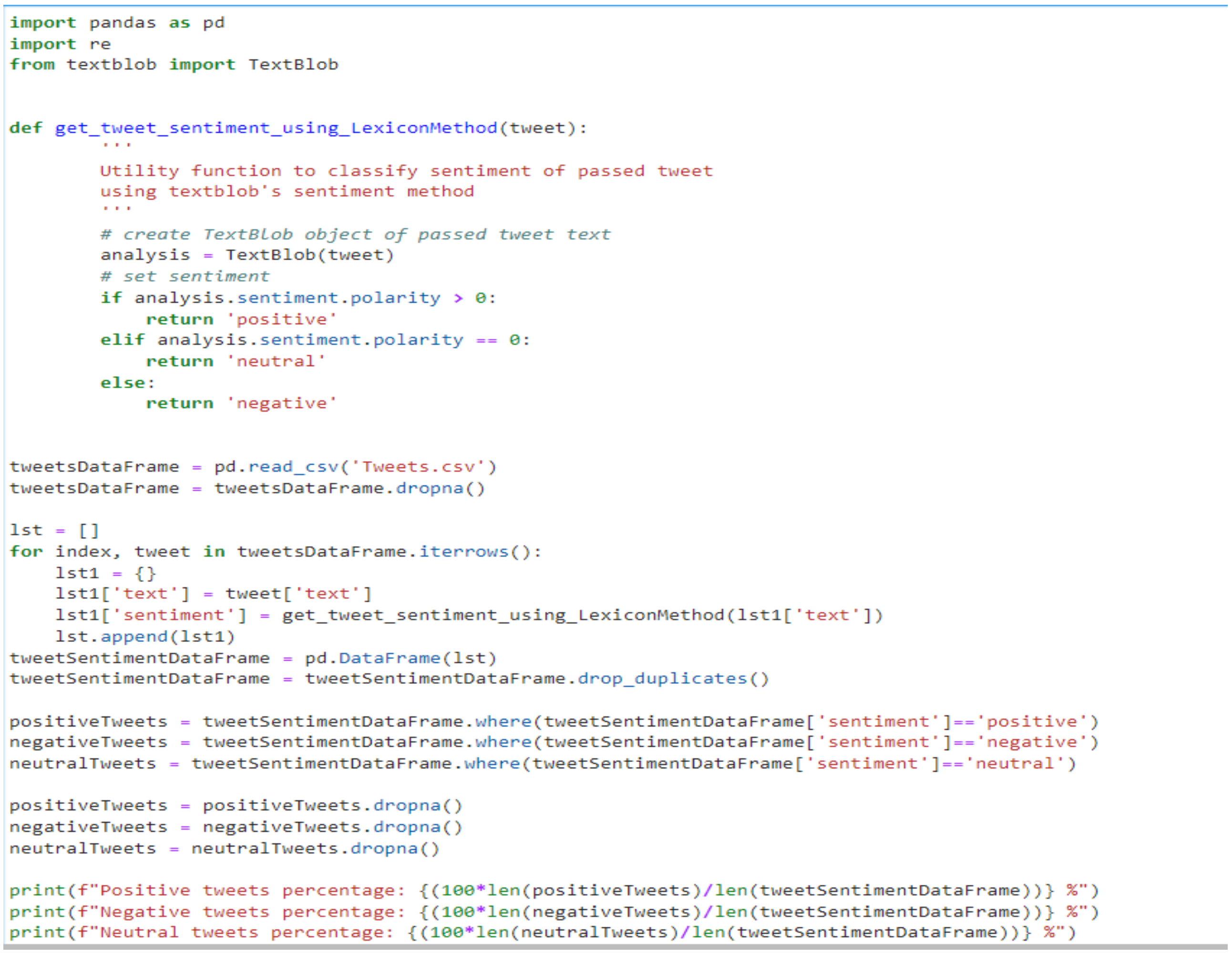
Before purchasing a product, people often search for reviews online to help them decide if they want to buy it. These reviews usually contain expressions that carry so-called emotional valence, such as “great” (positive valence) or “terrible” (negative valence), leaving readers with a positive or negative impression.

In lexicon-based sentiment analysis, words in texts are labelled as positive or negative (and sometimes as neutral) with the help of a so-called valence dictionary. Take the phrase “Good people sometimes have bad days.”. A valence dictionary would label the word “Good” as positive; the word “bad” as negative; and the other words as neutral.

Once each word in the text is labelled, we can derive an overall sentiment score by counting the numbers of positive and negative words, and combining these values mathematically. A popular formula to calculate the sentiment score (StSc) is:

$$Sentiment\ Score\ (Stsc)\ =\ \frac{(number\ of\ positive\ words - number\ of\ negative\ words)}{(total\ umber\ of\ words)}$$

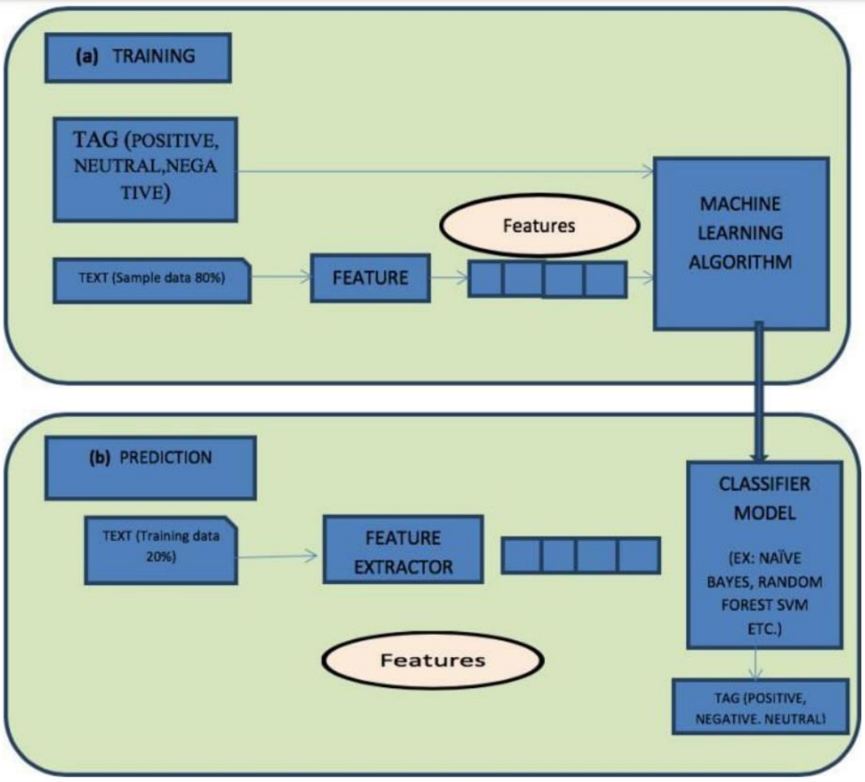
If the sentiment score is negative, the text is classified as negative. It follows that a positive score means a positive text, and a score of zero means the text is classified as neutral. To perform Lexical Based Twitter Sentiment Analysis, we used a python Library called “TextBlob” which is a Python (2 and 3) library for processing textual data. It provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more.



```
import pandas as pd
import re
from textblob import TextBlob


def get_tweet_sentiment_using_LexiconMethod(tweet):
        '''
        Utility function to classify sentiment of passed tweet
        using textblob's sentiment method
        '''
        # create TextBlob object of passed tweet text
        analysis = TextBlob(tweet)
        # set sentiment
        if analysis.sentiment.polarity > 0:
            return 'positive'
        elif analysis.sentiment.polarity == 0:
            return 'neutral'
        else:
            return 'negative'


tweetsDataFrame = pd.read_csv('Tweets.csv')
tweetsDataFrame = tweetsDataFrame.dropna()

lst = []
for index, tweet in tweetsDataFrame.iterrows():
    lst1 = {}
    lst1['text'] = tweet['text']
    lst1['sentiment'] = get_tweet_sentiment_using_LexiconMethod(lst1['text'])
    lst.append(lst1)
tweetSentimentDataFrame = pd.DataFrame(lst)
tweetSentimentDataFrame = tweetSentimentDataFrame.drop_duplicates()

positiveTweets = tweetSentimentDataFrame.where(tweetSentimentDataFrame['sentiment']=='positive')
negativeTweets = tweetSentimentDataFrame.where(tweetSentimentDataFrame['sentiment']=='negative')
neutralTweets = tweetSentimentDataFrame.where(tweetSentimentDataFrame['sentiment']=='neutral')

positiveTweets = positiveTweets.dropna()
negativeTweets = negativeTweets.dropna()
neutralTweets = neutralTweets.dropna()

print(f"Positive tweets percentage: {(100*len(positiveTweets)/len(tweetSentimentDataFrame))} %")
print(f"Negative tweets percentage: {(100*len(negativeTweets)/len(tweetSentimentDataFrame))} %")
print(f"Neutral tweets percentage: {(100*len(neutralTweets)/len(tweetSentimentDataFrame))} %")
```

B. Machine Learning Based Approach:

Machine learning ways for sentiment analysis usually rely upon supervised classification methods, wherever tagged/labelled data is employed for the approach. In below Fig we have tried to provide an overview of the entire architecture. Below, it depicts two methods (a) Training method, and (b) Prediction method. Within the Training method (a) model trains itself to adapt to a specific input (text) to the corresponding output (tag) which is based on sample data provided for training purpose, based on 80:20 principle. Here 80 % data is fed into the application with intention to train it. Rest 20 % is meant for the next phase that is prediction phase. Feature extractor function is to transfer text input from previous step into a feature vector, where the text-tag matric is built and then these feature vectors and tags (e.g., positive, negative, or neutral) are fed into the machine learning codes/algorithms that will generate a model.


(a) TRAINING
TAG (POSITIVE, NEUTRAL,NEGATIVE)
TEXT (Sample data 80%)
FEATURE
Features
MACHINE LEARNING ALGORITHM
(b) PREDICTION
TEXT (Training data 20%)
FEATURE EXTRACTOR
Features
CLASSIFIER MODEL
(EX: NAÏVE BAYES, RANDOM FOREST SVM ETC.)
TAG (POSITIVE, NEGATIVE, NEUTRAL)

To perform Machine Learning Based Twitter Sentiment analysis we have used Naive-Bayes Classification method to classify the sentiment of people based on the string or hash tags provided by the used.

Bayes rule is merely describing the probability of an event on prior knowledge of the occurrence of another event related to it. Then the probability of occurrence of event A given that event B has already occurred is

$$P(A / B) = \frac{P(A) \times P(B / A)}{P(B)}$$

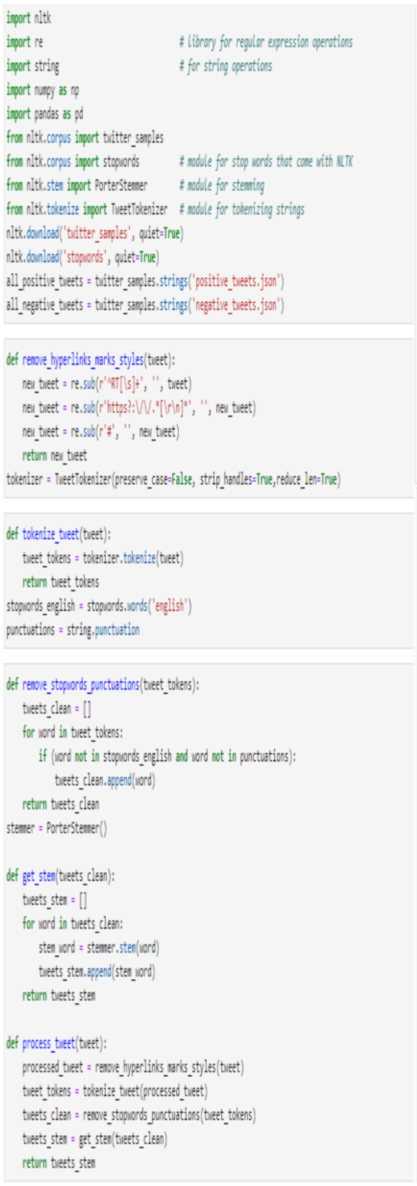
$$where,$$

$$A, B = Event$$

$$P(A|B) = probability\ of\ A\ given\ B\ is\ true$$

$$P(B / A) = probability\ of\ B\ given\ A\ is\ true$$

$$P(A), P(B) = the\ independent\ probabilities\ of\ A\ and\ B$$

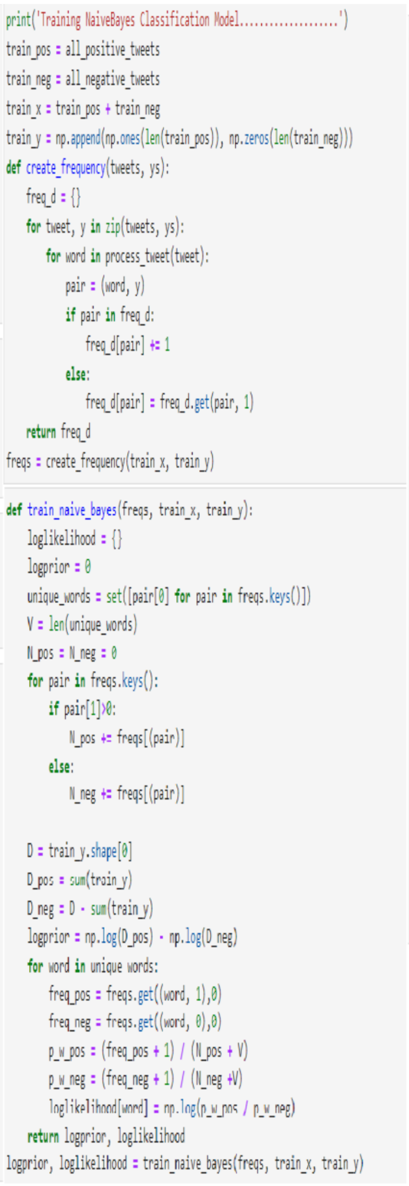


```python
import nltk
import re                                    # library for regular expression operations
import string                                # for string operations
import numpy as np
import pandas as pd
from nltk.corpus import twitter_samples
from nltk.corpus import stopwords            # module for stop words that come with NLTK
from nltk.stem import PorterStemmer          # module for stemming
from nltk.tokenize import TweetTokenizer     # module for tokenizing strings
nltk.download('twitter_samples', quiet=True)
nltk.download('stopwords', quiet=True)
all_positive_tweets = twitter_samples.strings('positive_tweets.json')
all_negative_tweets = twitter_samples.strings('negative_tweets.json')
```

```python
def remove_hyperlinks_marks_styles(tweet):
    new_tweet = re.sub(r'^RT[\s]+', '', tweet)
    new_tweet = re.sub(r'https?:\/\/.*[\r\n]*', '', new_tweet)
    new_tweet = re.sub(r'#', '', new_tweet)
    return new_tweet
tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True,reduce_len=True)
```

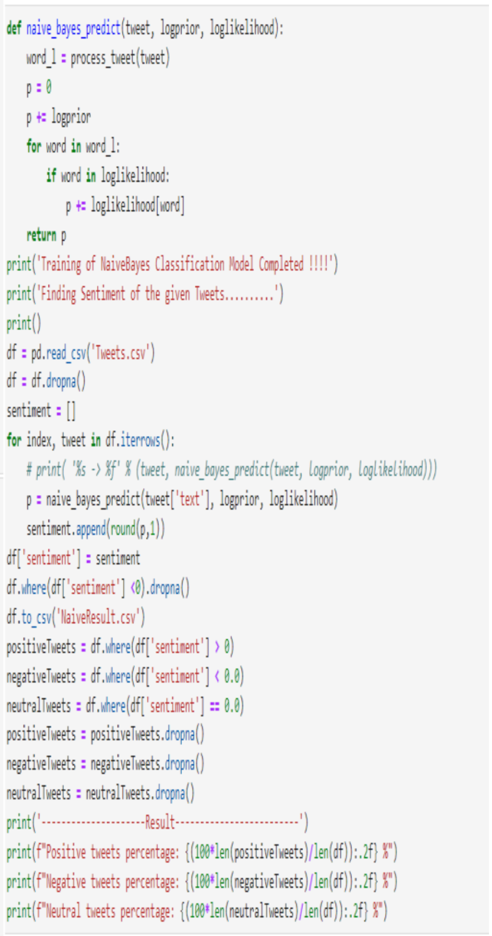
```python
def tokenize_tweet(tweet):
    tweet_tokens = tokenizer.tokenize(tweet)
    return tweet_tokens
stopwords_english = stopwords.words('english')
punctuations = string.punctuation
```

```python
def remove_stopwords_punctuations(tweet_tokens):
    tweets_clean = []
    for word in tweet_tokens:
        if (word not in stopwords_english and word not in punctuations):
            tweets_clean.append(word)
    return tweets_clean
stemmer = PorterStemmer()

def get_stem(tweets_clean):
    tweets_stem = []
    for word in tweets_clean:
        stem_word = stemmer.stem(word)
        tweets_stem.append(stem_word)
    return tweets_stem

def process_tweet(tweet):
    processed_tweet = remove_hyperlinks_marks_styles(tweet)
    tweet_tokens = tokenize_tweet(processed_tweet)
    tweets_clean = remove_stopwords_punctuations(tweet_tokens)
    tweets_stem = get_stem(tweets_clean)
    return tweets_stem
```



```python
print('Training NaiveBayes Classification Model....................')
train_pos = all_positive_tweets
train_neg = all_negative_tweets
train_x = train_pos + train_neg
train_y = np.append(np.ones(len(train_pos)), np.zeros(len(train_neg)))
def create_frequency(tweets, ys):
    freq_d = {}
    for tweet, y in zip(tweets, ys):
        for word in process_tweet(tweet):
            pair = (word, y)
            if pair in freq_d:
                freq_d[pair] += 1
            else:
                freq_d[pair] = freq_d.get(pair, 1)
    return freq_d
freqs = create_frequency(train_x, train_y)
```

```python
def train_naive_bayes(freqs, train_x, train_y):
    loglikelihood = {}
    logprior = 0
    unique_words = set([pair[0] for pair in freqs.keys()])
    V = len(unique_words)
    N_pos = N_neg = 0
    for pair in freqs.keys():
        if pair[1]>0:
            N_pos += freqs[(pair)]
        else:
            N_neg += freqs[(pair)]

    D = train_y.shape[0]
    D_pos = sum(train_y)
    D_neg = D - sum(train_y)
    logprior = np.log(D_pos) - np.log(D_neg)
    for word in unique words:
        freq_pos = freqs.get((word, 1),0)
        freq_neg = freqs.get((word, 0),0)
        p_w_pos = (freq_pos + 1) / (N_pos + V)
        p_w_neg = (freq_neg + 1) / (N_neg +V)
        loglikelihood[word] = np.log(p_w_pos / p_w_neg)
    return logprior, loglikelihood
logprior, loglikelihood = train_naive_bayes(freqs, train_x, train_y)
```



```python
def naive_bayes_predict(tweet, logprior, loglikelihood):
    word_l = process_tweet(tweet)
    p = 0
    p += logprior
    for word in word_l:
        if word in loglikelihood:
            p += loglikelihood[word]
    return p
print('Training of NaiveBayes Classification Model Completed !!!!')
print('Finding Sentiment of the given Tweets..........')
print()
df = pd.read_csv('Tweets.csv')
df = df.dropna()
sentiment = []
for index, tweet in df.iterrows():
    # print( '%s -> %f' % (tweet, naive_bayes_predict(tweet, logprior, loglikelihood)))
    p = naive_bayes_predict(tweet['text'], logprior, loglikelihood)
    sentiment.append(round(p,1))
df['sentiment'] = sentiment
df.where(df['sentiment'] <0).dropna()
df.to_csv('NaiveResult.csv')
positiveTweets = df.where(df['sentiment'] > 0)
negativeTweets = df.where(df['sentiment'] < 0.0)
neutralTweets = df.where(df['sentiment'] == 0.0)
positiveTweets = positiveTweets.dropna()
negativeTweets = negativeTweets.dropna()
neutralTweets = neutralTweets.dropna()
print('--------------------Result-----------------------')
print(f"Positive tweets percentage: {(100*len(positiveTweets)/len(df)):.2f} %")
print(f"Negative tweets percentage: {(100*len(negativeTweets)/len(df)):.2f} %")
print(f"Neutral tweets percentage: {(100*len(neutralTweets)/len(df)):.2f} %")
```

- **Future Scope:**

The data collected from Twitter was limited (10,000 - 15,000 Tweets for specific string or hash tags provided by the user). The project could be extended for a greater number of days. This project does not analyse the in real time which can be fixed if we use other technology like Apache Spark, Hadoop, etc. The error can be minimised using another algorithm.

CONCLUSION

We have collected Tweets using Twitter’s TweePy API, which is an open-source Python package that gives you a very convenient way to access the Twitter API with Python. Tweepy includes a set of classes and methods that represent Twitter's models and API endpoints, and it transparently handles various implementation details, such as: Data encoding and decoding.

After Collecting Tweets, we then remove all the unwanted characters like special character (? :, >, <, :, ;, “”,’’, etc...) and hyperlinks.

Now we select the one Lexical based approach i.e., Python TextBlob package and one Machine Learning based approach i.e., Naive Bayes Classification. It is found that Naive Bayes Approach is much more accurate than the Lexical based approach.

<u>THANK YOU</u>